aws

開発者ガイド

Amazon Comprehend

Amazon Comprehend: 開発者ガイド

Table of Contents

Amazon Comprehend とは 1
Amazon Comprehend のインサイト 2
Amazon Comprehend Custom 2
フライホイール 3
ドキュメントのクラスタリング (トピックモデリング) 3
例 3
利点 4
Amazon Comprehend の料金 4
Amazon Comprehend は初めてですか ? 5
仕組み 6
インサイト 6
エンティティ 7
イベント 9
キーフレーズ 16
主要言語 17
感情 24
ターゲット感情 25
構文分析 42
Amazon Comprehend Custom 46
トピックのモデリング 47
ドキュメント処理モード 51
単一ドキュメント処理 51
複数ドキュメントの同期処理 51
非同期バッチ処理 54
サポートされている言語 56
サポートされている言語 56
Amazon Comprehend の機能でサポートされている言語 57
設定 59
にサインアップする AWS アカウント 59
管理アクセスを持つユーザーを作成する 60
のセットアップ AWS CLI 61
プログラマチックアクセス権を付与する 61
開始 64
コンソールを使用する場合 66

リアルタイム分析 66
エンティティ 67
キーフレーズ 68
[言語] 69
個人を特定できる情報 (PII) 70
感情 72
ターゲット感情 73
構文 75
分析ジョブ (コンソール) 76
API を使用する場合 80
AWS SDKsの使用 80
リアルタイム分析 (API) 81
主要言語の検出 82
名前付きエンティティを検出する 83
キーフレーズの検出 84
感情の判断 85
ターゲット感情のリアルタイム分析 87
構文の検出 89
リアルタイムバッチ API 91
非同期分析ジョブ (API) 97
Amazon Comprehend のインサイト 97
ターゲット感情 103
イベント検出 105
トピックのモデリング 109
信頼と安全性 113
有害性検出 114
API を使用した有害コンテンツの検出 115
迅速な安全性分類 117
API を使用した迅速な安全性分類 117
PII の検出と削除 119
個人を特定できる情報 (PII) 121
PII エンティティの検出 121
PII エンティティを検索します。 122
PII エンティティの編集 123
PII ユニバーサルエンティティタイプ 123
国固有の PII エンティティタイプ 126

PII エンティティのラベル付け 127
リアルタイム分析 (コンソール) 128
オフセット 70
ラベル 72
非同期分析ジョブ (コンソール) 131
リアルタイム分析 (API) 133
PII リアルタイムエンティティ (API) の検索 133
PII リアルタイムエンティティのラベル付け (API) 134
非同期分析ジョブ (API) 135
PII エンティティの検索 136
PII エンティティの編集 141
ドキュメント処理 145
リアルタイム分析用の入力 145
プレーンテキストドキュメント 146
半構造化ドキュメント 146
イメージファイルとスキャンした PDF ファイル 146
Amazon Textract 出力 146
リアルタイム分析用の最大ドキュメントサイズ 146
半構造化ドキュメントのエラー 147
非同期分析の入力 148
プレーンテキストドキュメント 148
半構造化ドキュメント 149
イメージファイルとスキャンした PDF ファイル 150
Amazon Textract 出力 JSON サイズ 150
テキスト抽出オプションの設定 150
画像のベストプラクティス 152
カスタム分類 153
トレ–ニングデータの準備 153
調教ファイルの形式 154
マルチクラスモード 156
マルチラベルモード 158
調教分類モデル 162
カスタム分類子の調教 (コンソール) 163
カスタム分類子の調教 (API) 168
調教データのテスト 170
分類子の調教出力 171

メトリクス 175
リアルタイム分析の実行 180
リアルタイム分析 (コンソール) 181
リアルタイム分析 (API) 183
リアルタイム分析の出力 186
非同期分析ジョブの実行 188
入力ファイル形式 188
分析ジョブ (コンソール) 190
分析ジョブ (API) 192
分析ジョブの出力 193
カスタムエンティティ認識 198
トレ–ニングデータの準備 199
注釈の使用とエンティティリストの使用 200
エンティティリスト 201
注釈 204
トレーニングレコグナイザーのモデル 217
カスタムレコグナイザーをトレーニングする (コンソール) 218
レコグナイザー をトレーニングする (API) 225
メトリクス 227
リアルタイム分析の実行 231
リアルタイム分析 (コンソール) 232
リアルタイム分析 (API) 234
リアルタイム分析の出力 236
非同期分析ジョブの実行 243
分析ジョブ (コンソール) 244
分析ジョブ (API) 246
分析ジョブの出力 249
カスタムモデルの管理 255
Amazon Comprehend によるモデルバージョニング 255
AWS アカウント の間でのカスタムモデルのコピー 258
カスタムモデルの共有 259
カスタムモデルのインポート 268
フライホイール 276
フライホイールの概要 276
フライホイールのデータセット 277
フライホイールの作成 278

フライホイールの状態 278
フライホイールのイテレーション 279
フライホイールのデータレイク 279
データレイクのフォルダ構造 280
データレイクの管理 281
IAM ポリシーとアクセス許可 282
IAM ユーザーのアクセス許可を設定する 282
AWS KMS キーに対するアクセス許可を設定する 283
データアクセスロールを作成する 283
フライホイールの設定 (コンソール) 283
フライホイールを作成する 284
フライホイールを更新する 286
フライホイールを削除する 287
フライホイールの設定 (API) 287
既存のモデル用のフライホイールを作成する 287
新しいモデル用のフライホイールを作成する 288
フライホイールの説明を表示する 288
フライホイールを更新する 289
フライホイールを削除する 290
フライホイールを一覧表示する 290
データセットの設定 291
データセットの作成 (コンソール) 291
データセットの作成 (API) 292
データセットを記述します 292
フライホイールイテレーション 293
イテレーションワークフロー 293
イテレーションの管理 (コンソール) 294
イテレーションの管理 (API) 295
フライホイールの使用 298
リアルタイム分析 298
非同期ジョブ 298
エンドポイントの管理 300
エンドポイントの概要 300
エンドポイントの使用法 301
エンドポイントのモニタリング 302
エンドポイントの更新 304

の使用 Trusted Advisor 306
Amazon Comprehend の使用率の低いエンドポイント 307
Amazon Comprehend エンドポイントアクセスリスク 309
エンドポイントの削除 311
自動スケーリングとエンドポイント 312
ターゲット追跡 313
スケジュールされたスケーリング 316
タグ付け 321
新しいリソースへのタグ付け 322
タグの表示、編集、削除 323
コードの例 325
アクション 326
CreateDocumentClassifier 327
DeleteDocumentClassifier 332
DescribeDocumentClassificationJob 334
DescribeDocumentClassifier 337
DescribeTopicsDetectionJob 340
DetectDominantLanguage 342
DetectEntities 347
DetectKeyPhrases 354
DetectPiiEntities 362
DetectSentiment 367
DetectSyntax 372
ListDocumentClassificationJobs 379
ListDocumentClassifiers 382
ListTopicsDetectionJobs 385
StartDocumentClassificationJob 388
StartTopicsDetectionJob 392
シナリオ 398
ドキュメントの要素を検出する 398
サンプルデータに対してトピックモデリングジョブを実行する。 404
カスタム分類子をトレーニングしてドキュメントを分類します。 409
クロスサービスの例 421
Amazon Transcribe ストリーミングアプリケーションを構築する 422
Amazon Lex chatbot を構築する 422
Slack メッセージングアプリケーションを作成します。 424

顧客からのフィードバックを分析するアプリケーションの作成 425
画像から抽出されたテキスト内のエンティティを検出する 431
セキュリティ 433
データ保護 434
Amazon Comprehend における暗号化 435
サービス間の混乱した代理の防止 438
仮想プライベートクラウド (VPC) を使用する場合 440
VPC エンドポイントAWS PrivateLink 447
Identity and Access Management 449
対象者 449
アイデンティティを使用した認証 450
ポリシーを使用したアクセスの管理 454
Amazon Comprehend と IAM 連携の仕組み 456
アイデンティティベースポリシーの例 464
AWS マネージドポリシー 476
トラブルシューティング 480
AWS CloudTrail での Amazon Comprehend API コールのログ記録 482
の Amazon Comprehend 情報 CloudTrail 483
例: Amazon Comprehend ログファイルのエントリ 486
コンプライアンス検証 487
耐障害性 488
インフラストラクチャセキュリティ 488
ガイドラインとクォータ 490
サポートされるリージョン 490
組み込みモデルのクォータ 491
リアルタイム (同期) 分析 491
非同期分析 493
カスタムモデルのクォータ 496
一般的なクォータ 496
エンドポイントのクォータ 496
ドキュメント分類 497
カスタムエンティティ認識 501
フライホイールのクォータ 506
フライホイールの一般的なクォータ 506
カスタム分類モデルのデータセットクォータ 506
カスタムエンティティレコグナイザーモデルのデータセットクォータ 506

チュートリアル 508
レビューからインサイトを分析します 508
前提条件 510
ステップ 1: Amazon S3 にドキュメントを追加する 512
ステップ 2: (CLI のみ) IAM ロールを作成する 516
ステップ 3: 分析ジョブの実行 520
ステップ 4: 出力の準備 524
ステップ 5: 出力の可視化 536
S3 Object Lambda アクセスポイントの PII への使用 542
PII によるドキュメントへのアクセスの制御 543
ドキュメントからの PII を編集する 545
を使用したテキストの分析 OpenSearch 547
API リファレンス 548
ドキュメント履歴 549
AWS 用語集 566
..... dlxvii

Amazon Comprehend とは

Amazon Comprehend では自然言語処理 (NLP)を使用して、ドキュメントの内容に関するインサイトを抽出します。ドキュメント内のエンティティ、キーフレーズ、言語、感情、その他の共通要素を認識することでインサイトを作り上げます。Amazon Comprehend を利用することで、ドキュメントの構造を理解して新しい製品を作成できます。たとえば、Amazon Comprehend してソーシャルネットワーキングフィードから製品に関するメンションを検索したり、ドキュメントリポジトリ全体をスキャンしてキーフレーズを探し出したりできます。

Amazon Comprehend のドキュメント分析機能には、Amazon Comprehend コンソールまたは Amazon Comprehend API を使用してアクセスできます。小規模なワークロードに対してリアルタイムの分析を実行したり、大規模なドキュメントセットに対して非同期の分析ジョブを開始したりできます。Amazon Comprehend が提供するトレーニング済みモデルを使用することも、独自のカスタムモデルをトレーニングして分類やエンティティ認識を行うこともできます。

Amazon Comprehend は、コンテンツを保存して、トレーニング済みモデルの品質を継続的に向上させることができます。詳細については、「Amazon Comprehend FAQ」を参照してください。

Amazon Comprehend のすべての機能は、その入力として UTF-8 テキストドキュメントを受け付けることができます。また、カスタム分類とカスタムエンティティ認識は、その入力として画像ファイル、PDF ファイル、Word ファイルを受け付けることができます。

Amazon Comprehend は、その機能に応じて、さまざまな言語のドキュメントを検査および分析することができます。詳細については、「Amazon Comprehend でサポートされている言語」を参照してください。Amazon Comprehendの 主要言語 機能はドキュメントを検査し、主要言語を判断して、より幅広い言語が選択できるようにします。

トピック

- Amazon Comprehend のインサイト
- Amazon Comprehend Custom
- フライホイール
- ドキュメントのクラスタリング (トピックモデリング)
- 例
- 利点
- Amazon Comprehend の料金
- Amazon Comprehend は初めてですか ?

Amazon Comprehend のインサイト

Amazon Comprehend では、トレーニング済みモデルを使用して 1 つまたは複数のドキュメントを検査して分析し、インサイトを収集することができます。このモデルは、大量の医療テキストで継続的にトレーニングされるため、トレーニングデータを提供する必要はありません。

Amazon Comprehend は、以下の種類のインサイトを分析します。

- Entities — ドキュメント内での人物、場所、アイテム、場所への言及 (名前)。
- Key phrases — ドキュメントに現れるフレーズ。例えば、バスケットボールの試合に関するドキュメントでは、チームの名前、会場の名前、最終スコアを返すことができます。
- 個人を特定できる情報 (PII) — 住所や銀行口座番号、電話番号など、個人を特定できる個人データ。
- Language — ドキュメントの主要言語。
- Sentiment — ドキュメントの主要な感情。肯定的、中立、否定的、混在のいずれからにできます。
- Targeted sentiment — ドキュメント内の特定のエンティティに関連する感情。出現する各エンティティに対する感情は、肯定的、否定的、中立、混在のいずれかにできます。
- Syntax — ドキュメント内の各単語の品詞。

詳細については、「インサイト」を参照してください。

Amazon Comprehend Custom

Amazon Comprehend は、機械学習ベースの NLP ソリューションを構築するために必要なスキルを必要とすることなく、お客様それぞれのニーズに合わせてカスタマイズすることができます。Amazon Comprehend Custom は、自動機械学習 (AutoML) を使用して、すでにあるデータを使用して自動的にカスタマイズされた NLP モデルを構築できます。

カスタム分類 — カスタム分類モデル (分類子) を作成して、ドキュメントを専用のカテゴリに整理できます。

カスタムエンティティ認識 — カスタムエンティティ認識モデル (レコグナイザー) を作成し、特定の用語および名詞ベースの語句についてテキストを分析できます。

詳細については、「Amazon Comprehend Custom」を参照してください。

フライホイール

フライホイールを利用すると、時間の経過に伴うカスタムモデルのバージョンのトレーニングと管理のプロセスを簡素化できます。フライホイールは、モデルの新しいバージョンのトレーニングと評価に関連するタスクをオーケストレーションするのに役立ちます。フライホイールは、カスタム分類とカスタムエンティティ認識を行うためのプレーンテキストのカスタムモデルをサポートします。詳細については、「フライホイール」を参照してください。

ドキュメントのクラスタリング (トピックモデリング)

Amazon Comprehend を利用すると、大量のドキュメントを検査し、類似キーワードに基づいてドキュメントを整理することもできます。ドキュメントのクラスタリング (トピックモデリング) は、大量のドキュメントを単語の出現頻度に基づいて類似したトピックまたはクラスターに整理するのに便利です。詳細については、「トピックのモデリング」を参照してください。

例

以下は、アプリケーションにおける Amazon Comprehend オペレーションの使用例です。

Example 1: 1 つの話題に関するドキュメントの検出

Amazon Comprehend トピックモデリングを利用すると、特定の話題に関するドキュメントを検出することができます。一連のドキュメントをスキャンして、扱われているトピックを決定し、各トピックに関連するドキュメントを検出することができます。Amazon Comprehend がドキュメントセットから返すトピックの数を指定できます。

Example 2: 商品に関するお客様の感情を特定する

商品カタログを刊行する場合は、お客様がその商品をどのように考えているか、 Amazon Comprehend に調べさせてください。すべてのお客様のコメントを `DetectSentiment` オペレーションに送信すると、お客様が商品に対して肯定的、否定的、中立、混在のどの感情を持っているか知らせてくれます。

Example 3: お客様にとって大切なことを発見する

Amazon Comprehend のトピックモデリングを利用すると、お客様がフォーラムや掲示板で話題にしているトピックを発見し、エンティティ検出を使用して、そのトピックに関連する人物や場所、物

を突き止めることができます。　感情分析を利用することで、お客様がトピックに対して抱いている感情を知ることができます。

利点

Amazon Comprehend を使用する利点には、以下のようなものがあります。

- アプリケーションへの自然言語処理の簡単かつ強力な組み込み — Amazon Comprehend では、シンプルな API を使用して強力で正確な自然言語処理を利用できるようにすることで、アプリケーションにテキスト分析機能を組み込む際の煩雑さを排除します。Amazon Comprehend が生成するインサイトを活用するためのテキスト分析の専門知識は必要ありません。
- 深層学習ベースの自然言語処理 — Amazon Comprehend は、深層学習テクノロジーを使用してテキストを正確に分析します。精度を向上させるために、複数のドメインにわたって新しいデータを使用して常にモデルがトレーニングされます。
- スケーラブルな自然言語処理 — Amazon Comprehend では、何百万ものドキュメントを分析して、そこに含まれるインサイトを検出することができます。
- 他の AWS サービスとの統合 – Amazon Comprehend は、Amazon S3、 AWS KMS、 などの他の AWS サービスとシームレスに連携するように設計されています AWS Lambda。ドキュメントを Amazon S3 に保存するか、Firehose でリアルタイムデータを分析します。 AWS Identity and Access Management (IAM) のサポートにより、Amazon Comprehend オペレーションへのアクセスを安全に制御することが容易になります。IAM を利用して、ユーザーおよびグループを作成・管理し、開発者やエンドユーザーに適切なアクセス権を付与できます。
- 出力結果とボリュームデータの暗号化 — Amazon S3 はすでに入力ドキュメントを暗号化できるようになっており、Amazon Comprehend ではこの暗号化機能をさらに拡張しています。独自の KMS キーを使用することで、ジョブの出力結果と、分析ジョブを処理するコンピュートインスタンスに追加されたストレージボリューム上のデータを暗号化できます。その結果、セキュリティが大幅に強化されます。
- 低コスト — Amazon Comprehend では、最低料金や前払いの義務はありません。課金は、分析したドキュメントとトレーニングしたカスタムモデルに対して発生します。

Amazon Comprehend の料金

Amazon Comprehend では、使用したリソースに対して課金が発生するだけです。 AWS の新規のお客様の場合、当初は Amazon Comprehend を無料で利用できます。詳細については、「AWS の無料利用枠」を参照してください。

リアルタイムまたは非同期の分析ジョブの実行には課金が発生します。カスタムモデルのトレーニングやカスタムモデル管理では課金が発生します。カスタムモデルを使用するリアルタイムリクエストでは、エンドポイントを開始してからエンドポイントを削除するまで、エンドポイントに対する課金が発生します。フライホイールの利用に追加料金はかかりません。ただし、フライホイールのイテレーションを実行すると、新しいモデルバージョンのトレーニングとモデルデータの保存に標準料金がかかります。

料金とその他の詳細情報については、「Amazon Comprehend の料金」を参照してください。

Amazon Comprehend は初めてですか ?

Amazon Comprehend を初めてご利用いただく場合は、以下のセクションを順に読むことをお勧めします。

1. 仕組み — このセクションでは、Amazon Comprehend の概念を説明しています。
2. セットアップ - このセクションでは、アカウントを作成して、 AWS CLIをセットアップします。
3. Amazon Comprehend の開始方法 — このセクションでは、Amazon Comprehend 分析ジョブを実行します。
4. チュートリアル:Amazon Comprehend を使用してカスタマーレビューからインサイトを分析する — このセクションでは、感情とエンティティの分析を行い、結果を可視化します。
5. Amazon Comprehend API リファレンス — Amazon Comprehend オペレーションのリファレンスドキュメントです。

AWS は、Amazon Comprehend サービスについて学習するための以下のリソースを提供します。

- AWS 機械学習ブログ - Amazon Comprehend 関係の有用な記事が掲載されています。
- Amazon Comprehend リソース - Amazon Comprehend 関係の有用なビデオとチュートリアルを提供しています。

仕組み

Amazon Comprehend は、事前にトレーニングされたモデルを使用して、1 つのドキュメントまたは一連のドキュメントに関する洞察を収集します。このモデルは、大量の医療テキストで継続的にトレーニングされるため、トレーニングデータを提供する必要はありません。

Amazon Comprehend では、カスタム分類とカスタムエンティティ認識のための独自のカスタムモデルを構築することができます。フライホイール を使用してカスタムモデルの管理することができます。

Amazon Comprehend では、組み込みモデルを使用してトピックモデリングを行うことができます。トピックモデリングでは、大量のドキュメントが調べられ、その中の類似キーワードに基づいてドキュメントが整理されます。

Amazon Comprehend には、同期および非同期のドキュメント処理モードがあります。同期モードは、1 つのドキュメント、または最大 25 個のドキュメントをバッチ処理する場合に使用します。非同期ジョブは多数の文書を処理する場合に使用します。

Amazon Comprehend と AWS Key Management Service (AWS KMS) と連携することで、データの暗号化を強化することができます。詳細については、「Amazon Comprehend における暗号化」を参照してください。

主要なコンセプト

- インサイト
- Amazon Comprehend Custom
- トピックのモデリング
- ドキュメント処理モード

インサイト

Amazon Comprehend は、事前にトレーニングされたモデルを使用して、1 つのドキュメントまたは一連のドキュメントに関する洞察を収集します。Amazon Comprehend がドキュメントに関して示す洞察には、次のようなものがあります。

- エンティティ — Amazon Comprehend は、ドキュメントで特定されたエンティティ (人、位置所、場所など) のリストを返します。

- イベント — Amazon Comprehend は、特定の種類のイベントと関連する詳細を検出します。
- キーフレーズ — Amazon Comprehend は、ドキュメントに含まれるキーフレーズを抽出します。例えば、バスケットボールの試合に関するドキュメントでは、チームの名前、会場の名前、最終スコアを返すことができます。
- 個人を特定できる情報 (PII) — Amazon Comprehend は文書を分析して、住所や銀行口座番号、電話番号など、個人を特定する個人情報を検出します。
- 主要言語 — Amazon Comprehend はドキュメント内の主要言語を特定します。Amazon Comprehend は 100 の言語を識別できます。
- センチメント — Amazon Comprehend はドキュメントの主要センチメントを判断します。センチメントは、ポジティブのこともあれば、中立やネガティブ、あるいはそれらが混在することもあります。
- ターゲットセンチメント — Amazon Comprehend は、ドキュメントに記載されている特定のエンティティのセンチメントを特定します。センチメントは、ポジティブのこともあれば、中立やネガティブ、あるいはそれらが混在することもあります。
- 構文分析 — Amazon Comprehend はドキュメント内の各単語を解析し、その単語の品詞を特定します。たとえば、「It is raining today in Seattle」という文では、「it」は代名詞、「raining」は動詞、「Seattle」は固有名詞と認識されます。

エンティティ

エンティティとは、人や場所、あるいは商品などの現実世界のオブジェクトの固有の名前をテキストで表したものであり、日付や数量などの測定値を正確に表すことができます。

たとえば、「John moved to 1313 Mockingbird Lane in 2012」というテキストでは、「John」は `PERSON` と認識され、「1313 Mockingbird Lane」は `LOCATION`、「2012」 は`DATE` と認識されます。

各エンティティには、エンティティタイプが正しく検出されたという Amazon Comprehend の信頼レベルを示すスコアも含まれます。スコアの低いエンティティを除外して、誤検出を使ってしまう行うリスクを減らすことができます。

以下の表はエンティティのタイプをまとめています。

型	説明
COMMERCIAL_ITEM	ブランド製品

型	説明
DATE	日付 (11/25/2017 など)、曜日 (Tuesday)、月 (May)、または時刻 (8:30 a.m.)
EVENT	フェスティバル、コンサート、選挙などのイベント
LOCATION	国、都市、湖、建物などの特定の場所
ORGANIZATION	政府、企業、宗教、スポーツチームなどの大規模な組織
OTHER	他のどのエンティティカテゴリにも当てはまらないエンティティ
PERSON	個人、グループ、ニックネーム、架空の人物
QUANTITY	通貨、パーセント、数値、バイト数などの数量。
TITLE	映画、本、歌など、あらゆる作品や創作作品に付けられた正式名称。

エンティティの検出オペレーションは、Amazon Comprehend がサポートする主要言語のいずれかを使用して実行できます。これには、定義済み (カスタムではない) エンティティの検出のみが対象になります。ドキュメントの言語はすべて同じである必要があります。

次の API オペレーションのいずれかを使用して、ドキュメントまたはドキュメントセット内のエンティティを検出できます。

- DetectEntities
- BatchDetectEntities
- StartEntitiesDetectionJob

これらのオペレーションは、ドキュメント内のエンティティごとに 1 つの割合で、API エンティティオブジェクトのリストを返します。BatchDetectEntities オペレーションは、バッチ内のドキュメントごとに 1 つのリストの割合で Entity オブジェクトのリストを返します。StartEntitiesDetectionJob オペレーションは、ジョブ内のドキュメントごとに Entity オブジェクトのリスト 1 つを含むファイルを生成する非同期ジョブを開始します。

以下は、DetectEntities オペレーションからのレスポンスの例です。

```
{
    "Entities": [
        {
            "Text": "today",
            "Score": 0.97,
            "Type": "DATE",
            "BeginOffset": 14,
            "EndOffset": 19
        },
        {
            "Text": "Seattle",
            "Score": 0.95,
            "Type": "LOCATION",
            "BeginOffset": 23,
            "EndOffset": 30
        }
    ],
    "LanguageCode": "en"
}
```

イベント

イベント検出を使用して、特定のタイプのイベントとそれに関連するエンティティについてテキストドキュメントを分析することができます。Amazon Comprehend では、非同期分析ジョブを使用して、大量のドキュメントコレクションにまたがるイベント検出を行うことができます。イベント分析ジョブの例を含むイベントの詳細については、「Amazon Comprehend Events の開始のお知らせ」を参照してください。

エンティティ

Amazon Comprehend は、入力テキストから、検出されたイベントに関連するエンティティのリストを抽出します。エンティティは、人や位置、あるいは場所などの現実世界のオブジェクトである場合もあれば、測定値や日付、あるいは数量などの概念である場合もあります。エンティティが出現すると、それぞれメンションで識別されます。メンションは、入力テキスト内のエンティティに対するテキスト参照です。一意のエンティティごとに、すべてのメンションが 1 つのリストにまとめられます。このリストには、エンティティが出現する入力テキスト内の各場所の詳細情報が提供されます。Amazon Comprehend は、サポートされているイベントタイプに関連付けられているエンティティのみを検出します。

サポートされているイベントタイプに関連付けられている各エンティティについて、以下の関連情報が返されます。

- MenMentions: 入力テキストにおける同じエンティティの各出現に関する詳細情報。
  - BeginOffset: 言及が始まる場所を示す入力テキストの文字オフセット (最初の文字は位置 0)。
  - EndOffset: 言及が終了する場所を示す入力テキストの文字オフセット。
  - Score: Amazon Comprehend におけるエンティティのタイプの精度の信頼度。
  - GroupScore: Amazon Comprehend から、同じエンティティの他のメンションと正しくグループ化されているという信頼度。
  - Text: エンティティのテキスト。
  - Type: エンティティのタイプ。サポートされているすべてのエンティティタイプについては、「エンティティタイプ」を参照してください。

イベント

Amazon Comprehend は、入力テキストで検出されたイベント(サポートされているイベントタイプのイベント)のリストを返します。各イベントについて、以下の関連情報が返されます。

- Type: イベントのタイプ。サポートされているすべてのエンティティタイプについては、「イベントタイプ」を参照してください。
- Arguments: 検出されたイベントに関連する引数のリスト。引数は、検出されたイベントに関連するエンティティ 1 つで構成されます。引数の役割は、誰がいつ、どこで何をしたかなどの関係を表すことです。
  - EntityIndex: Amazon Comprehend がこの分析のために返したエンティティのリストからエンティティを識別するインデックス値。
  - Role: 引数のタイプ。この引数のエンティティとイベントとの関係を表します。サポートされているすべての引数の型については、引数の型を参照してください。
  - Score: Amazon Comprehend における役割検出の精度の信頼度。
- Triggers: 検出されたイベントのトリガーのリスト。トリガーとは、イベントの発生を示す 1 つの単語またはフレーズです。
  - BeginOffset: トリガーが始まる場所を示す入力テキストの文字オフセット (最初の文字は位置 0)。
  - EndOffset: トリガーの終了位置を示す入力テキストの文字オフセット。
  - Score: Amazon Comprehend における検出の精度の信頼度。
  - Text: トリガーのテキスト。
  - GroupScore: トリガーが同じイベントの他のトリガーと正しくグループ化されているという Amazon Comprehend の信頼度。

- Type: このトリガーが示すイベントのタイプ。

イベント検出結果の形式

イベント検出ジョブが完了すると、Amazon Comprehend は、ジョブの開始時に指定された Amazon S3 上の出力場所に分析結果を書き込みます。

検出された各イベントの出力には、以下の形式でその詳細が提供されます。

```
{
   "Entities": [
     {
       "Mentions": [
         {
           "BeginOffset": number,
           "EndOffset": number,
           "Score": number,
           "GroupScore": number,
           "Text": "string",
           "Type": "string"
         }, ...
       ]
     }, ...
   ],
   "Events": [
     {
       "Type": "string",
       "Arguments": [
         {
           "EntityIndex": number,
           "Role": "string",
           "Score": number
         }, ...
       ],
       "Triggers": [
         {
           "BeginOffset": number,
           "EndOffset": number,
           "Score": number,
           "Text": "string",
           "GroupScore": number,
           "Type": "string"
```

```
        }, ...
      ]
    }, ...
  ]
}
```

サポートされているエンティティ、イベント、引数のタイプ

エンティティタイプ

型	説明
DATE	日付または時刻の参照 (具体的か否かを問わず)。
FACILITY	建物、空港、高速道路、橋、その他の恒久的な人工構造物および不動産の改良。
LOCATION	道路、都市、州、国、水域、地理座標などの物理的な場所。
MONETARY_VALUE	米国またはその他の通貨でのモノの価値。価値は具体的なこともあれば、概算値のこともあります。
ORGANIZATION	企業または確立された組織体制で団体。
PERSON	個人または架空の人物の名前またはニックネーム。
PERSON_TITLE	個人を説明する任意のタイトル。通常は雇用カテゴリー (CEO など) または敬語 (Mr. など)。
QUANTITY	数値または値と測定単位。
STOCK_CODE	AMZN、国際証券識別番号 (ISIN)、統一証券識別手続き委員会 (CUSIP)、証券取引所日報公式リスト (SEDOL) などの株式ティッカーシンボル。

イベントタイプ

型	説明
BANKRUPTCY	個人または企業が未払いの債務を返済できないこに伴う法的手続き。
EMPLOYMENT	従業員の採用、解雇、退職、その他の雇用状態の変化があると発生するイベント。
CORPORATE_ACQUISTION	会社が他社の株式または有形資産のほとんどまたは全部の所有権を取得して、その会社の支配権を獲得すると発生するイベント。
INVESTMENT_GENERAL	個人または企業が将来の収入または評価を生み出す見込みのある資産を購入すると発生するイベント。
CORPORATE_MERGER	2 つ以上の会社が合併して新しい法人を設立すると発生するイベント。
IPO	新規株式発行時に民間企業の株式を一般に公開する新規株式公開（IPO）を行うと発生するベント。
RIGHTS_ISSUE	既存の株主に既存の保有株式に比例して追加の株式を購入するために提供される一群の権利 (新株予約権と呼ばれる)。
SECONDARY_OFFERING	会社の株主による有価証券の売り出し。
SHELF_OFFERING	証券取引委員会 (SEC) の規定。発行者が証券を再登録したり、罰金を科されたりすることなく、新規発行証券を登録し、一定期間にわたってその一部を売却することを許可する規定。シェルフレジストレーションともいいます。
TENDER_OFFERING	会社の株主の株式の一部または全部を購入する申し出。

型	説明
STOCK_SPLIT	企業の取締役会が、現在の株主に発行する株式数を増やして発行済株式数を増やすと発生するイベント。このイベントは株式の逆分割にも当てはまります。

引数の型

BANKRUPTCYの引数の型

引数の型	説明
FILER	破産を申請した個人または会社。
DATE	破産した日付または時刻。
PLACE	破産が発生した (またはその場所に最も近い) 場所または施設。

EMPLOYMENTの引数の型

型	説明
EMPLOYEES	会社に雇用されている人。
EMPLOYEE_TITLE	従業員の役職。
EMPLOYER	従業員を雇用している個人または会社。
START_DATE	雇用の開始日または時刻。
END_DATE	雇用の終了日または時刻。

CORPORATE_ACQUISTION、INVESTMENT_GENERAL の引数の型

型	説明
AMOUNT	取引に関係する金銭的価値。
INVESTEE	投資に関係する個人または会社。
INVESTOR	資産に投資にした個人または会社。
DATE	買収または投資の日付または時刻。
PLACE	買収または投資が行われた (またはその場所に最も近い) 場所。

CORPORATE_MERGER の引数の型

型	説明
DATE	合併の日付または時刻。
NEW_COMPANY	合併で生まれた新しい法人。
PARTICIPANT	合併に関わった会社。

IPO、RIGHTS_ISSUE、SECONDARY_OFFERING、SHELF_OFFERING、TENDER_OFFERING の引数の型

型	説明
有効期限	売り出しの有効期限または時刻。
INVESTOR	資産に投資にした個人または会社。
OFFEREE	売り出しを受けた個人または会社。
OFFERING_AMOUNT	売り出しに関係する金銭的価値。
OFFERING_DATE	売り出しの日付または時刻。

型	説明
OFFEROR	売り出しを開始した個人または会社。
OFFEROR_TOTAL_VALUE	売り出しに関係する金銭的価値の合計。
RECORD_DATE	売り出しの記録日付または時刻。
SELLING_AGENT	売り出しの販売を促進する個人または会社。
SHARE_PRICE	株式価格に関係する金銭的価値。
SHARE_QUANTITY	売り出しに関係する株式数。
UNDERWRITERS	売り出しの引受に関係する会社。

STOCK_SPLIT の引数の型

型	説明
COMPANY	株式分割の株式を発行する会社。
DATE	株式分割の日付または時刻。
SPLIT_RATIO	株式分割前の現在の株式数に対する新規発行済株式数の増加率。

キーフレーズ

キーフレーズとは、特定のことを説明する名詞句を含む文字列です。一般にはキーフレーズは、名詞 1 つとそれが名詞であることを示す修飾語で構成されます。たとえば、「day」は名詞で、「beautiful day」は冠詞 (「a」) と形容詞 (「美しい」) を含む名詞句です。各キーフレーズには、文字列が名詞句であるという Amazon Comprehend の信頼度を示すスコアが含まれます。このスコアに基づいて、検出がアプリケーションに必要な信頼度があるかどうかを判断することができます。

キーフレーズの検出オペレーションは、Amazon Comprehend がサポートするプライマリ言語が何であっても実行できます。ドキュメントの言語はすべて同じである必要があります。

次の API オペレーションのいずれかを使用して、ドキュメントまたはドキュメントセット内のキーフレーズを検出できます。

- DetectKeyPhrases
- BatchDetectKeyPhrases
- StartKeyPhrasesDetectionJob

オペレーションは、ドキュメント内のキーフレーズごとに 1 つずつ、KeyPhraseオブジェクトのリストを返します。BatchDetectKeyPhrases オペレーションは、バッチ内のドキュメントごとに 1 つの割合で KeyPhrase オブジェクトのリストを返します。StartKeyPhrasesDetectionJob オペレーションは、ジョブ内のドキュメントごとに KeyPhrase オブジェクトのリスト 1 つを含むファイルを生成する非同期ジョブを開始します。

以下は、DetectKeyPhrases オペレーションからのレスポンスの例です。

```
{
    "LanguageCode": "en",
    "KeyPhrases": [
        {
            "Text": "today",
            "Score": 0.89,
            "BeginOffset": 14,
            "EndOffset": 19
        },
        {
            "Text": "Seattle",
            "Score": 0.91,
            "BeginOffset": 23,
            "EndOffset": 30
        }
    ]
}
```

主要言語

Amazon Comprehend を使用してテキストを調べ、主要言語を判断できます。Amazon Comprehend は、RFC 5646 の識別子を使用して言語を識別します。2 文字の ISO 639-1 識別子があり、必要に応じて地域のサブタグがある場合は、それを使用します。それ以外の場合は ISO 639-2 の 3 文字コードを使用します。

RFC 5646 の詳細は、IETF ツールウェブサイトの「言語識別用タグ」を参照してください。

応答には、特定の言語がドキュメント内の主要な言語であるという Amazon Comprehend の信頼レベルを示すスコアが含まれます。各スコアは他のスコアとは無関係です。スコアは、ある言語が文書の特定の割合を占めていることを示すものではありません。

長い文書 (本など) に複数の言語が含まれている場合は、長い文書を小さく分割して、個々の部分に対して `DetectDominantLanguage` 演算を実行できます。その結果を集計して、長い文書に含まれる各言語の割合を判断できます。

Amazon Comprehend の言語検出には次の制約があります。

- 音声言語検出には対応していません。たとえば、「arigato」を日本語として、「nihao」を中国語として検出しません。
- インドネシア語とマレー語、ボスニア語、クロアチア語、セルビア語など、近い言語ペアを区別するのが難しい場合があります。
- 最良の結果を得るには、20 文字以上のテキストを入力してください。

Amazon Comprehend は次の言語を検出します。

Code	言語
af	アフリカーンス語
am	アムハラ語
ar	アラビア語
as	アッサム語
az	アゼルバイジャン語
ba	バシキール語
be	ベラルーシ語
bn	ベンガル語
bs	ボスニア語

Code	言語
bg	ブルガリア語
ca	カタロニア語
ceb	セブアノ語
cs	チェコ語
cv	チュヴァシュ語
cy	ウェールズ語
da	デンマーク語
de	ドイツ語
el	ギリシャ語
en	英語
eo	エスペラント語
et	エストニア語
eu	バスク語
fa	ペルシャ語
fi	フィンランド語
fr	フランス語
gd	スコティッシュゲール語
ga	アイルランド語
gl	ガリシア語
gu	グジャラート語

Code	言語
ht	ハイチ語
he	ヘブライ語
ha	ハウサ語
hi	ヒンディー語
hr	クロアチア語
hu	ハンガリー語
hy	アルメニア語
ilo	イロコ語
id	インドネシア語
is	アイスランド語
it	イタリア語
jv	ジャワ語
ja	日本語
kn	カンナダ語
ka	グルジア語
kk	カザフ語
km	中部クメール語
ky	キルギス語
ko	韓国語
ku	クルド語

Code	言語
lo	ラオス語
la	ラテン語
lv	ラトビア語
lt	リトアニア語
lb	ルクセンブルク語
ml	マラヤーラム語
mt	マルタ語
mr	マラーティー語
mk	マケドニア語
mg	マダガスカル語
mn	モンゴル語
ms	マレー語
my	ビルマ語
ne	ネパール語
new	ネワール語
nl	オランダ語
no	ノルウェー語
or	オリヤー語
om	オロモ語
pa	パンジャブ語

Code	言語
pl	ポーランド語
pt	ポルトガル語
ps	プシュトン語
qu	ケチュア語
ro	ルーマニア語
ru	ロシア語
sa	サンスクリット語
si	シンハラ語
sk	スロバキア語
sl	スロベニア語
sd	シンディー
so	ソマリ語
es	スペイン語
sq	アルバニア語
sr	セルビア語
su	スンダ語
sw	スワヒリ語
sv	スウェーデン語
ta	タミル語
tt	タタール語

Code	言語
te	テルグ語
tg	タジク語
tl	タガログ語
th	タイ語
tk	トルクメン語
tr	トルコ語
ug	ウイグル語
uk	ウクライナ語
ur	ウルドゥー語
uz	ウズベク語
vi	ベトナム語
yi	イディッシュ語
yo	ヨルバ語
zh	簡体字中国語
zh-TW	繁体字中国語

次の API 演算機能のいずれかを使用して、1 つまたは複数のドキュメントの主要言語を検出できます。

- DetectDominantLanguage
- BatchDetectDominantLanguage
- StartDominantLanguageDetectionJob

DetectDominantLanguage オペレーションは DominantLanguage オブジェクトを返します。BatchDetectDominantLanguage の演算では、バッチ内のドキュメントごとに 1 つずつ、DominantLanguage オブジェクトのリストを返します。StartDominantLanguageDetectionJob の演算では、非同期ジョブが開始され、ジョブ内のドキュメントごとに 1 つずつ、DominantLanguage オブジェクトのリストが入ったファイルが作成されます。

次の例は、DetectDominantLanguage の演算からの応答です。

```
{
    "Languages": [
        {
            "LanguageCode": "en",
            "Score": 0.9793661236763
        }
    ]
}
```

感情

Amazon Comprehend を使用して、UTF-8 エンコードしたテキストドキュメントの内容の感情を判定することができます。例えば、感情分析を使用してブログ投稿に対するコメントの感情を判定して、読者がその投稿を気に入ったかどうかを判断することができます。

Amazon Comprehend がサポートする主要言語のどの言語のドキュメントについても感情を判定できます。1つのジョブ内のドキュメントはすべて、同じ言語である必要があります。

テキストに対する感情判定では次の値が返されます。

- 肯定的 — 全体的に肯定的。
- 否定的 — 全体的に否定的。
- 混在 – 肯定的と否定的の両方。
- ニュートラル – 肯定的と否定的のどちらでもない。

次の API オペレーションのいずれかを使用して、1 つまたは複数のドキュメントの感情を検出できます。

- DetectSentiment

- BatchDetectSentiment
- StartSentimentDetectionJob

これらのオペレーションでは、テキストで最も優勢な感情と各感情のスコアが返されます。スコアはセンチメントが正しく検出された可能性を表します。例えば、次の例では、感情が Positive である可能性は 95% です。感情が Negative である可能性は 1% 未満です。SentimentScore を使用して、検出の正確さがアプリケーションの要件を満たしているかどうかを判断することができます.

DetectSentiment オペレーションは、検出された感情と オブジェクトを含む SentimentScore オブジェクトを返します。BatchDetectSentiment オペレーションは、バッチ内のドキュメントごとに 1 つの割合で、感情と SentimentScore オブジェクトのリストを返します。StartSentimentDetectionJob オペレーションは非同期ジョブを開始し、ジョブ内のドキュメントごとに 1 つの割合で、感情と SentimentScore オブジェクトのリストを含むファイルを生成します。

以下は、DetectSentiment オペレーションからのレスポンスの例です。

```
{
"SentimentScore": {
        "Mixed": 0.030585512690246105,
        "Positive": 0.94992071056365967,
        "Neutral": 0.0141543131828308,
        "Negative": 0.00893945890665054
    },
    "Sentiment": "POSITIVE",
    "LanguageCode": "en"
}
```

ターゲット感情

ターゲット感情を使用すると、入力ドキュメント内で特定のエンティティ (ブランドや製品など) に関連付けられた感情を詳細に把握できます。

ターゲット感情と「感情」の違いは、出力データの粒度のレベルです。感情分析は、各入力ドキュメントの主要な感情を決定しますが、詳細な分析のためのデータは提供しません。ターゲット感情分析では、各入力ドキュメント内の特定のエンティティのエンティティレベルの感情を判断します。出力データを分析して、肯定的または否定的なフィードバックを得た特定の製品やサービスを判断できます。

たとえば、あるレストランのレビューの中で、顧客が「タコスはおいしかったし、スタッフもフレンドリーだった」というレビューを投稿したとします。このレビューを分析すると、次の結果になります。

- 感情分析は、各レストランレビューの全体的な感情が肯定的、否定的、中間、または混合のいずれであるかを判断します。この例では、全体的な感情は肯定的です。
- ターゲット感情分析は、顧客がレビューで言及したレストランのエンティティと属性に対する感情を判断します。この例では、顧客は「タコス」と「スタッフ」について肯定的なコメントをしています。

ターゲット感情では、分析ジョブごとに次の出力が得られます。

- ドキュメントで言及されているエンティティの識別。
- 各エンティティメンションに対するエンティティタイプの分類。
- 言及された各エンティティの感情と感情スコア。
- 単一エンティティに対応する参照グループ (共通参照グループ)。

コンソールまたは API を使用して、ターゲット感情分析を実行できます。コンソールと API は、ターゲット感情のリアルタイム分析と非同期分析の両方をサポートします。

Amazon Comprehend は、英語ドキュメントのターゲット感情に対応しています。

チュートリアルを含むターゲット感情の詳細については、AWS 機械学習ブログの「Amazon Comprehend ターゲット感情を使ってテキスト内の詳細な感情を抽出する」を参照してください。

トピック

- エンティティタイプ
- 共参照グループ
- 出力ファイルの構成
- コンソールを使用したリアルタイム分析
- ターゲット感情の出力例

エンティティタイプ

ターゲット感情は、次のエンティティタイプを識別します。エンティティが他のどのカテゴリにも属さない場合は、エンティティタイプ「OTHER」(その他) を割り当てます。出力ファイル内の各エンティティには、`"Type": "PERSON"` などのエンティティタイプが含まれます。

エンティティタイプの定義

エンティティタイプ	定義
PERSON	例としては、個人、グループ、ニックネーム、架空の人物、動物の名前などがあります。
LOCATION	国、都市、州、住所、地形、水域、自然史跡、天文学的位置などの地理的位置。
ORGANIZATION	例としては、政府、企業、スポーツチーム、宗教などがあります。
FACILITY	建物、空港、高速道路、橋、その他の恒久的な人工構造物および不動産の改良。
BRAND	特定の商品または製品ラインの組織、グループ、または生産者。
COMMERCIAL_ITEM	購入または取得可能な非ジェネリック品目 (車両、および 1 つの品目しか生産されていない大型製品を含む) 。
MOVIE	映画またはテレビ番組。考えられるエンティティには、フルネーム、ニックネーム、またはサブタイトルがあります。
MUSIC	曲 (全体または一部)。また、アルバムやアンソロジーなど、個々の音楽作品のコレクションもあります。
BOOK	職業的に、または自費出版された本。
SOFTWARE	正式にリリースされたソフトウェア製品。
GAME	ビデオゲーム、ボードゲーム、一般的なゲーム、スポーツなどのゲーム。
PERSONAL_TITLE	学長、博士、博士などの正式な肩書きと敬称。

エンティティタイプ	定義
EVENT	例としては、フェスティバル、コンサート、選挙、戦争、会議、プロモーションイベントなどがあります。
DATE	具体的、一般的、絶対的、相対的であるかを問わず、日付や時刻に関するあらゆる言及。
QUANTITY	すべての測定値とその単位 (通貨、パーセント、数値、バイトなど)。
ATTRIBUTE	製品の「品質」、電話の「価格」、CPU の「速度」など、エンティティの属性、特性、または特質。
OTHER	他のカテゴリに属さないエンティティ。

共参照グループ

ターゲット感情は、各入力ドキュメントの共参照グループを識別します。共参照グループとは、現実世界の 1 つのエンティティに対応するドキュメント内の感情グループです。

Example

以下のカスタマーレビューの例では、「spa」はエンティティーであり、そのエンティティータイプは `FACILITY` です。このエンティティには、代名詞 (「it」) としてさらに 2 つの追加言及があります。

出力ファイルの構成

ターゲットの感情分析ジョブは JSON テキスト出力ファイルを作成します。このファイルには、入力ドキュメントごとに 1 つの JSON オブジェクトが含まれます。各 JSON オブジェクトには、以下のフィールドが含まれています。

- エンティティ — ドキュメント内にあるエンティティの配列。
- ファイル — 入力ドキュメントのファイル名。
- 行 – 入力ファイルが 1 行につき 1 つのドキュメントである場合、[エンティティ]にはファイル内のドキュメントの行番号が含まれます。

> **Note**
>
> ターゲット感情が入力テキスト中のエンティティを認識していない場合、エンティティの結果として空の配列を返します。

次の例は、3 行入力されている入力ファイルのエンティティを示しています。入力フォーマットは ONE_DOC_PER_LINE であるため、入力の各行は 1 つのドキュメントになります。

```
{ "Entities":[
    {entityA},
    {entityB},
    {entityC}
    ],
  "File": "TargetSentimentInputDocs.txt",
  "Line": 0
}
{ "Entities": [
    {entityD},
    {entityE}
  ],
  "File": "TargetSentimentInputDocs.txt",
  "Line": 1
}
{ "Entities": [
    {entityF},
    {entityG}
    ],
  "File": "TargetSentimentInputDocs.txt",
```

```
  "Line": 2
}
```

エンティティ配列のエンティティは、文書内で検出されたエンティティ参照の論理グループ (共参照グループと呼ばれる) を含みます。各エンティティの全体構造は以下のとおりです。

```
{"DescriptiveMentionIndex": [0],
  "Mentions": [
     {mentionD},
     {mentionE}
    ]
}
```

エンティティには以下のフィールドが含まれます。

- メンション — ドキュメント内のエンティティに関するメンションの配列。配列は共参照グループを表します。例については、「the section called “共参照グループ”」を参照してください。メンション配列内のメンション順序は、ドキュメント内での位置 (オフセット) の順序です。各メンションには、そのメンションの感情スコアとグループスコアが含まれます。グループスコアは、これらのメンションが同じエンティティに属していることの確実性レベルを示します。
- DescriptiveMentionIndex — エンティティグループの最適な名前を提供する「メンション」配列への 1 つ以上のインデックス。たとえば、あるエンティティにテキスト値が「ABC Hotel」、「ABC Hotel」、「it」の 3 つのメンションがあるとします。最適な名前は「ABCTAK」で、DescriptiveMentionIndex 値は [0,1] です。

各メンションには、次のフィールドが含まれます。

- BeginOffset - 言及が始まるドキュメントテキストのオフセット。
- EndOffset – 言及が終了するドキュメントテキストのオフセット。
- GroupScore - グループに記載されているすべてのエンティティが同じエンティティに関連する信頼度。
- テキスト — エンティティを識別するドキュメント内のテキストです。
- タイプ – エンティティのタイプ。Amazon Comprehend は、さまざまなエンティティタイプをサポートしています。
- スコア — エンティティの関連性についてのモデルの信頼性です。値の範囲は 0 ~ 1 で、1 が最も高い信頼度です。

- MentionSentiment - 言及の感情と感情スコアが含まれます。
- Sentiment — メンションの感情。値には、「肯定的」、「中間」、「否定的」、「混合」が含まれます。
- SentimentScore - 考えられる各感情のモデル信頼度を提供します。値の範囲は 0 ~ 1 で、1 が最も高い信頼度です。

Sentiment 値には以下の意味があります。

- 肯定的 — エンティティメンションは肯定的な感情を表します。
- 否定的 — エンティティメンションは否定的な感情を表します。
- 混合 – 肯定的感情と否定的感情を表すエンティティメンション。
- 中間 – 肯定的感情または否定的感情のいずれも表さないエンティティメンション。

次の例では、エンティティは入力ドキュメントに 1 つのメンションしか記述していないため、DescriptiveMentionIndexはゼロです (メンション配列の最初のメンション)。識別されるエンティティは「I」という名前の PERSON です。感情スコアは中間です。

```
{"Entities":[
  {
    "DescriptiveMentionIndex": [0],
    "Mentions": [
      {
       "BeginOffset": 0,
       "EndOffset": 1,
       "Score": 0.999997,
       "GroupScore": 1,
       "Text": "I",
       "Type": "PERSON",
       "MentionSentiment": {
         "Sentiment": "NEUTRAL",
         "SentimentScore": {
           "Mixed": 0,
           "Negative": 0,
           "Neutral": 1,
           "Positive": 0
         }
       }
     }
   ]
```

```
  }
 ],
 "File": "Input.txt",
 "Line": 0
}
```

コンソールを使用したリアルタイム分析

Amazon Comprehend コンソールを使用すると、the section called “ターゲット感情” をリアルタイムで実行できます。サンプルテキストを使用するか、入力テキストボックスに独自のテキストを貼り付けて、 [分析]を選択します。

[インサイト] パネルでは、コンソールにターゲット感情分析の 3 つのビューが表示されます。

- テキスト解析 — テキスト解析を表示し、各エンティティに下線を引きます。下線の色は、分析によってエンティティに割り当てられた感情値 (肯定的、中間、否定的、または混合のいずれか) を表します。コンソールでは、テキスト解析ボックスの右上隅にカラーマッピングが表示されます。エンティティの上にカーソルを置くと、そのエンティティの分析値 (エンティティタイプ、感情スコア) を含むポップアップパネルがコンソールに表示されます。
- 結果 — テキスト内で識別された各エンティティが参照する行を含むテーブルを表示します。各エンティティについて、この表にはエンティティとエンティティスコアが表示されます。この行には、主な感情と各感情値のスコアも含まれています。同じエンティティに 「the section called “共参照グループ”」 と呼ばれる複数の参照がある場合、この表には、マスターエンティティに関連付けられた折りたたみ可能な行のセットとしてこれらの参照が表示されます。

  結果 テーブルのエンティティ行にカーソルを合わせると、コンソールは [テキスト解析)]パネル内のエンティティメンションを強調表示します。
- アプリケーション統合 — API リクエストのパラメータ値と API レスポンスで返された JSON オブジェクトの構造を表示します。JSON オブジェクトのフィールドの説明については、「the section called “出力ファイルの構成”」 を参照してください。

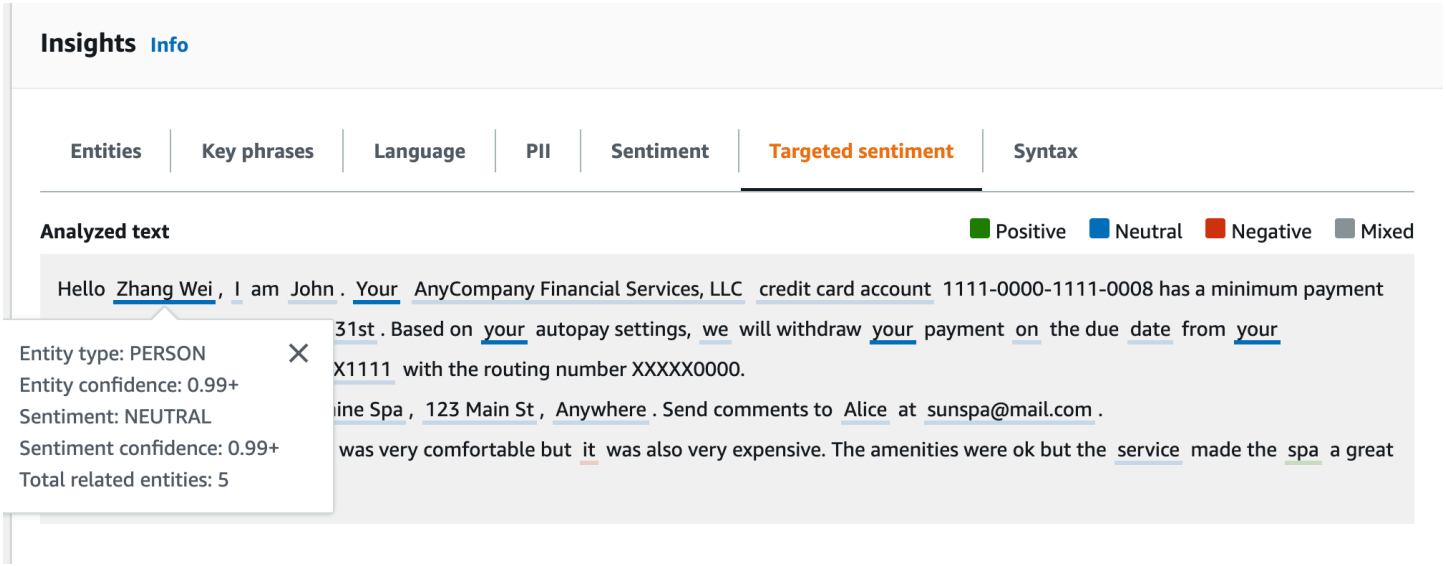
コンソールのリアルタイム分析の例

この例では、以下のテキストを入力として使用します。これはコンソールが提供するデフォルトの入力テキストです。

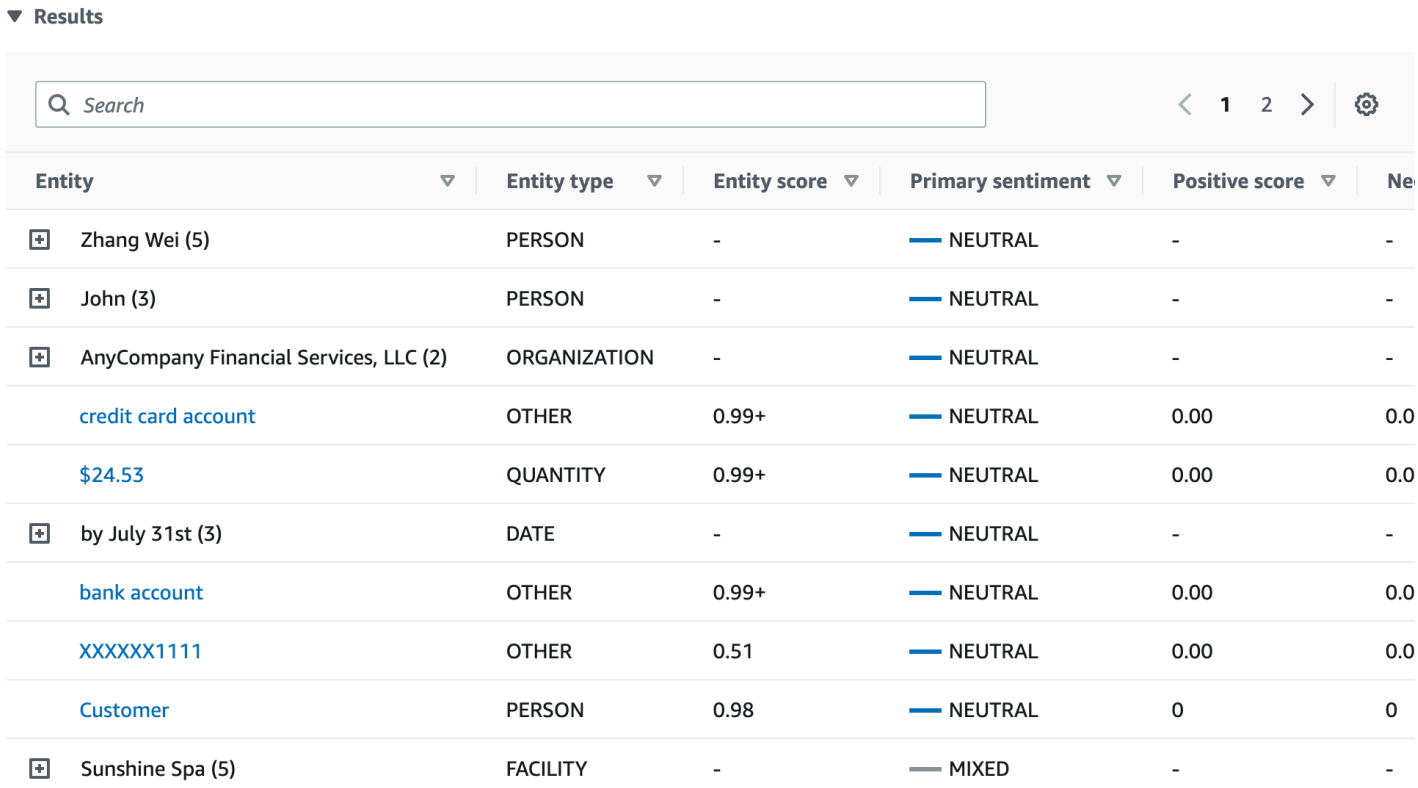
```
Hello Zhang Wei, I am John. Your AnyCompany Financial Services, LLC credit card account
 1111-0000-1111-0008 has a minimum payment
```

```
 of $24.53 that is due by July 31st. Based on your autopay settings, we will withdraw
your payment on the due date from your
 bank account number XXXXXX1111 with the routing number XXXXX0000.
 Customer feedback for Sunshine Spa, 123 Main St, Anywhere. Send comments to Alice at
sunspa@mail.com.
 I enjoyed visiting the spa. It was very comfortable but it was also very expensive.
The amenities were ok but the service made
 the spa a great experience.
```

この例では、[分析済みテキスト] パネルには次の出力が表示されます。テキスト Zhang Wei の上にマウスカーソルを合わせると、このエンティティのポップアップパネルが表示されます。



[結果] テーブルには、エンティティスコア、主要感情、各感情のスコアなど、各エンティティに関する追加の詳細が表示されます。



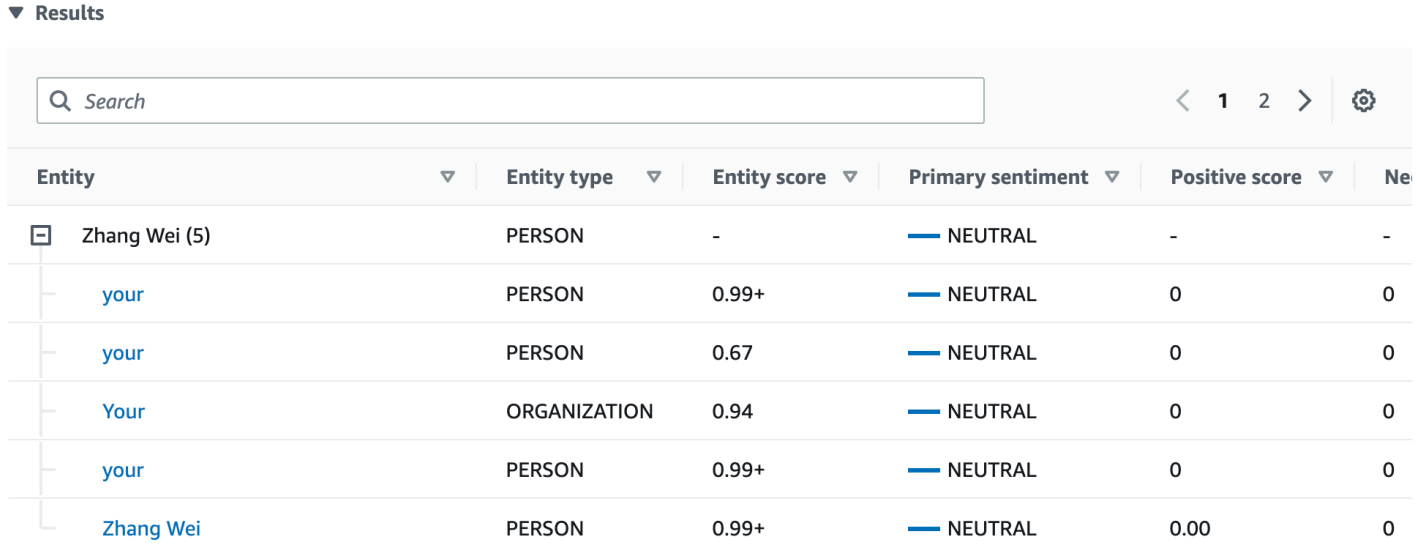
Results

Search

1 2

Entity	Entity type	Entity score	Primary sentiment	Positive score	Ne
Zhang Wei (5)	PERSON	-	NEUTRAL	-	-
John (3)	PERSON	-	NEUTRAL	-	-
AnyCompany Financial Services, LLC (2)	ORGANIZATION	-	NEUTRAL	-	-
credit card account	OTHER	0.99+	NEUTRAL	0.00	0.0
$24.53	QUANTITY	0.99+	NEUTRAL	0.00	0.0
by July 31st (3)	DATE	-	NEUTRAL	-	-
bank account	OTHER	0.99+	NEUTRAL	0.00	0.0
XXXXXX1111	OTHER	0.51	NEUTRAL	0.00	0.0
Customer	PERSON	0.98	NEUTRAL	0	0
Sunshine Spa (5)	FACILITY	-	MIXED	-	-

この例では、ターゲット感情分析では、入力テキストで あなたのについて言及するたびに、人物エンティティ Zhang Wei への参照であることが認識されます。コンソールには、これらのメンションがメインエンティティに関連付けられた折りたたみ可能な行のセットとして表示されます。



Results

Search

1 2

Entity	Entity type	Entity score	Primary sentiment	Positive score	Ne
Zhang Wei (5)	PERSON	-	NEUTRAL	-	-
your	PERSON	0.99+	NEUTRAL	0	0
your	PERSON	0.67	NEUTRAL	0	0
Your	ORGANIZATION	0.94	NEUTRAL	0	0
your	PERSON	0.99+	NEUTRAL	0	0
Zhang Wei	PERSON	0.99+	NEUTRAL	0.00	0

アプリケーション統合パネルには、 DetectTargetedSentiment API が生成する JSON オブジェクトが表示されます。詳細な例については、次のセクションを参照してください。

ターゲット感情の出力例

次の例は、ターゲット感情分析ジョブの出力ファイルを示しています。入力ファイルは次の 3 つのシンプルなドキュメントで構成されています。

```
The burger was very flavorful and the burger bun was excellent. However, customer
 service was slow.
My burger was good, and it was warm. The burger had plenty of toppings.
The burger was cooked perfectly but it was cold. The service was OK.
```

この入力ファイルのターゲット感情分析では、次の出力が生成されます。

```
 {"Entities":[
   {
     "DescriptiveMentionIndex": [
       0
     ],
     "Mentions": [
       {
         "BeginOffset": 4,
         "EndOffset": 10,
         "Score": 0.999991,
         "GroupScore": 1,
         "Text": "burger",
         "Type": "OTHER",
         "MentionSentiment": {
           "Sentiment": "POSITIVE",
           "SentimentScore": {
             "Mixed": 0,
             "Negative": 0,
             "Neutral": 0,
             "Positive": 1
           }
         }
       }
     ]
   },
   {
     "DescriptiveMentionIndex": [
       0
     ],
     "Mentions": [
       {
```

```
          "BeginOffset": 38,
          "EndOffset": 44,
          "Score": 1,
          "GroupScore": 1,
          "Text": "burger",
          "Type": "OTHER",
          "MentionSentiment": {
            "Sentiment": "NEUTRAL",
            "SentimentScore": {
              "Mixed": 0.000005,
              "Negative": 0.000005,
              "Neutral": 0.999591,
              "Positive": 0.000398
            }
          }
        }
      ]
    },
    {
      "DescriptiveMentionIndex": [
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 45,
          "EndOffset": 48,
          "Score": 0.961575,
          "GroupScore": 1,
          "Text": "bun",
          "Type": "OTHER",
          "MentionSentiment": {
            "Sentiment": "POSITIVE",
            "SentimentScore": {
              "Mixed": 0.000327,
              "Negative": 0.000286,
              "Neutral": 0.050269,
              "Positive": 0.949118
            }
          }
        }
      ]
    },
    {
      "DescriptiveMentionIndex": [
```

```
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 73,
          "EndOffset": 89,
          "Score": 0.999988,
          "GroupScore": 1,
          "Text": "customer service",
          "Type": "ATTRIBUTE",
          "MentionSentiment": {
            "Sentiment": "NEGATIVE",
            "SentimentScore": {
              "Mixed": 0.000001,
              "Negative": 0.999976,
              "Neutral": 0.000017,
              "Positive": 0.000006
            }
          }
        }
      ]
    }
  ],
  "File": "TargetSentimentInputDocs.txt",
  "Line": 0
}
{
  "Entities": [
    {
      "DescriptiveMentionIndex": [
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 0,
          "EndOffset": 2,
          "Score": 0.99995,
          "GroupScore": 1,
          "Text": "My",
          "Type": "PERSON",
          "MentionSentiment": {
            "Sentiment": "NEUTRAL",
            "SentimentScore": {
              "Mixed": 0,
```

```
            "Negative": 0,
            "Neutral": 1,
            "Positive": 0
          }
        }
      }
    ]
  },
  {
    "DescriptiveMentionIndex": [
      0,
      2
    ],
    "Mentions": [
      {
        "BeginOffset": 3,
        "EndOffset": 9,
        "Score": 0.999999,
        "GroupScore": 1,
        "Text": "burger",
        "Type": "OTHER",
        "MentionSentiment": {
          "Sentiment": "POSITIVE",
          "SentimentScore": {
            "Mixed": 0.000002,
            "Negative": 0.000001,
            "Neutral": 0.000003,
            "Positive": 0.999994
          }
        }
      },
      {
        "BeginOffset": 24,
        "EndOffset": 26,
        "Score": 0.999756,
        "GroupScore": 0.999314,
        "Text": "it",
        "Type": "OTHER",
        "MentionSentiment": {
          "Sentiment": "POSITIVE",
          "SentimentScore": {
            "Mixed": 0,
            "Negative": 0.000003,
            "Neutral": 0.000006,
```

```
            "Positive": 0.999991
          }
        }
      },
      {
        "BeginOffset": 41,
        "EndOffset": 47,
        "Score": 1,
        "GroupScore": 0.531342,
        "Text": "burger",
        "Type": "OTHER",
        "MentionSentiment": {
          "Sentiment": "POSITIVE",
          "SentimentScore": {
            "Mixed": 0.000215,
            "Negative": 0.000094,
            "Neutral": 0.00008,
            "Positive": 0.999611
          }
        }
      }
    ]
  },
  {
    "DescriptiveMentionIndex": [
      0
    ],
    "Mentions": [
      {
        "BeginOffset": 52,
        "EndOffset": 58,
        "Score": 0.965462,
        "GroupScore": 1,
        "Text": "plenty",
        "Type": "QUANTITY",
        "MentionSentiment": {
          "Sentiment": "NEUTRAL",
          "SentimentScore": {
            "Mixed": 0,
            "Negative": 0,
            "Neutral": 1,
            "Positive": 0
          }
        }
```

```
        }
      ]
    },
    {
      "DescriptiveMentionIndex": [
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 62,
          "EndOffset": 70,
          "Score": 0.998353,
          "GroupScore": 1,
          "Text": "toppings",
          "Type": "OTHER",
          "MentionSentiment": {
            "Sentiment": "NEUTRAL",
            "SentimentScore": {
              "Mixed": 0,
              "Negative": 0,
              "Neutral": 0.999964,
              "Positive": 0.000036
            }
          }
        }
      ]
    }
  ],
  "File": "TargetSentimentInputDocs.txt",
  "Line": 1
}
{
  "Entities": [
    {
      "DescriptiveMentionIndex": [
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 4,
          "EndOffset": 10,
          "Score": 1,
          "GroupScore": 1,
          "Text": "burger",
```

```
          "Type": "OTHER",
          "MentionSentiment": {
            "Sentiment": "POSITIVE",
            "SentimentScore": {
              "Mixed": 0.001515,
              "Negative": 0.000822,
              "Neutral": 0.000243,
              "Positive": 0.99742
            }
          }
        },
        {
          "BeginOffset": 36,
          "EndOffset": 38,
          "Score": 0.999843,
          "GroupScore": 0.999661,
          "Text": "it",
          "Type": "OTHER",
          "MentionSentiment": {
            "Sentiment": "NEGATIVE",
            "SentimentScore": {
              "Mixed": 0,
              "Negative": 0.999996,
              "Neutral": 0.000004,
              "Positive": 0
            }
          }
        }
      ]
    },
    {
      "DescriptiveMentionIndex": [
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 53,
          "EndOffset": 60,
          "Score": 1,
          "GroupScore": 1,
          "Text": "service",
          "Type": "ATTRIBUTE",
          "MentionSentiment": {
            "Sentiment": "NEUTRAL",
```

```
            "SentimentScore": {
              "Mixed": 0.000033,
              "Negative": 0.000089,
              "Neutral": 0.993325,
              "Positive": 0.006553
            }
          }
        }
      ]
    }
  ],
  "File": "TargetSentimentInputDocs.txt",
  "Line": 2
}
  }
```

構文分析

構文分析を使用してドキュメント内の単語を解析し、ドキュメント内の各単語の品詞または構文関数を返します。ドキュメント内の名詞、動詞、形容詞などを識別できます。この情報を利用して、ドキュメントの内容をより深く理解し、単語間の関係を理解できます。

たとえば、ドキュメント内の名詞を検索し、その名詞に関連する動詞を探すことができます。「祖母がソファを動かした」のような文には、名詞「祖母」と「ソファ」、動詞「動く」があります。この情報を使用してアプリケーションを構築し、興味のある単語の組み合わせについてテキストを分析することができます。

Amazon Comprehend はソーステキストを解析して、テキスト内の個々の単語を検出し、分析を開始します。テキストが解析されると、各単語にはソーステキストに含まれる品詞が割り当てられます。

Amazon Comprehend では、以下の品詞を識別できます。

トークン	品詞
ADJ	形容詞 通常は名詞を修飾する単語。
ADP	接置詞

トークン	品詞
	前置詞または後置句の頭。
ADV	副詞 通常は動詞を修飾する単語。形容詞やその他の副詞を修飾することもできる。
AUX	助動詞 動詞句の動詞に付随する機能語。
CCONJ	等位接続詞 等位接続詞は、文中の単語、句または節を繋げる働きをし、1 つを他に従属させない。
CONJ	接続詞 接続詞は、文中の単語、句、節をつなげる。
DET	限定詞 特定の名詞句を指定する冠詞およびその他の単語。
INTJ	間投詞 感嘆符または感嘆符の一部として使用される単語。
NOUN	名詞 人物、場所、モノ、動物、アイデアを表す言葉。

トークン	品詞
NUM	数字 数字を表す単語 (通常は限定詞、形容詞、または代名詞) 。
O	その他 品詞カテゴリを割り当てることができない単語。
PART	不変化詞 意味を表すために他の語や句と結び付けられた機能語。
PRON	代名詞 名詞または名詞句の代わりとなる単語。
PROPN	固有名詞 特定の個人、場所、または物の名前を表す名詞。
PUNCT	句読点 テキストを区切るアルファベット以外の文字。
SCONJ	従属接続詞 従属節と文をつなぐ接続詞。従属接続詞の例には、「なぜなら」があります。

トークン	品詞
SYM	記号 ドル記号 ($) や数学記号などの単語のようなエンティティ。
VERB	動詞 出来事や行動を示す単語。

品詞の詳細については、「ユニバーサルディペンデンシーズ」ウェブサイトの「ユニバーサル POS タグ」を参照してください。

オペレーションは、テキスト内の単語とその単語が表す品詞を識別するトークンを返します。各トークンはソーステキスト内の 1 つの単語を表します。ソース内の単語の位置、テキスト内での単語の品詞、品詞の識別に関する Amazon Comprehend の確実性、およびソーステキストから解析された単語がわかります。

構文トークンのリスト構造は次のとおりです。ドキュメント内の単語ごとに 1 つの構文トークンが生成されます。

```
{
   "SyntaxTokens": [
      {
         "BeginOffset": number,
         "EndOffset": number,
         "PartOfSpeech": {
            "Score": number,
            "Tag": "string"
         },
         "Text": "string",
         "TokenId": number
      }
   ]
}
```

各トークンは以下の情報を提供します。

- `BeginOffset` と `EndOffset` —入力テキスト内の単語の位置を提供します。
- `PartOfSpeech` — 2 つの情報を提供します。 `Tag` は品詞を識別する情報、 `Score` は品詞の識別に関する Amazon Comprehend 構文の確実性を示しています。
- `Text` — 識別された単語を提供します。
- `TokenId` — トークンの識別子を提供します。識別子は、トークンリストにおけるトークンの位置です。

Amazon Comprehend Custom

Amazon Comprehend は、機械学習ベースの NLP ソリューションを構築するために必要なスキルを必要とすることなく、お客様それぞれのニーズに合わせてカスタマイズすることができます。Comprehend Custom は、自動機械学習 (AutoML) を使用して、お客様から提供されたトレーニングデータを使用して、お客様に代わってカスタム NLP モデルを構築します。

入力ドキュメント処理 — Amazon Comprehend は、カスタム分類とカスタムエンティティ認識のためのワンステップドキュメント処理をサポートしています。例えば、プレーンテキストドキュメントと半構造化ドキュメント (PDF ドキュメント、Microsoft Word ドキュメント、画像など) を組み合わせてカスタム分析ジョブに入力できます。詳細については、「ドキュメント処理」を参照してください。

カスタム分類 — カスタム分類モデル (分類子) を作成して、ドキュメントを専用のカテゴリに整理できます。分類ラベルごとに、そのラベルを最もよく表すドキュメントセットを作成し、それに基づいて分類子をトレーニングします。トレーニングした分類子は、ラベルの付いていないドキュメントセットに対していくつでも使用できます。ドキュメント数の制限はありません。コンソールを使用してコード不要の体験をしたり、最新の AWS SDK をインストールしたりすることができます。詳細については、「カスタム分類」を参照してください。

カスタムエンティティ認識 — カスタムエンティティ認識モデル (レコグナイザー) を作成し、特定の用語および名詞ベースの語句についてテキストを分析できます。レコグナイザーをトレーニングすることで、保険契約番号などの用語や、顧客のエスカレーションを示すフレーズを抽出することができます。モデルをトレーニングするには、エンティティのリストとそれらを含む文書一式を提供します。モデルのトレーニングが完了すると、モデルに対して分析ジョブを送信してカスタムエンティティを抽出できます。詳細については、「カスタムエンティティ認識」を参照してください。

トピックのモデリング

Amazon Comprehend を使用してドキュメントコレクションの内容を調べ、共通のテーマを判断できます。たとえば、Amazon Comprehend にニュース記事のコレクションを渡すと、スポーツ、政治、エンターテイメントなどのテーマを判断します。ドキュメント内のテキストには、注釈を付ける必要はありません。

Amazon Comprehend は、 潜在的ディリクレ配分法ベースの学習モデルを使用して、一連のドキュメント内のトピックを判断します。各ドキュメントを調べて、単語の文脈と意味を判断します。ドキュメントセット全体で同じ文脈に属することが多い単語の集合がトピックを構成します。

単語は、そのトピックがドキュメント内でどの程度一般的であるか、およびトピックが単語とどの程度親和性を持っているかに基づいて、ドキュメント内のトピックに関連付けられます。特定のドキュメント内のトピック分布に基づいて、同じ単語を異なるドキュメントの異なるトピックに関連付けることができます。

たとえば、主にスポーツに関する記事の「グルコース」という単語を「スポーツ」というトピックに割り当てて、「医学」に関する記事内の同じ単語を「医学」というトピックに割り当てることができます。

トピックに関連する各単語には、その単語がトピックの定義にどの程度役立つかを示す重みが付けられます。　重みは、ドキュメントセット全体で、その単語がトピック内の他の単語と比較して何回出現するかを示します。

最も正確な結果を得るには、Amazon Comprehend に可能な限り多くのコーパスを提供する必要があります。　最善の結果を得るには:

- 各トピックモデリングのジョブでは、少なくとも 1,000 件のドキュメントを使用する必要があります。
- 各ドキュメントは 3 センテンス以上必要です。
- ドキュメントの大部分が数値データで構成されている場合は、コーパスから削除する必要があります。

トピックモデリングは非同期プロセスです。　ドキュメントのリストは、StartTopicsDetectionJobオペレーションを使用して Amazon S3 バケットから Amazon Comprehend に送信します。 Amazon S3 レスポンスは Amazon S3 バケットに送信されます。入力バケットと出力バケットの両方を設定できます。ListTopicsDetectionJobs オペレーションを使用して送信したトピックモデリングジョブのリストを取得し、 DescribeTopicsDetectionJobオペレーションを使用して

ジョブに関する情報を表示します。Amazon S3 バケットに配信されるコンテンツには、カスタマーコンテンツが含まれている場合があります。機密データの削除の詳細については、「S3 バケットを空にする方法」または「S3 バケットを削除する方法」を参照してください。

ドキュメントは、UTF-8 形式のテキストファイルである必要があります。ドキュメントは 2 つの方法で送信できます。次の表にオプションを示します。

形式	説明
ファイルごとに 1 文書	各ファイルには 1 つの入力ドキュメントが含まれます。これはサイズの大きいドキュメントのコレクションに最適です。
1 行に 1 文書	入力は単一ファイルです。ファイル内の各行が文書とみなされます。これは、ソーシャルメディアへの投稿など、短いドキュメントに最適です。 各行は、改行 (LF、\ n)、キャリッジリターン (CR、\r)、またはその両方 (CRLF、\r\n) で終える必要があります。Unicode の行区切り文字 (u+2028) は行の終わりには使用できません。

詳細については、「InputDataConfig」データ型を参照してください。

Amazon Comprehend はドキュメントコレクションを処理すると、`topic-terms.csv` と `doc-topics.csv` の 2 つのファイルを含む圧縮アーカイブを返します。出力ファイルの詳細については、「」を参照してくださいOutputDataConfig。

最初の出力ファイル`topic-terms.csv` は、コレクション内のトピックのリストです。デフォルトでは、リストには、各トピックの上位の言葉が重みに応じてトピック別に含まれています。たとえば、Amazon Comprehend に新聞記事のコレクションを渡すと、コレクションの最初の 2 つのトピックを説明する次の内容が返されます。

トピック	言葉	重み
000	team	0.118533

トピック	言葉	重み
000	ゲーム	0.106072
000	player	0.031625
000	season	0.023633
000	play	0.02118
000	yard	0.024454
000	coach	0.016012
000	games	0.016191
000	football	0.015049
000	quarterback	0.014239
001	cup	0.205236
001	food	0.040686
001	minutes	0.036062
001	add	0.029697
001	tablespoon	0.028789
001	oil	0.021254
001	pepper	0.022205
001	teaspoon	0.020040
001	wine	0.016588
001	sugar	0.015101

重みは、特定のトピックに含まれる単語の確率分布を表します。Amazon Comprehend は各トピックの上位 10 語のみを返すため、重みの合計は 1.0 にはなりません。まれに、1 つのトピックに含まれる単語が 10 語未満の場合、重みの合計が 1.0 になります。

単語は、すべてのトピックでの出現率を考慮して、その識別力によってソートされます。通常、これはキーワードの重みと同じですが、表の「play」や「yard」という単語など、場合によっては重みと異なる順序になることがあります。

返されるトピックの数を指定できます。たとえば、Amazon Comprehend に 25 個のトピックを返すようにリクエストすると、コレクション内で最も目立つ 25 個のトピックが返されます。Amazon Comprehend はコレクション内で最大 100 トピックを検出できます。トピックの数は、分野に関する自分の知識に基づいて選択してください。正しい数にたどり着くには、ある程度の実験が必要な場合があります。

2 つ目のファイル `doc-topics.csv` には、トピックに関連するドキュメントと、そのトピックに関係するドキュメントの割合が一覧表示されます。`ONE_DOC_PER_FILE` を指定した場合、ドキュメントはファイル名で識別されます。`ONE_DOC_PER_LINE` を指定した場合、ドキュメントはファイル名とファイル内の 0 で始まる行番号で識別されます。たとえば、Amazon Comprehend は、1 ファイルにつき 1 つのドキュメントで送信されたドキュメントのコレクションに対して以下を返す場合があります。

ドキュメント	トピック	割合
sample-doc1	000	0.999330137
sample-doc2	000	0.998532187
sample-doc3	000	0.998384574
...		
sample-docN	000	3.57E-04

Amazon Comprehend は、「MBM の語彙化リストデータセット」を利用します。これは、オープンデータベースライセンス (ODBL) v1.0 の下、こちらで入手可能です。

ドキュメント処理モード

Amazon Comprehend は 3 つのドキュメント処理モードをサポートしています。どのモードを選択するかは、処理の必要があるドキュメントの数と、結果をどれだけ早く表示する必要があるかによって異なります。

- 単一ドキュメント同期 — 単一のドキュメントで Amazon Comprehend を呼び出し、アプリケーション (またはコンソール) にすぐに配信される同期レスポンスを受け取ります。
- マルチドキュメント同期 — 最大 25 個のドキュメントのコレクションを使用して Amazon Comprehend API を呼び出し、同期レスポンスを受け取ります。
- 非同期バッチ — 大量のドキュメントの場合は、ドキュメントを Amazon S3 バケットに入れ、(コンソールまたは API オペレーションを使用して) 非同期ジョブを開始してドキュメントを分析します。Amazon Comprehend は、リクエストで指定した S3 バケット/フォルダで分析結果を保存します。

トピック

- 単一ドキュメント処理
- 複数ドキュメントの同期処理
- 非同期バッチ処理

単一ドキュメント処理

同期オペレーション: 単一のドキュメントで分析を実行でき、分析結果をアプリケーションに直接返します。単一ドキュメントオペレーションは、一度に 1 つのドキュメントで動作するインタラクティブなアプリケーションを作成するときに使用します。

同期 API オペレーションの詳細については、組み込みモデルを使用したリアルタイム分析 (コンソール用) と API を使用したリアルタイムの分析 を参照してください。

複数ドキュメントの同期処理

処理するドキュメントが複数ある場合は、`Batch*` API オペレーションを使用して一度に複数のドキュメントを Amazon Comprehend に送信できます。リクエストごとに最大 25 件のドキュメントを送信できます。Amazon Comprehend は、リクエスト内のドキュメントごとに 1 つずつ、レスポンスのリストを返します。これらの操作で行われるリクエストは同期的に行われます。アプリケーションはこの操作を呼び出し、サービスのレスポンスを待ちます。

Batch* 操作を使用することは、リクエスト内の各ドキュメントに対して単一ドキュメント API を呼び出すことと同じです。これらの API を使用すると、アプリケーションのパフォーマンスが向上します。

各 API への入力は、処理するドキュメントを含む JSON 構造体です。BatchDetectDominantLanguage 以外のすべての操作では、入力言語を設定する必要があります。リクエストごとに入力言語を 1 つのみ設定できます。たとえば、以下は BatchDetectEntities 操作への入力です。これには 2 つのドキュメントが含まれており、英語で書かれています。

```
{
   "LanguageCode": "en",
   "TextList": [
      "I have been living in Seattle for almost 4 years",
      "It is raining today in Seattle"
   ]
}
```

Batch* 操作からの応答には、ResultList と ErrorList の 2 つのリストが含まれます。ResultList には、正常に処理されたドキュメントごとに 1 つのレコードが含まれます。リクエスト内の各ドキュメントの結果は、そのドキュメントに対して 1 つのドキュメント操作を実行した場合に得られる結果と同じです。各ドキュメントの結果には、入力ファイル内のドキュメントの順序に基づいてインデックスが割り当てられます。BatchDetectEntities オペレーションからのレスポンスは次のようになります。

```
{
   "ResultList"  : [
      {
         "Index": 0,
         "Entities": [
            {
               "Text": "Seattle",
               "Score": 0.95,
               "Type": "LOCATION",
               "BeginOffset": 22,
               "EndOffset": 29
            },
            {
               "Text": "almost 4 years",
               "Score": 0.89,
               "Type": "QUANTITY",
```

```
                "BeginOffset": 34,
                "EndOffset": 48
            }
         ]
      },
      {
         "Index": 1,
         "Entities": [
            {
              "Text": "today",
              "Score": 0.87,
              "Type": "DATE",
              "BeginOffset": 14,
              "EndOffset": 19
            },
            {
               "Text": "Seattle",
               "Score": 0.96,
               "Type": "LOCATION",
               "BeginOffset": 23,
               "EndOffset": 30
            }
         ]
      }
   ],
   "ErrorList": []
}
```

リクエストでエラーが発生した場合、レスポンスにはエラーを含むドキュメントを識別するErrorList が含まれます。ドキュメントは入力リスト内のインデックスによって識別されます。たとえば、BatchDetectLanguage 操作の次の入力には処理できないドキュメントが含まれています。

```
{
   "TextList": [
     "hello friend",
     "$$$$$$",
     "hola amigo"
   ]
}
```

Amazon Comprehend からのレスポンスには、エラーを含むドキュメントを識別するエラーリストが含まれています。

```
{
    "ResultList": [
        {
          "Index": 0,
          "Languages":[
            {
              "LanguageCode":"en",
              "Score": 0.99
            }
          ]
        },
        {
          "Index": 2
          "Languages":[
            {
              "LanguageCode":"es",
              "Score": 0.82
            }
          ]
        }
    ],
    "ErrorList": [
      {
        "Index": 1,
        "ErrorCode": "InternalServerException",
        "ErrorMessage": "Unexpected Server Error. Please try again."
      }
    ]
}
```

使用できる API オペレーションの詳細については、「リアルタイムバッチ API」を参照してください。

非同期バッチ処理

大量のドキュメントや大量のドキュメントコレクションを分析するには、Amazon Comprehend の非同期オペレーションを使用します。

ドキュメントのコレクションを分析するには、通常、以下のステップを実行します。

1. ドキュメントを Amazon S3 バケットに保存します。
2. 1 つ以上の分析ジョブを開始してドキュメントを分析します。
3. 分析ジョブの進行状況をモニタリングします。
4. ジョブが完了したら、S3 バケットから分析結果を取得します。

同期 API オペレーションの使用に関する詳細は、コンソールを使用した分析ジョブの実行 (コンソール用) と API を使用した非同期分析ジョブ を参照してください。

Amazon Comprehend でサポートされている言語

Amazon Comprehend は、多様な機能で種々の言語をサポートしています。次の表に、サポートされている言語とサポートされる機能を示します。

トピック

- サポートされている言語
- Amazon Comprehend の機能でサポートされている言語

サポートされている言語

Amazon Comprehend (主要言語の検出機能を除く) は、1 つまたは複数の機能で以下の言語をサポートしています。

Code	言語
de	ドイツ語
en	英語
es	スペイン語
it	イタリア語
pt	ポルトガル語
fr	フランス語
ja	日本語
ko	韓国語
hi	ヒンディー語
ar	アラビア語
zh	簡体字中国語
zh-TW	繁体字中国語

> **Note**
>
> Amazon Comprehend は、RFC 5646 の識別子を使用して言語を識別します。2 文字の ISO 639-1 識別子があり、必要に応じて地域のサブタグがある場合は、それを使用します。それ以外の場合は ISO 639-2 の 3 文字コードを使用します。
> RFC 5646 の詳細は、IETF ツールウェブサイトの「言語識別用タグ」を参照してください。

Amazon Comprehend の機能でサポートされている言語

機能	サポートされている言語
主要言語	主要言語 を参照してください。
エンティティ	サポートされているすべての言語
キーフレーズ	サポートされているすべての言語
PII エンティティの検出	英語とスペイン語。
PII エンティティのラベル付け	英語とスペイン語。
感情	サポートされているすべての言語
ターゲット感情	英語。
構文分析	ドイツ語 (de)、英語 (en)、スペイン語 (es)、フランス語 (fr)、イタリア語 (it)、ポルトガル語 (pt)。
トピックのモデリング	使用言語に依存しません。中国語、日本語、韓国語などの文字ベースの言語はサポートされていません。
カスタム分類	プレーンテキストモデルは、ドイツ語 (de)、英語 (en)、スペイン語 (es)、フランス語 (fr)、イタリア語 (it)、ポルトガル語 (pt) の各言語をサポートしています。

機能	サポートされている言語
	ネイティブドキュメントモデルは英語ドキュメントのみサポートします。
カスタムエンティティ認識	ドイツ語 (de)、英語 (en)、スペイン語 (es)、フランス語 (fr)、イタリア語 (it)、ポルトガル語 (pt)。 PDF と Word のカスタムエンティティ認識は、英語の文書のみサポートします。

セットアップ

Amazon Comprehend を初めて使用する場合は、事前に以下のタスクをすべて実行しておいてください。

タスクのセットアップ

- にサインアップする AWS アカウント
- 管理アクセスを持つユーザーを作成する
- AWS Command Line Interface （ AWS CLI) のセットアップ
- プログラマチックアクセス権を付与する

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] を選んで、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理できます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、 を保護し AWS アカウントのルートユーザー、 を有効にして AWS IAM Identity Center、日常的なタスクにルートユーザーを使用しないように管理ユーザーを作成します。

のセキュリティ保護 AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS Management Consoleとして にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドの「ルートユーザーとしてサインインする」を参照してください。
2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、「IAM ユーザーガイド」の AWS アカウント 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。
2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、「 ユーザーガイド」の「デフォルト でユーザーアクセス IAM アイデンティティセンターディレクトリを設定するAWS IAM Identity Center 」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、「 AWS サインイン ユーザーガイド」の AWS 「 アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。
2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの参加」を参照してください。

AWS Command Line Interface （ AWS CLI) のセットアップ

入門演習のステップを実行する AWS CLI ために は必要ありません。ただし、このガイドの他の実習では必要になります。必要に応じて、このステップをスキップして に移動しAmazon Comprehend の開始方法、 AWS CLI 後でセットアップできます。

をインストールして設定するには AWS CLI

1. をインストールします AWS CLI。手順については、『AWS Command Line Interface ユーザーガイド』の次のトピックを参照してください。

   の最新バージョンのインストールまたは更新 AWS Command Line Interface
2. AWS CLIを設定します。手順については、『AWS Command Line Interface ユーザーガイド』の次のトピックを参照してください。

   AWS Command Line Interfaceの設定

プログラマチックアクセス権を付与する

ユーザーが の AWS 外部で を操作する場合は、プログラムによるアクセスが必要です AWS Management Console。プログラムによるアクセスを許可する方法は、 にアクセスするユーザーのタイプによって異なります AWS。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM Identity Center で管理されているユーザー)	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 ユーザーガイド」の AWS CLI 「 を使用するための の設定 AWS IAM Identity CenterAWS Command Line Interface 」を参照してください。 • AWS SDKs 、ツール、AWS APIs「 SDK とツールのリファレンスガイド」の「IAM Identity Center 認証」を参照してください。AWS SDKs
IAM	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールのリファレンスガイド」の「長期的な認証情報を使

プログラマチックアクセス権を必要とするユーザー	目的	方法
		用した認証」を参照してください。 AWS SDKs • AWS APIsユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。

Amazon Comprehend の開始方法

次の演習では、Amazon Comprehend コンソールを使用して非同期エンティティ検出ジョブを作成して実行します。この演習は、Amazon Simple Storage Service (Amazon S3) に精通していることを前提としています。より簡単な例については、「組み込みモデルを使用したリアルタイム分析」を参照してください。

エンティティ検出ジョブを作成するには

1. AWS Management Console にサインインして、Amazon Comprehend コンソール (https://console. .amazon.com/comprehend/) を開きます
2. 左側のメニューから、[分析ジョブ] を選択し、[ジョブの作成] を選択します。
3. [Job 設定] で、ジョブに名前を付けます。名前は、アカウントと リージョン内で一意である必要があります。
4. [分析タイプ]で、[エンティティ]を選択します。
5. [言語] では、入力ドキュメントの言語を選択します。
6. [入力データ] の [データソース] で [ドキュメント例] を選択します。コンソールは S3 ロケーションをパブリックサンプルを含むフォルダに設定します。
7. [出力データ] の [S3 の場所] に、出力ファイルの URL またはフォルダの場所を Amazon S3 に貼り付けます。
8. [アクセス許可]セクションで、[IAM ロールを作成]を選択します。コンソールは、Amazon Comprehend が入出力バケットにアクセスするための適切な権限を持つ新しい IAM ロールを作成します。
9. フォームの入力が完了したなら、[ジョブの作成] を選択してトピック検出ジョブを作成し、開始します。

   新しいジョブがジョブリストに表示され、ステータスフィールドにはジョブのステータスが表示されます。このフィールドは、`IN_PROGRESS` 処理中のジョブ、`COMPLETED` 正常に終了したジョブ、`FAILED` エラーのあるジョブのいずれでもかまいません。
10. ジョブを選択して [ジョブの詳細パネル] を開きます。
11. [出力] の [出力データの場所] から、Amazon S3 コンソールを開くリンクを選択します。
12. Amazon S3 コンソールで、[ダウンロード] を選択し、`output.tar.gz` ファイルを保存します。
13. ファイルを解凍し、JSON ファイルとして保存します。

14. 検出された各エンティティのエンティティタイプとフィールドの説明については、「the section called “エンティティ”」を参照してください。

Amazon Comprehend コンソールを使用した分析

Amazon Comprehend コンソールを使用して、ドキュメントをリアルタイムで分析したり、非同期分析ジョブを実行したりできます。

組み込みモデルによるリアルタイム分析を使用すると、エンティティの認識、キーフレーズの抽出、主要言語の検出、PII の検出、感情の判別、ターゲット感情の分析、構文の分析を行うことができます。

組み込みモデルを使用して分析ジョブを実行すると、エンティティ、イベント、フレーズ、主要言語、感情、ターゲット感情、個人を特定できる情報 (PII) などのインサイトを見つけることができます。トピックモデリングジョブを実行することもできます。

コンソールは、カスタムモデルを使用したリアルタイム分析と非同期分析もサポートしています。詳細については、「カスタム分類」および「カスタムエンティティ認識」を参照してください。

トピック

- 組み込みモデルを使用したリアルタイム分析
- コンソールを使用した分析ジョブの実行

組み込みモデルを使用したリアルタイム分析

Amazon Comprehend コンソールを利用して、UTF-8 でエンコードされたテキストドキュメントのリアルタイム分析を実行することができます。ドキュメントは英語でも、Amazon Comprehend がサポートする他の言語でもかまいません。分析を確認できるように、結果がコンソールに表示されます。

ドキュメントの分析を開始するには、 にサインイン AWS Management Console し、Amazon Comprehend コンソールを開きます。

サンプルテキストを独自のテキストに置き換え、[分析] を選択することで、テキストの分析を行うことができます。分析中のテキストの下にある [結果] ペインには、テキストに関する詳細情報が表示されます。

組み込みモデルを使用したリアルタイム分析を実行する

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/comprehend/ で Amazon Comprehend コンソールを開きます。

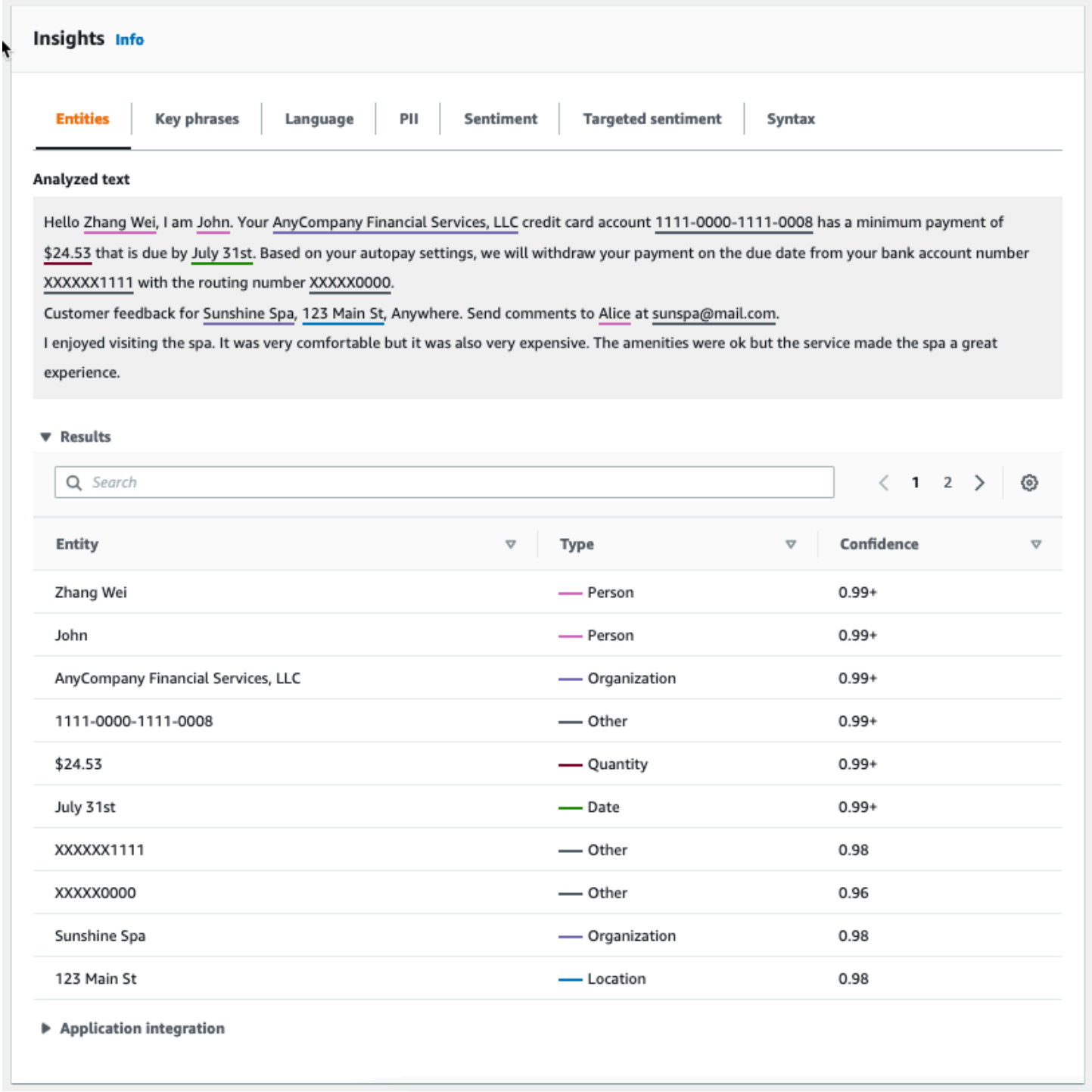
2. 左側のメニューで、[リアルタイム分析] を選択します。
3. [入力タイプ] の [分析タイプ] で [ビルトイン] を選択します。
4. 分析するテキストを入力します。
5. [分析] を選択します。コンソールのインサイトパネルにテキスト分析の結果が表示されます。[インサイトパネル] には、インサイトタイプごとにタブがあります。以下のセクションでは、インサイトタイプの結果について説明します。

トピック

- エンティティ
- キーフレーズ
- [言語]
- 個人を特定できる情報 (PII)
- 感情
- ターゲット感情
- 構文

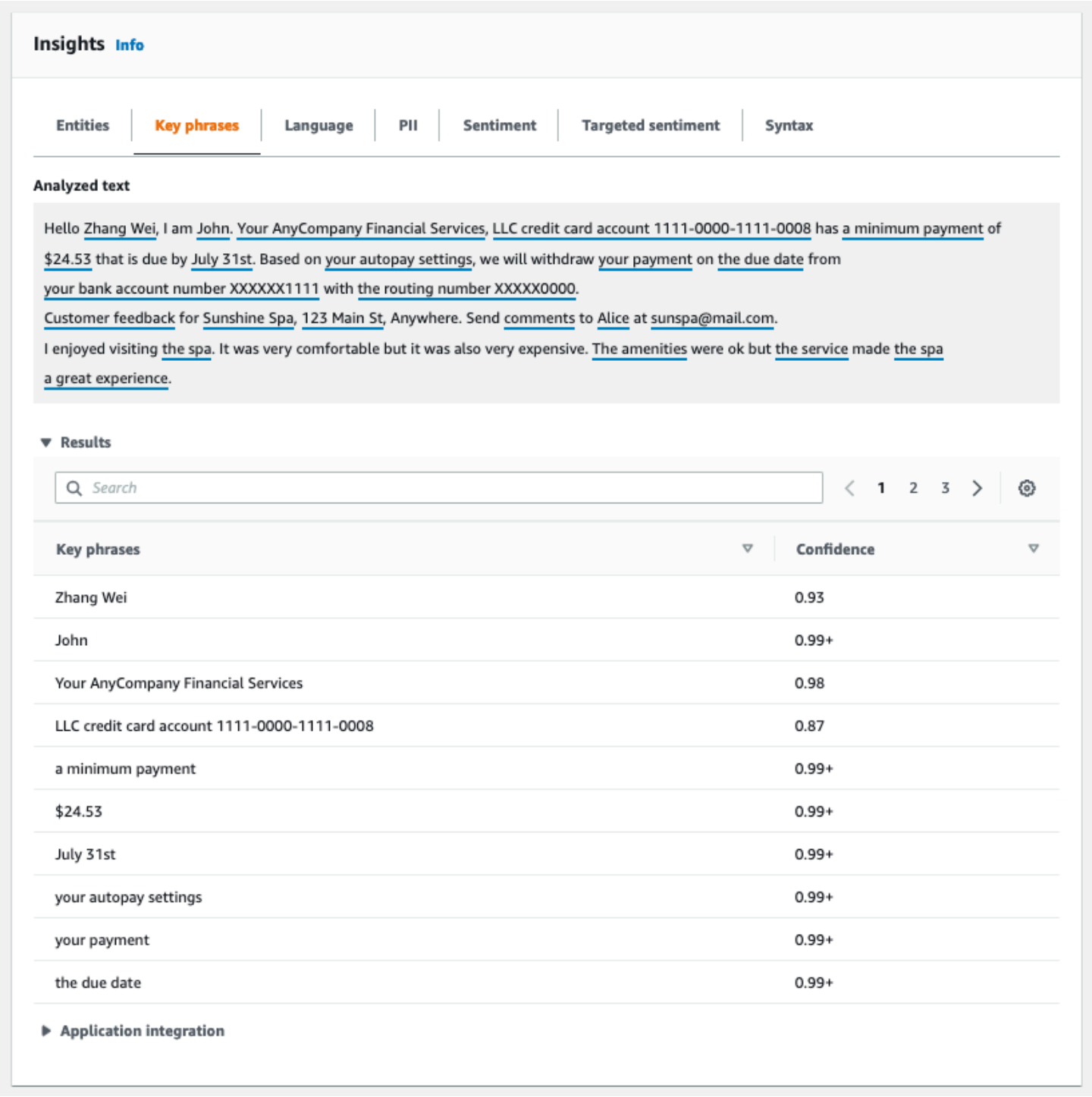
エンティティ

[エンティティ] タブは、Amazon Comprehend が入力テキストで検出した各エンティティとそのカテゴリ、信頼度のリストです。結果は、組織、場所、日付、人物など、さまざまなエンティティに応じて色分けされています。詳細については、「エンティティ」を参照してください。



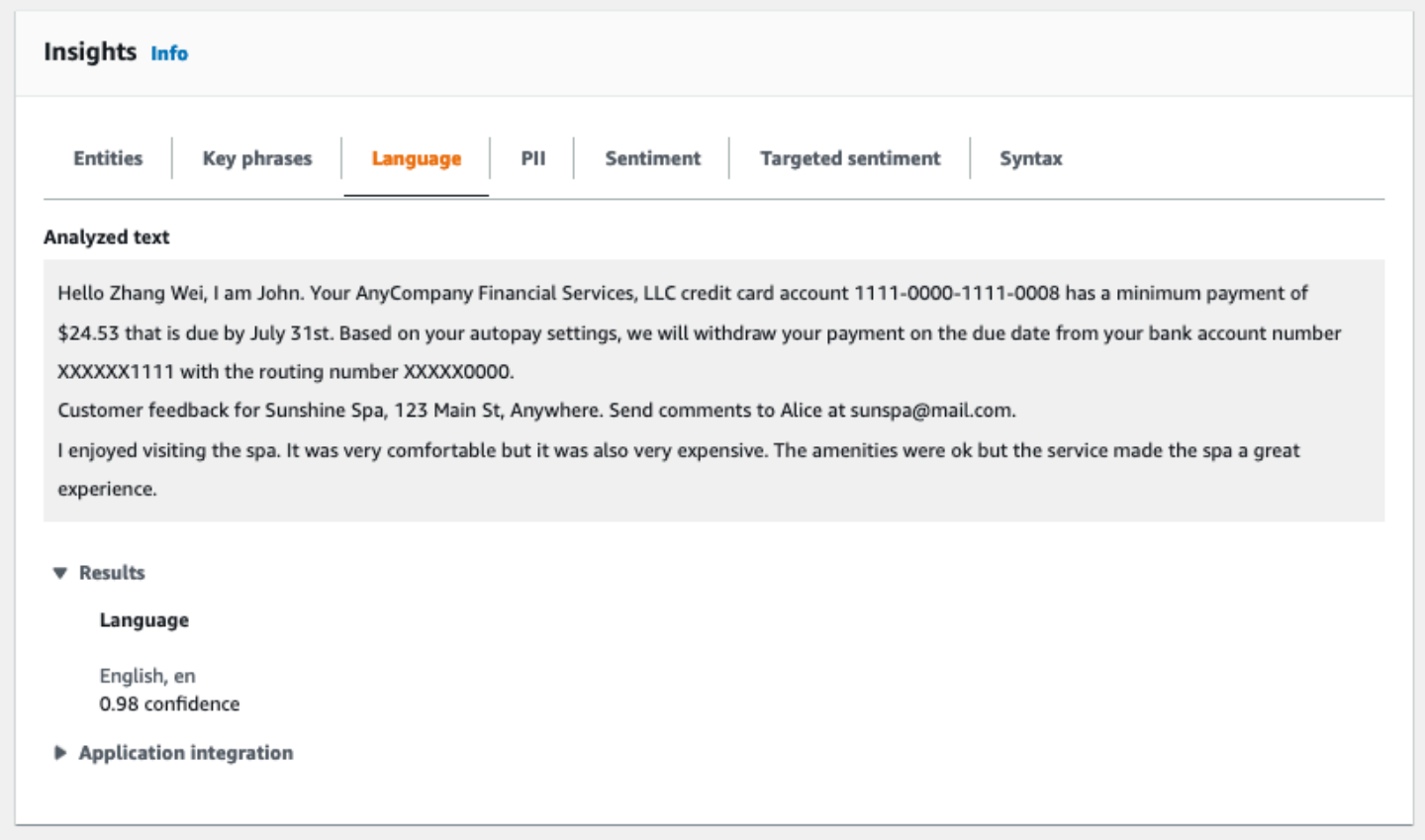
キーフレーズ

[キーフレーズ] タブは、Amazon Comprehend が入力テキストで検出した主な名詞フレーズと、その信頼度のリストです。詳細については、「キーフレーズ」を参照してください。



[言語]

[言語] タブには、テキストの主要言語と、主要言語が正しく検出されたかどうかの Amazon Comprehend 信頼度が表示されます。Amazon Comprehend は 100 通りの言語を認識できます。詳細については、「主要言語」を参照してください。



個人を特定できる情報 (PII)

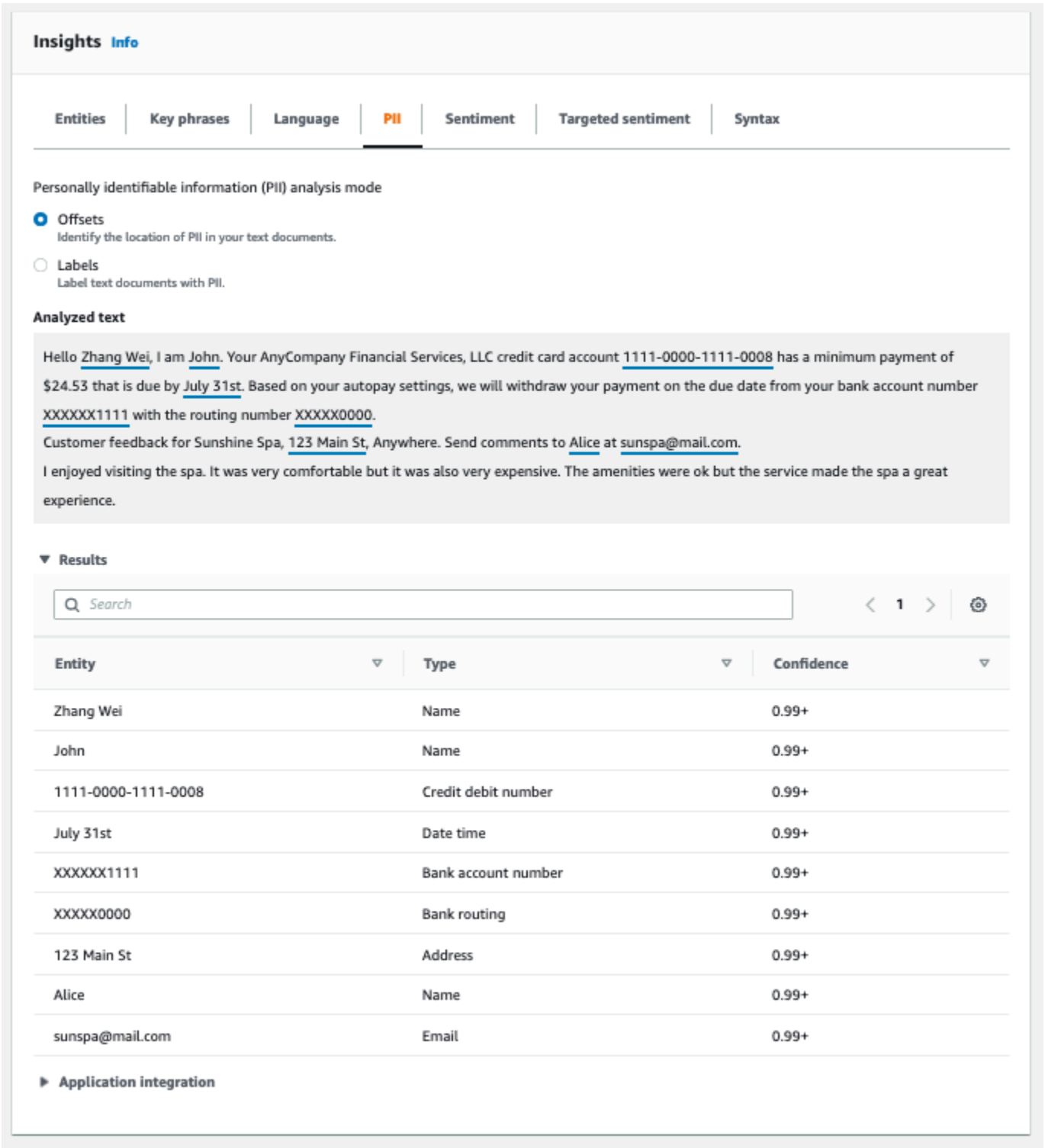
[PII] タブは、個人を特定できる情報 (PII) を含む入力テキスト内のエンティティのリストです。PII エンティティとは、住所や銀行口座番号、電話番号など、個人の特定に使用できる個人情報をテキスト形式の参照です。詳細については、「PII エンティティの検出」を参照してください。

[PII] タブには 2 つの分析モードがあります。

- オフセット
- ラベル

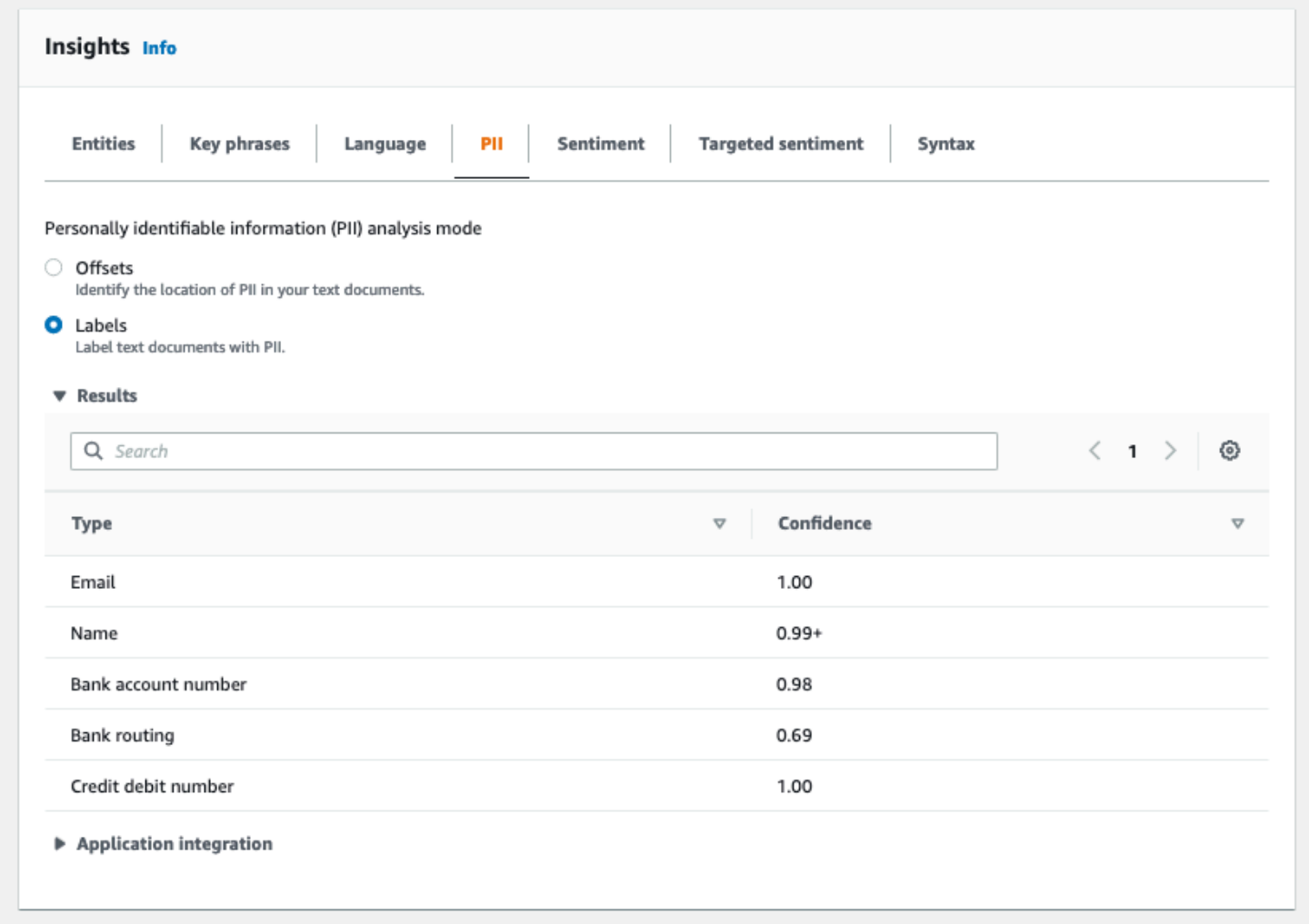
オフセット

オフセット 分析モードでは、テキストドキュメント内の PII の位置が特定されます。詳細については、「PII エンティティを検索します。」を参照してください。


Insights Info
Entities
Key phrases
Language
PII
Sentiment
Targeted sentiment
Syntax
Personally identifiable information (PII) analysis mode
Offsets
Identify the location of PII in your text documents.
Labels
Label text documents with PII.
Analyzed text
Hello Zhang Wei, I am John. Your AnyCompany Financial Services, LLC credit card account 1111-0000-1111-0008 has a minimum payment of $24.53 that is due by July 31st. Based on your autopay settings, we will withdraw your payment on the due date from your bank account number XXXXXX1111 with the routing number XXXXX0000.
Customer feedback for Sunshine Spa, 123 Main St, Anywhere. Send comments to Alice at sunspa@mail.com.
I enjoyed visiting the spa. It was very comfortable but it was also very expensive. The amenities were ok but the service made the spa a great experience.
Results
Search
1
Entity
Type
Confidence
Zhang Wei
Name
0.99+
John
Name
0.99+
1111-0000-1111-0008
Credit debit number
0.99+
July 31st
Date time
0.99+
XXXXXX1111
Bank account number
0.99+
XXXXX0000
Bank routing
0.99+
123 Main St
Address
0.99+
Alice
Name
0.99+
sunspa@mail.com
Email
0.99+
Application integration

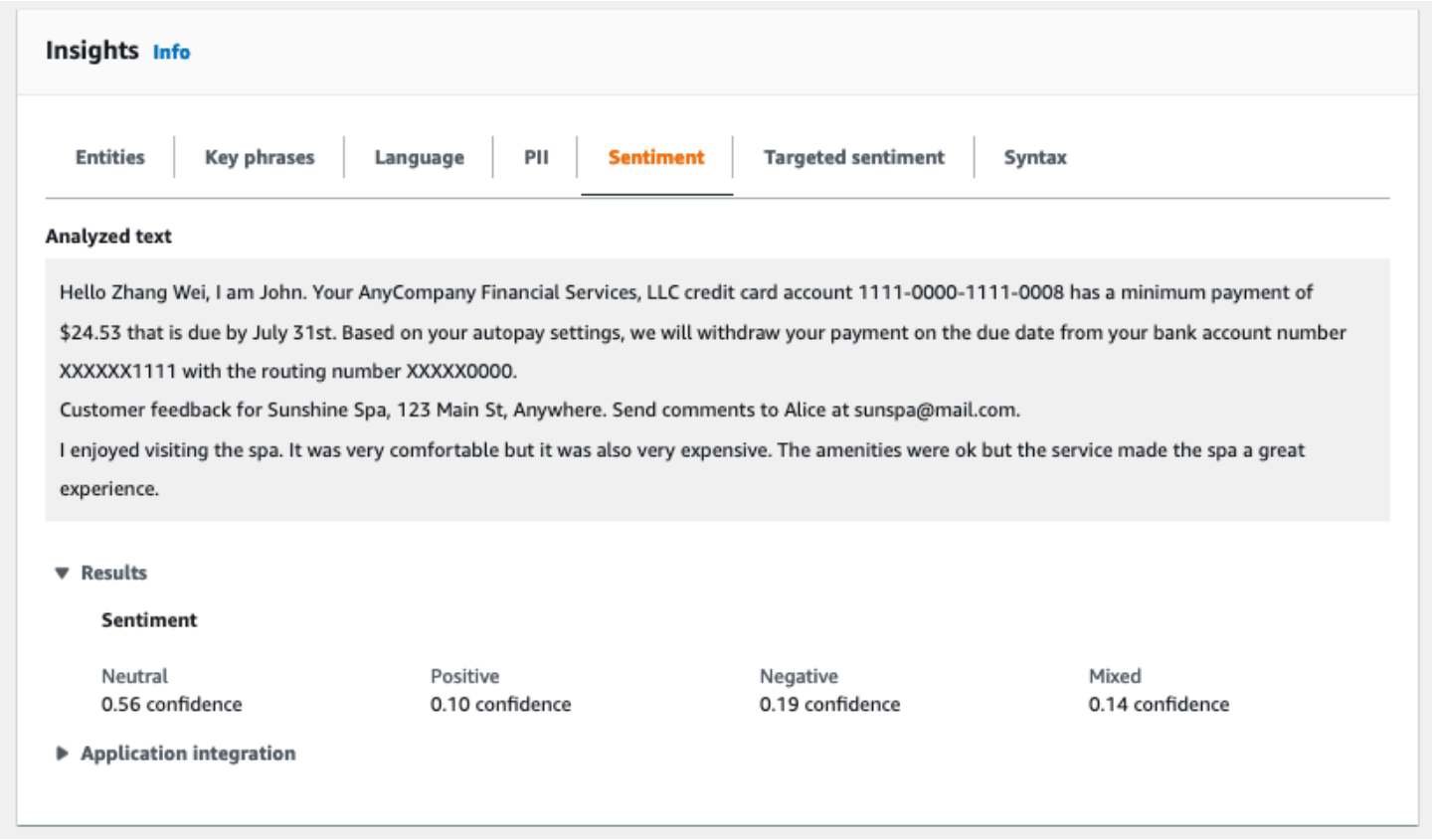
ラベル

Labels 分析モードでは、テキストドキュメントに PII が存在するかどうかがチェックされ、特定された PII エンティティタイプのラベルが返されます。詳細については、「PII エンティティのラベル付け」を参照してください。



感情

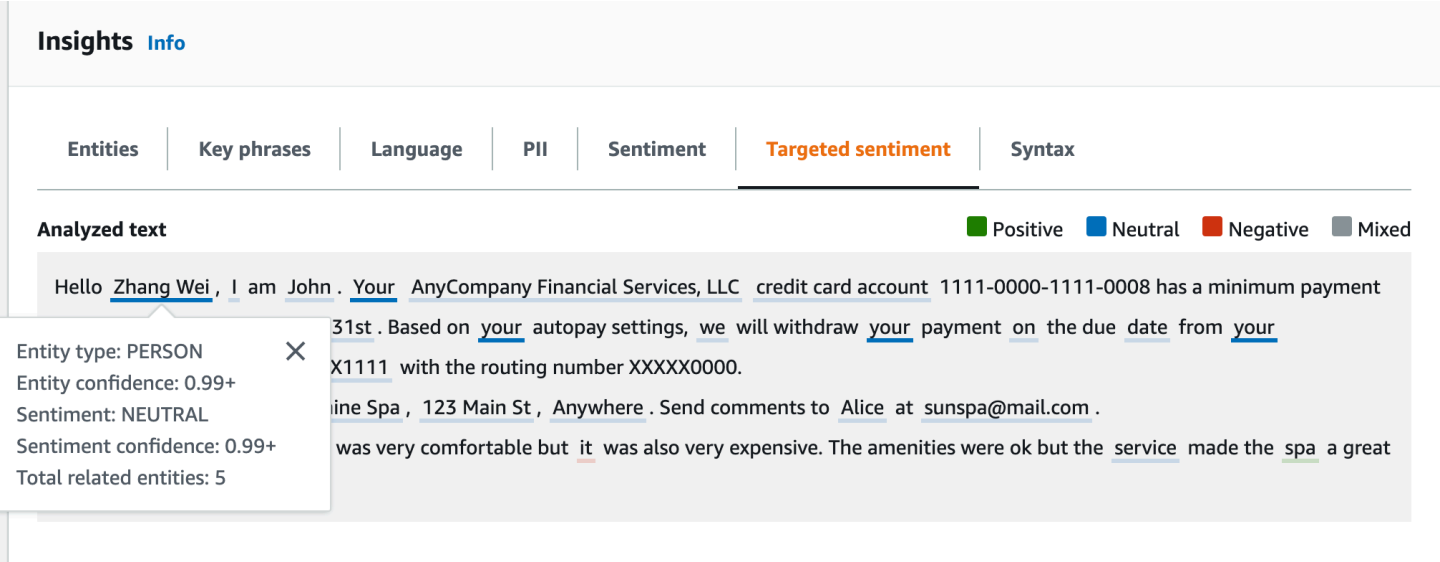
[感情] タブには、テキストの主要感情が示されます。感情は、中立、肯定的、否定的、混在のいずれかに評価されます。この場合、各感情には信頼度評価があり、それぞれ優勢な感情に対する Amazon Comprehend の評価です。詳細については、「感情」を参照してください。



ターゲット感情

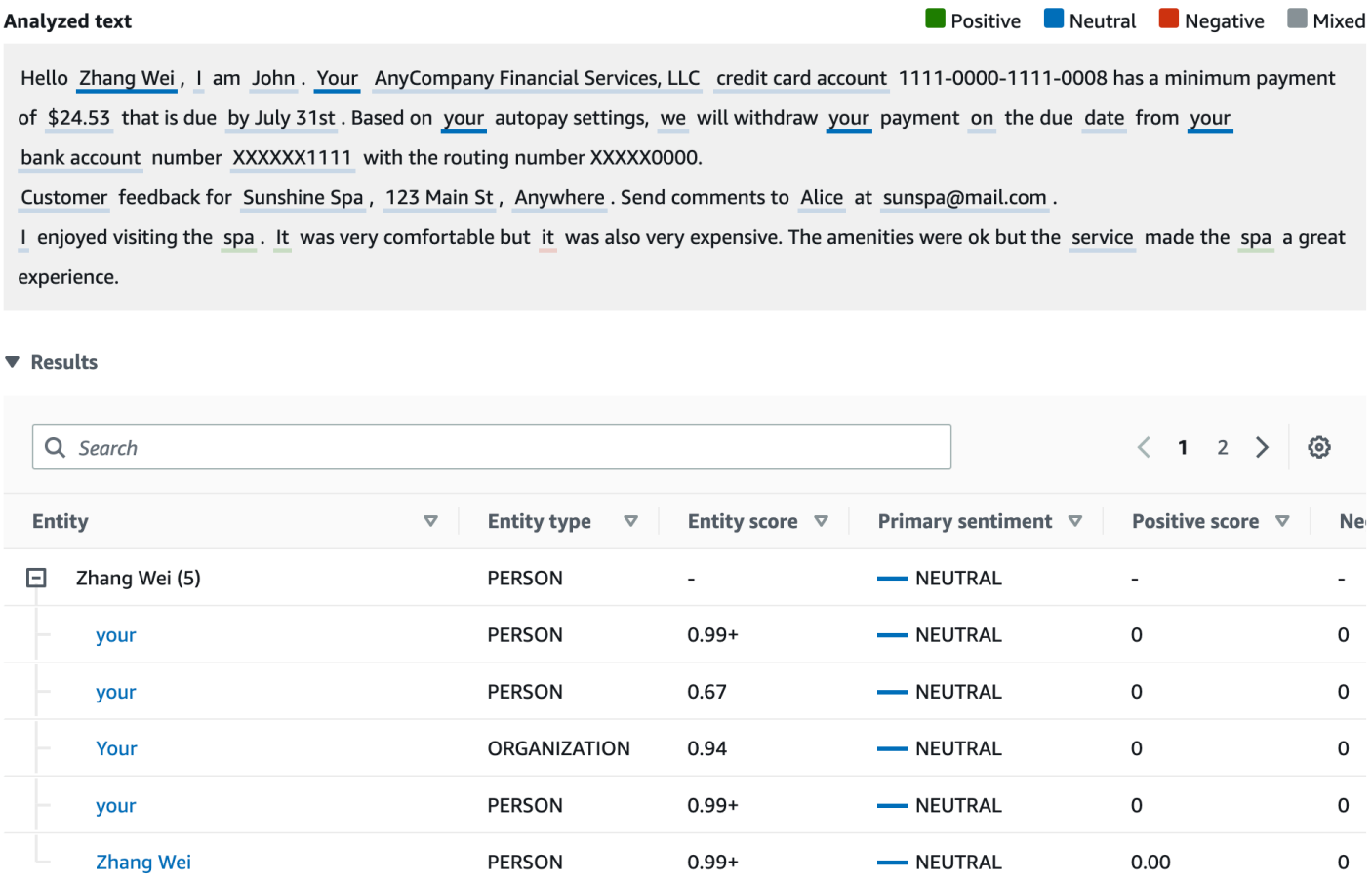
ターゲット感情分析では、テキストで言及 (メンション) されているエンティティについて表われている感情を特定できます。Amazon Comprehend は、エンティティへのメンションがあるたびに、信頼度やその他の情報とともに感情評価を割り当てます。感情評価は、中立、肯定的、否定的、混在のいずれかに評価されます。

コンソールの [分析テキスト] パネルでは、分析対象の各エンティティに下線が引かれます。下線が引かれたテキストの色は、エンティティの全体的な感情を示しています。エンティティの上にカーソルを置くと、コンソールのポップアップウィンドウに追加の情報が表示されます。



[結果] テーブルには、各エンティティに関するその他の詳細が表示されます。同じエンティティに同一指示と呼ばれる複数のメンションがある場合、この表には、メインのエンティティに関連付けられた折りたたみ可能な行セットとしてそれらメンションが表示されます。

以下の例では、エンティティは Zhang Wei という名前の人物です。ターゲット感情分析では、your のメンションはすべて同一人物への言及であることが認識されています。コンソールには、これらのメンションがメインエンティティのサブエントリとして表示されます。

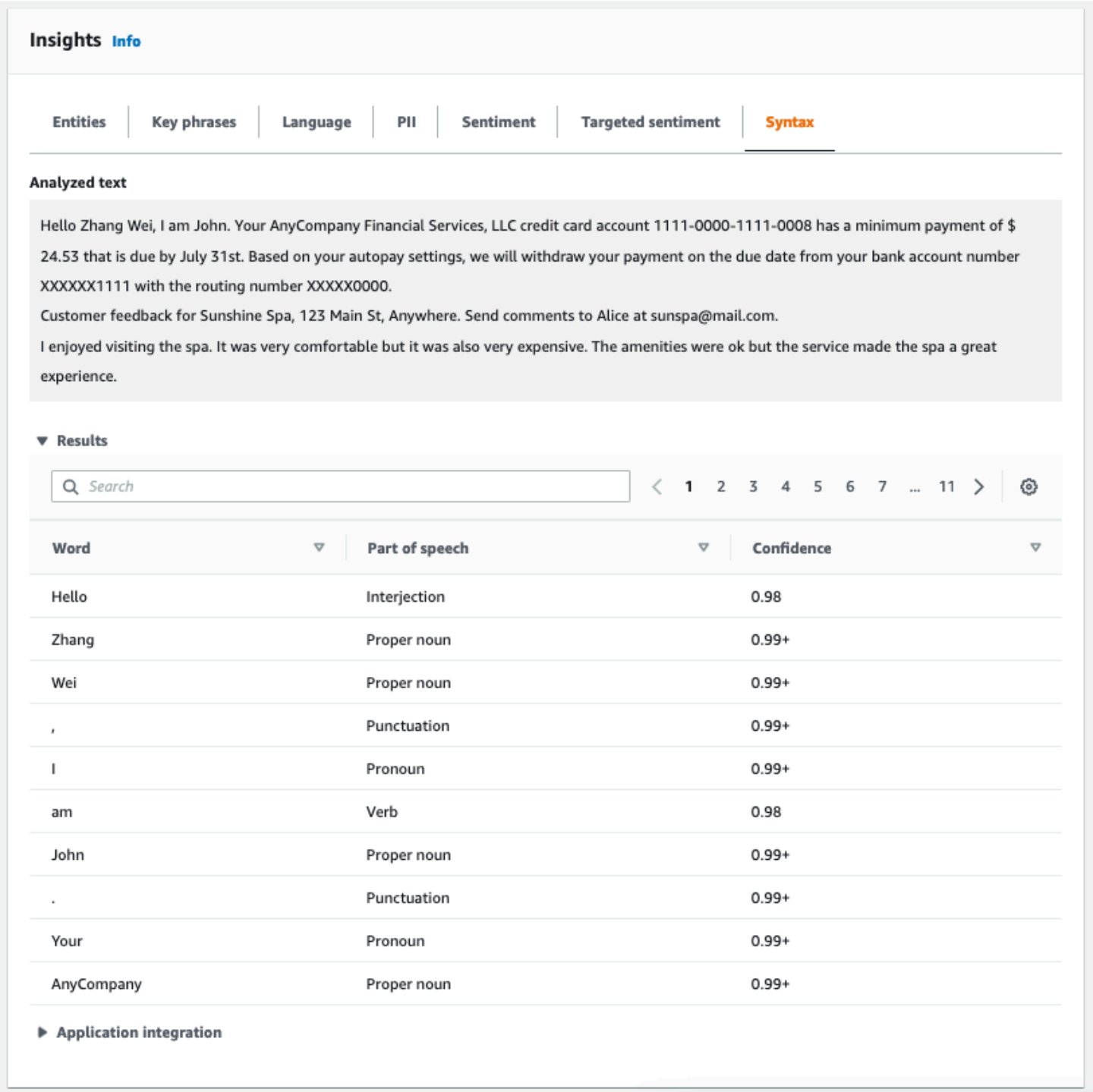


分析するテキストにターゲット感情 エンティティタイプ がない場合は、空の結果フィールドが表示されます。

コンソールを使用してターゲット感情のリアルタイム分析を行う方法については、「コンソールを使用したリアルタイム分析」を参照してください。

構文

[構文] タブには、テキスト内の各要素の内訳と、品詞および関連する信頼度スコアが表示されます。詳細については、「構文分析」を参照してください。



コンソールを使用した分析ジョブの実行

Amazon Comprehend コンソールを使用して、非同期分析ジョブを作成および管理できます。ジョブは Amazon S3 に保存されているドキュメントを分析して、イベント、フレーズ、主要言語、感情、個人を特定できる情報 (PII) などのエンティティを見つけます。

分析ジョブを作成するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/comprehend/ で Amazon Comprehend コンソールを開きます。
2. 左側のメニューから、[分析ジョブ] を選択し、[ジョブの作成] を選択します。
3. [ジョブの設定] で、分析ジョブに一意の名前を付けます。
4. [分析タイプ] には、[ビルトイン]分析タイプのいずれかを選択します。

   プライマリ言語またはトピックモデリングを選択した場合は、次のステップをスキップできます。
5. 選択した分析タイプに応じて、コンソールには以下の追加フィールドが 1 つ以上表示されます。

   - 主要言語とトピックモデリングを除くすべての組み込み分析タイプには言語が必要です。

     入力ドキュメントの言語を選択します。
   - イベント分析タイプには [ターゲットイベントタイプ] が必要です。

     入力ドキュメントで検出するイベントの種類を選択します。サポートされるイベントタイプについては、「イベントタイプ」を参照してください。
   - PII 分析タイプには [PII 検出設定] が必要です。

     出力モードを選択します。PII 検出設定の詳細については、「PII エンティティの検出」を参照してください。
6. [入力データ] で、入力文書が Amazon S3 のどこにあるかを指定します。
   - 独自の文書を分析するには、[マイドキュメント] を選択し、[S3 を参照] を選択して、ファイルを含むバケットまたはフォルダへのパスを指定します。
   - Amazon Comprehend が提供するサンプルを分析するには、「サンプル文書」を選択します。この場合、Amazon Comprehend は によって管理されるバケットを使用し AWS、場所を指定しません。
7. (オプション) [入力フォーマット] では、入力ファイルのフォーマットのいずれかを指定します。

   - 1 ファイルに 1 文書 — 各ファイルには 1 つの入力文書が含まれます。これはサイズの大きいドキュメントのコレクションに最適です。
   - 1 行に 1 文書 — 入力は 1 つ以上のファイルです。ファイル内の各行は 1 つのドキュメントとみなされます。これは、ソーシャルメディアへの投稿など、短いドキュメントに最適です。各行は、改行 (LF、\ n)、キャリッジリターン (CR、\r)、またはその両方 (CRLF、\r\n) で終え

る必要があります。行の終了に、UTF-8 の行区切り文字 (u+2028) を使用することはできません。

8. [出力データ] で [S3 をブラウズ] を選択します。分析によって生成された出力データを Amazon Comprehend に書き込ませる Amazon S3 バケットまたはフォルダを選択します。
9. (オプション) ジョブの出力結果を暗号化するには、[暗号化] を選択します。次に、現在のアカウントに関連付けられた KMS キーを使用するか、別のアカウントの KMS キーを使用するかを選択します。
   - 現在のアカウントに関連付けられているキーを使用している場合は、KMS キー ID のキーエイリアスまたは ID を選択します。
   - 別のアカウントに関連付けられているキーを使用している場合は、KMS キー ID の下にキーエイリアス の ARN または ID を入力します。

   > **Note**
   >
   > KMS キーの作成と使用や関連する暗号化の詳細については、「キー管理サービス (KMS)」を参照してください。

10. アクセス権で、次のような IAM ロールを指定します。
    - 入力文書の Amazon S3 の場所に対する読み取りアクセス権を与えます。
    - 出力文書の Amazon S3 の場所に対する書き込みアクセス権を与えます。
    - `comprehend.amazonaws.com` サービスプリンシパルがロールを引き受け、アクセス許可を取得することを許可する信頼ポリシーを含む。

    このような権限と適切な信頼ポリシーを持つ IAM ロールがまだない場合は、[IAM ロールの作成] を選択して作成してください。
11. フォームの入力が完了したなら、[ジョブの作成] を選択してトピック検出ジョブを作成し、開始します。

新しいジョブがジョブリストに表示され、ステータスフィールドにはジョブのステータスが表示されます。このフィールドは、`IN_PROGRESS` 処理中のジョブ、`COMPLETED` 正常に終了したジョブ、`FAILED` エラーのあるジョブのいずれでもかまいません。ジョブをクリックすると、エラーメッセージを含むそのジョブに関する詳細情報を取得できます。

ジョブが完了すると、Amazon Comprehend は、そのジョブに対して指定した出力用 Amazon S3 の場所に、分析結果を保存します。各インサイトタイプの分析結果の説明については、「インサイト」を参照してください。

Amazon Comprehend API を使用する

Amazon Comprehend API では、リアルタイム (同期) 分析を実行するオペレーションと、非同期分析ジョブを開始および管理するオペレーションを実行することができます。

Amazon Comprehend API オペレータは直接使用ことも、あるいは CLI または SDK を使用して間接的に使用することもできます。この章の例では、CLI、Python SDK、および Java SDK を使用しています。

AWS CLI と Python の例を実行するには、 をインストールする必要があります AWS CLI。詳細については、「AWS Command Line Interface （ AWS CLI) のセットアップ」を参照してください。

Java の例を実行するには、 AWS SDK for Javaをインストールする必要があります。SDK for Java をインストールする手順については、AWS 「 SDK for Java のセットアップ」を参照してください。

トピック

- AWS SDK での Amazon Comprehend の使用
- API を使用したリアルタイムの分析
- API を使用した非同期分析ジョブ

AWS SDK での Amazon Comprehend の使用

AWS Software Development Kit (SDKsは、多くの一般的なプログラミング言語で使用できます。各 SDK には、デベロッパーが好みの言語でアプリケーションを簡単に構築できるようにする API、コード例、およびドキュメントが提供されています。

SDK ドキュメント	コード例
AWS SDK for C++	AWS SDK for C++ コード例
AWS CLI	AWS CLI コード例
AWS SDK for Go	AWS SDK for Go コード例
AWS SDK for Java	AWS SDK for Java コード例
AWS SDK for JavaScript	AWS SDK for JavaScript コード例

SDK ドキュメント	コード例
AWS SDK for Kotlin	AWS SDK for Kotlin コード例
AWS SDK for .NET	AWS SDK for .NET コード例
AWS SDK for PHP	AWS SDK for PHP コード例
AWS Tools for PowerShell	PowerShell コード例のツール
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) コード例
AWS SDK for Ruby	AWS SDK for Ruby コード例
AWS SDK for Rust	AWS SDK for Rust コード例
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP コード例
AWS SDK for Swift	AWS SDK for Swift コード例

> **可用性の例**
>
> 必要なものが見つからなかった場合。このページの下側にある [Provide feedback (フィードバックを送信)] リンクから、コードの例をリクエストしてください。

API を使用したリアルタイムの分析

以下の例では、 AWS CLI と、AWS SDKにNET、Java、おおびPythonを使用して、リアルタイム分析にAmazon Comprehend APIをどのように使用するかを示しています。これらの例を使用して、Amazon Comprehend 非同期オペレーションについて、またご自身のアプリケーションの構成要素として説明します。

このセクションの .NET 例では AWS SDK for .NET を使用します。AWS Toolkit for Visual Studio を使用し、.NET を使用して AWS アプリケーションを開発できます。便利なテンプレートと AWS Explorer を使用して、アプリケーションをデプロイし、サービスを管理できます。AWS の .NET デベロッパーの視点の場合は、.NET デベロッパー 向けの AWS ガイドを参照してください。

トピック

- 主要言語の検出
- 名前付きエンティティを検出する
- キーフレーズの検出
- 感情の判断
- ターゲット感情のリアルタイム分析
- 構文の検出
- リアルタイムバッチ API

主要言語の検出

テキストで使用される主要言語を特定するには、 DetectDominantLanguageオペレーションを使用します。バッチ内の最大 25 のドキュメントの主要言語を検出するには、BatchDetectDominantLanguageオペレーションを使用します。詳細については、「リアルタイムバッチ API」を参照してください。

トピック

- AWS Command Line Interface の使用
- AWS SDK for Java、SDK for Python、または AWS SDK for .NET の使用

AWS Command Line Interface の使用

次の例は、AWS CLI で `DetectDominantLanguage` オペレーションを使用する方法を示しています。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend detect-dominant-language \
    --region region \
    --text "It is raining today in Seattle."
```

Amazon Comprehend は次のように応答します。

```
{
    "Languages": [
        {
```

```
            "LanguageCode": "en",
            "Score": 0.9793661236763
        }
    ]
}
```

AWS SDK for Java、SDK for Python、または AWS SDK for .NET の使用

主要言語を判別する方法を示す SDK の例については、「AWS SDK または CLI DetectDominantLanguage で を使用する」を参照してください。

名前付きエンティティを検出する

ドキュメント内の名前付きエンティティを決定するには、 DetectEntitiesオペレーションを使用します。バッチ内の最大 25 個のドキュメント内のエンティティを検出するには、 BatchDetectEntitiesオペレーションを使用します。詳細については、「リアルタイムバッチ API」を参照してください。

トピック

- AWS Command Line Interface の使用
- AWS SDK for Java、SDK for Python、または AWS SDK for .NET の使用

AWS Command Line Interface の使用

次の例は、AWS CLI で DetectEntities オペレーションを使用する方法を示しています。入力テキストの言語を指定する必要があります。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend detect-entities \
    --region region \
    --language-code "en" \
    --text "It is raining today in Seattle."
```

Amazon Comprehend は次のように応答します。

```
{
    "Entities": [
        {
```

```
            "Text": "today",
            "Score": 0.97,
            "Type": "DATE",
            "BeginOffset": 14,
            "EndOffset": 19
        },
        {
            "Text": "Seattle",
            "Score": 0.95,
            "Type": "LOCATION",
            "BeginOffset": 23,
            "EndOffset": 30
        }
    ],
    "LanguageCode": "en"
}
```

AWS SDK for Java、SDK for Python、または AWS SDK for .NET の使用

主要言語を判別する方法を示す SDK の例については、「AWS SDK または CLI DetectEntitiesで を使用する」を参照してください。

キーフレーズの検出

テキストで使用される名詞キーフレーズを特定するには、 DetectKeyPhrasesオペレーション を使用します。バッチ内の最大 25 個のドキュメントでキー名詞フレーズを検出するには、 BatchDetectKeyPhrasesオペレーションを使用します。詳細については、「リアルタイムバッチ API」を参照してください。

トピック

- AWS Command Line Interface の使用
- AWS SDK for Java、SDK for Python、または AWS SDK for .NET の使用

AWS Command Line Interface の使用

次の例は、AWS CLI で DetectKeyPhrases オペレーションを使用する方法を示しています。入力テキストの言語を指定する必要があります。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend detect-key-phrases \
    --region region \
    --language-code "en" \
    --text "It is raining today in Seattle."
```

Amazon Comprehend は次のように応答します。

```
{
    "LanguageCode": "en",
    "KeyPhrases": [
        {
            "Text": "today",
            "Score": 0.89,
            "BeginOffset": 14,
            "EndOffset": 19
        },
        {
            "Text": "Seattle",
            "Score": 0.91,
            "BeginOffset": 23,
            "EndOffset": 30
        }
    ]
}
```

AWS SDK for Java、SDK for Python、または AWS SDK for .NET の使用

キーフレーズを検出する SDK の例については、「AWS SDK または CLI DetectKeyPhrases で を使用する」を参照してください。

感情の判断

Amazon Comprehend には、感情を分析するための次の API オペレーションが用意されています。

- DetectSentiment — ドキュメントの全体的な感情を決定します。
- BatchDetectSentiment — バッチ内の最大 25 件のドキュメントの全体的な感情を判断します。詳細については、「リアルタイムバッチ API」を参照してください。
- StartSentimentDetectionJob – ドキュメントのコレクションに対して非同期感情検出ジョブを開始します。
- ListSentimentDetectionJobs – 送信した感情検出ジョブのリストを返します。

- DescribeSentimentDetectionJob – 指定された感情検出ジョブに関連付けられたプロパティ (ステータスを含む) を取得します。
- StopSentimentDetectionJob – 指定された進行中の感情ジョブを停止します。

トピック

- AWS Command Line Interfaceの使用
- AWS SDK for Java、SDK for Python、または AWS SDK for .NET の使用

AWS Command Line Interfaceの使用

次の例は、AWS CLI で DetectSentiment オペレーションを使用する方法を示しています。この例では、入力テキストの言語を指定します。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend detect-sentiment \
    --region region \
    --language-code "en" \
    --text "It is raining today in Seattle."
```

Amazon Comprehend は次のように応答します。

```
{
    "SentimentScore": {
        "Mixed": 0.014585512690246105,
        "Positive": 0.31592071056365967,
        "Neutral": 0.5985543131828308,
        "Negative": 0.07093945890665054
    },
    "Sentiment": "NEUTRAL",
    "LanguageCode": "en"
}
```

AWS SDK for Java、SDK for Python、または AWS SDK for .NET の使用

入力テキストの感情を決定する SDK の例については、「AWS SDK または CLI DetectSentimentで を使用する」を参照してください。

ターゲット感情のリアルタイム分析

Amazon Comprehend には、ターゲット感情のリアルタイム分析のための次の API オペレーションが用意されています。

- DetectTargetedSentiment – ドキュメントに記載されているエンティティの感情を分析します。
- BatchDetectTargetedSentiment – バッチ内の最大 25 件のドキュメントについて、ターゲット感情を分析します。詳細については、「リアルタイムバッチ API」を参照してください。

分析しているテキストにターゲット感情「エンティティタイプ」が含まれていない場合、API は空のエンティティ配列を返します。

AWS Command Line Interfaceの使用

次の例は、AWS CLI で DetectTargetedSentiment オペレーションを使用する方法を示しています。この例では、入力テキストの言語を指定します。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend detect-targeted-sentiment \
    --region region \
    --language-code "en" \
    --text "The burger was cooked perfectly but it was cold. The service was OK."
```

Amazon Comprehend は次のように応答します。

```
{
"Entities": [
    {
      "DescriptiveMentionIndex": [
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 4,
          "EndOffset": 10,
          "Score": 1,
          "GroupScore": 1,
          "Text": "burger",
          "Type": "OTHER",
```

```
          "MentionSentiment": {
            "Sentiment": "POSITIVE",
            "SentimentScore": {
              "Mixed": 0.001515,
              "Negative": 0.000822,
              "Neutral": 0.000243,
              "Positive": 0.99742
            }
          }
        },
        {
          "BeginOffset": 36,
          "EndOffset": 38,
          "Score": 0.999843,
          "GroupScore": 0.999661,
          "Text": "it",
          "Type": "OTHER",
          "MentionSentiment": {
            "Sentiment": "NEGATIVE",
            "SentimentScore": {
              "Mixed": 0,
              "Negative": 0.999996,
              "Neutral": 0.000004,
              "Positive": 0
            }
          }
        }
      ]
    },
    {
      "DescriptiveMentionIndex": [
        0
      ],
      "Mentions": [
        {
          "BeginOffset": 53,
          "EndOffset": 60,
          "Score": 1,
          "GroupScore": 1,
          "Text": "service",
          "Type": "ATTRIBUTE",
          "MentionSentiment": {
            "Sentiment": "NEUTRAL",
            "SentimentScore": {
```

```
              "Mixed": 0.000033,
              "Negative": 0.000089,
              "Neutral": 0.993325,
              "Positive": 0.006553
            }
          }
        }
      ]
    }
  ]
}
```

構文の検出

テキストを解析して個々の単語を抽出し、各単語の音声部分を決定するには、 DetectSyntaxオペレーションを使用します。バッチ内の最大 25 個のドキュメントの構文を解析するには、BatchDetectSyntaxオペレーションを使用します。詳細については、「リアルタイムバッチ API」を参照してください。

トピック

- AWS Command Line Interface の使用
- AWS SDK for Java、SDK for Python、または AWS SDK for .NET の使用

AWS Command Line Interface の使用

次の例は、AWS CLI で DetectSyntax オペレーションを使用する方法を示しています。この例では、入力テキストの言語を指定します。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend detect-syntax \
   --region region \
   --language-code "en" \
   --text "It is raining today in Seattle."
```

Amazon Comprehend は次のように応答します。

```
{
    "SyntaxTokens": [
```

```
        {
            "Text": "It",
            "EndOffset": 2,
            "BeginOffset": 0,
            "PartOfSpeech": {
                "Tag": "PRON",
                "Score": 0.8389829397201538
            },
            "TokenId": 1
        },
        {
            "Text": "is",
            "EndOffset": 5,
            "BeginOffset": 3,
            "PartOfSpeech": {
                "Tag": "AUX",
                "Score": 0.9189288020133972
            },
            "TokenId": 2
        },
        {
            "Text": "raining",
            "EndOffset": 13,
            "BeginOffset": 6,
            "PartOfSpeech": {
                "Tag": "VERB",
                "Score": 0.9977611303329468
            },
            "TokenId": 3
        },
        {
            "Text": "today",
            "EndOffset": 19,
            "BeginOffset": 14,
            "PartOfSpeech": {
                "Tag": "NOUN",
                "Score": 0.9993606209754944
            },
            "TokenId": 4
        },
        {
            "Text": "in",
            "EndOffset": 22,
            "BeginOffset": 20,
```

```
            "PartOfSpeech": {
                "Tag": "ADP",
                "Score": 0.9999061822891235
            },
            "TokenId": 5
        },
        {
            "Text": "Seattle",
            "EndOffset": 30,
            "BeginOffset": 23,
            "PartOfSpeech": {
                "Tag": "PROPN",
                "Score": 0.9940338730812073
            },
            "TokenId": 6
        },
        {
            "Text": ".",
            "EndOffset": 31,
            "BeginOffset": 30,
            "PartOfSpeech": {
                "Tag": "PUNCT",
                "Score": 0.9999997615814209
            },
            "TokenId": 7
        }
    ]
}
```

AWS SDK for Java、SDK for Python、または AWS SDK for .NET の使用

入力テキストの構文を検出する SDK の例については、「AWS SDK または CLI DetectSyntaxで を使用する」 を参照してください。

リアルタイムバッチ API

Amazon Comprehend のリアルタイムバッチオペレーションを使用すると、最大 25 件のドキュメントのバッチを送信できます。バッチオペレーションを呼び出すことは、リクエスト内のドキュメントごとに単一ドキュメント API を呼び出すことと同じです。バッチ API を使用すると、アプリケーションのパフォーマンスが向上します。詳細については、「複数ドキュメントの同期処理」を参照してください。

トピック

- AWS CLI によるバッチ処理
- AWS SDK for .NET によるバッチ処理

AWS CLI によるバッチ処理

以下の例では、AWS Command Line Interface でバッチ API オペレーションを使用する方法を示します。BatchDetectDominantLanguage 以外のすべての操作では、 process.json という名前の次のJSONファイルがインプットとして使用されます。その操作に LanguageCode エンティティは含まれません。

JSON ファイル ("$$$$$$$$") の 3 番目のドキュメントは、バッチ処理中に エラーの原因となります。これは、オペレーションがレスポンスBatchItemErrorに を含めるようにするために含まれています。

```
{
   "LanguageCode": "en",
   "TextList": [
      "I have been living in Seattle for almost 4 years",
      "It is raining today in Seattle",
      "$$$$$$$$"
   ]
}
```

例は、Unix、Linux、macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

トピック

- バッチ (AWS CLI) を使用して主要言語を検出します。
- バッチ (AWS CLI) を使用してエンティティを検出します。
- バッチ (AWS CLI) を使用してキーフレーズを検出します。
- バッチ (AWS CLI) を使用して感情を検出します。

バッチ (AWS CLI) を使用して主要言語を検出します。

BatchDetectDominantLanguage オペレーションは、バッチ内の各ドキュメントの主要な言語を決定します。Amazon Comprehend が検出できる言語のリストについては、「主要言語」を参照してく

ださい。以下の AWS CLI コマンドは、BatchDetectDominantLanguage オペレーションを呼び出します。

```
aws comprehend batch-detect-dominant-language \
    --endpoint endpoint \
    --region region \
    --cli-input-json file://path to input file/process.json
```

BatchDetectDominantLanguage オペレーションからのレスポンスは次のとおりです。

```
{
    "ResultList": [
        {
          "Index": 0,
          "Languages":[
            {
              "LanguageCode":"en",
              "Score": 0.99
            }
          ]
        },
        {
          "Index": 1
          "Languages":[
            {
              "LanguageCode":"en",
              "Score": 0.82
            }
          ]
        }
    ],
    "ErrorList": [
      {
        "Index": 2,
        "ErrorCode": "InternalServerException",
        "ErrorMessage": "Unexpected Server Error. Please try again."
      }
    ]
}
```

バッチ (AWS CLI) を使用してエンティティを検出します。

BatchDetectEntities 演算を使用して、ドキュメントのバッチに存在するエンティティを検索します。エンティティの詳細については、「エンティティ」を参照してください。以下の AWS CLI コマンドは、BatchDetectEntities オペレーションを呼び出します。

```
aws comprehend batch-detect-entities \
    --endpoint endpoint \
    --region region \
    --cli-input-json file://path to input file/process.json
```

バッチ (AWS CLI) を使用してキーフレーズを検出します。

BatchDetectKeyPhrases オペレーションは、キー名詞フレーズをドキュメントのバッチで返します。以下の AWS CLI コマンドは、BatchDetectKeyNounPhrases オペレーションを呼び出します。

```
aws comprehend batch-detect-key-phrases
    --endpoint endpoint
    --region region
    --cli-input-json file://path to input file/process.json
```

バッチ (AWS CLI) を使用して感情を検出します。

BatchDetectSentiment オペレーションを使用して、ドキュメントのバッチの全体的な感情を検出します。以下の AWS CLI コマンドは、BatchDetectSentiment オペレーションを呼び出します。

```
aws comprehend batch-detect-sentiment \
    --endpoint endpoint \
    --region region \
    --cli-input-json file://path to input file/process.json
```

AWS SDK for .NET によるバッチ処理

次のサンプルプログラムは、 で BatchDetectEntitiesオペレーションを使用する方法を示していますAWS SDK for .NET。サーバーからのレスポンスには、正常に処理された各ドキュメントの BatchDetectEntitiesItemResult オブジェクトが含まれています。ドキュメントの処理中にエラーが発生した場合は、レスポンスのエラーリストにレコードが記録されます。この例では、エラーのある各ドキュメントを取得して再送信します。

このセクションの .NET 例では AWS SDK for .NET を使用します。AWS Toolkit for Visual Studio を使用し、.NET を使用して AWS アプリケーションを開発できます。便利なテンプレートと AWS Explorer を使用して、アプリケーションをデプロイし、サービスを管理できます。AWS の .NET デベロッパーの視点の場合は、.NET デベロッパー 向けの AWS ガイドを参照してください。

```
using System;
using System.Collections.Generic;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

namespace Comprehend
{
    class Program
    {
        // Helper method for printing properties
        static private void PrintEntity(Entity entity)
        {
            Console.WriteLine("     Text: {0}, Type: {1}, Score: {2}, BeginOffset: {3}
 EndOffset: {4}",
                entity.Text, entity.Type, entity.Score, entity.BeginOffset,
 entity.EndOffset);
        }

        static void Main(string[] args)
        {
            AmazonComprehendClient comprehendClient = new
 AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            List<String> textList = new List<String>()
            {
                { "I love Seattle" },
                { "Today is Sunday" },
                { "Tomorrow is Monday" },
                { "I love Seattle" }
            };

            // Call detectEntities API
            Console.WriteLine("Calling BatchDetectEntities");
            BatchDetectEntitiesRequest batchDetectEntitiesRequest = new
 BatchDetectEntitiesRequest()
            {
                TextList = textList,
                LanguageCode = "en"
```

```
            };
            BatchDetectEntitiesResponse batchDetectEntitiesResponse =
 comprehendClient.BatchDetectEntities(batchDetectEntitiesRequest);

            foreach (BatchDetectEntitiesItemResult item in
 batchDetectEntitiesResponse.ResultList)
            {
                Console.WriteLine("Entities in {0}:", textList[item.Index]);
                foreach (Entity entity in item.Entities)
                    PrintEntity(entity);
            }

            // check if we need to retry failed requests
            if (batchDetectEntitiesResponse.ErrorList.Count != 0)
            {
                Console.WriteLine("Retrying Failed Requests");
                List<String> textToRetry = new List<String>();
                foreach(BatchItemError errorItem in
 batchDetectEntitiesResponse.ErrorList)
                    textToRetry.Add(textList[errorItem.Index]);

                batchDetectEntitiesRequest = new BatchDetectEntitiesRequest()
                {
                    TextList = textToRetry,
                    LanguageCode = "en"
                };

                batchDetectEntitiesResponse =
 comprehendClient.BatchDetectEntities(batchDetectEntitiesRequest);

                foreach(BatchDetectEntitiesItemResult item in
 batchDetectEntitiesResponse.ResultList)
                {
                    Console.WriteLine("Entities in {0}:", textList[item.Index]);
                    foreach (Entity entity in item.Entities)
                        PrintEntity(entity);
                }
            }
            Console.WriteLine("End of DetectEntities");
        }
    }
}
```

API を使用した非同期分析ジョブ

以下の例では、AWS CLI から Amazon Comprehend 非同期 API を使用して、分析ジョブを作成および管理しています。

トピック

- Amazon Comprehend のインサイトのための非同期分析
- ターゲット感情の非同期分析
- イベント検出の同期分析
- トピックモデリングの非同期分析

Amazon Comprehend のインサイトのための非同期分析

以下のセクションでは、Amazon Comprehend API を使用して非同期オペレーションを実行し、Amazon Comprehend のインサイトを分析します。

トピック

- 前提条件
- 分析の開始
- 分析ジョブのモニタリング
- 分析結果の取得

前提条件

ドキュメントは、UTF-8 形式のテキストファイルである必要があります。ドキュメントは 2 つの形式で送信できます。次の表に示すように、使用する形式は分析する文書のタイプによって異なります。

説明	形式
各ファイルには 1 つの入力ドキュメントを含めます。これはサイズの大きいドキュメントのコレクションに最適です。	ファイルごとに 1 文書
入力は 1 つまたは複数のファイルです。ファイル内の各行は 1 つのドキュメントとみなされ	1 行に 1 文書

説明	形式
ます。これは、ソーシャルメディアへの投稿など、短いドキュメントに最適です。 各行は、改行 (LF、\ n)、キャリッジリターン (CR、\r)、またはその両方 (CRLF、\r\n) で終える必要があります。行の終了に、UTF-8 の行区切り文字 (u+2028) を使用することはできません。	

分析ジョブを開始するときは、入力データの S3 上の場所を指定します。URI は、呼び出す API エンドポイントと同じ AWS リージョンである必要があります。URI は 1 つのファイルを指すものでも、データファイルのコレクションのプレフィックスでもかまいません。詳細については、「InputDataConfig」データ型を参照してください。

Amazon Comprehend には、ドキュメントコレクションと出力ファイルを含む Amazon S3 バケットへのアクセス許可を付与する必要があります。詳細については、「バッチ操作に必要なロールベースのアクセス許可」を参照してください。

分析の開始

分析ジョブを送信するには、Amazon Comprehend コンソールまたは適切な `Start*` オペレーションを使用します。

- StartDominantLanguageDetectionJob — コレクション内の各ドキュメントの主要な言語を検出するジョブを開始します。ドキュメントの主要言語の詳細については、「主要言語」を参照してください。
- StartEntitiesDetectionJob — コレクション内の各ドキュメント内のエンティティを検出するジョブを開始します。エンティティの詳細については、「エンティティ」を参照してください。
- StartKeyPhrasesDetectionJob — コレクション内の各ドキュメント内のキーフレーズを検出するジョブを開始します。キーフレーズの詳細については、「キーフレーズ」を参照してください。
- StartPiiEntitiesDetectionJob — コレクション内の各ドキュメント内の個人を特定できる情報 (PII) を検出するジョブを開始します。PII の詳細については、「PII エンティティの検出」を参照してください。
- StartSentimentDetectionJob — コレクション内の各ドキュメント内の感情を検出するジョブを開始します。詳細については、「感情」を参照してください。

分析ジョブのモニタリング

Start* オペレーションでは、ジョブの進行状況の監視に利用できる ID が返されます。

この API を使用して進行状況を監視するには、個別ジョブの進行状況を監視するのか、複数のジョブの進行状況を監視するのかに応じて、2 つあるオペレーションのいずれかを使用します。

個別分析ジョブの進行状況を監視するには、Describe* オペレーションを使用します。Start* オペレーションによって返されるジョブ ID を指定します。Describe* オペレーションからのレスポンスには、ジョブのステータスを示す JobStatus フィールドが含まれます。

複数の分析ジョブの進行状況を監視するには、List* オペレーションを使用します。 List* オペレーションでは、Amazon Comprehend に送信したジョブのリストが返されます。レスポンスには、ジョブごとにそのステータスを示す JobStatus フィールドが含まれます。

ステータスフィールドが COMPLETED または FAILED に設定されている場合、そのジョブの処理は完了しています。

個別ジョブのステータスを取得するには、実行する分析で Describe* オペレーションを使用します。

- DescribeDominantLanguageDetectionJob
- DescribeEntitiesDetectionJob
- DescribeKeyPhrasesDetectionJob
- DescribePiiEntitiesDetectionJob
- DescribeSentimentDetectionJob

複数のジョブのステータスを取得するには、実行する分析で List* オペレーションを使用します。

- ListDominantLanguageDetectionJobs
- ListEntitiesDetectionJobs
- ListKeyPhrasesDetectionJobs
- ListPiiEntitiesDetectionJobs
- ListSentimentDetectionJobs

結果を特定の条件に一致するジョブに制限するには、List* オペレーションの Filter パラメータを使用します。フィルタ条件には、ジョブ名やジョブステータス、またはジョブの送信日時を使用

できます。詳細については、Amazon Comprehend API リファレンスの各 List* オペレーションの Filter パラメータを参照してください。

分析結果の取得

分析ジョブが終了すると、Describe* オペレーションを使用して結果の場所を取得します。ジョブのステータスが COMPLETED の場合、応答には出力ファイルの Amazon S3 の場所を示すフィールドを含む OutputDataConfig フィールドが含まれます。このファイル output.tar.gz は、分析の結果を含む圧縮アーカイブ形式です。

ジョブのステータスが FAILED の場合、 レスポンスには、ジョブが正常に完了しなかった理由を説明する Message フィールドが含まれます。

個別ジョブのステータスを取得するには、適切な Describe* オペレーションを使用します。

- DescribeDominantLanguageDetectionJob
- DescribeEntitiesDetectionJob
- DescribeKeyPhrasesDetectionJob
- DescribeSentimentDetectionJob

結果はドキュメントごとに 1 つの JSON 構造を持つ 1 つのファイルの形式で返されます。各レスポンスファイルには、ステータスフィールドが FAILED に設定されているジョブのエラーメッセージも含まれます。

以下の各セクションでは、2 つの入力形式の出力例を示しています。

主要言語の検出結果の取得

次に、主要言語を検出した分析からの出力ファイルの例を示します。入力の形式は、行ごとにドキュメント 1 つです。詳細については、DetectDominantLanguage オペレーションを参照してください。

```
{"File": "0_doc", "Languages": [{"LanguageCode": "en", "Score": 0.9514502286911011},
 {"LanguageCode": "de", "Score": 0.02374090999364853}, {"LanguageCode": "nl", "Score":
 0.003208699868991971}, "Line": 0}
{"File": "1_doc", "Languages": [{"LanguageCode": "en", "Score": 0.982271254062652 6},
 {"LanguageCode": "de", "Score": 0.00262139202095568 2}, {"LanguageCode": "es", "Score":
 0.002386554144322872}], "Line": 1}
```

以下は、入力の形式がファイルごとにドキュメント 1 つの場合の分析の出力例です。

```
{"File": "small_doc", "Languages": [{"LanguageCode": "en", "Score":
 0.9728053212165833}, {"LanguageCode": "de", "Score": 0.007670710328966379},
 {"LanguageCode": "es", "Score": 0.0028472368139773607}]}
{"File": "huge_doc", "Languages": [{"LanguageCode": "en", "Score": 0.984955906867981},
 {"LanguageCode": "de", "Score": 0.0026436643674969673}, {"LanguageCode": "fr",
 "Score": 0.0014206881169229746}]}
```

エンティティ検出結果の取得

以下は、ドキュメント内のエンティティを検出した分析からの出力ファイル例です。入力の形式は、行ごとにドキュメント 1 つです。詳細については、DetectEntities オペレーションを参照してください。出力には 2 つのエラーメッセージが含まれています。1 つはドキュメントが長すぎる、もう 1 つはドキュメントが UTF-8 形式ではないというエラーです。

```
{"File": "50_docs", "Line": 0, "Entities": [{"BeginOffset": 0, "EndOffset": 22,
 "Score": 0.9763959646224976, "Text": "Cluj-NapocaCluj-Napoca", "Type": "LOCATION"}"]}
{"File": "50_docs", "Line": 1, "Entities": [{"BeginOffset": 11, "EndOffset": 15,
 "Score": 0.9615424871444702, "Text": "Maat", "Type": "PERSON"}}]}
{"File": "50_docs", "Line": 2, "ErrorCode": "DOCUMENT_SIZE_EXCEEDED", "ErrorMessage":
 "Document size exceeds maximum size limit 102400 bytes."}
{"File": "50_docs", "Line": 3, "ErrorCode": "UNSUPPORTED_ENCODING", "ErrorMessage":
 "Document is not in UTF-8 format and all subsequent lines are ignored."}
```

以下は、入力の形式がファイルごとにドキュメント 1 つの場合の分析の出力例です。出力には 2 つのエラーメッセージが含まれています。1 つはドキュメントが長すぎる、もう 1 つはドキュメントが UTF-8 形式ではないというエラーです。

```
{"File": "non_utf8.txt", "ErrorCode": "UNSUPPORTED_ENCODING", "ErrorMessage": "Document
 is not in UTF-8 format and all subsequent line are ignored."}
{"File": "small_doc", "Entities": [{"BeginOffset": 0, "EndOffset": 4, "Score":
 0.645766019821167, "Text": "Maat", "Type": "PERSON"}]}
{"File": "huge_doc", "ErrorCode": "DOCUMENT_SIZE_EXCEEDED", "ErrorMessage": "Document
 size exceeds size limit 102400 bytes."}
```

キーフレーズ検出結果の取得

以下は、ドキュメント内のエンティティを検出した分析からの出力ファイル例です。入力の形式は、行ごとにドキュメント 1 つです。詳細については、DetectKeyPhrases オペレーションを参照してください。

```
{"File": "50_docs", "KeyPhrases": [{"BeginOffset": 0, "EndOffset": 22, "Score":
 0.8948641419410706, "Text": "Cluj-NapocaCluj-Napoca"}, {"BeginOffset": 45,
 "EndOffset": 49, "Score": 0.9989854693412781, "Text": "Cluj"}], "Line": 0}
```

以下は、入力の形式がファイルごとにドキュメント 1 つの場合の分析の出力例です。

```
{"File": "1_doc", "KeyPhrases": [{"BeginOffset": 0, "EndOffset": 22, "Score":
 0.8948641419410706, "Text": "Cluj-NapocaCluj-Napoca"}, {"BeginOffset": 45,
 "EndOffset": 49, "Score": 0.9989854693412781, "Text": "Cluj"}]}
```

個人を特定できる情報 (PII) 検出結果の取得

以下は、ドキュメント内の PII エンティティを検出した分析ジョブからの出力ファイル例です。入力の形式は、行ごとにドキュメント 1 つです。

```
{"Entities":[{"Type":"NAME","BeginOffset":40,"EndOffset":69,"Score":0.999995},
{"Type":"ADDRESS","BeginOffset":247,"EndOffset":253,"Score":0.998828},
{"Type":"BANK_ACCOUNT_NUMBER","BeginOffset":406,"EndOffset":411,"Score":0.693283}],"File":"doc.
{"Entities":[{"Type":"SSN","BeginOffset":1114,"EndOffset":1124,"Score":0.999999},
{"Type":"EMAIL","BeginOffset":3742,"EndOffset":3775,"Score":0.999993},
{"Type":"PIN","BeginOffset":4098,"EndOffset":4102,"Score":0.999995}],"File":"doc.txt","Line":1}
```

以下は、入力の形式がファイルごとにドキュメント 1 つの場合の分析の出力例です。

```
{"Entities":[{"Type":"NAME","BeginOffset":40,"EndOffset":69,"Score":0.999995},
{"Type":"ADDRESS","BeginOffset":247,"EndOffset":253,"Score":0.998828},
{"Type":"BANK_ROUTING","BeginOffset":279,"EndOffset":289,"Score":0.999999}],"File":"doc.txt"}
```

センチメント検出結果の取得

以下は、ドキュメントで表されているセンチメントを検出した分析からの出力ファイル例です。1 つのドキュメントが長すぎるというエラーメッセージが含まれています。入力の形式は、行ごとにドキュメント 1 つです。詳細については、DetectSentiment オペレーションを参照してください。

```
{"File": "50_docs", "Line": 0, "Sentiment": "NEUTRAL", "SentimentScore": {"Mixed":
 0.002734508365392685, "Negative": 0.008935936726629734, "Neutral": 0.9841893315315247,
 "Positive": 0.004140198230743408}}
{"File": "50_docs", "Line": 1, "ErrorCode": "DOCUMENT_SIZE_EXCEEDED", "ErrorMessage":
 "Document size is exceeded maximum size limit 5120 bytes."}
```

```
{"File": "50_docs", "Line": 2, "Sentiment": "NEUTRAL", "SentimentScore":
 {"Mixed": 0.0023119584657251835, "Negative": 0.0029857370536774397, "Neutral":
 0.9866572022438049, "Positive": 0.00804515462368726 7}}
```

以下は、入力の形式がファイルごとにドキュメント 1 つの場合の分析の出力例です。

```
{"File": "small_doc", "Sentiment": "NEUTRAL", "SentimentScore": {"Mixed":
 0.0023450672160834074, "Negative": 0.0009663937962614, "Neutral": 0.9795311689376831,
 "Positive": 0.017157377675175667}}
{"File": "huge_doc", "ErrorCode": "DOCUMENT_SIZE_EXCEEDED", "ErrorMessage": "Document
 size is exceeds the limit of 5120 bytes."}
```

ターゲット感情の非同期分析

ターゲット感情のリアルタイム分析の詳細については、「the section called “ターゲット感情のリアルタイム分析”」を参照してください。

Amazon Comprehend には、非同期のターゲット感情分析を開始および管理するための次の API オペレーションがあります。

- StartTargetedSentimentDetectionJob – ドキュメントのコレクションに対して、非同期ターゲット感情検出ジョブを開始します。
- ListTargetedSentimentDetectionJobs – 送信したターゲット感情検出ジョブのリストを返します。
- DescribeTargetedSentimentDetectionJob – 指定されたターゲット感情検出ジョブに関連付けられたプロパティ (ステータスを含む) を取得します。
- StopTargetedSentimentDetectionJob – 指定された進行中のターゲット感情ジョブを停止します。

トピック

- 開始する前に
- AWS CLI を使用したターゲット感情の分析

開始する前に

このチュートリアルを開始する前に、以下の要件を満たしていることを確認してください。

- 入出力バケット — 入出力に使用する Amazon S3 バケットを指定します。バケットは、呼び出す API と同じリージョンに存在している必要があります。

- IAM サービスロール — 入出力バケットにアクセス許可を持つ IAM サービスロールが必要です。詳細については、「バッチ操作に必要なロールベースのアクセス許可」を参照してください。

AWS CLI を使用したターゲット感情の分析

以下は、AWS CLI での StartTargetedSentimentDetectionJob オペレーションの使用例です。この例では、入力テキストの言語を指定します。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend start-targeted-sentiment-detection-job \
       --job-name "job name" \
       --language-code "en" \
       --cli-input-json file://path to JSON input file
```

次の例に示すように、cli-input-json パラメータには、リクエストデータを含む JSON ファイルへのパスを指定します。

```
{
    "InputDataConfig": {
        "S3Uri": "s3://input bucket/input path",
        "InputFormat": "ONE_DOC_PER_FILE"
    },
    "OutputDataConfig": {
        "S3Uri": "s3://output bucket/output path"
    },
    "DataAccessRoleArn": "arn:aws:iam::account ID:role/data access role"
}
```

ジョブの開始リクエストが成功すると、以下のようなレスポンスが返されます。

```
{
    "JobStatus": "SUBMITTED",
    "JobArn": "job ARN"
    "JobId": "job ID"
}
```

イベント検出の同期分析

トピック

- 開始する前に
- AWS CLI を使用してイベントを検出する
- AWS CLI を使用してイベントを一連表示する
- AWS CLI を使用してイベントを説明する
- イベント検出結果を取得する

ドキュメントセット内のイベントを検出するには、 StartEventsDetectionJob を使用して非同期ジョブを開始します。

開始する前に

このチュートリアルを開始する前に、以下の要件を満たしていることを確認してください。

- 入出力バケット — 入出力に使用する Amazon S3 バケットを指定します。バケットは、呼び出す API と同じリージョンに存在している必要があります。
- IAM サービスロール — 入出力バケットにアクセス許可を持つ IAM サービスロールが必要です。詳細については、「バッチ操作に必要なロールベースのアクセス許可」を参照してください。

AWS CLI を使用してイベントを検出する

以下は、AWS CLI での StartEventsDetectionJob オペレーションの使用例です。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend start-events-detection-job \
  --region region \
  --job-name job name \
  --cli-input-json file://path to JSON input file
```

次の例に示すように、cli-input-json パラメータには、リクエストデータを含む JSON ファイルへのパスを指定します。

```
{
  "InputDataConfig": {
```

```
        "S3Uri": "s3://input bucket/input path",
        "InputFormat": "ONE_DOC_PER_LINE"
    },
    "OutputDataConfig": {
        "S3Uri": "s3://output bucket/output path"
    },
    "DataAccessRoleArn": "arn:aws:iam::account ID:role/data access role"
    "LanguageCode": "en",
    "TargetEventTypes": [
        "BANKRUPTCY",
        "EMPLOYMENT",
        "CORPORATE_ACQUISITION",
        "INVESTMENT_GENERAL",
        "CORPORATE_MERGER",
        "IPO",
        "RIGHTS_ISSUE",
        "SECONDARY_OFFERING",
        "SHELF_OFFERING",
        "TENDER_OFFERING",
        "STOCK_SPLIT"
    ]
}
```

イベント検出ジョブの開始リクエストが成功すると、以下のようなレスポンスが返されます。

```
{
    "JobStatus": "SUBMITTED",
    "JobId": "job ID"
}
```

AWS CLI を使用してイベントを一連表示する

ListEventsDetectionJobs オペレーションを使用して、送信したイベント検出ジョブのリストを表示します。このリストには、使用された入力および出力の場所、各検出ジョブのステータスに関する情報が含まれます 例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend list-events-detection-jobs --region region
```

レスポンスには、次のような JSON が返されます。

```
{
```

```
 "EventsDetectionJobPropertiesList": [
    {
       "DataAccessRoleArn": "arn:aws:iam::account ID:role/data access role",
       "EndTime": timestamp,
       "InputDataConfig": {
          "InputFormat": "ONE_DOC_PER_LINE",
          "S3Uri": "s3://input bucket/input path"
       },
       "JobId": "job ID",
       "JobName": "job name",
       "JobStatus": "COMPLETED",
       "LanguageCode": "en",
       "Message": "message",
       "OutputDataConfig": {
          "S3Uri": "s3://output bucket/ouput path"
       },
       "SubmitTime": timestamp,
       "TargetEventTypes": [
         "BANKRUPTCY",
         "EMPLOYMENT",
         "CORPORATE_ACQUISITION",
         "INVESTMENT_GENERAL",
         "CORPORATE_MERGER",
         "IPO",
         "RIGHTS_ISSUE",
         "SECONDARY_OFFERING",
         "SHELF_OFFERING",
         "TENDER_OFFERING",
         "STOCK_SPLIT"
  ]
    }
 ],
 "NextToken": "next token"
}
```

AWS CLI を使用してイベントを説明する

DescribeEventsDetectionJob オペレーションを使用して、既存のジョブのステータスを取得できます。例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend describe-events-detection-job \
  --region region \
```

```
  --job-id job ID
```

レスポンスには、次のような JSON が返されます。

```
{
 "EventsDetectionJobProperties": {
    "DataAccessRoleArn": "arn:aws:iam::account ID:role/data access role",
    "EndTime": timestamp,
    "InputDataConfig": {
       "InputFormat": "ONE_DOC_PER_LINE",
       "S3Uri": "S3Uri": "s3://input bucket/input path"
    },
    "JobId": "job ID",
    "JobName": "job name",
    "JobStatus": "job status",
    "LanguageCode": "en",
    "Message": "message",
    "OutputDataConfig": {
       "S3Uri": "s3://output bucket/output path"
    },
    "SubmitTime": timestamp,
    "TargetEventTypes": [
      "BANKRUPTCY",
      "EMPLOYMENT",
      "CORPORATE_ACQUISITION",
      "INVESTMENT_GENERAL",
      "CORPORATE_MERGER",
      "IPO",
      "RIGHTS_ISSUE",
      "SECONDARY_OFFERING",
      "SHELF_OFFERING",
      "TENDER_OFFERING",
      "STOCK_SPLIT"
  ]
 }
}
```

イベント検出結果を取得する

以下は、ドキュメントでイベントを検出した分析ジョブの出力ファイルの例です。入力の形式は、行ごとにドキュメント 1 つです。

```
{"Entities": [{"Mentions": [{"BeginOffset": 12, "EndOffset": 27, "GroupScore":
 1.0, "Score": 0.916355, "Text": "over a year ago", "Type": "DATE"}]}, {"Mentions":
 [{"BeginOffset": 33, "EndOffset": 39, "GroupScore": 1.0, "Score": 0.996603,
 "Text": "Amazon", "Type": "ORGANIZATION"}]}, {"Mentions": [{"BeginOffset":
 66, "EndOffset": 77, "GroupScore": 1.0, "Score": 0.999283, "Text": "Whole
 Foods", "Type": "ORGANIZATION"}]}], "Events": [{"Arguments": [{"EntityIndex":
 2, "Role": "INVESTEE", "Score": 0.999283}, {"EntityIndex": 0, "Role": "DATE",
 "Score": 0.916355}, {"EntityIndex": 1, "Role": "INVESTOR", "Score": 0.996603}],
 "Triggers": [{"BeginOffset": 373, "EndOffset": 380, "GroupScore": 0.999984,
 "Score": 0.999955, "Text": "acquire", "Type": "CORPORATE_ACQUISITION"}], "Type":
 "CORPORATE_ACQUISITION"}, {"Arguments": [{"EntityIndex": 2, "Role": "PARTICIPANT",
 "Score": 0.999283}], "Triggers": [{"BeginOffset": 115, "EndOffset": 123, "GroupScore":
 1.0, "Score": 0.999967, "Text": "combined", "Type": "CORPORATE_MERGER"}], "Type":
 "CORPORATE_MERGER"}], "File": "doc.txt", "Line": 0}
```


イベント出力ファイルの構造およびサポートされているイベントタイプの詳細については、「イベント」を参照してください。

トピックモデリングの非同期分析

ドキュメントセット内のトピックを確認するには、 StartTopicsDetectionJob を使用して非同期ジョブを開始します。英語またはスペイン語のドキュメントのトピックを監視できます。

トピック

- 開始する前に
- AWS Command Line Interface の使用
- SDK for Python または AWS SDK for .NET を使用する場合

開始する前に

このチュートリアルを開始する前に、以下の要件を満たしていることを確認してください。

- 入出力バケット — 入出力に使用する Amazon S3 バケットを指定します。バケットは、呼び出す API と同じリージョンに存在している必要があります。
- IAM サービスロール — 入出力バケットにアクセス許可を持つ IAM サービスロールが必要です。詳細については、「バッチ操作に必要なロールベースのアクセス許可」を参照してください。

AWS Command Line Interface の使用

以下は、AWS CLI での StartTopicsDetectionJob オペレーションの使用例です。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend start-topics-detection-job \
                --number-of-topics topics to return \
                --job-name "job name" \
                --region region \
                --cli-input-json file://path to JSON input file
```

次の例に示すように、cli-input-json パラメータには、リクエストデータを含む JSON ファイルへのパスを指定します。

```
{
    "InputDataConfig": {
        "S3Uri": "s3://input bucket/input path",
        "InputFormat": "ONE_DOC_PER_FILE"
    },
    "OutputDataConfig": {
        "S3Uri": "s3://output bucket/output path"
    },
    "DataAccessRoleArn": "arn:aws:iam::account ID:role/data access role"
}
```

トピック検出ジョブの開始リクエストが成功すると、以下のようなレスポンスが返されます。

```
{
    "JobStatus": "SUBMITTED",
    "JobId": "job ID"
}
```

ListTopicsDetectionJobs オペレーションを使用して、送信したトピック検出ジョブのリストを表示します。このリストには、使用された入力および出力の場所、各検出ジョブのステータスに関する情報が含まれます 例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend list-topics-detection-jobs \-- region
```

レスポンスには、次のような JSON が返されます。

```
{
    "TopicsDetectionJobPropertiesList": [
        {
            "InputDataConfig": {
                "S3Uri": "s3://input bucket/input path",
                "InputFormat": "ONE_DOC_PER_LINE"
            },
            "NumberOfTopics": topics to return,
            "JobId": "job ID",
            "JobStatus": "COMPLETED",
            "JobName": "job name",
            "SubmitTime": timestamp,
            "OutputDataConfig": {
                "S3Uri": "s3://output bucket/output path"
            },
            "EndTime": timestamp
        },
        {
            "InputDataConfig": {
                "S3Uri": "s3://input bucket/input path",
                "InputFormat": "ONE_DOC_PER_LINE"
            },
            "NumberOfTopics": topics to return,
            "JobId": "job ID",
            "JobStatus": "RUNNING",
            "JobName": "job name",
            "SubmitTime": timestamp,
            "OutputDataConfig": {
                "S3Uri": "s3://output bucket/output path"
            }
        }
    ]
}
```

DescribeTopicsDetectionJob オペレーションを使用して、既存のジョブのステータスを取得できます。例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend describe-topics-detection-job --job-id job ID
```

レスポンスには、次のような JSON が返されます。

```
{
    "TopicsDetectionJobProperties": {
        "InputDataConfig": {
            "S3Uri": "s3://input bucket/input path",
            "InputFormat": "ONE_DOC_PER_LINE"
        },
        "NumberOfTopics": topics to return,
        "JobId": "job ID",
        "JobStatus": "COMPLETED",
        "JobName": "job name",
        "SubmitTime": timestamp,
        "OutputDataConfig": {
            "S3Uri": "s3://output bucket/ouput path"
        },
        "EndTime": timestamp
    }
}
```

SDK for Python または AWS SDK for .NET を使用する場合

トピックモデリングジョブの開始方法を示す SDK の例については、「AWS SDK または CLI StartTopicsDetectionJobで を使用する」を参照してください。

信頼と安全性

ユーザーは、オンラインアプリケーション (peer-to-peer チャットやフォーラムのディスカッションなど) 、ウェブサイトに投稿されたコメント、および生成 AI アプリケーション (生成 AI モデルからの入力プロンプトと出力) を通じて大量のテキストコンテンツを生成します。Amazon Comprehend Trust and Safety 機能によって、このコンテンツをモデレートし、ユーザーに安全で包括的な環境を提供することができます。

Amazon Comprehend Trust and Safety 機能を使用するメリットは次のとおりです。

- モデレーションの高速化: 大量のテキストを迅速かつ正確にモデレートして、オンラインプラットフォームに不適切なコンテンツが含まれないようにします。
- カスタマイズ可能: API レスポンスのモデレーションしきい値をアプリケーションのニーズに合わせてカスタマイズできます。
- 使いやすい: LangChain 統合、または AWS CLI または SDKsを使用して、信頼と安全性の機能を設定します。

Amazon Comprehend Trust and Safety は、コンテンツモデレーションの以下の側面に対応しています。

- Toxicity detection — 有害、攻撃的、または不適切な可能性のあるコンテンツを検出します。例としては、ヘイトスピーチ、脅迫、虐待などがあります。
- Intent classification — 明示的または暗示的な悪意のある意図を持つコンテンツを検出します。例としては、差別的または違法なコンテンツ、医療、法律、政治、物議を醸す、個人的、または金銭的な問題について助言を表明または要求するコンテンツが含まれます。
- Privacy protection — ユーザーは、個人を特定できる情報 (PII) を明らかにする可能性のあるコンテンツを誤って提供する可能性があります。Amazon Comprehend PII では、個人識別情報を検出して編集することができます。

トピック

- 有害性検出
- 迅速な安全性分類
- PII の検出と削除

有害性検出

Amazon Comprehend 毒性検出では、テキストベースのインタラクションに含まれる有害性コンテンツをリアルタイムで検出できます。組織検出を使用して、オンラインプラットフォームでの peer-to-peer 会話をモデレートしたり、生成 AI の入力と出力をモニタリングしたりできます。

毒性検出では、以下のカテゴリの攻撃的なコンテンツを検出します。

GRAPHIC (どぎつい)

グラフィックスピーチは、視覚的に説明的で詳細、不快かつ鮮明な画像を使用します。このような言葉は、受信者への侮辱、不快感、危害を増幅・冗長されることがよくあります。

HARASSMENT_OR_ABUSE (ハラスメントまたは虐待)

意図に関わらず、話し手と聞き手の間に破壊的な権力の動態を押し付け、受け手のメンタルヘルスに影響を与えようとしたり、人をモノ化しようとしたりする言説。

HATE_SPEECH (ヘイトスピーチ)

人種、民族、性同一性、宗教、性的指向、能力、出身国、その他のアイデンティティグループなど、アイデンティティに基づいて個人またはグループを批判、侮辱、非人間化する言葉。

INSULT (侮辱)

侮辱的、屈辱的、嘲笑的、侮辱的、または軽蔑的な言葉を含む発言。

PROFANITY (不敬)

無礼な、下品な、または攻撃的な言葉、フレーズ、または頭字語を含むスピーチは、不敬と見なされます。

SEXUAL (性的)

体の一部、身体的特徴、性別への直接的または間接的な言及により、性的関心、活動、性的興奮を示す発言。

VIOLENCE_OR_THREAT (暴力または脅威)

個人または集団に対して苦痛や痛み、敵意を与えることを意図する脅迫的な発言。

TOXICITY (有害性)

上記のカテゴリのいずれかに当てはまり、本質的に有害と見なされる可能性のある単語、フレーズ、または頭字語を含む言葉。

API を使用した有害コンテンツの検出

テキスト内の有害なコンテンツを検出するには、 同期DetectToxicContentオペレーションを使用します。このオペレーションは、入力として提供されたテキスト文字列のリストを分析します。API レスポンスには、入力リストのサイズと一致する結果リストが含まれます。

現在、有害コンテンツ検出は英語のみをサポートしています。入力テキストには、最大 10 個のテキスト文字列のリストを指定できます。各文字列は、最大 1 KB のサイズまで可能です。

有害成分検出では、入力文字列ごとに 1 つのエントリを含む分析結果のリストが返されます。エントリには、テキスト文字列で特定された有害コンテンツタイプのリストと、各コンテンツタイプの信頼性スコアが含まれます。　エントリには文字列の有害性スコアも含まれています。

次の例では、DetectToxicContent および Python を使用した AWS CLI オペレーションの使用方法を示します。

AWS CLI

有害性物質は、 AWS CLIの以下のコマンドを使用して検出できます。

```
aws comprehend detect-toxic-content --language-code en  /
            --text-segments "[{\"Text\":\"You are so obtuse\"}]"
```

は次の結果で AWS CLI 応答します。このテキストセグメントは、INSULT カテゴリ内では高い信頼度スコアを獲得し、その結果、有害性スコアも高くなっています。

```
{
   "ResultList": [
      {
         "Labels": [
                {
                    "Name": "PROFANITY",
                    "Score": 0.0006000000284984708
                },
                {
                    "Name": "HATE_SPEECH",
                    "Score": 0.00930000003427267
                },
                {
                    "Name": "INSULT",
                    "Score": 0.9204999804496765
```

```
                },
                {
                    "Name": "GRAPHIC",
                    "Score": 9.999999747378752e-05
                },
                {
                    "Name": "HARASSMENT_OR_ABUSE",
                    "Score": 0.0052999998442828655
                },
                {
                    "Name": "SEXUAL",
                    "Score": 0.01549999974668026
                },
                {
                    "Name": "VIOLENCE_OR_THREAT",
                    "Score": 0.007799999788403511
                }
            ],
            "Toxicity": 0.7192999720573425
      }
   ]
}
```

text-segments パラメータには次の形式を使用し、最大 10 個のテキスト文字列を入力できます。

```
   --text-segments "[{\"Text\":\"text string 1\"},
                    {\"Text\":\"text string2\"},
                    {\"Text\":\"text string3\"}]"
```

は次の結果で AWS CLI 応答します。

```
{
   "ResultList": [
      {
         "Labels": [ (truncated) ],
            "Toxicity": 0.3192999720573425
      },
      {
         "Labels": [ (truncated) ],
            "Toxicity": 0.1192999720573425
```

```
      },
      {
         "Labels": [ (truncated) ],
            "Toxicity": 0.0192999720573425
      }
   ]
}
```

Python (Boto)

以下の例は、Python を使用して有害コンテンツを検出する方法を示しています。

```
import boto3
client = boto3.client(
    service_name='comprehend',
    region_name=region) # For example, 'us-west-2'

response = client.detect_toxic_content(
    LanguageCode='en',
    TextSegments=[{'Text': 'You are so obtuse'}]
)
print("Response: %s\n" % response)
```

迅速な安全性分類

Amazon Comprehend には、大規模言語モデル (LLM) やその他の生成系 AI モデルのプレーンテキスト入力プロンプトを分類するための、事前トレーニング済みのバイナリ分類子が用意されています。

プロンプト安全分類子は入力プロンプトを分析し、プロンプトが安全かどうかの信頼スコアを割り当てます。

安全でないプロンプトとは、個人情報や個人情報の要求、攻撃的または違法なコンテンツを生成する、医療、法律、政治、金融の主題に関するアドバイスを要求するなど、悪意のある意図を表す入力プロンプトです。

API を使用した迅速な安全性分類

テキスト文字列のプロンプト安全分類を実行するには、同期ClassifyDocumentオペレーションを使用します。入力には、英語のプレーンテキスト文字列を指定します。文字列の最大サイズは 10 KB です。

レスポンスには 2 つのクラス (SAFE と UNSAFE) と、各クラスの信頼度スコアが含まれます。スコアの値範囲は 0 ~ 1 で、1 が最も高い信頼度です。

次の例は、 AWS CLI および Python でプロンプト安全性分類を使用する方法を示しています。

AWS CLI

次の例は、 AWS CLIでプロンプト安全性分類子を使用する方法を示しています。

```
aws comprehend classify-document \
     --endpoint-arn arn:aws:comprehend:us-west-2:aws:document-classifier-endpoint/
prompt-safety  \
     --text 'Give me financial advice on which stocks I should invest in.'
```

は次の出力で AWS CLI 応答します。

```
{
    "Classes": [
        {
            "Score": 0.6312999725341797,
            "Name": "UNSAFE_PROMPT"
        },
        {
            "Score": 0.3686999976634979,
            "Name": "SAFE_PROMPT"
        }
    ]
}
```

> **Note**
>
> `classify-document` コマンドを使用する場合、 `--endpoint-arn`パラメータには、 AWS CLI 設定 AWS リージョン と同じ を使用する ARN を渡す必要があります。を設定するには AWS CLI、 `aws configure` コマンドを実行します。この例のエンドポイント ARN にはリージョンコード `us-west-2` があります。プロンプト安全性分類子は、以下のどのリージョンでも使用できます。
>
> - us-east-1
> - us-west-2
> - eu-west-1

- ap-southeast-2

Python (Boto)

次の例は、Python でプロンプト安全性分類子を使用する方法を示しています。

```
import boto3
client = boto3.client(service_name='comprehend', region_name='us-west-2')

response = client.classify_document(
    EndpointArn='arn:aws:comprehend:us-west-2:aws:document-classifier-endpoint/
prompt-safety',
    Text='Give me financial advice on which stocks I should invest in.'
)
print("Response: %s\n" % response)
```

> **Note**
>
> `classify_document` メソッドを使用する場合、`EndpointArn` 引数には boto3 SDK クライアントと同じ AWS リージョン を使用する ARN を渡す必要があります。この例では、クライアントとエンドポイント ARN はどちらも `us-west-2` を使用します。プロンプト安全性分類子は、以下のどのリージョンでも使用できます。
>
> - us-east-1
> - us-west-2
> - eu-west-1
> - ap-southeast-2

PII の検出と削除

Amazon Comprehend コンソールまたは APIs、英語またはスペイン語のテキストドキュメントで個人を特定できる情報 (PII) を検出できます。PII は、個人を特定できる個人データをテキストで参照したものです。PII の例には、住所、銀行口座番号、電話番号などがあります。

テキスト内の PII エンティティを検出または削除できます。PII エンティティを検出するには、リアルタイム分析または非同期バッチジョブを使用できます。PII エンティティを編集するには、非同期バッチジョブを使用する必要があります。

詳細については、「個人を特定できる情報 (PII)」を参照してください。

個人を特定できる情報 (PII)

Amazon Comprehend コンソールまたは APIs、英語またはスペイン語のテキストドキュメントで個人を特定できる情報 (PII) を検出できます。PII は、個人を特定するために使用できる個人データをテキストで参照したものです。PII の例には、住所、銀行口座番号、電話番号などがあります。

PII 検出では、PII エンティティを特定するか、テキスト内の PII エンティティを編集するかを選択できます。PII エンティティを見つけるには、リアルタイム分析または非同期バッチジョブを使用できます。PII エンティティを編集するには、非同期バッチジョブを使用する必要があります。

個人を特定できる情報 (PII) 用の Amazon S3 Object Lambda アクセスポイントを使用して、Amazon S3 バケットからの文書取得を制御できます。PII を含む文書へのアクセスを制御し、文書から個人を特定できる情報を削除できます。詳細については、「個人を特定できる情報 (PII) のための Amazon S3 Object Lambda アクセスポイントの使用」を参照してください。

トピック

- PII エンティティの検出
- PII エンティティのラベル付け
- PII リアルタイム分析 (コンソール)
- PII 非同期分析ジョブ (コンソール)
- PII リアルタイム分析 (API)
- PII 非同期分析ジョブ (API)

PII エンティティの検出

Amazon Comprehend を使用して、英語またはスペイン語のテキストドキュメントで PII エンティティを検出できます。PII エンティティは、特定の種類の個人を特定できる情報 (PII) です。PII 検出機能を使用して PII エンティティを検索したり、テキスト内の PII エンティティを編集したりします。

トピック

- PII エンティティを検索します。
- PII エンティティの編集
- PII ユニバーサルエンティティタイプ
- 国固有の PII エンティティタイプ

PII エンティティを検索します。

テキスト内の PII エンティティを見つけるには、リアルタイム分析を使用して 1 つの文書をすばやく分析できます。また、複数の文書に対して非同期バッチジョブを開始することもできます。

コンソールまたは API を使用して、1 つの文書をリアルタイムで分析できます。入力テキストには、UTF-8 エンコード文字で 100 バイトまで含めることができます。

たとえば、以下の入力テキストを送信して PII エンティティを検索できます。

パウロ・サントス様 クレジットカード口座1111-0000-1111-0000の最新の明細書を 123 Any Street, Seattle, WA 98109 宛に郵送いたしました。

出力には「Paul Santos」がタイプNAME、「1111-0000-1111-0000」がタイプCREDIT_DEBIT_NUMBER、「123 Any Street, Seattle, WA 98109」がタイプADDRESSという情報が含まれています。

Amazon Comprehend は、検出された PII エンティティのリストと、各 PII エンティティについて以下の情報を返します。

- 検出されたテキストスパンが検出されたエンティティタイプである確率を推定するスコア。
- PII エンティティタイプ 。
- 文書内の PII エンティティの位置。エンティティの開始と終了の文字オフセットとして指定します。

例えば、前述の入力テキストでは次のような応答が返されます。

```
{
    "Entities": [
        {
            "Score": 0.9999669790267944,
            "Type": "NAME",
            "BeginOffset": 6,
            "EndOffset": 18
        },
        {
            "Score": 0.8905550241470337,
            "Type": "CREDIT_DEBIT_NUMBER",
            "BeginOffset": 69,
            "EndOffset": 88
        },
```

```
        {
            "Score": 0.9999889731407166,
            "Type": "ADDRESS",
            "BeginOffset": 103,
            "EndOffset": 138
        }
    ]
}
```

PII エンティティの編集

テキスト内の PII エンティティを編集するには、コンソールまたは API を使用して非同期バッチジョブを開始します。Amazon Comprehend は、各 PII エンティティの入力テキストのコピーに編集を加えたものを返します。

たとえば、以下の入力テキストを送信して PII エンティティを編集できます。

パウロ・サントス様 クレジットカード口座1111-0000-1111-0000の最新の明細書を 123 Any Street, Seattle, WA 98109 宛に郵送いたしました。

出力には以下のテキストが含まれています。

***** ******様 クレジットカード口座**************** が *** ******* ******* 宛に郵送されました。

PII ユニバーサルエンティティタイプ

メールアドレスやクレジットカード番号など、一部の PII エンティティタイプは汎用です (個々の国に固有ではない) 。Amazon Comprehend は、以下のタイプのユニバーサル PII エンティティを検出します。

ADDRESS

「100 Main Street, Anytown, USA」や「Suite #12, Building 123」などの住所。住所には、通り、建物、場所、市、州、国、郡、郵便番号、地区、近隣などの情報を含めることができます。

AGE

個人の年齢 (時間の数値や単位を含む)。たとえば、「私は40歳です」というフレーズでは、Amazon Comprehendは「40歳」を年齢として認識します。

AWS_ACCESS_KEY

シークレットアクセスキーに関連付けられている一意の識別子。アクセスキー ID とシークレットアクセスキーを使用して、プログラムによる AWS リクエストに暗号で署名します。

AWS_SECRET_KEY

アクセスキーに関連付けられた一意の識別子。アクセスキー ID とシークレットアクセスキーを使用して、プログラムによる AWS リクエストに暗号で署名します。

CREDIT_DEBIT_CVV

VISA に存在する 3 桁のカード検証コード (CVV) MasterCard、Discover クレジットカードとデビットカード。アメリカンエキスプレスのクレジットカードやデビットカードの場合、CVV は 4 桁の数字コードです。

CREDIT_DEBIT_EXPIRY

クレジットカードまたはデビットカードの有効期限日 この数字は通常 4 桁で、多くの場合、月/年または MM/YY という形式になっています。Amazon Comprehend では、01/21、01/2021、Jan 2021などの有効期限を認識します。

CREDIT_DEBIT_NUMBER

クレジットカードまたはデビットカードの番号。これらの番号は 13 桁から 16 桁までさまざまです。ただし、Amazon Comprehend は、最後の 4 桁しかない場合でもクレジットカード番号またはデビットカード番号を認識します。

DATE_TIME

日付には、年、月、日、曜日、または時刻を含めることができます。たとえば、Amazon Comprehend は「January 19, 2020」や「11 am」を日付として認識します。Amazon Comprehend は、日付の一部、日付範囲、日付間隔を認識します。また「the 1990s (1990 年代) 」などの 10 年間も認識されます。

DRIVER_ID

運転免許証に割り当てられる番号。運転免許証は、個人が公道で1台または複数の自動車を運転することを許可する公式文書です。運転免許証番号は英数字です。

EMAIL

marymajor@email.com などの電子メールアドレス。

INTERNATIONAL_BANK_ACCOUNT_NUMBER

国際銀行口座番号の形式は国によって異なります。www.iban.com/structure を参照してください。

IP_ADDRESS

198.51.100.0 などの IPv4 アドレス。

LICENSE_PLATE

車両のナンバープレートは、車両が登録されている州または国によって発行されます。乗用車の形式は通常 5 ~ 8 桁で、大文字と数字で構成されます。形式は発行国または国の所在地によって異なります。

MAC_ADDRESS

メディアアクセスコントロール (MAC) アドレスは、ネットワークインターフェースコントローラー (NIC) に割り当てられる固有の識別子です。

NAME

個人の氏名。このエンティティタイプには、Dr.、Mr.、Miss などの敬称は含まれません。Amazon Comprehend は、組織または住所の一部である名前にはこのエンティティタイプを適用しません。たとえば、Amazon Comprehend は「John Doe Organization」を組織として認識し、「Jane Doe Street」を住所として認識します。

PASSWORD

「*very20special#pass*」のように、パスワードとして使用される英数字の文字列。

PHONE

電話番号 このエンティティタイプには、ファックス番号とポケットベル番号も含まれます。

PIN

銀行口座にアクセスするための 4 桁の個人識別番号 (PIN)。

SWIFT_CODE

SWIFT コードは、特定の銀行または支店を指定するために使用する銀行識別コード (BIC) の標準形式です。銀行は、これらのコードを国際電信送金などの送金に使用します。

SWIFT コードは 8 文字または 11 文字で構成されています。11 桁のコードは特定の支店を指し、8 桁のコード (または「XXX」で終わる 11 桁のコード) は本社または本店を表します。

URL

www.example.com などのウェブアドレス。

USERNAME

ログイン名、スクリーンネーム、ニックネーム、ハンドル名など、アカウントを識別するユーザー名。

VEHICLE_IDENTIFICATION_NUMBER

車両識別番号 (VIN) は、車両を一意に識別します。VIN の内容と形式は ISO 3779 仕様で定義されています。VIN のコードと形式は国ごとに異なります。

国固有の PII エンティティタイプ

パスポート番号や政府発行のその他のID番号など、一部のPIIエンティティタイプは国固有のものです。Amazon Comprehend は、以下の国固有の PII エンティティタイプを検出します。

CA_HEALTH_NUMBER

カナダの医療保健番号で、個人が医療給付を受けるために必要な 10 桁の固有識別番号です。

CA_SOCIAL_INSURANCE_NUMBER

カナダの社会保険番号 (SIN) は 9 桁の固有の識別子で、個人が政府のプログラムや特典を利用する際に必要です。

SIN の形式は、「123-456-789」のように 3 桁の 3 グループになっています。SIN は Luhn アルゴリズムと呼ばれる単純な数字チェックプロセスによって検証できます。

IN_AADHAAR

インドのAadhaarは、インド政府がインドの居住者に発行する12桁の固有の識別番号です。Aadhaar 形式では、4 桁目と 8 桁目の後にスペースまたはハイフンが付きます。

IN_NREGA

インドの全国農村雇用保証法（NREGA）の番号は、2文字とそれに続く14桁の数字で構成されています。

IN_PERMANENT_ACCOUNT_NUMBER

インドの永久口座番号は、所得税局が発行する一意の10桁英数字です。

IN_VOTER_NUMBER

インドの有権者IDは、3文字とそれに続く7つの数字で構成されています。

UK_NATIONAL_HEALTH_SERVICE_NUMBER

英国の国民医療保健番号は、485 777 3456などの10〜17桁の番号です。現行システムの形式は、10 桁の番号を 3 桁目と 6 桁目の後にスペースが入ります。最後の桁はエラー検出チェックサムです。

17 桁の数字形式では、10 桁目と 13 桁目の後にスペースがあります。

UK_NATIONAL_INSURANCE_NUMBER

英国の国民保険番号 (NINO) により、個人は国民保険 (社会保障) の給付を受けることができます。また、英国の税制ではいくつかの目的にも使用されています。

数字は 9 桁で、2 文字で始まり、6 つの数字と 1 つの文字が続きます。NINO の形式は、2 文字の後と 2 桁、4 桁、6 桁目の後にスペースまたはダッシュを入れます。

UK_UNIQUE_TAXPAYER_REFERENCE_NUMBER

英国固有納税者番号 (UTR) は、納税者または事業者を識別する 10 桁の番号です。

BANK_ACCOUNT_NUMBER

米国の銀行口座番号。通常は 10 ~ 12 桁です。Amazon Comprehend は、最後の 4 桁しかない場合でも銀行口座番号を認識します。

BANK_ROUTING

米国の銀行口座の支店コード。通常 9 桁の長さですが、Amazon Comprehend は最後の 4 桁しかない場合も支店コードを認識します。

PASSPORT_NUMBER

米国パスポート番号。パスポート番号は 6 文字から 9 文字の英数字です。

US_INDIVIDUAL_TAX_IDENTIFICATION_NUMBER

米国の個人納税者識別番号 (ITIN) は、「9」で始まり、4 桁目に「7」または「8」が含まれる 9 桁の番号です。ITIN の形式は、3 桁目と 4 桁目の後にスペースまたはダッシュを付けます。

SSN

米国社会保障番号 (SSN) は、米国市民、永住者、および臨時就労者に発行される 9 桁の番号です。Amazon Comprehend は、最後の 4 桁しかない場合でも社会保障番号を認識します。

PII エンティティのラベル付け

PII 検出を実行すると、Amazon Comprehend は識別された PII エンティティタイプのラベルを返します。たとえば、以下の入力テキストを Amazon Comprehend に送信したとします。

パウロ・サントス様 クレジットカード口座1111-0000-1111-0000の最新の明細書を 123 Any Street, Seattle, WA 98109 宛に郵送いたしました。

出力には、PII エンティティタイプを表すラベルと、精度の信頼度スコアが含まれます。この場合「パウロ・サントス」、「1111-0000-1111-0000」、「123 Any Street, WA 98109, Seattle, WA 98109」という文書テキストからNAME、CREDIT_DEBIT_NUMBER、ADDRESS のラベルが、それぞれ PII エンティティタイプとして生成されます。サポートされるエンティティタイプについては、「PII ユニバーサルエンティティタイプ」を参照してください。

Amazon Comprehend は、ラベルごとに以下の情報を提供します。

- PII エンティティタイプのラベル名。
- 検出されたテキストが PII エンティティタイプとしてラベル付けされている確率を推定するスコア。

上記のテキスト入力例では、次の JSON 出力が得られます。

```
{
    "Labels": [
        {
            "Name": "NAME",
            "Score": 0.9149109721183777
        },
        {
            "Name": "CREDIT_DEBIT_NUMBER",
            "Score": 0.5698626637458801
        }
         {
            "Name": "ADDRESS",
            "Score": 0.9951046109199524
        }
    ]
}
```

PII リアルタイム分析 (コンソール)

コンソールを使用して、テキスト文書の PII リアルタイム検出を実行できます。最大テキストサイズは、UTF-8 文字で 100 バイトです。結果がコンソールに表示され、分析を確認できます。

組み込みモデルを使用して PII 検出、リアルタイム分析を実行します。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/comprehend/ で Amazon Comprehend コンソールを開きます。

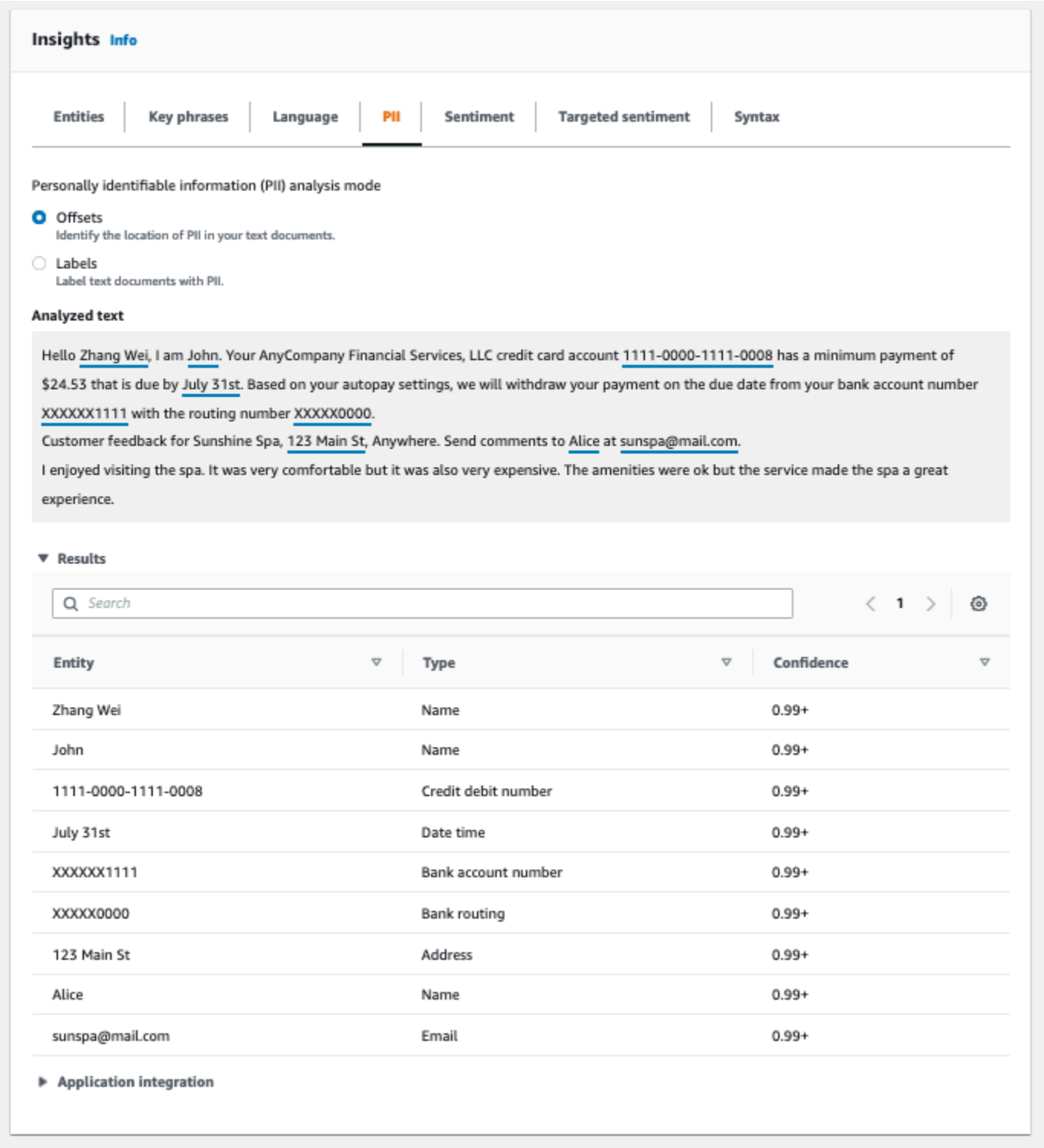
2. 左側のメニューで、[リアルタイム分析] を選択します。
3. [入力タイプ] の [分析タイプ] で [ビルトイン] を選択します。
4. 分析するテキストを入力します。
5. [分析] を選択します。コンソールのインサイトパネルにテキスト分析の結果が表示されます。PII タブには、入力テキストで検出された PII エンティティが一覧表示されます。

インサイトパネルの PII タブには、次の 2 つの分析モードの結果が表示されます。

- オフセット — テキスト文書内の PII の位置を特定します。
- ラベル — 識別された PII エンティティタイプのラベルを識別します。

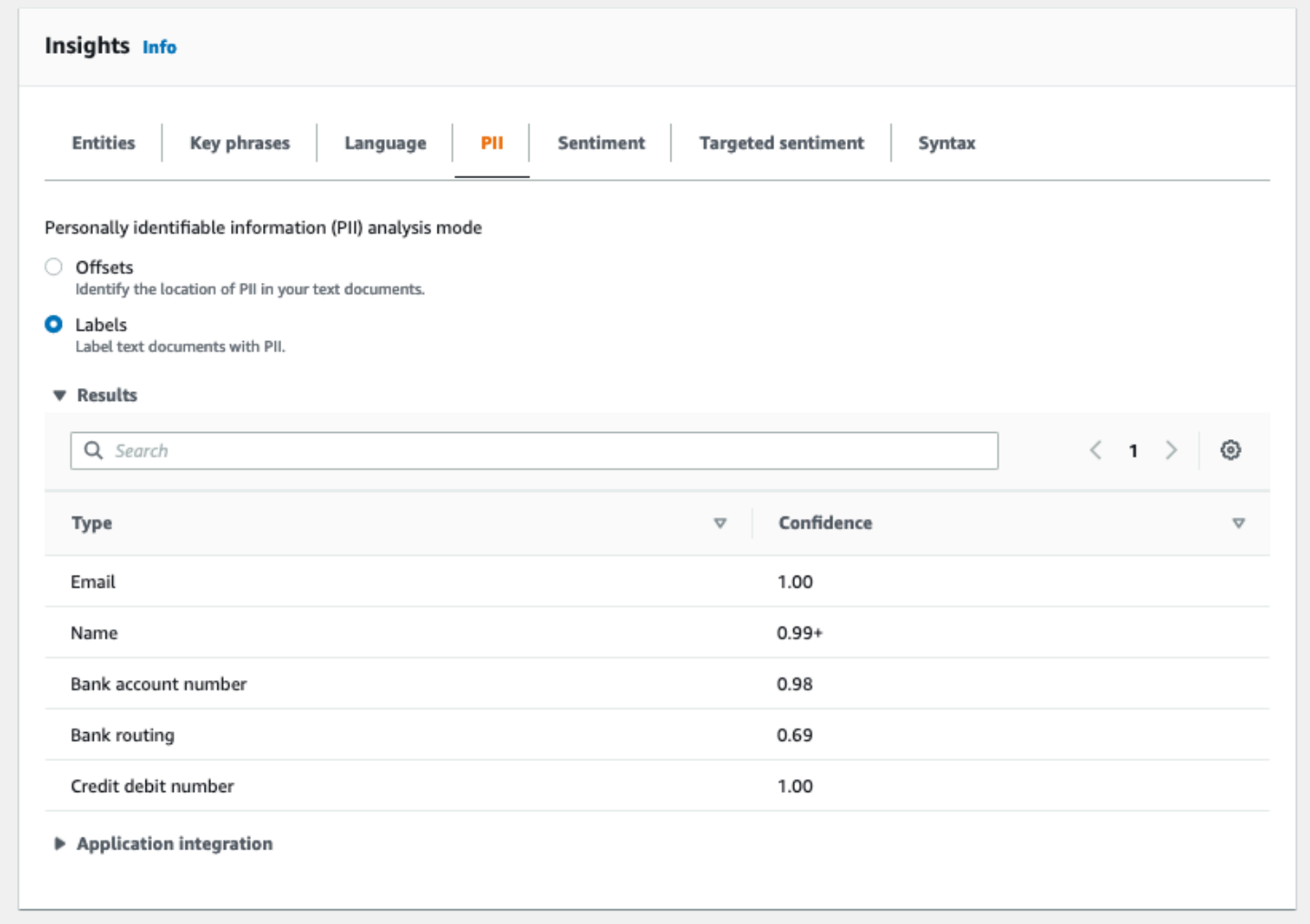
オフセット

オフセット 分析モードでは、テキストドキュメント内の PII の位置が特定されます。詳細については、「PII エンティティを検索します。」を参照してください。



ラベル

ラベル分析モードは、識別された PII エンティティタイプのラベルを返します。詳細については、「PII エンティティのラベル付け」を参照してください。



PII 非同期分析ジョブ (コンソール)

コンソールを使用して PII エンティティを検出する非同期分析ジョブを作成できます。PII エンティティタイプについては、「PII エンティティの検出」を参照してください。

分析ジョブを作成するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/comprehend/ で Amazon Comprehend コンソールを開きます。
2. 左側のメニューから、[分析ジョブ] を選択し、[ジョブの作成] を選択します。
3. [ジョブの設定] で、分析ジョブに一意の名前を付けます。
4. [分析タイプ] には、[個人を特定できる情報 (PII)] を選択します。
5. 言語 で、サポートされている言語 (英語またはスペイン語) のいずれかを選択します。
6. 「出力モード」から、次の選択肢からどれか 1 つを選択します。

- オフセット — ジョブ出力は各 PII エンティティの位置を返します。
- 編集 — ジョブ出力は、各 PII エントリが編集された入力テキストのコピーを返します。

7. (オプション) 出力モードとして「編集」を選択すると、編集する PII エンティティタイプを選択できます。
8. [入力データ] で、入力文書が Amazon S3 のどこにあるかを指定します。
   - 独自の文書を分析するには、[マイドキュメント] を選択し、[S3 を参照] を選択して、ファイルを含むバケットまたはフォルダへのパスを指定します。
   - Amazon Comprehend が提供するサンプルを分析するには、「サンプル文書」を選択します。この場合、Amazon Comprehend は によって管理されるバケットを使用し AWS、場所は指定しません。
9. (オプション) [入力フォーマット] では、入力ファイルのフォーマットのいずれかを指定します。
   - 1 ファイルに 1 文書 — 各ファイルには 1 つの入力文書が含まれます。これはサイズの大きいドキュメントのコレクションに最適です。
   - 1 行に 1 文書 — 入力は 1 つ以上のファイルです。ファイル内の各行は 1 つのドキュメントとみなされます。これは、ソーシャルメディアへの投稿など、短いドキュメントに最適です。各行は、改行 (LF、\ n)、キャリッジリターン (CR、\r)、またはその両方 (CRLF、\r\n) で終える必要があります。行の終了に、UTF-8 の行区切り文字 (u+2028) を使用することはできません。
10. [出力データ] で [S3 をブラウズ] を選択します。分析によって生成された出力データを Amazon Comprehend に書き込ませる Amazon S3 バケットまたはフォルダを選択します。
11. (オプション) ジョブの出力結果を暗号化するには、[暗号化] を選択します。次に、現在のアカウントに関連付けられた KMS キーを使用するか、別のアカウントの KMS キーを使用するかを選択します。
    - 現在のアカウントに関連付けられているキーを使用している場合は、KMS キー ID のキーエイリアスまたは ID を選択します。
    - 別のアカウントに関連付けられているキーを使用している場合は、KMS キー ID の下にキーエイリアス の ARN または ID を入力します。

> **Note**
>
> KMS キーの作成と使用や関連する暗号化の詳細については、「キー管理サービス (KMS)」を参照してください。

12. アクセス権で、次のような IAM ロールを指定します。

    - 入力文書の Amazon S3 の場所に対する読み取りアクセス権を与えます。
    - 出力文書の Amazon S3 の場所に対する書き込みアクセス権を与えます。
    - `comprehend.amazonaws.com` サービスプリンシパルがロールを引き受け、アクセス許可を取得することを許可する信頼ポリシーを含む。

    このような権限と適切な信頼ポリシーを持つ IAM ロールがまだない場合は、[IAM ロールの作成] を選択して作成してください。
13. フォームの入力が完了したなら、[ジョブの作成] を選択してトピック検出ジョブを作成し、開始します。

新しいジョブがジョブリストに表示され、ステータスフィールドにはジョブのステータスが表示されます。このフィールドは、`IN_PROGRESS` 処理中のジョブ、`COMPLETED` 正常に終了したジョブ、`FAILED` エラーのあるジョブのいずれでもかまいません。ジョブをクリックすると、エラーメッセージを含むそのジョブに関する詳細情報を取得できます。

ジョブが完了すると、Amazon Comprehend は、そのジョブに対して指定した出力用 Amazon S3 の場所に、分析結果を保存します。分析結果の説明については、「PII エンティティの検出」を参照してください。

PII リアルタイム分析 (API)

Amazon Comprehend には、文書内の個人を特定できる情報 (PII) を分析するためのリアルタイム同期 API 演算機能があります。

トピック

- PII リアルタイムエンティティ (API) の検索
- PII リアルタイムエンティティのラベル付け (API)

PII リアルタイムエンティティ (API) の検索

1 つのドキュメントで PII を見つけるには、Amazon Comprehend DetectPiiEntitiesオペレーションを使用できます。入力テキストには、UTF-8 エンコード文字で 100 バイトまで含めることができます。サポートされている言語には、英語とスペイン語が含まれます。

(CLI) を使用して PII を検索する

次の例では、 AWS CLIで `DetectPiiEntities` 演算機能を使用します。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend detect-pii-entities \
  --text "Hello Paul Santos. The latest statement for your credit card \
  account 1111-0000-1111-0000 was mailed to 123 Any Street, Seattle, WA \
   98109." \
  --language-code en
```

Amazon Comprehend は次のように応答します。

```
{
      "Entities": [
          {
              "Score": 0.9999669790267944,
              "Type": "NAME",
              "BeginOffset": 6,
              "EndOffset": 18
          },
          {
              "Score": 0.8905550241470337,
              "Type": "CREDIT_DEBIT_NUMBER",
              "BeginOffset": 69,
              "EndOffset": 88
          },
          {
              "Score": 0.9999889731407166,
              "Type": "ADDRESS",
              "BeginOffset": 103,
              "EndOffset": 138
          }
      ]
  }
```

PII リアルタイムエンティティのラベル付け (API)

リアルタイムの同期 API 演算機能を使用して、識別された PII エンティティタイプのラベルを返すことができます。詳細については、「PII エンティティのラベル付け」を参照してください。

PII エンティティのラベル付け (CLI)

次の例では、 AWS CLIで `ContainsPiiEntities` 演算機能を使用します。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend contains-pii-entities \
--text "Hello Paul Santos. The latest statement for your credit card \
account 1111-0000-1111-0000 was mailed to 123 Any Street, Seattle, WA \
 98109." \
--language-code en
```

Amazon Comprehend は次のように応答します。

```
{
    "Labels": [
        {
            "Name": "NAME",
            "Score": 0.9149109721183777
        },
        {
            "Name": "CREDIT_DEBIT_NUMBER",
            "Score": 0.8905550241470337
        }
         {
            "Name": "ADDRESS",
            "Score": 0.9951046109199524
        }
    ]
}
```

PII 非同期分析ジョブ (API)

PII 非同期分析 (API)

非同期 API 演算機能を使用して PII エンティティを検索または編集する分析ジョブを作成できます。PII エンティティタイプについては、「PII エンティティの検出」を参照してください。

トピック

- 非同期ジョブによる PII エンティティの検索 (API)

- 非同期ジョブによる PII エンティティの編集 (API)

非同期ジョブによる PII エンティティの検索 (API)

非同期バッチジョブを実行して、文書の集団から PII を検索します。ジョブを実行するには、ドキュメントを Amazon S3 にアップロードし、 StartPiiEntitiesDetectionJob リクエストを送信します。

トピック

- 開始する前に
- 入力パラメータ
- 非同期ジョブメソッド
- 出力ファイル形式
- を使用した非同期分析 AWS Command Line Interface

開始する前に

始める前に、次の有無を確認します。

- 入出力バケット — 入力ファイルと出力ファイルに使用する Amazon S3 バケットを識別します。バケットは、呼び出す API と同じリージョンに存在している必要があります。
- IAM サービスロール — 入出力バケットにアクセス許可を持つ IAM サービスロールが必要です。詳細については、「バッチ操作に必要なロールベースのアクセス許可」を参照してください。

入力パラメータ

リクエストには次のパラメータを含めます。

- `InputDataConfig` – ジョブの入力プロパティを含むリクエストInputDataConfigの定義を指定します。`S3Uri` パラメータには、入力文書の Amazon S3 の場所を指定します。
- `OutputDataConfig` – ジョブの出力プロパティを含むリクエストOutputDataConfigの定義を指定します。`S3Uri`パラメータには、Amazon Comprehend が分析結果を書き込む Amazon S3 の場所を指定します。
- `DataAccessRoleArn` – AWS Identity and Access Management ロールの Amazon リソースネーム (ARN) を指定します。このロールは、Amazon Comprehend S3 内の入力データに対する読み取りアクセス権と出力場所への書き込みアクセス権を付与します。詳細については、「バッチ操作に必要なロールベースのアクセス許可」を参照してください。

- Mode — このパラメータを ONLY_OFFSETS に設定します。この設定では、入力テキスト内の各 PII エンティティを特定する文字オフセットが出力されます。出力には信頼度スコアと PII エンティティタイプも含まれます。
- LanguageCode – このパラメータを enまたは に設定しますes。Amazon Comprehend は、英語またはスペイン語のテキストでの PII 検出をサポートしています。

非同期ジョブメソッド

StartPiiEntitiesDetectionJob はジョブ ID を返すので、ジョブの進行状況を監視し、完了時にジョブのステータスを取得できます。

分析ジョブの進行状況をモニタリングするには、 DescribePiiEntitiesDetectionJobオペレーションにジョブ ID を指定します。DescribePiiEntitiesDetectionJob からの応答には、ジョブの現在のステータスを示す JobStatus フィールドが含まれます。正常なジョブの進展は次のようになります。

[送信] -> [進行中] -> [完了]。

分析ジョブが終了したなら (JobStatus が完了、失敗、または停止)、DescribePiiEntitiesDetectionJob を使用して結果の場所を取得します。ジョブのステータスが COMPLETED の場合、応答には出力ファイルの Amazon S3 の場所を示すフィールドを含む OutputDataConfig フィールドが含まれます。

Amazon Comprehend 非同期分析の手順の詳細については、「非同期バッチ処理」を参照してください。

出力ファイル形式

出力ファイルでは、入力ファイルの名前と末尾に .out が付加されたものが使用されます。これには、分析の結果が含まれます。

以下は、ドキュメント内の PII エンティティを検出した分析ジョブからの出力ファイル例です。入力の形式は、行ごとにドキュメント 1 つです。

```
{
  "Entities": [
    {
      "Type": "NAME",
      "BeginOffset": 40,
      "EndOffset": 69,
      "Score": 0.999995
```

```
    },
    {
      "Type": "ADDRESS",
      "BeginOffset": 247,
      "EndOffset": 253,
      "Score": 0.998828
    },
    {
      "Type": "BANK_ACCOUNT_NUMBER",
      "BeginOffset": 406,
      "EndOffset": 411,
      "Score": 0.693283
    }
  ],
  "File": "doc.txt",
  "Line": 0
},
{
  "Entities": [
    {
      "Type": "SSN",
      "BeginOffset": 1114,
      "EndOffset": 1124,
      "Score": 0.999999
    },
    {
      "Type": "EMAIL",
      "BeginOffset": 3742,
      "EndOffset": 3775,
      "Score": 0.999993
    },
    {
      "Type": "PIN",
      "BeginOffset": 4098,
      "EndOffset": 4102,
      "Score": 0.999995
    }
  ],
  "File": "doc.txt",
  "Line": 1
 }
```

以下は、入力の形式がファイルごとにドキュメント 1 つの場合の分析の出力例です。

```
{
  "Entities": [
    {
      "Type": "NAME",
      "BeginOffset": 40,
      "EndOffset": 69,
      "Score": 0.999995
    },
    {
      "Type": "ADDRESS",
      "BeginOffset": 247,
      "EndOffset": 253,
      "Score": 0.998828
    },
    {
      "Type": "BANK_ROUTING",
      "BeginOffset": 279,
      "EndOffset": 289,
      "Score": 0.999999
    }
  ],
  "File": "doc.txt"
}
```

を使用した非同期分析 AWS Command Line Interface

次の例では、 AWS CLIで StartPiiEntitiesDetectionJob 演算機能を使用します。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend start-pii-entities-detection-job \
    --region region \
    --job-name job name \
    --cli-input-json file://path to JSON input file
```

次の例に示すように、cli-input-json パラメータには、リクエストデータを含む JSON ファイルへのパスを指定します。

```
{
  "InputDataConfig": {
```

```
      "S3Uri": "s3://input bucket/input path",
      "InputFormat": "ONE_DOC_PER_LINE"
  },
  "OutputDataConfig": {
      "S3Uri": "s3://output bucket/output path"
  },
  "DataAccessRoleArn": "arn:aws:iam::account ID:role/data access role"
  "LanguageCode": "en",
  "Mode": "ONLY_OFFSETS"
}
```

イベント検出ジョブの開始リクエストが成功すると、以下のような応答が表示されます。

```
{
  "JobId": "5d2fbe6e...e2c"
  "JobArn":  "arn:aws:comprehend:us-west-2:123456789012:pii-entities-detection-
job/5d2fbe6e...e2c"
  "JobStatus": "SUBMITTED",
}
```

DescribeEventsDetectionJob オペレーションを使用して、既存のジョブのステータスを取得できます。イベント検出ジョブの開始リクエストが成功すると、以下のような応答が表示されます。

```
aws comprehend describe-pii-entities-detection-job \
    --region region \
    --job-id job ID
```

ジョブが正常に終了すると、以下のような応答が表示されます。

```
{
    "PiiEntitiesDetectionJobProperties": {
  "JobId": "5d2fbe6e...e2c"
  "JobArn":  "arn:aws:comprehend:us-west-2:123456789012:pii-entities-detection-
job/5d2fbe6e...e2c"
  "JobName": "piiCLItest3",
  "JobStatus": "COMPLETED",
  "SubmitTime": "2022-05-05T14:54:06.169000-07:00",
  "EndTime": "2022-05-05T15:00:17.007000-07:00",
  "InputDataConfig": {
      (identical to the input data that you provided with the request)
    }
```

```
}
```

非同期ジョブによる PII エンティティの編集 (API)

テキスト内の PII エンティティを編集するには、非同期バッチジョブを開始します。ジョブを実行するには、ドキュメントを Amazon S3 にアップロードし、StartPiiEntitiesDetectionJobリクエストを送信します。

トピック

- 開始する前に
- 入力パラメータ
- 出力ファイル形式
- を使用した PII リダクション AWS Command Line Interface

開始する前に

始める前に、次の有無を確認します。

- 入出力バケット — 入力ファイルと出力ファイルに使用する Amazon S3 バケットを識別します。バケットは、呼び出す API と同じリージョンに存在している必要があります。
- IAM サービスロール — 入出力バケットにアクセス許可を持つ IAM サービスロールが必要です。詳細については、「バッチ操作に必要なロールベースのアクセス許可」を参照してください。

入力パラメータ

リクエストには次のパラメータを含めます。

- `InputDataConfig` – ジョブの入力プロパティを含むリクエストInputDataConfigの定義を指定します。`S3Uri` パラメータには、入力文書の Amazon S3 の場所を指定します。
- `OutputDataConfig` – ジョブの出力プロパティを含むリクエストOutputDataConfigの定義を指定します。`S3Uri`パラメータには、Amazon Comprehend が分析結果を書き込む Amazon S3 の場所を指定します。
- `DataAccessRoleArn` - AWS Identity and Access Management ロールの Amazon リソースネーム (ARN) を示します。このロールは、Amazon Comprehend S3 内の入力データに対する読み取りアクセス権と出力場所への書き込みアクセス権を付与します。詳細については、「バッチ操作に必要なロールベースのアクセス許可」を参照してください。

- Mode — このパラメータを ONLY_REDACTION に設定します。この設定では、Amazon Comprehend は入力文書のコピーを Amazon S3 の出力場所に書き込みます。このコピーでは、各 PII エンティティが編集されます。
- RedactionConfig – 秘匿化の設定パラメータを含むリクエストRedactionConfigの定義を指定します。編集する PII タイプを指定し、各 PII エンティティをその種類の名前と任意の文字に置き換えるかどうかを指定します。
  - 編集する PII エンティティタイプを PiiEntityTypes 配列に指定します。すべてのエンティティタイプを編集するには、配列の値を ["ALL"] に設定します。
  - 各 PII エンティティをそのタイプに置き換えるには、MaskModeパラメータを REPLACE_WITH_PII_ENTITY_TYPE に設定します。たとえば、この設定では、PII エンティティ「Jane Doe」が「[NAME]」に置き換えられます。
  - 各 PII エンティティの文字を任意の文字に置き換えるには、MaskMode パラメータを MASK に設定し、MaskCharacter パラメータを置換文字に設定します。1 文字のみ指定してください。有効な文字は !、#、$、%、&、*、@ です。たとえば、この設定では、PII エンティティ「Jane Doe」が「**** ***」に置き換えられます。
- LanguageCode – このパラメータを enまたは に設定しますes。Amazon Comprehend は、英語またはスペイン語のテキストでの PII 検出をサポートしています。

出力ファイル形式

次の例は、PII を編集する分析ジョブの入力ファイルと出力ファイルを示しています。入力の形式は、行ごとにドキュメント 1 つです。

```
{
Managing Your Accounts Primary Branch Canton John Doe Phone Number 443-573-4800 123
 Main StreetBaltimore, MD 21224
Online Banking HowardBank.com  Telephone 1-877-527-2703 Bank 3301 Boston Street,
 Baltimore, MD 21224
```

この入力ファイルを編集する分析ジョブは、次の出力ファイルを生成します。

```
{
Managing Your Accounts Primary Branch ****** ******** Phone Number ************
 *********************************
Online Banking **************  Telephone ************** Bank
 ************************************
```

```
 }
```

を使用した PII リダクション AWS Command Line Interface

次の例では、 AWS CLIで StartPiiEntitiesDetectionJob 演算機能を使用します。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend start-pii-entities-detection-job \
    --region region \
    --job-name job name \
    --cli-input-json file://path to JSON input file
```

次の例に示すように、cli-input-json パラメータには、リクエストデータを含む JSON ファイルへのパスを指定します。

```
{
    "InputDataConfig": {
        "S3Uri": "s3://input bucket/input path",
        "InputFormat": "ONE_DOC_PER_LINE"
    },
    "OutputDataConfig": {
        "S3Uri": "s3://output bucket/output path"
    },
    "DataAccessRoleArn": "arn:aws:iam::account ID:role/data access role"
    "LanguageCode": "en",
    "Mode": "ONLY_REDACTION"
    "RedactionConfig": {
        "MaskCharacter": "*",
        "MaskMode": "MASK",
        "PiiEntityTypes": ["ALL"]
    }
}
```

イベント検出ジョブの開始リクエストが成功すると、以下のような応答が表示されます。

```
{
  "JobId": "7c4fbe6e...e5b"
  "JobArn":  "arn:aws:comprehend:us-west-2:123456789012:pii-entities-detection-
job/7c4fbe6e...e5b"
```

```
  "JobStatus": "SUBMITTED",
}
```

DescribeEventsDetectionJob オペレーションを使用して、既存のジョブのステータスを取得できます。

```
aws comprehend describe-pii-entities-detection-job \
    --region region \
    --job-id job ID
```

ジョブが正常に終了すると、以下のような応答が表示されます。

```
{
  "PiiEntitiesDetectionJobProperties": {
     "JobId": "7c4fbe6e...e5b"
     "JobArn":  "arn:aws:comprehend:us-west-2:123456789012:pii-entities-detection-
job/7c4fbe6e...e5b"
     "JobName": "piiCLIredtest1",
     "JobStatus": "COMPLETED",
     "SubmitTime": "2022-05-05T14:54:06.169000-07:00",
     "EndTime": "2022-05-05T15:00:17.007000-07:00",
     "InputDataConfig": {
        (identical to the input data that you provided with the request)
  }
}
```

ドキュメント処理

Amazon Comprehend では、カスタム分類とカスタムエンティティ認識でワンステップドキュメント処理を行うことができます。例えば、プレーンテキストドキュメントと半構造化ドキュメント（PDF ドキュメント、Microsoft Word ドキュメント、画像など）を組み合わせてカスタム分析ジョブに入力できます。

テキスト抽出が必要な入力ファイルの場合、Amazon Comprehend は分析を実行する前にテキスト抽出を自動的に実行します。テキストコンテンツを抽出する際、Amazon Comprehend はネイティブの半構造化ドキュメントに内部パーサーを使用し、画像やスキャンされたドキュメントには Amazon Textract API を使用します。

Amazon Comprehend ドキュメント処理は、Amazon Comprehend の各 で使用できます。ただしサポートされるリージョン、アジアパシフィック (東京) および AWS GovCloud（米国西部) では、カスタム分類のプレーンテキストモデルのみがサポートされています。

以下のトピックでは、Amazon Comprehend がカスタム分析でサポートしている入力ドキュメントの種類を詳しく説明します。

トピック

- リアルタイムカスタム分析用の入力
- 非同期カスタム分析の入力
- テキスト抽出オプションの設定
- 画像のベストプラクティス

リアルタイムカスタム分析用の入力

カスタムモデルを使用したリアルタイム分析では、1 つのドキュメントを入力として扱います。次のトピックでは、使用可能な入力ドキュメントタイプについて説明します。

トピック

- プレーンテキストドキュメント
- 半構造化ドキュメント
- イメージファイルとスキャンした PDF ファイル
- Amazon Textract 出力

- リアルタイム分析用の最大ドキュメントサイズ
- 半構造化ドキュメントのエラー

プレーンテキストドキュメント

入力ドキュメントは UTF-8 形式のテキストを提供してください。

半構造化ドキュメント

半構造化ドキュメントには、ネイティブ PDF ドキュメントと Word ドキュメントが含まれます。

デフォルトでは、リアルタイムカスタム分析は Amazon Comprehend パーサーを使用して Word ファイルとデジタル PDF ファイルからテキストを抽出します。PDF ファイルの場合は、このデフォルトをオーバーライドして、Amazon Textract を使用してテキストを抽出できます。テキスト抽出オプションの設定 を参照してください。

イメージファイルとスキャンした PDF ファイル

サポートされている画像タイプには JPEG、PNG、TIFF があります。

デフォルトでは、カスタムエンティティレコグナイザーは Amazon Textract `DetectDocumentText` API オペレーションを使用して、画像ファイルとスキャンした PDF ファイルからテキストを抽出します。このデフォルトをオーバーライドして、代わりに `AnalyzeDocument` API オペレーションを使用できます。テキスト抽出オプションの設定 を参照してください。

Amazon Textract 出力

Amazon Textract `DetectDocumentText` API または `AnalyzeDocument` API からの JSON 出力を、カスタム分類とカスタムエンティティレコグナイザー用のリアルタイム API オペレーションへの入力として提供できます。Amazon Comprehend は、リアルタイム API オペレーションではこの入力タイプをサポートしていますが、コンソールではサポートしていません。

リアルタイム分析用の最大ドキュメントサイズ

すべての入力ドキュメントタイプで、入力ファイルの最大数は 1 ページで、10,000 文字以下です。

次の表は、入力ドキュメントの最大ファイルサイズを示しています。

ファイルタイプ	最大サイズ (API)	最大サイズ (コンソール)
A UTF-8 テキストドキュメント	10 KB	10 KB
PDF ドキュメント	10 MB	5 MB
Word ドキュメント	10 MB	1 MB
画像ファイル	10 MB	5 MB
Textract 出力ファイル	1 MB	該当なし

半構造化ドキュメントのエラー

ClassifyDocument または DetectEntities API オペレーションでは、半構造化ドキュメントまたは画像ファイルからテキストを抽出するときに、ドキュメントレベルまたはページレベルのエラーが発生する可能性があります。

ページレベルのエラー

入力ドキュメントのページの処理中に ClassifyDocumentまたは DetectEntities API オペレーションでエラーが発生した場合、API レスポンスには各エラーのエラーリストにエントリが含まれます。

`ErrorCode` エラーリストのエントリには、次のいずれかの値が含まれます。

- TEXTRACT_BAD_PAGE — Amazon Textract はページを読み取ることができません。Amazon Textract のページ制限の詳細については、「Amazon Textract のページクォータ」 を参照してください。
- TEXTRACT_PROVISIONED_THROUTPUT_EXCEEDED — リクエストの数がスループット制限を超えました。Amazon Textract のスループットクォータの詳細については、「Amazon Textract のデフォルトクォータ」 を参照してください。
- PAGE_CHARACTERS_EXCEEDED — ページ上のテキスト文字数が多すぎます (最大 10,000 文字)。
- PAGE_SIZE_EXCEEDED — 最大ページサイズは 10 MB です。
- INTERNAL_SERVER_ERROR — リクエストにサービスの問題が発生しました。API リクエストを再試行してください。

ドキュメントレベルのエラー

ClassifyDocument または DetectEntities API オペレーションが入力ドキュメントでドキュメントレベルのエラーを検出すると、API は`InvalidRequestException`エラーレスポンスを返します。

エラーレスポンスの Reason フィールドには `INVALID_DOCUMENT` 値が含まれています。

Detail フィールドは、次のいずれかの値を含みます。

- DOCUMENT_SIZE_EXCEEDED — ドキュメントのサイズが大きすぎます。 ファイルのサイズを確認して、リクエストを再送信してください。
- UNSUPPORTED_DOC_TYPE — ドキュメントタイプはサポートされていません。 ファイルタイプを確認して、リクエストを再送信してください。
- PAGE_LIMIT_EXCEEDED — ドキュメント内のページ数が多すぎます。 ファイルのページ数を確認して、リクエストを再送信してください。
- TEXTRACT_ACCESS_DENIED_EXCEPTION — Amazon Textract へのアクセスが拒否されました。 アカウントに Amazon Textract DetectDocumentTextおよび AnalyzeDocument API オペレーションを使用するアクセス許可があることを確認し、リクエストを再送信します。

非同期カスタム分析の入力

カスタム非同期分析ジョブには複数のドキュメントを入力できます。次のトピックでは、使用可能な入力ドキュメントタイプについて説明します。 最大ファイルサイズは、入力ドキュメントの種類によって異なります。

トピック

- プレーンテキストドキュメント
- 半構造化ドキュメント
- イメージファイルとスキャンした PDF ファイル
- Amazon Textract 出力 JSON サイズ

プレーンテキストドキュメント

プレーンテキストの入力ドキュメントはすべて UTF-8 形式のテキストを提供してください。次の表に、最大ファイルサイズとその他のガイドラインを示します。

> **Note**
>
> これらの制限は、すべての入力ファイルがプレーンテキストの場合に適用されます。

説明	クォータ/ガイドライン
ファイル形式ごとのドキュメントの最大ファイルサイズ(カスタム分類)	1 バイト ~ 10 MB。
ドキュメントサイズ (カスタムエンティティ認識)	1 バイト ~ 1 MB。
最大ファイル数 (1 ファイルあたり 1 ドキュメント)	1,000,000
1 行につき 1 つのドキュメントの最大合計数 (リクエスト中のすべてのファイル)	1,000,000
ドキュメントコーパスサイズ (プレーンテキストの全ドキュメントを含む)	1 バイト ~ 5 GB

半構造化ドキュメント

半構造化ドキュメントには、ネイティブ PDF ドキュメントと Word ドキュメントが含まれます。

次の表に、最大ファイルサイズとその他のガイドラインを示します。

説明	クォータ/ガイドライン
ドキュメントサイズ (PDF)	1 バイト ~ 50 MB
ドキュメントサイズ (Docx)	1 バイト ~ 5 MB
ファイルの最大数	500
PDF または Docx ファイルの最大ページ数	100
テキスト抽出後のドキュメントコーパスサイズ (プレーンテキスト、すべてのファイルを含む)	1 バイト ~ 5 GB

デフォルトでは、カスタム分析は Amazon Comprehend パーサーを使用して Word ファイルおよびデジタル PDF ファイルからテキストを抽出します。PDF ファイルの場合は、このデフォルトをオーバーライドして、Amazon Textract を使用してテキストを抽出できます。テキスト抽出オプションの設定 を参照してください。

イメージファイルとスキャンした PDF ファイル

カスタム分析は JPEG、PNG、TIFF 画像をサポートします。

次の表に、イメージの最大ファイルサイズを示します。スキャンした PDF ファイルには、ネイティブ PDF ファイルと同じく最大サイズが適用されます。

説明	クォータ/ガイドライン
画像サイズ (JPG または PNG)	1 バイト ~ 10 MB
画像サイズ (TIFF)	1 バイト ~ 10 MB。最大 1 ページ。

画像の詳細については、「画像のベストプラクティス」を参照してください。

デフォルトでは、Amazon Comprehend は Amazon Textract `DetectDocumentText` API オペレーションを使用して、画像ファイルおよびスキャンされた PDF ファイルからテキストを抽出します。このデフォルトをオーバーライドして、代わりに `AnalyzeDocument` API オペレーションを使用できます。テキスト抽出オプションの設定 を参照してください。

Amazon Textract 出力 JSON サイズ

カスタムエンティティ認識の場合は、カスタム分類ではなく、Amazon Textract `AnalyzeDocument` API オペレーションからの出力ファイルを分析ジョブの入力として指定できます。

テキスト抽出オプションの設定

デフォルトでは、Amazon Comprehend は入力ファイルのタイプに基づいて次のアクションを実行してファイルからテキストを抽出します。

- Word ファイル — Amazon Comprehend パーサーがテキストを抽出します。
- デジタル PDF ファイル — Amazon Comprehend パーサーがテキストを抽出します。

- 画像ファイルおよびスキャンされた PDF ファイル — Amazon Comprehend は Amazon Textract `DetectDocumentText` API を使用してテキストを抽出します。

画像ファイルや PDF ファイルの場合は、`DocumentReaderConfig` パラメーターを使用してデフォルトのテキスト抽出アクションをオーバーライドできます。このパラメータは、リアルタイムまたは非同期カスタム分析に Amazon Comprehend コンソールまたは API を使用すると利用できるようになります。

この `DocumentReaderConfig` パラメータには次の 3 つのフィールドがあります。

- DocumentReadMode – Amazon Comprehend `SERVICE_DEFAULT`がデフォルトのアクションを実行するには、 に設定します。

  Amazon Textract を使用してデジタル PDF ファイルを解析するには、`FORCE_DOCUMENT_READ_ACTION` に設定します。
- DocumentReadAction – Amazon Comprehend がテキスト抽出に Amazon Textract を使用する場合に使用する Amazon Textract API (DetectDocumentText または AnalyzeDocument) を設定します。
- FeatureTypes – AnalyzeDocument API オペレーションを使用するDocumentReadActionように を設定した場合、 `FeatureTypes` (TABLES、FORMS) の一方または両方を追加できます。これらの機能は、ドキュメント内の表とフォームに関する追加情報を提供します。これらの機能の詳細については、「Amazon Textract のドキュメント分析のレスポンスオブジェクト」を参照してください。

以下の例は、具体的なユースケースに応じた `DocumentReaderConfig` の設定方法を示しています。

1. すべての PDF ファイルに Amazon Textract を使用する。
   a. DocumentReadMode – に設定しますFORCE_DOCUMENT_READ_ACTION。
   b. DocumentReadAction – に設定しますTEXTRACT_DETECT_DOCUMENT_TEXT。
   c. FeatureTypes – 必須ではありません。
2. すべての PDF および画像ファイルに Amazon Textract を使用する。
   a. DocumentReadMode – に設定しますFORCE_DOCUMENT_READ_ACTION。
   b. DocumentReadAction – に設定しますTEXTRACT_ANALYZE_DOCUMENT。
   c. FeatureTypes - に設定するFORMSかTABLES、両方の機能。
3. スキャンされたすべての PDF およびすべての画像ファイルに Amazon Textract を使用する。

a. DocumentReadMode – に設定しますSERVICE_DEFAULT。
b. DocumentReadAction – に設定しますTEXTRACT_ANALYZE_DOCUMENT。
c. FeatureTypes – に設定するFORMSかTABLES、両方の機能。

Amazon Textract オプションの詳細については、「」を参照してくださいDocumentReaderConfig。

画像のベストプラクティス

カスタム分類またはカスタムエンティティ認識に画像ファイルを使用する場合、最良の結果を得るには以下のガイドラインに従ってください。

- できれば150 DPI 以上の高画質の画像を提供してください。
- 画像ファイルがサポートされている形式 (TIFF、JPEG、PNG) のいずれかを使用している場合は、Amazon S3 にアップロードする前に、ファイルを変換またはダウンサンプリングしないでください。

ドキュメント内のテーブルからテキストを抽出する際に、最良の結果を得るには、以下の方法に従ってください。

- ドキュメント内のテーブルは、ページ上の周囲の要素から視覚的に分離されています。たとえば、テーブルが画像や複雑なパターンに覆われていないこと。
- テーブル内のテキストは垂直していること。たとえば、テキストはページ上の他のテキストと比べて回転していないこと。

テーブルからテキストを抽出すると、次のような場合に一貫性のない結果が表示されることがあります。

- 結合された表のセルが複数の列にまたがっている。
- 表に、同じ表の他の部分とは異なるセル、行、または列がある。

カスタム分類

カスタム分類 を使用すると、自分で決めたカテゴリ (クラス) に文書を整理できます。カスタム分類は 2 ステップの処理です。まず、対象となるクラスを認識するようにカスタム分類モデル (分類子とも呼ばれる) を調教します。次に、そのモデルを使用して任意の数の文書セットを分類します。

たとえば、サポートリクエストの内容を分類して、リクエストを適切なサポートチームに送ることができます。また、顧客から受け取ったメールを分類して、顧客の要求の種類に基づいて案内できます。Amazon Comprehend と Amazon Transcribe を組み合わせて音声をテキストに変換し、サポート電話からのリクエストを分類できます。

1 文書に対してカスタム分類を同期的に (リアルタイム) 実行することも、非同期ジョブを開始して一連の文書を分類することもできます。アカウントには複数のカスタム分類子があり、それぞれが別データを使用して調教できます。カスタム分類では、プレーンテキスト、PDF、Word、画像など、種々の入力文書がサポートされます。

分類ジョブを送信するときは、分析する必要のある文書の種類に基づいて、使用する分類モデルを選択します。たとえば、プレーンテキスト文書を分析する場合、プレーンテキスト文書で調教したモデルを使用すると最も正確な結果が得られます。半構造化文書 (PDF、Word、画像、Amazon Textract 出力、スキャンファイルなど) を分析するには、ネイティブ文書で調教したモデルを使用すると最も正確な結果が得られます。

トピック

- 分類子調教データの作成
- 調教分類モデル
- リアルタイム分析の実行
- 非同期ジョブの実行

分類子調教データの作成

カスタム分類では、マルチクラスモードまたはマルチラベルモードでモデルを調教します。マルチクラスモードでは、各文書に 1 つのクラスが関連付けられます。マルチラベルモードでは、1 つ以上のクラスが各文書に関連付けられます。入力ファイル形式はモードごとに異なるため、調教データを作成する前に使用するモードを選択してください。

> **Note**
>
> Amazon Comprehend コンソールでは、マルチクラスモードをシングルラベルモードと呼んでいます。

カスタム分類は、プレーンテキストの文書で調教するモデルと、ネイティブ文書 (PDF、Word、画像など) で調教するモデルをサポートします。分類子モデルとそれらがサポートする文書タイプの詳細については、「調教分類モデル」を参照してください。

カスタム分類子モデルの調教データを作成するには:

1. この分類子に分析させたいクラスを特定します。使用するモード (マルチクラスまたはマルチラベル) を決定します。
2. モデルがプレーンテキスト文書の分析用か、半構造化文書の分析用かに応じて、分類子モデルのタイプを決定します。
3. 各クラスの文書例を集めます。最小調教要件については、「ドキュメント分類の一般的なクォータ」を参照してください。
4. プレーンテキストモデルの場合は、使用する調教ファイル形式 (CSV ファイルまたは拡張マニフェストファイル) を選択します。ネイティブ文書モデルを調教するには、必ず CSV ファイルを使用します。

トピック

- 分類子調教ファイルの形式
- マルチクラスモード
- マルチラベルモード

分類子調教ファイルの形式

プレーンテキストモデルの場合、分類子トレーニングデータを CSV ファイルまたは SageMaker Ground Truth を使用して作成した拡張マニフェストファイルとして提供できます。CSV ファイルまたは拡張マニフェストファイルには、各調教文書のテキストとそれに関連するラベルが含まれます。

ネイティブ文書モデルの場合は、分類子調教データを CSV ファイルとして提供します。CSV ファイルには、各調教文書のテキストとそれに関連するラベルが含まれます。調教文書は、調教グジョブの Amazon S3 入力フォルダに含めます。

CSV ファイル

ラベル付き調教データを UTF-8 でエンコードされたテキストとして CSV ファイルで提供します。ヘッダー行を含めないでください。ファイルにヘッダー行を追加すると、ランタイムエラーが発生する可能性があります。

CSV ファイルの各行の最初の列には 1 つ以上のクラスラベルが含まれます。クラスラベルは、有効な UTF-8 文字列であれば何でもかまいません。意味が重複しない明確なクラス名を使用することをお勧めします。名前には空白を含めることができ、複数の単語をアンダースコアまたはハイフンでつなげてもかまいません。

行内の値を区切るカンマの前後にスペース文字を入れないでください。

CSV ファイルの正確な内容は、分類子モードと調教データのタイプによって異なります。詳細については、「マルチクラスモード」と「マルチラベルモード」のセクションを参照してください。

拡張マニフェストファイル

拡張マニフェストファイルは、 SageMaker Ground Truth を使用して作成するラベル付きデータセットです。Ground Truth は、自分または自分の雇用する従業員が、機械学習モデルの調教データセットを構築するのに役立つデータラベル付けサービスです。

Ground Truth とその出力の詳細については、Amazon SageMaker デベロッパーガイドの「 SageMaker Ground Truth を使用してデータにラベルを付ける」を参照してください。

拡張マニフェストファイルは JSON 行形式になります。これらのファイルでは、各行は調教文書と関連ラベルを含む完全な JSON オブジェクトです。各行の正確な内容は、分類子モードによって異なります。詳細については、「マルチクラスモード」と「マルチラベルモード」のセクションを参照してください。

調教データを Amazon Comprehend に提供するときは、1 つ以上のラベル属性名を指定します。指定する属性名の数は、拡張マニフェストファイルが単一のラベリングジョブの出力であるか、チェーンラベリングジョブの出力であるかによって異なります。

ファイルが 1 つのラベル付けジョブの出力である場合は、Ground Truth ジョブの単一ラベル属性名を指定します。

ファイルがチェーンラベリングジョブの出力である場合は、チェーン内の 1 つ以上のジョブに対するラベル属性名を指定します。各ラベル属性名には、それぞれ 1 つのジョブのアノテーションが含まれます。チェーンラベリングジョブの拡張マニフェストファイルには、これらの属性のうち最大 5 つを指定できます。

連鎖ラベル付けジョブの詳細と、それらが生成する出力の例については、Amazon SageMaker デベロッパーガイドの「連鎖ラベル付けジョブ」を参照してください。

マルチクラスモード

マルチクラス分類では、各文書に 1 つのクラスが割り当てられます。個々のクラスは相互に排他的です。たとえば、映画をコメディーかサイエンスフィクションに分類できますが、両方には分類できません。

> **Note**
>
> Amazon Comprehend コンソールでは、マルチクラスモードをシングルラベルモードと呼んでいます。

トピック

- プレーンテキストモデル
- ネイティブ文書モデル

プレーンテキストモデル

プレーンテキストモデルをトレーニングするには、ラベル付きトレーニングデータを CSV ファイルまたは SageMaker Ground Truth の拡張マニフェストファイルとして提供できます。

CSV ファイル

調教分類子用 CSV ファイルの使用に関する一般的な情報は、「CSV ファイル」を参照してください。

調教データを 2 列の CSV ファイルとして提供します。各行の最初の列にはクラスラベルの値が入ります。2 列目には、そのクラスのサンプルテキスト文書が含まれています。各行は\n または\r\n で終えなければなりません。

3 つの文書を含んだ CSV ファイルの例を以下に示します。

```
CLASS,Text of document 1
CLASS,Text of document 2
CLASS,Text of document 3
```

次の例は、電子メールメッセージがスパムかどうかを検出するようにカスタム分類子を調教する CSV ファイルの 1 行を示しています。

```
SPAM,"Paulo, your $1000 award is waiting for you! Claim it while you still can at
 http://example.com."
```

拡張マニフェストファイル

調教分類子用拡張マニフェストファイルの使用に関する一般的な情報は、「拡張マニフェストファイル」を参照してください。

プレーンテキスト文書の場合、拡張マニフェストファイルの各行は、調教文書、単一のクラス名、Ground Truth からのその他のメタデータを含んだ完全な JSON オブジェクトです。次の例は、スパムメールメッセージを認識するようにカスタム分類子を調教する拡張マニフェストファイルです。

```
{"source":"Document 1 text", "MultiClassJob":0, "MultiClassJob-metadata":
{"confidence":0.62, "job-name":"labeling-job/multiclassjob", "class-name":"not_spam",
 "human-annotated":"yes", "creation-date":"2020-05-21T17:36:45.814354",
 "type":"groundtruth/text-classification"}}
{"source":"Document 2 text", "MultiClassJob":1, "MultiClassJob-metadata":
{"confidence":0.81, "job-name":"labeling-job/multiclassjob", "class-name":"spam",
 "human-annotated":"yes", "creation-date":"2020-05-21T17:37:51.970530",
 "type":"groundtruth/text-classification"}}
{"source":"Document 3 text", "MultiClassJob":1, "MultiClassJob-metadata":
{"confidence":0.81, "job-name":"labeling-job/multiclassjob", "class-name":"spam",
 "human-annotated":"yes", "creation-date":"2020-05-21T17:37:51.970566",
 "type":"groundtruth/text-classification"}}
```

次の例は、拡張マニフェストファイル内の 1 つの JSON オブジェクトを、読みやすくフォーマットして示しています。

```
{
   "source": "Paulo, your $1000 award is waiting for you! Claim it while you still can
 at http://example.com.",
   "MultiClassJob": 0,
   "MultiClassJob-metadata": {
       "confidence": 0.98,
       "job-name": "labeling-job/multiclassjob",
       "class-name": "spam",
       "human-annotated": "yes",
       "creation-date": "2020-05-21T17:36:45.814354",
```

```
        "type": "groundtruth/text-classification"
    }
}
```

この例では、source 属性は調教文書のテキストを示し、MultiClassJob 属性は分類リストからクラスのインデックスを割り当てます。job-name 属性は、Ground Truth でラベル付けジョブ用に定義した名前です。

Amazon Comprehend で分類子調教ジョブを開始するときは、同じラベリングジョブ名を指定します。

ネイティブ文書モデル

ネイティブ文書モデルは、ネイティブ文書 (PDF、DOCX、画像など) を使用して調教するモデルです。調教データを CSV ファイルとして提供します。

CSV ファイル

調教分類子用 CSV ファイルの使用に関する一般的な情報は、「CSV ファイル」を参照してください。

調教データを 3 列の CSV ファイルとして提供します。各行の最初の列にはクラスラベルの値が入ります。2 列目には、そのクラスのサンプル文書が入ります。3 列目にはページ番号が入ります。サンプル文書が画像の場合、ページ番号は省略可能です。

3 つの入力文書を示す CSV ファイルの例を以下に示します。

```
CLASS,input-doc-1.pdf,3
CLASS,input-doc-2.docx,1
CLASS,input-doc-3.png
```

次の例は、電子メールメッセージがスパムかどうかを検出するようにカスタム分類子を調教する CSV ファイルの 1 行を示しています。PDF ファイルの 2 ページ目には、スパムの例が含まれています。

```
SPAM,email-content-3.pdf,2
```

マルチラベルモード

マルチラベルモードでは、個々のクラスは相互に排他的ではない異種カテゴリを表します。マルチクラス分類では、各文書に 1 つまたは複数のクラスが割り当てられます。たとえば、ある映画をド

キュメンタリーとして分類し、別の映画をサイエンスフィクション、アクション、コメディーとして分類できます。

調教用に、マルチラベルモードでは最大 100 のユニークなクラスを含む最大 100 万件のサンプルがサポートされます。

トピック

- プレーンテキストモデル
- ネイティブ文書モデル

プレーンテキストモデル

プレーンテキストモデルをトレーニングするには、ラベル付きトレーニングデータを CSV ファイルまたは SageMaker Ground Truth の拡張マニフェストファイルとして提供できます。

CSV ファイル

調教分類子用 CSV ファイルの使用に関する一般的な情報は、「CSV ファイル」を参照してください。

調教データを 2 列の CSV ファイルとして提供します。各行の最初の列にはクラスラベルの値が含まれ、2 番目の列にはこれらのクラスのサンプルテキスト文書が含まれます。1 列目に複数のクラスを入力するには、各クラスの間に区切り文字 (| など) を使用します。

```
CLASS,Text of document 1
CLASS,Text of document 2
CLASS|CLASS|CLASS,Text of document 3
```

次の例は、映画の抄録に含まれるジャンルを検出するようにカスタム分類子を調教する CSV ファイルの 1 行を示しています。

```
COMEDY|MYSTERY|SCIENCE_FICTION|TEEN,"A band of misfit teens become unlikely detectives
 when they discover troubling clues about their high school English teacher. Could the
 strange Mrs. Doe be an alien from outer space?"
```

クラス名の間のデフォルトの区切り文字はパイプ (|) です。ただし、別の文字を区切り文字として使用できます。区切り文字はクラス名のすべての文字と区別する必要があります。たとえば、クラスが CLASS_1、CLASS_2、CLASS_3 の場合、アンダースコア (_) はクラス名の一部です。そのため、クラス名を区切る区切り文字として、アンダースコアは使用しないでください。

拡張マニフェストファイル

調教分類子用拡張マニフェストファイルの使用に関する一般的な情報は、「拡張マニフェストファイル」を参照してください。

プレーンテキスト文書の場合、拡張マニフェストファイルの各行は完全な JSON オブジェクトです。これには、調教文書、クラス名、Ground Truthのその他のメタデータが含まれています。次の例は、映画の抄録内のジャンルを検出するようにカスタム分類子を調教する拡張マニフェストファイルです。

```
{"source":"Document 1 text", "MultiLabelJob":[0,4], "MultiLabelJob-metadata":{"job-
name":"labeling-job/multilabeljob", "class-map":{"0":"action", "4":"drama"}, "human-
annotated":"yes", "creation-date":"2020-05-21T19:02:21.521882", "confidence-map":
{"0":0.66}, "type":"groundtruth/text-classification-multilabel"}}
{"source":"Document 2 text", "MultiLabelJob":[3,6], "MultiLabelJob-metadata":{"job-
name":"labeling-job/multilabeljob", "class-map":{"3":"comedy", "6":"horror"}, "human-
annotated":"yes", "creation-date":"2020-05-21T19:00:01.291202", "confidence-map":
{"1":0.61,"0":0.61}, "type":"groundtruth/text-classification-multilabel"}}
{"source":"Document 3 text", "MultiLabelJob":[1], "MultiLabelJob-metadata":
{"job-name":"labeling-job/multilabeljob", "class-map":{"1":"action"}, "human-
annotated":"yes", "creation-date":"2020-05-21T18:58:51.662050", "confidence-map":
{"1":0.68}, "type":"groundtruth/text-classification-multilabel"}}
```

次の例は、拡張マニフェストファイル内の 1 つの JSON オブジェクトを、読みやすくフォーマットして示しています。

```
{
      "source": "A band of misfit teens become unlikely detectives when
                  they discover troubling clues about their high school English
 teacher.
                    Could the strange Mrs. Doe be an alien from outer space?",
      "MultiLabelJob": [
          3,
          8,
          10,
          11
      ],
      "MultiLabelJob-metadata": {
          "job-name": "labeling-job/multilabeljob",
          "class-map": {
              "3": "comedy",
              "8": "mystery",
```

```
            "10": "science_fiction",
            "11": "teen"
        },
        "human-annotated": "yes",
        "creation-date": "2020-05-21T19:00:01.291202",
        "confidence-map": {
            "3": 0.95,
            "8": 0.77,
            "10": 0.83,
            "11": 0.92
        },
        "type": "groundtruth/text-classification-multilabel"
    }
}
```

この例では、source 属性は調教文書のテキストを示し、MultiLabelJob 属性は分類リストから複数のクラスのインデックスを割り当てます。MultiLabelJob メタデータのジョブ名は、Ground Truth でラベル付けジョブ用に定義した名前です。

ネイティブ文書モデル

ネイティブ文書モデルは、ネイティブ文書 (PDF、DOCX、画像ファイルなど) を使用して調教するモデルです。ラベル付きの調教データを CSV ファイルとして提供します。

CSV ファイル

調教分類子用 CSV ファイルの使用に関する一般的な情報は、「CSV ファイル」を参照してください。

調教データを 3 列の CSV ファイルとして提供します。各行の最初の列にはクラスラベルの値が入ります。2 列目には、そのクラスのサンプル文書が入ります。3 列目にはページ番号が入ります。サンプル文書が画像の場合、ページ番号は省略可能です。

1 列目に複数のクラスを入力するには、各クラスの間に区切り文字 (| など) を使用します。

```
CLASS,input-doc-1.pdf,3
CLASS,input-doc-2.docx,1
CLASS|CLASS|CLASS,input-doc-3.png,2
```

次の例は、映画の抄録に含まれるジャンルを検出するようにカスタム分類子を調教する CSV ファイルの 1 行を示しています。PDF ファイルの 2 ページ目には、コメディー/ティーン向け映画の例が含まれています。

```
COMEDY|TEEN,movie-summary-1.pdf,2
```

クラス名の間のデフォルトの区切り文字はパイプ (|) です。ただし、別の文字を区切り文字として使用できます。区切り文字はクラス名のすべての文字と区別する必要があります。たとえば、クラスが CLASS_1、CLASS_2、CLASS_3 の場合、アンダースコア (_) はクラス名の一部です。そのため、クラス名を区切る区切り文字として、アンダースコアは使用しないでください。

調教分類モデル

カスタム分類用にモデルを調教するには、カテゴリを定義し、カスタムモデルを調教するためのサンプル文書を指定します。モデルをマルチクラスモードまたはマルチラベルモードで調教します。マルチクラスモードでは、各文書に 1 つのクラスが関連付けられます。マルチラベルモードでは、1 つ以上のクラスが各文書に関連付けられます。

カスタム分類では、プレーンテキストモデルとネイティブ文書モデルの 2 種類の分類モデルがサポートされます。プレーンテキストモデルでは、テキストコンテンツに基づいて文書を分類します。ネイティブ文書モデルも、テキストコンテンツに基づいて文書を分類できます。ネイティブ文書モデルでは、文書のレイアウトなどからの付加的なシグナルも使用できます。ネイティブ文書モデルをネイティブ文書で調教し、モデルにレイアウト情報を学習させます。

プレーンテキストモデルには以下の特性があります。

- UTF-8 でエンコードされたテキスト文書を使用してモデルを調教します。
- 英語、スペイン語、ドイツ語、イタリア語、フランス語、ポルトガル語のいずれかの言語の文書を使用してモデルを調教できます。
- 特定の分類子の調教文書は、どれも同じ言語を使用する必要があります。
- 調教文書はプレーンテキストなので、テキスト抽出に追加料金はかかりません。

ネイティブ文書モデルには以下の特性があります。

- 以下の文書タイプを含む半構造化文書を使用してモデルを調教します。
  - デジタル文書とスキャンした PDF 文書。
  - Word 文書 (.docx)
  - 画像：JPG ファイル、PNG ファイル、単一ページの TIFF ファイル。
  - API 出力 JSON ファイルをテキスト抽出します。

- 英語の文書を使用してモデルを調教します。
- 調教文書にスキャンした文書ファイルが含まれている場合は、テキスト抽出に追加料金がかかります。詳細については、Amazon Comprehend の料金ページを参照してください。

どちらのタイプのモデルを使用しても、サポートされているどの文書タイプでも分類できます。ただし、最も正確な結果を得るには、プレーンテキストモデルを使用してプレーンテキスト文書、ネイティブ文書モデルを使用して半構造化文書を分類するようお勧めします。

トピック

- カスタム分類子の調教 (コンソール)
- カスタム分類子の調教 (API)
- 調教データのテスト
- 分類子の調教出力
- カスタム分類子メトリクス

カスタム分類子の調教 (コンソール)

コンソールを使用してカスタム分類子を作成して調教し、そのカスタム分類子を使用して文書を分析できます。

カスタム分類子を調教するには、一連の調教文書が必要です。これらの文書には、文書分類子に認識させたいカテゴリのラベルを付けます。調教文書の作成については、「分類子調教データの作成」を参照してください。

文書分類モデルを作成して調教するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/comprehend/ で Amazon Comprehend コンソールを開きます。
2. 左側のメニューから [カスタマイズ] を選択し、[カスタム分類] を選択します。
3. [モデルの作成] を選択します。
4. [モデルの設定] に分類子のモデル名を入力します。この名前は、自分のアカウント内と現在のリージョンで一意でなければなりません。

   (オプション) バージョン名を入力します。この名前は、自分のアカウント内と現在のリージョンで一意でなければなりません。

5. 調教文書の言語を選択します。分類子がサポートする言語については、「調教分類モデル」を参照してください。
6. (オプション) Amazon Comprehend が調教ジョブを処理している間にストレージボリューム内のデータを暗号化する場合は、[分類子の暗号化]を選択します。次に、自分の現在のアカウントに関連付けられた KMS キーを使用するか、別のアカウントの KMS キーを使用するかを選択します。
   - 現在のアカウントに関連付けられているキーを使用している場合は、KMS キー ID のキー ID を選択します。
   - 別のアカウントに関連付けられているキーを使用している場合は、KMS キー ARN の下にキー ID の ARN を入力します。

   > **Note**
   >
   > KMS キーの作成と使用や関連する暗号化の詳細については、AWS Key Management Service (AWS KMS) を参照してください。

7. 「データ仕様」で、使用する [調教モデルタイプ] を選択します。
   - プレーンテキスト文書：このオプションを選択すると、プレーンテキストモデルが作成されます。プレーンテキスト文書を使用してモデルを調教します。
   - ネイティブ文書：ネイティブ文書モデルを作成するには、このオプションを選択します。ネイティブ文書 (PDF、Word、画像) を使用してモデルを調教します。
8. 調教データの [データ形式] を選択します。データ形式の詳細については、「分類子調教ファイルの形式」を参照してください。
   - CSV ファイル：調教データが CSV ファイル形式を使用している場合は、このオプションを選択してください。
   - 拡張マニフェスト：Ground Truth を使用して調教データ用の拡張マニフェストファイルを作成した場合は、このオプションを選択してください。この形式は、調教モデルタイプとして [プレーンテキスト文書] を選択した場合に使用できます。
9. 使用する [分類子モード] を選択します。
   - シングルラベルモード：文書に割り当てるカテゴリが相互に排他的であり、各文書に 1 つのラベルを割り当てるように分類子を学習させる場合は、このモードを選択します。Amazon Comprehend API では、シングルラベルモードはマルチクラスモードと呼ばれています。

- マルチラベルモード：1 つの文書に複数のカテゴリを同時に適用でき、各文書に 1 つまたは複数のラベルを割り当てるように分類子を調教する場合は、このモードを選択します。

10. マルチラベルモードを選択すると、ラベルの区切り文字を選択できます。調教文書に複数のクラスがある場合は、この区切り文字を使用してラベルを区切ります。デフォルトの区切り文字はパイプ文字です。
11. (オプション) データ形式として拡張マニフェストを選択した場合、最大 5 つの拡張マニフェストファイルを入力できます。各拡張マニフェストファイルには、調教データセットまたはテストデータセットが含まれます。少なくとも 1 つの調教データセットを指定する必要があります。テストデータセットは任意です。次の手順に従って、拡張マニフェストファイルを設定します。

    a. 「調教とテストデータセット」で、「入力場所」パネルを展開します。

    b. [データセットタイプ] で [調教データ] または [テストデータ] を選択します。

    c. SageMaker Ground Truth 拡張マニフェストファイル S3 の場所 には、マニフェストファイルを含む Amazon S3 バケットの場所を入力するか、Browse S3 を選択してそこに移動します。調教ジョブのアクセス許可に使用する IAM ロールには、S3 バケットに対する読み取り許可が必要です。

    d. [属性名] に、注釈を含む属性の名前を入力します。ファイルに複数のチェーンラベリングジョブのアノテーションが含まれている場合は、ジョブごとに属性を追加します。

    e. 別の入力場所を追加するには、[入力場所を追加] を選択し、次の場所を設定します。

12. (オプション) データ形式として CSV ファイルを選択した場合は、次の手順に従って調教データセットとオプションのテストデータセットを設定します。

    a. [調教データセット] で、調教データの CSV ファイルを含む Amazon S3 バケットの場所を入力するか、[S3 を参照] を選択してその場所に移動します。調教ジョブのアクセス許可に使用する IAM ロールには、S3 バケットに対する読み取り許可が必要です。

    (オプション) 調教モデルタイプとしてネイティブ文書を選択した場合は、調教サンプルファイルの入った Amazon S3 フォルダの URL も指定します。

    b. [テストデータセット] で、調教済みモデルをテストするための追加データを Amazon Comprehend に提供するかどうかを選択します。

    - Autosplit: Autosplit は調教データの 10% を自動的に選択し、テストデータとして使用するために確保します。
    - (オプション) 顧客提供: Amazon S3 のテストデータの CSV ファイルの URL を入力します。Amazon S3 内のその場所に移動して [フォルダを選択] を選択することもできます。

(オプション) 調教モデルタイプとしてネイティブ文書を選択した場合は、テストファイルの入った Amazon S3 フォルダの URL も指定します。

13. (オプション) 文書読み取りモードでは、デフォルトのテキスト抽出アクションをオーバーライドできます。このオプションはスキャンされた文書のテキスト抽出に適用されるため、プレーンテキストモデルには必要ありません。詳細については、「テキスト抽出オプションの設定」を参照してください。
14. (プレーンテキストモデルの場合はオプション) [出力データ] には、混同行列などの調教出力データを保存する Amazon S3 バケットの場所を入力します。詳細については、「混同行列」を参照してください。

    (オプション) 調教ジョブの出力結果を暗号化する場合は、[暗号化] を選択します。次に、現在のアカウントに関連付けられた KMS キーを使用するか、別のアカウントの KMS キーを使用するかを選択します。

    - 現在のアカウントに関連付けられているキーを使用している場合は、KMS キー ID のキーエイリアスを選択します。
    - 別のアカウントに関連付けられているキーを使用している場合は、KMS キー ID の下にキーエイリアス の ARN または ID を入力します。
15. IAM ロールには、[既存の IAM ロールを選択] を選択し、調教文書を含む S3 バケットの読み取り権限を持つ既存の IAM ロールを選択します。ロールには、`comprehend.amazonaws.com`で始まる信頼ポリシーが必要です。

    このような権限を持つ IAM ロールがまだない場合は、[IAM ロールの作成] を選択して作成してください。このロールを付与するアクセス権限を選択し、アカウント内の IAM ロールと区別できるように名前のサフィックスを選択します。

    > **Note**
    >
    > 暗号化された入力文書の場合、使用する IAM ロールにも権限が必要です。`kms:Decrypt`詳細については、「KMS 暗号化を使用するために必要なアクセス許可」を参照してください。

16. (オプション) VPC から Amazon Comprehend にリソースを起動するには、VPC の下に VPC ID を入力するか、ドロップダウンリストから ID を選択します。

    1. [サブネット] でサブネットを選択します。最初のサブネットを選択すると、追加のサブネットを選択できます。

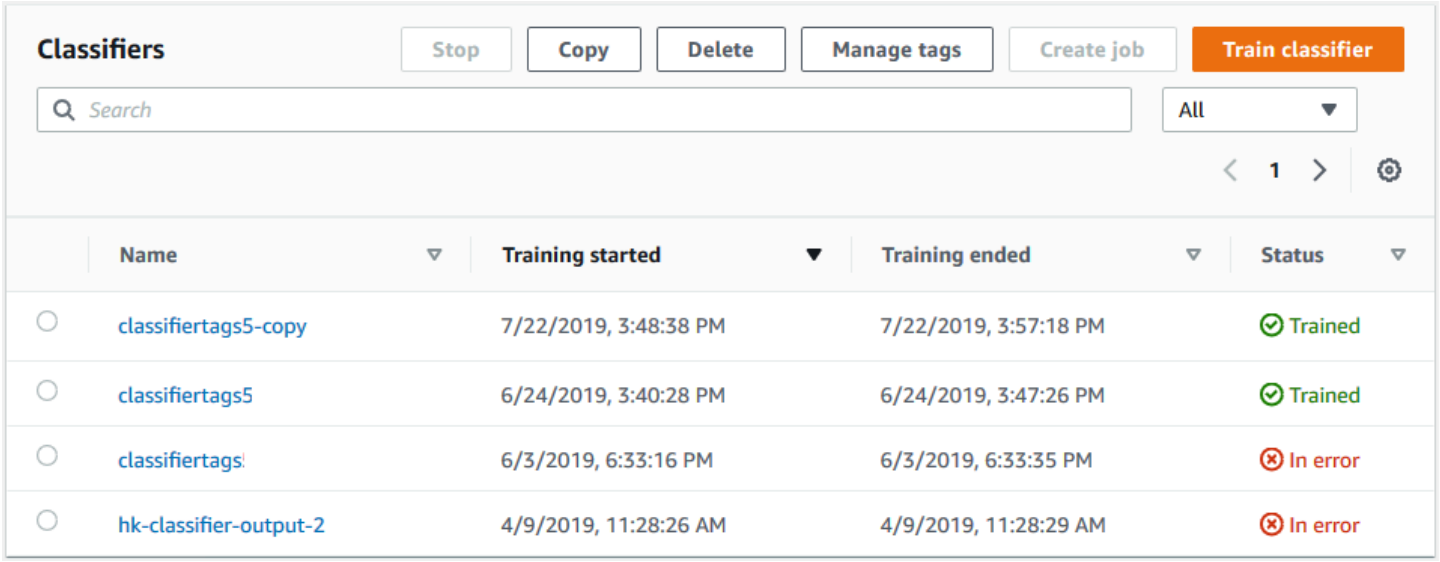
2. セキュリティグループを指定した場合は、[セキュリティグループ] で、使用するセキュリティグループを選択します。最初のセキュリティグループを選択すると、追加のセキュリティグループを選択できます。

> **Note**
>
> 分類ジョブで VPC を使用する場合、[作成] と [開始] 操作に使用する DataAccessRole には、入力文書と出力バケットへの VPC アクセス権限が必要です。

17. (オプション) カスタム分類子にタグを追加するには、[タグ] にキーと値のペアを入力します。[タグを追加] を選択します。分類子を作成する前にこのペアを削除するには、[タグを削除] を選択します。詳細については、「リソースのタグ付け」を参照してください
18. [Create] (作成) を選択します。

コンソールに「分類子」ページが表示されます。新しい分類子が表で表示され、そのステータス Submitted が表示されます。分類子が調教文書の処理を開始すると、ステータスが Training に変わります。分類子が使用できるようになると、ステータスが Trained または Trained with warnings に変わります。ステータスが TRAINED_WITH_WARNINGS の場合、分類子の調教出力 のスキップしたファイルのフォルダを確認してください。

Amazon Comprehend の作成中または調教中にエラーが発生した場合、ステータスは In error に変わります。表中の分類子ジョブを選択すると、エラーメッセージを含む分類子に関する詳細情報を取得できます。



カスタム分類子の調教 (API)

カスタム分類子を作成してトレーニングするには、 CreateDocumentClassifierオペレーションを使用します。

DescribeDocumentClassifier オペレーションを使用して、リクエストの進行状況をモニタリングできます。Status フィールドが TRAINED に移行すると、分類子を使用して文書を分類できます。ステータスが TRAINED_WITH_WARNINGS の場合、演算 CreateDocumentClassifier でスキップした 分類子の調教出力 の中のファイルフォルダを確認してください。

トピック

- を使用したカスタム分類のトレーニング AWS Command Line Interface
- AWS SDK for Java または SDK for Python の使用

を使用したカスタム分類のトレーニング AWS Command Line Interface

以下の例は、演算機能 CreateDocumentClassifier、演算機能 DescribeDocumentClassificationJob、その他のカスタム分類子 API を AWS CLIで使用する方法を示しています。

例は、Unix、Linux、macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

演算機能 create-document-classifier を使用してプレーンテキストのカスタム分類子を作成します。

```
aws comprehend create-document-classifier \
     --region region \
     --document-classifier-name testDelete \
     --language-code en \
     --input-data-config S3Uri=s3://S3Bucket/docclass/file name \
     --data-access-role-arn arn:aws:iam::account number:role/testFlywheelDataAccess
```

ネイティブカスタム分類子を作成するには、create-document-classifier リクエストに以下の追加パラメータを指定します。

1. DocumentType: 値を SEMI_STRUCTURED_DOCUMENT に設定します。
2. 文書：調教文書 (およびオプションでテスト文書) を保管する S3 の場所。

3. OutputDataConfig: 出力ドキュメントの S3 の場所 (およびオプションの KMS キー) を指定します。
4. DocumentReaderConfig: テキスト抽出設定のオプションフィールド。

```
aws comprehend create-document-classifier \
     --region region \
     --document-classifier-name testDelete \
     --language-code en \
     --input-data-config
          S3Uri=s3://S3Bucket/docclass/file name \
           DocumentType \
             Documents  \
     --output-data-config S3Uri=s3://S3Bucket/docclass/file name \
     --data-access-role-arn arn:aws:iam::account number:role/testFlywheelDataAccess
```

演算機能 DescribeDocumentClassifier を使用して、文書分類子 ARN を含んだカスタム分類子に関する情報を取得します。

```
aws comprehend describe-document-classifier \
     --region region \
     --document-classifier-arn arn:aws:comprehend:region:account number:document-
classifier/file name
```

演算機能 DeleteDocumentClassifier を使用してカスタム分類子を削除します。

```
aws comprehend delete-document-classifier \
     --region region \
     --document-classifier-arn arn:aws:comprehend:region:account number:document-
classifier/testDelete
```

演算機能 ListDocumentClassifiers を使用して、アカウント内のすべてのカスタム分類子を一覧表示します。

```
aws comprehend list-document-classifiers
     --region region
```

AWS SDK for Java または SDK for Python の使用

カスタム分類子を作成して調教する方法の SDK の例については、「AWS SDK または CLI CreateDocumentClassifierで を使用する」を参照してください。

調教データのテスト

モデルを調教した後、Amazon Comprehend はカスタム分類子モデルをテストします。テストデータセットを指定しないと、Amazon Comprehend は調教データの 90% を使用してモデルを調教します。調教データの 10% はテスト用に確保されます。テストデータセットを与える場合、テストデータには調教データセット内の固有のラベルごとに少なくとも 1 つの例が含まれている必要があります。

モデルをテストすると、モデルの精度推定に使用できる指標が得られます。コンソールの分類子の詳細ページの分類子パフォーマンスセクションに指標が表示されます。これらは、DescribeDocumentClassifierオペレーションによって返されるMetricsフィールドにも返されます。

次の調教データ例では、DOCUMENTARY、DOCUMENTARY、SCIENCE_FICTION、DOCUMENTARY、ROMANTIC_COMEDY の 5 つのラベルがあります。DOCUMENTARY、SCIENCE_FICTION、ROMANTIC_COMEDY の 3 つの固有クラスがあります。

列 1	列 2
DOCUMENTARY	文書テキスト 1
DOCUMENTARY	文書テキスト 2
SCIENCE_FICTION	文書テキスト 3
DOCUMENTARY	文書テキスト 4
ROMANTIC_COMEDY	文書テキスト 5

自動分割 (Amazon Comprehend が調教データの 10% をテスト用に確保) では、調教データに含まれている特定のラベルが限られていると、テストデータセットにはそのラベルの例が全くないという結果になり得ます。たとえば、調教データセットに DOCUMENTARY クラスのインスタンスが 1000 個、SCIENCE_FICTION のインスタンスが 900 個、ROMANTIC_COMEDY クラスのインスタンスが 1 つしかない場合、テストデータセットには 100 個のドキュメンタリーインスタンスと 90 個の

SCIENCE_FICTION インスタンスが含まれていても、ROMANTIC_COMEDY インスタンスはゼロになる可能性があります。

モデルの調教が終わると、調教指標で得られる情報から、モデルがニーズに十分合っているかどうかを判断できます。

分類子の調教出力

Amazon Comprehend は、カスタム分類子モデルトレーニングを完了すると、CreateDocumentClassifierAPI リクエストまたは同等のコンソールリクエストで指定した Amazon S3 出力場所に出力ファイルを作成します。

Amazon Comprehend では、プレーンテキストモデルまたはネイティブ文書モデルを調教すると、混同行列が生成されます。ネイティブ文書モデルを調教すると、追加の出力ファイルが作成されることがあります。

トピック

- 混同行列
- ネイティブ文書モデルの追加出力

混同行列

カスタム分類子モデルを調教すると、Amazon Comprehend はモデルが調教でどの程度うまく機能したかを示す指標を提示した混同行列を生成します。この行列には、モデルが予測したラベルの行列と、実際の文書ラベルとの比較が表示されます。Amazon Comprehend は調教データの一部を使用して混同行列を生成します。

混同行列は、どのクラスがより多くのデータを使用してモデルのパフォーマンスを改善できるかを示します。正しい予測の割合が高いクラスは、行列の対角線上の結果の数値が最大になります。対角線上の数値が小さい場合、そのクラスの予測の正解率は低くなります。このクラスにさらに調教例を追加して、モデルを再調教できます。たとえば、ラベル A のサンプルの 40% がラベル D に分類される場合、ラベル A とラベル D のサンプルを増やすと分類子のパフォーマンスが向上します。

Amazon Comprehend が分類子モデルを作成すると、S3 出力場所の `confusion_matrix.json` ファイル内で混同行列が使用できるようになります。

混同行列の形式は、分類子をマルチクラスモードとマルチラベルモードのどちらで調教したかによって異なります。

トピック

- マルチクラスモードの混同行列
- マルチラベルモードの混同行列

マルチクラスモードの混同行列

マルチクラスモードでは、個々のクラスは相互に排他的であるため、分類によって各文書に 1 つのラベルが割り当てられます。たとえば、動物は犬でも猫でもかまいませんが、両方を同時にはできません。

マルチクラス調教分類子の混同行列の例を考えてみましょう。

```
  A B X Y <-(predicted label)
A 1 2 0 4
B 0 3 0 1
X 0 0 1 0
Y 1 1 1 1
^
|
(actual label)
```

この場合、モデルの予測は以下のとおりです。

- 1 つの「A」ラベルが正確に予測され、2 つの「A」ラベルが「B」ラベルとして誤って予測され、4 つの「A」ラベルが「Y」ラベルとして誤って予測されました。
- 3 つの「B」ラベルが正確に予測され、1 つの「B」ラベルが「Y」ラベルとして誤って予測されました。
- 1 つの「X」が正確に予測されました。
- 1 つの「Y」ラベルが正確に予測され、1 つは「A」ラベルとして誤って予測され、もう 1 つは「B」ラベルとして誤って予測され、もう 1 つは「X」ラベルとして誤って予測されました。

行列内の対角線 (A:A、B:B、X:X、Y:Y) は、正確な予測を示しています。予測誤差は対角線の外側の値です。この場合、行列に以下の誤予測率が表示されます。

- A ラベル: 86%
- B ラベル:25%
- X ラベル:0%

- Y ラベル: 75%

分類子は、混同行列を JSON 形式のファイルとして返します。次の JSON ファイルは、前の例の行列を示しています。

```
{
 "type": "multi_class",
 "confusion_matrix": [
 [1, 2, 0,4],
 [0, 3, 0, 1],
 [0, 0, 1, 0],
 [1, 1, 1, 1]],
 "labels": ["A", "B", "X", "Y"],
 "all_labels": ["A", "B", "X", "Y"]
}
```

マルチラベルモードの混同行列

マルチラベルモードでは、各文書に 1 つまたは複数のクラスが割り当てられます。マルチクラス調教分類子の混同行列の例を考えてみましょう。

この例では、Comedy、Action、Drama の 3 つのラベルが考えられます。マルチラベル混同行列は、ラベルごとに 1 つの 2x2 行列を生成します。

```
Comedy                   Action                   Drama
     No Yes                   No Yes                   No Yes   <-(predicted label)

 No  2   1                No  1   1                No  3   0

Yes  0   2               Yes  2   1               Yes  1   1
 ^                        ^                        ^
 |                        |                        |
 |-----------(was this label actually used)--------|
```

この場合、モデルは Comedy ラベルに対して以下を返しました。

- Comedy ラベルの存在が正確に予測された 2 つの例。真陽性 (TP)。
- Comedy ラベルの不在が正確に予測された 2 つの例。真陰性 (TN)。
- Comedy ラベルの存在が誤って予測された例がゼロ。偽陽性 (FP)。

- Comedy ラベルの不在が誤って予測された 1 つの例。偽陰性 (FN)。

マルチクラス混同行列と同様、各行列の対角線には正確な予測値が表示されます。

この場合、モデルは 80% の確率で Comedy ラベルを正確に予測し (TP と TN)、20% の確立で予測を誤りました (FP と FN)。

分類子は、混同行列を JSON 形式のファイルとして返します。次の JSON ファイルは、前の例の行列を示しています。

```
{
"type": "multi_label",
"confusion_matrix": [
 [[2, 1],
 [0, 2]],
 [[1, 1],
 [2, 1]],
 [[3, 0],
 [1, 1]]
],
"labels": ["Comedy", "Action", "Drama"]
"all_labels": ["Comedy", "Action", "Drama"]
}
```

ネイティブ文書モデルの追加出力

Amazon Comprehend では、ネイティブ文書モデルを調教すると、追加の出力ファイルが作成されることがあります。

Amazon Textract 出力

Amazon Comprehend が Amazon Textract API を呼び出して調教文書のテキストを抽出した場合、Amazon Textract の出力ファイルを S3 の出力場所に保存します。以下のディレクトリ構造を使用します。

- 調教文書:

  amazon-textract-output/train/<file_name>/<page_num>/textract_output.json

- テスト文書:

```
amazon-textract-output/test/<file_name>/<page_num>/textract_output.json
```

API リクエストでテスト文書を指定した場合、Amazon Comprehend はテストフォルダにデータを入力します。

文書注釈の失敗

注釈が失敗すると、Amazon Comprehend は Amazon S3 の出力場所 (skipped_documents/ フォルダ) に次のファイルを作成します。

- failed_annotations_train.jsonl

  調教データで注釈が失敗しても、ファイルは存在します。
- failed_annotations_test.jsonl

  リクエストにテストデータが含まれていて、そのテストデータ内の注釈が失敗しても、ファイルは存在します。

注釈が失敗したファイルは、以下の形式の JSONL ファイルになります。

```
{
     "File": "String", "Page": Number, "ErrorCode": "...", "ErrorMessage": "..."}
    {"File": "String", "Page": Number, "ErrorCode": "...", "ErrorMessage": "..."
  }
```

カスタム分類子メトリクス

Amazon Comprehend には、カスタム分類子のパフォーマンスを推定するのに役立つメトリクスが用意されています。Amazon Comprehend は、分類子調教ジョブのテストデータを使用してメトリクスを計算します。メトリクスは調教中のモデルのパフォーマンスを正確に表しているため、類似データを分類する際のモデルのパフォーマンスに近い値になります。

などの API オペレーションDescribeDocumentClassifierを使用して、カスタム分類子のメトリクスを取得します。

> **Note**
>
> 基礎となるプレシジョン、リコール、F1 スコアのメトリクスについて詳しくは、「メトリクス:精度、再現率、FScore」を参照してください。これらのメトリクスはクラスレベルで定義

されます。Amazon Comprehend は、以下で説明するように、マクロ平均化によりこれらのメトリクスをテストセット P、R、F1 に結合します。

トピック

- メトリクス
- カスタム分類子のパフォーマンス向上

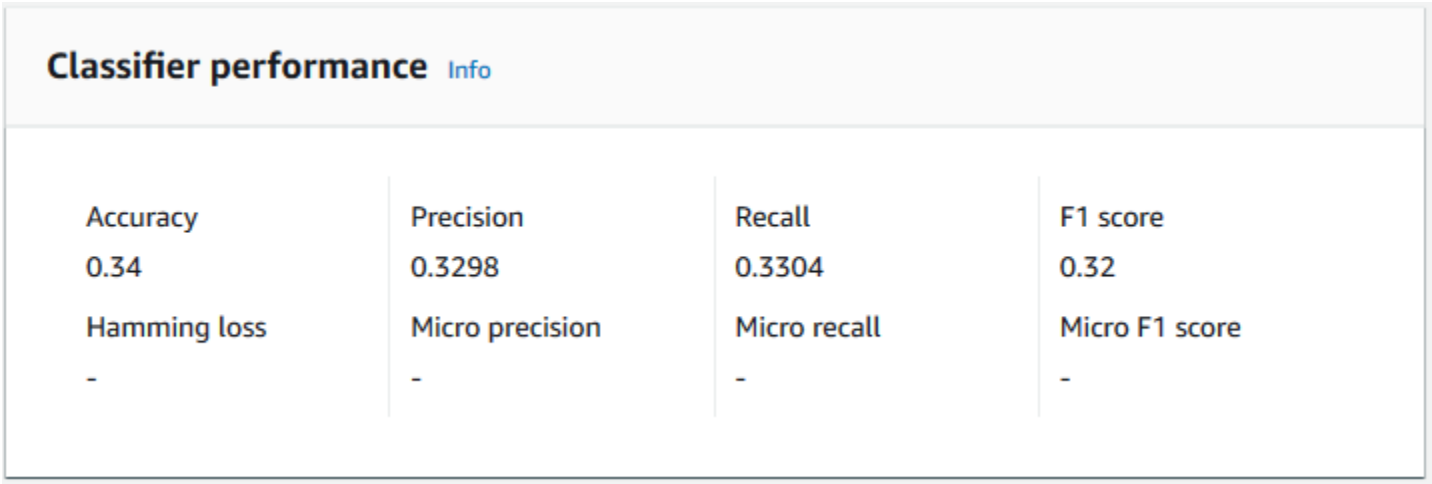
メトリクス

Amazon Comprehend は以下のメトリクスをサポートしています。

トピック

- 正解率
- 精度 (マクロ精度)
- リコール (マクロリコール)
- F1 スコア (マクロ F1 スコア)
- ハミングロス
- マイクロ精度
- マイクロリコール
- マイクロ F1 スコア

分類子のメトリクスを表示するには、コンソールの分類子の詳細ページを開きます。



正解率

精度は、テストデータに含まれるラベルのうち、モデルが正確に予測したラベルの割合を示します。精度を計算するには、テスト文書内の正確に予測されたラベル数を、テスト文書内のラベルの総数で割ります。

例

実際のラベル	予測ラベル	正確/不正確
1	1	正確
0	1	不正確
2	3	不正確
3	3	正確
2	2	正確
1	1	正確
3	3	正確

精度は、正確な予測数を全体のテストサンプル数で割った値 = 5/7 = 0.714、つまり 71.4% です。

精度 (マクロ精度)

精度は、テストデータにおける分類結果の有用性の尺度です。正確に分類された文書の数を、そのクラスの分類総数で割ったものとして定義されます。精度が高いということは、分類子が無関係な結果よりもかなり関連性の高い結果を返したということです。

`Precision`この指標はマクロ精度とも呼ばれます。

次の例は、テストセットの精度結果を示しています。

ラベル	サンプルサイズ	ラベル精度
Label_1	400	0.75
Label_2	300	0.80

ラベル	サンプルサイズ	ラベル精度
Label_3	30000	0.90
Label_4	20	0.50
Label_5	10	0.40

したがって、モデルの精度 (マクロ精度) 指標は次のようになります。

```
Macro Precision = (0.75 + 0.80 + 0.90 + 0.50 + 0.40)/5 = 0.67
```

リコール (マクロリコール)

これは、テキストに含まれる正しいカテゴリのうち、モデルが予測できる正しいカテゴリの割合を示します。この指標は、使用可能なすべてのラベルのリコールスコアを平均して算出されます。リコールは、分類子の結果がテストデータにどの程度完全であるかを示す尺度です。

リコール率が高いということは、分類子が関連する結果のほとんどを返したことを意味します。

Recall 指標はマクロリコールとも呼ばれます。

次の例は、テストセットのリコール結果を示しています。

ラベル	サンプルサイズ	ラベルリコール
Label_1	400	0.70
Label_2	300	0.70
Label_3	30000	0.98
Label_4	20	0.80
Label_5	10	0.10

したがって、モデルのリコール (マクロリコール) 指標は次のようになります。

```
Macro Recall = (0.70 + 0.70 + 0.98 + 0.80 + 0.10)/5 = 0.656
```

F1 スコア (マクロ F1 スコア)

F1 スコアは Precision と Recall の値から算出されます。分類子全体の精度を測定します。最高スコアは 1 で、最低スコアは 0 です。

Amazon Comprehend はマクロ F1 スコアを計算します。これはラベルの F1 スコアの加重されていない平均です。次のテストセットを例に取ります。

ラベル	サンプルサイズ	ラベル F1 スコア
Label_1	400	0.724
Label_2	300	0.824
Label_3	30000	0.94
Label_4	20	0.62
Label_5	10	0.16

モデルの F1 スコア (マクロ F1 スコア) は次のように計算されます。

```
Macro F1 Score = (0.724 + 0.824 + 0.94 + 0.62 + 0.16)/5 = 0.6536
```

ハミングロス

予測違いのラベルの割合。ラベル総数に対する不正確なラベルの割合とも見なされます。0 に近いほど良いスコアです。

マイクロ精度

オリジナル：

精度指標と似ていますが、マイクロ精度はすべての精度スコアを足した総合スコアに基づく点が異なります。

マイクロリコール

リコール指標と似ていますが、マイクロリコールはすべてのリコールスコアを足した総合スコアに基づく点が異なります。

マイクロ F1 スコア

Micro F1 スコアは、マイクロ精度指標とマイクロリコール指標を組み合わせたものです。

カスタム分類子のパフォーマンス向上

指標から分類ジョブ中にカスタム分類子がどのように機能するかについての洞察が得られます。指標が低い場合、分類モデルは使用事例に合わない可能性があります。分類子パフォーマンスを改善するには、幾つか方法があります。

1. 調教データには、カテゴリを明確に分ける具体的な例を指定してください。たとえば、カテゴリを表すために固有の単語や文を使用する文書を用意します。
2. 調教データで、表示頻度が低いラベルについては、さらにデータを追加します。
3. カテゴリ内の偏りを減らすようにしてください。文書の数がデータ内の最大ラベルに最小ラベルの 10 倍以上ある場合は、最小ラベルの文書数を増やしてみてください。表現率の高いクラスと低いクラスのスキュー比を最大で 10:1 に減らしてください。また、表現数の多いクラスから入力文書を削除してみることもできます。

リアルタイム分析の実行

カスタム分類子を調教した後で、リアルタイム分析を使用して文書を分類できます。リアルタイム分析では、1 つの文書を入力として受け取り、結果を同期的に返します。カスタム分類では、さまざまな文書タイプをリアルタイム分析の入力として受け入れます。詳細については、「リアルタイムカスタム分析用の入力」を参照してください。

画像ファイルまたはスキャンされた PDF ドキュメントを分析する予定の場合、IAM ポリシーは 2 つの Amazon Textract API メソッド (DetectDocumentText および) を使用するアクセス許可を付与する必要があります AnalyzeDocument。Amazon Comprehend は、テキスト抽出中にこれらのメソッドを呼び出します。ポリシーの例については、「 ドキュメント分析アクションを実行するために必要なアクセス許可」を参照してください。

カスタム分類モデルを使用してリアルタイム分析を実行するには、エンドポイントを作成する必要があります。

トピック

- カスタム分類のリアルタイム分析 (コンソール)
- カスタム分類のリアルタイム分析 (API)
- リアルタイム分析の出力

カスタム分類のリアルタイム分析 (コンソール)

Amazon Comprehend コンソールを使用して、カスタム分類モデルを使用したリアルタイム分析を実行できます。

リアルタイム分析を実行するエンドポイントを作成します。エンドポイントには、リアルタイム推論にカスタムモデルを使用できるようにする管理対象リソースが含まれます。

エンドポイントのスループットのプロビジョニングとそれに関連するコストについては、「Amazon Comprehend エンドポイントの使用法」を参照してください。

トピック

- カスタム分類用のエンドポイントの作成
- リアルタイムカスタム分類の実行

カスタム分類用のエンドポイントの作成

エンドポイントを作成するには (コンソール)

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/comprehend/ で Amazon Comprehend コンソールを開きます。
2. 左側のメニューから [エンドポイント] を選択し、[エンドポイントの作成] ボタンを選択します。「エンドポイントの作成」画面が開きます。
3. エンドポイントに名前を付けます。名前は、自分のアカウント内と現在のリージョンで一意でなければなりません。
4. 新しいエンドポイントをアタッチするカスタムモデルを選択します。ドロップダウンから、モデル名で検索できます。

   > **Note**
   >
   > エンドポイントをモデルにアタッチする前に、モデルを作成する必要があります。まだモデルがない場合は、「調教分類モデル」を参照してください。

5. (オプション) エンドポイントにタグを追加するには、[タグ] にキーと値のペアを入力し、[タグを追加] を選択します。エンドポイントを作成する前にこのペアを削除するには、[タグを削除] を選択します

6. エンドポイントに割り当てる推論単位 (IU) の数を入力します。各単位は、1 秒あたり最大 2 つの文書に対して 100 文字/秒のスループットを表します。最大スループットの詳細については、「Amazon Comprehend エンドポイントの使用法」を参照してください。
7. (オプション) 新しいエンドポイントを作成する場合は、IU Estimator を使用することもできます。スループットや 1 秒あたりの分析文字数によっては、必要な推論単位がわかりにくい場合があります。このオプションの手順は、リクエストする IU の数を決定するのに役立ちます。
8. 購入概要から、時間単位、日単位、月単位の推定エンドポイントコストを確認します。
9. 起動から削除までの間、エンドポイントの料金が発生することを了解している場合は、このチェックボックスを選択してください。
10. [エンドポイントの作成] を選択します。

リアルタイムカスタム分類の実行

エンドポイントを作成したなら、カスタムモデルを使用してリアルタイム分析を実行できます。コンソールからリアルタイム分析を実行するには、2 つの方法があります。以下に示すように、テキストを入力するか、ファイルをアップロードできます。

カスタムモデル (コンソール) を使用してリアルタイム分析を実行するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/comprehend/ で Amazon Comprehend コンソールを開きます。
2. 左側のメニューで、[リアルタイム分析] を選択します。
3. [入力タイプ] で、[分析タイプ] に [カスタム] を選択します。
4. [カスタムモデルタイプ] で [カスタム分類] を選択します。
5. [エンドポイント] で、使用するエンドポイントを選択します。このエンドポイントは特定のカスタムモデルにリンクしています。
6. 分析用の入力データを指定するには、テキストを入力するか、ファイルをアップロードします。
   - テキストを入力するには
     a. 入力テキストを選択します。
     b. 分析するテキストを入力します。
   - ファイルをアップロードするには
     a. [ファイルをアップロード] を選択し、アップロードするファイル名を入力します。

b. (オプション) 高度な読み取りアクションでは、デフォルトのテキスト抽出アクションをオーバーライドできます。詳細については、「テキスト抽出オプションの設定」を参照してください。

最良の結果を得るには、入力タイプを分類子モデルのタイプと一致させてください。ネイティブ文書をプレーンテキストモデルに送信したり、プレーンテキストをネイティブ文書モデルに送信したりすると、コンソールに警告が表示されます。詳細については、「調教分類モデル」を参照してください。

7. [分析] を選択します。Amazon Comprehend は、カスタムモデルを使用して入力データを分析します。Amazon Comprehend には、検出されたクラスと各クラスの信頼性評価が表示されます。

カスタム分類のリアルタイム分析 (API)

Amazon Comprehend API を使用して、カスタム分類モデルでリアルタイム分析を実行できます。まずリアルタイム分析を実行するエンドポイントを作成します。エンドポイントを作成したら、リアルタイム分類を実行します。

このセクションの例では、UNIX、Linux、macOS 用のコマンド形式を使用しています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

エンドポイントのスループットのプロビジョニングとそれに関連するコストについては、「Amazon Comprehend エンドポイントの使用法」を参照してください。

トピック

- カスタム分類用のエンドポイントの作成
- リアルタイムカスタム分類の実行

カスタム分類用のエンドポイントの作成

次の例は、 を使用した CreateEndpoint API オペレーションを示しています AWS CLI。

```
aws comprehend create-endpoint \
    --desired-inference-units number of inference units \
    --endpoint-name endpoint name \
    --model-arn arn:aws:comprehend:region:account-id:model/example \
    --tags Key=My1stTag,Value=Value1
```

Amazon Comprehend は次のように応答します。

```
{
   "EndpointArn": "Arn"
}
```

リアルタイムカスタム分類の実行

カスタム分類モデルのエンドポイントを作成したら、エンドポイントを使用して ClassifyDocument API オペレーションを実行します。text または bytes パラメータを使用してテキストを提供できます。その他の種類の入力には、bytes パラメータを利用します。

画像ファイルや PDF ファイルの場合は、DocumentReaderConfig パラメーターを使用してデフォルトのテキスト抽出アクションをオーバーライドできます。詳細については、「テキスト抽出オプションの設定」を参照してください。

最良の結果を得るには、入力タイプを分類子モデルのタイプと一致させてください。ネイティブ文書をプレーンテキストモデルに送信したり、プレーンテキストファイルをネイティブ文書モデルに送信したりすると、API 応答に警告が含まれます。詳細については、「調教分類モデル」を参照してください。

の使用 AWS Command Line Interface

以下の例は、classify-document CLI コマンドの使用方法を示しています。

を使用してテキストを分類する AWS CLI

次の例では、テキストブロックに対してリアルタイム分類を実行します。

```
aws comprehend classify-document \
     --endpoint-arn arn:aws:comprehend:region:account-id:endpoint/endpoint name \
     --text 'From the Tuesday, April 16th, 1912 edition of The Guardian newspaper: The
 maiden voyage of the White Star liner Titanic,
     the largest ship ever launched ended in disaster. The Titanic started her trip
 from Southampton for New York on Wednesday. Late
     on Sunday night she struck an iceberg off the Grand Banks of Newfoundland. By
 wireless telegraphy she sent out signals of distress,
     and several liners were near enough to catch and respond to the call.'
```

Amazon Comprehend は次のように応答します。

```
{
    "Classes": [
       {
          "Name": "string",
          "Score": 0.9793661236763
       }
    ]
 }
```

を使用して半構造化ドキュメントを分類する AWS CLI

PDF、Word、画像ファイルのカスタム分類を分析するには、bytes パラメーターに入力ファイルを指定して classify-document コマンドを実行します。

次の例では、画像を入力ファイルとして使用します。fileb オプションを使用して画像ファイルのバイトを Base-64 でエンコードします。詳細については、「 ユーザーガイド」の「バイナリラージオブジェクト AWS Command Line Interface 」を参照してください。

この例では、テキスト抽出オプションを設定するために config.json という名前の JSON ファイルも渡しています。

```
$ aws comprehend classify-document \
> --endpoint-arn arn \
> --language-code en \
> --bytes fileb://image1.jpg   \
> --document-reader-config file://config.json
```

config.json ファイルには次のコンテンツが含まれます。

```
 {
    "DocumentReadMode": "FORCE_DOCUMENT_READ_ACTION",
    "DocumentReadAction": "TEXTRACT_DETECT_DOCUMENT_TEXT"
 }
```

Amazon Comprehend は次のように応答します。

```
{
    "Classes": [
```

```
        {
           "Name": "string",
           "Score": 0.9793661236763
        }
     ]
 }
```

詳細については、Amazon ComprehendClassifyDocument」の「」を参照してください。

リアルタイム分析の出力

テキスト入力の出力

テキスト入力の場合、出力には分類子分析によって特定されたクラスまたはラベルのリストが含まれます。次の例は、2 つのクラスを持つリストを示しています。

```
"Classes": [
  {
     "Name": "abc",
     "Score": 0.2757999897003174,
     "Page": 1
  },
  {
    "Name": "xyz",
    "Score": 0.2721000015735626,
    "Page": 1
  }
]
```

半構造化入力の出力

半構造化入力ドキュメントまたはテキストファイルの場合、出力には以下の追加フィールドが含まれる場合があります。

- DocumentMetadata – ドキュメントに関する抽出情報。メタデータには、ドキュメント内のページのリストと、各ページから抽出された文字数が含まれます。リクエストに Byte パラメータがあると、このフィールドが応答に含まれます。
- DocumentType – 入力ドキュメントの各ページのドキュメントタイプ。リクエストに Byte パラメータがあると、このフィールドが応答に含まれます。

- エラー — 入力文書の処理中にシステムが検出したページレベルのエラー。エラーが検出されなかった場合、このフィールドは空です。
- 警告 — 入力文書の処理中に警告が検出されました。入力文書タイプと指定したエンドポイントに関連付けられているモデルタイプが一致しない場合、応答には警告が含まれます。システムが警告を生成しなかった場合、このフィールドは空になります。

これらの出力フィールドの詳細については、Amazon ComprehendClassifyDocument」の「」を参照してください。

次の例は、1 ページのネイティブ PDF 入力文書の出力例です。

```
{
  "Classes": [
      {
          "Name": "123",
          "Score": 0.39570000767707825,
          "Page": 1
      },
      {
          "Name": "abc",
          "Score": 0.2757999897003174,
          "Page": 1
      },
      {
          "Name": "xyz",
          "Score": 0.2721000015735626,
          "Page": 1
      }
  ],
  "DocumentMetadata": {
      "Pages": 1,
      "ExtractedCharacters": [
          {
              "Page": 1,
              "Count": 2013
          }
      ]
  },
  "DocumentType": [
      {
          "Page": 1,
          "Type": "NATIVE_PDF"
```

```
      }
  ]
}
```

非同期ジョブの実行

カスタム分類子を調教したなら、非同期ジョブを使用して大きな文書や複数の文書を 1 つのバッチで分析できます。

カスタム分類では、さまざまな入力文書タイプを受け入れます。詳細については、「非同期カスタム分析の入力 」を参照してください。

画像ファイルまたはスキャンされた PDF ドキュメントを分析する予定の場合、IAM ポリシーは 2 つの Amazon Textract API メソッド (DetectDocumentText および) を使用するアクセス許可を付与する必要があります AnalyzeDocument。Amazon Comprehend は、テキスト抽出中にこれらのメソッドを呼び出します。ポリシーの例については、「 ドキュメント分析アクションを実行するために必要なアクセス許可」を参照してください。

プレーンテキストモデルを使用して半構造化文書 (画像、PDF、または Docx ファイル) を分類する場合は、`one document per file` 入力形式を使用します。また、StartDocumentClassificationJobリクエストに `DocumentReaderConfig`パラメータを含めます。

トピック

- 非同期分析用のファイル形式
- カスタム分類の分析ジョブ (コンソール)
- カスタム分類の分析ジョブ (API)
- 非同期分析ジョブの出力

非同期分析用のファイル形式

モデルを使用して非同期解析を実行する場合、入力文書の形式には `One document per line` か `one document per file` を選択できます。次の表に示すように、使用する形式は分析する文書のタイプによって異なります。

説明	[形式]
入力には複数のファイルが含まれます。各ファイルには 1 つの入力ドキュメントが含まれます。この形式は、新聞記事や科学論文など、サイズの大きい文書の集団に最適です。 また、ネイティブ文書分類子を使用する半構造化文書 (画像、PDF、または Docx ファイル) にもこの形式を使用してください。	ファイルごとに 1 文書
入力は 1 つまたは複数のファイルです。ファイル内の各行は個別の入力文書です。この形式は、テキストメッセージやソーシャルメディアへの投稿など、短い文書に最適です。	1 行に 1 文書

ファイルごとに 1 文書

`one document per file` 形式では、各ファイルが 1 つの入力文書を表します。

1 行に 1 文書

`One document per line` 形式では、各文書は別々の行に配置され、ヘッダーは使いません。ラベルは各行に含まれません (文書のラベルがまだわからないため)。ファイルの各行 (個々の文書の末尾) は、改行 (LF、\n) キャリッジリターン (CR、\r) またはその両方 (CRLF、\r\n) でなければなりません。UTF-8 の行区切り文字 (u+2028) を使用して行を終了してはなりません。

以下の例は、 入力ファイルの形式を示しています。

```
Text of document 1 \n
Text of document 2 \n
Text of document 3 \n
Text of document 4 \n
```

どちらの形式でも、テキストファイルには UTF-8 エンコードを使用します。ファイルを作成したなら、入力データに使用している S3 バケットにファイルを配置します。

分類ジョブを開始するときに、この Amazon S3 ロケーションを入力データとして指定します。URI は、呼び出す API エンドポイントと同じリージョンである必要があります。URI は 1 つのファイル

(「1 行に 1 つの文書」を使用する場合など) を指すことも、データファイルのコレクションのプレフィックスにすることもできます。

たとえば、URI `S3://bucketName/prefix` を使用する場合、プレフィックスが単一ファイルの場合、Amazon Comprehend はそのファイルを入力として使用します。複数のファイルがプレフィックスで始まる場合、Amazon Comprehend はそれらすべてを入力として使用します。

Amazon Comprehend に文書コレクションおよび出力ファイルが含まれる S3 バケットへのアクセス許可を付与します。詳細については、「バッチ操作に必要なロールベースのアクセス許可」を参照してください。

カスタム分類の分析ジョブ (コンソール)

カスタム文書分類子を作成して調教したなら、コンソールを使用してモデルでカスタム分類ジョブを実行できます。

カスタム分類ジョブを作成するには (コンソール)

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/comprehend/ で Amazon Comprehend コンソールを開きます。
2. 左側のメニューから、[分析ジョブ] を選択し、[ジョブの作成] を選択します。
3. 分類ジョブに名前を付けます。この名前は、自分のアカウントと現在のリージョンで一意でなければなりません。
4. [分析タイプ] で [カスタム分類] を選択します。
5. 「分類子の選択」から、使用するカスタム分類子を選択します。
6. (オプション) Amazon Comprehend がジョブの処理中に使用するデータの暗号化を選択する場合は、[ジョブの暗号化] を選択します。次に、現在のアカウントに関連付けられた KMS キーを使用するか、別のアカウントの KMS キーを使用するかを選択します。
   - 現在のアカウントに関連付けられているキーを使用している場合は、KMS キー ID のキー ID を選択します。
   - 別のアカウントに関連付けられているキーを使用している場合は、KMS キー ARN の下にキー ID の ARN を入力します。

> **Note**
>
> KMS キーの作成と使用や関連する暗号化の詳細については、「キー管理サービス (KMS)」を参照してください。

7. [入力データ] で、入力文書を含む Amazon S3 バケットの場所を入力するか、[S3 を参照] を選択してその場所に移動します。このバケットは、呼び出している API と同じリージョン内になければなりません。分類ジョブのアクセス許可に使用する IAM ロールには、S3 バケットに対する読み取り許可が必要です。

   モデルの調教を最高レベルの精度で行うには、入力のタイプを分類子のモデルタイプと一致させる必要があります。ネイティブ文書をプレーンテキストモデルに送信したり、プレーンテキスト文書をネイティブ文書モデルに送信したりすると、分類器ジョブは警告を返します。詳細については、「調教分類モデル」を参照してください。

8. (オプション) 入力形式では、入力ドキュメントの形式を選択できます。形式は、ファイルごとに 1 文書にすることも、1 つのファイルの 1 行に 1 文書にすることもできます。1 行に 1 つの文書が適用されるのはテキスト文書だけです。

9. (オプション) 文書読み取りモードでは、デフォルトのテキスト抽出アクションをオーバーライドできます。詳細については、「テキスト抽出オプションの設定」を参照してください。

10. [出力データ] で、Amazon Comprehend がジョブの出力データを書き込む Amazon S3 バケットの場所を入力するか、[S3 を参照] を選択してその場所に移動します。このバケットは、呼び出している API と同じリージョン内になければなりません。分類ジョブのアクセス許可に使用する IAM ロールには、S3 バケットに対する書き込み許可が必要です。

11. (オプション) ジョブの出力結果を暗号化する場合は、[暗号化] を選択します。次に、現在のアカウントに関連付けられた KMS キーを使用するか、別のアカウントの KMS キーを使用するかを選択します。

    - 現在のアカウントに関連付けられているキーを使用している場合は、KMS キー ID のキーエイリアスまたは ID を選択します。
    - 別のアカウントに関連付けられているキーを使用している場合は、KMS キー ID の下にキーエイリアス の ARN または ID を入力します。

12. (オプション) VPC から Amazon Comprehend にリソースを起動するには、VPC の下に VPC ID を入力するか、ドロップダウンリストから ID を選択します。

1. [サブネット] でサブネットを選択します。最初のサブネットを選択すると、追加のサブネットを選択できます。
2. セキュリティグループを指定した場合は、[セキュリティグループ] で、使用するセキュリティグループを選択します。最初のセキュリティグループを選択すると、追加のセキュリティグループを選択できます。

> **Note**
>
> 分類ジョブで VPC を使用する場合、[作成] と [開始] 操作に使用する `DataAccessRole` には、出力バケットへの VPC アクセス権限を与える必要があります。

13. [ジョブの作成] を選択して文書分類ジョブを作成します。

カスタム分類の分析ジョブ (API)

カスタム文書分類子を作成して調教したなら、その分類子を使用して分析ジョブを実行できます。

StartDocumentClassificationJob オペレーションを使用して、ラベルなしドキュメントの分類を開始します。入力文書が入った S3 バケット、出力文書の S3 バケット、使用する分類子を指定します。

モデルの調教を最高レベルの精度で行うには、入力のタイプを分類子のモデルタイプと一致させる必要があります。ネイティブ文書をプレーンテキストモデルに送信したり、プレーンテキスト文書をネイティブ文書モデルに送信したりすると、分類器ジョブは警告を返します。詳細については、「調教分類モデル」を参照してください。

StartDocumentClassificationJob は非同期です。ジョブを開始したら、DescribeDocumentClassificationJobオペレーションを使用して進行状況をモニタリングします。応答の `Status` フィールドに `COMPLETED` が表示されたなら、指定した場所にある出力にアクセスできます。

トピック

- の使用 AWS Command Line Interface
- AWS SDK for Java または SDK for Python の使用

の使用 AWS Command Line Interface

以下の例は、演算機能 StartDocumentClassificationJob、その他のカスタム分類子 API を AWS CLIで使用する方法を示しています。

次の例は、Unix、Linux、macOS 用のコマンド形式を使用しています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

演算機能 StartDocumentClassificationJob を使用してカスタム分類子を実行します。

```
aws comprehend start-document-classification-job \
     --region region \
     --document-classifier-arn arn:aws:comprehend:region:account number:document-
classifier/testDelete \
     --input-data-config S3Uri=s3://S3Bucket/docclass/file
 name,InputFormat=ONE_DOC_PER_LINE \
     --output-data-config S3Uri=s3://S3Bucket/output \
     --data-access-role-arn arn:aws:iam::account number:role/resource name
```

演算機能 DescribeDocumentClassificationJob を使用して、ジョブ ID を含んだカスタム分類子に関する情報を取得します。

```
aws comprehend describe-document-classification-job \
     --region region \
     --job-id job id
```

演算機能 ListDocumentClassificationJobs を使用して、アカウント内のすべてのカスタム分類ジョブを一覧表示します。

```
aws comprehend list-document-classification-jobs
     --region region
```

AWS SDK for Java または SDK for Python の使用

カスタム分類子を開始する方法の SDK の例については、「AWS SDK または CLI StartDocumentClassificationJobで を使用する」を参照してください。

非同期分析ジョブの出力

分析ジョブが完了すると、リクエストに指定した S3 バケットに結果が保存されます。

テキスト入力の出力

どちらの形式のテキスト入力文書 (マルチクラスまたはマルチラベル) でも、ジョブの出力は output.tar.gz という名前の 1 ファイルで構成されます。これは圧縮されたアーカイブファイルで、出力を含むテキストファイルが含まれています。

マルチクラス出力

マルチクラスモードで調教した分類子を使用すると、結果に classes が表示されます。classes のそれぞれは、分類子を調教する際に一連のカテゴリを作成するために使用するクラスです。

これらの出力フィールドの詳細については、Amazon ComprehendClassifyDocument」の「」を参照してください。

以下の例では、相互に排他的なクラスを使用しています。

```
DOCUMENTARY
SCIENCE_FICTION
ROMANTIC_COMEDY
SERIOUS_DRAMA
OTHER
```

入力データ形式が 1 行に 1 文書の場合、出力ファイルには入力の 1 行につき 1 行が含まれます。各行には、ファイル名、入力行の 0 から始まる行番号、文書内の 1 つまたは複数のクラスが含まれます。最後は、Amazon Comprehend が個々のインスタンスを正しく分類したという確信で終わります。

例:

```
{"File": "file1.txt", "Line": "0", "Classes": [{"Name": "Documentary", "Score":
 0.8642}, {"Name": "Other", "Score": 0.0381}, {"Name": "Serious_Drama", "Score":
 0.0372}]}
{"File": "file1.txt", "Line": "1", "Classes": [{"Name": "Science_Fiction", "Score":
 0.5}, {"Name": "Science_Fiction", "Score": 0.0381}, {"Name": "Science_Fiction",
 "Score": 0.0372}]}
{"File": "file2.txt", "Line": "2", "Classes": [{"Name": "Documentary", "Score": 0.1},
 {"Name": "Documentary", "Score": 0.0381}, {"Name": "Documentary", "Score": 0.0372}]}
{"File": "file2.txt", "Line": "3", "Classes": [{"Name": "Serious_Drama", "Score":
 0.3141}, {"Name": "Other", "Score": 0.0381}, {"Name": "Other", "Score": 0.0372}]}
```

入力データ形式が 1 ファイルにつき 1 文書の場合、出力ファイルには文書ごとに 1 行ずつ含まれます。各行には、ファイルの名前と文書内のクラス (1 つまたは複数) があります。最後は、Amazon Comprehend が個々のインスタンスを正しく分類したという確信で終わります。

例:

```
{"File": "file0.txt", "Classes": [{"Name": "Documentary", "Score": 0.8642}, {"Name":
 "Other", "Score": 0.0381}, {"Name": "Serious_Drama", "Score": 0.0372}]}
{"File": "file1.txt", "Classes": [{"Name": "Science_Fiction", "Score": 0.5}, {"Name":
 "Science_Fiction", "Score": 0.0381}, {"Name": "Science_Fiction", "Score": 0.0372}]}
{"File": "file2.txt", "Classes": [{"Name": "Documentary", "Score": 0.1}, {"Name":
 "Documentary", "Score": 0.0381}, {"Name": "Domentary", "Score": 0.0372}]}
{"File": "file3.txt", "Classes": [{"Name": "Serious_Drama", "Score": 0.3141}, {"Name":
 "Other", "Score": 0.0381}, {"Name": "Other", "Score": 0.0372}]}
```

マルチラベル出力

マルチラベルモードで調教した分類子を使用すると、結果に `labels` が表示されます。`labels` のそれぞれは、分類子を調教する際に一連のカテゴリを作成するために使用するラベルです。

以下の例では、これらの固有のラベルを使用しています。

```
SCIENCE_FICTION
ACTION
DRAMA
COMEDY
ROMANCE
```

入力データ形式が 1 行に 1 文書の場合、出力ファイルには入力の 1 行につき 1 行が含まれます。各行には、ファイル名、入力行の 0 から始まる行番号、文書内の 1 つまたは複数のクラスが含まれます。最後は、Amazon Comprehend が個々のインスタンスを正しく分類したという確信で終わります。

例:

```
{"File": "file1.txt", "Line": "0", "Labels": [{"Name": "Action", "Score": 0.8642},
 {"Name": "Drama", "Score": 0.650}, {"Name": "Science Fiction", "Score": 0.0372}]}
{"File": "file1.txt", "Line": "1", "Labels": [{"Name": "Comedy", "Score": 0.5},
 {"Name": "Action", "Score": 0.0381}, {"Name": "Drama", "Score": 0.0372}]}
{"File": "file1.txt", "Line": "2", "Labels": [{"Name": "Action", "Score": 0.9934},
 {"Name": "Drama", "Score": 0.0381}, {"Name": "Action", "Score": 0.0372}]}
```

```
{"File": "file1.txt", "Line": "3", "Labels": [{"Name": "Romance", "Score": 0.9845},
 {"Name": "Comedy", "Score": 0.8756}, {"Name": "Drama", "Score": 0.7723}, {"Name":
 "Science_Fiction", "Score": 0.6157}]}
```

入力データ形式が 1 ファイルにつき 1 文書の場合、出力ファイルには文書ごとに 1 行ずつ含まれます。各行には、ファイルの名前と文書内のクラス (1 つまたは複数) があります。最後は、Amazon Comprehend が個々のインスタンスを正しく分類したという確信で終わります。

例:

```
{"File": "file0.txt", "Labels": [{"Name": "Action", "Score": 0.8642}, {"Name": "Drama",
 "Score": 0.650}, {"Name": "Science Fiction", "Score": 0.0372}]}
{"File": "file1.txt", "Labels": [{"Name": "Comedy", "Score": 0.5}, {"Name": "Action",
 "Score": 0.0381}, {"Name": "Drama", "Score": 0.0372}]}
{"File": "file2.txt", "Labels": [{"Name": "Action", "Score": 0.9934}, {"Name": "Drama",
 "Score": 0.0381}, {"Name": "Action", "Score": 0.0372}]}
{"File": "file3.txt", "Labels": [{"Name": "Romance", "Score": 0.9845}, {"Name":
 "Comedy", "Score": 0.8756}, {"Name": "Drama", "Score": 0.7723}, {"Name":
 "Science_Fiction", "Score": 0.6157}]}
```

半構造化された入力文書の出力

半構造化入力ドキュメントの場合、出力には以下の追加フィールドが含まれる場合があります。

- DocumentMetadata – ドキュメントに関する抽出情報。メタデータには、ドキュメント内のページのリストと、各ページから抽出された文字数が含まれます。リクエストに Byte パラメータがあると、このフィールドが応答に含まれます。
- DocumentType – 入力ドキュメントの各ページのドキュメントタイプ。リクエストに Byte パラメータがあると、このフィールドが応答に含まれます。
- エラー — 入力文書の処理中にシステムが検出したページレベルのエラー。エラーが検出されなかった場合、このフィールドは空です。

これらの出力フィールドの詳細については、Amazon ComprehendClassifyDocument」の「」を参照してください。

次の例は、スキャンした 2 ページのPDF ファイルの出力を示しています。

```
[{ #First page output
```

```
    "Classes": [
        {
            "Name": "__label__2 ",
            "Score": 0.9993996620178223
        },
        {
            "Name": "__label__3 ",
            "Score": 0.0004330444789957255
        }
    ],
    "DocumentMetadata": {
        "PageNumber": 1,
        "Pages": 2
    },
    "DocumentType": "ScannedPDF",
    "File": "file.pdf",
    "Version": "VERSION_NUMBER"
},
#Second page output
{
    "Classes": [
        {
            "Name": "__label__2 ",
            "Score": 0.9993996620178223
        },
        {
            "Name": "__label__3 ",
            "Score": 0.0004330444789957255
        }
    ],
    "DocumentMetadata": {
        "PageNumber": 2,
        "Pages": 2
    },
    "DocumentType": "ScannedPDF",
    "File": "file.pdf",
    "Version": "VERSION_NUMBER"
}]
```

カスタムエンティティ認識

カスタムエンティティレコグナイザーは、あらかじめ設定された汎用エンティティタイプ にはない特定の新しいエンティティタイプを識別できるようにすることで、Amazon Comprehend の機能を拡張します。つまり、ドキュメントを分析し、製品コードやビジネス固有のエンティティなど、特定のニーズに合ったエンティティを抽出できます。

正確なカスタムエンティティレコグナイザーを自分で構築する場合、手作業で注釈を付けた大量のトレーニングドキュメントを準備したり、モデルトレーニングに適したアルゴリズムとパラメーターを選択したりする必要があり、複雑なプロセスになる場合があります。Amazon Comprehend は、カスタムエンティティレコグナイザーモデルを作成するための自動アノテーションとモデル開発を提供することで、複雑さを軽減します。

カスタムエンティティレコグナイザーモデルを作成する方が、文字列マッチングや正規表現を使用してドキュメントからエンティティを抽出するよりも効果的です。たとえば、ドキュメント内の ENGINEER 名を抽出する場合、考えられるすべての名前を列挙することは困難です。さらに、コンテキストがないと、エンジニア名とアナリスト名を区別するのが難しくなります。カスタムエンティティレコグナイザーモデルは、それらの名前が出現する可能性のあるコンテキストを学習できます。さらに、文字列の照合では、タイプミスのあるエンティティや新しい命名規則に従っているエンティティは検出されませんが、カスタムモデルを使用すると検出できます。

カスタムモデルを作成するには、次の 2 つのオプションがあります。

1. 注釈 — モデルトレーニング用の注釈付きエンティティを含むデータセットを提供します。
2. エンティティリスト (プレーンテキストのみ) — エンティティとそのタイプラベル (PRODUCT_CODES や、モデルのトレーニングに使用するエンティティを含む注釈なしドキュメントのセットなど) を提供します。

注釈付きの PDF ファイルを使用してカスタムエンティティレコグナイザーを作成すると、そのレコグナイザーをさまざまな入力ファイル形式 (プレーンテキスト、画像ファイル (JPG、PNG、TIFF)、PDF ファイル、Word ドキュメント) で使用できます。前処理やドキュメントの統合は不要です。Amazon Comprehend は、画像ファイルや Word ドキュメントの注釈をサポートしていません。

> **Note**
>
> 注釈付き PDF ファイルを使用するカスタムエンティティレコグナイザーは、英語のドキュメントのみをサポートします。

1 つのモデルは最大 25 のカスタムエンティティで同時にトレーニングできます。詳細については、ガイドラインとクォータページをご覧ください。

モデルをトレーニングしたら、そのモデルをリアルタイムのエンティティ検出やエンティティ検出ジョブに使用できます。

トピック

- エンティティレコグナイザーのトレーニングデータの準備
- カスタムエンティティレコグナイザーモデルのトレーニング
- リアルタイムのカスタムレコグナイザー分析を実行する
- カスタムエンティティ認識の分析ジョブの実行

エンティティレコグナイザーのトレーニングデータの準備

カスタムエンティティレコグナイザーモデルのトレーニングを成功させるには、モデルトレーナーへの入力として高品質データを提供することが重要です。適切なデータがないと、モデルはエンティティを正しく識別する方法を学習できません。

カスタムエンティティレコグナイザーモデルをトレーニングするために、Amazon Comprehend へのデータ提供方法として次のいずれかを選択します。

- エンティティリスト — Amazon Comprehend がトレーニングしてカスタムエンティティを識別できるように、特定のエンティティを一覧表示します。注: エンティティリストはプレーンテキストのドキュメントにのみ使用できます。
- 注釈 — Amazon Comprehend がエンティティとそのコンテキストの両方についてトレーニングできるように、多数のドキュメント内のエンティティの位置を示します。　画像ファイル、PDF、Word ドキュメントを分析するためのモデルを作成するには、PDF 注釈を使用してレコグナイザーをトレーニングする必要があります。

いずれの場合も、Amazon Comprehend はドキュメントの種類と、エンティティが発生するコンテキストを学習し、ドキュメントを分析するときに新しいエンティティを検出するように一般化できるレコグナイザーを構築します。

カスタムモデルを作成 (または新しいバージョンをトレーニング) するときに、テストデータセットを提供できます。　テストデータを提供しない場合、Amazon Comprehend は入力ドキュメントの 10% をモデルのテスト用に取っておきます。Amazon Comprehend は残りのドキュメントを使用してモデルをトレーニングします。

注釈トレーニングセット用のテストデータセットを提供する場合、テストデータには、作成リクエストで指定されたエンティティタイプごとに少なくとも 1 つの注釈が含まれている必要があります。

トピック

- 注釈の使用とエンティティリストの使用
- エンティティリスト (プレーンテキストのみ)
- 注釈

注釈の使用とエンティティリストの使用

注釈の作成はエンティティリストの作成よりも手間がかかりますが、生成されるモデルの精度は大幅に向上します。エンティティリストを使用する方が迅速で手間はかかりませんが、結果の精度や精度は低下します。　これは、注釈によって Amazon Comprehend がモデルをトレーニングする場合、より多くのコンテキストが提供されるためです。このコンテキストがないと、Amazon Comprehend がエンティティを識別する際の誤検出の数が多くなります。

注釈の使用にかかる高い費用やワークロードを避ける方が、業務上道理にかなう場合もあります。たとえば、John Johnson という名前は検索には重要ですが、それがまさにその個人であるかどうかは重要ではない。または、エンティティリストを使用する際の指標が、必要なレコグナイザー結果を得るのに十分優れている。　このような場合は、代わりにエンティティリストを使用する方が効果的です。

次の場合は、注釈モードを使用することをお勧めします。

- 画像ファイル、PDF、Word ドキュメントの推論を行う予定がある場合。このシナリオでは、注釈付き PDF ファイルを使用してモデルをトレーニングし、そのモデルを使用して画像ファイル、PDF、Word ドキュメントの推論ジョブを実行します。

- エンティティの意味があいまいで、コンテキストに依存する可能性がある場合。　たとえば、「Amazon」という言葉は、ブラジルの川を指す場合もあれば、オンライン小売業者の Amazon.com を指す場合もあります。「Amazon」などの企業体を識別するカスタムエンティティレコグナイザーを作成する場合は、エンティティリストの代わりに注釈を使用する必要があります。これは、コンテキストを使用してエンティティを検索できるためです。
- 注釈を取得するプロセスを設定することに慣れている場合 (これには多少の労力が必要となる場合があります)。

次のような場合にはエンティティリストを使用することをお勧めします。

- エンティティリストがすでにある場合や、エンティティの包括的なリストを比較的簡単に作成できる場合。エンティティリストを使用する場合、リストは完全であるか、少なくともトレーニング用に提供するドキュメントに含まれている可能性のある有効なエンティティの大半を網羅している必要があります。
- 初めて使用するユーザーには、通常、エンティティリストの使用をお勧めします。これは、注釈を作成するよりも手間がかからないためです。ただし、トレーニングされたモデルは、注釈を使用した場合ほど正確ではない可能性があることに注意する必要があります。

エンティティリスト (プレーンテキストのみ)

エンティティリストを使用してモデルをトレーニングするには、2 つの情報を指定します。1 つはエンティティ名とそれに対応するカスタムエンティティタイプのリスト、もう 1 つはエンティティが表示されると予想される注釈の付いていないドキュメントのコレクションです。

エンティティリストを指定すると、Amazon Comprehend はインテリジェントアルゴリズムを使用してドキュメント内のエンティティの出現を検出し、それを基にしてカスタムエンティティレコグナイザーモデルをトレーニングします。

エンティティリストの場合、エンティティリスト内のエンティティタイプごとに 25 件以上のエンティティマッチを指定します。

カスタムエンティティ認識用のエンティティリストには、以下の列を含むカンマ区切り値 (CSV) ファイルが必要です。

- テキスト — 添付の文書ドキュメントコーパスとまったく同じエントリ例のテキスト。

- タイプ — 顧客定義のエンティティタイプ。エンティティタイプは、MANAGER や SENIOR_MANAGER のように大文字で下線で区切られた文字列でなければなりません。モデルごとに、最大 25 のエンティティタイプをトレーニングできます。

ファイル documents.txt には 4 行あります。

```
Jo Brown is an engineer in the high tech industry.
John Doe has been a engineer for 14 years.
Emilio Johnson is a judge on the Washington Supreme Court.
Our latest new employee, Jane Smith, has been a manager in the industry for 4 years.
```

エンティティのリストを含む CSV ファイルには以下の行があります。

```
Text, Type
Jo Brown, ENGINEER
John Doe, ENGINEER
Jane Smith, MANAGER
```

> **Note**
>
> エンティティリストには Emilio Johnson のエントリは存在しません。このエントリには ENGINEER エンティティも MANAGER エンティティも含まれていないためです。

データファイルを作成する

エンティティリストファイルに問題が発生する可能性を最小限に抑えるため、エンティティリストは適切に構成された CSV ファイルに保存することが重要です。CSV ファイルを手動で構成するには、以下が満たされている必要があります。

- UTF-8 エンコーディングは、ほとんどの場合、デフォルトとして使用されている場合でも、明示的に指定する必要があります。
- 列名 (Type と Text) を含める必要があります。

潜在的な問題を避けるため、CSV 入力ファイルはプログラムで生成することを強くお勧めします。

次の例では、Python を使用して前述の注釈の CSV を生成します。

```
import csv
with open("./entitylist/entitylist.csv", "w", encoding="utf-8") as csv_file:
    csv_writer = csv.writer(csv_file)
    csv_writer.writerow(["Text", "Type"])
    csv_writer.writerow(["Jo Brown", " ENGINEER"])
    csv_writer.writerow(["John Doe", " ENGINEER"])
    csv_writer.writerow(["Jane Smith", " MANAGER"])
```

ベストプラクティス

エンティティリストを使用するときに最良の結果を得るには、次のような点を考慮する必要があります。

- リスト内のエンティティの順序はモデルトレーニングには影響しません。
- エンティティリストの項目は、注釈のないドキュメントコーパスで言及されているポジティブなエンティティの例の 80% ~ 100% を網羅したものを使用してください。
- 一般的な単語やフレーズを削除して、ドキュメントコーパス内の非エンティティと一致するエンティティの例は避けてください。不正確なマッチングは、たとえ少数でも最終的なモデルの精度に大きな影響を与えることがあります。たとえば、エンティティリスト内の the のような単語は、探しているエンティティーである可能性が低い多数の一致が発生し、精度に大きな影響を与えます。
- 入力データには重複がないようにしてください。サンプルが重複していると、テストセットが汚染され、トレーニングプロセス、モデルメトリクス、動作に悪影響を及ぼす可能性があります。
- 実際のユースケースにできるだけ類似したドキュメントを提供してください。生産システムに、玩具データや合成データを使用しないでください。入力データは、過剰適合を避け、基礎となるモデルが実際の例に基づいてより一般化しやすくなるように、できるだけ多様でなければなりません。
- エンティティリストでは大文字と小文字が区別され、正規表現は現在サポートされていません。ただし、トレーニング済みモデルでは、エンティティリストに記載されている大文字と小文字が完全に一致しなくても、エンティティを認識できることがよくあります。
- 別のエンティティのサブストリングであるエンティティ (「Smith」や「Jane Smith」など) がある場合は、エンティティリストに両方を指定してください。

その他の推奨事項は「カスタムエンティティレコグナイザーのパフォーマンスの向上」を確認してください。

注釈

注釈は、カスタムエンティティタイプをトレーニングドキュメント内の出現場所に関連付けることで、コンテキスト内のエンティティにラベル付けを行います。

ドキュメントと一緒に注釈を提出することで、モデルの精度を高めることができます。注釈を使用すると、探しているエンティティの場所を提供するだけでなく、探しているカスタムエンティティのより正確なコンテキストも提供できます。

たとえば、John Johnson という名前をエンティティタイプ JUDGEで検索する場合、注釈を指定すると、検索したい人が裁判官 であることをモデルが理解しやすくなる場合があります。コンテキストを使用できれば、Amazon Comprehend は 弁護士または証人である John Johnson という名前の人物は見つけません。注釈を付けない場合、Amazon Comprehend は独自の注釈を作成しますが、裁判官のみを含めるという点でそれほど効果的ではありません。独自の注釈を提供することで、より良い結果が得られ、カスタムエンティティを抽出する際にコンテキストをより有効に活用可能なモデルを生成できる場合があります。

トピック

- エンティティごとの最小注釈数
- 注釈のベストプラクティス
- プレーンテキストの注釈ファイル
- PDF アノテーションファイル
- PDF ファイルへの注釈

エンティティごとの最小注釈数

モデルのトレーニングに必要な入力ドキュメントと注釈の最小数は、注釈の種類によって異なります。

PDF アノテーション

画像ファイル、PDF、Word ドキュメントを分析するためのモデルを作成するには、PDF 注釈を使用してレコグナイザーをトレーニングする必要があります。PDF 注釈の場合は、エンティティごとに 250 個以上の入力ドキュメントと 100 個以上の注釈を用意してください。

テストデータセットを提供する場合、テストデータには、作成リクエストで指定されたエンティティタイプごとに少なくとも 1 つの注釈が含まれている必要があります。

プレーンテキストの注釈

テキストドキュメントを分析するためのモデルを作成するには、プレーンテキスト注釈を使用してレコグナイザーをトレーニングできます。

プレーンテキスト注釈の場合は、1 つのエンティティにつき 3 つ以上の注釈付きの入力ドキュメントと 25 個以上の注釈を用意します。入力する注釈の合計が 50 個未満の場合、Amazon Comprehend は入力ドキュメントの 10% 以上をモデルのテスト用に予約します (トレーニングリクエストでテストデータセットを提供した場合を除く)。ドキュメントコーパスの最小サイズは 5 KB です。

入力に含まれるトレーニングドキュメントの数が少ない場合、トレーニング入力データに含まれるエンティティの 1 つに言及するドキュメントが少なすぎるというエラーが発生する可能性があります。そのエンティティに言及した追加のドキュメントを添えて、ジョブを再度提出してください。

テストデータセットを提供する場合、テストデータには、作成リクエストで指定されたエンティティタイプごとに少なくとも 1 つの注釈が含まれている必要があります。

小さなデータセットでモデルをベンチマークする方法の例については、AWS ブログサイトで「Amazon Comprehend カスタムエンティティ認識の注釈制限引き下げ発表」を参照してください。

注釈のベストプラクティス

注釈を使用する際に最良の結果を得るには、次のような点を考慮する必要があります：

- データには注意して注釈を付け、エンティティについて言及するたびに必ず注釈を付けてください。注釈が不正確だと、結果が不良になる可能性があります。
- 入力データには、注釈を付ける PDF の複製のように、重複したものを含めないでください。サンプルが重複していると、テストセットが汚染され、トレーニングプロセス、モデルメトリクス、モデルの動作に悪影響を及ぼす可能性があります。
- すべてのドキュメントに注釈が付けられていること、および注釈のないドキュメントが正当なエンティティの欠如によるものであって、過失によるものではないことを確認してください。たとえば、「J Doe はエンジニアになってから 14 年になります」というドキュメントがある場合は、「John Doe」だけでなく「J Doe」にも注釈を付ける必要があります。そうしないと、モデルが混乱し、モデルが「J Doe」をエンジニアとして認識しなくなる可能性があります。これは同じドキュメント内でもドキュメント間でも一貫しています。
- 一般的に、注釈が多いほど良い結果が得られます。

- 「最小限」のドキュメントと注釈でモデルをトレーニングできますが、通常はデータを追加することでモデルが改善されます。モデルの精度を上げるために、注釈付きデータの量を 10% 増やすことをお勧めします。テストデータセットに対して推論を実行できます。このデータセットは変更されず、異なるバージョンのモデルでテストできます。その後、後続のモデルバージョンのメトリクスを比較できます。
- 実際のユースケースにできるだけ類似したドキュメントを提供してください。パターンが繰り返される合成データは避けるべきです。入力データは、過剰適合を避け、基礎となるモデルが実際の例に基づいてより一般化しやすくなるように、できるだけ多様でなければなりません。
- ドキュメントは単語数の点で多様であることが重要です。たとえば、トレーニングデータに含まれるすべてのドキュメントが短い場合、生成されるモデルが長いドキュメントに含まれるエンティティを予測するのが難しくなる可能性があります。
- カスタムエンティティを実際に検出するとき (推論時間)、使用予定のデータと同じデータをトレーニングに配分してみてください。たとえば、推論時にエンティティを含まないドキュメントを送付する予定であれば、これもトレーニングドキュメントセットの一部にしてください。

その他の提案については、「カスタムエンティティ認識機能のパフォーマンスの向上」を参照してください。

プレーンテキストの注釈ファイル

プレーンテキストの注釈の場合は、注釈のリストを含むカンマ区切り値 (CSV) ファイルを作成します。トレーニングファイルの入力形式が 1 行に 1 ドキュメント の場合、CSV ファイルには次の列が含まれている必要があります。

File	線グラフ	オフセットを開始する	オフセットを終了する	タイプ
ドキュメントを含むファイル名。たとえば、ドキュメントファイルの 1 つが `s3://my-S3-bucket/test-files/documen`	エンティティを含む行番号。入力形式が 1 ファイルにつき 1 つのドキュメントである場合は、この列を省略してください。	エンティティの開始位置を示す入力テキストの文字のオフセット (行の先頭を基準とした相対値)。最初の文字位置は 0 です。	エンティティの終了位置を示す入力テキストの文字オフセット。	顧客定義のエンティティタイプ。エンティティタイプは、アンダースコアで区切られた、大文字の文字列でなければなりませ

File	線グラフ	オフセットを開始する	オフセットを終了する	タイプ
ts.txt にある場合、File 列の値は documents.txt になります。ファイル名にはファイル拡張子 (この場合は「 .txt 」) を含める必要があります。				ん。MANAGER、SENIOR_NAGER 、PRODUCT_CODE などのわかりやすいエンティティタイプを使用することをお勧めします。モデルごとに、最大 25 のエンティティタイプをトレーニングできます。

トレーニングファイルの入力形式が [1 ファイルにつき 1 つのドキュメント] である場合は、行番号列を省略し、[オフセット開始] と [オフセット終了] の値は、ドキュメントの先頭からのエンティティのオフセットになります。

次の例は、1 行に 1 つのドキュメントを対象としています。documents.txt ファイルには 4 行 (行 0、1、2、3) が含まれています。

```
Diego Ramirez is an engineer in the high tech industry.
Emilio Johnson has been an engineer for 14 years.
J Doe is a judge on the Washington Supreme Court.
Our latest new employee, Mateo Jackson, has been a manager in the industry for 4 years.
```

注釈のリストを含む CSV ファイルは次のとおりです。

```
File, Line, Begin Offset, End Offset, Type
documents.txt, 0, 0, 13, ENGINEER
documents.txt, 1, 0, 14, ENGINEER
documents.txt, 3, 25, 38, MANAGER
```

> **Note**
>
> 注釈ファイルでは、エンティティを含む行番号は 0 行目から始まります。この例では、2 行目にはエンティティがないため、CSV ファイルには 2 行目の `documents.txt` エントリが含まれていません。

データファイルを作成する

エラーのリスクを減らすには、注釈を適切に構成された CSV ファイルに入れることが重要です。CSV ファイルを手動で構成するには、以下が満たされている必要があります。

- UTF-8 エンコーディングは、ほとんどの場合、デフォルトとして使用されている場合でも、明示的に指定する必要があります。
- 最初の行には列ヘッダー `File`、`Line` (オプション)、`Begin Offset`、`End Offset`、`Type` が含まれます。

潜在的な問題を避けるため、CSV 入力ファイルはプログラムで生成することを強くお勧めします。

次の例では、Python を使用して前述の注釈の CSV を生成します。

```
import csv
with open("./annotations/annotations.csv", "w", encoding="utf-8") as csv_file:
    csv_writer = csv.writer(csv_file)
    csv_writer.writerow(["File", "Line", "Begin Offset", "End Offset", "Type"])
    csv_writer.writerow(["documents.txt", 0, 0, 11, "ENGINEER"])
    csv_writer.writerow(["documents.txt", 1, 0, 5, "ENGINEER"])
    csv_writer.writerow(["documents.txt", 3, 25, 30, "MANAGER"])
```

PDF アノテーションファイル

PDF 注釈の場合は、 SageMaker Ground Truth を使用して、拡張マニフェストファイルにラベル付きデータセットを作成します。Ground Truth は、お客様 (またはお客様が雇用している労働力) が機械学習モデル用のトレーニングデータセットを構築するのに役立つデータラベリングサービスです。Amazon Comprehend は、カスタムモデル用トレーニングデータとして拡張マニフェストファイルを受け付けます。Amazon Comprehend コンソールまたは CreateEntityRecognizer API アクションを使用して、カスタムエンティティレコグナイザーを作成するときに、これらのファイルを提供できます。

Ground Truth の組み込みタスクタイプである名前付きエンティティ認識を使用してラベリングジョブを作成することで、ワーカーはテキスト内のエンティティを識別することができます。詳細については、「Amazon SageMaker デベロッパーガイド」の「固有表現認識」を参照してください。Amazon SageMaker Ground Truth の詳細については、「Amazon SageMaker Ground Truth を使用してデータにラベルを付ける」を参照してください。

> **Note**
>
> Ground Truth を使用すると、重複するラベル (複数のラベルに関連付けられるテキスト) を定義できます。ただし、Amazon Comprehend のエンティティ認識では、ラベルの重複はサポートされていません。

拡張マニフェストファイルは JSON 行形式になります。ファイル内の各行は、トレーニングドキュメントとそのラベルを含む完全な JSON オブジェクトで構成します。次の例は、テキストで言及されている個人の職業を検出するようにエンティティレコグナイザーをトレーニングする拡張マニフェストファイルです。

```
{"source":"Diego Ramirez is an engineer in the high tech
 industry.","NamedEntityRecognitionDemo":{"annotations":{"entities":
[{"endOffset":13,"startOffset":0,"label":"ENGINEER"}],"labels":
[{"label":"ENGINEER"}]}},"NamedEntityRecognitionDemo-metadata":
{"entities":[{"confidence":0.92}],"job-name":"labeling-job/
namedentityrecognitiondemo","type":"groundtruth/text-span","creation-
date":"2020-05-14T21:45:27.175903","human-annotated":"yes"}}
{"source":"J Doe is a judge on the Washington Supreme
 Court.","NamedEntityRecognitionDemo":{"annotations":{"entities":
[{"endOffset":5,"startOffset":0,"label":"JUDGE"}],"labels":
[{"label":"JUDGE"}]}},"NamedEntityRecognitionDemo-metadata":
{"entities":[{"confidence":0.72}],"job-name":"labeling-job/
namedentityrecognitiondemo","type":"groundtruth/text-span","creation-
date":"2020-05-14T21:45:27.174910","human-annotated":"yes"}}
{"source":"Our latest new employee, Mateo Jackson, has been a manager in
 the industry for 4 years.","NamedEntityRecognitionDemo":{"annotations":
{"entities":[{"endOffset":38,"startOffset":26,"label":"MANAGER"}],"labels":
[{"label":"MANAGER"}]}},"NamedEntityRecognitionDemo-metadata":
{"entities":[{"confidence":0.91}],"job-name":"labeling-job/
namedentityrecognitiondemo","type":"groundtruth/text-span","creation-
date":"2020-05-14T21:45:27.174035","human-annotated":"yes"}}
```

この JSON 行ファイルの各行は完全な JSON オブジェクトであり、属性にはドキュメントテキスト、アノテーション、および Ground Truth からのその他のメタデータが含まれます。次の例は、拡張マニフェストファイル内の 1 つの JSON オブジェクトですが、読みやすいようにフォーマットされています。

```
{
  "source": "Diego Ramirez is an engineer in the high tech industry.",
  "NamedEntityRecognitionDemo": {
    "annotations": {
      "entities": [
        {
          "endOffset": 13,
          "startOffset": 0,
          "label": "ENGINEER"
        }
      ],
      "labels": [
        {
          "label": "ENGINEER"
        }
      ]
    }
  },
  "NamedEntityRecognitionDemo-metadata": {
    "entities": [
      {
        "confidence": 0.92
      }
    ],
    "job-name": "labeling-job/namedentityrecognitiondemo",
    "type": "groundtruth/text-span",
    "creation-date": "2020-05-14T21:45:27.175903",
    "human-annotated": "yes"
  }
}
```

この例では、source 属性はトレーニングドキュメントのテキストを提供し、NamedEntityRecognitionDemo 属性はテキスト内のエンティティのアノテーションを提供します。NamedEntityRecognitionDemo 属性の名前は任意です。Ground Truth でのラベリングジョブの定義では任意の名前を指定できます。

この例では、NamedEntityRecognitionDemo 属性はラベルの属性名 です。これは、Ground Truth ワーカーがトレーニングデータに割り当てるラベルを提供する属性です。トレーニングデータを Amazon Comprehend に提供するときは、1 つ以上のラベル属性名を指定する必要があります。指定する属性名の数は、拡張マニフェストファイルが単一のラベリングジョブの出力であるか、チェーンラベリングジョブの出力であるかによって異なります。

ファイルが 1 つのラベルリングジョブの出力である場合は、Ground Truth でジョブが作成されたときに使用された 1 つのラベル属性名を指定します。

ファイルがチェーンラベリングジョブの出力である場合は、チェーン内の 1 つ以上のジョブに対するラベル属性名を指定します。各ラベル属性名には、それぞれ 1 つのジョブのアノテーションが含まれます。チェーンラベリングジョブによって生成される拡張マニフェストファイルには、これらの属性を 5 つまで指定できます。

通常、拡張マニフェストファイルでは、source キーの後にラベル属性名が付きます。ファイルがチェーンジョブの出力である場合は、複数のラベル属性名が存在します。トレーニングデータを Amazon Comprehend に提供するときは、モデルに関連するアノテーションを含む属性のみを提供してください。「-metadata」で終わる属性は指定しないでください。

連鎖ラベル付けジョブの詳細と、それらが生成する出力の例については、「Amazon SageMaker デベロッパーガイド」の「連鎖ラベル付けジョブ」を参照してください。

PDF ファイルへの注釈

SageMaker Ground Truth でトレーニング PDFs注釈を付ける前に、以下の前提条件を満たしてください。

- Python3.8.x のインストール。
- jq のインストール
- AWS CLI のインストール

  us-east-1 リージョンを使用している場合は、Python 環境に既にインストールされているため、AWS CLI のインストールをスキップできます。この場合、 AWS Cloud9 で Python 3.8 を使用する仮想環境を作成します。
- AWS 認証情報を設定する
- 注釈をサポートするプライベート SageMaker Ground Truth ワークフォースを作成する

  選択したワークチーム名は、インストール時に使用するので、新しいプライベートワークフォースに必ず記録してください。

トピック

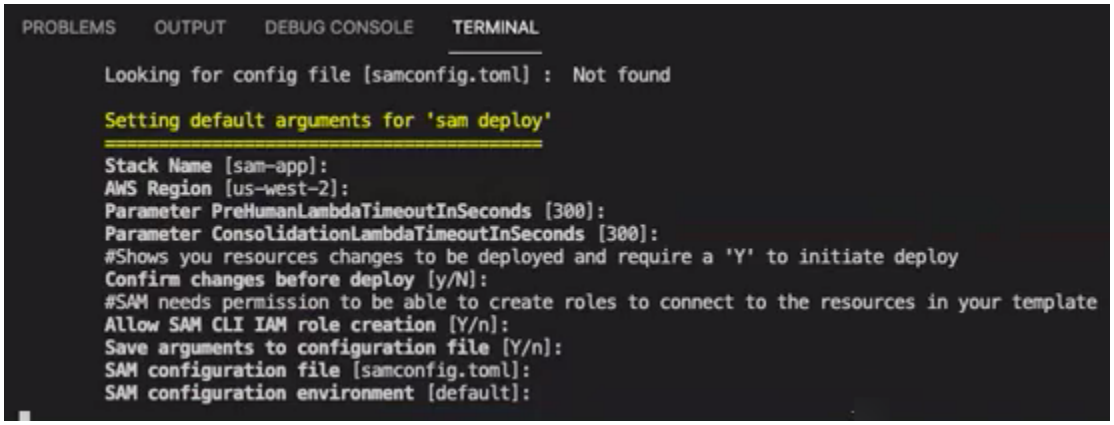
- 環境のセットアップ
- PDF を S3 バケットにアップロードする
- 注釈ジョブの作成
- SageMaker Ground Truth での注釈付け

環境のセットアップ

1. Windows を使用している場合は Cygwin をインストールし、Linux または Mac の場合は、この手順をスキップします。
2. から注釈アーティファクトをダウンロードします GitHub。ファイル を解凍します。
3. ターミナルウィンドウから、解凍したフォルダ (amazon-comprehend-semi-structured-documents-annotation-tools-main) に移動します。
4. このフォルダーには、依存関係のインストール、Python virtualenv の設定、および必要なリソースのデプロイを行うために実行する `Makefiles` オプションが含まれています。readme ファイルを確認して選択してください。
5. 推奨オプションでは、単一のコマンドを使用してすべての依存関係を virtualenv にインストールし、テンプレートから AWS CloudFormation スタックを構築し、インタラクティブガイダンス AWS アカウント を使用してスタックを にデプロイします。次のコマンドを実行します。

   ```
   make ready-and-deploy-guided
   ```

   このコマンドは一連の設定オプションを表します。 AWS リージョン が正しいことを確認してください。他のすべてのフィールドで、デフォルト値をそのまま使用するか、カスタム値を入力できます。 AWS CloudFormation スタック名を変更する場合は、次のステップで必要に応じて書き留めます。



CloudFormation スタックは、注釈ツールに必要な AWS Lambdas 、AWS IAM ロール、および AWS S3 バケットを作成および管理します。

これらの各リソースは、 CloudFormation コンソールのスタックの詳細ページで確認できます。

6. コマンドは、デプロイを開始するように促します。 は、指定されたリージョン内のすべてのリソース CloudFormation を作成します。

CloudFormation スタックのステータスが create-complete に移行すると、リソースは使用できる状態になります。

PDF を S3 バケットにアップロードする

セットアップセクションで、comprehend-semi-structured-documents-${AWS::Region}-${AWS::AccountId} という名前の S3 バケットを作成する CloudFormation スタックをデプロイしました。ソース PDF ドキュメントをこのバケットにアップロードします。

> **Note**
>
> このバケットには、ラベル付けジョブに必要なデータが含まれています。 Lambda 実行ロールポリシーは、このバケットへのアクセスを Lambda 関数に付与します。
> S3 バケット名は、CloudFormation スタックの詳細でSemiStructuredDocumentsS3Bucket」キーを使用して確認できます。

1. S3 バケットに新規フォルダを作成します。この新しいフォルダに「src」という名前を付けます。
2. PDF ソースファイルを 「src」フォルダに追加します。次の手順では、これらのファイルに注釈を付けて、レコグナイザーを訓練します。

3. （オプション) ソースドキュメントをローカルディレクトリから S3 バケットにアップロードするために使用できる AWS CLI の例を次に示します。

   ```
   aws s3 cp --recursive local-path-to-your-source-docs s3://deploy-guided/src/
   ```

   または、リージョンとアカウント ID を使用します:

   ```
   aws s3 cp --recursive local-path-to-your-source-docs s3://deploy-guided-Region-AccountID/src/
   ```

4. これで、 SageMaker Ground Truth のプライベートワークフォースがあり、ソースファイルを S3 バケットの deploy-guided/src/ にアップロードしました。注釈付けを開始する準備が整いました。

注釈ジョブの作成

bin ディレクトリの comprehend-ssie-annotation-tool-cli.py スクリプトは、 SageMaker Ground Truth ラベル付けジョブの作成を効率化するシンプルなラッパーコマンドです。Python スクリプトは S3 バケットからソースドキュメントを読み取り、単一ページのマニフェストファイルを作成します。これは、1 行あたり 1 つのソースドキュメントが対応します。　次に、スクリプトはマニフェストファイルを入力として必要とするラベル付けジョブを作成します。

Python スクリプトは、「セットアップ」セクションで設定した S3 ???バケットと CloudFormation スタックを使用します。このスクリプトに必要な入力パラメータには以下が含まれます。

- input-s3-path: S3 バケットにアップロードしたソースドキュメントへの S3 URI。例: `s3://deploy-guided/src/`。このパスにリージョンとアカウント ID を追加することもできます。例: `s3://deploy-guided-Region-AccountID/src/`。
- cfn-name: CloudFormation スタック名。スタック名にデフォルト値を使用した場合、CFN 名は sam-app になります。
- work-team-name: SageMaker Ground Truth でプライベートワークフォースを構築したときに作成したワークフォース名。
- job-name-prefix: SageMaker Ground Truth ラベル付けジョブのプレフィックス。このフィールドには 29 文字の制限があることに注意してください。この値にはタイムスタンプが付加されます。例: `my-job-name-20210902T232116`。
- entity-types: ラベル付けジョブ中に使用する必要があるエンティティ。カンマで区切られています。このリストには、トレーニングデータセットで注釈を付けたいエンティティがすべて含まれて

いる必要があります。Ground Truth のラベル付けジョブでは、注釈者が PDF ドキュメント内のコンテンツにラベル付けできるように、これらのエンティティのみが表示されます。

スクリプトがサポートするその他の引数を表示するには、-h オプションを使用してヘルプコンテンツを表示します。

- 前述のリストで説明したように、入力パラメータを指定して次のスクリプトを実行します。

```
python bin/comprehend-ssie-annotation-tool-cli.py \
--input-s3-path s3://deploy-guided-Region-AccountID/src/ \
--cfn-name sam-app \
--work-team-name my-work-team-name \
--region us-east-1 \
--job-name-prefix my-job-name-20210902T232116 \
--entity-types "EntityA, EntityB, EntityC" \
--annotator-metadata "key=info,value=sample,key=Due Date,value=12/12/2021"
```

スクリプトは以下の出力を生成します。

```
Downloaded files to temp local directory /tmp/a1dc0c47-0f8c-42eb-9033-74a988ccc5aa
Deleted downloaded temp files from /tmp/a1dc0c47-0f8c-42eb-9033-74a988ccc5aa
Uploaded input manifest file to s3://comprehend-semi-structured-documents-
us-west-2-123456789012/input-manifest/my-job-name-20220203-labeling-
job-20220203T183118.manifest
Uploaded schema file to s3://comprehend-semi-structured-documents-us-
west-2-123456789012/comprehend-semi-structured-docs-ui-template/my-job-
name-20220203-labeling-job-20220203T183118/ui-template/schema.json
Uploaded template UI to s3://comprehend-semi-structured-documents-us-
west-2-123456789012/comprehend-semi-structured-docs-ui-template/my-job-
name-20220203-labeling-job-20220203T183118/ui-template/template-2021-04-15.liquid
Sagemaker GroundTruth Labeling Job submitted: arn:aws:sagemaker:us-
west-2:123456789012:labeling-job/my-job-name-20220203-labeling-job-20220203t183118
(amazon-comprehend-semi-structured-documents-annotation-tools-main)
 user@3c063014d632 amazon-comprehend-semi-structured-documents-annotation-tools-
main %
```

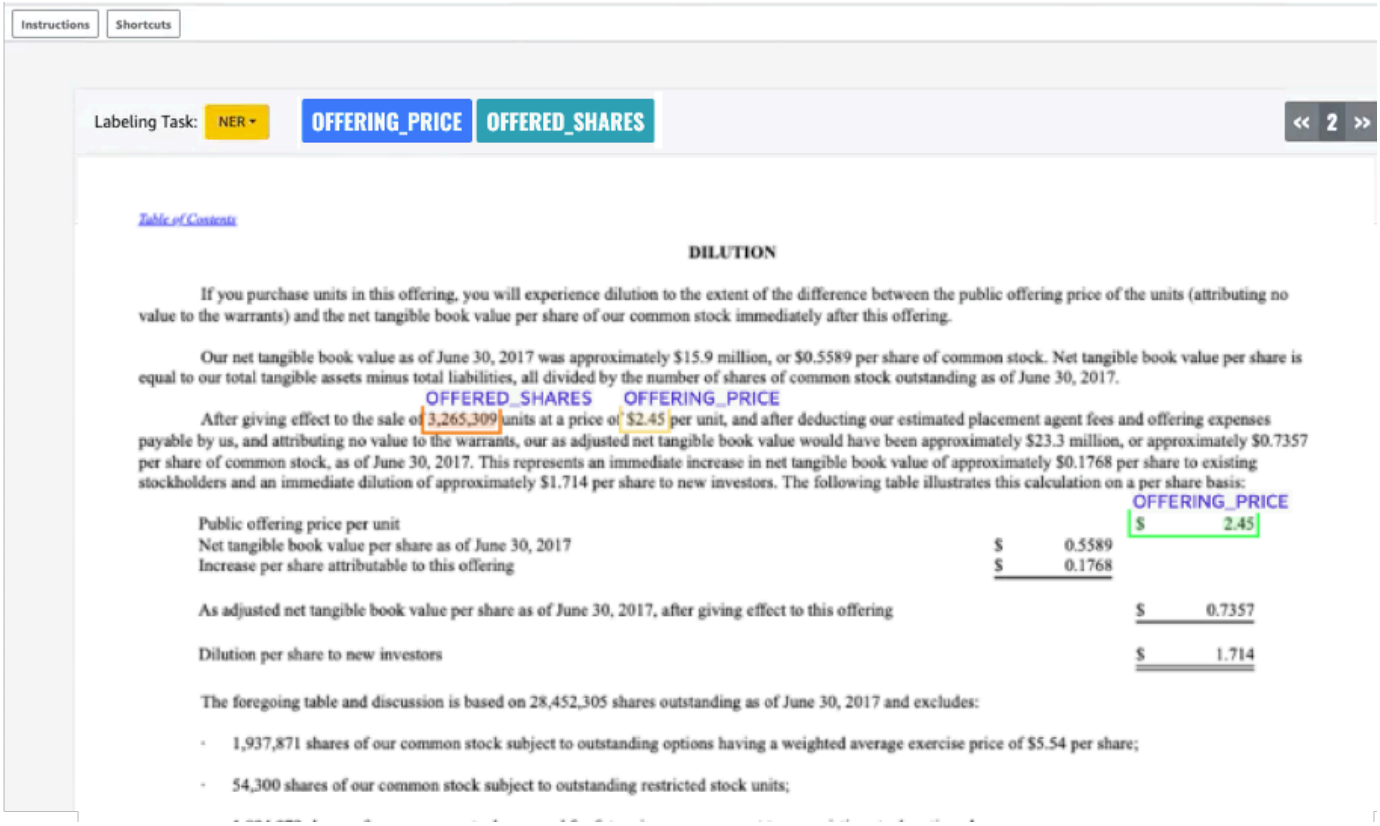
SageMaker Ground Truth での注釈付け

必要なリソースを設定し、ラベル付けジョブを作成したら、ラベル付けポータルにログインして PDF に注釈を付けることができます。

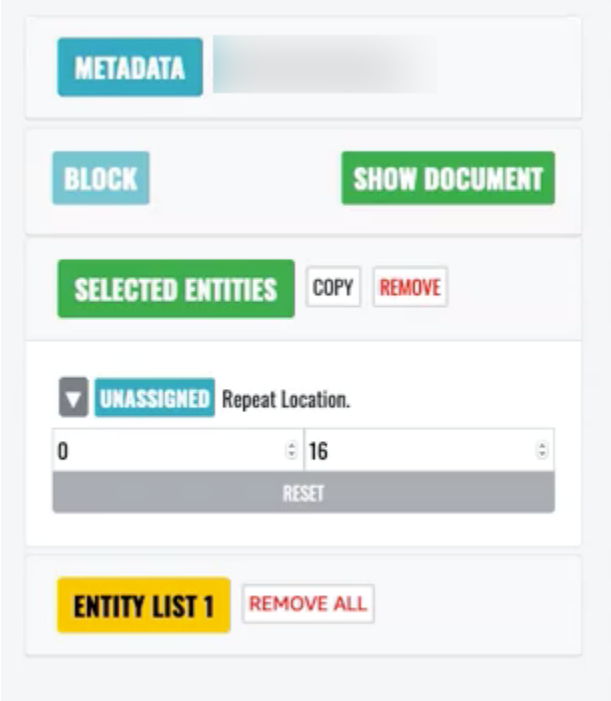
1. Chrome または Firefox ウェブブラウザSageMaker を使用してコンソールにログインします。
2. [ラベル付けワークフォース] を選択し、[プライベート] を選択します。
3. [プライベートワークフォースの概要] で、プライベートワークフォースで作成したラベリングポータルのサインイン URL を選択します。適切な認証情報を使用してサインインします。

   リストにジョブが表示されなくても心配はいりません。注釈用にアップロードしたファイルの数によっては、更新に時間がかかる場合があります。
4. タスクを選択し、右上で [作業開始] を選択して注釈画面を開きます。

   注釈画面でいずれかのドキュメントが開き、その上に、設定時に指定したエンティティタイプが表示されます。　エンティティタイプの右側には、ドキュメント間を移動できる矢印があります。



開いているドキュメントに注釈を付けます。また、各ドキュメントの注釈を削除、元に戻す、または自動タグ付けすることもできます。これらのオプションは注釈ツールの右側のパネルにあります。



自動タグを使用するには、エンティティのいずれかのインスタンスに注釈を付けます。その特定の単語以外のすべてのインスタンスには、そのエンティティタイプで自動的に注釈が付けられます。

完了したら、右下の [送信] を選択し、ナビゲーション矢印を使用して次のドキュメントに移動します。すべての PDF に注釈が付けられるまで、これを繰り返します。

すべてのトレーニングドキュメントに注釈を付けると、次の場所にある Amazon S3 バケットに JSON 形式の注釈が表示されます。

```
/output/your labeling job name/annotations/
```

出力フォルダには、トレーニングドキュメント内のすべての注釈を一覧表示する出力マニフェストファイルも含まれています。　出力マニフェストファイルは次の場所にあります。

```
/output/your labeling job name/manifests/
```

カスタムエンティティレコグナイザーモデルのトレーニング

カスタムエンティティレコグナイザーは、モデルをトレーニングするときに含めるエンティティタイプのみを識別します。プリセットのエンティティタイプは自動で含みません。場所、日付、人などの

プリセットエンティティタイプも特定したい場合は、そのエンティティに追加のトレーニングデータを提供する必要があります。

注釈付きの PDF ファイルを使用してカスタムエンティティレコグナイザー機能を作成すると、そのレコグナイザーをさまざまな入力ファイル形式 (プレーンテキスト、画像ファイル (JPG、PNG、TIFF)、PDF ファイル、Word ドキュメントで使用できます。前処理やドキュメントの統合は不要です。Amazon Comprehend は、画像ファイルや Word ドキュメントの注釈をサポートしていません。

> **Note**
>
> 注釈付き PDF ファイルを使用するカスタムエンティティレコグナイザーは、英語のドキュメントのみをサポートします。

カスタムエンティティレコグナイザーを作成したら、 DescribeEntityRecognizerオペレーションを使用してリクエストの進行状況をモニタリングできます。`Status` フィールドが `TRAINED` になると、レコグナイザーモデルを使用してカスタムエンティティが認識できます。

トピック

- カスタムレコグナイザーをトレーニングする (コンソール)
- カスタムエンティティレコグナイザーをトレーニングする (API)
- カスタムエンティティレコグナイザーメトリクス

カスタムレコグナイザーをトレーニングする (コンソール)

カスタムエンティティレコグナイザーは、Amazon Comprehend コンソールを使用して作成することができます。このセクションでは、カスタムエンティティレコグナイザーを作成してトレーニングする方法を示します。

コンソールを使用したカスタムエンティティレコグナイザーの作成 - CSV 形式

カスタムエンティティレコグナイザーを作成するには、最初にモデルをトレーニングするためのデータセットを用意します。このデータセットには、アノテーション付き文書一式を含めるか、エンティティのリストとそのタイプラベル、それらエンティティを含む文書一式を含めます。詳細については、「カスタムエンティティ認識」を参照してください。

CSV ファイルを使用してカスタムエンティティレコグナイザーをトレーニングする

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/comprehend/ で Amazon Comprehend コンソールを開きます。
2. 左側のメニューから [カスタマイズ] を選択し、[カスタムエンティティ認識] を選択します。
3. [モデルの作成] を選択します。
4. レコグナイザーに名前を付けます。名前は、 リージョンおよびアカウント内で一意である必要があります。
5. 言語を選択します。
6. [カスタムエンティティタイプ] でカスタムラベルを入力します。レコグナイザーは、このカスタムレベルを使用してデータセットからエンティティタイプを検索します。

   エンティティタイプは大文字でなければならず、複数の単語で構成されている場合は、単語をアンダースコアで区切ります。
7. [Add type (タイプを追加)] を選択します。
8. 別のエンティティタイプを追加する場合は、そのエンティティタイプを入力し、[タイプを追加] を選択します。追加したエンティティタイプの 1 つを削除する場合は、[タイプを削除] を選択し、リストから削除するエンティティタイプを選択します。最大 25 個のエンティティタイプを一覧表示できます。
9. トレーニングジョブを暗号化するには、[Recognizer encryption (レコグナイザーの暗号化)] を選択し、現在のアカウントに関連付けられている KMS キーを使用するか、別のアカウントの KMS キーを使用するか選択します。

   - 現在のアカウントに関連付けられているキーを使用する場合は、[KMS キー ID] でキー ID を選択します。
   - 別のアカウントに関連付けられているキーを使用する場合は、[KMS キーの ARN] でキー ID の ARN を入力します。

   > **Note**
   >
   > KMS キーの作成と使用、および関連する暗号化の詳細については、「AWS Key Management Service」を参照してください。
10. 「データ仕様」で、トレーニングドキュメントの形式を選択します。

- CSV ファイル — トレーニングドキュメントを補完する CSV ファイル。CSV ファイルには、トレーニングを受けたモデルが検出するカスタムエンティティに関する情報が含まれます。必要なファイルの形式は、アノテーションを提供するのか、エンティティリストを提供するのかによって異なります。
- 拡張マニフェスト — Amazon SageMaker Ground Truth によって生成されるラベル付きデータセット。このファイルは JSON 行形式です。各行は、トレーニングドキュメントとそのラベルを含む完全な JSON オブジェクトで構成します。各ラベルは、トレーニングドキュメント内の名前付きエンティティに対するアノテーションです。拡張マニフェストファイルは最大 5 つ指定できます。

使用可能な形式の詳細については、「カスタムエンティティレコグナイザーモデルのトレーニング」を参照してください。

11. [トレーニングタイプ]で、使用するトレーニングタイプを選択します。

    - アノテーションとトレーニングドキュメントを使用
    - エンティティリストとトレーニングドキュメントを使用

    アノテーションを選択する場合は、Amazon S3 上のアノテーションファイルの URL を入力します。アノテーションファイルがあるAmazon S3 上のバケットまたはフォルダに移動して、[Browse S3] を選択することもできます。

    エンティティリストを選択する場合は、Amazon S3 上のエンティティリストの URL を入力します。エンティティリストがある Amazon S3 上のバケットまたはフォルダに移動して、[Browse S3] を選択することもできます。

12. トレーニングドキュメントを含む Amazon S3 上の入力データセットの URL を入力します。トレーニングドキュメントがある Amazon S3 上のバケットまたはフォルダに移動して、[フォルダの選択] を選択することもできます。

13. [テストデータセット] で、トレーニング済みモデルのパフォーマンスを評価する方法を選択します。これは、アノテーションとエンティティリストトレーニングタイプの両方で行うことができます。

    - Autosplit: Autosplit は、テスト用データとして提供されているトレーニングデータの 10% を自動的に選択して使用します。
    - (オプション) お客様指定:お客様指定を選択すると、使用するテスト用データを正確に指定できます。

14. [お客様指定のテストデータセット] を選択した場合は、Amazon S3 上のアノテーションファイルの URL を入力します。アノテーションファイルがある Amazon S3 上のバケットまたはフォルダに移動して、[Select folder (フォルダの選択)] を選択することもできます。
15. [IAM ロールを選択] セクションで、既存の IAM ロールを選択するか、新しい IAM ロールを作成します。
    - 既存の IAM ロールを選択 — 入出力の Amazon S3 バケットへのアクセス許可を持つ IAM ロールがすでにある場合は、このオプションを選択します。
    - 新しい IAM ロールの作成 — Amazon Comprehend が入出力バケットに対する適切なアクセス許可を持つ新しい IAM ロールを作成する場合は、このオプションを選択します。

    > **Note**
    >
    > 入力ドキュメントが暗号化されている場合、使用する IAM ロールには `kms:Decrypt` アクセス許可が必要です。詳細については、「KMS 暗号化を使用するために必要なアクセス許可」を参照してください。

16. (オプション) VPC から Amazon Comprehend にリソースを起動するには、VPC の下に VPC ID を入力するか、ドロップダウンリストから ID を選択します。
    1. [サブネット] でサブネットを選択します。最初のサブネットを選択すると、追加のサブネットを選択できます。
    2. セキュリティグループを指定した場合は、[セキュリティグループ] で、使用するセキュリティグループを選択します。最初のセキュリティグループを選択すると、追加のセキュリティグループを選択できます。

    > **Note**
    >
    > カスタムエンティティ認識ジョブで VPC を使用する場合、Create および Start オペレーションに使用する `DataAccessRole` には、入力ドキュメントと出力バケットへのアクセスに使用する VPC へのアクセス許可が必要です。

17. (オプション) カスタムエンティティレコグナイザーにタグを追加するには、[タグ] でキーと値のペアを入力します。[Add tag] (タグを追加) を選択します。レコグナイザーを作成する前にこのペアを削除するには、[タグを削除] を選択します。
18. [Train] を選択します。

ステータスと共に新しいレコグナイザーがリストに表示されます。最初は Submitted と表示されます。続いて、トレーニングドキュメントを処理している分類子には Training、使用できる分類子には Trained、エラーがある分類子には In error が表示されます。ジョブをクリックすると、エラーメッセージなど、レコグナイザーに関する詳細情報を取得できます。

コンソールを使用したカスタムエンティティレコグナイザーの作成 - 拡張マニフェスト

プレーンテキスト、PDF、または Word ドキュメントを使用してカスタムエンティティレコグナイザーをトレーニングする

1. にサインイン AWS Management Console し、Amazon Comprehend コンソールを開きます。
2. 左側のメニューから [カスタマイズ] を選択し、[カスタムエンティティ認識] を選択します。
3. [レコグナイザーをトレーニング] を選択します。
4. レコグナイザーに名前を付けます。名前は、 リージョンおよびアカウント内で一意である必要があります。
5. 言語を選択します。注: PDF または Word ドキュメントでトレーニングする場合、サポートされている言語は英語だけです。
6. [カスタムエンティティタイプ] でカスタムラベルを入力します。レコグナイザーは、このカスタムレベルを使用してデータセットからエンティティタイプを検索します。

   エンティティタイプは大文字でなければならず、複数の単語で構成されている場合は、単語をアンダースコアで区切ります。
7. [Add type (タイプを追加)] を選択します。
8. 別のエンティティタイプを追加する場合は、そのエンティティタイプを入力し、[タイプを追加] を選択します。追加したエンティティタイプの 1 つを削除する場合は、[タイプを削除] を選択し、リストから削除するエンティティタイプを選択します。最大 25 個のエンティティタイプを一覧表示できます。
9. トレーニングジョブを暗号化するには、[Recognizer encryption (レコグナイザーの暗号化)] を選択し、現在のアカウントに関連付けられている KMS キーを使用するか、別のアカウントの KMS キーを使用するか選択します。

   - 現在のアカウントに関連付けられているキーを使用する場合は、[KMS キー ID] でキー ID を選択します。
   - 別のアカウントに関連付けられているキーを使用する場合は、[KMS キーの ARN] でキー ID の ARN を入力します。

> **Note**
>
> KMS キーの作成と使用、および関連する暗号化の詳細については、「AWS Key Management Service」を参照してください。

10. [トレーニングデータ] で、データ形式として拡張マニフェストを選択します。

    - 拡張マニフェスト — は、Amazon SageMaker Ground Truth によって生成されるラベル付きデータセットです。このファイルは JSON 行形式です。ファイル内の各行は、トレーニングドキュメントとそのラベルを含む完全な JSON オブジェクトで構成します。各ラベルは、トレーニングドキュメント内の名前付きエンティティに対するアノテーションです。拡張マニフェストファイルは最大 5 つ指定できます。トレーニング用データに PDF ドキュメントを使用する場合は、拡張マニフェストを選択する必要があります。拡張マニフェストファイルは最大 5 つ指定できます。トレーニングデータとして使用する属性は、1 つのファイルにつき 5 つまで指定することができます。

    使用可能な形式の詳細については、「カスタムエンティティレコグナイザーモデルのトレーニング」を参照してください。

11. トレーニングモデルタイプを選択します。

    プレーンテキストドキュメント を選択した場合は、入力場所 で、Amazon SageMakerGround Truth 拡張マニフェストファイルの Amazon S3URL を入力します。拡張マニフェストがある Amazon S3 上のバケットまたはフォルダに移動して、[フォルダの選択] を選択することもできます。

12. [属性名] で、アノテーションを含んでいる属性の名前を入力します。ファイルに複数のチェーンラベリングジョブのアノテーションが含まれている場合は、ジョブごとに属性を追加します。この場合、各属性にはラベリングジョブのアノテーション一式が含まれます。注: ファイルごと 5 つまで属性名を指定できます。

13. [追加] を選択します。

14. PDF、Word ドキュメントを入力場所 で選択した場合は、Amazon SageMaker Ground Truth 拡張マニフェストファイルの Amazon S3URL を入力します。拡張マニフェストがある Amazon S3 上のバケットまたはフォルダに移動して、[フォルダの選択] を選択することもできます。

15. アノテーションデータファイルの S3 プレフィックスを入力します。これらは、ラベル付けした PDF ドキュメントです。

16. ソースドキュメントの S3 プレフィックスを入力します。これらは、ラベリングジョブ用に Ground Truth に指定した元の PDF 文書 (データオブジェクト) です。

17. アノテーションを含む属性名を入力します。注: ファイルごと 5 つまで属性名を指定できます。ファイルにあって指定されなかった属性は無視されます。

18. IAM ロールのセクションで、既存の IAM ロールを選択するか、新しい IAM ロールを作成します。

    - 既存の IAM ロールを選択 — 入出力の Amazon S3 バケットへのアクセス許可を持つ IAM ロールがすでにある場合は、このオプションを選択します。
    - 新しい IAM ロールの作成 — Amazon Comprehend が入出力バケットに対する適切なアクセス許可を持つ新しい IAM ロールを作成する場合は、このオプションを選択します。

    > **Note**
    >
    > 入力ドキュメントが暗号化されている場合、使用する IAM ロールには `kms:Decrypt` アクセス許可が必要です。詳細については、「KMS 暗号化を使用するために必要なアクセス許可」を参照してください。

19. (オプション) VPC から Amazon Comprehend にリソースを起動するには、VPC の下に VPC ID を入力するか、ドロップダウンリストから ID を選択します。

    1. [サブネット] でサブネットを選択します。最初のサブネットを選択すると、追加のサブネットを選択できます。
    2. セキュリティグループを指定した場合は、[セキュリティグループ] で、使用するセキュリティグループを選択します。最初のセキュリティグループを選択すると、追加のセキュリティグループを選択できます。

    > **Note**
    >
    > カスタムエンティティ認識ジョブで VPC を使用する場合、Create および Start オペレーションに使用する `DataAccessRole` には、入力ドキュメントと出力バケットへのアクセスに使用する VPC へのアクセス許可が必要です。

20. (オプション) カスタムエンティティレコグナイザーにタグを追加するには、[タグ] でキーと値のペアを入力します。[Add tag] (タグを追加) を選択します。レコグナイザーを作成する前にこのペアを削除するには、[タグを削除] を選択します。

21. [Train] を選択します。

ステータスと共に新しいレコグナイザーがリストに表示されます。最初は Submitted と表示されます。続いて、トレーニングドキュメントを処理している分類子には Training、使用できる分類子には Trained、エラーがある分類子には In error が表示されます。ジョブをクリックすると、エラーメッセージなど、レコグナイザーに関する詳細情報を取得できます。

カスタムエンティティレコグナイザーをトレーニングする (API)

カスタムエンティティ認識モデルを作成してトレーニングするには、Amazon Comprehend CreateEntityRecognizer API オペレーションを使用します。

トピック

- AWS Command Line Interface を使用したカスタムエンティティレコグナイザーのトレーニング
- AWS SDK for Java を使用したカスタムエンティティレコグナイザーのトレーニング
- Python (Boto3) を使用したカスタムエンティティレコグナイザーのトレーニング

AWS Command Line Interface を使用したカスタムエンティティレコグナイザーのトレーニング

以下は、AWS CLI での CreateEntityRecognizer オペレーションとその他の関連する API の使用例を示しています。

これらの例は、Unix、Linux、および macOS 用の形式になっています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

create-entity-recognizer CLI コマンドを使用してカスタムエンティティレコグナイザーを作成します。 input-data-config パラメータの詳細については、「Amazon Comprehend API リファレンスCreateEntityRecognizer」の「」を参照してください。

```
aws comprehend create-entity-recognizer \
     --language-code en \
     --recognizer-name test-6 \
     --data-access-role-arn "arn:aws:iam::account number:role/service-role/
AmazonComprehendServiceRole-role" \
     --input-data-config "EntityTypes=[{Type=PERSON}],Documents={S3Uri=s3://Bucket
 Name/Bucket Path/documents},
                Annotations={S3Uri=s3://Bucket Name/Bucket Path/annotations}" \
```

```
     --region region
```

`list-entity-recognizers` CLI コマンドを使用して、リージョン内のすべてのエンティティレコグナイザーを一覧表示します。

```
aws comprehend list-entity-recognizers \
     --region region
```

`describe-entity-recognizer` CLI コマンドを使用して、カスタムエンティティレコグナイザーのジョブステータスを確認します。

```
aws comprehend describe-entity-recognizer \
     --entity-recognizer-arn arn:aws:comprehend:region:account number:entity-
recognizer/test-6 \
     --region region
```

AWS SDK for Java を使用したカスタムエンティティレコグナイザーのトレーニング

この例では、Java を使用してカスタムエンティティレコグナイザーを作成し、モデルをトレーニングします。

Java を使用した Amazon Comprehend の例については、「Amazon Comprehend の Java の例」を参照してください。

Python (Boto3) を使用したカスタムエンティティレコグナイザーのトレーニング

Boto3 SDK のインスタンスを作成する:

```
import boto3
import uuid
comprehend = boto3.client("comprehend", region_name="region")
```

エンティティレコグナイザーを作成する:

```
response = comprehend.create_entity_recognizer(
    RecognizerName="Recognizer-Name-Goes-Here-{}".format(str(uuid.uuid4())),
    LanguageCode="en",
    DataAccessRoleArn="Role ARN",
    InputDataConfig={
        "EntityTypes": [
            {
```

```
                "Type": "ENTITY_TYPE"
            }
        ],
        "Documents": {
            "S3Uri": "s3://Bucket Name/Bucket Path/documents"
        },
        "Annotations": {
            "S3Uri": "s3://Bucket Name/Bucket Path/annotations"
        }
    }
)
recognizer_arn = response["EntityRecognizerArn"]
```

すべてのレコグナイザーを一覧表示する:

```
response = comprehend.list_entity_recognizers()
```

エンティティレコグナイザーが TRAINED ステータスになるのを待つ:

```
while True:
    response = comprehend.describe_entity_recognizer(
        EntityRecognizerArn=recognizer_arn
    )

    status = response["EntityRecognizerProperties"]["Status"]
    if "IN_ERROR" == status:
        sys.exit(1)
    if "TRAINED" == status:
        break

    time.sleep(10)
```

カスタムエンティティレコグナイザーメトリクス

Amazon Comprehend には、エンティティレコグナイザーが、業務にどの程度役立つかを見積もるのに役立つメトリクスが用意されています。これらはレコグナイザーモデルのトレーニングに基づいているため、トレーニング中のモデルのパフォーマンスを正確に表していますが、エンティティ検出中の API パフォーマンスの概算にすぎません。

メトリクスは、トレーニング済みのエンティティレコグナイザーからメタデータが返されるたびに返されます。

Amazon Comprehend は、一度に最大 25 のカスタムエンティティでモデルのトレーニングをサポートします。トレーニング済みのエンティティレコグナイザーからメトリクスが返されると、レコグナイザー全体 (グローバルメトリクス) と個々のエンティティ (エンティティメトリクス) の両方に対してスコアが計算されます。

グローバルメトリクスとエンティティメトリクスの両方として、次の 3 つのメトリクスを使用できます。

- 精度

  これは、正しく識別され、正しくラベル付けされた、システムによって生成されたエンティティの割合を示します。これは、モデルのエンティティ ID が本当に良いIDである回数を示します。ID の総数に対する割合です。

  つまり、精度は真陽性 (tp) と偽陽性 (fp) に基づいており、精度 = tp/(tp + fp) として計算されます。

  たとえば、あるドキュメントにエンティティの例が 2 つあるとモデルが予測し、実際には 1 つしか存在しない場合、結果は 1 つが真陽性、もう 1 つが偽陽性になります。この場合、精度は 1/(1 + 1) です。モデルによって識別された 2 つのエンティティのうち、正しいのは 1 つであるため、精度は 50% です。

- リコール

  これは、ドキュメントに含まれるエンティティのうち、システムによって正しく識別され、ラベル付けされたエンティティの割合を示しています。数学的には、これは正しい ID の総数、真陽性 (tp) と正しく認識しなかった ID の合計数、偽陰性 (fn) で定義されます。

  リコール = tp/(tp + fn) として計算されます。たとえば、あるモデルが 1 つのエンティティを正しく識別しても、そのエンティティが存在する他の 2 つのインスタンスを見逃した場合、結果は 1 つの真陽性と 2 つの偽陰性になります。この場合、リコールは 1/(1 + 2) です。考えられる3つの例のうち、1つのエンティティが正しいため、リコール率は 33.33% です。

- F1 スコア

  これは、カスタムエンティティ認識のモデルの全体の精度を測定する精度メトリクスとリコールメトリクスを組み合わせたものです。F1 スコアは、精度とリコールのメトリクスの調和平均です。F1 = 2 * 精度 * 再現率/(精度 + 再現率)。

> **Note**
>
> 直感的に見ると、調和平均は単純平均やその他の平均よりも両極端に不利になります (例: precision = 0、recall = 1 は、可能なすべてのスパンを予測することで簡単に実現できます。ここでは、単純平均は 0.5 ですが、F1 は 0 として不利にします)。

上の例では precision = 50%、 recall = 33.33% なので、 F1 = 2 * 0.5 * 0.3333/(0.5 + 0.3333) です。F1 スコアは 0.3975、つまり 39.75% です。

グローバルメトリクスと個別エンティティメトリクス

グローバルエンティティメトリクスと個別エンティティメトリクスの関係は、次の文を場所または人物であるエンティティについて分析するとわかります。

```
John Washington and his friend Smith live in San Francisco, work in San Diego, and own
    a house in Seattle.
```

この例では、モデルは次のような予測を行います。

```
John Washington = Person
Smith = Place
San Francisco = Place
San Diego = Place
Seattle = Person
```

しかし、予測は以下のようになっているはずです。

```
John Washington = Person
Smith = Person
San Francisco = Place
San Diego = Place
Seattle = Place
```

この場合の個々のエンティティメトリクスは次のようになります。

```
entity:  Person
```

```
  True positive (TP) = 1 (because John Washington is correctly predicted to be a
    Person).
  False positive (FP) = 1 (because Seattle is incorrectly predicted to be a Person,
    but is actually a Place).
  False negative (FN) = 1 (because Smith is incorrectly predicted to be a Place, but
    is actually a Person).
  Precision = 1 / (1 + 1) = 0.5 or 50%
  Recall = 1 / (1+1) = 0.5 or 50%
  F1 Score = 2 * 0.5 * 0.5 / (0.5 + 0.5) = 0.5 or 50%

entity:  Place
  TP = 2 (because San Francisco and San Diego are each correctly predicted to be a
    Place).
  FP = 1 (because Smith is incorrectly predicted to be a Place, but is actually a
    Person).
  FN = 1 (because Seattle is incorrectly predicted to be a Person, but is actually a
    Place).
  Precision = 2 / (2+1) = 0.6667 or 66.67%
  Recall = 2 / (2+1) = 0.6667 or 66.67%
  F1 Score = 2 * 0.6667 * 0.6667 / (0.6667 + 0.6667) = 0.6667 or  66.67%
```

この場合のグローバルメトリクスは次のようになります。

グローバル:

```
Global:
  TP = 3 (because John Washington, San Francisco and San Diego are predicted
 correctly.
    This is also the sum of all individual entity TP).
  FP = 2 (because Seattle is predicted as Person and Smith is predicted as Place. This
    is the sum of all individual entity FP).
  FN = 2 (because Seattle is predicted as Person and Smith is predicted as Place. This
    is the sum of all individual FN).
  Global Precision = 3 / (3+2) = 0.6 or 60%
    (Global Precision = Global TP / (Global TP + Global FP))
  Global Recall = 3 / (3+2) = 0.6 or 60%
    (Global Recall = Global TP / (Global TP + Global FN))
  Global F1Score = 2 * 0.6 * 0.6 / (0.6 + 0.6) = 0.6 or 60%
    (Global F1Score = 2 * Global Precision *  Global Recall / (Global Precision +
    Global Recall))
```

カスタムエンティティレコグナイザーのパフォーマンスの向上

これらのメトリクスから、トレーニング済みモデルを使用してエンティティを識別したときに、そのモデルがどの程度正確に機能するかについての洞察が得られます。メトリクスが予想よりも低い場合に、メトリクスを改善するために使用できるオプションをいくつか紹介します。

1. 「注釈」または「エンティティリスト (プレーンテキストのみ)」を使用するかどうかに応じて、データ品質を向上させるために、それぞれのドキュメントのガイドラインに必ず従ってください。データを改善して、モデルを再トレーニングした結果、より良いメトリクスが得られた場合は、繰り返しデータ品質を改善してモデルのパフォーマンスを向上させることができます。
2. エンティティリストを使用している場合は、代わりに注釈の使用を検討してください。通常、手動でコメントを付けると結果が改善されます。
3. データ品質に問題がないと判断しても、メトリクスが不当に低い場合は、サポートリクエストを提出してください。

リアルタイムのカスタムレコグナイザー分析を実行する

リアルタイム分析は、小さなドキュメントが到着したときに処理するアプリケーションに役立ちます。たとえば、ソーシャルメディアへの投稿、サポートチケット、カスタマーレビューに含まれるカスタムエンティティを検出できます。

> **開始する前に**
>
> カスタムエンティティを検出するには、カスタムエンティティ認識モデル (レコグナイザーとも呼ばれる) が必要です。これらのモデルの詳細については、「the section called “トレーニングレコグナイザーのモデル”」を参照してください。

プレーンテキストの注釈でトレーニングされたレコグナイザーは、プレーンテキストドキュメントのエンティティ検出のみをサポートします。PDF ドキュメントの注釈でトレーニングされたレコグナイザーは、プレーンテキストドキュメント、画像、PDF ファイル、Word ドキュメントのエンティティ検出をサポートします。入力ファイルの詳細については、「リアルタイムカスタム分析用の入力」を参照してください。

画像ファイルまたはスキャンされた PDF ドキュメントを分析する場合は、IAM ポリシーで 2 つの Amazon Textract API メソッド (DetectDocumentText および AnalyzeDocument) を使用するアクセス許可を付与する必要があります。Amazon Comprehend は、テキスト抽出中にこれらのメソッドを

呼び出します。ポリシーの例については、「ドキュメント分析アクションを実行するために必要なアクセス許可」を参照してください。

トピック

- カスタムエンティティ認識のリアルタイム分析 (コンソール)
- カスタムエンティティ認識でのリアルタイム分析 (API)
- リアルタイム分析の出力

カスタムエンティティ認識のリアルタイム分析 (コンソール)

Amazon Comprehend コンソールを使用して、カスタムモデルを使用したリアルタイム分析を実行できます。まずリアルタイム分析を実行するエンドポイントを作成します。エンドポイントの作成が完了したなら、リアルタイム分析を実行します。

エンドポイントのスループットのプロビジョニングとそれに関連するコストについては、「Amazon Comprehend エンドポイントの使用法」を参照してください。

トピック

- カスタムエンティティ検出用のエンドポイントの作成
- カスタムエンティティ検出のリアルタイム実行

カスタムエンティティ検出用のエンドポイントの作成

エンドポイントを作成するには (コンソール)

1. AWS Management Console にサインインして、Amazon Comprehend コンソール (https://console. .amazon.com/comprehend/) を開きます
2. 左側のメニューから [エンドポイント] を選択し、[エンドポイントの作成] ボタンを選択します。「エンドポイントの作成」画面が開きます。
3. エンドポイントに名前を付けます。名前は、自分のアカウント内と現在のリージョンで一意でなければなりません。
4. 新しいエンドポイントをアタッチするカスタムモデルを選択します。ドロップダウンから、モデル名で検索できます。

> **Note**
>
> エンドポイントをモデルにアタッチする前に、モデルを作成する必要があります。まだモデルがない場合は、「カスタムエンティティレコグナイザーモデルのトレーニング」を参照してください。

5. (オプション) エンドポイントにタグを追加するには、[タグ] にキーと値のペアを入力し、[タグを追加] を選択します。分類子を作成する前にこのペアを削除するには、[タグを削除] を選択します。
6. エンドポイントに割り当てる推論単位 (IU) の数を入力します。各単位は、1 秒あたり最大 2 つの文書に対して 100 文字/秒のスループットを表します。エンドポイントスループットの詳細については、「Amazon Comprehend エンドポイントの使用法」を参照してください。
7. (オプション) 新しいエンドポイントを作成する場合は、IU Estimator を使用することもできます。見積ツールは、リクエストする IU の数を判断するのに役立ちます。推論ユニットの数は、スループットまたは 1 秒あたりの分析文字数によって異なります。
8. 購入概要から、時間単位、日単位、月単位の推定エンドポイントコストを確認します。
9. 起動から削除までの間、エンドポイントの料金が発生することを了解している場合は、このチェックボックスを選択してください。
10. [エンドポイントの作成] を選択します。

カスタムエンティティ検出のリアルタイム実行

カスタムエンティティ認識モデルのエンドポイントを作成したなら、リアルタイム分析を実行して個々の文書内のエンティティを検出できます。

以下のステップを完了して、Amazon Comprehend コンソールを使用して、テキスト内のカスタムエンティティを検出します。

1. AWS Management Console にサインインして、Amazon Comprehend コンソール (https://console. .amazon.com/comprehend/) を開きます
2. 左側のメニューで、[リアルタイム分析] を選択します。
3. [入力テキスト] セクションの [分析タイプ] で [カスタム] を選択します。
4. [エンドポイントの選択] で、使用するエンティティ検出モデルに関連付けられているエンドポイントを選択します。
5. 分析用の入力データを指定するには、テキストを入力するか、ファイルをアップロードします。

- テキストを入力するには

  a. 入力テキストを選択します。

  b. 分析するテキストを入力します。

- ファイルをアップロードするには、

  a. [ファイルをアップロード] を選択し、アップロードするファイル名を入力します。

  b. (オプション) 文書読み取りモードでは、デフォルトのテキスト抽出アクションをオーバーライドできます。詳細については、「テキスト抽出オプションの設定」を参照してください。

6. [分析] を選択します。コンソールには、分析の出力と信頼性評価が表示されます。

カスタムエンティティ認識でのリアルタイム分析 (API)

Amazon Comprehend API では、カスタムモデルを使用してリアルタイム分析を実行できます。最初に、リアルタイム分析を実行するためのエンドポイントを作成します。エンドポイントの作成が完了したなら、リアルタイム分析を実行します。

エンドポイントのスループットのプロビジョニングとそれに関連するコストについては、「Amazon Comprehend エンドポイントの使用法」を参照してください。

トピック

- カスタムエンティティ検出用のエンドポイントの作成
- カスタムエンティティ検出のリアルタイム実行

カスタムエンティティ検出用のエンドポイントの作成

プロキシエンドポイントに関係するコストについては、「Amazon Comprehend エンドポイントの使用法」を参照してください。

AWS CLI を使用したエンドポイントの作成

AWS CLI を使用してエンドポイントサービスを作成するには、`create-endpoint` コマンドを使用します。

```
$ aws comprehend create-endpoint \
> --desired-inference-units number of inference units \
```

```
> --endpoint-name endpoint name \
> --model-arn arn:aws:comprehend:region:account-id:model/example \
> --tags Key=Key,Value=Value
```

コマンドが成功すると、Amazon Comprehend はその応答としてエンドポイントの ARN を返します。

```
{
   "EndpointArn": "Arn"
}
```

このコマンドとそのパラメータ引数、出力の詳細については、『AWS CLI コマンドリファレンス』の「create-endpoint」を参照してください。

カスタムエンティティ検出のリアルタイム実行

カスタムエンティティレコグナイザーモデルのエンドポイントを作成したら、エンドポイントを使用して DetectEntities API オペレーションを実行します。text または bytes のパラメータを使用してテキストを入力できます。その他の種類の入力には、bytes パラメータを利用します。

画像および PDF ファイルの場合は、DocumentReaderConfig パラメーターを使用してデフォルトのテキスト抽出アクションをオーバーライドできます。詳細については、「テキスト抽出オプションの設定」を参照してください。

AWS CLI を使用したテキスト内のエンティティの検出

テキスト内のカスタムエンティティを検出するには、text パラメータに入力テキストを指定して detect-entities コマンド実行します。

Example CLI を使用して入力テキスト内のエンティティを検出する

```
$ aws comprehend detect-entities \
> --endpoint-arn arn \
> --language-code en \
> --text  "Andy Jassy is the CEO of Amazon."
```

コマンドの実行に成功すると、Amazon Comprehend はその応答として分析を開始します。Amazon Comprehend は、検出したエンティティごとにそのエンティティタイプとテキスト、場所、および信頼スコアを提供します。

AWS CLI を使用した半構造化ドキュメント内のエンティティの検出

PDF や Word、あるいは画像ファイル内のカスタムエンティティを検出するには、bytes パラメータに入力ファイルを指定して、detect-entities コマンドを実行します。

Example : CLI を使用してイメージファイル内のエンティティを検出する

この例では、fileb オプションを使用して画像ファイルを渡して、画像のバイトをbase64 でエンコードする方法を示します。詳細については、『AWS Command Line Interface ユーザーガイド』の「バイナリラージオブジェクト」を参照してください。

この例では、テキスト抽出オプションを設定するために config.json という名前の JSON ファイルも渡しています。

```
$ aws comprehend detect-entities \
> --endpoint-arn arn \
> --language-code en \
> --bytes fileb://image1.jpg   \
> --document-reader-config file://config.json
```

config.json ファイルには次のコンテンツが含まれます。

```
{
   "DocumentReadMode": "FORCE_DOCUMENT_READ_ACTION",
   "DocumentReadAction": "TEXTRACT_DETECT_DOCUMENT_TEXT"
}
```

コマンド構文の詳細については、「Amazon Comprehend API リファレンスDetectEntities」の「」を参照してください。

リアルタイム分析の出力

テキスト入力の出力

Text パラメータを使用してテキストを入力した場合、出力は解析で検出された多数のエンティティで構成されます。以下は、2 つの JUDGE エンティティを検出した分析例を示しています。

```
{
```

```
        "Entities":
        [
            {
                "BeginOffset": 0,
                "EndOffset": 22,
                "Score": 0.9763959646224976,
                "Text": "John Johnson",
                "Type": "JUDGE"
            },
            {
                "BeginOffset": 11,
                "EndOffset": 15,
                "Score": 0.9615424871444702,
                "Text": "Thomas Kincaid",
                "Type": "JUDGE"
            }
        ]
    }
```

半構造化入力の出力

半構造化入力ドキュメントまたはテキストファイルの場合、出力には以下の追加フィールドが含まれる場合があります。

- DocumentMetadata - ドキュメントに関する抽出情報。メタデータには、ドキュメント内のページのリストと、各ページから抽出された文字数が含まれます。リクエストに Byte パラメータがあると、このフィールドが応答に含まれます。
- DocumentType – 入力ドキュメントの各ページのドキュメントタイプ。リクエストに Byte パラメータがある場合、レスポンスにはこのフィールドが含まれます。
- Blocks — 入力ドキュメントのテキストの各ブロックに関する情報。Blocks は入れ子の形式になっています。　1 つのページブロックはテキスト行ごとの 1 つのブロックで構成され、このブロックは単語ごとに 1 つのブロックで構成されます。リクエストに Byte パラメータがある場合、レスポンスにはこのフィールドが含まれます。
- BlockReferences – このエンティティの各ブロックへの参照。リクエストに Byte パラメータがある場合、レスポンスにはこのフィールドが含まれます。テキストファイルの場合、このフィールドは存在しません。
- エラー — 入力文書の処理中にシステムが検出したページレベルのエラー。エラーが検出されなかった場合、このフィールドは空です。

これらの出力フィールドの説明については、DetectEntities「Amazon Comprehend API リファレンス」の「」を参照してください。レイアウト要素の詳細については、『Amazon Textract デベロッパーガイド』の「Amazon Textract 分析オブジェクト」を参照してください。

次の例は、スキャンされた 1 ページの PDF 入力ドキュメントの出力例です。


```
{
    "Entities": [{
        "Score": 0.9984670877456665,
        "Type": "DATE-TIME",
        "Text": "September 4,",
        "BlockReferences": [{
            "BlockId": "42dcaaee-c484-4b5d-9e3f-ae0be928b3e1",
            "BeginOffset": 0,
            "EndOffset": 12,
            "ChildBlocks": [{
                    "ChildBlockId": "6e9cbb43-f8be-4da0-9a4b-ff9a6c350a14",
                    "BeginOffset": 0,
                    "EndOffset": 9
                },
                {
                    "ChildBlockId": "599e0d53-ae9f-491b-a762-459b22c79ff5",
                    "BeginOffset": 0,
                    "EndOffset": 2
                },
                {
                    "ChildBlockId": "599e0d53-ae9f-491b-a762-459b22c79ff5",
                    "BeginOffset": 0,
                    "EndOffset": 2
                }
            ]
        }]
    }],
    "DocumentMetadata": {
        "Pages": 1,
        "ExtractedCharacters": [{
            "Page": 1,
            "Count": 609
        }]
    },
    "DocumentType": [{
        "Page": 1,
        "Type": "SCANNED_PDF"
```

```
    }],
    "Blocks": [{
        "Id": "ee82edf3-28de-4d63-8883-40e2e4938ccb",
        "BlockType": "LINE",
        "Text": "Your Band",
        "Page": 1,
        "Geometry": {
            "BoundingBox": {
                "Height": 0.024125460535287857,
                "Left": 0.11745482683181763,
                "Top": 0.06821706146001816,
                "Width": 0.12074867635965347
            },
            "Polygon": [{
                    "X": 0.11745482683181763,
                    "Y": 0.06821706146001816
                },
                {
                    "X": 0.2382034957408905,
                    "Y": 0.06821706146001816
                },
                {
                    "X": 0.2382034957408905,
                    "Y": 0.09234252572059631
                },
                {
                    "X": 0.11745482683181763,
                    "Y": 0.09234252572059631
                }
            ]
        },
        "Relationships": [{
            "Ids": [
                "b105c561-c8d9-485a-a728-7a5b1a308935",
                "60ecb119-3173-4de2-8c5d-de182a5f86a5"
            ],
            "Type": "CHILD"
        }]
    }]
}
```


次の例は、ネイティブ PDF ドキュメントの解析の出力例です。

Example PDF ドキュメントのカスタムエンティティ認識分析の出力例


```
{
        "Blocks":
        [
            {
                "BlockType": "LINE",
                "Geometry":
                {
                    "BoundingBox":
                    {
                        "Height": 0.012575757575757575,
                        "Left": 0.0,
                        "Top": 0.0015063131313131314,
                        "Width": 0.02262091503267974
                    },
                    "Polygon":
                    [
                        {
                            "X": 0.0,
                            "Y": 0.0015063131313131314
                        },
                        {
                            "X": 0.02262091503267974,
                            "Y": 0.0015063131313131314
                        },
                        {
                            "X": 0.02262091503267974,
                            "Y": 0.014082070707070706
                        },
                        {
                            "X": 0.0,
                            "Y": 0.014082070707070706
                        }
                    ]
                },
                "Id": "4330efed-6334-4fc4-ba48-e050afa95c8d",
                "Page": 1,
                "Relationships":
                [
                    {
                        "ids":
                        [
                            "f343ce48-583d-4abe-b84b-a232e266450f"
```

```
                    ],
                    "type": "CHILD"
                }
            ],
            "Text": "S-3"
        },
        {
            "BlockType": "WORD",
            "Geometry":
            {
                "BoundingBox":
                {
                    "Height": 0.012575757575757575,
                    "Left": 0.0,
                    "Top": 0.0015063131313131314,
                    "Width": 0.02262091503267974
                },
                "Polygon":
                [
                    {
                        "X": 0.0,
                        "Y": 0.0015063131313131314
                    },
                    {
                        "X": 0.02262091503267974,
                        "Y": 0.0015063131313131314
                    },
                    {
                        "X": 0.02262091503267974,
                        "Y": 0.014082070707070706
                    },
                    {
                        "X": 0.0,
                        "Y": 0.014082070707070706
                    }
                ]
            },
            "Id": "f343ce48-583d-4abe-b84b-a232e266450f",
            "Page": 1,
            "Relationships":
            [],
            "Text": "S-3"
        }
    ],
```

```
        "DocumentMetadata":
        {
            "PageNumber": 1,
            "Pages": 1
        },
        "DocumentType": "NativePDF",
        "Entities":
        [
            {
                "BlockReferences":
                [
                    {
                        "BeginOffset": 25,
                        "BlockId": "4330efed-6334-4fc4-ba48-e050afa95c8d",
                        "ChildBlocks":
                        [
                            {
                                "BeginOffset": 1,
                                "ChildBlockId": "cbba5534-ac69-4bc4-beef-306c659f70a6",
                                "EndOffset": 6
                            }
                        ],
                        "EndOffset": 30
                    }
                ],
                "Score": 0.9998825926329088,
                "Text": "0.001",
                "Type": "OFFERING_PRICE"
            },
            {
                "BlockReferences":
                [
                    {
                        "BeginOffset": 41,
                        "BlockId": "f343ce48-583d-4abe-b84b-a232e266450f",
                        "ChildBlocks":
                        [
                            {
                                "BeginOffset": 0,
                                "ChildBlockId": "292a2e26-21f0-401b-a2bf-03aa4c47f787",
                                "EndOffset": 9
                            }
                        ],
                        "EndOffset": 50
```

```
                }
            ],
            "Score": 0.9809727537330395,
            "Text": "6,097,560",
            "Type": "OFFERED_SHARES"
        }
    ],
    "File": "example.pdf",
    "Version": "2021-04-30"
}
```

カスタムエンティティ認識の分析ジョブの実行

非同期分析ジョブを実行して、1 つ以上のドキュメントセット内のカスタムエンティティを検出できます。

> **開始する前に**
>
> カスタムエンティティを検出するには、カスタムエンティティ認識モデル (レコグナイザーとも呼ばれる) が必要です。これらのモデルの詳細については、「the section called “トレーニングレコグナイザーのモデル”」を参照してください。

プレーンテキストの注釈でトレーニングされたレコグナイザーは、プレーンテキストドキュメントのエンティティ検出のみをサポートします。PDF ドキュメントの注釈でトレーニングされたレコグナイザーは、プレーンテキストドキュメント、画像、PDF ファイル、Word ドキュメントのエンティティ検出をサポートします。テキストファイル以外のファイルについては、Amazon Comprehend は分析を実行する前にテキスト抽出を実行します。入力ファイルの詳細については、「非同期カスタム分析の入力 」を参照してください。

画像ファイルまたはスキャンされた PDF ドキュメントを分析する場合は、IAM ポリシーで 2 つの Amazon Textract API メソッド (DetectDocumentText および AnalyzeDocument) を使用するアクセス許可を付与する必要があります。Amazon Comprehend は、テキスト抽出中にこれらのメソッドを呼び出します。ポリシーの例については、「 ドキュメント分析アクションを実行するために必要なアクセス許可」を参照してください。

非同期分析ジョブの実行には、次のステップを実行します。

1. ドキュメントを Amazon S3 バケットに保存します。

2. API またはコンソールを使用して分析ジョブを開始します。
3. 分析ジョブの進行状況をモニタリングします。
4. ジョブの実行が完了するまで、ジョブ開始時に指定した S3 バケットを確認します。

トピック

- カスタムエンティティ検出ジョブの開始 (コンソール)
- カスタムエンティティ検出ジョブの開始 (API)
- 非同期分析ジョブの出力

カスタムエンティティ検出ジョブの開始 (コンソール)

コンソールを使用して、カスタムエンティティ認識の非同期分析ジョブを開始・監視できます。

非同期分析ジョブを開始するには:

1. AWS Management Console にサインインして、Amazon Comprehend コンソール (https://console. .amazon.com/comprehend/) を開きます
2. 左側のメニューから、[分析ジョブ] を選択し、[ジョブの作成] を選択します。
3. 分類ジョブに名前を付けます。この名前は、自分のアカウントと現在のリージョンで一意でなければなりません。
4. [分析タイプ] で [カスタムエンティティ認識] を選択します。
5. [レコグナイザーモデル] から、使用するカスタムエンティティレコグナイザーを選択します。
6. [バージョン] から、使用するレコグナイザーのバージョンを選択します。
7. (オプション) Amazon Comprehend がジョブの処理中に使用するデータの暗号化を選択する場合は、[ジョブの暗号化] を選択します。次に、現在のアカウントに関連付けられた KMS キーを使用するか、別のアカウントの KMS キーを使用するかを選択します。
   - 現在のアカウントに関連付けられているキーを使用している場合は、KMS キー ID のキー ID を選択します。
   - 別のアカウントに関連付けられているキーを使用している場合は、KMS キー ARN の下にキー ID の ARN を入力します。

> **Note**
>
> KMS キーの作成と使用や関連する暗号化の詳細については、「キー管理サービス (KMS)」を参照してください。

8. [入力データ] で、入力文書を含む Amazon S3 バケットの場所を入力するか、[S3 を参照] を選択してその場所に移動します。このバケットは、呼び出している API と同じリージョン内になければなりません。分析ジョブのアクセス許可に使用する IAM ロールには、S3 バケットに対する読み取り許可が必要です。
9. (オプション) 入力形式では、入力ドキュメントの形式を選択できます。形式は、ファイルごとに 1 文書にすることも、1 つのファイルの 1 行に 1 文書にすることもできます。1 行に 1 つの文書が適用されるのはテキスト文書だけです。
10. (オプション) 文書読み取りモードでは、デフォルトのテキスト抽出アクションをオーバーライドできます。詳細については、「テキスト抽出オプションの設定」を参照してください。
11. [出力データ] で、Amazon Comprehend がジョブの出力データを書き込む Amazon S3 バケットの場所を入力するか、[S3 を参照] を選択してその場所に移動します。このバケットは、呼び出している API と同じリージョン内になければなりません。分類ジョブのアクセス許可に使用する IAM ロールには、S3 バケットに対する読み取り許可が必要です。
12. (オプション) ジョブの出力結果を暗号化する場合は、[暗号化] を選択します。次に、現在のアカウントに関連付けられた KMS キーを使用するか、別のアカウントの KMS キーを使用するかを選択します。

    - 現在のアカウントに関連付けられているキーを使用している場合は、KMS キー ID のキーエイリアスまたは ID を選択します。
    - 別のアカウントに関連付けられているキーを使用している場合は、KMS キー ID の下にキーエイリアス の ARN または ID を入力します。
13. (オプション) VPC から Amazon Comprehend にリソースを起動するには、VPC の下に VPC ID を入力するか、ドロップダウンリストから ID を選択します。

    1. [サブネット] でサブネットを選択します。最初のサブネットを選択すると、追加のサブネットを選択できます。
    2. セキュリティグループを指定した場合は、[セキュリティグループ] で、使用するセキュリティグループを選択します。最初のセキュリティグループを選択すると、追加のセキュリティグループを選択できます。

> **Note**
>
> 分析ジョブで VPC を使用する場合、[作成] と [開始] 操作に使用する DataAccessRole には、出力バケットへの VPC アクセス権限が必要です。

14. [ジョブの作成] を選択してエンティティ認識ジョブを作成します。

カスタムエンティティ検出ジョブの開始 (API)

API を使用して、カスタムエンティティ認識の非同期分析ジョブを開始・監視できます。

StartEntitiesDetectionJob オペレーションでカスタムエンティティ検出ジョブを開始するには、トレーニング済みモデルの Amazon リソースネーム (ARN) EntityRecognizerArnである を指定します。この ARN は、 CreateEntityRecognizerオペレーションへのレスポンスで確認できます。

トピック

- を使用したカスタムエンティティの検出 AWS Command Line Interface
- AWS SDK for Javaを使用したカスタムエンティティの検出
- を使用したカスタムエンティティの検出 AWS SDK for Python (Boto3)
- PDF ファイルに対する API アクションのオーバーライド

を使用したカスタムエンティティの検出 AWS Command Line Interface

次の例は、Unix、Linux および macOS 環境向けです。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。ドキュメントセットにあるカスタムエンティティを検出するには、以下のリクエスト構文を使用します。

```
aws comprehend start-entities-detection-job \
     --entity-recognizer-arn "arn:aws:comprehend:region:account number:entity-
recognizer/test-6" \
     --job-name infer-1 \
     --data-access-role-arn "arn:aws:iam::account number:role/service-role/
AmazonComprehendServiceRole-role" \
     --language-code en \
     --input-data-config "S3Uri=s3://Bucket Name/Bucket Path" \
     --output-data-config "S3Uri=s3://Bucket Name/Bucket Path/" \
```

```
    --region region
```

Amazon Comprehendは、応答として JobID および JobStatus を返すと共に、リクエストで指定された S3 バケット内のジョブの出力を返します。

AWS SDK for Javaを使用したカスタムエンティティの検出

Java を使用した Amazon Comprehend の例については、「Amazon Comprehend の Java の例」を参照してください。

を使用したカスタムエンティティの検出 AWS SDK for Python (Boto3)

この例では、カスタムエンティティレコグナイザーを作成して、モデルをトレーニングしてから、AWS SDK for Python (Boto3)を使用してエンティティレコグナイザジョブで実行します。

SDK 用 Python のインスタンスを作成します。

```
import boto3
import uuid
comprehend = boto3.client("comprehend", region_name="region")
```

エンティティレコグナイザーを作成する:

```
response = comprehend.create_entity_recognizer(
    RecognizerName="Recognizer-Name-Goes-Here-{}".format(str(uuid.uuid4())),
    LanguageCode="en",
    DataAccessRoleArn="Role ARN",
    InputDataConfig={
        "EntityTypes": [
            {
                "Type": "ENTITY_TYPE"
            }
        ],
        "Documents": {
            "S3Uri": "s3://Bucket Name/Bucket Path/documents"
        },
        "Annotations": {
            "S3Uri": "s3://Bucket Name/Bucket Path/annotations"
        }
    }
)
```

```
recognizer_arn = response["EntityRecognizerArn"]
```

すべてのレコグナイザーを一覧表示する:

```
response = comprehend.list_entity_recognizers()
```

エンティティレコグナイザーが TRAINED ステータスになるのを待つ。

```
while True:
    response = comprehend.describe_entity_recognizer(
        EntityRecognizerArn=recognizer_arn
    )

    status = response["EntityRecognizerProperties"]["Status"]
    if "IN_ERROR" == status:
        sys.exit(1)
    if "TRAINED" == status:
        break

    time.sleep(10)
```

カスタムエンティティ検出ジョブを開始する:

```
response = comprehend.start_entities_detection_job(
    EntityRecognizerArn=recognizer_arn,
    JobName="Detection-Job-Name-{}".format(str(uuid.uuid4())),
    LanguageCode="en",
    DataAccessRoleArn="Role ARN",
    InputDataConfig={
        "InputFormat": "ONE_DOC_PER_LINE",
        "S3Uri": "s3://Bucket Name/Bucket Path/documents"
    },
    OutputDataConfig={
        "S3Uri": "s3://Bucket Name/Bucket Path/output"
    }
)
```

PDF ファイルに対する API アクションのオーバーライド

画像ファイルおよび PDF ファイルの場合は、InputDataConfig で DocumentReaderConfig パラメータを使用してデフォルトのテキスト抽出アクションをオーバーライドできます。

次の例では、 myInputDataConfig.json という名前の JSON ファイルを定義してInputDataConfig値を設定します。すべての PDF ファイルに対して Amazon Textract DetectDocumentText API を使用するように DocumentReadConfig を設定します。

Example

```
"InputDataConfig": {
  "S3Uri": s3://Bucket Name/Bucket Path",
  "InputFormat": "ONE_DOC_PER_FILE",
  "DocumentReaderConfig": {
      "DocumentReadAction": "TEXTRACT_DETECT_DOCUMENT_TEXT",
      "DocumentReadMode": "FORCE_DOCUMENT_READ_ACTION"
  }
}
```

StartEntitiesDetectionJob オペレーションで、 myInputDataConfig.json ファイルをInputDataConfigパラメータとして指定します。

```
  --input-data-config file://myInputDataConfig.json
```

DocumentReaderConfig パラメータの詳細については、「 テキスト抽出オプションの設定」を参照してください。

非同期分析ジョブの出力

分析ジョブが完了すると、リクエストに指定した S3 バケットに結果が保存されます。

テキスト入力の出力

テキスト入力ファイルの場合、出力は各入力ドキュメントのエンティティのリストで構成されます。

次の例は、 50_docs という名前の入力ファイルにある 2 つのドキュメントに対する出力を示しています。1 行に 1 つのドキュメントの形式です。

```
{
        "File": "50_docs",
        "Line": 0,
        "Entities":
        [
            {
                "BeginOffset": 0,
```

```
                "EndOffset": 22,
                "Score": 0.9763959646224976,
                "Text": "John Johnson",
                "Type": "JUDGE"
            }
        ]
    }
    {
        "File": "50_docs",
        "Line": 1,
        "Entities":
        [
            {
                "BeginOffset": 11,
                "EndOffset": 15,
                "Score": 0.9615424871444702,
                "Text": "Thomas Kincaid",
                "Type": "JUDGE"
            }
        ]
    }
```

半構造化入力の出力

半構造化入力ドキュメントの場合、出力には以下の追加フィールドが含まれる場合があります。

- DocumentMetadata - ドキュメントに関する抽出情報。メタデータには、ドキュメント内のページのリストと、各ページから抽出された文字数が含まれます。リクエストに Byte パラメータがあると、このフィールドが応答に含まれます。
- DocumentType – 入力ドキュメントの各ページのドキュメントタイプ。リクエストに Byte パラメータがある場合、レスポンスにはこのフィールドが含まれます。
- Blocks — 入力ドキュメントのテキストの各ブロックに関する情報。1つのブロックに、ブロックが入れ子になっている場合があります。1 つのページブロックはテキスト行ごとの 1 つのブロックで構成され、このブロックは単語ごとに 1 つのブロックで構成されます。リクエストに Byte パラメータがある場合、レスポンスにはこのフィールドが含まれます。
- BlockReferences – このエンティティの各ブロックへの参照。リクエストに Byte パラメータがある場合、レスポンスにはこのフィールドが含まれます。テキストファイルの場合、このフィールドは存在しません。
- エラー — 入力文書の処理中にシステムが検出したページレベルのエラー。エラーが検出されなかった場合、このフィールドは空です。

これらの出力フィールドの詳細については、「Amazon Comprehend API リファレンスDetectEntities」の「」を参照してください。

次の例は、1 ページのネイティブ PDF 入力文書の出力例です。

Example PDF ドキュメントのカスタムエンティティ認識分析の出力例


```
{
        "Blocks":
        [
            {
                "BlockType": "LINE",
                "Geometry":
                {
                    "BoundingBox":
                    {
                        "Height": 0.012575757575757575,
                        "Left": 0.0,
                        "Top": 0.0015063131313131314,
                        "Width": 0.02262091503267974
                    },
                    "Polygon":
                    [
                        {
                            "X": 0.0,
                            "Y": 0.0015063131313131314
                        },
                        {
                            "X": 0.02262091503267974,
                            "Y": 0.0015063131313131314
                        },
                        {
                            "X": 0.02262091503267974,
                            "Y": 0.014082070707070706
                        },
                        {
                            "X": 0.0,
                            "Y": 0.014082070707070706
                        }
                    ]
                },
                "Id": "4330efed-6334-4fc4-ba48-e050afa95c8d",
                "Page": 1,
                "Relationships":
```

```
                [
                    {
                        "ids":
                        [
                            "f343ce48-583d-4abe-b84b-a232e266450f"
                        ],
                        "type": "CHILD"
                    }
                ],
                "Text": "S-3"
            },
            {
                "BlockType": "WORD",
                "Geometry":
                {
                    "BoundingBox":
                    {
                        "Height": 0.012575757575757575,
                        "Left": 0.0,
                        "Top": 0.0015063131313131314,
                        "Width": 0.02262091503267974
                    },
                    "Polygon":
                    [
                        {
                            "X": 0.0,
                            "Y": 0.0015063131313131314
                        },
                        {
                            "X": 0.02262091503267974,
                            "Y": 0.0015063131313131314
                        },
                        {
                            "X": 0.02262091503267974,
                            "Y": 0.014082070707070706
                        },
                        {
                            "X": 0.0,
                            "Y": 0.014082070707070706
                        }
                    ]
                },
                "Id": "f343ce48-583d-4abe-b84b-a232e266450f",
                "Page": 1,
```

```
                "Relationships":
                [],
                "Text": "S-3"
            }
        ],
        "DocumentMetadata":
        {
            "PageNumber": 1,
            "Pages": 1
        },
        "DocumentType": "NativePDF",
        "Entities":
        [
            {
                "BlockReferences":
                [
                    {
                        "BeginOffset": 25,
                        "BlockId": "4330efed-6334-4fc4-ba48-e050afa95c8d",
                        "ChildBlocks":
                        [
                            {
                                "BeginOffset": 1,
                                "ChildBlockId": "cbba5534-ac69-4bc4-beef-306c659f70a6",
                                "EndOffset": 6
                            }
                        ],
                        "EndOffset": 30
                    }
                ],
                "Score": 0.9998825926329088,
                "Text": "0.001",
                "Type": "OFFERING_PRICE"
            },
            {
                "BlockReferences":
                [
                    {
                        "BeginOffset": 41,
                        "BlockId": "f343ce48-583d-4abe-b84b-a232e266450f",
                        "ChildBlocks":
                        [
                            {
                                "BeginOffset": 0,
```

```
                            "ChildBlockId": "292a2e26-21f0-401b-a2bf-03aa4c47f787",
                            "EndOffset": 9
                        }
                    ],
                    "EndOffset": 50
                }
            ],
            "Score": 0.9809727537330395,
            "Text": "6,097,560",
            "Type": "OFFERED_SHARES"
        }
    ],
    "File": "example.pdf",
    "Version": "2021-04-30"
}
```

カスタムモデルの作成と管理

Amazon Comprehend には、インサイト分析やトピックモデリングに利用できる NLP (自然言語処理) モデルが内蔵されています。Amazon Comprehend を使用して、カスタムエンティティ認識およびドキュメント分ル用のカスタムモデルを作成することもできます。

モデルのバージョニングでは、モデルの履歴を追跡できます。新しいモデルバージョンを作成してトレーニングすると、トレーニングデータセットに変更を加えることができます。Amazon Comprehend は、モデルの詳細ページにモデルの各バージョンの詳細 (モデルのパフォーマンスを含む) を表示します。時間の経過と共に、トレーニングデータセットに変更を加えることでモデルパフォーマンスがどのように変化するのかを確認できます。

モデルのバージョンは、Amazon Comprehend コンソールまたは API を使用して作成できます。もう一つ方法として、Amazon Comprehend に フライホイール があり、新しいカスタムモデルのバージョンのトレーニングと評価に関連するタスクを簡素化できます。

カスタムモデルを作成したら、他の AWS アカウント がモデルのコピーをインポートできるようにすることで、他のユーザーとモデルを共有できます。

トピック

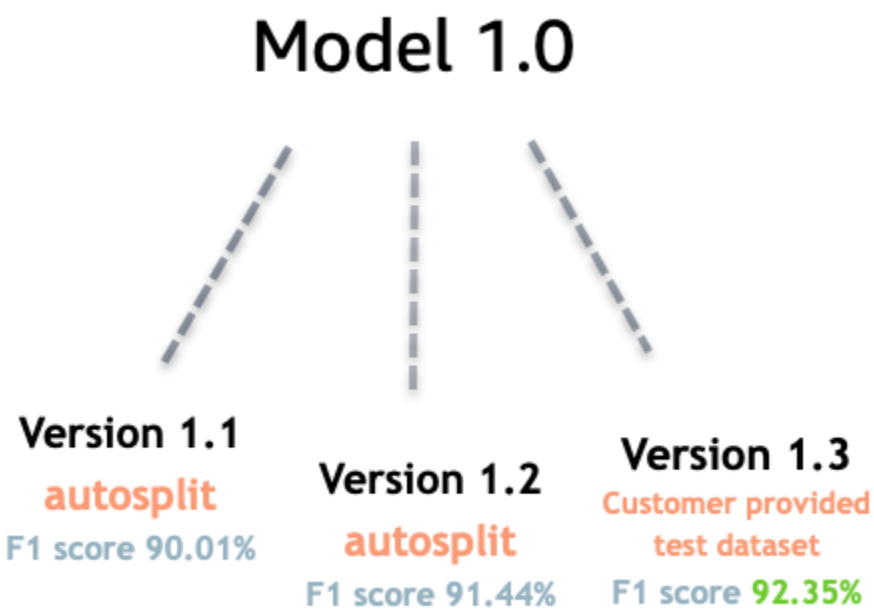
- Amazon Comprehend によるモデルバージョニング
- AWS アカウント の間でのカスタムモデルのコピー

Amazon Comprehend によるモデルバージョニング

人工知能と機械学習 (AI/ML) は、実験の迅速さがすべてです。Amazon Comprehend では、モデルをトレーニングして構築し、データに関する洞察を得ることができます。モデルのバージョニングを使用すると、提供するデータセットが増えたり変わったりといった、モデルの実行結果に関連するモデリング履歴とスコアを追跡できます。バージョニングは、カスタム分類モデルまたはカスタムエンティティ認識モデルで利用できます。時間の経過と共に変わるバージョンを観察することで、そのバージョンがどの程度成功したか、また、成功状態に至るまでにどのようなパラメータを使用したかの洞察を得ることができます。

既存のカスタム分類子モデルまたはエンティティ認識モデルの新しいバージョンのトレーニングでは、モデルの詳細ページから新しいバージョンを作成するだけで、すべての詳細情報が自動的に入力されます。新しいバージョンには以前のモデルと同じ名前 (VersionID と言う) が付けられますが、作成時には固有のバージョン名を付けることができます。モデルに新しいバージョンを追加すると、モ

デルの詳細ページから以前のすべてのバージョンとその詳細を 1 つのビューで確認できます。バージョニングでは、トレーニングデータセットに変更を加えたときにモデルのパフォーマンスがどのように変化するかを確認できます。



新しいカスタム分類子バージョン (コンソール) を作成する

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/comprehend/ で Amazon Comprehend コンソールを開きます。
2. 左側のメニューから [カスタマイズ] を選択し、[カスタム分類] を選択します。
3. [分類子] の一覧から、新しいバージョンを作成するカスタムモデルの名前を選択します。カスタムモデルの詳細 ページが表示されます。
4. 右上の [モデルの新規作成] を選択します。親カスタム分類モデルの詳細があらかじめ入力されている画面が開きます。
5. [バージョン名] で、新しいバージョンに一意の名前を追加します。
6. バージョン詳細で、新しいモデルに関連付ける言語とラベル数を変更できます。
7. [データ仕様] セクションで、新しいバージョンへのデータの提供方法を設定します。以前のモデルのドキュメントと新しいドキュメントを含む完全なデータを必ず提供してください。分類子モード (シングルラベルまたはマルチラベル)、データ形式 (CSV ファイル、拡張マニフェス

ト)、トレーニングデータセット、テストデータセット (自動分割、またはカスタムテストデータ設定) を変更できます。

8. (オプション) 出力データの S3 上の場所を更新する
9. [アクセス権限] で、IAM ロールを作成するか、既存の IAM ロールを使用します。
10. (オプション) VPC 設定を更新する
11. (オプション) 詳細を追跡しやすいよう、新しいバージョンにタグを追加します。

カスタム分類子の作成については、「カスタム分類子の作成」を参照してください。

カスタムエンティティレコグナイザーの新しいバージョンを作成する (コンソール)

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/comprehend/ で Amazon Comprehend コンソールを開きます。
2. 左側のメニューから [カスタマイズ] を選択し、[カスタムエンティティ認識] を選択します。
3. [レコグナイザーモデル] のリストから、新しいバージョンを作成するレコグナイザーの名前を選択します。詳細ページが表示されます。
4. 右上の [新バージョンをトレーニング] を選択します。親エンティティレコグナイザーからの詳細があらかじめ入力されている画面が開きます。
5. [バージョン名] で、新しいバージョンに一意の名前を追加します。
6. [カスタムエンティティタイプ] で、1つまたは複数のカスタムラベルを追加し、[タイプを追加] を選択します。レコグナイザーは、このカスタムラベルを使用してデータセット内のエンティティタイプを識別します。指定したアノテーションまたはエンティティリストからカスタムエンティティタイプを選択します。レコグナイザーは、ジョブの実行時に、含まれているすべてのエンティティタイプを使用してデータセット内のエンティティを識別します。複数の単語を使用する場合は各エンティティタイプを大文字にし、アンダースコアで区切る必要があります。最大 25 のタイプを使用できます。
7. (オプション) ジョブの処理中にストレージボリューム内のデータを暗号化する場合は、[レコグナイザーの暗号化] を選択します。
8. トレーニングデータセクションで、アノテーションとデータ形式の詳細 (CSV ファイル、拡張マニフェスト)、シングルラベル、マルチラベル)、データ形式 (CSV、拡張マニフェスト)、トレーニングデータセット、テストデータセット (自動分割またはカスタムテストデータ設定) を指定します。
9. (オプション) 出力データの S3 上の場所を更新する
10. [アクセス権限] で、IAM ロールを作成するか、既存の IAM ロールを使用します。

11. (オプション) VPC 設定を更新する
12. (オプション) 詳細を追跡しやすいよう、新しいバージョンにタグを追加します。

カスタムエンティティレコグナイザーの詳細については、「カスタムエンティティ認識」と「コンソールを使用したカスタムエンティティレコグナイザーの作成」を参照してください。

AWS アカウント の間でのカスタムモデルのコピー

Amazon Comprehend のユーザーは 2 段階のプロセスで AWS アカウント 間で調教済みカスタムモデルをコピーできます。まず、AWS アカウント のユーザー (アカウント A) が自分のアカウントにあるカスタムモデルを共有します。次に、別の AWS アカウント のユーザー (アカウント B) がそのモデルを自分のアカウントにインポートします。アカウント B のユーザーはモデルを調教する必要がなく、元の調教データやテストデータをコピー (またはアクセス) する必要もありません。

アカウント A のカスタムモデルを共有するには、ユーザーはモデルバージョンに AWS Identity and Access Management (IAM) ポリシーをアタッチします。このポリシーは、アカウント B のエンティティ (ユーザーやロールなど) に、その AWS アカウント の Amazon Comprehend にモデルバージョンをインポートすることを許可します。アカウント B のユーザーは、AWS リージョン モデルを元のモデルと同じモデルにインポートする必要があります。

アカウント B のモデルをインポートするには、このアカウントのユーザーは Amazon Comprehend に、モデルの Amazon リソースネーム (ARN) などの必要な詳細を Amazon Comprehend に提示します。モデルをインポートすることで、このユーザーは、インポートしたモデルを複製する AWS アカウント に新しいカスタムモデルを作成します。このモデルは完全に調教されており、文書の分類や名前付きエンティティの認識などの推論作業にすぐ使用できます。

カスタムモデルのコピーは、以下の場合に役立ちます。

- 複数の AWS アカウント を使用する組織に所属している。たとえば、自分の組織では、ビルド、ステージ、テスト、デプロイなど、開発の各フェーズごとに AWS アカウント を用意している場合があります。また、データサイエンスやエンジニアリングなど、ビジネス機能ごとに異なる AWS アカウント を持つ場合もあります。
- 自分の組織は、Amazon Comprehend でカスタムモデルを調教し、クライアントとして自分に提供する AWS パートナーなどの他の組織と協働しています。

このようなシナリオでは、調教を受けたカスタムエンティティレコグナイザーやドキュメント分類子をある AWS アカウント から別のものにすばやくコピーできます。この方法でモデルをコピーす

る方が、調教データ AWS アカウント 間でコピーして重複するモデルを調教する方法よりも簡単です。

トピック

- カスタムモデルを別の AWS アカウント と共有する
- 別の AWS アカウント からのカスタムモデルのインポート

カスタムモデルを別の AWS アカウント と共有する

Amazon Comprehend を使用すると、カスタムモデルを他のユーザーと共有し、他のユーザーは自分の AWS アカウントにモデルをインポートできます。ユーザーがカスタムモデルのいずれかをインポートすると、そのユーザーは自分のアカウントに新しいカスタムモデルを作成します。その新しいモデルは共有されたものを再現しています。

カスタムモデルを共有するには、他のユーザーにインポートを許可するポリシーをそのモデルにアタッチします。次に、それらのユーザーに必要な詳細を提供します。

> **Note**
>
> 共有したカスタムモデルを他のユーザーがインポートする場合は、共有したモデルを含む同じ AWS リージョン (たとえば、米国東部 (バージニア北部)) を使用する必要があります。

トピック

- 開始する前に
- カスタムモデル用のリソースベースのポリシー
- ステップ 1: カスタムモデルにリソースベースのポリシーを追加します
- ステップ 2: 他のユーザーがインポートする必要がある詳細情報を入力します。

開始する前に

モデルを共有する前に、Amazon Comprehend にトレーニング済みのカスタム分類子またはカスタムエンティティレコグナイザーを AWS アカウント に用意しておく必要があります。モデルのトレーニングに関する詳細については、「カスタム分類」または「カスタムエンティティ認識」を参照してください。

必要なアクセス許可

IAM ポリシーステートメント

リソースベースのポリシーをカスタムモデルに追加する前に、AWS Identity and Access Management (IAM) のアクセス権限が必要です。次の例に示すように、モデルポリシーを作成、取得、削除できるように、ユーザー、グループ、またはロールにポリシーをアタッチする必要があります。

Example カスタムモデルのリソースベースのポリシーを管理するための IAM ポリシー

```
{
  "Effect": "Allow",
  "Action": [
    "comprehend:PutResourcePolicy",
    "comprehend:DeleteResourcePolicy",
    "comprehend:DescribeResourcePolicy"
  ],
  "Resource": "arn:aws:comprehend:us-west-2:111122223333:document-classifier/foo/
version/*"
}
```

IAM ポリシーの作成については、IAM ユーザーガイドの「IAM ポリシーの作成」を参照してください。IAM ポリシーのアタッチに関する詳細については、IAM ユーザーガイドの「IAM アイデンティティのアクセス許可の追加および削除」を参照してください。

AWS KMS キーポリシーステートメント

暗号化されたモデルを共有する場合は、AWS KMS の権限を追加する必要がある場合があります。この要件は、Amazon Comprehend でモデルを暗号化するために使用する KMS キーの種類によって異なります。

AWS 所有のキー は AWS サービスが所有、管理しています。AWS 所有のキー を使用する場合、AWS KMS に権限を追加する必要はないため、このセクションは省略可能です。

カスタマーマネージドキーは AWS アカウント のキーで、その作成、所有、管理をユーザーが行います。カスタマーマネージドキーを使用する場合は、KMS キーポリシーにステートメントを追加する必要があります。

ポリシーステートメントは、モデルの復号化に必要な AWS KMS 操作を実行することを 1 つ以上のエンティティ (ユーザーやアカウントなど) に付与します。

不分別な代理処理の問題を防止するには、条件キーを使用します。詳細については、「the section called “サービス間の混乱した代理の防止”」を参照してください。

ポリシー内の以下の条件キーを使用して、KMS キーにアクセスするエンティティを検証します。ユーザーがモデルをインポートすると、AWS KMS は、ソースモデルバージョンの ARN が条件と一致することを確認します。ポリシーに条件を含めない場合、指定されたプリンシパルは KMS キーを使用して任意のモデルバージョンを復号できます。

- aws:SourceArn – この条件キーを kms:GenerateDataKeyおよび kms:Decryptアクションで使用します。
- kms:EncryptionContext — この条件キーを kms:GenerateDataKey、kms:Decrypt、および kms:CreateGrantアクションで使用します。

次の例では、ポリシーは、AWS アカウント 111122223333 が所有する指定の分類子モデルのバージョン 1 を使用することを AWS アカウント 444455556666に許可します。

Example 特定の分類子モデルバージョンにアクセスするための KMS キーポリシー

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
          "AWS":
                "arn:aws:iam::444455556666:root"
      },
      "Action": [
          "kms:Decrypt",
          "kms:GenerateDataKey"
      ],
      "Resource": "*",
      "Condition": {
          "StringEquals": {
              "aws:SourceArn":
                "arn:aws:comprehend:us-west-2:111122223333:document-
classifier/classifierName/version/1"
          }
      }
    },
    {
```

```
      "Effect": "Allow",
      "Principal": {
          "AWS":  "arn:aws:iam::444455556666:root"
      },
      "Action": "kms:CreateGrant",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
           "kms:EncryptionContext:aws:comprehend:arn":
              "arn:aws:comprehend:us-west-2:111122223333:document-
classifier/classifierName/version/1"
        }
      }
    }
  ]
}
```

次のポリシー例では、Amazon Comprehend サービスを介してこの KMS キーにアクセスするAWS アカウント123456789012ことを ExampleUser AWS アカウント444455556666と ExampleRoleのユーザーに許可します。

Example Amazon Comprehend サービスへのアクセスを許可する KMS キーポリシー (代替案 1)。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": [
                "arn:aws:iam::444455556666:user/ExampleUser",
                "arn:aws:iam::123456789012:role/ExampleRole"
              ]
        },
        "Action": [
            "kms:Decrypt",
            "kms:GenerateDataKey"
        ],
        "Resource": "*",
        "Condition": {
            "StringLike": {
                "aws:SourceArn": "arn:aws:comprehend:*"
```

```
            }
        }
    },
    {
      "Effect": "Allow",
      "Principal": {
         "AWS": [
                "arn:aws:iam::444455556666:user/ExampleUser",
                "arn:aws:iam::123456789012:role/ExampleRole"
            ]
      },
      "Action": "kms:CreateGrant",
      "Resource": "*",
      "Condition": {
          "StringLike": {
              "kms:EncryptionContext:aws:comprehend:arn": "arn:aws:comprehend:*"
          }
      }
    }
  ]
}
```

次の例のポリシーは、AWS アカウント 444455556666 が前の例とは異なる構文を使用して、Amazon Comprehend サービスを介してこの KMS キーにアクセスすることを許可します。

Example Amazon Comprehend サービスへのアクセスを許可する KMS キーポリシー (代替案 2)。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
          "AWS": "arn:aws:iam::444455556666:root"
      },
      "Action": [
          "kms:Decrypt",
          "kms:GenerateDataKey",
          "kms:CreateGrant"
      ],
      "Resource": "*",
      "Condition": {
          "StringLike": {
```

```
                "kms:EncryptionContext:aws:comprehend:arn": "arn:aws:comprehend:*"
            }
        }
    }
  ]
}
```

詳細については、「AWS Key Management Service デベロッパーガイド」の「AWS KMS でのキーポリシー」を参照してください。

カスタムモデル用のリソースベースのポリシー

別の AWS アカウント の Amazon Comprehend がお客様の AWS アカウントからカスタムモデルをインポートする前に、それを許可する必要があります。承認するには、共有するモデルバージョンにリソースベースのポリシーを追加します。リソースベースのポリシーは、AWS でリソースにアタッチする IAM ポリシーです。

リソースポリシーをカスタムモデルバージョンに追加すると、そのポリシーは、ユーザー、グループ、またはロールにモデルバージョンで `comprehend:ImportModel` 動作を実行することを許可します。

Example カスタムモデルバージョンのリソースベースのポリシー

この例では、権限のある エンティティを `Principal` 属性に指定しています。リソース「`*`」は、ポリシーを追加する特定のモデルバージョンを指します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "comprehend:ImportModel",
      "Resource": "*",
      "Principal": {
        "AWS": [
                "arn:aws:iam::111122223333:root",
                "arn:aws:iam::444455556666:user/ExampleUser",
                "arn:aws:iam::123456789012:role/ExampleRole"
         ]
      }
    }
```

```
  ]
}
```

カスタムモデルに追加するポリシーの場合、`comprehend:ImportModel` は Amazon Comprehend がサポートする唯一のアクションです。

相違点の詳細については、「IAM ユーザーガイド」の「アイデンティティベースのポリシーおよびリソースベースのポリシー」を参照してください。

ステップ 1: カスタムモデルにリソースベースのポリシーを追加します

AWS Management Console 、AWS CLI、または Amazon Comprehend API を使用してリソースベースのポリシーを追加できます。

AWS Management Console

AWS Management Console で Amazon Comprehend を使用できます。

リソースベースのポリシーを追加するには

1. AWS Management Console にサインインして、Amazon Comprehend コンソール (https://console. .amazon.com/comprehend/) を開きます
2. 左側のナビゲーションメニューの [カスタマイズ] で、カスタムモデルを含むページを選択します。
   a. カスタムドキュメント分類子を共有する場合は、[カスタム分類]を選択します。
   b. カスタムエンティティレコグナイザーを共有する場合は、[カスタムエンティティ認識] を選択します。
3. モデルのリストから、詳細ページを開くモデル名を選択します。
4. バージョン から、共有するモデルバージョンの名前を選択します。
5. バージョン詳細ページで、[タグ、VPC、ポリシー] タブを選択します。
6. [リソースベースのポリシー] セクションで、[編集]を選択します。
7. [リソースベースのポリシー] ページで、次を実行します。
   a. [ポリシー名] には、ポリシーを作成した後、わかりやすい名前を入力します。
   b. [承認] で、モデルのインポートを許可する以下のエンティティを 1 つ以上指定します。

フィールド	定義とサンプル
サービスプリンシパル	このモデルバージョンにアクセスできるサービスのサービスプリンシパル識別子。例: comprehend.amazonaws.com
AWS アカウント IDs	このモデルバージョンにアクセスできる AWS アカウント。アカウントに属しているすべてのユーザーを承認します。例: 111122223333、123456789012
IAM エンティティ	このモデルバージョンにアクセスできるユーザーまたはロールの ARN。例: arn:aws:iam::111122223333:user/ExampleUser、arn:aws:iam::444455556666:role/ExampleRole

8. [共有] では、モデルバージョンの ARN をコピーして、モデルをインポートするユーザーと共有しやすくすることができます。誰かが別の AWS アカウント からカスタムモデルをインポートする場合、モデルバージョン ARN が必要です。
9. [保存] を選択します。Amazon Comprehend はリソースベースのポリシーを作成し、それをモデルにアタッチします。

AWS CLI

を使用してリソースベースのポリシーをカスタムモデルに追加するにはAWS CLI、PutResourcePolicy コマンドを使用します。 コマンドでは、以下のパラメータを使用します。

- `resource-arn` — モデルバージョンを含むカスタムモデルの ARN。
- `resource-policy` — カスタムモデルに追加するリソースベースのポリシーを定義する JSON ファイル。

  ポリシーをインライン JSON 文字列として指定することもできます。ポリシーに有効な JSON を提供するには、属性名と値を二重引用符で囲みます。JSON 本文も二重引用符で囲まれている場合は、ポリシー内の二重引用符をエスケープします。

- `policy-revision-id` — Amazon Comprehend が更新中のポリシーに割り当てたリビジョン ID。以前のバージョンがない新しいポリシーを作成する場合は、このパラメータを使用しないでください。Amazon Comprehend が自動的にリビジョン ID を作成します。

Example **`put-resource-policy`** コマンドを使用して、リソースベースのポリシーをカスタムモデルに追加する

この例では、PolicyFile.json という名前の JSON ファイルにポリシーを定義して、モデルに関連付けます。このモデルは mycf1 という分類子のバージョン v2 です。

```
$ aws comprehend put-resource-policy \
> --resource-arn arn:aws:comprehend:us-west-2:111122223333:document-classifier/mycf1/
version/v2 \
> --resource-policy file://policyFile.json \
> --policy-revision-id revision-id
```

リソースポリシーの JSON ファイルには以下の内容が含まれています。

- アクション — ポリシーは、指定されたプリンシパルに `comprehend:ImportModel` を使用することを許可します。
- リソース — カスタムモデルの ARN。リソース「`*`」は、`put-resource-policy` コマンドで指定したモデルバージョンを指します。
- プリンシパル — このポリシーは AWS アカウント 444455556666 からのユーザー `jane` と AWS アカウント 123456789012 からのすべてのユーザーを認証します。

```
{
"Version":"2012-10-17",
 "Statement":[
    {"Sid":"ResourcePolicyForImportModel",
     "Effect":"Allow",
     "Action":["comprehend:ImportModel"],
     "Resource":"*",
     "Principal":
         {"AWS":
            ["arn:aws:iam::444455556666:user/jane",
             "123456789012"]
         }
```

```
    }
  ]
}
```

Amazon Comprehend API

Amazon Comprehend API を使用してリソースベースのポリシーをカスタムモデルに追加するには、PutResourcePolicy API オペレーションを使用します。

モデルを作成する API リクエストで、カスタムモデルにポリシーを追加することもできます。これを行うには、 CreateDocumentClassifierまたは CreateEntityRecognizerリクエストを送信するときに、 ModelPolicy パラメータのポリシー JSON を指定します。

ステップ 2: 他のユーザーがインポートする必要がある詳細情報を入力します。

これで、リソースベースのポリシーをカスタムモデルに追加し、他の Amazon Comprehend ユーザーにお客様のモデルを AWS アカウント にインポートする権限を与えました。ただし、インポートする前に、以下の詳細情報を提供する必要があります。

- モデルバージョンの Amazon リソースネーム (ARN)。
- モデルを含む AWS リージョン。モデルをインポートする人は誰もが同じ AWS リージョン を使用する必要があります。
- モデルが暗号化されているかどうか、暗号化されている場合は、使用している AWS KMS キーのタイプ AWS 所有のキー またはカスタマーマネージドキー。
- モデルがカスタマーマネージドキーで暗号化されている場合は、KMS キーの ARN を指定する必要があります。モデルをインポートする人は誰もが、AWS アカウント 内のIAM サービスロールに ARN を含める必要があります。このロールは、インポート時に KMS キーを使用してモデルを復号できる KMS キーを使用する許可を Amazon Comprehend に付与します。

他のユーザーがモデルをインポートする方法の詳細については、「別の AWS アカウント からのカスタムモデルのインポート」を参照してください。

別の AWS アカウント からのカスタムモデルのインポート

Amazon Comprehend では、別の AWS アカウント からカスタムモデルをインポートできます。モデルをインポートすると、アカウントに新しいカスタムモデルが作成されます。新しいカスタムモデルは、インポートしたモデルと完全にトレーニングされたモデルの複製です。

トピック

- 開始する前に
- カスタムモデルのインポート

開始する前に

別の AWS アカウント からカスタムモデルをインポートする前に、そのモデルを共有した人が以下を行っていることを確認してください。

- インポートを行う権限をユーザーに与える。この承認は、モデルバージョンに添付されているリソースベースのポリシーで付与されます。詳細については、「カスタムモデル用のリソースベースのポリシー」を参照してください。
- 以下に関する情報を提供します。
  - モデルバージョンの Amazon リソースネーム (ARN)。
  - モデルを含む AWS リージョン。インポートには同じ AWS リージョン を使用する必要があります。
  - モデルが AWS KMS キーを使用して暗号化されているかどうか、されている場合はキーのタイプ。

モデルが暗号化されている場合、使用する KMS キーの種類によっては、追加の手順が必要になることがあります。

- AWS 所有のキー — このタイプの KMS キーは AWS によって所有、管理されます。モデルが AWS 所有のキー で暗号化されている場合、追加の手順は不要です。
- カスタマーマネージドキー — このタイプの KMS キーは、AWS 顧客が AWS アカウント で作成、所有、管理します モデルがカスタマーマネージドキーで暗号化されている場合、モデルを共有したユーザーは次のことを行う必要があります。
  - あなたにモデルを復号することを許可する。この認証は、カスタマーマネージドキーの KMS キーポリシーで付与されます。詳細については、「AWS KMS キーポリシーステートメント」を参照してください。
  - カスタマーマネージドキーの ARN を提供する。この ARN は、IAM サービスロールを作成するときに使用します。このロールは、モデルの暗号化に KMS キーを使用する許可を Amazon Comprehend に付与します。

必要なアクセス許可

カスタムモデルをインポートする前に、ユーザーまたは管理者が AWS Identity and Access Management (IAM) で必要なアクションを承認する必要があります。Amazon Comprehend ユーザーには、IAM ポリシーステートメントでのインポート権限が必要です。インポート時に暗号化または復号化が必要な場合は、Amazon Comprehend に必要な AWS KMS キーを使用する権限が必要です。

IAM ポリシーステートメント

次の例で示されているように、ユーザー、グループ、またはロールには `ImportModel` アクションを許可するポリシーが追加されている必要があります。

Example カスタムモデルをインポートするための IAM ポリシー

```
{
  "Effect": "Allow",
  "Action": [
    "comprehend:ImportModel"
  ],
  "Resource": "arn:aws:comprehend:us-west-2:111122223333:document-classifier/foo/
version/*"
}
```

IAM ポリシーの作成については、IAM ユーザーガイドの「IAM ポリシーの作成」を参照してください。IAM ポリシーのアタッチに関する詳細については、IAM ユーザーガイドの「IAM アイデンティティのアクセス許可の追加および削除」を参照してください。

AWS KMS 暗号化用の IAM サービスロール

カスタムモデルをインポートする場合、以下のいずれかの場合に Amazon Comprehend に AWS KMS キーの使用を許可する必要があります。

- AWS KMS で、カスタマーマネージドキーを使用して暗号化されたカスタムモデルをインポートします。この場合、Amazon Comprehend は KMS キーにアクセスして、インポート中にモデルを復号化する必要があります。
- インポートによって作成された新しいカスタムモデルを暗号化し、カスタマーマネージドキーを使用する場合。この場合、できるように、インポート中にモデルを復号化Amazon Comprehend は KMS キーにアクセスする必要があります。

Amazon Comprehend にこれらの AWS KMS キーの使用を許可するには、[IAM サービスロール] を作成します。このタイプの IAM ロールは、AWS サービスがユーザーに代わって他のサービスのリソースにアクセスするのことを許可します。サービスロールの詳細については、IAM ユーザーガイドの「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。

Amazon Comprehend コンソールを使用してインポートする場合は、Amazon Comprehend に自動的にサービスロールを作成させることができます。それ以外の場合は、インポートする前に IAM でサービスロールを作成する必要があります。

次の例で示されているように、IAM サービスロールにはアクセス許可ポリシーと信頼ポリシーが必要です。

Example アクセス許可ポリシー

以下のアクセス権限ポリシーは、Amazon Comprehend がカスタムモデルの暗号化と復号化に使用する AWS KMS 操作を許可します。これにより 2 つの KMS キーへのアクセスが許可されます。

- インポートするモデルを含む AWS アカウント に、1 つのKMSキーがあります。モデルの暗号化に使用され、Amazon Comprehend はインポート時にこれを使用してモデルを復号化します。
- もう 1 つの KMSキーは、インポートされたモデルの AWS アカウント にあります。Amazon Comprehend は、このキーを使用して、インポートによって作成された新しいカスタムモデルを暗号化します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "kms:CreateGrant"
        ],
        "Resource": [
            "arn:aws:kms:us-west-2:111122223333:key/key-id",
            "arn:aws:kms:us-west-2:444455556666:key/key-id"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "kms:Decrypt",
```

```
            "kms:GenerateDatakey"
        ],
        "Resource": [
            "arn:aws:kms:us-west-2:111122223333:key/key-id",
            "arn:aws:kms:us-west-2:444455556666:key/key-id"
        ],
        "Condition": {
            "StringEquals": {
                "kms:ViaService": [
                    "s3.us-west-2.amazonaws.com"
                ]
            }
        }
    }
  ]
}
```

Example 信頼ポリシー

次の信頼ポリシーでは、ロールを引き受け、アクセス許可を取得することを Amazon Comprehend に許可します。comprehend.amazonaws.com サービスプリンシパルが sts:AssumeRole 操作を実行できることを許可します。混乱した代理の防止に役立つようにするには、1 つ以上のグローバル条件コンテキストキーを使用してアクセス許可の範囲を制限します。aws:SourceAccount には、モデルをインポートするユーザーのアカウント ID を指定します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "comprehend.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "444455556666"
        }
      }
    }
  ]
}
```

カスタムモデルのインポート

AWS Management Console、AWS CLI、または Amazon Comprehend API を使用してカスタムモデルをインポートできます。

AWS Management Console

AWS Management Console で Amazon Comprehend を使用できます。

カスタムモデルをインポートするには

1. AWS Management Console にサインインして、Amazon Comprehend コンソール (https://console. .amazon.com/comprehend/) を開きます
2. 左側のナビゲーションメニューの [カスタマイズ] で、インポートするモデルのタイプに対応するページを選択します。

   a. カスタムドキュメント分類子を共有する場合は、[カスタム分類] を選択します。
   b. カスタムエンティティレコグナイザーをインポートする場合は、[カスタムエンティティ認識] を選択します。
3. [バージョンをインポート] を選択します。
4. [モデルバージョンをインポート] ページに、以下のように入力します。

   - [モデルバージョン ARN] — インポートするモデルバージョンの ARN。
   - [モデル名] — インポートによって作成される新しいモデルのカスタム名。
   - [バージョン名] — インポートによって作成される新しいモデルのカスタム名。
5. [モデル暗号化] では、インポート時に作成する新しいカスタムモデルの暗号化に使用する KMS キーの種類を選択します。

   - [AWS が所有するキーの使用] — Amazon Comprehend は AWS が自分の代わりに作成、管理、および使用する AWS Key Management Service (AWS KMS) のキーを使用してモデルを暗号化します。
   - 別の AWS KMS キーを選択 (詳細) — Amazon Comprehend は、AWS KMS で管理しているカスタマーマネージドキーを使用してモデルを暗号化します。

     このオプションを選択する場合は、AWS アカウント の KMS キーを選択するか、[AWS KMS キーの作成]を選択して新しいキーを作成します。
6. [サービスアクセス] セクションで、Amazon Comprehend に必要なすべての AWS KMS キーへのアクセス権を付与します。

- インポートしたカスタムモデルを復号化します。
- インポートで作成した新しいカスタムモデルを暗号化します。

Amazon Comprehend が KMS キーを使用できるようにする IAM サービスロールを使用してアクセスを許可します。

[サービスロール] で、次のいずれかの操作を行います。

- 使用する既存のサービスロールがある場合は、[既存の IAM ロールを使用] を選択します。次に、[ロール名] でそのロールを選択します。
- Amazon Comprehend に自動的にロールを作成してもらいたい場合は、[IAM ロール作成] を選択してください。

7. ロールは Amazon Comprehend に作成するように選択した場合は、以下を実行します。

   a. [ロール名] には、後でそのロールを認識しやすいようにロール名のサフィックスを入力します。

   b. [ソース KMS キー ARN] には、インポートするモデルの暗号化に使用する KMS キーの ARN を入力します。Amazon Comprehend は、インポート時にこのキーを使用してモデルを復号化します。

8. (オプション) [タグ] セクションでは、インポートして作成した新しいカスタムモデルにタグを追加できます。カスタムモデルのタグ付けの詳細については、「新しいリソースへのタグ付け」を参照してください。

9. [確認] を選択します。

AWS CLI

Amazon Comprehend を使用するには、AWS CLI でコマンドを実行します。

Example Import-model コマンド

カスタムモデルをインポートするには、`import-model` コマンドを使用します。

```
$ aws comprehend import-model \
> --source-model arn:aws:comprehend:us-west-2:111122223333:document-classifier/foo/
version/bar \
> --model-name importedDocumentClassifier \
> --version-name versionOne \
> --data-access-role-arn arn:aws:iam::444455556666:role/comprehendAccessRole \
```

```
> --model-kms-key-id kms-key-id
```

この例は以下のパラメータを使用します。

- `source-model` — インポートするカスタムモデルの ARN。
- `model-name` — インポートによって作成される新しいモデルのカスタム名。
- `version-name` — インポートによって作成される新しいモデルバージョンのカスタム名。
- `data-access-role-arn` — Amazon Comprehend が必要な AWS KMS キーを使用してカスタムモデルを暗号化または復号化できるようにする IAM サービスロールの ARN。
- `model-kms-key-id` — Amazon Comprehend がこのインポートで作成したカスタムモデルを暗号化するために使用する KMS キーの ARN または ID。このキーは AWS KMS 内の AWS アカウント 内にある必要があります。

Amazon Comprehend API

Amazon Comprehend API を使用してカスタムモデルをインポートするには、 ImportModel API アクションを使用します。

フライホイール

Amazon Comprehend フライホイールを使用すると、カスタムモデルを時間をかけて改良するプロセスが簡単になります。フライホイールは、モデルの新しいバージョンのトレーニングと評価に関連するタスクを調整するのに役立ちます。フライホイールは、カスタム分類とカスタムエンティティ認識を行うためのプレーンテキストのカスタムモデルをサポートします。

トピック

- フライホイールの概要
- フライホイールのデータレイク
- IAM ポリシーとアクセス許可
- コンソールを使用したフライホイールの設定
- API を使用したフライホイールの設定
- データセットの設定
- フライホイールイテレーション
- フライホイールを使用した分析の実行

フライホイールの概要

フライホイールは、カスタムモデルの新しいバージョンのトレーニングと評価のオーケストレーションを行う Amazon Comprehend のリソースです。フライホイールを作成することで、既存のトレーニング済みモデルを使用することができ、Amazon Comprehend はそのフライホイール用の新しいモデルを作成してトレーニングすることができます。フライホイールとプレーンテキストのカスタムモデルを組み合わせることで、カスタム分類やカスタムエンティティ認識を行うことができます。

フライホイールは、Amazon Comprehend コンソールまたは API を使用して設定および管理することができます。AWS CloudFormation を使用してフライホイールを設定することもできます。

フライホイールを作成すると、Amazon Comprehend はアカウントにデータレイクを作成します。データレイク は、モデルの全バージョンのトレーニングデータやテストデータなど、すべてのフライホイールデータを保存および管理します。

推論ジョブまたは Amazon Comprehend エンドポイントに使用するフライホイールモデルのバージョンには、アクティブなモデルバージョンを設定します。当初、フライホイールにはモデルの 1 つのバージョンのみが含まれています。時間の経過と共に、新しいモデルバージョンをトレーニングしていくと、最もパフォーマンスの高いバージョンをアクティブなモデルバージョンとして選択

できます。ユーザーが推論ジョブを実行するためのフライホイールの ARN を指定すると、Amazon Comprehend はフライホイールのアクティブなモデルバージョンを使用してジョブを実行します。

モデルの新しいラベル付きデータ (トレーニングデータ、または、テストデータ) を定期的に取得するようにしてください。1 つ以上のデータセットが作成されると、フライホイールが新しいデータを利用できるようになります。データセットには、フライホイールに関連づけられたカスタムモデルをトレーニングまたはテストするための入力データが含まれます。Amazon Comprehend は、入力データをフライホイールのデータレイクにアップロードします。

新しいデータセットをカスタムモデルに反映させるには、フライホイールのイテレーションを作成して実行します。フライホイールのイテレーションは、新しいデータセットを使用して、アクティブなモデルバージョンを評価し、新しいモデルバージョンをトレーニングするワークフローです。既存のモデルバージョンと新しいモデルバージョンのメトリックに基づいて、新しいモデルバージョンをアクティブなバージョンに昇格させるかどうかを決定します。

フライホイールのアクティブなモデルバージョンを使用して、カスタム分析 (リアルタイムまたは非同期ジョブ) を実行できます。フライホイールモデルをリアルタイム分析に使用するには、フライホイール用のエンドポイント を作成する必要があります。

フライホイールの利用に追加料金はかかりません。ただし、フライホイールのイテレーションを実行すると、新しいモデルバージョンのトレーニングとモデルデータの保存に標準料金がかかります。料金の詳細については、「Amazon Comprehend の料金」を参照してください。

トピック

- フライホイールのデータセット
- フライホイールの作成
- フライホイールの状態
- フライホイールのイテレーション

フライホイールのデータセット

フライホイールに新しいラベル付きデータを追加するには、データセットを作成します。各データセットは、トレーニング用にもテスト用にも設定できます。　データセットは、特定のフライホイールおよびカスタムモデルに関連付けることができます。

データセットを作成すると、Amazon Comprehend はデータをフライホイールのデータレイクにアップロードします。　詳細については、「フライホイールのデータレイク」を参照してください。

フライホイールの作成

フライホイールの作成では、そのフライホイールを既存のトレーニング済みモデルに関連付け、フライホイールは新しいモデルを作成することができます。

既存のモデルを使用したフライホイールの作成では、アクティブなモデルバージョンを指定できます。　Amazon Comprehend は、モデルのトレーニング用データとテスト用データをフライホイールのデータレイクにコピーします。　モデルのトレーニングとテスト用データが、モデルを作成した時と同じ Amazon S3 の場所に存在することを確認します。

新規モデル用のフライホイールを作成するには、フライホイールの作成時にトレーニングデータ用のデータセット (テストデータ用のオプションデータセット) を指定します。フライホイールを実行してその最初のイテレーションを作成すると、フライホイールによって新しいモデルがトレーニングされます。

カスタムモデルのトレーニングでは、モデルが認識するカスタムラベル (カスタム分類) またはカスタムエンティティ (カスタムエンティティ認識) のリストを指定します。カスタムラベル/エンティティに関する以下の重要ポイントに注意してください。

- 新しいモデル用のフライホイールの作成では、フライホイールの作成時に指定したラベル/エンティティのリストがフライホイールの最終リストになります。
- 既存のモデルからのフライホイールの作成では、そのモデルに関連付けられているラベル/エンティティのリストがフライホイールの最終リストになります。
- 新しいデータセットをフライホイールに関連付け、そのデータセットに追加のラベル/エンティティが含まれている場合、Amazon Comprehend は新しいラベル/エンティティを無視します。
- DescribeFlywheel API オペレーションを使用して、フライホイールのラベル/エンティティリストを確認できます。

> **Note**
>
> カスタム分類の場合、Amazon Comprehend はフライホイールのステータスが ACTIVE になった後にラベルリストにデータを入力します。 DescribeFlywheel API オペレーションを呼び出す前に、フライホイールがアクティブになるまで待ちます。

フライホイールの状態

フライホイールの以下の状態遷移をします。

- CREATING - Amazon Comprehend がフライホイールリソースを作成しています。フライホイールに、`DescribeFlywheel` などの読み取りオペレーションを実行できます。
- ACTIVE - フライホイールはアクティブです。フライホイールのイテレーションが進行中かどうか、またイテレーションのステータスを確認できます。フライホイールに読み取りアクションを実行したり、`DeleteFlywheel` や `UpdateFlywheel` などのアクションを実行したりできます。
- UPDATING - Amazon Comprehend がフライホイールをアップデートしています。フライホイールに読み取りオペレーションを実行できます。
- DELETING - Amazon Comprehend はフライホイールを削除しています。フライホイールに読み取りオペレーションを実行できます。
- FAILED - フライホイールの作成オペレーションが失敗しました。

Amazon Comprehend がフライホイールを削除した後でも、フライホイールデータレイク内のすべてのモデルデータには引き続きアクセスできます。Amazon Comprehend は、フライホイールリソースの管理に必要な内部メタデータをすべて削除します。Amazon Comprehend また、このフライホイールに関連付けられているデータセットも削除します (モデルデータはデータレイクに保存されます)。

フライホイールのイテレーション

フライホイールモデル用の新しいトレーニングデータまたはテストデータを取得すると、1 つ以上の新しいデータセットを作成して、フライホイールのデータレイクにアップロードすることができます。

フライホイールを実行して新しいイテレーションを作成できます。フライホイールのイテレーションでは、新しいデータを使用して現在のアクティブなモデルバージョンが評価され、その結果がデータレイクに保存されます。また、フライホイールは新しいモデルバージョンを作成して、トレーニングします。

新しいモデルのパフォーマンスが、現在のアクティブなバージョンに勝っている場合は、新しいモデルバージョンをアクティブなモデルバージョンに昇格させることができます。コンソールまたは UpdateFlywheel API オペレーションを使用して、アクティブなモデルバージョンを更新できます。

フライホイールのデータレイク

フライホイールが作成されると、Amazon Comprehend は、モデルのすべてのバージョンに必要な入出力データなど、すべてのフライホイールデータを格納するためのデータレイクをアカウントに作成します。

Amazon Comprehend は、フライホイールの作成時に指定された Amazon S3 上の場所にデータレイクを作成します。場所は Amazon S3 バケットまたは Amazon S3 バケット内の新規フォルダとして指定できます。

データレイクのフォルダ構造

Amazon Comprehend は、データレイクを作成すると Amazon S3 上の場所に次のフォルダ構造をセットアップします。

> **Warning**
>
> Amazon Comprehend は、データレイクのフォルダ編成とコンテンツを管理します。データレイクのフォルダの変更には、必ず Amazon Comprehend API オペレーションを使用してください。さもないと、フライホイールが正しく動作しない場合があります。

```
Document Pool
Annotations Pool
Staging
Model Datasets
  (data for each version of the model)
  VersionID-1
    Training
    Test
    ModelStats
  VersionID-2
    Training
    Test
    ModelStats
```

モデルバージョンのトレーニング評価を確認するには、以下の手順を実行します。

1. データレイクのルートレベルにある Model Datasets という名前のフォルダを開きます。このフォルダーには、モデルの各バージョンのサブフォルダーが含まれます。
2. 目的のモデルバージョンが入っているフォルダーを開きます。
3. という名前のフォルダを開いてModelStats、モデルの統計を表示します。

データレイクの管理

Amazon Comprehend は代わりに以下のタスクを自動的に実行してデータレイクを管理します。

- データレイクのフォルダ構造を定義し、データセットを適切なフォルダに取り込む。
- モデルのトレーニングに必要な入力ドキュメント (テキストファイルやアノテーションファイルなど) を管理する。
- モデルの各バージョンに関連付けられているトレーニングと評価の出力データを管理する。
- データレイクに保存されているファイルの暗号化を管理する。

Amazon Comprehend は、データレイク用のデータの作成および更新オペレーションのすべてを実行します。データレイク内のデータへの完全なアクセス権が必要です。例:

- データレイクのすべてのコンテンツへの完全なアクセス権が必要です。
- これによりフライホイールの削除後も、データレイクを引き続き利用できます。
- データレイクを含む Amazon S3 バケットに関するアクセスログを設定できます。
- データに対する暗号化キーを指定できます。これらはフライホイールのを作成時に指定します。

推奨されるベストプラクティスを以下に示します：

- ご自分のフォルダやファイルをデータレイクに手動で追加しない。データレイク内のファイルを変更および削除しない。
- データレイク内のデータを追加または変更するときは、必ず Amazon Comprehend の作成および更新オペレーションを使用する。例えば、 トレーニング用データやテスト用データの提供に `CreateDataset`、 モデルのバージョンの評価データの生成に `StartFlywheelIteration` を使用できます。
- データレイクの構造は、時間の経過とともに進化する場合があります。明示的にデータレイク構造に依存するダウンストリームスクリプトやプログラムを作成しない。
- フライホイールにデータレイクの場所を指定する場合は、すべてのフライホイールに関連するデータに共通のプレフィックスを作成するか、フライホイールごとに異なるプレフィックスを使用することをお勧めします。あるフライホイールの完全なデータレイクパスを別のフライホイールのプレフィックスとして使用しないでください。

IAM ポリシーとアクセス許可

フライホイールを使用するには、以下のポリシーとアクセス許可を設定します。

- the section called “IAM ユーザーのアクセス許可を設定する” - フライホイールのオペレーションへのアクセスに必要。
- (オプション) the section called “AWS KMS キーに対するアクセス許可を設定する” - データレイクに必要。
- the section called “データアクセスロールを作成する” - Amazon Comprehend によるデータレイクへのアクセス許可。

IAM ユーザーのアクセス許可を設定する

フライホイール機能を利用できるようにするには、AWS Identity and Access Management (IAM) アイデンティティ (ユーザー、グループ、ロール) に適切なアクセス許可ポリシーを追加します。

次の例は、データセットの作成、フライホイールの作成と管理、およびフライホイールの実行を行うためのアクセス許可ポリシーを示しています。

Example フライホイールを管理するための IAM ポリシー

```
{
  "Effect": "Allow",
  "Action": [
    "comprehend:CreateFlywheel",
    "comprehend:DeleteFlywheel",
    "comprehend:UpdateFlywheel",
    "comprehend:ListFlywheels",
    "comprehend:DescribeFlywheel",
    "comprehend:CreateDataset",
    "comprehend:DescribeDataset",
    "comprehend:ListDatasets",
    "comprehend:StartFlywheelIteration",
    "comprehend:DescribeFlywheelIteration",
    "comprehend:ListFlywheelIterationHistory"
  ],
  "Resource": "*"
}
```

Amazon Comprehend 用の IAM ポリシーの作成の詳細については、「Amazon Comprehend と IAM 連携の仕組み」を参照してください。

AWS KMS キーに対するアクセス許可を設定する

データレイク上のデータに AWS KMS キーを使用する場合は、そのために必要なアクセス許可を設定します。詳細については、「KMS 暗号化を使用するために必要なアクセス許可」を参照してください。

データアクセスロールを作成する

IAM for Amazon Comprehend でデータアクセスロールを作成して、データレイク上のフライホイールデータにアクセスできるようにします。コンソールを使用してフライホイールを作成する場合は、必要に応じて新しいロールを作成できます。詳細については、「バッチ操作に必要なロールベースのアクセス許可」を参照してください。

コンソールを使用したフライホイールの設定

Amazon Comprehend API を使用して、フライホイールを作成、更新および削除することができます。

フライホイールが作成されると、Amazon Comprehend は、モデルの各バージョンのトレーニングデータやテストデータなど、フライホイールに必要なすべてのデータを保持するデータレイクを作成します。

フライホイールを削除しても、Amazon Comprehend はフライホイールに関連付けられているデータレイクやモデルを削除しません。

新規フライホイールを作成するにあたっては、セクション「フライホイールの作成」の情報を確認しておいてください。

トピック

- フライホイールを作成する
- フライホイールを更新する
- フライホイールを削除する

フライホイールを作成する

フライホイールの作成で必須の設定フィールドは、そのフライホイールが既存のカスタムモデル用か新規モデル用かによって異なります。

フライホイールを作成する

1. AWS Management Console にサインインし、Amazon Comprehend コンソールを開きます。
2. 左側のメニューから [フライホイール] を選択します。
3. [フライホイール] テーブルから [新規フライホイールの作成] を選択します。
4. [フライホイール名] でフライホイールの名前を入力します。
5. (オプション) 既存のモデル用のフライホイールを作成するには、[アクティブなモデルバージョン] のフィールドを設定します。

   a. [モデル] ドロップダウンリストからモデルを選択します。
   b. [バージョン] ドロップダウンリストから、モデルのバージョンを選択します。
6. (オプション) フライホイール用の新しい分類モデルを作成するには、[カスタムモデルタイプ] で [カスタム分類] を選択し、次の手順に従ってパラメータを設定します。

   a. [言語] で、モデルの言語を選択します。
   b. [分類子モード] で、シングルラベルモードまたはマルチラベルモードを選択します。
   c. [カスタムラベル] で、モデルのトレーニングに使用する 1 つ以上のカスタムラベルを入力します。各ラベルは、入力トレーニングデータ内のいずれかのクラスと一致している必要があります。
7. (オプション) フライホイール用の新しいエンティティ認識モデルを作成するには、[カスタムモデルタイプ] で [カスタムエンティティ認識] を選択し、次の手順に従ってパラメータを設定します。

   a. [言語] で、モデルの言語を選択します。
   b. [カスタムエンティティタイプ] で、モデルのトレーニングに使用するカスタムエンティティを最大 25 個入力します。各ラベルは、入力トレーニングデータ内のいずれかのクラスと一致している必要があります。

      複数のラベルを作成するには、その個数に応じて次の手順を実行します。

      i. カスタムラベルを入力します。ラベルはすべて大文字である必要があります。ラベル内の単語間の区切りにはアンダースコアを使用します。

ii. [タイプを追加] を選択します。

追加したラベルを削除するには、ラベル名の右側にある [X] を選択します。

8. ボリューム暗号化、モデル暗号化、データレイク暗号化の選択肢を設定します。選択肢のそれぞれについて、AWS 所有の KMS キーまたはアクセス許可を持っているキーのどちらを使用するかを選択します。

- AWS 所有の KMS キーを使用する場合、追加のパラメータはありません。
- 既存の別のキーを使用する場合は、[KMS キーの ARN] で キー ID の ARN を入力します。
- 新しいキーを作成する場合は、[新しい AWS KMS キーの作成]を選択します。

KMS キーの作成と使用、および関連する暗号化の詳細については、「AWS Key Management Service」 を参照してください。

a. ボリューム暗号化キーを設定します。Amazon Comprehend は、このキーを使用して、ジョブの処理中にストレージボリューム内のデータを暗号化します。AWS 所有の KMS キー、またはアクセス許可を持っているキーのどちらを使用するを選択します。

b. モデル暗号化キー を設定します。Amazon Comprehend はこのキーを使用して、そのモデルバージョン用のモデルデータを暗号化します。

9. データレイクの場所 を設定します。詳細については、「データレイクの管理」を参照してください。

10. (オプション) データレイク暗号化キーを設定します。Amazon Comprehend は、このキーを使用してデータレイク内のすべてのファイルを暗号化します。

11. (オプション) VPC 設定の設定をします。[VPC] で VPC ID を入力するか、ドロップダウンリストから ID を選択します。

1. [サブネット] でサブネットを選択します。最初のサブネットを選択すると、追加のサブネットを選択できます。
2. セキュリティグループを指定した場合は、[セキュリティグループ] で、使用するセキュリティグループを選択します。最初のセキュリティグループを選択すると、追加のセキュリティグループを選択できます。

12. サービスアクセス のアクセス許可を設定します。

1. [既存の IAM ロールを使用する] を選択した場合は、ドロップダウンリストでロール名を選択します。

2. [IAM ロールを作成する] を選択した場合は、Amazon Comprehend によって新しいロールが作成されます。Amazon Comprehend によってこのロールに設定されたアクセス許可がコンソールに表示されます。[ロール名] でロールの名前を入力します。

13. (オプション) タグ設定の設定をします。タグを追加するには、[タグ] でキーと値のペアを入力します。タグを追加 を選択します。フライホイールを作成する前にこのペアを削除するには、[タグを削除] を選択します。詳細については、「リソースのタグ付け」を参照してください
14. [Create] (作成) を選択します。

フライホイールを更新する

フライホイール名、データレイクの場所、モデルタイプ、およびモデル設定を設定できるのは、フライホイールの作成時だけです。

モデルタイプと設定オプションが現在のモデルと同じ場合は、フライホイールの更新で別のモデルを指定できます。新しいアクティブなモデルバージョンを設定できます。暗号化の詳細、サービスアクセス許可、VPC 設定も更新できます。

フライホイールを更新する

1. AWS Management Console にサインインし、Amazon Comprehend コンソールを開きます。
2. 左側のメニューから [フライホイール] を選択します。
3. [フライホイール] テーブルから、更新するフライホイールを選択します。
4. [アクティブなモデルバージョン] にある [モデル] ドロップダウンリストからモデルを選択して、モデルのバージョンを選択します。

   フォームにモデルタイプとモデル設定が反映されます。
5. (オプション) ボリューム暗号化とモデル暗号化 設定の設定をします。
6. (オプション) データレイク暗号化設定の設定をします。
7. サービスアクセス のアクセス許可を設定します。
8. (オプション) VPC 設定の設定をします。
9. (オプション) タグ設定の設定をします。
10. [保存] を選択します。

フライホイールを削除する

フライホイールを削除する

1. AWS Management Console にサインインし、Amazon Comprehend コンソールを開きます。
2. 左側のメニューから [フライホイール] を選択します。
3. [フライホイール] テーブルから、削除するフライホイールを選択します。
4. [削除] をクリックします。

API を使用したフライホイールの設定

Amazon Comprehend API を使用して、フライホイールを作成、更新、および削除することができます。

フライホイールが作成されると、Amazon Comprehend は、モデルの各バージョンのトレーニングデータやテストデータなど、フライホイールに必要なすべてのデータを保持するデータレイクを作成します。

フライホイールを削除しても、Amazon Comprehend はフライホイールに関連付けられているデータレイクやモデルを削除しません。

フライホイールがイテレーションを実行しているか、データセットを作成している場合、フライホイールの削除操作は失敗します。

新規フライホイールを作成するにあたっては、セクション「フライホイールの作成」の情報を確認しておいてください。

既存のモデル用のフライホイールを作成する

CreateFlywheel オペレーションを使用して、既存のモデルのフライホイールを作成します。

Example

```
aws comprehend create-flywheel  \
    --flywheel-name "myFlywheel2"  \
    --active-model-arn  "modelArn"  \
    --data-access-role-arn   arn:aws::iam::111122223333:role/testFlywheelDataAccess \
    --data-lake-s3-uri": "https://s3-bucket-endpoint"   \
```

オペレーションが成功すると、レスポンスにはフライホイールの ARN が含まれます。

```
{
  "FlywheelArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/name",
  "ActiveModelArn": "modelArn"
}
```

新しいモデル用のフライホイールを作成する

CreateFlywheel オペレーションを使用して、新しいカスタム分類モデルのフライホイールを作成します。

Example

```
aws comprehend create-flywheel \
    --flywheel-name "myFlywheel2" \
    --data-access-role-arn  arn:aws::iam::111122223333:role/testFlywheelDataAccess \
    --model-type "DOCUMENT_CLASSIFIER" \
    --data-lake-s3-uri  "s3Uri"  \
    --task-config  file://taskConfig.json
```

taskConfig.json ファイルには以下の内容が含まれます。

```
{
    "LanguageCode": "en",
    "DocumentClassificationConfig": {
        "Mode": "MULTI_LABEL",
        "Labels": ["optimism", "anger"]
    }
}
```

API レスポンスの本体には以下の内容が含まれます。

```
{
  "FlywheelArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/name",
  "ActiveModelArn": "modelArn"
}
```

フライホイールの説明を表示する

Amazon Comprehend DescribeFlywheelオペレーションを使用して、フライホイールに関する設定済みの情報を取得します。

```
aws comprehend describe-flywheel \
    --flywheel-arn  "flywheelArn"
```

API レスポンスの本体には以下の内容が含まれます。

```
{
  "FlywheelProperties": {
      "FlywheelArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/
myTestFlywheel",
      "DataAccessRoleArn": "arn:aws::iam::111122223333:role/Admin",
      "TaskConfig": {
          "LanguageCode": "en",
          "DocumentClassificationConfig": {
              "Mode": "MULTI_LABEL"
          }
      },
      "DataLakeS3Uri": "s3://my-test-datalake/flywheelbasictest/myTestFlywheel/
schemaVersion=1/20220801T014326Z",
      "Status": "ACTIVE",
      "ModelType":  "DOCUMENT_CLASSIFIER",
      "CreationTime": 1659318206.102,
      "LastModifiedTime": 1659318249.05
  }
}
```

フライホイールを更新する

UpdateFlywheel オペレーションを使用して、フライホイールの変更可能な設定値を更新します。

一部の設定フィールドは、サブフィールドを含む JSON 構造になっています。サブフィールドを更新するには、一部であれ、すべてのサブフィールドに値を指定します (Amazon Comprehend は、リクエストにないサブフィールドの値を NULL に設定します)。

UpdateFlywheel リクエストで最上位のパラメータを省略した場合、Amazon Comprehend はフライホイール内のパラメータやそのサブフィールドの値を変更しません。

フライホイールのタグを追加または削除するには、 TagResourceおよび UntagResourceオペレーションを使用します。

次の例に示すように、ActiveModelArn パラメータを設定することでモデルのバージョンをプロモートすることができます。

```
aws comprehend update-flywheel \
    --region aws-region \
    --flywheel-arn  "flywheelArn" \
    --active-model-arn  "modelArn" \
```

API レスポンスの本体には以下の内容が含まれます。

```
{
  "FlywheelArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/name",
  "ActiveModelArn": "modelArn"
}
```

フライホイールを削除する

Amazon Comprehend DeleteFlywheelオペレーションを使用してフライホイールを削除します。

```
aws comprehend delete-flywheel \
    --flywheel-arn  "flywheelArn"
```

正常な API レスポンスのメッセージ本体は空になります。

フライホイールを一覧表示する

Amazon Comprehend ListFlywheelsオペレーションを使用して、現在のリージョンのフライホイールのリストを取得します。

```
aws comprehend list-flywheel \
    --region aws-region \
    --endpoint-url  "uri"
```

API レスポンスの本体には以下の内容が含まれます。

```
{
    "FlywheelSummaryList": [
        {
            "FlywheelArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/
myTestFlywheel",
            "DataLakeS3Uri": "s3://my-test-datalake/flywheelbasictest/myTestFlywheel/
schemaVersion=1/20220801T014326Z",
```

```
            "Status": "ACTIVE",
            ""ModelType":  "DOCUMENT_CLASSIFIER",
            "CreationTime": 1659318206.102,
            "LastModifiedTime": 1659318249.05
        }
    ]
}
```

データセットの設定

ラベル付きのトレーニング用データまたはテスト用データをフライホイールに追加するには、Amazon Comprehend コンソールまたは API を使用してデータセットを作成します。

各データセットは、トレーニング用にも、テスト用にも設定できます。　データセットは、特定のフライホイールおよびカスタムモデルに関連付けることができます。　データセットが作成されると、Amazon Comprehend はフライホイールのデータレイクにそのデータをアップロードします。　トレーニング用データのファイル形式の詳細については、「分類子調教データの作成」または「エンティティレコグナイザーのトレーニングデータの準備」を参照してください。

フライホイールが削除されると、Amazon Comprehend はデータセットを削除します。データレイクにアップロードされたデータは、引き続き利用できます。

データセットの作成 (コンソール)

データセットを作成する

1. AWS Management Console にサインインし、Amazon Comprehend コンソールを開きます。
2. 左側のメニューから [フライホイール] を選択し、データを追加するフライホイールを選択します。
3. [データセット] タブを選択します。
4. [トレーニング用データセット] または [テスト用データセット] テーブルで、[データセットの作成] を選択します。
5. [データセットの詳細] で、データセットの名前とオプションの説明を入力します。
6. [データ仕様] で、データ形式およびデータセットタイプの設定フィールドを選択します。
7. (オプション) [入力形式] で、入力文書の形式を選択できます。
8. [S3 上のアノテーションの場所] で、Amazon S3 上のアノテーションファイルの場所を入力します。

9. [S3 上のトレーニングデータの場所] で、Amazon S3 上のドキュメントファイルの場所を入力します。
10. [作成] を選択します。

データセットの作成 (API)

CreateDataset オペレーションを使用してデータセットを作成できます。

Example

```
aws comprehend create-dataset \
    --flywheel-arn "myFlywheel2" \
    --dataset-name "my-training-dataset"
    --dataset-type "TRAIN"
    --description "my training dataset"
    --cli-input-json file://inputConfig.json
}
```

inputConfig.json ファイルには次のコンテンツが含まれます。

```
{
    "DataFormat": "COMPREHEND_CSV",
    "DocumentClassifierInputDataConfig": {
        "S3Uri": "s3://my-comprehend-datasets/multilabel_train.csv"
    }
}
```

データセットのタグを追加または削除するには、 TagResourceおよび UntagResourceオペレーションを使用します。

データセットを記述します

Amazon Comprehend DescribeDatasetオペレーションを使用して、フライホイールに関する設定済みの情報を取得します。

```
aws comprehend describe-dataset \
    --dataset-arn  "datasetARN"
```

レスポンスの内容は次のとおりです。

```
{
   "DatasetProperties": {
      "DatasetArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/
myTestFlywheel/dataset/train-dataset",
      "DatasetName": "train-dataset",
      "DatasetType": "TRAIN",
      "DatasetS3Uri": "s3://my-test-datalake/flywheelbasictest/myTestFlywheel/
schemaVersion=1/20220801T014326Z/datasets/train-dataset/20220801T194844Z",
      "Description": "Good Dataset",
      "Status": "COMPLETED",
      "NumberOfDocuments": 90,
      "CreationTime": 1659383324.297
  }
}
```

フライホイールイテレーション

フライホイールイテレーションを使用すると、新しいモデルバージョンの作成と管理に役立ちます。

トピック

- イテレーションワークフロー
- イテレーションの管理 (コンソール)
- イテレーションの管理 (API)

イテレーションワークフロー

フライホイールは、トレーニング済みのモデルバージョンから開始するか、初期データセットを使用してモデルバージョンをトレーニングします。

時間が経つにつれて、新しいラベル付きデータを取得したら、フライホイールモデルのパフォーマンスを向上させるために新しいモデルバージョンをトレーニングします。フライホイールを実行すると、新しいモデルバージョンをトレーニングして評価する新しいイテレーションが作成されます。新しいモデルバージョンのパフォーマンスが既存のアクティブなモデルバージョンよりも優れている場合は、その新しいモデルバージョンを昇格させることができます。

フライホイールイテレーションのワークフローには、次の手順が含まれます。

1. ラベル付きの新しいデータ用のデータセットを作成します。

2. フライホイールを実行して新しいイテレーションを作成します。イテレーションは以下のステップに従って新しいモデルバージョンをトレーニングし、評価します。
   a. 新しいデータを使用してアクティブなモデルバージョンを評価します。
   b. 新しいデータを使用して新しいモデルバージョンをトレーニングします。
   c. 評価とトレーニングの結果をデータレイクに保存します。
   d. 両方のモデルの F1 スコアを返します。
3. イテレーションが完了すると、既存のアクティブモデルと新しいモデルの F1 スコアを比較できます。
4. 新しいモデルバージョンのパフォーマンスが優れている場合は、それをアクティブなモデルバージョンに昇格させることができます。コンソール または API を使用して新しいモデルバージョンを昇格させることができます。

イテレーションの管理 (コンソール)

コンソールを使用して新しいイテレーションを開始し、進行中のイテレーションのステータスを問い合わせることができます。また、完了したイテレーションの結果を表示することもできます。

フライホイールイテレーションを開始する (コンソール)

新しいイテレーションを開始する前に、1 つ以上の新しいトレーニングデータセットまたはテストデータセットを作成します。「データセットの設定」を参照

フライホイールイテレーションを開始する (コンソール)

1. AWS Management Console にサインインし、Amazon Comprehend コンソールを開きます。
2. 左側のメニューから [フライホイール] を選択します。
3. [フライホイール] テーブルからフライホイールを選択します。
4. [フライホイールを実行] を選択します。

イテレーション結果の分析 (コンソール)

フライホイールイテレーションを実行すると、コンソールは [フライホイールイテレーション] テーブルに結果を表示します。

新しいモデルバージョンを昇格させる (コンソール)

コンソールのモデル詳細ページから、新しいモデルバージョンをアクティブなモデルバージョンに昇格できます。

フライホイールモデルバージョンをアクティブモデルバージョン (コンソール) に昇格させます。

1. AWS Management Console にサインインし、Amazon Comprehend コンソールを開きます。
2. 左側のメニューから [フライホイール] を選択します。
3. [フライホイール] テーブルからフライホイールを選択します。
4. [フライホイール] の詳細ページの表から、 [フライホイールイテレーション] から昇格させるバージョンを選択します。
5. [アクティブモデルを作成] を選択します。

イテレーションの管理 (API)

Amazon Comprehend API を使用して新しいイテレーションを開始し、進行中のイテレーションのステータスを問い合わせることができます。また、完了したイテレーションの結果を表示することもできます。

フライホイールイテレーションを開始する (API)

Amazon Comprehend StartFlywheelIterationオペレーションを使用して、フライホイールのイテレーションを開始します。

```
aws comprehend start-flywheel-iteration \
    --flywheel-arn  "flywheelArn"
```

レスポンスの内容は次のとおりです。

```
{
  "FlywheelIterationArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/name"
}
```

新しいモデルバージョンを昇格させる (API)

UpdateFlywheel オペレーションを使用して、モデルバージョンをアクティブなモデルバージョンに昇格させます。

ActiveModelArn パラメータを新しいアクティブモデルバージョンの ARN に設定した UpdateFlywheel リクエストを送信します。

```
aws comprehend update-flywheel \
    --active-model-arn  "modelArn" \
```

レスポンスの内容は次のとおりです。

```
{
  "FlywheelArn": "arn:aws::comprehend:aws-region:111122223333:flywheel/name",
  "ActiveModelArn": "modelArn"
}
```

フライホイールイテレーション結果の説明 (API)

Amazon Comprehend DescribeFlywheelIterationオペレーションは、実行から完了までの反復に関する情報を返します。

```
aws comprehend describe-flywheel-iteration \
 --flywheel-arn "flywheelArn" \
 --flywheel-iteration-id  "flywheelIterationId" \
 --region aws-region
```

レスポンスの内容は次のとおりです。

```
{
    "FlywheelIterationProperties": {
        "FlywheelArn": "flywheelArn",
        "FlywheelIterationId": "iterationId",
        "CreationTime": <createdAt>,
        "EndTime": <endedAt>,
        "Status": <status>,
        "Message": <message>,
        "EvaluatedModelArn": "modelArn",
        "EvaluatedModelMetrics": {
            "AverageF1Score": <value>,
            "AveragePrecision": <value>,
            "AverageRecall": <value>,
            "AverageAccuracy": <value>
        },
        "TrainedModelArn": "modelArn",
```

```
        "TrainedModelMetrics": {
            "AverageF1Score": <value>,
            "AveragePrecision": <value>,
            "AverageRecall": <value>,
            "AverageAccuracy": <value>
        }
    }
}
```

イテレーション履歴の取得 (API)

ListFlywheelIterationHistory オペレーションを使用して、反復履歴に関する情報を取得します。

```
aws comprehend list-flywheel-iteration-history \
 --flywheel-arn "flywheelArn"
```

レスポンスの内容は次のとおりです。

```
{
    "FlywheelIterationPropertiesList": [
        {
            "FlywheelArn": "<flywheelArn>",
            "FlywheelIterationId": "20220907T214613Z",
            "CreationTime": 1662587173.224,
            "EndTime": 1662592043.02,
            "Status": "<status>",
            "Message": "<message>",
            "EvaluatedModelArn": "modelArn",
            "EvaluatedModelMetrics": {
                "AverageF1Score": 0.8333333333333333,
                "AveragePrecision": 0.75,
                "AverageRecall": 0.9375,
                "AverageAccuracy": 0.8125
            },
            "TrainedModelArn": "modelArn",
            "TrainedModelMetrics": {
                "AverageF1Score": 0.865497076023392,
                "AveragePrecision": 0.7636363636363637,
                "AverageRecall": 1.0,
                "AverageAccuracy": 0.84375
            }
        }
    ]
```

```
}
```

フライホイールを使用した分析の実行

フライホイールのアクティブモデルバージョンを使用して、カスタム分類やエンティティ認識の分析を実行できます。　アクティブモデルバージョンを設定可能です。コンソールまたは UpdateFlywheel API オペレーションを使用して、モデルの新しいバージョンをアクティブなモデルバージョンに設定できます。

フライホイールを使用するには、解析タスクの設定時にカスタムモデル ARN の代わりにフライホイール ARN を指定します。　Amazon Comprehend は、フライホイールのアクティブモデルバージョンで分析を実行します。

リアルタイム分析

エンドポイントを使用して、リアルタイム分析を実行します。　エンドポイントを作成または更新するときに、モデル ARN の代わりにフライホイール ARN を使用してエンドポイントを設定できます。　リアルタイム分析を実行するときは、フライホイールに関連するエンドポイントを選択します。　Amazon Comprehend は、フライホイールのアクティブモデルバージョンを使って分析を実行します。

UpdateFlywheel を使用してフライホイールに新しいアクティブモデルバージョンを設定すると、エンドポイントは自動的に更新され、新しいアクティブモデルバージョンの使用が開始されます。エンドポイントを自動的に更新しない場合は、モデルバージョン ARN を直接使用するようにエンドポイントを (を使用してUpdateEndpoint) 設定します。フライホイールのアクティブモデルバージョンが変更されても、エンドポイントは引き続きこのモデルバージョンを使用します。

カスタム分類の場合は、 ClassifyDocument API オペレーションを使用します。カスタムエンティティ認識には、 DetectEntities API リクエストを使用します。EndpointArn パラメータでフライホイールのエンドポイントを指定します。

コンソールを使用して、カスタム分類やカスタムエンティティ認識のリアルタイム分析を実行することもできます。

非同期ジョブ

カスタム分類の場合は、 StartDocumentClassificationJob API リクエストを使用して aysnchronous ジョブを開始します。FlywheelArn の代わりに DocumentClassifierArn パラメータを指定します。

カスタムエンティティ認識には、 StartEntitiesDetectionJob API リクエストを使用します。FlywheelArn の代わりに EntityRecognizerArn パラメータを指定します。

コンソールを使用して、カスタム分類やカスタムエンティティ認識の非同期分析を実行することができます。ジョブを作成するときに、[レコグナイザーモデル] または [分類子モデル] フィールドにフライホイール ARN を入力します。

Amazon Comprehend のエンドポイントの管理

Amazon Comprehend では、お客様のカスタムモデルは、エンドポイントによってリアルタイムの分類やエンティティ検出に使用できるようになります。エンドポイントを作成すると、ビジネスニーズの変化に応じてそのエンドポイントに変更を加えることができます。たとえば、エンドポイントの使用状況を監視したり、自動スケーリングを適用して、キャパシティのニーズに合わせてエンドポイントのプロビジョニングを自動的に設定したりできます。すべてのエンドポイントを 1 つのビューで管理でき、エンドポイントが不要になったら削除してコストを節約できます。

エンドポイントを管理するには、エンドポイントを作成する必要があります。詳細については、次の手順を参照してください。

- カスタム分類用のエンドポイントの作成
- カスタムエンティティ検出用のエンドポイントの作成

トピック

- Amazon Comprehend エンドポイントの概要
- Amazon Comprehend エンドポイントの使用法
- Amazon Comprehend エンドポイントのモニタリング
- Amazon Comprehend のエンドポイントの更新
- Amazon Comprehend Trusted Advisor で を使用する
- Amazon Comprehend エンドポイントの削除
- 自動スケーリングとエンドポイント

Amazon Comprehend エンドポイントの概要

Amazon Comprehend コンソールのエンドポイントページには、エンドポイントのグローバルビューが表示されます。エンドポイント概要ページでは、すべてのエンドポイントを 1 か所に表示して、エンドポイントの使用状況と実際のリソース使用状況を把握できます。エンドポイントページの右上で、表示するエンドポイント (すべて、カスタム分類子のエンドポイント、またはカスタムエンティティエンドポイント) を指定できます。

このページで、エンドポイントを作成、更新、モニタリング、削除できます。エンドポイントの概要セクションでは、エンドポイントのリスト、エンドポイントがホストしているカスタムモデル、作成

時間、プロビジョニングされたスループット、エンドポイントのステータスを確認できます。エンドポイント概要テーブルから特定のエンドポイントを選択すると、エンドポイントの詳細が表示されます。

また、「AWS ビジネスサポート」または「AWS エンタープライズサポート」のお客様であれば、エンドポイント固有のTrusted Advisorチェックにアクセスできます。詳細については、「Amazon Comprehend Trusted Advisor で を使用する」を参照してください。チェックと説明の完全なリストについては、「Trusted Advisorのベストプラクティス」を参照してください。

エンドポイントの管理の詳細については、次のトピックを参照してください。

- Amazon Comprehend エンドポイントの使用法
- Amazon Comprehend エンドポイントのモニタリング
- Amazon Comprehend のエンドポイントの更新
- Amazon Comprehend Trusted Advisor で を使用する
- Amazon Comprehend エンドポイントの削除

> **Important**
>
> リアルタイムカスタム分類のコストは、設定したスループットとエンドポイントがアクティブである時間の両方に基づいて決まります。エンドポイントを使用しなくなったり、長期間使用しない場合は、自動スケーリングポリシーを設定してコストを削減する必要があります。また、エンドポイントを使用しなくなった場合は、追加コストが発生しないようにエンドポイントを削除できます。詳細については、「自動スケーリングとエンドポイント」を参照してください。

Amazon Comprehend エンドポイントの使用法

エンドポイントを作成することで、カスタムモデルを使用してリアルタイム分析を実行することができます。エンドポイントには、リアルタイム推論にカスタムモデルを使用できるようにする管理対象リソースが含まれます。

Amazon Comprehend は、推論単位 (IU) を使用してエンドポイントにスループットを割り当てます。IU は 1 秒あたり 100 文字のデータスループットを表します。エンドポイントには最大 10 個の推論ユニットをプロビジョニングできます。エンドポイントのスループットは、エンドポイントを更新することで拡大することも、縮小することもできます。

入力ドキュメントに半構造化ドキュメントまたは画像ファイルが含まれている場合、入力ファイルから抽出された文字に対するスループットは 1 秒あたり 100 文字になります。エンドポイントにプロビジョニングする IU の数は、入力ドキュメントの文字密度によって異なります。

ClassifyDocument および DetectEntities API レスポンスには、入力の各ページの文字数が含まれます。この情報を使用して、目的のスループットを達成するためにプロビジョニングする推論ユニット数を見積もることができます。

リアルタイム分析が完了したら、エンドポイントを削除してください。エンドポイントをアクティブである限り、課金が続きます。さらにリアルタイム分析を実行する準備ができると、別のエンドポイントを作成できます。

詳細については、「Amazon Comprehend の料金」を参照してください。

エンドポイントを作成したら、Amazon でモニタリングしたり CloudWatch、推論単位を変更したり、不要になったら削除したりできます。詳細については、「Amazon Comprehend エンドポイントのモニタリング」を参照してください。

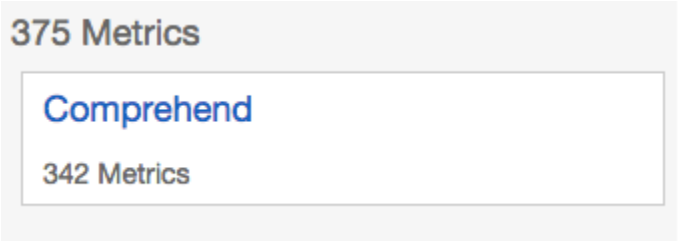
Amazon Comprehend エンドポイントのモニタリング

推論単位 (IUsの数を増減することで、エンドポイントのスループットを調整できます。エンドポイントの更新の詳細については、「the section called “エンドポイントの更新”」を参照してください。

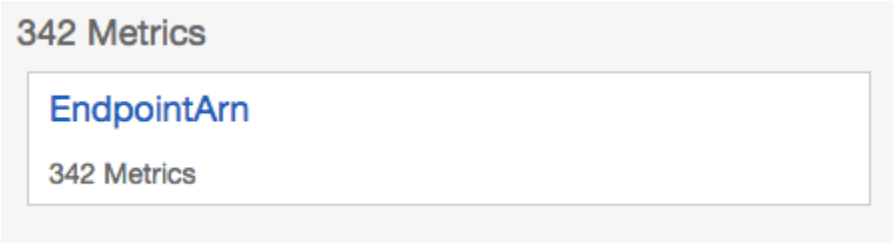
Amazon CloudWatch コンソールで使用状況を監視することで、エンドポイントのスループットを最適に調整する方法を決定できます。

でエンドポイントの使用状況を監視する CloudWatch

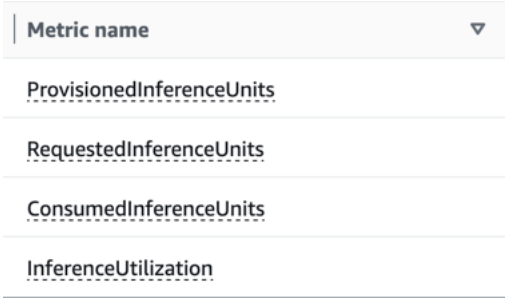
1. にサインイン AWS Management Console し、 CloudWatch コンソールを開きます。
2. 左の [指標] を選択し、次に [すべての指標] を選択します。
3. [すべての指標] で [理解] を選択します。



4. CloudWatch コンソールには、Comprehend メトリクスのディメンションが表示されます。EndpointArn ディメンションを選択します。



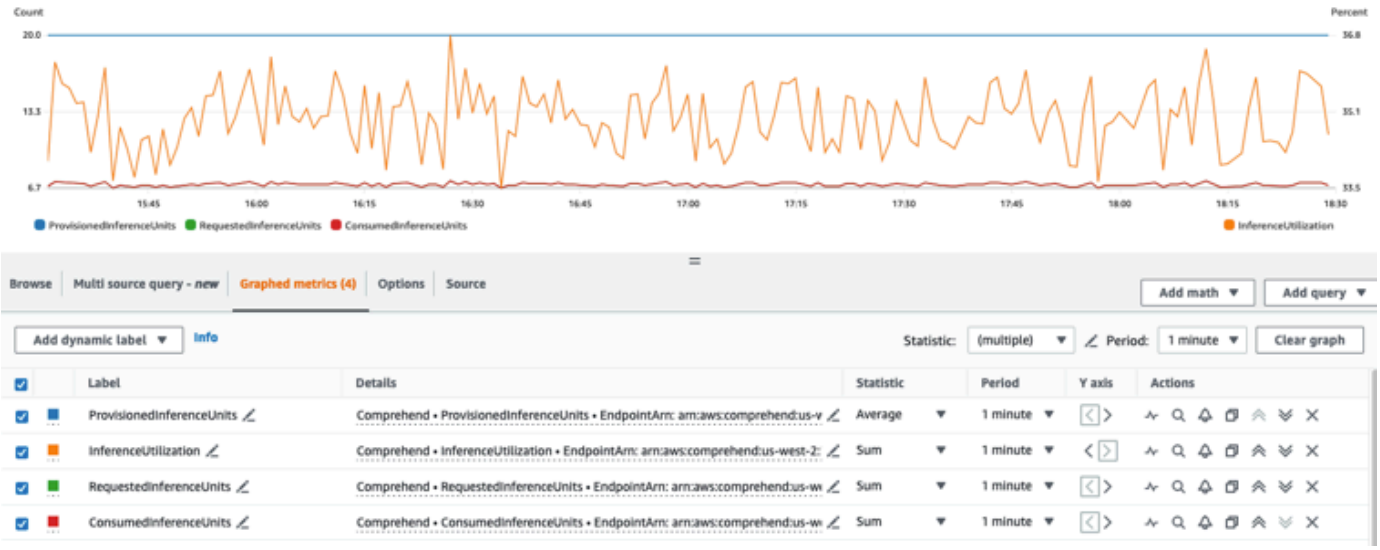
コンソールにはProvisionedInferenceUnits、エンドポイントInferenceUtilizationごとにRequestedInferenceUnits、ConsumedInferenceUnits、、および が表示されます。



4 つのメトリクスを選択し、グラフ化されたメトリクスタブに移動します。

5. RequestedInferenceUnits および の統計列ConsumedInferenceUnitsを Sum に設定します。
6. の統計列を SumInferenceUtilization に設定します。
7. の統計列ProvisionedInferenceUnitsを Average に設定します。
8. すべての指標の期間列を 1 分に変更します。
9. 矢印を選択してInferenceUtilization選択し、別の Y TAK に移動します。

これでグラフは分析できる状態になりました。



CloudWatch メトリクスに基づいて、エンドポイントのスループットを自動的に調整するように自動スケーリングを設定することもできます。エンドポイントを使用した自動スケーリングの使用に関する詳細については、「自動スケーリングとエンドポイント」を参照してください。

- ProvisionedInferenceUnits - このメトリクスは、リクエストが行われた時点のプロビジョニング済み IUs の平均数を表します。
- RequestedInferenceUnits - これは、処理のために送信されたサービスに送信された各リクエストの使用状況に基づいています。これは、送信されたリクエストを、スロットリングなしで実際に処理されたリクエスト () と比較するのに役立ちますConsumedInferenceUnits。このメトリクスの値は、処理するために送信される文字数を 1 分間に処理できる文字数で割って計算されます。
- ConsumedInferenceUnits - これは、正常に処理された (スロットリングされていない) サービスに送信された各リクエストの使用状況に基づいています。これは、消費しているものを準備された IU と比較するときに便利です。この指標の値は、処理された文字数を 1 IU の 1 分間に処理できる文字数で割って計算されます。
- InferenceUtilization - これはリクエストごとに発行されます。この値は、 で定義されている消費された IUsを取得しConsumedInferenceUnits、 で除算ProvisionedInferenceUnitsして 100 のパーセンテージに変換することによって計算されます。

> **Note**
>
> すべての指標は、リクエストが成功した場合にのみ出力されます。スロットリングされたリクエスト、内部サーバーエラーや顧客エラーにより失敗したリクエストからの指標は表示されません。

Amazon Comprehend のエンドポイントの更新

通常、求める必要なスループットのレベルは、エンドポイントの作成後に変化することはよくあることです。当初のニーズ予測も変化します。そのような場合は、エンドポイントを更新してスループットを増減する必要があります。スループットは、エンドポイントにプロビジョニングした推論ユニットの数によって決まります。各ユニットは、1 秒あたり最大 2 つの文書に対して毎秒 100 文字/秒のスループットに相当します。また、エンドポイントに関連付けられているモデルのバージョンを更新することもできます。エンドポイントの編集では、エンドポイントに対して別のバージョンのモデルを選択できます。

また、エンドポイントにタグを追加して整理しやすくすることもできます。タグ付けは、エンドポイントの更新時に行うこともできます。エンドポイントの詳細については、「リソースのタグ付け」を参照してください。

エンドポイントを更新する (コンソール)

1. AWS Management Console にサインインして、Amazon Comprehend コンソール (https://console. .amazon.com/comprehend/) を開きます
2. 左側のメニューで、[エンドポイント] を選択します。
3. [分類子] の一覧から、更新するエンドポイントがあるカスタムモデルの名前を選択し、リンクを辿ります。モデルの詳細ページが表示されます。
4. モデルの詳細ページから、バージョン詳細を選択します。エンドポイントリストが表示されます。
5. 目的のエンドポイントのエンドポイントチェックボックスを選択します。エンドポイントテーブルの右上にある [アクション] アイコンを選択します。
6. [編集] を選択します。プロビジョニングされている IU を更新したり、タグを編集したりできます。
7. 変更を保存します。
8. エンドポイントにプロビジョニングする推論ユニットの数を編集するには、[編集] を選択します。
9. エンドポイントに割り当てる推論ユニット (IU) の新しい個数を入力します。ユニット 1 つは 1 秒あたり 100 文字のデータスループットに相当します。1 エンドポイントあたり最大 10 個の推論ユニットを割り当てることができます。

   > **Note**
   >
   > エンドポイントの使用コストは、動作時間とスループット (推論ユニットの個数に基づく) に基づきます。したがって、推論ユニットの数を増やすと、運用コストも増加します。詳細については、「Amazon Comprehend の料金」を参照してください。

10. [エンドポイントの編集] を選択します。エンドポイントの詳細ページが表示されます。
11. ページ上部にあるパンくずリストからモデル名を選択して、エンドポイントが更新されていることを確認します。カスタムモデルの詳細ページで、エンドポイントのリストに移動し、エンドポイントの横に 更新中 と表示されていることを確認します。更新が完了すると、準備完了と表示されます。

次の例は、 CLI で AWS UpdateEndpointオペレーションを使用する方法を示しています。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend update-endpoint \
    --desired-inference-units updated number of inference units \
    --desired-model-arn arn:aws:comprehend:region:account-id:model type/model name
 \
    --desired-data-access-role-arn arn:aws:iam:account id:role/role name
    --endpoint-arn arn:aws:comprehend:region:account id:endpoint/endpoint name
```

アクションが成功すると、そのレスポンスとして、Amazon Comprehend から HTTP 200 レスポンスと空の HTTP 本文が返します。

12. エンドポイントにアタッチしたカスタムモデルを編集するには、カスタムモデルの詳細ページからエンドポイントのリストに移動します。
13. 変更するエンドポイントを選択し、[編集] を選択します。
14. エンドポイント設定ページの [分類子モデルの選択] または [レコグナイザーモデルの選択] のドロップダウンからモデルを検索することができます。目的のモデルを選択します。
15. [バージョンの選択] では、目的のモデルバージョンを検索することができます。バージョンを選択します。
16. [エンドポイントの編集] を選択して保存します。

Amazon Comprehend Trusted Advisor で を使用する

AWS Trusted Advisor は、 AWS のベストプラクティスに従ってリソースをプロビジョニングするのに役立つレコメンデーションを提供するオンラインツールです。

ベーシックサポートプランまたはデベロッパーサポートプランをご利用の場合は、 Trusted Advisor コンソールを使用して、サービス制限カテゴリのすべてのチェックとセキュリティカテゴリの 6 つのチェックにアクセスできます。ビジネスまたはエンタープライズサポートプランをお持ちの場合は、 Trusted Advisor コンソールと AWS Support API を使用してすべての Trusted Advisor チェックにアクセスできます。

Amazon Comprehend は、以下の Trusted Advisor チェックをサポートしており、お客様が実用的な推奨事項を提供することで、Amazon Comprehend エンドポイントのコストとセキュリティを最適化するのに役立ちます。

Amazon Comprehend の使用率の低いエンドポイント

Amazon Comprehend の使用率の低いエンドポイントチェックでは、エンドポイントのスループット構成をチェックします。このチェックでは、エンドポイントがリアルタイム推論リクエストでアクティブに使用されていない場合に警告が表示されます。15 日以上使用されないエンドポイントは十分に使用されていないと考えられます。すべてのエンドポイントは、スループットセット、およびエンドポイントがアクティブである時間の長さの両方に基づいて料金が発生します。過去 15 日間使用されていないエンドポイントは、Application Autoscaling を使用してリソースのスケーリングポリシーを定義することをお勧めします。過去 30 日間使用されておらず、自動スケーリングポリシーが定義されているエンドポイントについては、非同期推論を使用するか、削除することをお勧めします。これらのチェック結果は 1 日 1 回自動的に更新され、コンソールの CostOptimizationカテゴリで Trusted Advisor 表示できます。

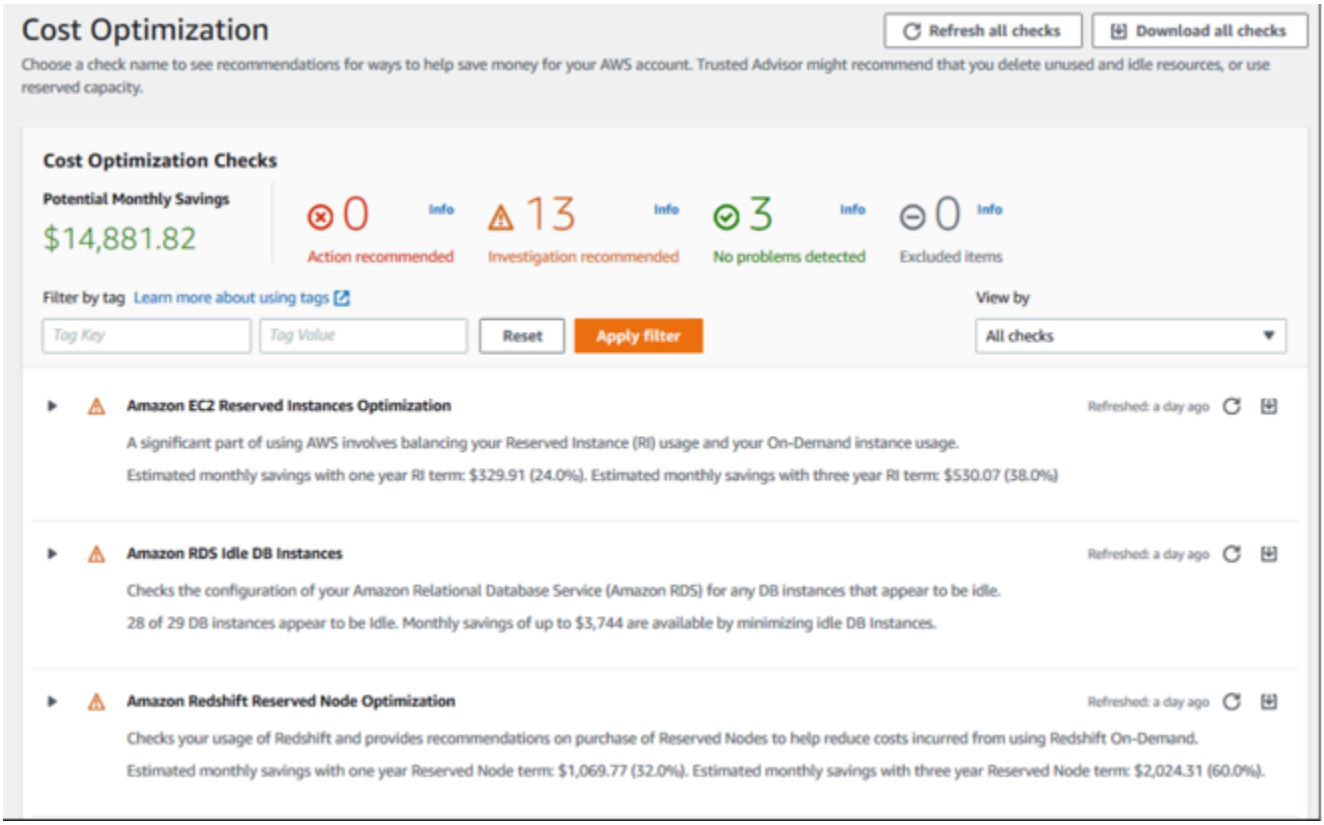
すべてのエンドポイントの使用状況とそれに対応するレコメンデーションを確認するには

1. にサインイン AWS Management Console し、 Trusted Advisor コンソールを開きます。
2. ナビゲーションペインで、CostOptimizationチェックカテゴリを選択します。
3. カテゴリページでは、各チェックカテゴリの概要が表示されます。
   - 推奨アクション (赤) — Trusted Advisor チェック用のアクションを推奨します。
   - [調査が推奨されるチェック項目 (黄色)] – Trusted Advisor は、チェックの潜在的な問題を検出します。
   - 問題は検出されませんでした (緑) — Trusted Advisor チェックの問題は検出されません。
   - 非表示の項目 (グレー) - チェックで無視するリソースなど、除外項目があるチェックの数。
4. Amazon Comprehend 使用率の低いエンドポイントチェックを選択すると、チェックの説明と以下の詳細が表示されます。
   - アラート基準 — チェックのステータスが変更されるときのしきい値を示します。
   - 推奨されるアクション — そのチェックの推奨アクションを示します。
   - リソーステーブル: レコメンデーションに基づいてエンドポイントの詳細と各エンドポイントのステータスを一覧表示した表。
5. リソーステーブルで、「過去 30 日間使用されていません」という警告により、エンドポイントに「調査推奨」のフラグが付けられている場合は、Amazon Comprehend コンソールの「エンドポイントの詳細」ページに移動できます。
   - このエンドポイントをこれ以上使用しない場合は、[削除] を選択します。

- [削除] をもう一度選択して、削除を確定します。カスタムモデルの詳細 ページが表示されます。削除したエンドポイントの横に「削除中」と表示されていることを確認します。削除すると、エンドポイントは [エンドポイント]リストから削除されます。

6. Trusted Advisor コンソールのリソーステーブルで、過去 15 日間使用されていないためにエンドポイントに調査推奨ステータスのフラグが付けられていて、 AutoScaling 無効にしている場合は、Amazon Comprehend コンソールのエンドポイント詳細ページに移動してエンドポイントを調整できます。
   - このエンドポイントに構成されているスループットを下げたい場合は、[編集]をクリックします。エンドポイントに割り当てる最新の推論ユニット数を入力し、確認するチェックボックスを選択して [エンドポイントを編集] を選択します。更新が完了すると、ステータスが [準備完了] と表示されます。
   - スループット構成を手動で調整するのではなく、エンドポイントに自動的にエンドポイントプロビジョニングを設定したい場合は、[アプリケーション自動スケーリング] を使用することをおすすめします。
7. Trusted Advisor コンソールのリソーステーブルで、使用中のアクティブの理由によりエンドポイントに問題が検出されないステータスのフラグが付けられている場合、エンドポイントがリアルタイム推論リクエストの実行にアクティブに使用されていて、アクションは推奨されません。

コンソールの CostOptimization カテゴリビューを表示する例を Trusted Advisor 次に示します。



Amazon Comprehend エンドポイントアクセスリスク

[Amazon Comprehend エンドポイントアクセスリスク] チェックは、基盤となるモデルがカスタマーマネージドキーを使用して暗号化されたエンドポイントの AWS Key Management Service (AWS KMS) キー許可を確認します。カスタマーマネージドキーが無効になっている場合、または、Amazon Comprehend の付与された許可を変更するようにキーポリシーが変更された場合、エンドポイントの可用性が影響を受ける可能性があります。キーが無効になっている場合は、有効にすることをお勧めします。キーポリシーが変更され、エンドポイントを引き続き使用する場合は、キーポリシーを更新することをお勧めします。チェック結果は 1 日に複数回、自動的に更新されます。このチェックは、 Trusted Advisor コンソールの [耐障害性] カテゴリで確認できます。

Amazon Comprehend エンドポイントの AWS KMS キーステータスを表示するには

1. にサインイン AWS Management Console し、 Trusted Advisor コンソールを開きます。
2. ナビゲーションペインで、FaultToleranceチェックカテゴリを選択します。
3. カテゴリページでは、各チェックカテゴリの概要が表示されます。

    - 推奨アクション (赤) — Trusted Advisor チェック用のアクションを推奨します。

- 調査が推奨されるチェック項目 (黄色) - Trusted Advisor は、チェックの潜在的な問題を検出します。
- 問題は検出されませんでした (緑) — Trusted Advisor チェックの問題は検出されません。
- [非表示の項目 (グレー)] — チェックで無視するリソースなど、除外項目があるチェックの数。

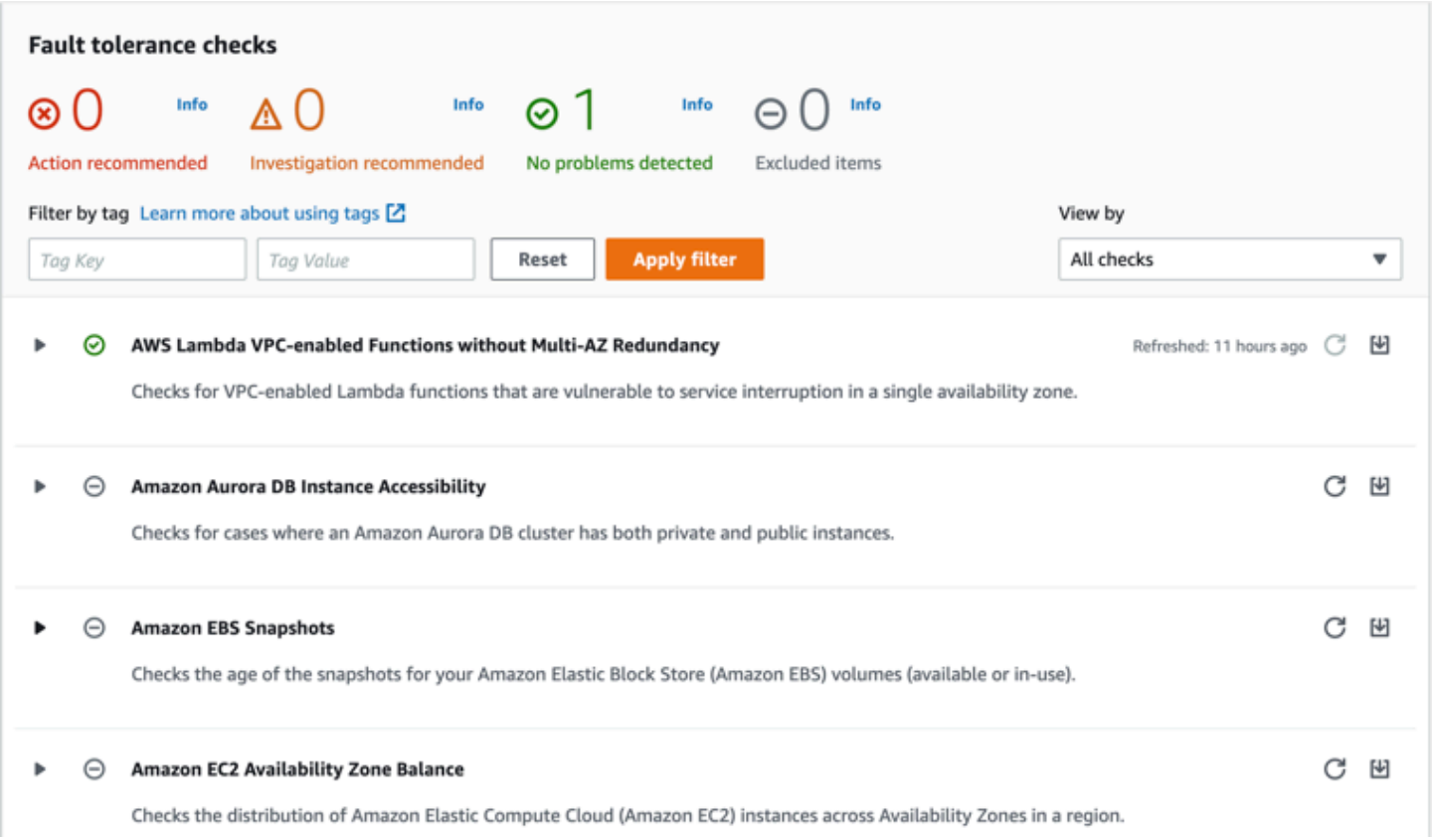
4. Amazon Comprehend エンドポイントアクセスリスクチェックを選択すると、チェックの説明と以下の詳細を表示できます。

   - アラート基準 — チェックのステータスが変更されるときのしきい値を示します。
   - 推奨されるアクション — そのチェックの推奨アクションを示します。
   - リソーステーブル: KMS 暗号化エンドポイントの詳細と、推奨アクションがあるかどうかに基づいて各エンドポイントのステータスを一覧表示した表。

5. リソーステーブルで、エンドポイントにアクション推奨ステータスのフラグが付けられている場合は、KMS KeyId 列のリンクを選択すると、対応する AWS KMS キーページにリダイレクトされます。

   - 無効化された AWS KMS キーを有効にするには、キーアクション を選択し、有効化 を選択します。
   - キーステータスが [有効] と表示されている場合は、「キーポリシー」セクションの [ポリシービューに切り替え] を選択してキーポリシーを更新します。キーポリシードキュメントを編集して Amazon Comprehend に必要なアクセス権限を付与し、[変更を保存] を選択します。

Trusted Advisor コンソールの FaultTolerance カテゴリビューの例を次に示します。



これらのチェックとその結果は、 AWS Support API の Trusted Advisor セクションを参照して表示することもできます。

を使用したアラームの設定の詳細については CloudWatch、「: を使用した Trusted Advisor アラームの作成 CloudWatch」を参照してください。 Trusted Advisor ベストプラクティスチェックの完全なセットについては、AWS Trusted Advisor 「ベストプラクティスチェックリスト」を参照してください。

Amazon Comprehend エンドポイントの削除

エンドポイントが不要になったなら、コストが発生しないように削除すべきです。必要になったなときにエンドポイントセクションで、いつでも別のエンドポイントを簡単に作成できます。

エンドポイントを削除するには (コンソール)

1. AWS Management Console にサインインして、Amazon Comprehend コンソール (https://console. .amazon.com/comprehend/) を開きます
2. 左側のメニューで、[エンドポイント] を選択します。

3. エンドポイントの表から、削除するエンドポイントを見つけます。すべてのエンドポイントを検索またはフィルタリングして、必要なエンドポイントを見つけることができます。
4. エンドポイントのリストから、削除するエンドポイントを選択します。エンドポイントテーブルの右上にある [アクション] アイコンを選択します。
5. [削除] を選択します。
6. [削除] をもう一度選択して、削除を確定します。エンドポイントページが表示されます。削除したエンドポイントの横に「削除中」と表示されていることを確認します。削除すると、エンドポイントはエンドポイント リストからなくなります。

エンドポイントを削除するには (AWS CLI)

次の例は、 CLI で AWS DeleteEndpointオペレーションを使用する方法を示しています。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws comprehend delete-endpoint \
    --endpoint-arn arn:aws:comprehend:region:account-id endpoint/endpoint name
```

アクションが成功すると、そのレスポンスとして、Amazon Comprehend から HTTP 200 レスポンスと空の HTTP 本文が返します。

自動スケーリングとエンドポイント

文書分類エンドポイントとエンティティ認識エンドポイントにプロビジョニングする推論ユニットの数を手動で調整するのではなく、自動スケーリングを利用することで、容量のニーズに合わせてエンドポイントのプロビジョニングを自動的に設定することができます。

自動スケーリングを利用してエンドポイントにプロビジョニングする推論ユニットの数を調節する方法は 2 通りあります。

- ターゲット追跡: 使用状況に基づき、容量ニーズに合わせてエンドポイントプロビジョニングを調整するよう設定します。
- スケジュールされたスケーリング: 指定したスケジュールに従い、容量ニーズに合わせてエンドポイントプロビジョニングを調整するよう設定します。

自動スケーリングの設定は AWS Command Line Interface (AWS CLI) を使用してのみ行うことができます。自動スケーリングの詳細については、「アプリケーションの自動スケーリングとは」を参照してください。

ターゲット追跡

ターゲット追跡を使用すると、使用状況に基づき容量ニーズに合わせてエンドポイントプロビジョニングを調整できます。推論ユニットの数は、利用された容量が提供された容量の目標パーセント内に収まるよう自動的に調整されます。ターゲット追跡を利用すると、ドキュメント分類エンドポイントとエンティティレコグナイザーエンドポイントの一時的な使用量の急増に対応できます。詳細については、「Application Auto Scaling のターゲット追跡スケーリングポリシー」を参照してください。

> **Note**
>
> 次の例は、Unix、Linux、および macOS 用の形式になっています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

ターゲット追跡のセットアップ

エンドポイントに対してターゲット追跡のセットアップをするには、AWS CLI コマンドを使用してスケーラブルターゲットを登録し、スケーリングポリシーを作成します。スケーラブルターゲットではエンドポイントプロビジョニングの調整に使用するリソースとして推論ユニットを定義し、スケーリングポリシーではプロビジョニング済みポリシーの自動スケーリングを制御するメトリックを定義します。

ターゲット追跡をセットアップする

1. スケーラブルターゲットを登録します。以下の例では、スケーラブルターゲットを登録することでエンドポイントプロビジョニングを調整しています。最小容量は推論ユニット 1 つ、最大容量は推論ユニット 2 つです。

   ドキュメント分類エンドポイントの場合は、次の AWS CLI コマンドを使用します。

   ```
   aws application-autoscaling register-scalable-target \
       --service-namespace comprehend \
       --resource-id arn:aws:comprehend:region:account-id:document-classifier-
   endpoint/name \
       --scalable-dimension comprehend:document-classifier-
   endpoint:DesiredInferenceUnits \
   ```

```
    --min-capacity 1 \
    --max-capacity 2
```

エンティティレコグナイザーエンドポイントの場合は、次の AWS CLI コマンドを使用します。

```
aws application-autoscaling register-scalable-target \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
endpoint/name \
    --scalable-dimension comprehend:entity-recognizer-
endpoint:DesiredInferenceUnits \
    --min-capacity 1 \
    --max-capacity 2
```

2. スケーラブルターゲットが登録されていることを確認するには、次の AWS CLI コマンドを使用します。

```
aws application-autoscaling describe-scalable-targets \
    --service-namespace comprehend \
    --resource-id endpoint ARN
```

3. スケーリングポリシーに対するターゲット追跡設定を作成し、config.json いう名前のファイルに保存します。以下は、使用容量が常にプロビジョニングされている容量の 70% になるように推論ユニットの数を自動的に調整するターゲット追跡設定の例です。

```
{
  "TargetValue": 70,
  "PredefinedMetricSpecification":
  {
  "PredefinedMetricType": "ComprehendInferenceUtilization"
  }
}
```

4. スケーリングポリシーを作成します。以下の例では、config.json ファイルに定義されているターゲット追跡設定に基づくスケーリングポリシーを作成しています。

   ドキュメント分類エンドポイントの場合は、次の AWS CLI コマンドを使用します。

```
aws application-autoscaling put-scaling-policy \
```

```
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:document-classifier-
endpoint/name \
    --scalable-dimension comprehend:document-classifier-
endpoint:DesiredInferenceUnits \
    --policy-name TestPolicy \
    --policy-type TargetTrackingScaling \
    --target-tracking-scaling-policy-configuration file://config.json
```

エンティティレコグナイザーエンドポイントの場合は、次の AWS CLI コマンドを使用します。

```
aws application-autoscaling put-scaling-policy \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
endpoint/name \
    --scalable-dimension comprehend:entity-recognizer-
endpoint:DesiredInferenceUnits \
    --policy-name TestPolicy \
    --policy-type TargetTrackingScaling \
    --target-tracking-scaling-policy-configuration file://config.json
```

ターゲット追跡の削除

エンドポイントのターゲット追跡を削除するには、AWS CLI コマンドを使用して、スケーラブルターゲットを削除し、スケーラブルターゲットの登録を解除します。

ターゲット追跡を削除する

1. スケーリングポリシーを削除します。次の例では、指定したスケーリングポリシーを削除しています。

   ドキュメント分類エンドポイントの場合は、次の AWS CLI コマンドを使用します。

```
aws application-autoscaling delete-scaling-policy \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:document-classifier-
endpoint/name \
    --scalable-dimension comprehend:document-classifier-
endpoint:DesiredInferenceUnits \
    --policy-name TestPolicy \
```

エンティティレコグナイザーエンドポイントの場合は、次の AWS CLI コマンドを使用します。

```
aws application-autoscaling delete-scaling-policy \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
endpoint/name \
    --scalable-dimension comprehend:entity-recognizer-
endpoint:DesiredInferenceUnits \
    --policy-name TestPolicy
```

2. スケーラブルなターゲットを登録解除します。以下の例では、指定したスケーラブルターゲットを登録解除しています。

   ドキュメント分類エンドポイントの場合は、次の AWS CLI コマンドを使用します。

```
aws application-autoscaling deregister-scalable-target \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:document-classifier-
endpoint/name \
    --scalable-dimension comprehend:document-classifier-
endpoint:DesiredInferenceUnits
```

   エンティティレコグナイザーエンドポイントの場合は、次の AWS CLI コマンドを使用します。

```
aws application-autoscaling deregister-scalable-target \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
endpoint/name \
    --scalable-dimension comprehend:entity-recognizer-
endpoint:DesiredInferenceUnits
```

スケジュールされたスケーリング

スケーリングスケジュールを利用すると、指定したスケジュールに従い、容量ニーズに合わせてエンドポイントプロビジョニングを調整することができます。スケーリングスケジュールでは、特定の時間帯での使用量の急増に応じて推論ユニットの数を自動的に調整することができます。ドキュメント分類エンドポイントおよびエンティティレコグナイザーエンドポイントには、スケーリング

スケジュール機能を利用することができます。スケーリングスケジュール機能の詳細については、「Application Auto Scaling におけるスケーリングスケジュール」を参照してください。

> **Note**
>
> 次の例は、Unix、Linux、および macOS 用の形式になっています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

スケーリングスケジュールのセットアップ

エンドポイントのターゲット追跡をセットアップするには、AWS CLI コマンドを使用してスケーラブルターゲットを登録し、アクションスケジュールを作成します。スケーラブルターゲットではエンドポイントプロビジョニングの調整に使用するリソースとして推論ユニットを定義し、アクションのスケジュールで特定の時間帯のプロビジョニング済み容量の自動スケーリングを制御します。

スケーリングスケジュールをセットアップする

1. スケーラブルターゲットを登録します。以下の例では、スケーラブルターゲットを登録することでエンドポイントプロビジョニングを調整しています。最小容量は推論ユニット 1 つ、最大容量は推論ユニット 2 つです。

   ドキュメント分類エンドポイントの場合は、次の AWS CLI コマンドを使用します。

   ```
   aws application-autoscaling register-scalable-target \
       --service-namespace comprehend \
       --resource-id arn:aws:comprehend:region:account-id:document-classifier-
   endpoint/name \
       --scalable-dimension comprehend:document-classifier-
   endpoint:DesiredInferenceUnits \
       --min-capacity 1 \
       --max-capacity 2
   ```

   エンティティレコグナイザーエンドポイントの場合は、次の AWS CLI コマンドを使用します。

   ```
   aws application-autoscaling register-scalable-target \
       --service-namespace comprehend \
       --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
   endpoint/name \
       --scalable-dimension comprehend:entity-recognizer-
   endpoint:DesiredInferenceUnits \
   ```

```
    --min-capacity 1 \
    --max-capacity 2
```

2. アクションスケジュールを作成します。以下の例では、毎日 12:00 UTC にプロビジョニング済み容量を自動的に調整するアクションスケジュールを作成しています。推論ユニットは最小 2 個、最大は 5 個です。時系列式とスケーリングスケジュールの詳細は、「スケジュール式」を参照してください。

   ドキュメント分類エンドポイントの場合は、次の AWS CLI コマンドを使用します。

```
aws application-autoscaling put-scheduled-action \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:document-classifier-
endpoint/name \
    --scalable-dimension comprehend:document-classifier-
endpoint:DesiredInferenceUnits \
    --scheduled-action-name TestScheduledAction \
    --schedule "cron(0 12 * * ? *)" \
    --scalable-target-action MinCapacity=2,MaxCapacity=5
```

   エンティティレコグナイザーエンドポイントの場合は、次の AWS CLI コマンドを使用します。

```
aws application-autoscaling put-scheduled-action \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
endpoint/name \
    --scalable-dimension comprehend:entity-recognizer-
endpoint:DesiredInferenceUnits \
    --scheduled-action-name TestScheduledAction \
    --schedule "cron(0 12 * * ? *)" \
    --scalable-target-action MinCapacity=2,MaxCapacity=5
```

スケーリングスケジュールの削除

エンドポイントに対するスケーリングスケジュールを削除するには、AWS CLI コマンドでアクションのスケジュールを削除し、スケーラブルターゲットを登録解除します。

スケーリングスケジュールを削除する

1. delete-scheduled-action 以下の例では、指定したアクションスケジュールを削除しています。

ドキュメント分類エンドポイントの場合は、次の AWS CLI コマンドを使用します。

```
aws application-autoscaling delete-scheduled-action \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:document-classifier-
endpoint/name \
    --scalable-dimension comprehend:document-classifier-
endpoint:DesiredInferenceUnits \
    --scheduled-action-name TestScheduledAction
```

エンティティレコグナイザーエンドポイントの場合は、次の AWS CLI コマンドを使用します。

```
aws application-autoscaling delete-scheduled-action \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
endpoint/name \
    --scalable-dimension comprehend:entity-recognizer-
endpoint:DesiredInferenceUnits \
    --scheduled-action-name TestScheduledAction
```

2. スケーラブルなターゲットを登録解除します。以下の例では、指定したスケーラブルターゲットを登録解除しています。

   ドキュメント分類エンドポイントの場合は、次の AWS CLI コマンドを使用します。

```
aws application-autoscaling deregister-scalable-target \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:document-classifier-
endpoint/name \
    --scalable-dimension comprehend:document-classifier-
endpoint:DesiredInferenceUnits
```

   エンティティレコグナイザーエンドポイントの場合は、次の AWS CLI コマンドを使用します。

```
aws application-autoscaling deregister-scalable-target \
    --service-namespace comprehend \
    --resource-id arn:aws:comprehend:region:account-id:entity-recognizer-
endpoint/name \
    --scalable-dimension comprehend:entity-recognizer-
endpoint:DesiredInferenceUnits
```

リソースのタグ付け

タグは、Amazon Comprehend リソースにメタデータとして追加できるキーと値のペアです。ジョブの分析 、カスタム分類モデル 、カスタムエンティティ認識モデル、エンドポイントにタグを追加できます。タグには主に 2 つの機能があります。1 つはリソースの整理で、他方はタグベースのアクセス制御です。

リソースをタグで整理するには、タグキー「Department」とタグ値「Sales」または「Legal」を追加します。その後、会社の法務部門に関するリソースを検索して絞り込むことができます。

タグベースのアクセス制御を実現するには、タグに基づく権限を持つ IAM ポリシーを作成します。ポリシーでは、リクエストで指定されたタグ (request-tags) または呼び出すリソースに関連するタグ (resource-tags) に基づいて演算を許可または禁止できます。IAM を指定してタグを使用する方法については、「IAM ユーザーガイド」の「タグを使用してアクセスを制御する」を参照してください。

Amazon Comprehend でタグを使用する際の考慮事項：

- リソースごとに最大 50 個のタグを追加でき、タグはリソースの作成時に追加することも、遡って追加することもできます。
- タグキーは必須フィールドですが、タグ値はオプションです。
- タグはリソース間で一意である必要はありませんが、特定のリソースに重複するタグキーは設定できません。
- タグのキーと値では、大文字と小文字が区別されます。
- タグキーは最大 127 文字、タグ値は最大 255 文字です。
- 「aws:」プレフィックスは AWS 用に予約されています。aws: で始まるキーのタグの追加・編集・削除はできません。これらのタグは、 tags-per-resource 上限の 50 個にはカウントされません。

> **Note**
>
> 複数の AWS サービス間およびリソース間でタグ付けスキーマを使用する場合、他のサービスで許可される文字の制限が異なることがあるのでご注意ください。

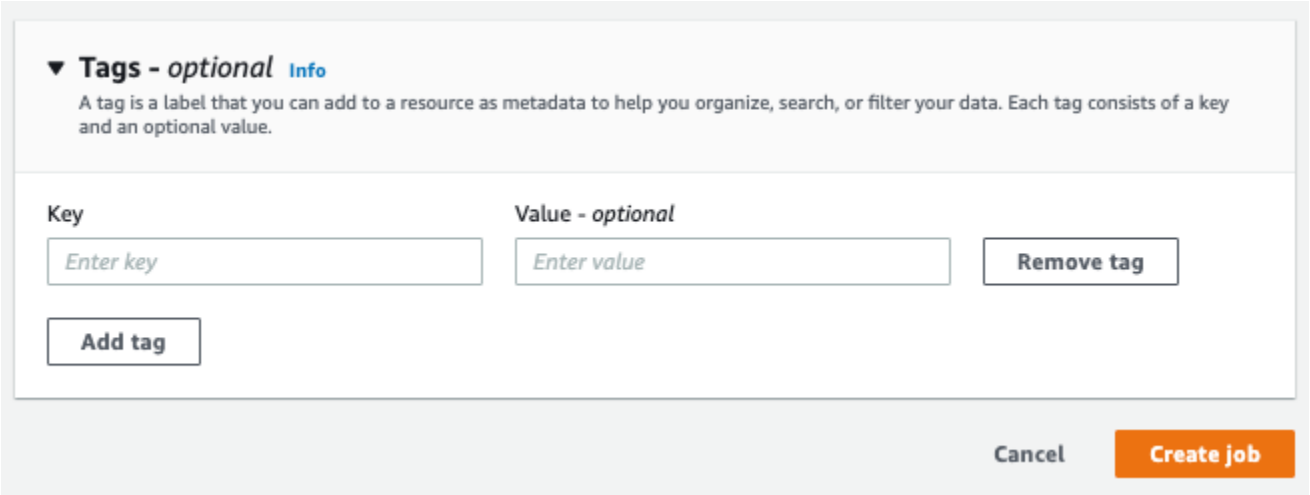
トピック

- 新しいリソースへのタグ付け
- リソースに関連付けられているタグの表示、編集、削除

新しいリソースへのタグ付け

分析ジョブやカスタム分類モデル 、 カスタムエンティティ認識モデル、エンドポイント にタグを付けることができます。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/comprehend/ で Amazon Comprehend コンソールを開きます。
2. 左側のナビゲーションペインから、作成するリソース (分析ジョブ、カスタム分類、またはカスタムエンティティ認識) を選択します。
3. [ジョブを作成] (または [新規モデル作成]) をクリックします。これにより、選択したリソース作成用のメインページが表示されます。このページの下部に「タグ - オプション」パネルが表示されます。



   タグキーおよびタグ値 (オプション) を入力します。[タグを追加] を選択して、リソースに別のタグを追加します。すべてのタグを追加するまで、このプロセスを繰り返してください。タグキーはリソースごとに一意である必要があります。
4. [作成] または [ジョブの作成] ボタンを選択して、リソースの作成を続けます。

AWS CLI を使用してタグを追加することもできます。この例では、 start-entities-detection-job コマンドでタグを追加する方法を示します。

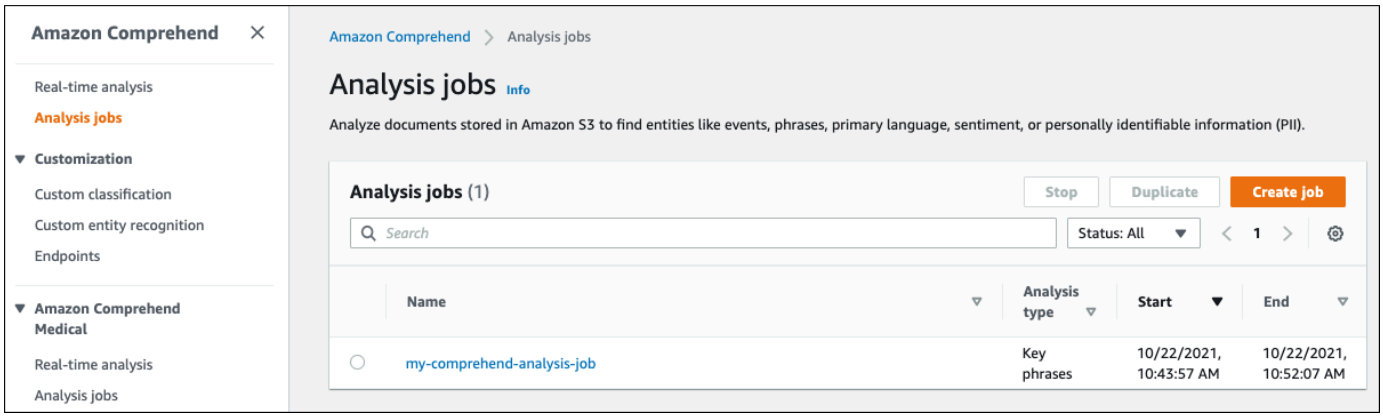
```
aws comprehend start-entities-detection-job \
```

```
--language-code "en" \
--input-data-config "{\"S3Uri\": \"s3://test-input/TEST.csv\"}" \
--output-data-config "{\"S3Uri\": \"s3://test-output\"}" \
--data-access-role-arn arn:aws:iam::123456789012:role/test \
--tags "[{\"Key\": \"color\",\"Value\": \"orange\"}]"
```

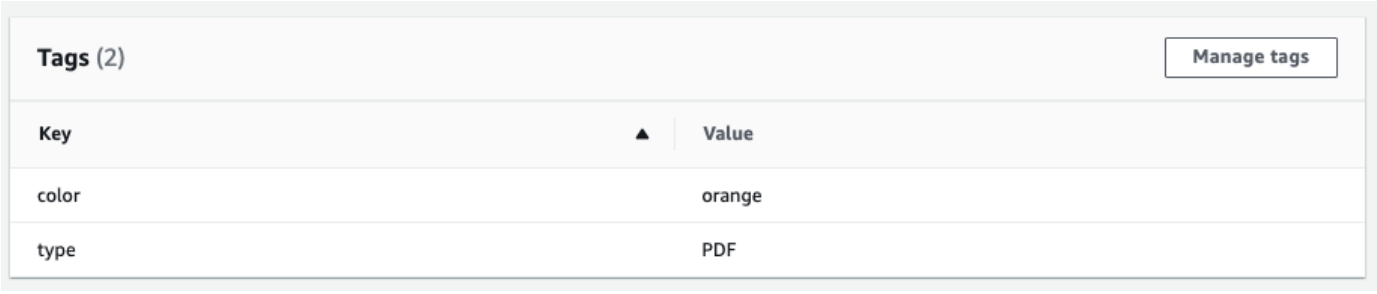
リソースに関連付けられているタグの表示、編集、削除

[分析ジョブ] 、 [カスタム分類モデル] 、[カスタムエンティティ認識]モデルに関連付けられたタグを表示できます。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/comprehend/ で Amazon Comprehend コンソールを開きます。
2. 表示、変更、または削除するタグを含むファイルが含まれたリソース (分析ジョブ、カスタム分類、またはカスタムエンティティ認識) を選択します。これにより、選択したリソースの既存のファイル一覧が表示されます。

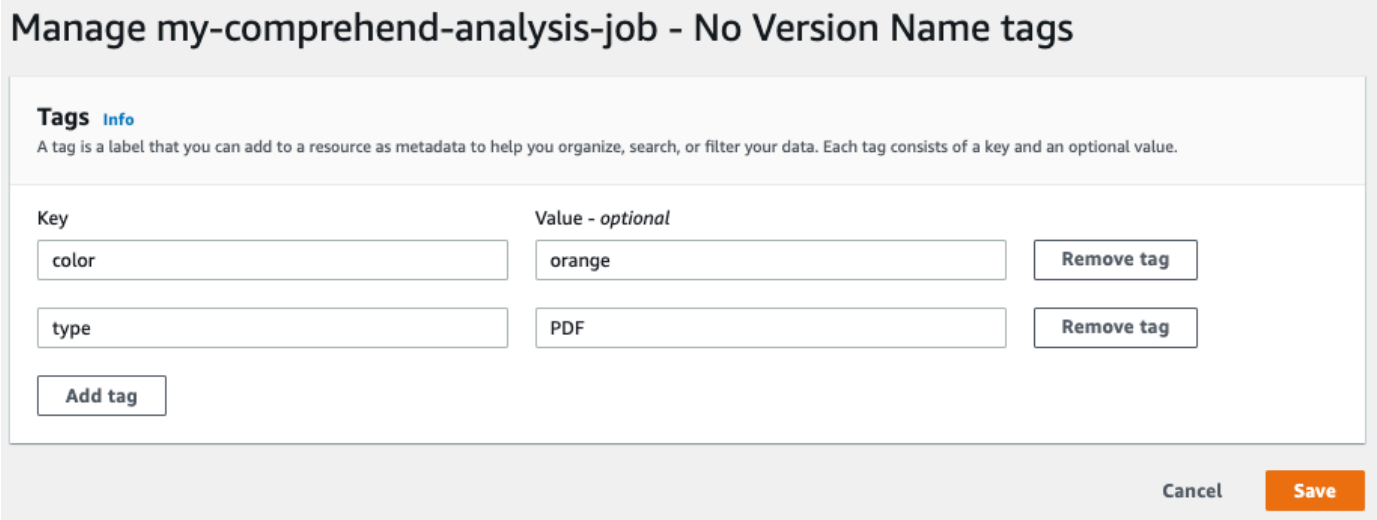


3. タグを表示、変更、または削除するファイル (またはモデル) の名前をクリックします。そのバックアップの詳細ページが表示されます。[タグ] ボックスが表示されるまで下にスクロールします。ここには、選択したファイル (またはモデル) に関連付けられているすべてのタグが表示されます。



[タグ管理] を選択して、リソースからタグを編集または削除します。

4. 変更するテキストをクリックし、タグを編集します。[タグを削除] を選択してタグを削除することもできます。新しいタグを追加するには、[タグ追加]を選択し、空白のフィールドに目的のテキストを入力します。



タグの変更が完了したら、[保存] を選択します。

SDK を使用した Amazon Comprehend のコード例 AWS SDKs

次のコード例は、 AWS Software Development Kit (SDK) で Amazon Comprehend を使用する方法を示しています。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、特定のタスクを実行する方法を示すコード例です。

クロスサービスの例は、複数の AWS のサービスで動作するサンプルアプリケーションです。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

コードの例

- SDK を使用した Amazon Comprehend のアクション AWS SDKs
  - AWS SDK または CLI CreateDocumentClassifierで を使用する
  - AWS SDK または CLI DeleteDocumentClassifierで を使用する
  - AWS SDK または CLI DescribeDocumentClassificationJobで を使用する
  - AWS SDK または CLI DescribeDocumentClassifierで を使用する
  - AWS SDK または CLI DescribeTopicsDetectionJobで を使用する
  - AWS SDK または CLI DetectDominantLanguageで を使用する
  - AWS SDK または CLI DetectEntitiesで を使用する
  - AWS SDK または CLI DetectKeyPhrasesで を使用する
  - AWS SDK または CLI DetectPiiEntitiesで を使用する
  - AWS SDK または CLI DetectSentimentで を使用する
  - AWS SDK または CLI DetectSyntaxで を使用する
  - AWS SDK または CLI ListDocumentClassificationJobsで を使用する
  - AWS SDK または CLI ListDocumentClassifiersで を使用する

- AWS SDK または CLI ListTopicsDetectionJobsで を使用する
- AWS SDK または CLI StartDocumentClassificationJobで を使用する
- AWS SDK または CLI StartTopicsDetectionJobで を使用する

- SDK を使用した Amazon Comprehend のシナリオ AWS SDKs
  - Amazon Comprehend と SDK を使用してドキュメント要素を検出する AWS
  - AWS SDK を使用してサンプルデータに対して Amazon Comprehend トピックモデリングジョブを実行する
  - AWS SDK を使用してカスタム Amazon Comprehend 分類子をトレーニングし、ドキュメントを分類する
- SDK を使用した Amazon Comprehend のクロスサービスの例 AWS SDKs
  - Amazon Transcribe ストリーミングアプリケーションを構築する
  - Amazon Lex チャットボットを作成して、ウェブサイトの訪問者を引き付けましょう
  - Amazon SQS を使用してメッセージを送受信するウェブアプリケーションを作成する
  - 顧客からのフィードバックを分析し、音声を合成するアプリケーションの作成
  - AWS SDK を使用してイメージから抽出されたテキスト内のエンティティを検出する

SDK を使用した Amazon Comprehend のアクション AWS SDKs

次のコード例は、 AWS SDKs を使用して個々の Amazon Comprehend アクションを実行する方法を示しています。これらの抜粋は、Amazon Comprehend API を呼び出し、コンテキスト内で実行する必要がある大規模なプログラムからのコード抜粋です。各例には GitHub、コードの設定と実行の手順を示す へのリンクが含まれています。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細な一覧については、『Amazon Comprehend API Reference』(Amazon S3 API リファレンス) を参照してください。

例

- AWS SDK または CLI CreateDocumentClassifierで を使用する
- AWS SDK または CLI DeleteDocumentClassifierで を使用する
- AWS SDK または CLI DescribeDocumentClassificationJobで を使用する
- AWS SDK または CLI DescribeDocumentClassifierで を使用する
- AWS SDK または CLI DescribeTopicsDetectionJobで を使用する
- AWS SDK または CLI DetectDominantLanguageで を使用する

- AWS SDK または CLI DetectEntitiesで を使用する
- AWS SDK または CLI DetectKeyPhrasesで を使用する
- AWS SDK または CLI DetectPiiEntitiesで を使用する
- AWS SDK または CLI DetectSentimentで を使用する
- AWS SDK または CLI DetectSyntaxで を使用する
- AWS SDK または CLI ListDocumentClassificationJobsで を使用する
- AWS SDK または CLI ListDocumentClassifiersで を使用する
- AWS SDK または CLI ListTopicsDetectionJobsで を使用する
- AWS SDK または CLI StartDocumentClassificationJobで を使用する
- AWS SDK または CLI StartTopicsDetectionJobで を使用する

AWS SDK または CLI **CreateDocumentClassifier**で を使用する

以下のコード例は、CreateDocumentClassifier の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- カスタム分類子をトレーニングしてドキュメントを分類します。

CLI

AWS CLI

ドキュメントを分類するドキュメント分類子を作成するには

次の create-document-classifier の例では、ドキュメント分類子モデルのトレーニングプロセスを開始します。トレーニングデータファイル「training.csv」は、--input-data-config タグにあります。training.csv は 2 列のドキュメントで、1 番目の列にはラベルまたは分類が、2 番目の列にはドキュメントが表示されます。

```
aws comprehend create-document-classifier \
    --document-classifier-name example-classifier \
    --data-access-arn arn:aws:comprehend:us-west-2:111122223333:pii-entities-
detection-job/123456abcdeb0e11022f22a11EXAMPLE \
    --input-data-config "S3Uri=s3://DOC-EXAMPLE-BUCKET/" \
    --language-code en
```

出力:

```
{
    "DocumentClassifierArn": "arn:aws:comprehend:us-west-2:111122223333:document-
classifier/example-classifier"
}
```

詳細については、「Amazon Comprehend 開発者ガイド」の「カスタム分類」を参照してください。

- API の詳細については、「 コマンドリファレンスCreateDocumentClassifier」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.comprehend.ComprehendClient;
import software.amazon.awssdk.services.comprehend.model.ComprehendException;
import
 software.amazon.awssdk.services.comprehend.model.CreateDocumentClassifierRequest;
import
 software.amazon.awssdk.services.comprehend.model.CreateDocumentClassifierResponse;
import
 software.amazon.awssdk.services.comprehend.model.DocumentClassifierInputDataConfig;

/**
 * Before running this code example, you can setup the necessary resources, such
 * as the CSV file and IAM Roles, by following this document:
 * https://aws.amazon.com/blogs/machine-learning/building-a-custom-classifier-
using-amazon-comprehend/
 *
 * Also, set up your development environment, including your credentials.
 *
```

```
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DocumentClassifierDemo {
    public static void main(String[] args) {
        final String usage = """

                Usage:    <dataAccessRoleArn> <s3Uri> <documentClassifierName>

                Where:
                  dataAccessRoleArn - The ARN value of the role used for this
 operation.
                  s3Uri - The Amazon S3 bucket that contains the CSV file.
                  documentClassifierName - The name of the document classifier.
                """;

        if (args.length != 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String dataAccessRoleArn = args[0];
        String s3Uri = args[1];
        String documentClassifierName = args[2];

        Region region = Region.US_EAST_1;
        ComprehendClient comClient = ComprehendClient.builder()
                .region(region)
                .build();

        createDocumentClassifier(comClient, dataAccessRoleArn, s3Uri,
 documentClassifierName);
        comClient.close();
    }

    public static void createDocumentClassifier(ComprehendClient comClient,
 String dataAccessRoleArn, String s3Uri,
            String documentClassifierName) {
        try {
            DocumentClassifierInputDataConfig config =
 DocumentClassifierInputDataConfig.builder()
                    .s3Uri(s3Uri)
```

```
                    .build();

            CreateDocumentClassifierRequest createDocumentClassifierRequest =
 CreateDocumentClassifierRequest.builder()
                    .documentClassifierName(documentClassifierName)
                    .dataAccessRoleArn(dataAccessRoleArn)
                    .languageCode("en")
                    .inputDataConfig(config)
                    .build();

            CreateDocumentClassifierResponse createDocumentClassifierResult =
 comClient
                    .createDocumentClassifier(createDocumentClassifierRequest);
            String documentClassifierArn =
 createDocumentClassifierResult.documentClassifierArn();
            System.out.println("Document Classifier ARN: " +
 documentClassifierArn);

        } catch (ComprehendException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「 API リファレンスCreateDocumentClassifier」の「」を参照してください。 AWS SDK for Java 2.x

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""
```

```
    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None


    def create(
        self,
        name,
        language_code,
        training_bucket,
        training_key,
        data_access_role_arn,
        mode,
    ):
        """
        Creates a custom classifier. After the classifier is created, it
immediately
        starts training on the data found in the specified Amazon S3 bucket.
Training
        can take 30 minutes or longer. The `describe_document_classifier`
function
        can be used to get training status and returns a status of TRAINED when
the
        classifier is ready to use.

        :param name: The name of the classifier.
        :param language_code: The language the classifier can operate on.
        :param training_bucket: The Amazon S3 bucket that contains the training
data.
        :param training_key: The prefix used to find training data in the
training
                             bucket. If multiple objects have the same prefix,
all
                             of them are used.
        :param data_access_role_arn: The Amazon Resource Name (ARN) of a role
that
                                     grants Comprehend permission to read from
the
                                     training bucket.
        :return: The ARN of the newly created classifier.
```

```
        """
        try:
            response = self.comprehend_client.create_document_classifier(
                DocumentClassifierName=name,
                LanguageCode=language_code,
                InputDataConfig={"S3Uri": f"s3://{training_bucket}/
{training_key}"},
                DataAccessRoleArn=data_access_role_arn,
                Mode=mode.value,
            )
            self.classifier_arn = response["DocumentClassifierArn"]
            logger.info("Started classifier creation. Arn is: %s.",
 self.classifier_arn)
        except ClientError:
            logger.exception("Couldn't create classifier %s.", name)
            raise
        else:
            return self.classifier_arn
```

- API の詳細については、CreateDocumentClassifierAWS 「 SDK for Python (Boto3) API リファレンス」の「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DeleteDocumentClassifier**で を使用する

以下のコード例は、DeleteDocumentClassifier の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- カスタム分類子をトレーニングしてドキュメントを分類します。

CLI

AWS CLI

カスタムドキュメント分類子を削除するには

次の delete-document-classifier の例では、カスタムドキュメント分類子モデルを削除します。

```
aws comprehend delete-document-classifier \
    --document-classifier-arn arn:aws:comprehend:us-west-2:111122223333:document-classifier/example-classifier-1
```

このコマンドでは何も出力されません。

詳細については、「Amazon Comprehend デベロッパーガイド」の「Amazon Comprehend のエンドポイントの管理」を参照してください。

- API の詳細については、「 コマンドリファレンスDeleteDocumentClassifier」の「」を参照してください。 AWS CLI

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None
```

```
    def delete(self):
        """
        Deletes the classifier.
        """
        try:
            self.comprehend_client.delete_document_classifier(
                DocumentClassifierArn=self.classifier_arn
            )
            logger.info("Deleted classifier %s.", self.classifier_arn)
            self.classifier_arn = None
        except ClientError:
            logger.exception("Couldn't deleted classifier %s.",
self.classifier_arn)
            raise
```

- API の詳細については、 DeleteDocumentClassifier AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DescribeDocumentClassificationJob**で を使用する

以下のコード例は、DescribeDocumentClassificationJob の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- カスタム分類子をトレーニングしてドキュメントを分類します。

CLI

AWS CLI

ドキュメント分類ジョブを記述するには

次の describe-document-classification-job の例では、非同期ドキュメント分類ジョブのプロパティを取得します。

```
aws comprehend describe-document-classification-job \
    --job-id 123456abcdeb0e11022f22a11EXAMPLE
```

出力:

```
{
    "DocumentClassificationJobProperties": {
        "JobId": "123456abcdeb0e11022f22a11EXAMPLE",
        "JobArn": "arn:aws:comprehend:us-west-2:111122223333:document-
classification-job/123456abcdeb0e11022f22a11EXAMPLE",
        "JobName": "exampleclassificationjob",
        "JobStatus": "COMPLETED",
        "SubmitTime": "2023-06-14T17:09:51.788000+00:00",
        "EndTime": "2023-06-14T17:15:58.582000+00:00",
        "DocumentClassifierArn": "arn:aws:comprehend:us-
west-2:111122223333:document-classifier/mymodel/version/1",
        "InputDataConfig": {
            "S3Uri": "s3://DOC-EXAMPLE-BUCKET/jobdata/",
            "InputFormat": "ONE_DOC_PER_LINE"
        },
        "OutputDataConfig": {
            "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/
testfolder/111122223333-CLN-123456abcdeb0e11022f22a11EXAMPLE/output/
output.tar.gz"
        },
        "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-servicerole"
    }
}
```

詳細については、「Amazon Comprehend 開発者ガイド」の「カスタム分類」を参照してください。

- API の詳細については、「 コマンドリファレンスDescribeDocumentClassificationJob」の「」を参照してください。 AWS CLI

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None


    def describe_job(self, job_id):
        """
        Gets metadata about a classification job.

        :param job_id: The ID of the job to look up.
        :return: Metadata about the job.
        """
        try:
            response =
 self.comprehend_client.describe_document_classification_job(
                JobId=job_id
            )
            job = response["DocumentClassificationJobProperties"]
            logger.info("Got classification job %s.", job["JobName"])
        except ClientError:
            logger.exception("Couldn't get classification job %s.", job_id)
            raise
        else:
            return job
```

- API の詳細については、 DescribeDocumentClassificationJob AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI `DescribeDocumentClassifier`で を使用する

以下のコード例は、DescribeDocumentClassifier の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- カスタム分類子をトレーニングしてドキュメントを分類します。

CLI

AWS CLI

ドキュメント分類子を記述するには

次の describe-document-classifier の例では、カスタムドキュメント分類子モデルのプロパティを取得します。

```
aws comprehend describe-document-classifier \
    --document-classifier-arn arn:aws:comprehend:us-west-2:111122223333:document-
classifier/example-classifier-1
```

出力:

```
{
    "DocumentClassifierProperties": {
        "DocumentClassifierArn": "arn:aws:comprehend:us-
west-2:111122223333:document-classifier/example-classifier-1",
        "LanguageCode": "en",
        "Status": "TRAINED",
        "SubmitTime": "2023-06-13T19:04:15.735000+00:00",
        "EndTime": "2023-06-13T19:42:31.752000+00:00",
        "TrainingStartTime": "2023-06-13T19:08:20.114000+00:00",
```

```
        "TrainingEndTime": "2023-06-13T19:41:35.080000+00:00",
        "InputDataConfig": {
            "DataFormat": "COMPREHEND_CSV",
            "S3Uri": "s3://DOC-EXAMPLE-BUCKET/trainingdata"
        },
        "OutputDataConfig": {},
        "ClassifierMetadata": {
            "NumberOfLabels": 3,
            "NumberOfTrainedDocuments": 5016,
            "NumberOfTestDocuments": 557,
            "EvaluationMetrics": {
                "Accuracy": 0.9856,
                "Precision": 0.9919,
                "Recall": 0.9459,
                "F1Score": 0.9673,
                "MicroPrecision": 0.9856,
                "MicroRecall": 0.9856,
                "MicroF1Score": 0.9856,
                "HammingLoss": 0.0144
            }
        },
        "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-example-role",
        "Mode": "MULTI_CLASS"
    }
}
```

詳細については、「Amazon Comprehend デベロッパーガイド」の「カスタムモデルの作成と管理」を参照してください。

- API の詳細については、「 コマンドリファレンスDescribeDocumentClassifier」の「」を参照してください。 AWS CLI

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None


    def describe(self, classifier_arn=None):
        """
        Gets metadata about a custom classifier, including its current status.

        :param classifier_arn: The ARN of the classifier to look up.
        :return: Metadata about the classifier.
        """
        if classifier_arn is not None:
            self.classifier_arn = classifier_arn
        try:
            response = self.comprehend_client.describe_document_classifier(
                DocumentClassifierArn=self.classifier_arn
            )
            classifier = response["DocumentClassifierProperties"]
            logger.info("Got classifier %s.", self.classifier_arn)
        except ClientError:
            logger.exception("Couldn't get classifier %s.", self.classifier_arn)
            raise
        else:
            return classifier
```

- API の詳細については、 DescribeDocumentClassifier AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DescribeTopicsDetectionJob**で を使用する

以下のコード例は、DescribeTopicsDetectionJob の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- サンプルデータに対してトピックモデリングジョブを実行する。

CLI

AWS CLI

トピック検出ジョブを説明するには

次の describe-topics-detection-job の例では、非同期トピック検出ジョブのプロパティを取得します。

```
aws comprehend describe-topics-detection-job \
    --job-id 123456abcdeb0e11022f22a11EXAMPLE
```

出力:

```
{
    "TopicsDetectionJobProperties": {
        "JobId": "123456abcdeb0e11022f22a11EXAMPLE",
        "JobArn": "arn:aws:comprehend:us-west-2:111122223333:topics-detection-
job/123456abcdeb0e11022f22a11EXAMPLE",
        "JobName": "example_topics_detection",
        "JobStatus": "IN_PROGRESS",
        "SubmitTime": "2023-06-09T18:44:43.414000+00:00",
        "InputDataConfig": {
            "S3Uri": "s3://DOC-EXAMPLE-BUCKET",
            "InputFormat": "ONE_DOC_PER_LINE"
        },
        "OutputDataConfig": {
            "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/
testfolder/111122223333-TOPICS-123456abcdeb0e11022f22a11EXAMPLE/output/
output.tar.gz"
        },
        "NumberOfTopics": 10,
```

```
        "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-examplerole"
    }
}
```

詳細については、「Amazon Comprehend デベロッパーガイド」の「Amazon Comprehend のインサイトのための非同期分析」を参照してください。

- API の詳細については、「 コマンドリファレンスDescribeTopicsDetectionJob」の「」を参照してください。 AWS CLI

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendTopicModeler:
    """Encapsulates a Comprehend topic modeler."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def describe_job(self, job_id):
        """
        Gets metadata about a topic modeling job.

        :param job_id: The ID of the job to look up.
        :return: Metadata about the job.
        """
        try:
            response = self.comprehend_client.describe_topics_detection_job(
                JobId=job_id
```

```
            )
            job = response["TopicsDetectionJobProperties"]
            logger.info("Got topic detection job %s.", job_id)
        except ClientError:
            logger.exception("Couldn't get topic detection job %s.", job_id)
            raise
        else:
            return job
```

- API の詳細については、 DescribeTopicsDetectionJob AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI `DetectDominantLanguage`で を使用する

以下のコード例は、DetectDominantLanguage の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- ドキュメントの要素を検出する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Threading.Tasks;
```

```
    using Amazon.Comprehend;
    using Amazon.Comprehend.Model;

    /// <summary>
    /// This example calls the Amazon Comprehend service to determine the
    /// dominant language.
    /// </summary>
    public static class DetectDominantLanguage
    {
        /// <summary>
        /// Calls Amazon Comprehend to determine the dominant language used in
        /// the sample text.
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle.";

            var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            Console.WriteLine("Calling DetectDominantLanguage\n");
            var detectDominantLanguageRequest = new
DetectDominantLanguageRequest()
            {
                Text = text,
            };

            var detectDominantLanguageResponse = await
comprehendClient.DetectDominantLanguageAsync(detectDominantLanguageRequest);
            foreach (var dl in detectDominantLanguageResponse.Languages)
            {
                Console.WriteLine($"Language Code: {dl.LanguageCode}, Score:
{dl.Score}");
            }

            Console.WriteLine("Done");
        }
    }
```

- API の詳細については、「 API リファレンスDetectDominantLanguage」の「」を参照してください。 AWS SDK for .NET

CLI

AWS CLI

入力テキストの主要言語を検出するには

以下の `detect-dominant-language` は、入力テキストを分析し、主要言語を特定します。事前トレーニング済みモデルの信頼スコアも出力されます。

```
aws comprehend detect-dominant-language \
    --text "It is a beautiful day in Seattle."
```

出力:

```
{
    "Languages": [
        {
            "LanguageCode": "en",
            "Score": 0.9877256155014038
        }
    ]
}
```

詳細については、「Amazon Comprehend デベロッパーガイド」の「主要言語」を参照してください。

- API の詳細については、「 コマンドリファレンスDetectDominantLanguage」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.comprehend.ComprehendClient;
```

```
import software.amazon.awssdk.services.comprehend.model.ComprehendException;
import
 software.amazon.awssdk.services.comprehend.model.DetectDominantLanguageRequest;
import
 software.amazon.awssdk.services.comprehend.model.DetectDominantLanguageResponse;
import software.amazon.awssdk.services.comprehend.model.DominantLanguage;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectLanguage {
    public static void main(String[] args) {
        // Specify French text - "It is raining today in Seattle".
        String text = "Il pleut aujourd'hui à Seattle";
        Region region = Region.US_EAST_1;

        ComprehendClient comClient = ComprehendClient.builder()
                .region(region)
                .build();

        System.out.println("Calling DetectDominantLanguage");
        detectTheDominantLanguage(comClient, text);
        comClient.close();
    }

    public static void detectTheDominantLanguage(ComprehendClient comClient,
 String text) {
        try {
            DetectDominantLanguageRequest request =
 DetectDominantLanguageRequest.builder()
                    .text(text)
                    .build();

            DetectDominantLanguageResponse resp =
 comClient.detectDominantLanguage(request);
            List<DominantLanguage> allLanList = resp.languages();
            for (DominantLanguage lang : allLanList) {
```

```
                System.out.println("Language is " + lang.languageCode());
            }

        } catch (ComprehendException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「 API リファレンスDetectDominantLanguage」の「」を参照してください。 AWS SDK for Java 2.x

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendDetect:
    """Encapsulates Comprehend detection functions."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def detect_languages(self, text):
        """
        Detects languages used in a document.

        :param text: The document to inspect.
        :return: The list of languages along with their confidence scores.
        """
```

```
        try:
            response = self.comprehend_client.detect_dominant_language(Text=text)
            languages = response["Languages"]
            logger.info("Detected %s languages.", len(languages))
        except ClientError:
            logger.exception("Couldn't detect languages.")
            raise
        else:
            return languages
```

- API の詳細については、 DetectDominantLanguage AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DetectEntities**で を使用する

以下のコード例は、DetectEntities の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- ドキュメントの要素を検出する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
```

```
    using System.Threading.Tasks;
    using Amazon.Comprehend;
    using Amazon.Comprehend.Model;

    /// <summary>
    /// This example shows how to use the AmazonComprehend service detect any
    /// entities in submitted text.
    /// </summary>
    public static class DetectEntities
    {
        /// <summary>
        /// The main method calls the DetectEntitiesAsync method to find any
        /// entities in the sample code.
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle";

            var comprehendClient = new AmazonComprehendClient();

            Console.WriteLine("Calling DetectEntities\n");
            var detectEntitiesRequest = new DetectEntitiesRequest()
            {
                Text = text,
                LanguageCode = "en",
            };
            var detectEntitiesResponse = await
comprehendClient.DetectEntitiesAsync(detectEntitiesRequest);

            foreach (var e in detectEntitiesResponse.Entities)
            {
                Console.WriteLine($"Text: {e.Text}, Type: {e.Type}, Score:
{e.Score}, BeginOffset: {e.BeginOffset}, EndOffset: {e.EndOffset}");
            }

            Console.WriteLine("Done");
        }
    }
```

- API の詳細については、「 API リファレンスDetectEntities」の「」を参照してください。AWS SDK for .NET

CLI

AWS CLI

入力テキスト内の名前付きエンティティを検出するには

次の detect-entities の例では、入力テキストを分析し、名前付きエンティティを返します。予測ごとに、事前トレーニング済みモデルの信頼スコアも出力されます。

```
aws comprehend detect-entities \
    --language-code en \
    --text "Hello Zhang Wei, I am John. Your AnyCompany Financial Services, LLC
 credit card \
    account 1111-XXXX-1111-XXXX has a minimum payment of $24.53 that is due by
 July 31st. Based on your autopay settings, \
    we will withdraw your payment on the due date from your bank account number
 XXXXXX1111 with the routing number XXXXX0000. \
    Customer feedback for Sunshine Spa, 123 Main St, Anywhere. Send comments to
 Alice at AnySpa@example.com."
```

出力:

```
{
    "Entities": [
        {
            "Score": 0.9994556307792664,
            "Type": "PERSON",
            "Text": "Zhang Wei",
            "BeginOffset": 6,
            "EndOffset": 15
        },
        {
            "Score": 0.9981022477149963,
            "Type": "PERSON",
            "Text": "John",
            "BeginOffset": 22,
            "EndOffset": 26
        },
        {
            "Score": 0.9986887574195862,
            "Type": "ORGANIZATION",
            "Text": "AnyCompany Financial Services, LLC",
            "BeginOffset": 33,
```

```
            "EndOffset": 67
        },
        {
            "Score": 0.9959119558334351,
            "Type": "OTHER",
            "Text": "1111-XXXX-1111-XXXX",
            "BeginOffset": 88,
            "EndOffset": 107
        },
        {
            "Score": 0.9708039164543152,
            "Type": "QUANTITY",
            "Text": ".53",
            "BeginOffset": 133,
            "EndOffset": 136
        },
        {
            "Score": 0.9987268447875977,
            "Type": "DATE",
            "Text": "July 31st",
            "BeginOffset": 152,
            "EndOffset": 161
        },
        {
            "Score": 0.9858865737915039,
            "Type": "OTHER",
            "Text": "XXXXXX1111",
            "BeginOffset": 271,
            "EndOffset": 281
        },
        {
            "Score": 0.9700471758842468,
            "Type": "OTHER",
            "Text": "XXXXX0000",
            "BeginOffset": 306,
            "EndOffset": 315
        },
        {
            "Score": 0.9591118693351746,
            "Type": "ORGANIZATION",
            "Text": "Sunshine Spa",
            "BeginOffset": 340,
            "EndOffset": 352
        },
```

```
        {
            "Score": 0.9797496795654297,
            "Type": "LOCATION",
            "Text": "123 Main St",
            "BeginOffset": 354,
            "EndOffset": 365
        },
        {
            "Score": 0.994929313659668,
            "Type": "PERSON",
            "Text": "Alice",
            "BeginOffset": 394,
            "EndOffset": 399
        },
        {
            "Score": 0.9949769377708435,
            "Type": "OTHER",
            "Text": "AnySpa@example.com",
            "BeginOffset": 403,
            "EndOffset": 418
        }
    ]
}
```

詳細については、「Amazon Comprehend デベロッパーガイド」の「[Entities] (エンティティ)」を参照してください。

- API の詳細については、「 コマンドリファレンスDetectEntities」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.comprehend.ComprehendClient;
```

```
import software.amazon.awssdk.services.comprehend.model.DetectEntitiesRequest;
import software.amazon.awssdk.services.comprehend.model.DetectEntitiesResponse;
import software.amazon.awssdk.services.comprehend.model.Entity;
import software.amazon.awssdk.services.comprehend.model.ComprehendException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectEntities {
    public static void main(String[] args) {
        String text = "Amazon.com, Inc. is located in Seattle, WA and was founded
 July 5th, 1994 by Jeff Bezos, allowing customers to buy everything from books to
 blenders. Seattle is north of Portland and south of Vancouver, BC. Other notable
 Seattle - based companies are Starbucks and Boeing.";
        Region region = Region.US_EAST_1;
        ComprehendClient comClient = ComprehendClient.builder()
                .region(region)
                .build();

        System.out.println("Calling DetectEntities");
        detectAllEntities(comClient, text);
        comClient.close();
    }

    public static void detectAllEntities(ComprehendClient comClient, String text)
 {
        try {
            DetectEntitiesRequest detectEntitiesRequest =
 DetectEntitiesRequest.builder()
                    .text(text)
                    .languageCode("en")
                    .build();

            DetectEntitiesResponse detectEntitiesResult =
 comClient.detectEntities(detectEntitiesRequest);
            List<Entity> entList = detectEntitiesResult.entities();
            for (Entity entity : entList) {
```

```
                System.out.println("Entity text is " + entity.text());
            }

        } catch (ComprehendException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「 API リファレンスDetectEntities」の「」を参照してください。 AWS SDK for Java 2.x

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendDetect:
    """Encapsulates Comprehend detection functions."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def detect_entities(self, text, language_code):
        """
        Detects entities in a document. Entities can be things like people and
 places
        or other common terms.

        :param text: The document to inspect.
```

```
        :param language_code: The language of the document.
        :return: The list of entities along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_entities(
                Text=text, LanguageCode=language_code
            )
            entities = response["Entities"]
            logger.info("Detected %s entities.", len(entities))
        except ClientError:
            logger.exception("Couldn't detect entities.")
            raise
        else:
            return entities
```

- API の詳細については、 DetectEntities AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DetectKeyPhrases**で を使用する

以下のコード例は、DetectKeyPhrases の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- ドキュメントの要素を検出する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Comprehend;
    using Amazon.Comprehend.Model;

    /// <summary>
    /// This example shows how to use the Amazon Comprehend service to
    /// search text for key phrases.
    /// </summary>
    public static class DetectKeyPhrase
    {
        /// <summary>
        /// This method calls the Amazon Comprehend method DetectKeyPhrasesAsync
        /// to detect any key phrases in the sample text.
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle";

            var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectKeyPhrases API
            Console.WriteLine("Calling DetectKeyPhrases");
            var detectKeyPhrasesRequest = new DetectKeyPhrasesRequest()
            {
                Text = text,
                LanguageCode = "en",
            };
            var detectKeyPhrasesResponse = await
comprehendClient.DetectKeyPhrasesAsync(detectKeyPhrasesRequest);
            foreach (var kp in detectKeyPhrasesResponse.KeyPhrases)
```

```
            {
                Console.WriteLine($"Text: {kp.Text}, Score: {kp.Score},
 BeginOffset: {kp.BeginOffset}, EndOffset: {kp.EndOffset}");
            }

            Console.WriteLine("Done");
        }
    }
```

- API の詳細については、「 API リファレンスDetectKeyPhrases」の「」を参照してください。 AWS SDK for .NET

CLI

AWS CLI

入力テキスト内のキーフレーズを検出するには

次の detect-key-phrases の例では、入力テキストを分析し、主要な名詞フレーズを特定します。予測ごとに、事前トレーニング済みモデルの信頼スコアも出力されます。

```
aws comprehend detect-key-phrases \
    --language-code en \
    --text "Hello Zhang Wei, I am John. Your AnyCompany Financial Services, LLC
 credit card \
        account 1111-XXXX-1111-XXXX has a minimum payment of $24.53 that is due
 by July 31st. Based on your autopay settings, \
        we will withdraw your payment on the due date from your bank account
 number XXXXXX1111 with the routing number XXXXX0000. \
        Customer feedback for Sunshine Spa, 123 Main St, Anywhere. Send comments
 to Alice at AnySpa@example.com."
```

出力:

```
{
    "KeyPhrases": [
        {
            "Score": 0.8996376395225525,
            "Text": "Zhang Wei",
            "BeginOffset": 6,
```

```
            "EndOffset": 15
        },
        {
            "Score": 0.9992469549179077,
            "Text": "John",
            "BeginOffset": 22,
            "EndOffset": 26
        },
        {
            "Score": 0.988385021686554,
            "Text": "Your AnyCompany Financial Services",
            "BeginOffset": 28,
            "EndOffset": 62
        },
        {
            "Score": 0.8740853071212769,
            "Text": "LLC credit card account 1111-XXXX-1111-XXXX",
            "BeginOffset": 64,
            "EndOffset": 107
        },
        {
            "Score": 0.9999437928199768,
            "Text": "a minimum payment",
            "BeginOffset": 112,
            "EndOffset": 129
        },
        {
            "Score": 0.9998900890350342,
            "Text": ".53",
            "BeginOffset": 133,
            "EndOffset": 136
        },
        {
            "Score": 0.9979453086853027,
            "Text": "July 31st",
            "BeginOffset": 152,
            "EndOffset": 161
        },
        {
            "Score": 0.9983011484146118,
            "Text": "your autopay settings",
            "BeginOffset": 172,
            "EndOffset": 193
        },
```

```
        {
            "Score": 0.9996572136878967,
            "Text": "your payment",
            "BeginOffset": 211,
            "EndOffset": 223
        },
        {
            "Score": 0.9995037317276001,
            "Text": "the due date",
            "BeginOffset": 227,
            "EndOffset": 239
        },
        {
            "Score": 0.9702621698379517,
            "Text": "your bank account number XXXXXX1111",
            "BeginOffset": 245,
            "EndOffset": 280
        },
        {
            "Score": 0.9179925918579102,
            "Text": "the routing number XXXXX0000.Customer feedback",
            "BeginOffset": 286,
            "EndOffset": 332
        },
        {
            "Score": 0.9978160858154297,
            "Text": "Sunshine Spa",
            "BeginOffset": 337,
            "EndOffset": 349
        },
        {
            "Score": 0.9706913232803345,
            "Text": "123 Main St",
            "BeginOffset": 351,
            "EndOffset": 362
        },
        {
            "Score": 0.9941995143890381,
            "Text": "comments",
            "BeginOffset": 379,
            "EndOffset": 387
        },
        {
            "Score": 0.9759287238121033,
```

```
            "Text": "Alice",
            "BeginOffset": 391,
            "EndOffset": 396
        },
        {
            "Score": 0.8376792669296265,
            "Text": "AnySpa@example.com",
            "BeginOffset": 400,
            "EndOffset": 415
        }
    ]
}
```

詳細については、「Amazon Comprehend 開発者ガイド」の「キーフレーズ」を参照してください。

- API の詳細については、「 コマンドリファレンスDetectKeyPhrases」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.comprehend.ComprehendClient;
import software.amazon.awssdk.services.comprehend.model.DetectKeyPhrasesRequest;
import software.amazon.awssdk.services.comprehend.model.DetectKeyPhrasesResponse;
import software.amazon.awssdk.services.comprehend.model.KeyPhrase;
import software.amazon.awssdk.services.comprehend.model.ComprehendException;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
```

```
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectKeyPhrases {
    public static void main(String[] args) {
        String text = "Amazon.com, Inc. is located in Seattle, WA and was founded
 July 5th, 1994 by Jeff Bezos, allowing customers to buy everything from books to
 blenders. Seattle is north of Portland and south of Vancouver, BC. Other notable
 Seattle - based companies are Starbucks and Boeing.";
        Region region = Region.US_EAST_1;
        ComprehendClient comClient = ComprehendClient.builder()
                .region(region)
                .build();

        System.out.println("Calling DetectKeyPhrases");
        detectAllKeyPhrases(comClient, text);
        comClient.close();
    }

    public static void detectAllKeyPhrases(ComprehendClient comClient, String
 text) {
        try {
            DetectKeyPhrasesRequest detectKeyPhrasesRequest =
 DetectKeyPhrasesRequest.builder()
                    .text(text)
                    .languageCode("en")
                    .build();

            DetectKeyPhrasesResponse detectKeyPhrasesResult =
 comClient.detectKeyPhrases(detectKeyPhrasesRequest);
            List<KeyPhrase> phraseList = detectKeyPhrasesResult.keyPhrases();
            for (KeyPhrase keyPhrase : phraseList) {
                System.out.println("Key phrase text is " + keyPhrase.text());
            }

        } catch (ComprehendException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「 API リファレンスDetectKeyPhrases」の「」を参照してください。 AWS SDK for Java 2.x

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendDetect:
    """Encapsulates Comprehend detection functions."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def detect_key_phrases(self, text, language_code):
        """
        Detects key phrases in a document. A key phrase is typically a noun and
 its
        modifiers.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The list of key phrases along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_key_phrases(
                Text=text, LanguageCode=language_code
            )
            phrases = response["KeyPhrases"]
            logger.info("Detected %s phrases.", len(phrases))
        except ClientError:
            logger.exception("Couldn't detect phrases.")
```

```
            raise
        else:
            return phrases
```

- API の詳細については、 DetectKeyPhrases AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DetectPiiEntities**で を使用する

以下のコード例は、DetectPiiEntities の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- ドキュメントの要素を検出する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Threading.Tasks;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

/// <summary>
/// This example shows how to use the Amazon Comprehend service to find
```

```
    /// personally identifiable information (PII) within text submitted to the
    /// DetectPiiEntitiesAsync method.
    /// </summary>
    public class DetectingPII
    {
        /// <summary>
        /// This method calls the DetectPiiEntitiesAsync method to locate any
        /// personally dientifiable information within the supplied text.
        /// </summary>
        public static async Task Main()
        {
            var comprehendClient = new AmazonComprehendClient();
            var text = @"Hello Paul Santos. The latest statement for your
                         credit card account 1111-0000-1111-0000 was
                         mailed to 123 Any Street, Seattle, WA 98109.";

            var request = new DetectPiiEntitiesRequest
            {
                Text = text,
                LanguageCode = "EN",
            };

            var response = await
comprehendClient.DetectPiiEntitiesAsync(request);

            if (response.Entities.Count > 0)
            {
                foreach (var entity in response.Entities)
                {
                    var entityValue = text.Substring(entity.BeginOffset,
entity.EndOffset - entity.BeginOffset);
                    Console.WriteLine($"{entity.Type}: {entityValue}");
                }
            }
        }
    }
```

- API の詳細については、「 API リファレンスDetectPiiEntities」の「」を参照してください。 AWS SDK for .NET

CLI

AWS CLI

入力テキストの PII エンティティを検出するには

次の detect-pii-entities の例では、入力テキストを分析し、個人を特定できる情報 (PII) を含むエンティティを特定します。予測ごとに、事前トレーニング済みモデルの信頼スコアも出力されます。

```
aws comprehend detect-pii-entities \
    --language-code en \
    --text "Hello Zhang Wei, I am John. Your AnyCompany Financial Services, LLC
 credit card \
        account 1111-XXXX-1111-XXXX has a minimum payment of $24.53 that is due
 by July 31st. Based on your autopay settings, \
        we will withdraw your payment on the due date from your bank account
 number XXXXXX1111 with the routing number XXXXX0000. \
        Customer feedback for Sunshine Spa, 123 Main St, Anywhere. Send comments
 to Alice at AnySpa@example.com."
```

出力:

```
{
    "Entities": [
        {
            "Score": 0.9998322129249573,
            "Type": "NAME",
            "BeginOffset": 6,
            "EndOffset": 15
        },
        {
            "Score": 0.9998878240585327,
            "Type": "NAME",
            "BeginOffset": 22,
            "EndOffset": 26
        },
        {
            "Score": 0.9994089603424072,
            "Type": "CREDIT_DEBIT_NUMBER",
            "BeginOffset": 88,
            "EndOffset": 107
        },
```

```
        {
            "Score": 0.9999760985374451,
            "Type": "DATE_TIME",
            "BeginOffset": 152,
            "EndOffset": 161
        },
        {
            "Score": 0.9999449253082275,
            "Type": "BANK_ACCOUNT_NUMBER",
            "BeginOffset": 271,
            "EndOffset": 281
        },
        {
            "Score": 0.9999847412109375,
            "Type": "BANK_ROUTING",
            "BeginOffset": 306,
            "EndOffset": 315
        },
        {
            "Score": 0.999925434589386,
            "Type": "ADDRESS",
            "BeginOffset": 354,
            "EndOffset": 365
        },
        {
            "Score": 0.9989161491394043,
            "Type": "NAME",
            "BeginOffset": 394,
            "EndOffset": 399
        },
        {
            "Score": 0.9994171857833862,
            "Type": "EMAIL",
            "BeginOffset": 403,
            "EndOffset": 418
        }
    ]
}
```

サポートされる PII エンティティタイプの一覧の詳細については、「Amazon Comprehend 開発者ガイド」の「個人を特定できる情報 (PII)」を参照してください。

- API の詳細については、「 コマンドリファレンスDetectPiiEntities」の「」を参照してください。 AWS CLI

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendDetect:
    """Encapsulates Comprehend detection functions."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def detect_pii(self, text, language_code):
        """
        Detects personally identifiable information (PII) in a document. PII can
 be
        things like names, account numbers, or addresses.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The list of PII entities along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_pii_entities(
                Text=text, LanguageCode=language_code
            )
            entities = response["Entities"]
            logger.info("Detected %s PII entities.", len(entities))
        except ClientError:
            logger.exception("Couldn't detect PII entities.")
            raise
        else:
            return entities
```

- API の詳細については、 DetectPiiEntities AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **DetectSentiment**で を使用する

以下のコード例は、DetectSentiment の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- ドキュメントの要素を検出する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Threading.Tasks;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

/// <summary>
/// This example shows how to detect the overall sentiment of the supplied
/// text using the Amazon Comprehend service.
/// </summary>
public static class DetectSentiment
{
```

```
        /// <summary>
        /// This method calls the DetetectSentimentAsync method to analyze the
        /// supplied text and determine the overal sentiment.
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle";

            var comprehendClient = new
AmazonComprehendClient(Amazon.RegionEndpoint.USWest2);

            // Call DetectKeyPhrases API
            Console.WriteLine("Calling DetectSentiment");
            var detectSentimentRequest = new DetectSentimentRequest()
            {
                Text = text,
                LanguageCode = "en",
            };
            var detectSentimentResponse = await
comprehendClient.DetectSentimentAsync(detectSentimentRequest);
            Console.WriteLine($"Sentiment: {detectSentimentResponse.Sentiment}");
            Console.WriteLine("Done");
        }
    }
```

- API の詳細については、「 API リファレンスDetectSentiment」の「」を参照してください。 AWS SDK for .NET

CLI

AWS CLI

入力テキストの感情を検出するには

次の detect-sentiment の例では、入力テキストを分析し、一般的な感情 (POSITIVE、NEUTRAL、MIXED、または NEGATIVE) の推論を返します。

```
aws comprehend detect-sentiment \
    --language-code en \
    --text "It is a beautiful day in Seattle"
```

出力:

```
{
    "Sentiment": "POSITIVE",
    "SentimentScore": {
        "Positive": 0.9976957440376282,
        "Negative": 9.653854067437351e-05,
        "Neutral": 0.002169104292988777,
        "Mixed": 3.857641786453314e-05
    }
}
```

詳細については、「Amazon Comprehend デベロッパーガイド」の「感情」を参照してください。

- API の詳細については、「 コマンドリファレンスDetectSentiment」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.comprehend.ComprehendClient;
import software.amazon.awssdk.services.comprehend.model.ComprehendException;
import software.amazon.awssdk.services.comprehend.model.DetectSentimentRequest;
import software.amazon.awssdk.services.comprehend.model.DetectSentimentResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectSentiment {
    public static void main(String[] args) {
        String text = "Amazon.com, Inc. is located in Seattle, WA and was founded
 July 5th, 1994 by Jeff Bezos, allowing customers to buy everything from books to
 blenders. Seattle is north of Portland and south of Vancouver, BC. Other notable
 Seattle - based companies are Starbucks and Boeing.";
        Region region = Region.US_EAST_1;
        ComprehendClient comClient = ComprehendClient.builder()
                .region(region)
                .build();

        System.out.println("Calling DetectSentiment");
        detectSentiments(comClient, text);
        comClient.close();
    }

    public static void detectSentiments(ComprehendClient comClient, String text)
 {
        try {
            DetectSentimentRequest detectSentimentRequest =
 DetectSentimentRequest.builder()
                    .text(text)
                    .languageCode("en")
                    .build();

            DetectSentimentResponse detectSentimentResult =
 comClient.detectSentiment(detectSentimentRequest);
            System.out.println("The Neutral value is " +
 detectSentimentResult.sentimentScore().neutral());

        } catch (ComprehendException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「 API リファレンスDetectSentiment」の「」を参照してください。 AWS SDK for Java 2.x

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendDetect:
    """Encapsulates Comprehend detection functions."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def detect_sentiment(self, text, language_code):
        """
        Detects the overall sentiment expressed in a document. Sentiment can
        be positive, negative, neutral, or a mixture.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The sentiments along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_sentiment(
                Text=text, LanguageCode=language_code
            )
            logger.info("Detected primary sentiment %s.", response["Sentiment"])
        except ClientError:
            logger.exception("Couldn't detect sentiment.")
            raise
        else:
            return response
```

- API の詳細については、DetectSentimentAWS 「 SDK for Python (Boto3) API リファレンス」の「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI `DetectSyntax`で を使用する

以下のコード例は、`DetectSyntax` の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- ドキュメントの要素を検出する

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using System;
using System.Threading.Tasks;
using Amazon.Comprehend;
using Amazon.Comprehend.Model;

/// <summary>
/// This example shows how to use Amazon Comprehend to detect syntax
/// elements by calling the DetectSyntaxAsync method.
/// </summary>
public class DetectingSyntax
{
    /// <summary>
    /// This method calls DetectSynaxAsync to identify the syntax elements
    /// in the sample text.
```

```
        /// </summary>
        public static async Task Main()
        {
            string text = "It is raining today in Seattle";

            var comprehendClient = new AmazonComprehendClient();

            // Call DetectSyntax API
            Console.WriteLine("Calling DetectSyntaxAsync\n");
            var detectSyntaxRequest = new DetectSyntaxRequest()
            {
                Text = text,
                LanguageCode = "en",
            };
            DetectSyntaxResponse detectSyntaxResponse = await
 comprehendClient.DetectSyntaxAsync(detectSyntaxRequest);
            foreach (SyntaxToken s in detectSyntaxResponse.SyntaxTokens)
            {
                Console.WriteLine($"Text: {s.Text}, PartOfSpeech:
 {s.PartOfSpeech.Tag}, BeginOffset: {s.BeginOffset}, EndOffset: {s.EndOffset}");
            }

            Console.WriteLine("Done");
        }
    }
```

- API の詳細については、「 API リファレンスDetectSyntax」の「」を参照してください。 AWS SDK for .NET

CLI

AWS CLI

入力テキスト内の品詞を検出するには

次の detect-syntax の例では、入力テキストの構文を分析し、さまざまな品詞を返します。予測ごとに、事前トレーニング済みモデルの信頼スコアも出力されます。

```
aws comprehend detect-syntax \
    --language-code en \
```

```
    --text "It is a beautiful day in Seattle."
```

出力:

```
{
    "SyntaxTokens": [
        {
            "TokenId": 1,
            "Text": "It",
            "BeginOffset": 0,
            "EndOffset": 2,
            "PartOfSpeech": {
                "Tag": "PRON",
                "Score": 0.9999740719795227
            }
        },
        {
            "TokenId": 2,
            "Text": "is",
            "BeginOffset": 3,
            "EndOffset": 5,
            "PartOfSpeech": {
                "Tag": "VERB",
                "Score": 0.999901294708252
            }
        },
        {
            "TokenId": 3,
            "Text": "a",
            "BeginOffset": 6,
            "EndOffset": 7,
            "PartOfSpeech": {
                "Tag": "DET",
                "Score": 0.9999938607215881
            }
        },
        {
            "TokenId": 4,
            "Text": "beautiful",
            "BeginOffset": 8,
            "EndOffset": 17,
            "PartOfSpeech": {
                "Tag": "ADJ",
```

```
                "Score": 0.9987351894378662
            }
        },
        {
            "TokenId": 5,
            "Text": "day",
            "BeginOffset": 18,
            "EndOffset": 21,
            "PartOfSpeech": {
                "Tag": "NOUN",
                "Score": 0.9999796748161316
            }
        },
        {
            "TokenId": 6,
            "Text": "in",
            "BeginOffset": 22,
            "EndOffset": 24,
            "PartOfSpeech": {
                "Tag": "ADP",
                "Score": 0.9998047947883606
            }
        },
        {
            "TokenId": 7,
            "Text": "Seattle",
            "BeginOffset": 25,
            "EndOffset": 32,
            "PartOfSpeech": {
                "Tag": "PROPN",
                "Score": 0.9940530061721802
            }
        }
    ]
}
```

詳細については、「Amazon Comprehend デベロッパーガイド」の「構文分析」を参照してください。

- API の詳細については、「 コマンドリファレンスDetectSyntax」の「」を参照してください。 AWS CLI

Java

SDK for Java 2.x

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.comprehend.ComprehendClient;
import software.amazon.awssdk.services.comprehend.model.ComprehendException;
import software.amazon.awssdk.services.comprehend.model.DetectSyntaxRequest;
import software.amazon.awssdk.services.comprehend.model.DetectSyntaxResponse;
import software.amazon.awssdk.services.comprehend.model.SyntaxToken;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DetectSyntax {
    public static void main(String[] args) {
        String text = "Amazon.com, Inc. is located in Seattle, WA and was founded
 July 5th, 1994 by Jeff Bezos, allowing customers to buy everything from books to
 blenders. Seattle is north of Portland and south of Vancouver, BC. Other notable
 Seattle - based companies are Starbucks and Boeing.";
       Region region = Region.US_EAST_1;
       ComprehendClient comClient = ComprehendClient.builder()
               .region(region)
               .build();

       System.out.println("Calling DetectSyntax");
       detectAllSyntax(comClient, text);
       comClient.close();
    }
```

```
    public static void detectAllSyntax(ComprehendClient comClient, String text) {
        try {
            DetectSyntaxRequest detectSyntaxRequest =
 DetectSyntaxRequest.builder()
                    .text(text)
                    .languageCode("en")
                    .build();

            DetectSyntaxResponse detectSyntaxResult =
 comClient.detectSyntax(detectSyntaxRequest);
            List<SyntaxToken> syntaxTokens = detectSyntaxResult.syntaxTokens();
            for (SyntaxToken token : syntaxTokens) {
                System.out.println("Language is " + token.text());
                System.out.println("Part of speech is " +
 token.partOfSpeech().tagAsString());
            }

        } catch (ComprehendException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「 API リファレンスDetectSyntax」の「」を参照してください。 AWS SDK for Java 2.x

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendDetect:
    """Encapsulates Comprehend detection functions."""
```

```
    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def detect_syntax(self, text, language_code):
        """
        Detects syntactical elements of a document. Syntax tokens are portions of
        text along with their use as parts of speech, such as nouns, verbs, and
        interjections.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The list of syntax tokens along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_syntax(
                Text=text, LanguageCode=language_code
            )
            tokens = response["SyntaxTokens"]
            logger.info("Detected %s syntax tokens.", len(tokens))
        except ClientError:
            logger.exception("Couldn't detect syntax.")
            raise
        else:
            return tokens
```

- API の詳細については、 DetectSyntax AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **ListDocumentClassificationJobs**で を使用する

以下のコード例は、ListDocumentClassificationJobs の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- カスタム分類子をトレーニングしてドキュメントを分類します。

CLI

AWS CLI

すべてのドキュメント分類ジョブを一覧表示するには

次の list-document-classification-jobs の例では、すべてのドキュメント分類ジョブを一覧表示しています。

```
aws comprehend list-document-classification-jobs
```

出力:

```
{
    "DocumentClassificationJobPropertiesList": [
        {
            "JobId": "123456abcdeb0e11022f22a11EXAMPLE",
            "JobArn": "arn:aws:comprehend:us-west-2:1234567890101:document-
classification-job/123456abcdeb0e11022f22a11EXAMPLE",
            "JobName": "exampleclassificationjob",
            "JobStatus": "COMPLETED",
            "SubmitTime": "2023-06-14T17:09:51.788000+00:00",
            "EndTime": "2023-06-14T17:15:58.582000+00:00",
            "DocumentClassifierArn": "arn:aws:comprehend:us-
west-2:1234567890101:document-classifier/mymodel/version/12",
            "InputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-BUCKET/jobdata/",
                "InputFormat": "ONE_DOC_PER_LINE"
            },
            "OutputDataConfig": {
```

```
                "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/
thefolder/1234567890101-CLN-e758dd56b824aa717ceab551f11749fb/output/
output.tar.gz"
            },
            "DataAccessRoleArn": "arn:aws:iam::1234567890101:role/service-role/
AmazonComprehendServiceRole-example-role"
        },
        {
            "JobId": "123456abcdeb0e11022f22a1EXAMPLE2",
            "JobArn": "arn:aws:comprehend:us-west-2:1234567890101:document-
classification-job/123456abcdeb0e11022f22a1EXAMPLE2",
            "JobName": "exampleclassificationjob2",
            "JobStatus": "COMPLETED",
            "SubmitTime": "2023-06-14T17:22:39.829000+00:00",
            "EndTime": "2023-06-14T17:28:46.107000+00:00",
            "DocumentClassifierArn": "arn:aws:comprehend:us-
west-2:1234567890101:document-classifier/mymodel/version/12",
            "InputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-BUCKET/jobdata/",
                "InputFormat": "ONE_DOC_PER_LINE"
            },
            "OutputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/
thefolder/1234567890101-CLN-123456abcdeb0e11022f22a1EXAMPLE2/output/
output.tar.gz"
            },
            "DataAccessRoleArn": "arn:aws:iam::1234567890101:role/service-role/
AmazonComprehendServiceRole-example-role"
        }
    ]
}
```

詳細については、「Amazon Comprehend 開発者ガイド」の「カスタム分類」を参照してください。

- API の詳細については、「 コマンドリファレンスListDocumentClassificationJobs」の「」を参照してください。 AWS CLI

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None


    def list_jobs(self):
        """
        Lists the classification jobs for the current account.

        :return: The list of jobs.
        """
        try:
            response = self.comprehend_client.list_document_classification_jobs()
            jobs = response["DocumentClassificationJobPropertiesList"]
            logger.info("Got %s document classification jobs.", len(jobs))
        except ClientError:
            logger.exception(
                "Couldn't get document classification jobs.",
            )
            raise
        else:
            return jobs
```

- API の詳細については、 ListDocumentClassificationJobs AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **ListDocumentClassifiers**で を使用する

以下のコード例は、ListDocumentClassifiers の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- カスタム分類子をトレーニングしてドキュメントを分類します。

CLI

AWS CLI

すべてのドキュメント分類子を一覧表示するには

次の list-document-classifiers の例は、トレーニング済みおよびトレーニング中のすべてのドキュメント分類子モデルを一覧表示します。

```
aws comprehend list-document-classifiers
```

出力:

```
{
    "DocumentClassifierPropertiesList": [
        {
            "DocumentClassifierArn": "arn:aws:comprehend:us-
west-2:111122223333:document-classifier/exampleclassifier1",
            "LanguageCode": "en",
            "Status": "TRAINED",
            "SubmitTime": "2023-06-13T19:04:15.735000+00:00",
            "EndTime": "2023-06-13T19:42:31.752000+00:00",
            "TrainingStartTime": "2023-06-13T19:08:20.114000+00:00",
            "TrainingEndTime": "2023-06-13T19:41:35.080000+00:00",
            "InputDataConfig": {
```

```
                "DataFormat": "COMPREHEND_CSV",
                "S3Uri": "s3://DOC-EXAMPLE-BUCKET/trainingdata"
            },
            "OutputDataConfig": {},
            "ClassifierMetadata": {
                "NumberOfLabels": 3,
                "NumberOfTrainedDocuments": 5016,
                "NumberOfTestDocuments": 557,
                "EvaluationMetrics": {
                    "Accuracy": 0.9856,
                    "Precision": 0.9919,
                    "Recall": 0.9459,
                    "F1Score": 0.9673,
                    "MicroPrecision": 0.9856,
                    "MicroRecall": 0.9856,
                    "MicroF1Score": 0.9856,
                    "HammingLoss": 0.0144
                }
            },
            "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-testorle",
            "Mode": "MULTI_CLASS"
        },
        {
            "DocumentClassifierArn": "arn:aws:comprehend:us-
west-2:111122223333:document-classifier/exampleclassifier2",
            "LanguageCode": "en",
            "Status": "TRAINING",
            "SubmitTime": "2023-06-13T21:20:28.690000+00:00",
            "InputDataConfig": {
                "DataFormat": "COMPREHEND_CSV",
                "S3Uri": "s3://DOC-EXAMPLE-BUCKET/trainingdata"
            },
            "OutputDataConfig": {},
            "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-testorle",
            "Mode": "MULTI_CLASS"
        }
    ]
}
```


詳細については、「Amazon Comprehend デベロッパーガイド」の「カスタムモデルの作成と管理」を参照してください。

- API の詳細については、「 コマンドリファレンスListDocumentClassifiers」の「」を参照してください。 AWS CLI

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None


    def list(self):
        """
        Lists custom classifiers for the current account.

        :return: The list of classifiers.
        """
        try:
            response = self.comprehend_client.list_document_classifiers()
            classifiers = response["DocumentClassifierPropertiesList"]
            logger.info("Got %s classifiers.", len(classifiers))
        except ClientError:
            logger.exception(
                "Couldn't get classifiers.",
            )
            raise
        else:
            return classifiers
```

- API の詳細については、 ListDocumentClassifiers AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **ListTopicsDetectionJobs**で を使用する

以下のコード例は、ListTopicsDetectionJobs の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- サンプルデータに対してトピックモデリングジョブを実行する。

CLI

AWS CLI

トピック検出ジョブをすべて一覧表示するには

次の list-topics-detection-jobs の例では、進行中および完了した非同期トピック検出ジョブをすべて一覧表示します。

```
aws comprehend list-topics-detection-jobs
```

出力:

```
{
    "TopicsDetectionJobPropertiesList": [
        {
            "JobId": "123456abcdeb0e11022f22a11EXAMPLE",
            "JobArn": "arn:aws:comprehend:us-west-2:111122223333:topics-
detection-job/123456abcdeb0e11022f22a11EXAMPLE",
            "JobName" "topic-analysis-1"
            "JobStatus": "IN_PROGRESS",
            "SubmitTime": "2023-06-09T18:40:35.384000+00:00",
```

```
            "EndTime": "2023-06-09T18:46:41.936000+00:00",
            "InputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-BUCKET",
                "InputFormat": "ONE_DOC_PER_LINE"
            },
            "OutputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/
thefolder/111122223333-TOPICS-123456abcdeb0e11022f22a11EXAMPLE/output/
output.tar.gz"
            },
            "NumberOfTopics": 10,
            "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-example-role"
        },
        {
            "JobId": "123456abcdeb0e11022f22a1EXAMPLE2",
            "JobArn": "arn:aws:comprehend:us-west-2:111122223333:topics-
detection-job/123456abcdeb0e11022f22a1EXAMPLE2",
            "JobName": "topic-analysis-2",
            "JobStatus": "COMPLETED",
            "SubmitTime": "2023-06-09T18:44:43.414000+00:00",
            "EndTime": "2023-06-09T18:50:50.872000+00:00",
            "InputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-BUCKET",
                "InputFormat": "ONE_DOC_PER_LINE"
            },
            "OutputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/
thefolder/111122223333-TOPICS-123456abcdeb0e11022f22a1EXAMPLE2/output/
output.tar.gz"
            },
            "NumberOfTopics": 10,
            "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-example-role"
        },
        {
            "JobId": "123456abcdeb0e11022f22a1EXAMPLE3",
            "JobArn": "arn:aws:comprehend:us-west-2:111122223333:topics-
detection-job/123456abcdeb0e11022f22a1EXAMPLE3",
            "JobName": "topic-analysis-2",
            "JobStatus": "IN_PROGRESS",
            "SubmitTime": "2023-06-09T18:50:56.737000+00:00",
            "InputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-BUCKET",
```

```
                "InputFormat": "ONE_DOC_PER_LINE"
            },
            "OutputDataConfig": {
                "S3Uri": "s3://DOC-EXAMPLE-DESTINATION-BUCKET/
thefolder/111122223333-TOPICS-123456abcdeb0e11022f22a1EXAMPLE3/output/
output.tar.gz"
            },
            "NumberOfTopics": 10,
            "DataAccessRoleArn": "arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-example-role"
        }
    ]
}
```

詳細については、「Amazon Comprehend デベロッパーガイド」の「Amazon Comprehend のインサイトのための非同期分析」を参照してください。

- API の詳細については、「 コマンドリファレンスListTopicsDetectionJobs」の「」を参照してください。 AWS CLI

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendTopicModeler:
    """Encapsulates a Comprehend topic modeler."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def list_jobs(self):
```

```
        """
        Lists topic modeling jobs for the current account.

        :return: The list of jobs.
        """
        try:
            response = self.comprehend_client.list_topics_detection_jobs()
            jobs = response["TopicsDetectionJobPropertiesList"]
            logger.info("Got %s topic detection jobs.", len(jobs))
        except ClientError:
            logger.exception("Couldn't get topic detection jobs.")
            raise
        else:
            return jobs
```

- API の詳細については、 ListTopicsDetectionJobs AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **StartDocumentClassificationJob**で を使用する

以下のコード例は、StartDocumentClassificationJob の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- カスタム分類子をトレーニングしてドキュメントを分類します。

CLI

AWS CLI

ドキュメント分類ジョブを開始するには

次の start-document-classification-job の例では、--input-data-config タグで指定されたアドレスにあるすべてのファイルに対して、カスタムモデルを使用してドキュメント分類ジョブを開始します。この例では、入力 S3 バケットには、SampleSMStext1.txt、SampleSMStext2.txt、SampleSMStext3.txt が含まれています。このモデルは以前、迷惑メールと迷惑メールでない正規のメール、または SMS メッセージにドキュメントを分類するトレーニングを受けていました。ジョブが完了すると、output.tar.gz は --output-data-config タグで指定された場所に配置されます。output.tar.gz には各ドキュメントの分類を一覧表示する predictions.jsonl が含まれています。Json の出力は、1 ファイルに 1 行で出力されますが、ここでは読みやすい形式で表示されています。

```
aws comprehend start-document-classification-job \
    --job-name exampleclassificationjob \
    --input-data-config "S3Uri=s3://DOC-EXAMPLE-BUCKET-INPUT/jobdata/" \
    --output-data-config "S3Uri=s3://DOC-EXAMPLE-DESTINATION-BUCKET/testfolder/"
 \
    --data-access-role-arn arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-example-role \
    --document-classifier-arn arn:aws:comprehend:us-west-2:111122223333:document-
classifier/mymodel/version/12
```

SampleSMStext1.txt の内容:

```
"CONGRATULATIONS! TXT 2155550100 to win $5000"
```

SampleSMStext2.txt の内容:

```
"Hi, when do you want me to pick you up from practice?"
```

SampleSMStext3.txt の内容:

```
"Plz send bank account # to 2155550100 to claim prize!!"
```

出力:

```
{
    "JobId": "e758dd56b824aa717ceab551fEXAMPLE",
    "JobArn": "arn:aws:comprehend:us-west-2:111122223333:document-classification-
job/e758dd56b824aa717ceab551fEXAMPLE",
    "JobStatus": "SUBMITTED"
```

```
}
```

predictions.jsonl の内容:

```
{"File": "SampleSMSText1.txt", "Line": "0", "Classes": [{"Name": "spam", "Score":
 0.9999}, {"Name": "ham", "Score": 0.0001}]}
{"File": "SampleSMStext2.txt", "Line": "0", "Classes": [{"Name": "ham", "Score":
 0.9994}, {"Name": "spam", "Score": 0.0006}]}
{"File": "SampleSMSText3.txt", "Line": "0", "Classes": [{"Name": "spam", "Score":
 0.9999}, {"Name": "ham", "Score": 0.0001}]}
```

詳細については、「Amazon Comprehend 開発者ガイド」の「カスタム分類」を参照してください。

- API の詳細については、「 コマンドリファレンスStartDocumentClassificationJob」の「」を参照してください。 AWS CLI

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None


    def start_job(
        self,
        job_name,
```

```
        input_bucket,
        input_key,
        input_format,
        output_bucket,
        output_key,
        data_access_role_arn,
    ):
        """
        Starts a classification job. The classifier must be trained or the job
        will fail. Input is read from the specified Amazon S3 input bucket and
        written to the specified output bucket. Output data is stored in a tar
        archive compressed in gzip format. The job runs asynchronously, so you
can
        call `describe_document_classification_job` to get job status until it
        returns a status of SUCCEEDED.

        :param job_name: The name of the job.
        :param input_bucket: The Amazon S3 bucket that contains input data.
        :param input_key: The prefix used to find input data in the input
                          bucket. If multiple objects have the same prefix, all
                          of them are used.
        :param input_format: The format of the input data, either one document
per
                             file or one document per line.
        :param output_bucket: The Amazon S3 bucket where output data is written.
        :param output_key: The prefix prepended to the output data.
        :param data_access_role_arn: The Amazon Resource Name (ARN) of a role
that
                                     grants Comprehend permission to read from
the
                                     input bucket and write to the output bucket.
        :return: Information about the job, including the job ID.
        """
        try:
            response = self.comprehend_client.start_document_classification_job(
                DocumentClassifierArn=self.classifier_arn,
                JobName=job_name,
                InputDataConfig={
                    "S3Uri": f"s3://{input_bucket}/{input_key}",
                    "InputFormat": input_format.value,
                },
                OutputDataConfig={"S3Uri": f"s3://{output_bucket}/{output_key}"},
                DataAccessRoleArn=data_access_role_arn,
            )
```

```
            logger.info(
                "Document classification job %s is %s.", job_name,
response["JobStatus"]
            )
        except ClientError:
            logger.exception("Couldn't start classification job %s.", job_name)
            raise
        else:
            return response
```

- API の詳細については、 StartDocumentClassificationJob AWS SDK for Python (Boto3) API リファレンスの「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI **StartTopicsDetectionJob**で を使用する

以下のコード例は、StartTopicsDetectionJob の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- サンプルデータに対してトピックモデリングジョブを実行する。

.NET

AWS SDK for .NET

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    using System;
    using System.Threading.Tasks;
```

```
    using Amazon.Comprehend;
    using Amazon.Comprehend.Model;

    /// <summary>
    /// This example scans the documents in an Amazon Simple Storage Service
    /// (Amazon S3) bucket and analyzes it for topics. The results are stored
    /// in another bucket and then the resulting job properties are displayed
    /// on the screen. This example was created using the AWS SDK for .NEt
    /// version 3.7 and .NET Core version 5.0.
    /// </summary>
    public static class TopicModeling
    {
        /// <summary>
        /// This methos calls a topic detection job by calling the Amazon
        /// Comprehend StartTopicsDetectionJobRequest.
        /// </summary>
        public static async Task Main()
        {
            var comprehendClient = new AmazonComprehendClient();

            string inputS3Uri = "s3://input bucket/input path";
            InputFormat inputDocFormat = InputFormat.ONE_DOC_PER_FILE;
            string outputS3Uri = "s3://output bucket/output path";
            string dataAccessRoleArn = "arn:aws:iam::account ID:role/data access
role";
            int numberOfTopics = 10;

            var startTopicsDetectionJobRequest = new
StartTopicsDetectionJobRequest()
            {
                InputDataConfig = new InputDataConfig()
                {
                    S3Uri = inputS3Uri,
                    InputFormat = inputDocFormat,
                },
                OutputDataConfig = new OutputDataConfig()
                {
                    S3Uri = outputS3Uri,
                },
                DataAccessRoleArn = dataAccessRoleArn,
                NumberOfTopics = numberOfTopics,
            };
```

```
            var startTopicsDetectionJobResponse = await
comprehendClient.StartTopicsDetectionJobAsync(startTopicsDetectionJobRequest);

            var jobId = startTopicsDetectionJobResponse.JobId;
            Console.WriteLine("JobId: " + jobId);

            var describeTopicsDetectionJobRequest = new
DescribeTopicsDetectionJobRequest()
            {
                JobId = jobId,
            };

            var describeTopicsDetectionJobResponse = await
comprehendClient.DescribeTopicsDetectionJobAsync(describeTopicsDetectionJobRequest);

PrintJobProperties(describeTopicsDetectionJobResponse.TopicsDetectionJobProperties);

            var listTopicsDetectionJobsResponse = await
comprehendClient.ListTopicsDetectionJobsAsync(new
ListTopicsDetectionJobsRequest());
            foreach (var props in
listTopicsDetectionJobsResponse.TopicsDetectionJobPropertiesList)
            {
                PrintJobProperties(props);
            }
        }

        /// <summary>
        /// This method is a helper method that displays the job properties
        /// from the call to StartTopicsDetectionJobRequest.
        /// </summary>
        /// <param name="props">A list of properties from the call to
        /// StartTopicsDetectionJobRequest.</param>
        private static void PrintJobProperties(TopicsDetectionJobProperties
props)
        {
            Console.WriteLine($"JobId: {props.JobId}, JobName: {props.JobName},
JobStatus: {props.JobStatus}");
            Console.WriteLine($"NumberOfTopics:
{props.NumberOfTopics}\nInputS3Uri: {props.InputDataConfig.S3Uri}");
            Console.WriteLine($"InputFormat: {props.InputDataConfig.InputFormat},
OutputS3Uri: {props.OutputDataConfig.S3Uri}");
        }
    }
```

- API の詳細については、「 API リファレンスStartTopicsDetectionJob」の「」を参照してください。 AWS SDK for .NET

CLI

AWS CLI

トピック検出分析ジョブを開始するには

次の start-topics-detection-job の例では、--input-data-config タグで指定されたアドレスにあるすべてのファイルの非同期トピック検出ジョブを開始します。ジョブが完了すると、フォルダ、output は --ouput-data-config タグで指定された場所に配置されます。output には topic-terms.csv と doc-topics.csv が含まれています。最初の出力ファイル topic-terms.csv は、コレクション内のトピックのリストです。デフォルトでは、リストには、各トピックの上位の言葉が重みに応じてトピック別に含まれています。2 つ目のファイル doc-topics.csv には、トピックに関連するドキュメントと、そのトピックに関係するドキュメントの割合が一覧表示されます。

```
aws comprehend start-topics-detection-job \
    --job-name example_topics_detection_job \
    --language-code en \
    --input-data-config "S3Uri=s3://DOC-EXAMPLE-BUCKET/" \
    --output-data-config "S3Uri=s3://DOC-EXAMPLE-DESTINATION-BUCKET/testfolder/"
 \
    --data-access-role-arn arn:aws:iam::111122223333:role/service-role/
AmazonComprehendServiceRole-example-role \
    --language-code en
```

出力:

```
{
    "JobId": "123456abcdeb0e11022f22a11EXAMPLE",
    "JobArn": "arn:aws:comprehend:us-west-2:111122223333:key-phrases-detection-
job/123456abcdeb0e11022f22a11EXAMPLE",
    "JobStatus": "SUBMITTED"
}
```

詳細については、「Amazon Comprehend デベロッパーガイド」の「トピックのモデリング」を参照してください。

- API の詳細については、「 コマンドリファレンスStartTopicsDetectionJob」の「」を参照してください。 AWS CLI

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
class ComprehendTopicModeler:
    """Encapsulates a Comprehend topic modeler."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def start_job(
        self,
        job_name,
        input_bucket,
        input_key,
        input_format,
        output_bucket,
        output_key,
        data_access_role_arn,
    ):
        """
        Starts a topic modeling job. Input is read from the specified Amazon S3
        input bucket and written to the specified output bucket. Output data is
 stored
        in a tar archive compressed in gzip format. The job runs asynchronously,
 so you
```

```
        can call `describe_topics_detection_job` to get job status until it
        returns a status of SUCCEEDED.

        :param job_name: The name of the job.
        :param input_bucket: An Amazon S3 bucket that contains job input.
        :param input_key: The prefix used to find input data in the input
                          bucket. If multiple objects have the same prefix,
all
                          of them are used.
        :param input_format: The format of the input data, either one document
per
                          file or one document per line.
        :param output_bucket: The Amazon S3 bucket where output data is written.
        :param output_key: The prefix prepended to the output data.
        :param data_access_role_arn: The Amazon Resource Name (ARN) of a role
that
                                     grants Comprehend permission to read from
the
                                     input bucket and write to the output bucket.
        :return: Information about the job, including the job ID.
        """
        try:
            response = self.comprehend_client.start_topics_detection_job(
                JobName=job_name,
                DataAccessRoleArn=data_access_role_arn,
                InputDataConfig={
                    "S3Uri": f"s3://{input_bucket}/{input_key}",
                    "InputFormat": input_format.value,
                },
                OutputDataConfig={"S3Uri": f"s3://{output_bucket}/{output_key}"},
            )
            logger.info("Started topic modeling job %s.", response["JobId"])
        except ClientError:
            logger.exception("Couldn't start topic modeling job.")
            raise
        else:
            return response
```

- API の詳細については、StartTopicsDetectionJobAWS 「 SDK for Python (Boto3) API リファレンス」の「」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用した Amazon Comprehend のシナリオ AWS SDKs

次のコード例は、 AWS SDKs を使用して Amazon Comprehend で一般的なシナリオを実装する方法を示しています。これらのシナリオは、Amazon Comprehend 内で複数の機能を呼び出すことによって特定のタスクを実行する方法を示しています。各シナリオには GitHub、コードの設定と実行の手順を示す へのリンクが含まれています。

例

- Amazon Comprehend と SDK を使用してドキュメント要素を検出する AWS
- AWS SDK を使用してサンプルデータに対して Amazon Comprehend トピックモデリングジョブを実行する
- AWS SDK を使用してカスタム Amazon Comprehend 分類子をトレーニングし、ドキュメントを分類する

Amazon Comprehend と SDK を使用してドキュメント要素を検出する AWS

次のコードサンプルは、以下の操作方法を示しています。

- ドキュメント内の言語、エンティティ、キーフレーズを検出する。
- ドキュメントから個人を特定できる情報 (PII)を検出する。
- ドキュメントのセンチメントを検出する。
- ドキュメントの構文要素を検出します。

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon Comprehend のアクションをラップするクラスを作成します。

```
import logging
from pprint import pprint
import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

class ComprehendDetect:
    """Encapsulates Comprehend detection functions."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def detect_languages(self, text):
        """
        Detects languages used in a document.

        :param text: The document to inspect.
        :return: The list of languages along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_dominant_language(Text=text)
            languages = response["Languages"]
            logger.info("Detected %s languages.", len(languages))
        except ClientError:
            logger.exception("Couldn't detect languages.")
            raise
```

```
        else:
            return languages


    def detect_entities(self, text, language_code):
        """
        Detects entities in a document. Entities can be things like people and
places
        or other common terms.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The list of entities along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_entities(
                Text=text, LanguageCode=language_code
            )
            entities = response["Entities"]
            logger.info("Detected %s entities.", len(entities))
        except ClientError:
            logger.exception("Couldn't detect entities.")
            raise
        else:
            return entities


    def detect_key_phrases(self, text, language_code):
        """
        Detects key phrases in a document. A key phrase is typically a noun and
its
        modifiers.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The list of key phrases along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_key_phrases(
                Text=text, LanguageCode=language_code
            )
            phrases = response["KeyPhrases"]
            logger.info("Detected %s phrases.", len(phrases))
        except ClientError:
```

```
            logger.exception("Couldn't detect phrases.")
            raise
        else:
            return phrases


    def detect_pii(self, text, language_code):
        """
        Detects personally identifiable information (PII) in a document. PII can
be
        things like names, account numbers, or addresses.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The list of PII entities along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_pii_entities(
                Text=text, LanguageCode=language_code
            )
            entities = response["Entities"]
            logger.info("Detected %s PII entities.", len(entities))
        except ClientError:
            logger.exception("Couldn't detect PII entities.")
            raise
        else:
            return entities


    def detect_sentiment(self, text, language_code):
        """
        Detects the overall sentiment expressed in a document. Sentiment can
        be positive, negative, neutral, or a mixture.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The sentiments along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_sentiment(
                Text=text, LanguageCode=language_code
            )
            logger.info("Detected primary sentiment %s.", response["Sentiment"])
        except ClientError:
```

```
            logger.exception("Couldn't detect sentiment.")
            raise
        else:
            return response


    def detect_syntax(self, text, language_code):
        """
        Detects syntactical elements of a document. Syntax tokens are portions of
        text along with their use as parts of speech, such as nouns, verbs, and
        interjections.

        :param text: The document to inspect.
        :param language_code: The language of the document.
        :return: The list of syntax tokens along with their confidence scores.
        """
        try:
            response = self.comprehend_client.detect_syntax(
                Text=text, LanguageCode=language_code
            )
            tokens = response["SyntaxTokens"]
            logger.info("Detected %s syntax tokens.", len(tokens))
        except ClientError:
            logger.exception("Couldn't detect syntax.")
            raise
        else:
            return tokens
```

Wrapper クラスの関数を呼び出して、ドキュメント内のエンティティ、フレーズなどを検出します。

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon Comprehend detection demo!")
    print("-" * 88)

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    comp_detect = ComprehendDetect(boto3.client("comprehend"))
    with open("detect_sample.txt") as sample_file:
```

```
        sample_text = sample_file.read()

demo_size = 3

print("Sample text used for this demo:")
print("-" * 88)
print(sample_text)
print("-" * 88)

print("Detecting languages.")
languages = comp_detect.detect_languages(sample_text)
pprint(languages)
lang_code = languages[0]["LanguageCode"]

print("Detecting entities.")
entities = comp_detect.detect_entities(sample_text, lang_code)
print(f"The first {demo_size} are:")
pprint(entities[:demo_size])

print("Detecting key phrases.")
phrases = comp_detect.detect_key_phrases(sample_text, lang_code)
print(f"The first {demo_size} are:")
pprint(phrases[:demo_size])

print("Detecting personally identifiable information (PII).")
pii_entities = comp_detect.detect_pii(sample_text, lang_code)
print(f"The first {demo_size} are:")
pprint(pii_entities[:demo_size])

print("Detecting sentiment.")
sentiment = comp_detect.detect_sentiment(sample_text, lang_code)
print(f"Sentiment: {sentiment['Sentiment']}")
print("SentimentScore:")
pprint(sentiment["SentimentScore"])

print("Detecting syntax elements.")
syntax_tokens = comp_detect.detect_syntax(sample_text, lang_code)
print(f"The first {demo_size} are:")
pprint(syntax_tokens[:demo_size])

print("Thanks for watching!")
print("-" * 88)
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の以下のトピックを参照してください。
  - DetectDominantLanguage
  - DetectEntities
  - DetectKeyPhrases
  - DetectPiiEntities
  - DetectSentiment
  - DetectSyntax

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用してサンプルデータに対して Amazon Comprehend トピックモデリングジョブを実行する

次のコードサンプルは、以下の操作方法を示しています。

- Amazon Comprehend トピックモデリングジョブをサンプルデータに対して実行します。
- ジョブに関する情報。
- Amazon S3 からジョブ出力データを抽出します。

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon Comprehend トピックモデリングアクションを呼び出すラッパークラスを作成します。

```
class ComprehendTopicModeler:
    """Encapsulates a Comprehend topic modeler."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
        """
        self.comprehend_client = comprehend_client


    def start_job(
        self,
        job_name,
        input_bucket,
        input_key,
        input_format,
        output_bucket,
        output_key,
        data_access_role_arn,
    ):
        """
        Starts a topic modeling job. Input is read from the specified Amazon S3
        input bucket and written to the specified output bucket. Output data is
 stored
        in a tar archive compressed in gzip format. The job runs asynchronously,
 so you
        can call `describe_topics_detection_job` to get job status until it
        returns a status of SUCCEEDED.

        :param job_name: The name of the job.
        :param input_bucket: An Amazon S3 bucket that contains job input.
        :param input_key: The prefix used to find input data in the input
                          bucket. If multiple objects have the same prefix,
 all
                          of them are used.
        :param input_format: The format of the input data, either one document
 per
                          file or one document per line.
        :param output_bucket: The Amazon S3 bucket where output data is written.
        :param output_key: The prefix prepended to the output data.
        :param data_access_role_arn: The Amazon Resource Name (ARN) of a role
 that
```

```
                                    grants Comprehend permission to read from
the
                                    input bucket and write to the output bucket.
        :return: Information about the job, including the job ID.
        """
        try:
            response = self.comprehend_client.start_topics_detection_job(
                JobName=job_name,
                DataAccessRoleArn=data_access_role_arn,
                InputDataConfig={
                    "S3Uri": f"s3://{input_bucket}/{input_key}",
                    "InputFormat": input_format.value,
                },
                OutputDataConfig={"S3Uri": f"s3://{output_bucket}/{output_key}"},
            )
            logger.info("Started topic modeling job %s.", response["JobId"])
        except ClientError:
            logger.exception("Couldn't start topic modeling job.")
            raise
        else:
            return response


    def describe_job(self, job_id):
        """
        Gets metadata about a topic modeling job.

        :param job_id: The ID of the job to look up.
        :return: Metadata about the job.
        """
        try:
            response = self.comprehend_client.describe_topics_detection_job(
                JobId=job_id
            )
            job = response["TopicsDetectionJobProperties"]
            logger.info("Got topic detection job %s.", job_id)
        except ClientError:
            logger.exception("Couldn't get topic detection job %s.", job_id)
            raise
        else:
            return job


    def list_jobs(self):
```

```
        """
        Lists topic modeling jobs for the current account.

        :return: The list of jobs.
        """
        try:
            response = self.comprehend_client.list_topics_detection_jobs()
            jobs = response["TopicsDetectionJobPropertiesList"]
            logger.info("Got %s topic detection jobs.", len(jobs))
        except ClientError:
            logger.exception("Couldn't get topic detection jobs.")
            raise
        else:
            return jobs
```

ラッパークラスを使用してトピックモデリングジョブを実行し、ジョブデータを取得します。

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon Comprehend topic modeling demo!")
    print("-" * 88)

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    input_prefix = "input/"
    output_prefix = "output/"
    demo_resources = ComprehendDemoResources(
        boto3.resource("s3"), boto3.resource("iam")
    )
    topic_modeler = ComprehendTopicModeler(boto3.client("comprehend"))

    print("Setting up storage and security resources needed for the demo.")
    demo_resources.setup("comprehend-topic-modeler-demo")
    print("Copying sample data from public bucket into input bucket.")
    demo_resources.bucket.copy(
        {"Bucket": "public-sample-us-west-2", "Key": "TopicModeling/Sample.txt"},
        f"{input_prefix}sample.txt",
    )

    print("Starting topic modeling job on sample data.")
```

```
    job_info = topic_modeler.start_job(
        "demo-topic-modeling-job",
        demo_resources.bucket.name,
        input_prefix,
        JobInputFormat.per_line,
        demo_resources.bucket.name,
        output_prefix,
        demo_resources.data_access_role.arn,
    )

    print(
        f"Waiting for job {job_info['JobId']} to complete. This typically takes "
        f"20 - 30 minutes."
    )
    job_waiter = JobCompleteWaiter(topic_modeler.comprehend_client)
    job_waiter.wait(job_info["JobId"])

    job = topic_modeler.describe_job(job_info["JobId"])
    print(f"Job {job['JobId']} complete:")
    pprint(job)

    print(
        f"Getting job output data from the output Amazon S3 bucket: "
        f"{job['OutputDataConfig']['S3Uri']}."
    )
    job_output = demo_resources.extract_job_output(job)
    lines = 10
    print(f"First {lines} lines of document topics output:")
    pprint(job_output["doc-topics.csv"]["data"][:lines])
    print(f"First {lines} lines of terms output:")
    pprint(job_output["topic-terms.csv"]["data"][:lines])

    print("Cleaning up resources created for the demo.")
    demo_resources.cleanup()

    print("Thanks for watching!")
    print("-" * 88)
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の以下のトピックを参照してください。

- DescribeTopicsDetectionJob
- ListTopicsDetectionJobs
- StartTopicsDetectionJob

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用してカスタム Amazon Comprehend 分類子をトレーニングし、ドキュメントを分類する

次のコードサンプルは、以下の操作方法を示しています。

- Amazon Comprehend マルチラベル分類子を作成します。
- 分類子をサンプルデータに基づいてトレーニングします。
- 2 番目のデータセットで分類ジョブを実行します。
- Amazon S3 からジョブ出力データを抽出します。

Python

SDK for Python (Boto3)

> **Note**
>
> については、「」を参照してください GitHub。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon Comprehend ドキュメント分類子アクションを呼び出すラッパークラスを作成します。

```
class ComprehendClassifier:
    """Encapsulates an Amazon Comprehend custom classifier."""

    def __init__(self, comprehend_client):
        """
        :param comprehend_client: A Boto3 Comprehend client.
```

```
        """
        self.comprehend_client = comprehend_client
        self.classifier_arn = None


    def create(
        self,
        name,
        language_code,
        training_bucket,
        training_key,
        data_access_role_arn,
        mode,
    ):
        """
        Creates a custom classifier. After the classifier is created, it
immediately
        starts training on the data found in the specified Amazon S3 bucket.
Training
        can take 30 minutes or longer. The `describe_document_classifier`
function
        can be used to get training status and returns a status of TRAINED when
the
        classifier is ready to use.

        :param name: The name of the classifier.
        :param language_code: The language the classifier can operate on.
        :param training_bucket: The Amazon S3 bucket that contains the training
data.
        :param training_key: The prefix used to find training data in the
training
                             bucket. If multiple objects have the same prefix,
all
                             of them are used.
        :param data_access_role_arn: The Amazon Resource Name (ARN) of a role
that
                                     grants Comprehend permission to read from
the
                                     training bucket.
        :return: The ARN of the newly created classifier.
        """
        try:
            response = self.comprehend_client.create_document_classifier(
                DocumentClassifierName=name,
```

```
                LanguageCode=language_code,
                InputDataConfig={"S3Uri": f"s3://{training_bucket}/
{training_key}"},
                DataAccessRoleArn=data_access_role_arn,
                Mode=mode.value,
            )
            self.classifier_arn = response["DocumentClassifierArn"]
            logger.info("Started classifier creation. Arn is: %s.",
 self.classifier_arn)
        except ClientError:
            logger.exception("Couldn't create classifier %s.", name)
            raise
        else:
            return self.classifier_arn


    def describe(self, classifier_arn=None):
        """
        Gets metadata about a custom classifier, including its current status.

        :param classifier_arn: The ARN of the classifier to look up.
        :return: Metadata about the classifier.
        """
        if classifier_arn is not None:
            self.classifier_arn = classifier_arn
        try:
            response = self.comprehend_client.describe_document_classifier(
                DocumentClassifierArn=self.classifier_arn
            )
            classifier = response["DocumentClassifierProperties"]
            logger.info("Got classifier %s.", self.classifier_arn)
        except ClientError:
            logger.exception("Couldn't get classifier %s.", self.classifier_arn)
            raise
        else:
            return classifier


    def list(self):
        """
        Lists custom classifiers for the current account.

        :return: The list of classifiers.
        """
```

```
        try:
            response = self.comprehend_client.list_document_classifiers()
            classifiers = response["DocumentClassifierPropertiesList"]
            logger.info("Got %s classifiers.", len(classifiers))
        except ClientError:
            logger.exception(
                "Couldn't get classifiers.",
            )
            raise
        else:
            return classifiers


    def delete(self):
        """
        Deletes the classifier.
        """
        try:
            self.comprehend_client.delete_document_classifier(
                DocumentClassifierArn=self.classifier_arn
            )
            logger.info("Deleted classifier %s.", self.classifier_arn)
            self.classifier_arn = None
        except ClientError:
            logger.exception("Couldn't deleted classifier %s.",
self.classifier_arn)
            raise


    def start_job(
        self,
        job_name,
        input_bucket,
        input_key,
        input_format,
        output_bucket,
        output_key,
        data_access_role_arn,
    ):
        """
        Starts a classification job. The classifier must be trained or the job
        will fail. Input is read from the specified Amazon S3 input bucket and
        written to the specified output bucket. Output data is stored in a tar
```

```
        archive compressed in gzip format. The job runs asynchronously, so you
can
        call `describe_document_classification_job` to get job status until it
        returns a status of SUCCEEDED.

        :param job_name: The name of the job.
        :param input_bucket: The Amazon S3 bucket that contains input data.
        :param input_key: The prefix used to find input data in the input
                          bucket. If multiple objects have the same prefix, all
                          of them are used.
        :param input_format: The format of the input data, either one document
per
                             file or one document per line.
        :param output_bucket: The Amazon S3 bucket where output data is written.
        :param output_key: The prefix prepended to the output data.
        :param data_access_role_arn: The Amazon Resource Name (ARN) of a role
that
                                     grants Comprehend permission to read from
the
                                     input bucket and write to the output bucket.
        :return: Information about the job, including the job ID.
        """
        try:
            response = self.comprehend_client.start_document_classification_job(
                DocumentClassifierArn=self.classifier_arn,
                JobName=job_name,
                InputDataConfig={
                    "S3Uri": f"s3://{input_bucket}/{input_key}",
                    "InputFormat": input_format.value,
                },
                OutputDataConfig={"S3Uri": f"s3://{output_bucket}/{output_key}"},
                DataAccessRoleArn=data_access_role_arn,
            )
            logger.info(
                "Document classification job %s is %s.", job_name,
response["JobStatus"]
            )
        except ClientError:
            logger.exception("Couldn't start classification job %s.", job_name)
            raise
        else:
            return response
```

```
    def describe_job(self, job_id):
        """
        Gets metadata about a classification job.

        :param job_id: The ID of the job to look up.
        :return: Metadata about the job.
        """
        try:
            response =
self.comprehend_client.describe_document_classification_job(
                JobId=job_id
            )
            job = response["DocumentClassificationJobProperties"]
            logger.info("Got classification job %s.", job["JobName"])
        except ClientError:
            logger.exception("Couldn't get classification job %s.", job_id)
            raise
        else:
            return job


    def list_jobs(self):
        """
        Lists the classification jobs for the current account.

        :return: The list of jobs.
        """
        try:
            response = self.comprehend_client.list_document_classification_jobs()
            jobs = response["DocumentClassificationJobPropertiesList"]
            logger.info("Got %s document classification jobs.", len(jobs))
        except ClientError:
            logger.exception(
                "Couldn't get document classification jobs.",
            )
            raise
        else:
            return jobs
```

シナリオを実行するクラスを作成します。

```
class ClassifierDemo:
    """
    Encapsulates functions used to run the demonstration.
    """

    def __init__(self, demo_resources):
        """
        :param demo_resources: A ComprehendDemoResources class that manages
 resources
                               for the demonstration.
        """
        self.demo_resources = demo_resources
        self.training_prefix = "training/"
        self.input_prefix = "input/"
        self.input_format = JobInputFormat.per_line
        self.output_prefix = "output/"

    def setup(self):
        """Creates AWS resources used by the demo."""
        self.demo_resources.setup("comprehend-classifier-demo")

    def cleanup(self):
        """Deletes AWS resources used by the demo."""
        self.demo_resources.cleanup()

    @staticmethod
    def _sanitize_text(text):
        """Removes characters that cause errors for the document parser."""
        return text.replace("\r", " ").replace("\n", " ").replace(",", ";")

    @staticmethod
    def _get_issues(query, issue_count):
        """
        Gets issues from GitHub using the specified query parameters.

        :param query: The query string used to request issues from the GitHub
 API.
        :param issue_count: The number of issues to retrieve.
        :return: The list of issues retrieved from GitHub.
        """
        issues = []
        logger.info("Requesting issues from %s?%s.", GITHUB_SEARCH_URL, query)
```

```
        response = requests.get(f"{GITHUB_SEARCH_URL}?
{query}&per_page={issue_count}")
        if response.status_code == 200:
            issue_page = response.json()["items"]
            logger.info("Got %s issues.", len(issue_page))
            issues = [
                {
                    "title": ClassifierDemo._sanitize_text(issue["title"]),
                    "body": ClassifierDemo._sanitize_text(issue["body"]),
                    "labels": {label["name"] for label in issue["labels"]},
                }
                for issue in issue_page
            ]
        else:
            logger.error(
                "GitHub returned error code %s with message %s.",
                response.status_code,
                response.json(),
            )
        logger.info("Found %s issues.", len(issues))
        return issues

    def get_training_issues(self, training_labels):
        """
        Gets issues used for training the custom classifier. Training issues are
        closed issues from the Boto3 repo that have known labels. Comprehend
        requires a minimum of ten training issues per label.

        :param training_labels: The issue labels to use for training.
        :return: The set of issues used for training.
        """
        issues = []
        per_label_count = 15
        for label in training_labels:
            issues += self._get_issues(
                f"q=type:issue+repo:boto/boto3+state:closed+label:{label}",
                per_label_count,
            )
            for issue in issues:
                issue["labels"] = issue["labels"].intersection(training_labels)
        return issues

    def get_input_issues(self, training_labels):
        """
```

```
        Gets input issues from GitHub. For demonstration purposes, input issues
        are open issues from the Boto3 repo with known labels, though in practice
        any issue could be submitted to the classifier for labeling.

        :param training_labels: The set of labels to query for.
        :return: The set of issues used for input.
        """
        issues = []
        per_label_count = 5
        for label in training_labels:
            issues += self._get_issues(
                f"q=type:issue+repo:boto/boto3+state:open+label:{label}",
                per_label_count,
            )
        return issues

    def upload_issue_data(self, issues, training=False):
        """
        Uploads issue data to an Amazon S3 bucket, either for training or for
input.
        The data is first put into the format expected by Comprehend. For
training,
        the set of pipe-delimited labels is prepended to each document. For
        input, labels are not sent.

        :param issues: The set of issues to upload to Amazon S3.
        :param training: Indicates whether the issue data is used for training or
                         input.
        """
        try:
            obj_key = (
                self.training_prefix if training else self.input_prefix
            ) + "issues.txt"
            if training:
                issue_strings = [
                    f"{'|'.join(issue['labels'])},{issue['title']}
{issue['body']}"
                    for issue in issues
                ]
            else:
                issue_strings = [
                    f"{issue['title']} {issue['body']}" for issue in issues
                ]
            issue_bytes = BytesIO("\n".join(issue_strings).encode("utf-8"))
```

```
            self.demo_resources.bucket.upload_fileobj(issue_bytes, obj_key)
            logger.info(
                "Uploaded data as %s to bucket %s.",
                obj_key,
                self.demo_resources.bucket.name,
            )
        except ClientError:
            logger.exception(
                "Couldn't upload data to bucket %s.",
self.demo_resources.bucket.name
            )
            raise

    def extract_job_output(self, job):
        """Extracts job output from Amazon S3."""
        return self.demo_resources.extract_job_output(job)

    @staticmethod
    def reconcile_job_output(input_issues, output_dict):
        """
        Reconciles job output with the list of input issues. Because the input
issues
        have known labels, these can be compared with the labels added by the
        classifier to judge the accuracy of the output.

        :param input_issues: The list of issues used as input.
        :param output_dict: The dictionary of data that is output by the
classifier.
        :return: The list of reconciled input and output data.
        """
        reconciled = []
        for archive in output_dict.values():
            for line in archive["data"]:
                in_line = int(line["Line"])
                in_labels = input_issues[in_line]["labels"]
                out_labels = {
                    label["Name"]
                    for label in line["Labels"]
                    if float(label["Score"]) > 0.3
                }
                reconciled.append(
                    f"{line['File']}, line {in_line} has labels {in_labels}.\n"
                    f"\tClassifier assigned {out_labels}."
                )
```

```
        logger.info("Reconciled input and output labels.")
        return reconciled
```

既知のラベル GitHub に関する問題のセットについて分類子をトレーニングし、2 つ目の GitHub 問題セットを分類子に送信してラベル付けできるようにします。

```
def usage_demo():
    print("-" * 88)
    print("Welcome to the Amazon Comprehend custom document classifier demo!")
    print("-" * 88)

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    comp_demo = ClassifierDemo(
        ComprehendDemoResources(boto3.resource("s3"), boto3.resource("iam"))
    )
    comp_classifier = ComprehendClassifier(boto3.client("comprehend"))
    classifier_trained_waiter = ClassifierTrainedWaiter(
        comp_classifier.comprehend_client
    )
    training_labels = {"bug", "feature-request", "dynamodb", "s3"}

    print("Setting up storage and security resources needed for the demo.")
    comp_demo.setup()

    print("Getting training data from GitHub and uploading it to Amazon S3.")
    training_issues = comp_demo.get_training_issues(training_labels)
    comp_demo.upload_issue_data(training_issues, True)

    classifier_name = "doc-example-classifier"
    print(f"Creating document classifier {classifier_name}.")
    comp_classifier.create(
        classifier_name,
        "en",
        comp_demo.demo_resources.bucket.name,
        comp_demo.training_prefix,
        comp_demo.demo_resources.data_access_role.arn,
        ClassifierMode.multi_label,
    )
    print(
```

```
        f"Waiting until {classifier_name} is trained. This typically takes "
        f"30-40 minutes."
    )
    classifier_trained_waiter.wait(comp_classifier.classifier_arn)

    print(f"Classifier {classifier_name} is trained:")
    pprint(comp_classifier.describe())

    print("Getting input data from GitHub and uploading it to Amazon S3.")
    input_issues = comp_demo.get_input_issues(training_labels)
    comp_demo.upload_issue_data(input_issues)

    print("Starting classification job on input data.")
    job_info = comp_classifier.start_job(
        "issue_classification_job",
        comp_demo.demo_resources.bucket.name,
        comp_demo.input_prefix,
        comp_demo.input_format,
        comp_demo.demo_resources.bucket.name,
        comp_demo.output_prefix,
        comp_demo.demo_resources.data_access_role.arn,
    )
    print(f"Waiting for job {job_info['JobId']} to complete.")
    job_waiter = JobCompleteWaiter(comp_classifier.comprehend_client)
    job_waiter.wait(job_info["JobId"])

    job = comp_classifier.describe_job(job_info["JobId"])
    print(f"Job {job['JobId']} complete:")
    pprint(job)

    print(
        f"Getting job output data from Amazon S3: "
        f"{job['OutputDataConfig']['S3Uri']}."
    )
    job_output = comp_demo.extract_job_output(job)
    print("Job output:")
    pprint(job_output)

    print("Reconciling job output with labels from GitHub:")
    reconciled_output = comp_demo.reconcile_job_output(input_issues, job_output)
    print(*reconciled_output, sep="\n")

    answer = input(f"Do you want to delete the classifier {classifier_name} (y/
n)? ")
```

```
    if answer.lower() == "y":
        print(f"Deleting {classifier_name}.")
        comp_classifier.delete()

    print("Cleaning up resources created for the demo.")
    comp_demo.cleanup()

    print("Thanks for watching!")
    print("-" * 88)
```

- API の詳細については、『AWS SDK for Python (Boto3) API リファレンス』の以下のトピックを参照してください。
  - CreateDocumentClassifier
  - DeleteDocumentClassifier
  - DescribeDocumentClassificationJob
  - DescribeDocumentClassifier
  - ListDocumentClassificationJobs
  - ListDocumentClassifiers
  - StartDocumentClassificationJob

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

SDK を使用した Amazon Comprehend のクロスサービスの例 AWS SDKs

次のサンプルアプリケーションでは AWS SDKs を使用して Amazon Comprehend を他の と組み合わせます AWS のサービス。各例には GitHub、アプリケーションのセットアップと実行の手順を示す へのリンクが含まれています。

例

- Amazon Transcribe ストリーミングアプリケーションを構築する

- Amazon Lex チャットボットを作成して、ウェブサイトの訪問者を引き付けましょう
- Amazon SQS を使用してメッセージを送受信するウェブアプリケーションを作成する
- 顧客からのフィードバックを分析し、音声を合成するアプリケーションの作成
- AWS SDK を使用してイメージから抽出されたテキスト内のエンティティを検出する

Amazon Transcribe ストリーミングアプリケーションを構築する

次のコード例は、ライブ音声をリアルタイムで記録、転写、翻訳し、結果を E メールで送信するアプリケーションを構築する方法を示しています。

JavaScript

SDK for JavaScript (v3)

Amazon Transcribe を使用して、ライブ音声をリアルタイムで記録、転写、翻訳し、Amazon Simple Email Service (Amazon SES) を使用して結果を E メールで送信するアプリケーションを構築する方法について説明します。

完全なソースコードとセットアップと実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Lex チャットボットを作成して、ウェブサイトの訪問者を引き付けましょう

次のコード例は、Web サイトの訪問者を引き付けるチャットボットの作成方法を示しています。

Java

SDK for Java 2.x

ウェブアプリケーション内に Amazon Lex chatbotを作成して、ウェブサイトの訪問者に対応することができます。

完全なソースコードとセットアップと実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

JavaScript

SDK for JavaScript (v3)

ウェブアプリケーション内に Amazon Lex chatbotを作成して、ウェブサイトの訪問者に対応することができます。

完全なソースコードとセットアップと実行の手順については、 AWS SDK for JavaScript デベロッパーガイドのAmazon Lexチャットボットの構築」の完全な例を参照してください。

この例で使用されているサービス

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon SQS を使用してメッセージを送受信するウェブアプリケーションを作成する

次のコード例は、Amazon SQS を使用してメッセージングアプリケーションを作成する方法を示しています。

Java

SDK for Java 2.x

Amazon SQS API を使用して、メッセージを送受信する Spring REST API を開発する方法を示します。

完全なソースコードとセットアップと実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Amazon Comprehend
- Amazon SQS

Kotlin

SDK for Kotlin

Amazon SQS API を使用して、メッセージを送受信する Spring REST API を開発する方法を示します。

完全なソースコードとセットアップと実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Amazon Comprehend
- Amazon SQS

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

顧客からのフィードバックを分析し、音声を合成するアプリケーションの作成

次のコード例は、顧客のコメントカードを分析し、それを元の言語から翻訳し、顧客の感情を判断し、翻訳されたテキストから音声ファイルを生成するアプリケーションの作成方法を示しています。

.NET

AWS SDK for .NET

このサンプルアプリケーションは、顧客フィードバックカードを分析し、保存します。具体的には、ニューヨーク市の架空のホテルのニーズを満たします。このホテルでは、お客様からのフィードバックをさまざまな言語で書かれた実際のコメントカードの形で受け取ります。そのフィードバックは、ウェブクライアントを通じてアプリにアップロードされます。コメントカードの画像をアップロードされると、次の手順が発生します。

- テキストは Amazon Textract を使用して、画像から抽出されます。
- Amazon Comprehend は、抽出されたテキストの感情とその言語を決定します。
- 抽出されたテキストは、Amazon Translate を使用して英語に翻訳されます。
- Amazon Polly は抽出されたテキストからオーディオファイルを合成します。

完全なアプリは AWS CDK を使用してデプロイすることができます。ソースコードとデプロイ手順については、「」の「 GitHubプロジェクト」を参照してください。

この例で使用されているサービス

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Java

SDK for Java 2.x

このサンプルアプリケーションは、顧客フィードバックカードを分析し、保存します。具体的には、ニューヨーク市の架空のホテルのニーズを満たします。このホテルでは、お客様から

のフィードバックをさまざまな言語で書かれた実際のコメントカードの形で受け取ります。そのフィードバックは、ウェブクライアントを通じてアプリにアップロードされます。コメントカードの画像をアップロードされると、次の手順が発生します。

- テキストは Amazon Textract を使用して、画像から抽出されます。
- Amazon Comprehend は、抽出されたテキストの感情とその言語を決定します。
- 抽出されたテキストは、Amazon Translate を使用して英語に翻訳されます。
- Amazon Polly は抽出されたテキストからオーディオファイルを合成します。

完全なアプリは AWS CDK を使用してデプロイすることができます。ソースコードとデプロイ手順については、「」の「 GitHubプロジェクト」を参照してください。

この例で使用されているサービス

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

JavaScript

SDK for JavaScript (v3)

このサンプルアプリケーションは、顧客フィードバックカードを分析し、保存します。具体的には、ニューヨーク市の架空のホテルのニーズを満たします。このホテルでは、お客様からのフィードバックをさまざまな言語で書かれた実際のコメントカードの形で受け取ります。そのフィードバックは、ウェブクライアントを通じてアプリにアップロードされます。コメントカードの画像をアップロードされると、次の手順が発生します。

- テキストは Amazon Textract を使用して、画像から抽出されます。
- Amazon Comprehend は、抽出されたテキストの感情とその言語を決定します。
- 抽出されたテキストは、Amazon Translate を使用して英語に翻訳されます。
- Amazon Polly は抽出されたテキストからオーディオファイルを合成します。

完全なアプリは AWS CDK を使用してデプロイすることができます。ソースコードとデプロイ手順については、「」の「 GitHubプロジェクト」を参照してください。以下の抜粋は、AWS SDK for JavaScript が Lambda 関数内でどのように使用されるかを示しています。

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";
```

```
/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
```

```
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});
```

```
  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

この例で使用されているサービス

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Ruby

SDK for Ruby

このサンプルアプリケーションは、顧客フィードバックカードを分析し、保存します。具体的には、ニューヨーク市の架空のホテルのニーズを満たします。このホテルでは、お客様からのフィードバックをさまざまな言語で書かれた実際のコメントカードの形で受け取ります。そのフィードバックは、ウェブクライアントを通じてアプリにアップロードされます。コメントカードの画像をアップロードされると、次の手順が発生します。

- テキストは Amazon Textract を使用して、画像から抽出されます。
- Amazon Comprehend は、抽出されたテキストの感情とその言語を決定します。
- 抽出されたテキストは、Amazon Translate を使用して英語に翻訳されます。
- Amazon Polly は抽出されたテキストからオーディオファイルを合成します。

完全なアプリは AWS CDK を使用してデプロイすることができます。ソースコードとデプロイ手順については、「」の「 GitHubプロジェクト」を参照してください。

この例で使用されているサービス

- Amazon Comprehend

- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用してイメージから抽出されたテキスト内のエンティティを検出する

次のコード例は、Amazon Comprehend を使用して、Amazon S3 に格納されている画像から Amazon Textract によって抽出されたテキスト内のエンティティを検出する方法を示しています。

Python

SDK for Python (Boto3)

Jupyter Notebook AWS SDK for Python (Boto3) で を使用して、イメージから抽出されたテキスト内のエンティティを検出する方法を示します。この例では、Amazon Textract を使用して Amazon Simple Storage Service (Amazon S3) に保存されている画像からテキストを抽出し、Amazon Comprehend を使用して、抽出されたテキスト内のエンティティを検出します。

この例は Jupyter Notebook であり、ノートブックをホストできる環境で実行する必要があります。Amazon を使用して例を実行する方法については SageMaker、「.TextractAndComprehendNotebookipynb」の手順を参照してください。

完全なソースコードとセットアップと実行の手順については、「」の詳細な例を参照してくださいGitHub。

この例で使用されているサービス

- Amazon Comprehend
- Amazon S3
- Amazon Textract

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「」を参照してくださいAWS SDK での Amazon Comprehend の使用。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Comprehend におけるセキュリティ

AWS ではクラウドセキュリティが最優先事項です。セキュリティを最も重視する組織の要件を満たすために構築された AWS のデータセンターとネットワークアーキテクチャは、お客様に大きく貢献します。

セキュリティは、AWS とお客様とが共有する責務です。責任共有モデルでは、このことをクラウドのセキュリティおよびクラウド内のセキュリティと表現しています。

- クラウドのセキュリティ - AWS は、AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する責任を負います。また AWS は、安全に使用できるサービスを提供します。AWS コンプライアンスプログラムの一環として、サードパーティー監査者が定期的にセキュリティの有効性をテストおよび検証します。Amazon Comprehend に該当するコンプライアンスプログラムの詳細については、「AWSコンプライアンスプログラムの対象範囲のサービス」「」を参照してください。
- クラウド内のセキュリティ - ユーザーの責任は、使用する AWS サービスに応じて異なります。また、お客様は、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、Amazon Comprehend を使用する際に責任共有モデルを適用する方法を理解するのに役立ちます。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するように Amazon Comprehend を設定する方法を説明しています。また、Amazon Comprehend のリソースの監視と安全確保に役立つ、他の AWS のサービスの使用方法も説明しています。

トピック

- Amazon Comprehend におけるデータ保護
- Amazon Comprehend における Identity and Access Management
- AWS CloudTrail での Amazon Comprehend API コールのログ記録
- Amazon Comprehend のコンプライアンス検証
- Amazon Comprehend における耐障害性
- Amazon Comprehend におけるインフラストラクチャセキュリティ

Amazon Comprehend におけるデータ保護

Amazon Comprehend でのデータ保護には、AWS の責任共有モデルが適用されます。このモデルで説明されているように、AWS は、AWS クラウド のすべてを実行するグローバルインフラストラクチャを保護するがあります。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービスのセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、「AWS セキュリティブログ」に投稿された「AWS 責任共有モデルおよび GDPR」のブログ記事を参照してください。

データを保護するため、AWS アカウント の認証情報を保護し、AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーをセットアップすることをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみを各ユーザーに付与できます。また、次の方法でデータを保護することをおすすめします。

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 が必須です。TLS 1.3 が推奨されます。
- AWS CloudTrail で API とユーザーアクティビティロギングをセットアップします。
- AWS のサービス内でデフォルトである、すべてのセキュリティ管理に加え、AWS の暗号化ソリューションを使用します。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API により AWS にアクセスするときに FIPS 140-2 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

お客様の E メールアドレスなどの機密情報やセンシティブ情報は、タグや [名前] フィールドなどの自由形式のフィールドに配置しないことを強くお勧めします。このことは、コンソールやAPI、AWS CLI、AWS SDK で Amazon Comprehend やその他の AWS のサービス を使用する場合にも当てはまります。名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

トピック

- Amazon Comprehend における暗号化
- サービス間の混乱した代理の防止
- Amazon Virtual Private Cloud を使用してジョブを保護する
- Amazon Comprehend とインターフェイス VPC エンドポイント (AWS PrivateLink)

Amazon Comprehend における暗号化

Amazon Comprehend と AWS Key Management Service (AWS KMS) と連携することで、データの暗号化を強化が図られています。Amazon S3 では、すでにテキスト分析、トピックモデリング、またはカスタム Amazon Comprehend でのジョブを作成する際に、入力ドキュメントを暗号化することができます。AWS KMS との統合により、ストレージボリューム内の Start* ジョブおよび Create* ジョブのデータを暗号化でき、これにより、Start* ジョブの出力結果は独自の KMS キーを使用して暗号化されます。

AWS Management Console の場合、Amazon Comprehend は独自の KMS キーを使用してカスタムモデルを暗号化します。AWS CLI の場合は、Amazon Comprehend 独自の KMS キーまたは提供されたカスタマーマネージドキー (CMK) を使用してカスタムモデルを暗号化できます。

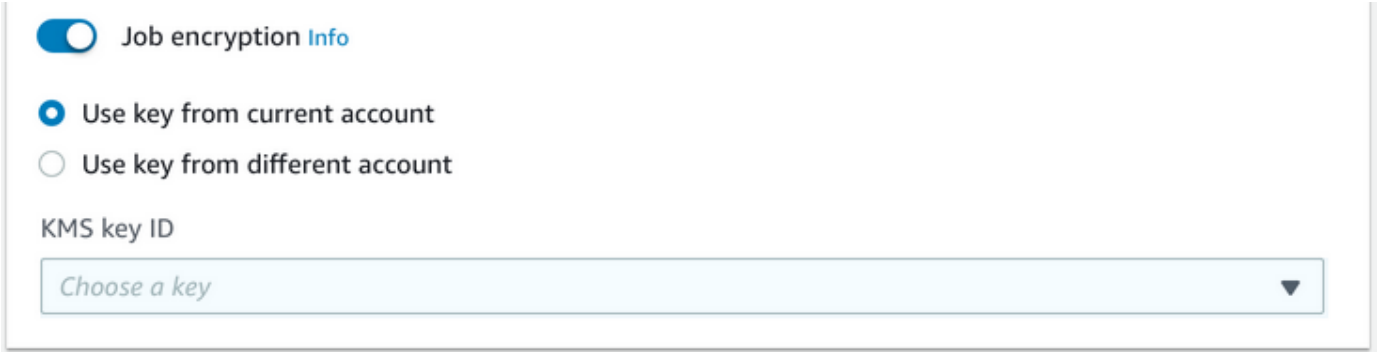
AWS Management Console を使用した KMS 暗号化

コンソール使用時には、次の 2 つの暗号化オプションがあります。

- ボリュームの暗号化
- 出力結果の暗号化

ボリュームで暗号化を有効にする

1. [ジョブ設定] で、[ジョブ暗号化] オプションを選択します。



2. KMS カスタマー管理キー (CMK) が、現在使用しているアカウントのものか、別のアカウントのものかを選択します。現在のアカウントのキーを使用する場合は、[KMS キー ID] でキーを選択します。別のアカウントのキーを使用する場合は、キーの ARN を入力する必要があります。

出力結果の暗号化を有効にするには

1. [出力設定] から、[暗号化] オプションを選択します。

2. カスタマーマネージドキー (CMK) が、現在使用しているアカウントのキーか、別のアカウントのキーかを選択します。現在のアカウントのキーを使用する場合は、[KMS キー ID] からキーを選択します。別のアカウントのキーを使用する場合は、キーの ARN を入力する必要があります。

S3 上の入力ドキュメントに SSE-KMS を使用した暗号化を以前にセットアップしたことがあると、セキュリティが強化されていることがあります。ただし、その場合は、使用する IAM ロールに、入力ドキュメントを暗号化する KMS キーに対する `kms:Decrypt` アクセス許可が必要です。詳細については、「KMS 暗号化を使用するために必要なアクセス許可」を参照してください。

API オペレーションによる KMS 暗号化

Amazon Comprehend `Start*` および `Create*` API オペレーションのすべてで、 KMS で暗号化された入力ドキュメントを利用することができます。元のジョブの入力が `KmsKeyId` である場合、`Describe*` および `List*` API オペレーションは `OutputDataConfig` に `KmsKeyId` を返します。入力でなかった場合は、返されません。

これは、 StartEntitiesDetectionJobオペレーションを使用した次の AWS CLI の例で確認できます。

```
aws comprehend start-entities-detection-job \
     --region region \
     --data-access-role-arn "data access role arn" \
```

```
    --entity-recognizer-arn "entity recognizer arn" \
    --input-data-config "S3Uri=s3://Bucket Name/Bucket Path" \
    --job-name job name \
    --language-code en \
    --output-data-config "KmsKeyId=Output S3 KMS key ID" "S3Uri=s3://Bucket
Name/Bucket Path/" \
    --volumekmskeyid "Volume KMS key ID"
```

> **Note**
>
> この例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

API オペレーションによるカスタマーマネージドキー (CMK) の暗号化

Amazon Comprehend のカスタムモデル API オペレーション、`CreateEntityRecognizer`、`CreateDocumentClassifier`、`CreateEndpoint` では、AWS CLI 経由でカスタマーマネージドキーによる暗号化を行うことができます。

プリンシパルによるカスタマーマネージドキーの使用または管理を許可するには、IAM ポリシーが必要です。これらのキーは、ポリシーステートメントの `Resource` 要素で指定されます。ベストプラクティスは、プリンシパルがポリシーステートメントで使用する必要があるキーにカスタマーマネージドキーを制限することです。

次の CLI AWS の例では、 CreateEntityRecognizerオペレーションを使用して、モデル暗号化でカスタムエンティティレコグナイザーを作成します。

```
aws comprehend create-entity-recognizer \
    --recognizer-name name \
    --data-access-role-arn data access role arn \
    --language-code en \
    --model-kms-key-id Model KMS Key ID \
    --input-data-config file:///path/input-data-config.json
```

> **Note**
>
> この例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

サービス間の混乱した代理の防止

混乱した代理問題とは、アクションを実行する許可を持たないエンティティが、より高い特権を持つエンティティにそのアクションの実行を強制できるというセキュリティ問題です。AWS では、サービス間でのなりすましによって、混乱した代理問題が発生する場合があります。サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。これを防ぐために、AWS には、アカウント内のリソースへのアクセス権が付与されたサービスプリンシパルですべてのサービスのデータを保護するために役立つツールが用意されています。

リソースポリシー内では aws:SourceArn および aws:SourceAccount のグローバル条件コンテキストキーを使用して、Amazon Comprehend が別のサービスに付与する、リソースへのアクセス許可を制限することをお勧めします。両方のグローバル条件コンテキストキーを同じポリシーステートメントで使用する場合は、aws:SourceAccount 値と、aws:SourceArn 値に含まれるアカウントが、同じアカウント ID を示している必要があります。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定して aws:SourceArn グローバル条件コンテキストキーを使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合は、aws:SourceArn グローバルコンテキスト条件キーを使用して、ARN の未知部分をワイルドカード (*) で表します。例えば、arn:aws:*servicename*::*123456789012*:* のように指定します。

ソースアカウントの使用

次の例は、Amazon Comprehend で aws:SourceAccount グローバル条件コンテキストキーを使用する方法を示しています。

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
          "Service": "comprehend.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
        "StringEquals": {
```

```
                "aws:SourceAccount":"111122223333"
            }
        }
    }
}
```

暗号化モデルのエンドポイントの信頼ポリシー

暗号化モデルのエンドポイントを作成または更新するには、信頼ポリシーを作成する必要があります。 aws:SourceAccount の値をアカウント ID に設定します。ArnEquals 条件を使用する場合は、aws:SourceArn 値をエンドポイントの ARN に設定します。

```
{
 "Version": "2012-10-17",
 "Statement": [
    {
        "Sid": "",
        "Effect": "Allow",
        "Principal": {
            "Service": "comprehend.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
               "aws:SourceAccount": "111122223333"
            },
            "ArnEquals": {
               "aws:SourceArn": "arn:aws:comprehend:us-west-2:111122223333:document-
classifier-endpoint/endpoint-name"
            }
        }
    }
  ]
}
```

カスタムモデルを作成する

カスタムモデルを作成するには、ポリシーを作成する必要があります。aws:SourceAccount 値をアカウント ID に設定します。ArnEquals 条件を使用する場合は、aws:SourceArn 値をカスタムモデルバージョンの ARN に設定します。

```
{
```

```
 "Version": "2012-10-17",
  "Statement": [
      {
          "Sid": "",
          "Effect": "Allow",
          "Principal": {
              "Service": "comprehend.amazonaws.com"
          },
          "Action": "sts:AssumeRole",
          "Condition": {
              "StringEquals": {
                  "aws:SourceAccount": "111122223333"
              },
              "ArnEquals": {
                  "aws:SourceArn": "arn:aws:comprehend:us-west-2:111122223333:
                            document-classifier/smallest-classifier-test/
version/version-name"
              }
          }
      }
  ]
}
```

Amazon Virtual Private Cloud を使用してジョブを保護する

Amazon Comprehend では、さまざまなセキュリティ対策を使用して、データおよび、Amazon Comprehend の使用中にそのデータが保存されるジョブコンテナの安全性を確保することができます。ただし、ジョブコンテナは、データやモデルアーティファクトを保存する Amazon S3 バケットなどの AWS リソースにインターネット経由でアクセスします。

データへのアクセスを制御するには、仮想プライベートクラウド (VPC) を作成して、データおよびコンテナがインターネット経由でアクセスできないように設定することをお勧めします。VPC の作成と設定の詳細については、「Amazon VPC ユーザーガイド」の「Amazon VPC の開始方法」を参照してください。VPC を利用すると、インターネットに接続されないように VPC を設定できるため、ジョブコンテナとデータを保護することができます。VPC 利用すると、VPC フローログを使ってジョブコンテナとの間のすべてのネットワークトラフィックを監視することもできます。詳細については、Amazon VPC ユーザーガイドのVPC フローログを参照してください。

VPC 設定は、サブネットとセキュリティグループを指定してモデルを作成するときに指定します。サブネットとセキュリティグループが指定されると、Amazon Comprehend はサブネットの 1 つのセキュリティグループに関連付けられている Elastic Network Interface (ENI) を作成しま

す。ENI により、ジョブコンテナが VPC のリソースに接続できるようになります。ENI については、『Amazon VPC ユーザーガイド』の「Elastic Network Interfaces」を参照してください。

> **Note**
>
> ジョブの場合は、デフォルトのテナンシー VPC を使用してのみサブネットを設定できます。この VPC では、インスタンスは共有ハードウェアで実行されます。VPCsAmazon EC2 ユーザーガイド」の「ハードウェア専有インスタンス」を参照してください。

Amazon VPC アクセス用のジョブを設定する

プライベート VPC 上のサブネットとセキュリティグループを指定するには、該当する API の `VpcConfig` リクエストパラメータを使用するか、Amazon Comprehend コンソールでコンパイルジョブを作成するときにその情報を指定します。Amazon Comprehend は、この情報を使用して ENI を作成し、その ENI をジョブコンテナにアタッチします。ENI は、インターネットに接続されていない VPC 上でのネットワーク接続機能をモデルコンテナに提供します。

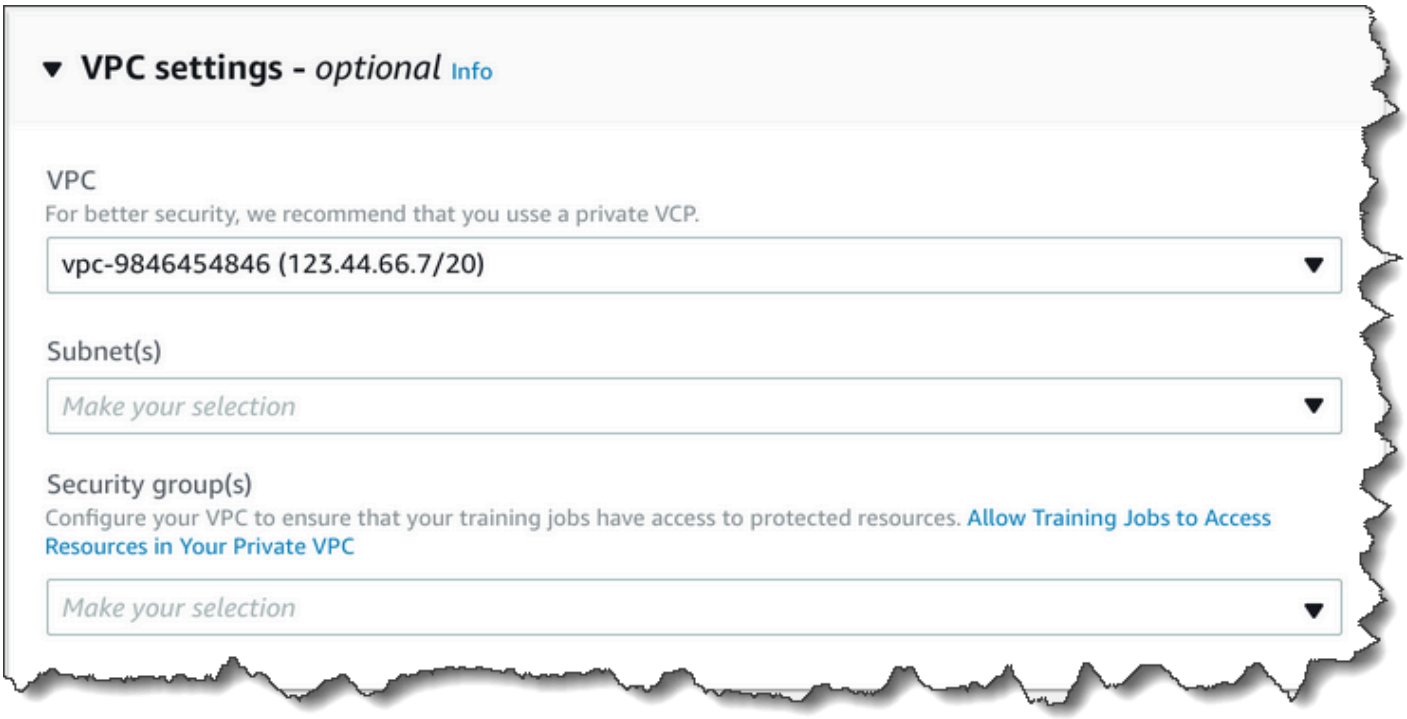
次の API には `VpcConfig` リクエストパラメータが含まれています。

- `Create*` API: `CreateDocumentClassifier`、 `CreateEntityRecognizer`
- `Start*` API: `StartDocumentClassificationJob`、 `StartDominantLanguageDetectionJob`、 `StartEntitiesDetectionJob`、 `StartKeyPhrasesDetectionJob`、 `StartSentimentDetectionJob`、 `StartTargetedSentimentDetectionJob`、 `StartTopicsDetectionJob`

API コールに含める VpcConfig パラメータの例を次に示します。

```
"VpcConfig": {
      "SecurityGroupIds": [
          " sg-0123456789abcdef0"
          ],
      "Subnets": [
          "subnet-0123456789abcdef0",
          "subnet-0123456789abcdef1",
          "subnet-0123456789abcdef2"
          ]
      }
```

Amazon Comprehend コンソールから VPC を設定するには、ジョブの作成時にオプションの [VPC 設定] セクションから設定の詳細を選択します。



Amazon Comprehend ジョブ用に VPC を設定する

Amazon Comprehend のジョブ用に VPC 設定する場合は、次のガイドラインに従ってください。VPC のセットアップについては、Amazon VPC ユーザーガイドの「VPC とサブネットの使用」を参照してください。

サブネットに十分な IP アドレスを確保する

VPC サブネットには、ジョブの各インスタンスにプライベート IP アドレスが少なくとも 2 つ必要です。詳細については、Amazon VPC ユーザーガイドの IPv4 用の VPC とサブネットのサイズ設定を参照してください。

Amazon S3 VPC エンドポイントを 1 つ作成する

モデルコンテナがインターネットにアクセスできないように VPC を設定した場合は、アクセスを許可する VPC エンドポイントを作成しない限り、モデルコンテナはデータを含む Amazon S3 バケットに接続できません。VPC エンドポイントを作成することで、トレーニングジョブと分析ジョブ中にジョブコンテナがデータにアクセスできるようになります。

VPC エンドポイントを作成するときは、次の値を設定します。

- サービスとしてAWS サービスカテゴリを選択する
- サービスを として指定します。 com.amazonaws.*region*.s3
- VPC エンドポイントタイプとしてゲートウェイを選択する

AWS CloudFormation を使用して VPC エンドポイントを作成する場合は、AWS CloudFormation VPCEndpointのドキュメントに従ってください。次の例は、 AWS CloudFormation テンプレート内の VPCEndpoint 設定を示しています。

```
VpcEndpoint:
  Type: AWS::EC2::VPCEndpoint
  Properties:
    PolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Action:
            - s3:GetObject
            - s3:PutObject
            - s3:ListBucket
            - s3:GetBucketLocation
            - s3:DeleteObject
            - s3:ListMultipartUploadParts
            - s3:AbortMultipartUpload
          Effect: Allow
          Resource:
            - "*"
          Principal: "*"
    RouteTableIds:
      - Ref: RouteTable
    ServiceName:
      Fn::Join:
        - ''
        - - com.amazonaws.
          - Ref: AWS::Region
          - ".s3"
    VpcId:
      Ref: VPC
```

プライベート VPC からのリクエストのみに S3 バケットへのアクセスを許可するカスタムポリシーも作成することをお勧めします。詳細については、Amazon VPC ユーザーガイドの「Amazon S3 におけるエンドポイント」を参照してください。

次のポリシーでは、S3 バケットへのアクセスを許可します。このポリシーを編集して、ジョブに必要なリソースのみへのアクセスを許可します。

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:DeleteObject",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload"
            ],
            "Resource": "*"
        }
    ]
}
```

エンドポイントルートテーブルのデフォルトの DNS 設定を使って、標準 Amazon S3 URL (例えば、`http://s3-aws-region.amazonaws.com/DOC-EXAMPLE-BUCKET`) が解決されるようにします。デフォルトの DNS 設定を使用しない場合は、エンドポイントルートテーブルを設定することで、ジョブのデータがある場所の指定に使用する URL が解決されるようにします。VPC エンドポイントルートテーブルについては、Amazon VPC ユーザーガイドの「ゲートウェイエンドポイントのルーティング」を参照してください。

デフォルトエンドポイントポリシーでは、ユーザーは、Amazon Linux と Amazon Linux 2 のリポジトリのパッケージをジョブコンテナにインストールできます。ユーザーがそのリポジトリからパッケージをインストールしないようにする場合は、Amazon Linux と Amazon Linux 2 のリポジトリへのアクセスを明示的に拒否するカスタムエンドポイントポリシーを作成します。Comprehend 自体にはそのようなパッケージは必要なく、機能的な影響はありません。これらのリポジトリへのアクセスを拒否するポリシーの例を次に示します。

```
{
    "Statement": [
      {
        "Sid": "AmazonLinuxAMIRepositoryAccess",
        "Principal": "*",
        "Action": [
            "s3:GetObject"
        ],
        "Effect": "Deny",
        "Resource": [
            "arn:aws:s3:::packages.*.amazonaws.com/*",
            "arn:aws:s3:::repo.*.amazonaws.com/*"
        ]
      }
    ]
}

{
    "Statement": [
        { "Sid": "AmazonLinux2AMIRepositoryAccess",
          "Principal": "*",
          "Action": [
              "s3:GetObject"
              ],
          "Effect": "Deny",
          "Resource": [
              "arn:aws:s3:::amazonlinux.*.amazonaws.com/*"
              ]
         }
    ]
}
```

DataAccessRole に対するアクセス許可

分析ジョブで VPC を使用する場合、DataAccessRole および Create* オペレーションに使用する Start* には、入力ドキュメントと出力バケットへのアクセスに使用される VPC に対するアクセス許可も必要です。

次のポリシーは、Create* および Start* オペレーションに使用される DataAccessRole に、必要とされるアクセス権を提供します。

```
{
```

```
    "Version": "2008-10-17",
    "Statement": [
        {
            "Effect": "Allow",
              "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DeleteNetworkInterface",
                "ec2:DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeVpcs",
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

VPC セキュリティグループを設定する

分散型ジョブでは、同じジョブにあるさまざまなコンテナ間の通信を許可する必要があります。そのためには、同じセキュリティグループのメンバー間のインバウンド接続を許可するセキュリティグループのルールを設定します。詳細については、『Amazon VPC ユーザーガイド』の「Security Group Rules」を参照してください。

VPC の外部のリソースに接続する

インターネットアクセスができないように VPC を設定している場合、その VPC を使用するジョブには、VPC の外部のリソースに対するアクセス権がありません。ジョブが VPC の外部のリソースにアクセスする必要がある場合は、次のいずれかのオプションを使用してアクセス権を提供します。

- ジョブがインターフェイス VPC エンドポイントをサポートする AWS サービスにアクセスする必要がある場合は、そのサービスに接続するためのエンドポイントを作成します。インターフェイスエンドポイントをサポートするサービスのリストについては、Amazon VPC ユーザーガイドの「VPC エンドポイント」を参照してください。インターフェイス VPC エンドポイントの作成の詳細については、Amazon VPC ユーザーガイドの「インターフェイス VPC エンドポイント (AWS PrivateLink) 」を参照してください。
- ジョブがインターフェイス VPC エンドポイントをサポートしていない AWS サービスまたは 外のリソースにアクセスする必要がある場合は AWS、NAT ゲートウェイを作成し、アウトバウンド接

続を許可するようにセキュリティグループを設定します。VPC 用の NAT ゲートウェイのセットアップについては、『Amazon VPC ユーザーガイド』の「シナリオ 2: パブリックサブネットとプライベートサブネットを持つ VPC (NAT)」を参照してください。

Amazon Comprehend とインターフェイス VPC エンドポイント (AWS PrivateLink)

インターフェイス VPC エンドポイントを作成すると、VPC と Amazon Comprehend との間のプライベート接続を確立できます。インターフェイスエンドポイントには AWS PrivateLink テクノロジーが利用されており、このテクノロジーによって、 インターネットゲートウェイやNAT デバイス、VPN 接続、AWS Direct Connect 接続がなくても、Amazon Comprehend API にプライベートにアクセスできます。VPC のインスタンスはパブリック IP アドレスがなくても Amazon Comprehend API と通信できます。VPC と Amazon Comprehend API 間のトラフィックが、Amazon ネットワークを離れることはありません。

各インターフェイスエンドポイントは、サブネット内の 1 つ以上の Elastic Network Interface によって表されます。

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

Amazon Comprehend VPC エンドポイントに関する考慮事項

Amazon Comprehend のインターフェイス VPC エンドポイントを設定するにあたっては、『Amazon VPC ユーザーガイド』の「インターフェイスエンドポイントのプロパティと制限」をお読みください。

Amazon Comprehend エンドポイントは、リージョン内のすべてのアベイラビリティーゾーンで使用できるわけではありません。アベイラビリティーゾーンは、エンドポイントの作成時に次のコマンドを使用して一覧表示できます。

```
aws ec2 describe-vpc-endpoint-services \
  --service-names com.amazonaws.us-west-2.comprehend
```

Amazon Comprehend では、VPC からすべての API アクションを呼び出すことができます。

Amazon Comprehend 用のインターフェイス VPC エンドポイントの作成

Amazon Comprehend サービス用の VPC エンドポイントは、Amazon VPC コンソールまたは AWS Command Line Interface (AWS CLI) を使用して作成できます。詳細については、「Amazon VPC ユーザーガイド」のインターフェイスエンドポイントの作成を参照してください。

Amazon Comprehend 用の VPC エンドポイントを作成には、次のサービス名を使用します。

- com.amazonaws.*region*.comprehend

エンドポイントに対してプライベート DNS を有効にすると、リージョンのデフォルト DNS 名 (*comprehend.us-east-1.amazonaws.com* など) を使用して、Amazon Comprehend への API リクエストを実行できます。

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントを介したサービスへのアクセス」を参照してください。

Amazon Comprehend 用の VPC エンドポイントポリシーの作成

Amazon Comprehend へのアクセスを制御する VPC エンドポイントにエンドポイントポリシーをアタッチできます。このポリシーでは、以下の情報を指定します。

- アクションを実行できるプリンシパル。
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイントでサービスへのアクセスを制御する」を参照してください。

例: Amazon Comprehend アクション用の VPC エンドポイントポリシー

Amazon Comprehend 用のエンドポイントポリシーの例を次に示します。このポリシーは、エンドポイントにアタッチされると、すべてのリソースのすべてのプリンシパルに対して、Amazon Comprehend `DetectEntities` アクションへのアクセスを許可します。

```
{
   "Statement":[
      {
```

```
            "Principal":"*",
            "Effect":"Allow",
            "Action":[
               "comprehend:DetectEntities"
            ],
            "Resource":"*"
         }
      ]
}
```

Amazon Comprehend における Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証するか (サインインさせるか) と、誰に Amazon Comprehend リソースの使用を許可するか (アクセス許可を付与するか) を制御します。IAM は、追加料金なしで AWS のサービス 使用できる です。

トピック

- 対象者
- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- Amazon Comprehend と IAM 連携の仕組み
- Amazon Comprehend の ID ベースのポリシー例
- Amazon Comprehend 向けの AWS マネージドポリシー
- Amazon Comprehend のアイデンティティとアクセスのトラブルシューティング

対象者

AWS Identity and Access Management (IAM) の使用方法は、Amazon Comprehend で行う作業によって異なります。

サービスユーザー – ジョブを実行するために Amazon Comprehend サービスを使用する場合は、管理者から必要なアクセス許可と認証情報が与えられます。さらに多くの Amazon Comprehend 機能を使用して作業を行う場合は、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解しておくと、管理者に適切な許可をリクエストするうえで役立ちます。Amazon

Comprehend の機能にアクセスできない場合は、「Amazon Comprehend のアイデンティティとアクセスのトラブルシューティング」を参照してください。

サービス管理者 – 社内の Amazon Comprehend リソースを担当しているユーザーには、通常、Amazon Comprehend への完全なアクセス権限があります。サービスユーザーがアクセスする Amazon Comprehend の機能とリソースを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。Amazon Comprehend で IAM を利用する方法の詳細については、「Amazon Comprehend と IAM 連携の仕組み」を参照してください。

IAM 管理者 – 通常 IAM 管理者には、Amazon Comprehend へのアクセスを管理するポリシーの作成方法に関する詳細情報が必要になります。IAM で使用できる Amazon Comprehend の ID ベースのポリシー例については、「Amazon Comprehend の ID ベースのポリシー例」を参照してください。

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けて認証（ にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center（ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報は、フェデレーション ID の例です。フェデレーティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 ユーザーガイド」の「 へのサインイン AWS アカウント方法AWS サインイン 」を参照してください。

AWS プログラムで にアクセスする場合、 は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストに暗号で署名します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。推奨される方法を使用してリクエストを自分で署名する方法の詳細については、IAM ユーザーガイドの API AWS リクエストの署名を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、 AWS では、多要素認証 (MFA) を使用してアカウントのセキュリティを向上させることをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「Multi-factor

authentication」(多要素認証) および「IAM ユーザーガイド」の「AWSでの多要素認証 (MFA) の使用」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての AWS のサービス およびリソースへの完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、IAM ユーザーガイドのルートユーザー認証情報が必要なタスクを参照してください。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、一時的な認証情報を使用して にアクセスするための ID プロバイダーとのフェデレーションの使用を要求 AWS のサービス します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、アイデンティティセンターディレクトリのユーザー、または ID ソースを通じて提供された認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーティッド ID が にアクセスすると AWS アカウント、ロールを引き受け、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。IAM Identity Center でユーザーとグループを作成することも、独自の ID ソース内のユーザーとグループのセットに接続して同期して、すべての AWS アカウント とアプリケーションで使用することもできます。IAM Identity Center の詳細については、「AWS IAM Identity Center ユーザーガイド」の「What is IAM Identity Center?」(IAM Identity Center とは) を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、IAM ユーザーガイドの長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションするを参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdminsという名前のグループを設定して、そのグループにIAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザー (ロールではなく) の作成が適している場合」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。ロール を切り替える AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API AWS CLI オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「Creating a role for a third-party Identity Provider」(サードパーティーアイデンティティプロバイダー向けロールの作成) を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、 (ロールをプロキシとして使用する代わりに) ポリシーをリソースに直接アタッチできま

す。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

- クロスサービスアクセス — 一部の は、他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して でアクションを実行する場合 AWS、ユーザーはプリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストのリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。
  - サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示できますが、編集することはできません。
- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。 AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、IAM ユーザーガイドのAmazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与するを参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、IAM ユーザーガイドの(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、 AWS ID またはリソースにアタッチします。ポリシーは AWS 、アイデンティティまたはリソースに関連付けられているときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、IAM ユーザーガイドのJSON ポリシー概要を参照してください。

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole`アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、IAM ユーザーガイドのIAM ポリシーの作成を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、IAM ユーザーガイドのマネージドポリシーとインラインポリシーの比較を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシー や Amazon S3 バケットポリシー があげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、 AWS WAF、および Amazon VPC は、ACLs。ACL の詳細については、Amazon Simple Storage Service デベロッパーガイドのアクセスコントロールリスト (ACL) の概要を参照してください。

その他のポリシータイプ

AWS は、一般的ではない追加のポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースのポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、IAM ユーザーガイドのIAM エンティティのアクセス許可の境界を参照してください。
- サービスコントロールポリシー (SCPs) – SCPs は、 の組織または組織単位 (OU) に対する最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、 AWS

アカウント ビジネスが所有する複数の をグループ化して一元管理するサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、AWS Organizations ユーザーガイドの「SCP の仕組み」を参照してください。

- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、IAM ユーザーガイドのセッションポリシーを参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、IAM ユーザーガイドの「ポリシー評価ロジック」を参照してください。

Amazon Comprehend と IAM 連携の仕組み

Amazon Comprehend へのアクセス権は、IAM を使用して管理できます。ここでは、Amazon Comprehend で利用できる IAM の機能を説明します。

Amazon Comprehend で利用できる IAM の機能

IAM 機能	Amazon Comprehend のサポート
アイデンティティベースのポリシー	あり
リソースベースのポリシー	あり
ポリシーアクション	あり
ポリシーリソース	はい
ポリシー条件キー (サービス固有)	はい

IAM 機能	Amazon Comprehend のサポート
ACL	なし
ABAC (ポリシー内のタグ)	部分的
一時的な認証情報	あり
転送アクセスセッション (FAS)	あり
サービスロール	あり
サービスリンクロール	なし

Amazon Comprehend およびその他の AWS のサービスがほとんどの IAM 機能と連携する方法の概要を把握するには、「IAM ユーザーガイド」のAWS 「IAM と連携する のサービス」を参照してください。

Amazon Comprehend の ID ベースのポリシー

アイデンティティベースポリシーをサポートする	あり

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、IAM ユーザーガイドのIAM ポリシーの作成を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素について学ぶには、IAM ユーザーガイドのIAM JSON ポリシーの要素のリファレンスを参照してください。

Amazon Comprehend の ID ベースのポリシー例

Amazon Comprehend の ID ベースのポリシー例については、「Amazon Comprehend の ID ベースのポリシー例」を参照してください。

Amazon Comprehend 内のリソースベースのポリシー

リソースベースのポリシーのサポート	あり

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシー や Amazon S3 バケットポリシー があげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、アカウント全体、または別のアカウントの IAM エンティティをリソースベースのポリシーのプリンシパルとして指定します。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる にある場合 AWS アカウント、信頼されたアカウントの IAM 管理者は、プリンシパルエンティティ (ユーザーまたはロール) にリソースへのアクセス許可も付与する必要があります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーをさらに付与する必要はありません。詳細については、「IAM ユーザーガイド」の「IAM でのクロスアカウントリソースアクセス」を参照してください。

Amazon Comprehend サービスは、カスタムモデルに追加された 1 つのタイプのリソースベースのポリシー (カスタムモデルポリシー) のみをサポートします。そのポリシーでは、カスタムモデルを利用できる他のアカウントを定義します。

リソースベースのポリシーをカスタムモデルにアタッチする方法については、「カスタムモデル用のリソースベースのポリシー」を参照してください。

Amazon Comprehend におけるポリシーアクション

ポリシーアクションに対するサポート	あり

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

JSON ポリシーのAction要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連付けられた AWS API オペレーションと同じです。一致する API オペレーションのない許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは、依存アクションと呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

Amazon Comprehend アクションのリストの表示については、『サービス認証リファレンス』の「Amazon Comprehend ID で定義されるアクション」を参照してください。

Amazon Comprehend のポリシーアクションでは、アクションの前に次のプレフィックスを使用します。

```
comprehend
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "comprehend:DetectSentiment",
      "comprehend:ClassifyDocument"
         ]
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、Describe という単語で始まるすべてのアクションを指定するには、次のアクションを含めます。

```
"Action": "comprehend:Describe*"
```

1 つのサービスに対してすべてのアクションを指定するワイルドカードを使用しないでください。ポリシーでのアクセス許可の指定では、最小特権を付与するというベストプラクティスに従います。

Amazon Comprehend の ID ベースのポリシー例については、「Amazon Comprehend の ID ベースのポリシー例」を参照してください。

Amazon Comprehend におけるポリシーリソース

ポリシーリソースに対するサポート	あり

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースにどのような条件でアクションを実行できるかということです。

Resource JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource 要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

Amazon Comprehend リソースのタイプと ARN については、『サービス認証リファレンス』の「Amazon Comprehend ID で定義されるリソースタイプ」のリストを参照してください。各リソースの ARN を指定できるアクションについては、「Amazon Comprehend で定義されているアクション」を参照してください。

Amazon Comprehend におけるポリシー条件キー

サービス固有のポリシー条件キーのサポート	あり

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどんなリソースにどんな条件でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定する場合、または 1 つの Condition 要素に複数のキーを指定する場合、 AWS では AND 論理演算子を使用してそれらを評価します。1 つの条

件キーに複数の値を指定すると、 は論理ORオペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、IAM ユーザーガイドのIAM ポリシーの要素: 変数およびタグを参照してください。

AWS は、グローバル条件キーとサービス固有の条件キーをサポートします。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「 グローバル条件コンテキストキー」を参照してください。

Amazon Comprehend の条件キーについては、『サービス認証リファレンス』の「Amazon Comprehend ID の条件キー」のリストを参照してください。条件キーを使用できるアクションおよびリソースについては、「Amazon Comprehend で定義されているアクション」を参照してください。

Amazon Comprehend の ID ベースのポリシー例については、「Amazon Comprehend の ID ベースのポリシー例」を参照してください。

Amazon Comprehend における ACL

ACL のサポート	なし

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかをコントロールします。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

ABAC と Amazon Comprehend

ABAC (ポリシー内のタグ) のサポート	部分的

属性ベースのアクセス制御 (ABAC) は、属性に基づいてアクセス許可を定義する認可戦略です。では AWS、これらの属性はタグ と呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール) および多くの AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。その後、プリンシパルのタグがアクセスしようとしているリソースのタグと一致した場合にオペレーションを許可するように ABAC ポリシーをします。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを管理するには、aws:ResourceTag/*key-name*、aws:RequestTag/*key-name*、または aws:TagKeys の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値はありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、IAM ユーザーガイドのABAC とは?を参照してください。ABAC をセットアップするステップを説明するチュートリアルについては、IAM ユーザーガイドの属性に基づくアクセスコントロール (ABAC) を使用するを参照してください。

Amazon Comprehend リソースに対するタグ付けの詳細については、「リソースのタグ付け」を参照してください。

Amazon Comprehend での一時的な認証情報の利用

一時的な認証情報のサポート	あり

一部の は、一時的な認証情報を使用してサインインすると機能 AWS のサービス しません。一時的な認証情報 AWS のサービス を使用する などの詳細については、IAM ユーザーガイドのAWS のサービス「IAM と連携する 」を参照してください。

ユーザー名とパスワード以外の AWS Management Console 方法で にサインインする場合、一時的な認証情報を使用します。例えば、会社の Single Sign-On (SSO) リンク AWS を使用して にアクセスすると、そのプロセスによって一時的な認証情報が自動的に作成されます。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、IAM ユーザーガイドのロールへの切り替え (コンソール)を参照してください。

一時的な認証情報は、 AWS CLI または AWS API を使用して手動で作成できます。その後、これらの一時的な認証情報を使用して . AWS recommends にアクセスできます AWS。これは、長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成することを推奨しています。詳細については、IAM の一時的セキュリティ認証情報を参照してください。

Amazon Comprehend のフォワードアクセスセッション

転送アクセスセッション (FAS) をサポート	あり

IAM ユーザーまたはロールを使用して でアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストのリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

Amazon Comprehend におけるサービスロール

サービスロールに対するサポート	あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。

> **Warning**
>
> サービスロールに対するアクセス許可を変更すると、Amazon Comprehend の機能に問題が発生する場合があります。Amazon Comprehend からの指示がある場合を除いて、サービスロールを編集しないでください。

Amazon Comprehend 非同期オペレーションを使用するには、Amazon Comprehend にドキュメントコレクションが含まれている Amazon S3 バケットへのアクセス許可を付与する必要があります。このためには、Amazon Comprehend サービスプリンシパルと信頼関係を構築できるように、アカウントにデータアクセスロールを作成します。

ポリシー例については、「バッチ操作に必要なロールベースのアクセス許可」を参照してください。

Amazon Comprehend におけるサービスリンクロール

サービスにリンクされたロールのサポート	なし

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示できますが、編集することはできません。

サービスリンクロールの作成または管理の詳細については、IAM と提携するAWS のサービスを参照してください。表の中から、[Service-linked role] (サービスにリンクされたロール) 列に Yes と記載されたサービスを見つけます。サービスリンクロールに関するドキュメントをサービスで表示するには、はい リンクを選択します。

Amazon Comprehend の ID ベースのポリシー例

デフォルトでは、ユーザーおよびロールには Amazon Comprehend リソースを作成または変更するアクセス許可がありません。また、AWS Management Console、AWS Command Line InterfaceAWS CLI、または AWS API を使用してタスクを実行することもできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者がロールに IAM ポリシーを追加すると、ユーザーはロールを引き受けることができます。

これらサンプルの JSON ポリシードキュメントを使用して、IAM アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

Amazon Comprehend が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、『サービス認可リファレンス』の「Amazon Comprehend のアクション、リソース、および条件キー」を参照してください。

トピック

- ポリシーのベストプラクティス
- Amazon Comprehend コンソールの使用法
- ユーザーが自分のアクセス許可を表示できるようにする方法
- ドキュメント分析アクションを実行するために必要なアクセス許可
- KMS 暗号化を使用するために必要なアクセス許可
- Amazon Comprehend の AWS マネージド (事前定義) ポリシー
- バッチ操作に必要なロールベースのアクセス許可

- Amazon Comprehend のすべてのアクションを許可するアクセス許可
- トピックモデリングアクションを許可するアクセス許可
- カスタム非同期分析ジョブに必要なアクセス許可

ポリシーのベストプラクティス

ID ベースのポリシーでは、アカウント内で誰が Amazon Comprehend リソースを作成、アクセス、または削除できるを決定します。これらのアクションを実行すると、AWS アカウント に料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS マネージドポリシーを使用して開始し、最小特権の権限に移行する – ユーザーとワークロードへの権限の付与を開始するには、多くの一般的なユースケースのために権限を付与する AWS マネージドポリシーを使用します。これらは AWS アカウントで使用できます。ユースケースに応じた AWS カスタマーマネージドポリシーを定義することで、権限をさらに減らすことをお勧めします。詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで権限を設定するときは、タスクの実行に必要な権限のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権権限とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、「IAM ユーザーガイド」の「IAM でのポリシーと権限」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。また、AWS CloudFormation などの特定の AWS のサービス を介して使用する場合、条件を使用してサービスアクションへのアクセスを許可することもできます。詳細については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素: 条件」を参照してください。
- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、『IAM ユーザーガイド』の「IAM Access Analyzer ポリシーの検証」を参照してください。
- 多要素認証 (MFA) を要求する - AWS アカウント内の IAM ユーザーまたはルートユーザーを要求するシナリオがある場合は、セキュリティを強化するために MFA をオンにします。API オペレー

ションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、「IAM ユーザーガイド」の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、『IAM ユーザーガイド』の「IAM でのセキュリティのベストプラクティス」を参照してください。

Amazon Comprehend コンソールの使用法

Amazon Comprehend コンソールにアクセスするには、最小限のアクセス許可セットが必要です。それらのアクセス許可では、AWS アカウント の Amazon Comprehend リソースに関する詳細の表示と参照を許可する必要があります。最小限必要なアクセス許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) ではコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソール権限を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスを許可します。

Amazon Comprehend コンソールの最小アクセス許可では、*ComprehendReadOnly* および AWS 管理ポリシーをエンティティにアタッチできます。詳細については、IAM ユーザーガイドの「ユーザーへの許可の追加」を参照してください。

Amazon Comprehend コンソールを使用するには、次のポリシーに示されているアクションのアクセス許可も必要です。

```
{
  "Version": "2012-10-17",
  "Statement": [
  {
      "Action": [
          "iam:ListRoles",
          "iam:GetRole",
          "s3:ListAllMyBuckets",
          "s3:ListBucket",
          "s3:GetBucketLocation"
      ],
      "Effect": "Allow",
      "Resource": "*"
  }
  ]
```

```
  }
```

Amazon Comprehend コンソールには、以下の理由でこれらの追加のアクセス許可が必要になります。

- アカウントで使用可能な IAM ロールをリストするための iam アクセス許可。
- トピックモデリング用のデータが含まれる Amazon S3 バケットおよびオブジェクトにアクセスするための s3 アクセス許可。

コンソールを使用して非同期バッチジョブまたはトピックモデリングジョブを作成する場合は、ジョブ用の IAM ロールをコンソールに作成させるオプションがあります。ユーザーが IAM ロールを作成するには、IAM ロールとポリシーを作成してロールにポリシーをアタッチするための以下のアクセス許可が追加で必要になります。

```
{
  "Version": "2012-10-17",
  "Statement":
    [
      {
        "Action":
          [
          "iam:CreateRole",
          "iam:CreatePolicy",
          "iam:AttachRolePolicy"
          ],
        "Effect": "Allow",
        "Resource": "*"
      },
      {
        "Action":
          [
          "iam:PassRole"
          ],
        "Effect": "Allow",
        "Resource": "arn:aws:iam::*:role/*Comprehend*"
      }
    ]
  }
```

Amazon Comprehend コンソールには、以下の理由でこれらの追加のアクセス許可が必要になります。

- ロールを作成し、ポリシーをロールにアタッチするための iam アクセス許可。iam:PassRole アクションにより、コンソールは Amazon Comprehend にロールを渡すことができます。

ユーザーが自分のアクセス許可を表示できるようにする方法

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI か AWS API を使用してプログラム的に、このアクションを完了する権限が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

ドキュメント分析アクションを実行するために必要なアクセス許可

次のポリシー例では、Amazon Comprehend ドキュメント分析アクションを使用するためのアクセス許可を付与しています。

```
{
 "Version": "2012-10-17",
 "Statement": [{
    "Sid": "AllowDetectActions",
    "Effect": "Allow",
    "Action": [
              "comprehend:DetectEntities",
              "comprehend:DetectKeyPhrases",
              "comprehend:DetectDominantLanguage",
              "comprehend:DetectSentiment",
              "comprehend:DetectTargetedSentiment",
              "comprehend:DetectSyntax",
              "textract:DetectDocumentText",
              "textract:AnalyzeDocument"
           ],
    "Resource": "*"
    }
 ]
}
```

このポリシーには、DetectEntities、DetectKeyPhrases、DetectDominantLanguage、DetectTargetedSentim および DetectSyntax アクションを使用するためのアクセス許可を付与するステートメントが 1 つあります。このポリシーステートメントでは、2 つの Amazon Textract API メソッドを使用するためのアクセス許可も付与しています。Amazon Comprehend はこれらのメソッドを呼び出すことで、画像ファイルやスキャンした PDF ドキュメントからテキストを抽出します。このような種類の入力ファイルに対してカスタム推論を実行することがないユーザーの場合は、これらのアクセス許可を削除できます。

このポリシーが適用されるユーザーが、アカウント内でバッチアクションや非同期アクションを実行することはできません。

アイデンティティベースのポリシーでアクセス権限を得るプリンシパルを指定していないため、ポリシーでは Principal エレメントを指定していません。ユーザーにポリシーをアタッチすると、そのユーザーが暗黙のプリンシパルになります。IAM ロールにアクセス権限ポリシーをアタッチすると、ロールの信頼ポリシーで識別されたプリンシパルがアクセス権限を得ることになります。

すべての Amazon Comprehend API アクションとそれらが適用されるリソースについては、『サービス認証リファレンス』の「Amazon Comprehend のアクション、リソース、および条件キー」の表を参照してください。

KMS 暗号化を使用するために必要なアクセス許可

Amazon Key Management Service (KMS) を非同期ジョブのデータおよびジョブの暗号化に完全に使用するには、次のポリシーに示すアクションに対するアクセス許可を付与する必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
        "Action": [
            "kms:CreateGrant"
        ],
        "Effect": "Allow",
        "Resource": "*"
    },
    {
        "Action": [
            "kms:Decrypt",
            "kms:GenerateDatakey"
        ],
        "Effect": "Allow",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "kms:ViaService": [
                    "s3.region.amazonaws.com"
                ]
            }
        }
    }
  ]
}
```

Amazon Comprehend を使用した非同期ジョブの作成では、Amazon S3 に保存されている入力データを使用します。S3 には、保存されたデータを暗号化するオプションがあり、その暗号化は Amazon Comprehend ではなく S3 によって行われます。Amazon Comprehend ジョブが使用するデータアクセスロールに、元の入力データの暗号化に使用されたキーへの `kms:Decrypt` アクセス許可を与えると、暗号化された入力データを復号化して読み取ることができます。

KMS カスタマーマネージドキー (CMK) を使用すると、 S3 で出力結果を暗号化したり、ジョブ処理中に使用されるストレージボリュームを暗号化したりすることもできます。こうした暗号化では、両方の種類の暗号化に同じ KMS キーを使用できますが、必須ではありません。ジョブの作成では出力暗号化とボリューム暗号化のキーをそれぞれ別のフィールドに指定できます。また、別のアカウントの KMS キーを使用することもできます。

KMS 暗号化を使用する場合、ボリュームの暗号化には `kms:CreateGrant` アクセス許可、出力データの暗号化には `kms:GenerateDataKey` アクセス許可がそれぞれ必要です。暗号化された入力を読み取るには (入力データがすでに Amazon S3 によって暗号化されている場合など)、`kms:Decrypt` アクセス許可が必要です。IAM ロールは、必要に応じてそれらアクセス許可を提供する必要があります。ただし、キーが現在使用されているものとは異なるアカウントのものである場合、その KMS キーの KMS キーポリシーは、ジョブのデータアクセスロールにもそれらアクセス許可を付与する必要があります。

Amazon Comprehend の AWS マネージド (事前定義) ポリシー

AWS は、AWS によって作成され管理されるスタンドアロンの IAM ポリシーを提供することで、多くの一般的ユースケースに対応します。これらの AWS 管理ポリシーは、一般的ユースケースに必要なアクセス権限を付与することで、どの権限が必要なのかをユーザーが調査する必要をなくすことができます。詳細については、IAM ユーザーガイドの AWS管理ポリシーを参照してください。

アカウントのユーザーにアタッチできる次の AWS 管理ポリシーは、Amazon Comprehend に固有です。

- ComprehendFullAccess — トピックモデリングジョブの実行を含む Amazon Comprehend リソースへのフルアクセスを許可します。IAM ロールをリストおよび取得するアクセス許可が含まれます。
- ComprehendReadOnly – `StartDominantLanguageDetectionJob`、、、`StartEntitiesDetectionJob`、`StartKeyPhrase` および を除くすべての Amazon Comprehend `StartSentimentDetectionJob`アクションを実行するアクセス許可を付与`StartTargetedSentimentDetectionJob`しますStartTopicsDetectionJob。

Amazon Comprehend を使用するユーザーには、次の追加ポリシーを適用する必要があります。

```
{
    "Version": "2012-10-17",
    "Statement":
      [
        {
          "Action":
            [
              "iam:PassRole"
            ],
          "Effect": "Allow",
          "Resource": "arn:aws:iam::*:role/*Comprehend*"
        }
      ]
  }
```

IAM コンソールにサインインし、特定のポリシーを検索することで、管理アクセス許可ポリシーを確認できます。

これらのポリシーは、AWS SDK または AWS CLI を使用すると機能します。

独自のカスタム IAM ポリシーを作成して、Amazon Comprehend アクションとリソースに対するアクセス許可を付与することもできます。こうしたカスタムポリシーは、該当するアクセス許可が必要なユーザーやグループ、ロールにアタッチできます。

バッチ操作に必要なロールベースのアクセス許可

Amazon Comprehend 非同期オペレーションを使用するには、Amazon Comprehend にドキュメントコレクションが含まれている Amazon S3 バケットへのアクセス許可を付与します。このためには、Amazon Comprehend サービスプリンシパルと信頼関係を構築できるように、アカウントにデータアクセスロールを作成します。ロールの作成の詳細は、『AWS Identity and Access Management ユーザーガイド』の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。

以下は、作成したロールに対する信頼ポリシーの例を示しています。混乱した代理の防止に役立つようにするには、1 つ以上のグローバル条件コンテキストキーを使用してアクセス許可の範囲を制限します。`aws:SourceAccount` 値をアカウント ID に設定します。`ArnEquals` 条件を使用する場合は、`aws:SourceArn` 値にジョブの ARN を設定します。ARN のジョブ番号にはワイルドカードを使用してください。この番号は、Amazon Comprehend によってジョブの作成時に生成されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
          "Service": "comprehend.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
          "StringEquals": {
            "aws:SourceAccount": "111122223333"
          },
          "ArnEquals": {
            "aws:SourceArn": "arn:aws:comprehend:us-west-2:111122223333:pii-entities-
detection-job/*"
          }
        }
      }
    ]
 }
```

ロールを作成したら、そのロールに対するアクセスポリシーを作成します。これにより、入力データを含む Amazon S3 バケットへのアクセス許可が Amazon S3 の `GetObject` および `ListBucket` に付与され、Amazon S3 出力データバケットへのアクセス許可が Amazon S3 の `PutObject` に付与されます。

Amazon Comprehend のすべてのアクションを許可するアクセス許可

AWS にサインアップしたら、ユーザーの作成やユーザーのアクセス許可の管理など、アカウントを管理するための管理者ユーザーを作成します。

Amazon Comprehend の操作では、Amazon Comprehend のすべてのアクションに対するアクセス許可を持つユーザー (このユーザーはサービス別の管理者とみなすことができる) を作成することもできます。このユーザーに以下のアクセス権限をアタッチできます。

```
{
  "Version": "2012-10-17",
  "Statement":
    [
      {
```

```
        "Sid": "AllowAllComprehendActions",
        "Effect": "Allow",
        "Action":
        [
            "comprehend:*",
            "iam:ListRoles",
            "iam:GetRole",
            "s3:ListAllMyBuckets",
            "s3:ListBucket",
            "s3:GetBucketLocation",
            "iam:CreateRole",
            "iam:CreatePolicy",
            "iam:AttachRolePolicy",
            "kms:CreateGrant",
            "kms:Decrypt",
            "kms:GenerateDatakey"
      ],
      "Resource": "*"
    },
    {
      "Action":
        [
          "iam:PassRole"
        ],
      "Effect": "Allow",
      "Resource": "arn:aws:iam::*:role/*Comprehend*"
    }
  ]
}
```

これらのアクセス許可の暗号化関係は、以下の方法で変更できます。

- 暗号化された S3 バケットに保存されているドキュメントを Amazon Comprehend が分析できるようにするには、IAM ロールに`kms:Decrypt` アクセス許可が必要です。
- 分析ジョブを処理するコンピュートインスタンスに接続されたストレージボリュームに保存されているドキュメントを Amazon Comprehend が暗号化できるようにするには、IAM ロールに `kms:CreateGrant` アクセス許可が必要です。
- S3 バケットの出力結果を Amazon Comprehend が暗号化できるようにするには、IAM ロールに `kms:GenerateDataKey` アクセス許可が必要です。

トピックモデリングアクションを許可するアクセス許可

次のアクセス許可ポリシーでは、Amazon Comprehend のトピックモデリングオペレーションを実行するためのアクセス許可がユーザーに付与しています。

```
{
  "Version": "2012-10-17",
  "Statement": [{
  "Sid": "AllowTopicModelingActions",
  "Effect": "Allow",
  "Action": [
            "comprehend:DescribeTopicsDetectionJob",
            "comprehend:ListTopicsDetectionJobs",
            "comprehend:StartTopicsDetectionJob",
         ],
        "Resource": "*"
        ]
    }
  ]
  }
```

カスタム非同期分析ジョブに必要なアクセス許可

> **Important**
>
> モデルへのアクセス権を制限する IAM ポリシーがある場合、カスタムモデルで推論ジョブを実行することはできません。IAM ポリシーを更新して、カスタム非同期分析ジョブ用のワイルドカードリソースを追加する必要があります。

StartDocumentClassificationJob および StartEntitiesDetectionJob APIs を使用している場合は、現在ワイルドカードをリソースとして使用していない限り、IAM ポリシーを更新する必要があります。事前トレーニング済みモデルStartEntitiesDetectionJobを使用して を使用している場合、これはユーザーに影響しないため、変更を加える必要はありません。

以下のポリシー例には、古いリファレンスが含まれています。

```
{
    "Action": [
        "comprehend:StartDocumentClassificationJob",
        "comprehend:StartEntitiesDetectionJob",
```

```
    ],
    "Resource": [
        "arn:aws:comprehend:us-east-1:123456789012:document-classifier/myClassifier",
        "arn:aws:comprehend:us-east-1:123456789012:entity-recognizer/myRecognizer"
    ],
    "Effect": "Allow"
}
```

これは、 StartDocumentClassificationJob と を正常に実行するために使用する必要がある更新されたポリシーです StartEntitiesDetectionJob。

```
{
    "Action": [
        "comprehend:StartDocumentClassificationJob",
        "comprehend:StartEntitiesDetectionJob",
    ],
    "Resource": [
        "arn:aws:comprehend:us-east-1:123456789012:document-classifier/myClassifier",
        "arn:aws:comprehend:us-east-1:123456789012:document-classification-job/*",
        "arn:aws:comprehend:us-east-1:123456789012:entity-recognizer/myRecognizer",
        "arn:aws:comprehend:us-east-1:123456789012:entities-detection-job/*"
    ],
    "Effect": "Allow"
}
```

Amazon Comprehend 向けの AWS マネージドポリシー

ユーザー、グループ、ロールにアクセス許可を追加するには、自分でポリシーを作成するよりも、AWS 管理ポリシーを使用する方が簡単です。チームに必要な権限のみを提供する IAM カスタマーマネージドポリシーを作成するには、時間と専門知識が必要です。すぐに使用を開始するために、AWS マネージドポリシーを使用できます。これらのポリシーは、一般的なユースケースをターゲット範囲に含めており、AWS アカウント で利用できます。AWS マネージドポリシーの詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

AWS のサービスは、AWS マネージドポリシーを維持および更新します。AWS マネージドポリシーの許可を変更することはできません。サービスでは、新しい機能を利用できるようにするために、AWS マネージドポリシーに権限が追加されることがあります。この種類の更新は、ポリシーがアタッチされている、すべてのアイデンティティ (ユーザー、グループおよびロール) に影響を与えます。新しい機能が立ち上げられた場合や、新しいオペレーションが使用可能になった場合に、各サービスが AWS マネージドポリシーを更新する可能性が最も高くなります。サービスは、AWS マ

ネージドポリシーから権限を削除しないため、ポリシーの更新によって既存の権限が破棄されることはありません。

さらに、AWS では、複数のサービスにまたがるジョブ機能のためのマネージドポリシーもサポートしています。例えば、ReadOnlyAccess AWS マネージドポリシーでは、すべての AWS のサービスおよびリソースへの読み取り専用アクセスを許可します。あるサービスで新しい機能を立ち上げる場合は、AWS は、追加された演算とリソースに対し、読み込み専用の権限を追加します。ジョブ機能のポリシー一覧と説明については、IAM ユーザーガイドの AWS ジョブ機能の管理ポリシーを参照してください。

AWS マネージドポリシー: ComprehendFullAccess

このポリシーは、トピックモデリングジョブの実行を含む、Amazon Comprehend リソースへのフルアクセスを付与します。このポリシーでは、Amazon S3 バケットと IAM ロールのリスト権限と取得権限も付与されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "comprehend:*",
                "iam:GetRole",
                "iam:ListRoles",
                "s3:GetBucketLocation",
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
            ],
            "Resource": "*"
        }
    ]
}
```

AWS マネージドポリシー: ComprehendReadOnly

このポリシーは、以下を除くすべての Amazon Comprehend アクションを実行する読み取り専用アクセスを付与します。

- StartDominantLanguageDetectionJob
- StartEntitiesDetectionJob
- StartKeyPhrasesDetectionJob
- StartSentimentDetectionJob
- StartTargetedSentimentDetectionJob
- StartTopicsDetectionJob

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "comprehend:BatchDetectDominantLanguage",
                "comprehend:BatchDetectEntities",
                "comprehend:BatchDetectKeyPhrases",
                "comprehend:BatchDetectSentiment",
                "comprehend:BatchDetectSyntax",
                "comprehend:ClassifyDocument",
                "comprehend:ContainsPiiEntities",
                "comprehend:DescribeDocumentClassificationJob",
                "comprehend:DescribeDocumentClassifier",
                "comprehend:DescribeDominantLanguageDetectionJob",
                "comprehend:DescribeEndpoint",
                "comprehend:DescribeEntitiesDetectionJob",
                "comprehend:DescribeEntityRecognizer",
                "comprehend:DescribeKeyPhrasesDetectionJob",
                "comprehend:DescribePiiEntitiesDetectionJob",
                "comprehend:DescribeResourcePolicy",
                "comprehend:DescribeSentimentDetectionJob",
                "comprehend:DescribeTargetedSentimentDetectionJob",
                "comprehend:DescribeTopicsDetectionJob",
                "comprehend:DetectDominantLanguage",
                "comprehend:DetectEntities",
                "comprehend:DetectKeyPhrases",
                "comprehend:DetectPiiEntities",
                "comprehend:DetectSentiment",
                "comprehend:DetectSyntax",
                "comprehend:ListDocumentClassificationJobs",
                "comprehend:ListDocumentClassifiers",
                "comprehend:ListDocumentClassifierSummaries",
```

```
                "comprehend:ListDominantLanguageDetectionJobs",
                "comprehend:ListEndpoints",
                "comprehend:ListEntitiesDetectionJobs",
                "comprehend:ListEntityRecognizers",
                "comprehend:ListEntityRecognizerSummaries",
                "comprehend:ListKeyPhrasesDetectionJobs",
                "comprehend:ListPiiEntitiesDetectionJobs",
                "comprehend:ListSentimentDetectionJobs",
                "comprehend:ListTargetedSentimentDetectionJobs",
                "comprehend:ListTagsForResource",
                "comprehend:ListTopicsDetectionJobs"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Amazon Comprehend の AWS マネージドポリシーに対する更新

Amazon Comprehend の AWS マネージドポリシーに対する更新について、Amazon Comprehend がこれらの変更の追跡を開始してからの詳細を確認します。このページへの変更に関する自動アラートについては、Amazon Comprehend の「ドキュメント履歴」ページで RSS フィードにサブスクライブしてください。

変更	説明	日付
ComprehendReadOnly - 既存ポリシーへの更新	Amazon Comprehend が ComprehendReadOnly ポリシーで comprehend:DescribeTargetedSentimentDetectionJob および comprehend:ListTargetedSentimentDetectionJobs アクションを許可するようになりました	2022 年 3 月 30 日

変更	説明	日付
ComprehendReadOnly – 既存ポリシーへの更新	Amazon Comprehend が ComprehendReadOnly ポリシーで comprehend:DescribeResourcePolicy アクションを許可するようになりました	2022 年 2 月 2 日
ComprehendReadOnly – 既存ポリシーへの更新	Amazon Comprehend が ComprehendReadOnly ポリシーで ListDocumentClassifierSummaries および ListEntityRecognizerSummaries アクションを許可するようになりました	2021 年 9 月 21 日
ComprehendReadOnly – 既存ポリシーへの更新	Amazon Comprehend が ComprehendReadOnly ポリシーで ContainsPIIEntities アクションを許可するようになりました	2021 年 3 月 26 日
Amazon Comprehend が変更の追跡を開始しました	Amazon Comprehend は、その AWS マネージドポリシーの変更の追跡を開始しました。	2021 年 3 月 1 日

Amazon Comprehend のアイデンティティとアクセスのトラブルシューティング

以下の情報は、Amazon Comprehend と IAM の使用時に発生する可能性がある一般的な問題の診断と修正に役立ちます。

トピック

- Amazon Comprehend でアクションを実行する権限がない

- iam を実行する権限がありません。PassRole
- 自分の 以外のユーザーに Amazon Comprehend リソース AWS アカウント へのアクセスを許可したい

Amazon Comprehend でアクションを実行する権限がない

アクションを実行する権限がないというエラーが表示された場合は、そのアクションを実行できるようにポリシーを更新する必要があります。

以下のエラー例は、mateojackson IAM ユーザーがコンソールを使用して架空の *my-example-widget* リソースに関する詳細情報を表示しようとしているが、架空の comprehend:*GetWidget* 権限がないという場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 comprehend:GetWidget on resource: my-example-widget
```

この場合、Mateo のポリシーでは、*my-example-widget* アクションを使用して comprehend:*GetWidget* リソースにアクセスすることを許可するように更新する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

iam を実行する権限がありません。PassRole

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、Amazon Comprehend にロールを渡すことを許可するようにポリシーを更新する必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

次の例では、marymajor という名前の IAM ユーザーがコンソールを使用して Amazon Comprehend でアクションを実行しようとした際に、エラーが発生しています。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに `iam:PassRole` アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

自分の 以外のユーザーに Amazon Comprehend リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- Amazon Comprehend がこれらの機能をサポートしているかどうかについては、「Amazon Comprehend と IAM 連携の仕組み」を参照してください。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティー に提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイドの外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いについては、IAM ユーザーガイドの「IAM でのクロスアカウントリソースアクセス」を参照してください。

AWS CloudTrail での Amazon Comprehend API コールのログ記録

Amazon Comprehend は と統合されています。これはAWS CloudTrail、Amazon Comprehend の Amazon Comprehend . CloudTrail captures API コールでユーザー、ロール、または Amazon ComprehendAWSのサービスによって実行されたアクションをイベントとして記録するサービスです。キャプチャされるコールには、Amazon Comprehend コンソールからのコールと、Amazon Comprehend API オペレーションへのコードコールが含まれます。証跡を作成する場合は、Amazon Comprehend の CloudTrail イベントなど、Amazon S3 バケットへのイベントの継続的な配信を有

効にすることができます。 Amazon S3 Amazon Comprehend 証跡を設定しない場合でも、コンソールのイベント履歴 で最新の CloudTrail イベントを表示できます。 で収集された情報を使用して CloudTrail、Amazon Comprehend に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

設定および有効化の方法など CloudTrail、 の詳細については、AWS CloudTrail「 ユーザーガイド」を参照してください。

の Amazon Comprehend 情報 CloudTrail

CloudTrail アカウントを作成するAWS アカウントと、 は で有効になります。Amazon Comprehend でサポートされているイベントアクティビティが発生すると、そのアクティビティは CloudTrail イベント履歴 の他のAWSサービスイベントとともにイベントに記録されます。最近のイベントは、AWS アカウント で表示、検索、ダウンロードできます。詳細については、「イベント履歴 を使用した CloudTrail イベントの表示」を参照してください。

Amazon Comprehend のイベントなどの AWS アカウント アカウントにおけるイベントを継続的に記録するには、証跡を作成します。証跡により、 はログファイル CloudTrail を Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、すべての AWS リージョンに証跡が適用されます。追跡は、AWS パーティションのすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、 CloudTrail ログで収集されたデータをより詳細に分析し、それに基づく対応を行うように他の AWSサービスを設定できます。詳細については、次を参照してください:

- 「証跡作成の概要」
- CloudTrail でサポートされているサービスと統合
- の Amazon SNS 通知の設定 CloudTrail
- 複数のリージョンからの CloudTrail ログファイルの受信と複数のアカウントからの CloudTrail ログファイルの受信

Amazon Comprehend では、以下のアクションをイベントとして CloudTrail ログファイルに記録できます。

- BatchDetectDominantLanguage
- BatchDetectEntities
- BatchDetectKeyPhrases
- BatchDetectSentiment

- BatchDetectSyntax
- ClassifyDocument
- CreateDocumentClassifier
- CreateEndpoint
- CreateEntityRecognizer
- DeleteDocumentClassifier
- DeleteEndpoint
- DeleteEntityRecognizer
- DescribeDocumentClassificationJob
- DescribeDocumentClassifier
- DescribeDominantLanguageDetectionJob
- DescribeEndpoint
- DescribeEntitiesDetectionJob
- DescribeEntityRecognizer
- DescribeKeyPhrasesDetectionJob
- DescribePiiEntitiesDetectionJob
- DescribeSentimentDetectionJob
- DescribeTargetedSentimentDetectionJob
- DescribeTopicsDetectionJob
- DetectDominantLanguage
- DetectEntities
- DetectKeyPhrases
- DetectPiiEntities
- DetectSentiment
- DetectSyntax
- ListDocumentClassificationJobs
- ListDocumentClassifiers
- ListDominantLanguageDetectionJobs
- ListEndpoints
- ListEntitiesDetectionJobs

- ListEntityRecognizers
- ListKeyPhrasesDetectionJobs
- ListPiiEntitiesDetectionJobs
- ListSentimentDetectionJobs
- ListTargetedSentimentDetectionJobs
- ListTagsForResource
- ListTopicsDetectionJobs
- StartDocumentClassificationJob
- StartDominantLanguageDetectionJob
- StartEntitiesDetectionJob
- StartKeyPhrasesDetectionJob
- StartPiiEntitiesDetectionJob
- StartSentimentDetectionJob
- StartTargetedSentimentDetectionJob
- StartTopicsDetectionJob
- StopDominantLanguageDetectionJob
- StopEntitiesDetectionJob
- StopKeyPhrasesDetectionJob
- StopPiiEntitiesDetectionJob
- StopSentimentDetectionJob
- StopTargetedSentimentDetectionJob
- StopTrainingDocumentClassifier
- StopTrainingEntityRecognizer
- TagResource
- UntagResource
- UpdateEndpoint

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するために役立ちます。

- リクエストが、ルートユーザーの認証情報で行われたかどうか。

- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが、別の AWS サービスによって送信されたかどうか。

詳細については、「CloudTrail userIdentity 要素」を参照してください。

例: Amazon Comprehend ログファイルのエントリ

証跡は、指定した Amazon S3 バケットにイベントをログファイルとして配信できるようにする設定です。 CloudTrail ログファイルには、1 つ以上のログエントリが含まれます。イベントは任意の送信元からの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。 CloudTrail ログファイルは、パブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次の例は、 `ClassifyDocument`アクションを示す CloudTrail ログエントリを示しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AROAICFHPEXAMPLE",
        "arn": "arn:aws:iam::12345678910:user/myadmin2",
        "accountId": "12345678910",
        "accessKeyId": "ASIA3VZEXAMPLE",
        "sessionContext": {
            "sessionIssuer": {},
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-10-19T14:22:09Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2023-10-19T17:31:20Z",
    "eventSource": "comprehend.amazonaws.com",
    "eventName": "ClassifyDocument",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "3.21.185.237",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0)
 Gecko/20100101 Firefox/115.0",
    "requestParameters": null,
```

```
    "responseElements": null,
    "requestID": "fd916e66-caac-46c9-a1fc-81a0ef33e61b",
    "eventID": "535ca22b-b3a3-4c13-b2c5-bf51ab082794",
    "readOnly": false,
    "resources": [
        {
            "accountId": "12345678910",
            "type": "AWS::Comprehend::DocumentClassifierEndpoint",
            "ARN": "arn:aws:comprehend:us-east-2:12345678910:document-classifier-
endpoint/endpointExample"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "12345678910"
}
```

Amazon Comprehend のコンプライアンス検証

サードパーティーの監査人は、さまざまな AWS コンプライアンスプログラムの一環として Amazon Comprehend のセキュリティとコンプライアンスを評価します。このプログラムには、PCI、FedRAMP、HIPAA などがあります。AWS Artifact を使用して、サードパーティーの監査レポートをダウンロードできます。詳細については、「Downloading reports in AWS Artifact」を参照してください。

Amazon Comprehend の使用時におけるユーザーのコンプライアンス責任は、データの機密性、会社のコンプライアンス目的、および適用される法律と規制に応じて異なります。AWS は、コンプライアンスに役立つ以下のリソースを提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドには、アーキテクチャ面の考慮事項に関する説明とともに、セキュリティとコンプライアンスに焦点を当てたベースライン環境を AWS にデプロイするためのステップが記載されています。
- Architecting for HIPAA Security and Compliance Whitepaper (HIPAA のセキュリティとコンプライアンスのためのアーキテクチャの設計に関するホワイトペーパー) - このホワイトペーパーは、企業が AWS を使用して HIPAA 準拠のアプリケーションを作成する方法について説明します。
- AWS コンプライアンスのリソース - このワークブックとガイドのコレクションは、お客様の業界や所在地に適用される場合があります。
- AWS Config - この AWS のサービスでは、自社プラクティス、業界ガイドライン、および規制に対するリソースの設定の準拠状態を評価します。

- AWS Security Hub: この AWS のサービスでは、AWS 内のセキュリティ状態を包括的に表示しており、セキュリティ業界の標準およびベストプラクティスへの準拠を確認するのに役立ちます。

特定のコンプライアンスプログラムの対象範囲に含まれる AWS のサービスのリストについては、「コンプライアンスプログラムによる対象範囲内の AWS のサービス」を参照してください。一般的な情報については、「AWS コンプライアンスプログラム」を参照してください。

Amazon Comprehend における耐障害性

AWS グローバルインフラストラクチャは AWS リージョン およびアベイラビリティーゾーンを中心に構築されています。AWS リージョン には、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立・隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、および拡張性が優れています。

AWS リージョン およびアベイラビリティーゾーンの詳細については、「AWS グローバルインフラストラクチャ」を参照してください。

Amazon Comprehend におけるインフラストラクチャセキュリティ

マネージドサービスである Amazon Comprehend は AWS グローバルネットワークセキュリティによって保護されています。AWS のセキュリティサービスと、AWS がインフラストラクチャをどのように保護するかについては、「AWS クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを利用して AWS 環境を設計するには、「セキュリティの柱 AWS Well-Architected Framework」の「インフラストラクチャ保護」を参照してください。

ネットワーク経由で Amazon Comprehend にアクセスするには、AWS がパブリッシュした API コールを使用します。クライアントは以下をサポートする必要があります:

- Transport Layer Security (TLS)。TLS 1.2 が必須です。TLS 1.3 が推奨されます。
- DHE (Ephemeral Diffie-Hellman) や ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) などの Perfect Forward Secrecy (PFS) を使用した暗号スイート。これらのモードは、Java 7 以降など、ほとんどの最新システムでサポートされています。

また、リクエストには、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token ServiceAWS STS を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

ガイドラインとクォータ

別に指定されていない限り、Amazon Comprehend のクォータは地域ごとに設定されます。アプリケーションに必要な場合は、調整可能なクォータの引き上げをリクエストできます。クォータの詳細およびクォータ増加リクエストについては、「AWS Service Quotas」を参照してください。

トピック

- サポートされるリージョン
- 組み込みモデルのクォータ
- カスタムモデルのクォータ
- フライホイールのクォータ

サポートされるリージョン

Amazon Comprehend AWS は以下のリージョンでご利用いただけます。

- 米国東部 (オハイオ)
- 米国東部 (バージニア北部)
- 米国西部 (オレゴン)
- アジアパシフィック (ムンバイ)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- カナダ (中部)
- 欧州 (フランクフルト)
- 欧州 (アイルランド)
- 欧州 (ロンドン)
- AWS GovCloud (米国西部)

デフォルトで、Amazon Comprehend はサポートされている各地域ですべての API オペレーションを提供します。例外については、「ドキュメント処理」を参照してください。

API エンドポイントの詳細については、「Amazon Web Services 全般リファレンス」の「Amazon Comprehend リージョンとエンドポイント」を参照してください。

地域の現在のクォータを確認したり、調整可能なクォータの引き上げをリクエストするには、「Service Quotas コンソール」を開いてください。

組み込みモデルのクォータ

Amazon Comprehend には、UTF-8 テキストドキュメントを分析するための組み込みモデルが用意されています。Amazon Comprehend には、組み込みモデルを使用する同期操作と非同期操作が用意されています。

トピック

- リアルタイム (同期) 分析
- 非同期分析

リアルタイム (同期) 分析

このセクションでは、組み込みモデルを使用したリアルタイム分析に関するクォータについて説明します。

トピック

- 単一ドキュメント操作
- 複数のドキュメント操作
- リアルタイム (同期) リクエストのリクエストスロットリング

単一ドキュメント操作

Amazon Comprehend API には、1 つのドキュメントを入力して受け取るオペレーションが用意されています。これらのオペレーションには、以下のクォータが適用されます。

単一ドキュメント操作の一般的なクォータ

エンティティ、キーフレーズ、または主要言語を検出するためのリアルタイム分析には、以下のクォータが適用されます。エンティティ検出の場合、これらのクォータは組み込みモデルによる検出に適用されます。カスタムエンティティ検出については、「カスタムエンティティ認識」のクォータを参照してください。

説明	クォータ/ガイドライン
最大ドキュメントサイズ	100 KB

1 つのドキュメント操作に対する操作固有のクォータ

センチメント、ターゲット感情、構文を検出するためのリアルタイム分析には、以下のクォータが適用されます。

説明	クォータ/ガイドライン
最大ドキュメントサイズ	5 KB

複数のドキュメント操作

Amazon Comprehend API は、1 回の API リクエストで複数のドキュメントを処理するバッチオペレーションを提供します。　バッチオペレーションには、以下のクォータが適用されます。

説明	クォータ/ガイドライン
最大ドキュメントサイズ	5 KB
1 回のリクエストあたりの最大ドキュメント数	25

バッチドキュメントオペレーションの詳細については、複数ドキュメントの同期処理 を参照してください。

リアルタイム (同期) リクエストのリクエストスロットリング

Amazon Comprehend は同期リクエストに動的スロットリングを適用します。　システム処理帯域幅が利用可能な場合、Amazon Comprehend は処理するリクエストの数を徐々に増やします。アプリケーションによる同期 API オペレーションの使用を制御するには、請求アラートをオンにするか、アプリケーションにレート制限を実装することをお勧めします。

非同期分析

このセクションでは、組み込みモデルを使用した非同期分析に関連するクォータについて説明します。

非同期 API オペレーションは、それぞれ最大 10 個のアクティブジョブをサポートします。 各 API オペレーションのクォータを確認するには、「Amazon Web Services 全般リファレンス」の「Amazon Comprehend エンドポイントとクオータ」の「Service Quotas」テーブルを参照してください。

調整可能なクォータの場合、Service Quotas コンソールを使用してクォータの引き上げをリクエストできます。

トピック

- 非同期オペレーションの一般的なクォータ
- 非同期ジョブの操作固有クォータ
- 非同期リクエストのリクエストスロットリング

非同期オペレーションの一般的なクォータ

非同期分析ジョブは、コンソールまたは任意の API `Start*` オペレーションを使用して実行できます。非同期オペレーションの使用タイミングについては、「非同期バッチ処理」を参照してください。以下のクォータは、組み込みモデルのほとんどの API `Start*` オペレーションに適用されます。例外については、「非同期ジョブの操作固有クォータ 」を参照してください。

説明	クォータ/ガイドライン
エンティティ、キーフレーズ、PII、言語を検出するジョブ内の各ドキュメントの最大サイズ	1 MB
リクエスト内のすべてのファイルの最大合計サイズ	5 GB
リクエスト内のすべてのファイルの最小合計サイズ	500 バイト
最大ファイル数 (1 ファイルあたり 1 ドキュメント)	1,000,000
最大合計行数 (1 行につき 1 ドキュメント)	1,000,000

非同期ジョブの操作固有クォータ

このセクションでは、特定の非同期操作のクォータについて説明します。　以下の表にクォータが指定されていない場合は、一般的なクォータ値が適用されます。

トピック

- 感情
- ターゲット感情
- イベント
- トピックのモデリング

感情

このオペレーションで作成する非同期センチメントジョブには、以下のクォータがあります。StartSentimentDetectionJob

説明	クォータ/ガイドライン
各入力ドキュメントの最大サイズ	5 KB

ターゲット感情

オペレーションで作成する非同期ターゲットセンチメントジョブには、以下のクォータがあります。StartTargetedSentimentDetectionJob

説明	クォータ/ガイドライン
サポートされるドキュメント形式	UTF-8
1 つのジョブ内の各入力ドキュメントの最大サイズ	10 KB
1 つのジョブ内の全ドキュメントの最大サイズ	300 MB
最大ファイル数 (1 ファイルあたり 1 ドキュメント)	30,000
最大合計行数、1 行につき 1 つのドキュメント (1 件のリクエストに含まれる全ファイル)	30,000

イベント

このオペレーションで作成する非同期イベント検出ジョブには、以下のクォータがあります。StartEventsDetectionJob

説明	クォータ
文字エンコーディング	UTF-8
1 つのジョブ内のすべてのファイルの合計サイズ	50 MB
1 つのジョブ内の各入力ドキュメントの最大サイズ	10 KB
最大ファイル数 (1 ファイルあたり 1 ドキュメント)	5,000
最大合計行数、1 行につき 1 つのドキュメント (リクエストに含まれる全ファイル)	5,000

トピックのモデリング

StartTopicsDetectionJobオペレーションで作成する非同期トピックモデリングジョブには、以下のクォータがあります。

説明	クォータ/ガイドライン
文字エンコーディング	UTF-8
返されるトピックの最大数	100
1 ファイルの最大ファイルサイズ、1 ファイルにつき 1 ドキュメント	100 MB

詳細については、「トピックのモデリング」を参照してください。

非同期リクエストのリクエストスロットリング

各非同期 API オペレーションは、1 秒あたりの最大リクエスト数 (リージョンごと、アカウントごと) と、最大 10 件のアクティブジョブをサポートします。 各 API オペレーションのクォータを確認

するには、「Amazon Web Services 全般リファレンス」の「Amazon Comprehend エンドポイントとクオータ」の「Service Quotas」テーブルを参照してください。

調整可能なクォータの場合、Service Quotas コンソールを使用してクォータの引き上げをリクエストできます。

カスタムモデルのクォータ

Amazon Comprehend では、カスタム分類とカスタムエンティティ認識のための独自のカスタムモデルを構築することができます。このセクションでは、カスタムモデルのトレーニングと使用に関するガイドラインとクォータについて説明します。　カスタムモデルの詳細については、「Amazon Comprehend Custom」を参照してください。

トピック

- 一般的なクォータ
- エンドポイントのクォータ
- ドキュメント分類
- カスタムエンティティ認識

一般的なクォータ

Amazon Comprehend は、カスタムモデルで分析できる入力ドキュメントの種類ごとに一般的なサイズ割り当てを設定します。リアルタイム分析のクォータについては、「リアルタイム分析用の最大ドキュメントサイズ 」 を参照してください。非同期分析クォータについては、「非同期カスタム分析の入力 」 を参照してください。

各非同期 API オペレーションは、1 秒あたりの最大リクエスト数 (リージョンごと、アカウントごと) と、最大 10 件のアクティブジョブをサポートします。　各 API オペレーションのクォータを確認するには、「Amazon Web Services 全般リファレンス」の「Amazon Comprehend エンドポイントとクオータ」の「Service Quotas」テーブルを参照してください。

調整可能なクォータの場合、Service Quotas コンソールを使用してクォータの引き上げをリクエストできます。

エンドポイントのクォータ

エンドポイントを作成して、カスタムモデルでリアルタイム分析を実行します。　エンドポイントの詳細については、Amazon Comprehend のエンドポイントの管理 を参照してください。

次のクォータがエンドポイントに適用されます。クォータ増加リクエストの詳細については、「AWS Service Quotas」を参照してください。

説明	クォータ/ガイドライン
各アカウントのリージョンあたりのアクティブなエンドポイントの最大数	20
各アカウントのリージョンあたりの推論単位の最大数	200
1 リージョン 1 エンドポイントごとの最大推論単位数	50
推論単位あたりの最大スループット (文字)	100/秒
推論単位あたりの最大スループット (ドキュメント)	2/秒

ドキュメント分類

このセクションでは、以下のドキュメント分類操作のガイドラインとクォータについて説明します。

- オペレーションから開始する分類器トレーニングジョブ。CreateDocumentClassifier
- 操作から開始する非同期ドキュメント分類ジョブ。StartDocumentClassificationJob
- オペレーションを使用する同期ドキュメント分類リクエスト。ClassifyDocument

ドキュメント分類の一般的なクォータ

次の表では、カスタム分類子のトレーニングに関連する一般的なクォータについて説明しています。

説明	クォータ/ガイドライン
クラス名の最大長	5,000 文字
クラス数 (複数クラスモード)	2 ~ 1,000
クラス数 (複数ラベルモード)	2 ~ 100
注釈の形式	

説明	クォータ/ガイドライン
クラスあたりの注釈の最小数 (複数クラスモード)	10
クラスあたりの注釈の最小数 (複数ラベルモード)	10
注釈の最小数 (複数ラベルモード)	50
CSV ファイル形式	
クラスあたりのトレーニングドキュメントの最小数 (複数クラスモード)	50
クラスあたりのトレーニングドキュメントの最小数 (複数ラベルモード)	10
トレーニングドキュメントの最小数 (複数ラベルモード)	50

プレーンテキストドキュメントの分類

プレーンテキスト入力ドキュメントを使用してプレーンテキストモデルを作成し、トレーニングします。Amazon Comprehend には、プレーンテキストモデルを使用してプレーンテキストドキュメントを分類するためのリアルタイム操作と非同期操作が用意されています。

トレーニング

次の表では、プレーンテキストドキュメントを使用したカスタム分類子のトレーニングに関連するクォータについて説明しています。

説明	クォータ/ガイドライン
1 つのトレーニングジョブ内の全ファイルの合計サイズ	5 GB
カスタム分類子をトレーニングするための拡張マニフェストファイルの最大数	5
各拡張マニフェストファイルの属性名の最大数	5
属性名の最大長	63 文字

リアルタイム (同期) 分析

次の表では、プレーンテキストドキュメントのリアルタイム分類に関連するクォータについて説明しています。

説明	クォータ/ガイドライン
同期リクエスト 1 回あたりの最大ドキュメント数	1
最大テキストドキュメントサイズ (UTF-8 エンコード)	10 KB

非同期分析

次の表では、プレーンテキストドキュメントの非同期分類に関連するクォータについて説明しています。

説明	クォータ/ガイドライン
1 つの非同期ジョブ内の全ファイルの合計サイズ	5 GB
1 ファイルの最大ファイルサイズ、1 ファイルにつき 1 ドキュメント	10 MB
最大ファイル数 (1 ファイルあたり 1 ドキュメント)	1,000,000
最大合計行数、1 行につき 1 つのドキュメント (リクエストに含まれる全ファイル)	1,000,000

半構造化ドキュメントの分類

このセクションでは、半構造化ドキュメントのドキュメント分類に関するガイドラインとクォータについて説明します。半構造化ドキュメントを分類するには、ネイティブ入力ドキュメントでトレーニングしたネイティブドキュメントモデルを使用してください。

半構造化ドキュメントによるネイティブドキュメントモデルのトレーニング

次の表は、PDF ドキュメント、Word ドキュメント、画像ファイルなどの半構造化ドキュメントによるカスタム分類子のトレーニングに関連するクォータを示しています。

説明	クォータ/ガイドライン
全ドキュメントの最大ページ数	10,000
注釈ファイルの最大サイズ (全 CSV ファイルサイズを合わせたサイズ)	5 MB
ドキュメントコーパスサイズ (トレーニングドキュメントとテストドキュメント)	10 GB
トレーニングファイルとテストファイルのファイルサイズ	
画像ファイルサイズ (JPG、PNG、TIFF)。	1 バイト ~ 10 MB。 TIFF ファイル:最大 1 ページ。
PDF ドキュメントのページサイズ	1 バイト ~ 10 MB
Word ドキュメントのページサイズ	1 バイト ~ 10 MB
Amazon Textract API 出力 JSON サイズ	1 バイト ~ 1 MB

リアルタイム (同期) 分析

このセクションでは、半構造化ドキュメントのリアルタイム分類に関するクォータについて説明します。

次の表は、入力ドキュメントの最大ファイルサイズを示しています。すべての入力ドキュメントタイプで、入力ファイルの最大数は 1 ページで、10,000 文字以下です。

ファイルタイプ	最大サイズ (API)	最大サイズ (コンソール)
A UTF-8 テキストドキュメント	10 KB	10 KB
PDF ドキュメント	10 MB	5 MB
Word ドキュメント	10 MB	5 MB

ファイルタイプ	最大サイズ (API)	最大サイズ (コンソール)
画像ファイル	10 MB	5 MB
Amazon Textract API 出力サイズ	1 MB	該当なし

非同期分析

次の表では、半構造化ドキュメントの非同期分類に関連するクォータについて説明しています。

説明	クォータ/ガイドライン
ジョブのすべての入力ドキュメントの最大ページ数	25,000
ドキュメントコーパスサイズ	25 GB
画像ファイルサイズ (JPG、PNG、TIFF)	1 バイト ~ 10 MB。 TIFF ファイル:最大 1 ページ。
PDF ドキュメントのページサイズ	1 バイト ~ 10 MB
Word ドキュメントのページサイズ	1 バイト ~ 10 MB
Textract API 出力 JSON サイズ	1 バイト ~ 1 MB。

カスタムエンティティ認識

このセクションでは、カスタムエンティティレコグナイザーの以下の操作に関するガイドラインとクォータについて説明します。

- CreateEntityRecognizerエンティティ認識トレーニングジョブは操作から開始されました。
- 非同期エンティティ認識ジョブは操作から開始されました。StartEntitiesDetectionJob
- オペレーションを使用した同期エンティティ認識リクエスト。DetectEntities

プレーンテキストドキュメントのカスタムエンティティレコグナイザー

Amazon Comprehend には、カスタムエンティティレコグナイザーを使用してプレーンテキストドキュメントを分析するための非同期操作と同期操作が用意されています。

トレーニング

このセクションでは、プレーンテキストドキュメントを分析するためのカスタムエンティティレコグナイザーのトレーニングに関連するクォータについて説明します。モデルをトレーニングするには、エンティティリストまたは注釈付きテキストドキュメントのセットを提供できます。

次の表では、エンティティリストを使用したモデルのトレーニングに関連するクォータについて説明しています。

説明	クォータ/ガイドライン
1 モデルあたりのエンティティ数	1 ~ 25
ドキュメントサイズ (UTF-8)	1 ~ 5,000 バイト
エンティティリスト内の項目数	1 ~ 100 万
エントリリスト内の個々のエントリ (ポストストリップ) の長さ	1 ~ 5,000
エンティティリストのコーパスサイズ (プレーンテキストの全ドキュメントを含む)	5 KB ~ 200 MB

次の表では、注釈付きテキストドキュメント使用したモデルのトレーニングに関連するクォータについて説明しています。

説明	クォータ/ガイドライン
モデル/カスタムエンティティレコグナイザーあたりのエンティティ数	1 ~ 25
ドキュメントサイズ (UTF-8)	1 ~ 5,000 バイト
ドキュメント数 (「プレーンテキスト注釈」 を参照)	3 ~ 200,000

説明	クォータ/ガイドライン
ドキュメントコーパスサイズ (プレーンテキストの全ドキュメントを含む)	5 KB ~ 200 MB
エンティティごとの最小注釈数	25

リアルタイム (同期) 分析

次の表では、プレーンテキストドキュメントのリアルタイム分析に関連するクォータについて説明しています。

説明	クォータ/ガイドライン
同期リクエスト 1 回あたりの最大ドキュメント数	1
最大テキストドキュメントサイズ (UTF-8 エンコード)	5 KB

非同期分析

次の表では、プレーンテキストドキュメントの非同期エンティティレコグナイザーに関連するクォータについて説明しています。

説明	クォータ/ガイドライン
ドキュメントサイズ (UTF-8)	1 バイト ~ 1 MB
最大ファイル数 (1 ファイルあたり 1 ドキュメント)	1,000,000
最大合計行数、1 行につき 1 つのドキュメント (リクエストに含まれる全ファイル)	1,000,000
ドキュメントコーパスサイズ (プレーンテキストの全ドキュメントを含む)	1 バイト ~ 5 GB

半構造化ドキュメントのカスタムエンティティレコグナイザー

Amazon Comprehend には、カスタムエンティティレコグナイザーを使用して半構造化ドキュメントを分析するための非同期操作と同期操作が用意されています。注釈付き PDF ドキュメントを使用してモデルをトレーニングする必要があります。

トレーニング

次の表は、半構造化文書を分析するためのカスタム・エンティティ・レコグナイザー (CreateEntityRecognizer) のトレーニングに関連するクォータを示しています。

説明	クォータ/ガイドライン
モデル/カスタムエンティティレコグナイザーあたりのエンティティ数	1 ~ 25
最大注釈ファイルサイズ (UTF-8 JSON)	5 MB
ドキュメント数	250 ~ 10,000
ドキュメントコーパスサイズ (プレーンテキストの全ドキュメントを含む)	5 KB ~ 1 GB
エンティティごとの最小注釈数	100
カスタムエンティティレコグナイザーをトレーニングするための拡張マニフェストファイルの最大数	5
各拡張マニフェストファイルの属性名の最大数	5
属性名の最大長	63 文字

リアルタイム (同期) 分析

このセクションでは、半構造化ドキュメントのリアルタイム分析に関するクォータについて説明します。

次の表は、入力ドキュメントの最大ファイルサイズを示しています。すべての入力ドキュメントタイプで、入力ファイルの最大数は 1 ページで、10,000 文字以下です。

ファイルタイプ	最大サイズ (API)	最大サイズ (コンソール)
A UTF-8 テキストドキュメント	10 KB	10 KB
PDF ドキュメント	10 MB	5 MB
Word ドキュメント	10 MB	5 MB
画像ファイル	10 MB	5 MB
Textract 出力ファイル	1 MB	該当なし

非同期分析

このセクションでは、半構造化ドキュメントの非同期分析のクォータについて説明します。

説明	クォータ/ガイドライン
画像サイズ (JPG または PNG)	1 バイト ~ 10 MB
画像サイズ (TIFF)	1 バイト ~ 10 MB。最大 1 ページ。
ドキュメントサイズ (PDF)	1 バイト ~ 50 MB
ドキュメントサイズ (Docx)	1 バイト ~ 5 MB
ドキュメントサイズ (UTF-8)	1 バイト ~ 1 MB
最大ファイル数、1 ファイルあたり 1 ドキュメント (画像ファイルや PDF/Word ドキュメントでは 1 行に 1 つのドキュメントは使用できません)	500
PDF または Docx ファイルの最大ページ数	100
テキスト抽出後のドキュメントコーパスサイズ (プレーンテキスト、すべてのファイルを含む)	1 バイト ~ 5 GB

画像の制限の詳細については、「Amazon Textract のハードリミット」を参照してください。

フライホイールのクォータ

フライホイールを使用して、カスタム分類とカスタムエンティティレコグナイザーのためのカスタムモデルバージョンのトレーニングと追跡を管理します。フライホイールの詳細については、「フライホイール」を参照してください。

フライホイールの一般的なクォータ

フライホイールとフライホイールイテレーションには以下のクォータが適用されます。

説明	クォータ/ガイドライン
フライホイールの最大数	50
「CREATING」(作成中) 状態のフライホイールの最大数	10
フライホイールあたりのトレーニングデータセットの最大数	50
フライホイールあたりのテストデータセットの最大数	50
「インジェスト中」 状態のデータセットの最大数	10
アカウントごとの進行中のフライホイールイテレーションの最大数	10

カスタム分類モデルのデータセットクォータ

カスタム分類モデルに関連するフライホイールにデータセットを取り込むと、次のクォータが適用されます。

説明	クォータ/ガイドライン
クラスあたりのトレーニングドキュメントの最小数 (複数ラベルモード)	50
トレーニングドキュメントの最大数	1,000,000

説明	クォータ/ガイドライン
最小データセットサイズ	500 バイト
最大データセットサイズ	5 GB
1 ファイルの最大ファイルサイズ、1 ファイルにつき 1 ドキュメント	10 MB

カスタムエンティティレコグナイザーモデルのデータセットクォータ

カスタムエンティティレコグナイザーモデルに関連するフライホイールのデータセットを取り込むと、次のクォータが適用されます。

説明	クォータ/ガイドライン
最大ドキュメントサイズ	5 KB
トレーニングドキュメントの最小数	3
トレーニングドキュメントの最大数	200,000 件の
エンティティごとの最小注釈数	25
最大データセットサイズ	200 MB

チュートリアルとその他のリソース

Amazon Comprehend のチュートリアルとその他のリソース

トピック

- チュートリアル:Amazon Comprehend を使用してカスタマーレビューからインサイトを分析する
- 個人を特定できる情報 (PII) のための Amazon S3 Object Lambda アクセスポイントの使用
- 解決策: Amazon Comprehend と を使用したテキストの分析 OpenSearch

チュートリアル:Amazon Comprehend を使用してカスタマーレビューからインサイトを分析する

このチュートリアルでは、Amazon Comprehend と Amazon Simple Storage Service 、AWS GlueAmazon Athena、、および Amazon QuickSight を使用して、ドキュメントに関する貴重なインサイトを得る方法について説明します。Amazon Comprehend では、非構造化テキストか感情 (ドキュメントの雰囲気) とエンティティ (人、組織、イベント、日付、製品、場所、数量、タイトルの名前) を抽出できます。

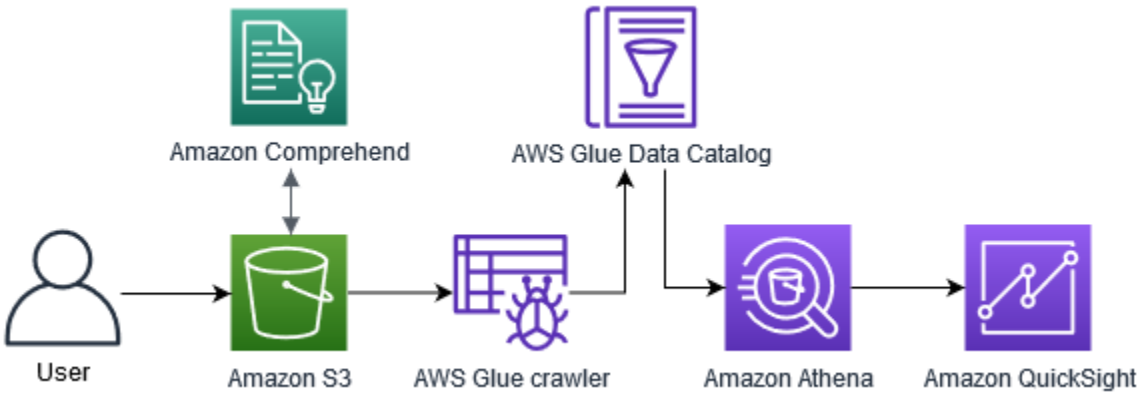
例えば、カスタマーレビューから実用的なインサイトを得ることができます。このチュートリアルでは、ある小説に関するカスタマーレビューのサンプルデータセットを分析します。Amazon Comprehend の感情分析を使用して、顧客が小説に対して肯定的か否定的かを判断します。また、Amazon Comprehend のエンティティ分析を使用して、関連する小説や著者などの重要なエンティティについての言及を見つけます。このチュートリアルに従うと、レビューの 50% 以上が肯定的であることが分かります。また、顧客が著者を比較したり、他の古典小説に興味を示したりしていることにも気づきます。

このチュートリアルでは、以下を実行しました。

- レビューのサンプルデータセットを Amazon Simple Storage Service (Amazon S3) に保存します。Amazon Simple Storage Service は、オブジェクトストレージサービスです。
- Amazon Comprehend を使用して、レビュードキュメントに含まれる感情とエンティティを分析します。
- AWS Glue クローラーを使用して分析結果をデータベースに保存します。AWS Glue は、抽出、変換、ロード (ETL) サービスであり、分析のためにデータを分類し、クレンジングすることができます。

- Amazon Athena クエリを実行してデータをクレンジングします。Amazon Athena はサーバーレスのインタラクティブクエリサービスです。
- Amazon QuickSight のデータを使用してビジュアライゼーションを作成します。Amazon QuickSight は、データからインサイトを抽出するためのサーバーレスビジネスインテリジェンスツールです。

以下の図に、ワークフローを示しています。



このチュートリアルを完了する予定時間: 1 時間

推定コスト: このチュートリアルの一部のアクションによって、AWS アカウント で支払いが発生する場合があります。これらの各サービスの料金については、次の料金ページを参照してください。

- Amazon S3 の料金
- Amazon Comprehend の料金
- AWS Glue 料金表
- Amazon Athena 料金表
- Amazon QuickSight の料金

トピック

- 前提条件
- ステップ 1: Amazon S3 にドキュメントを追加する
- ステップ 2: (CLI のみ) Amazon Comprehend 用の IAM ロールを作成する
- ステップ 3: Amazon S3 上のドキュメントに対する分析ジョブの実行
- ステップ 4: データ可視化用に Amazon Comprehend 出力を準備する

- ステップ 5: Amazon で Amazon Comprehend 出力を視覚化する QuickSight

前提条件

このチュートリアルを完了するには、以下が必要です。

- AWS アカウント。AWS アカウント の設定に関する詳細は、セットアップ を参照してください。
- (IAM) エンティティ (ユーザー、グループまたはロール)。アカウントのユーザーとグループをセットアップする方法については、「IAM ユーザーガイド」の「開始方法」セクションを参照してください。
- 以下の許可ポリシーをユーザー、グループまたはロールにアタッチします。ポリシーは、このチュートリアルを完了するために必要な許可の一部を付与します。次の前提条件では、必要な追加許可について説明します。

```
{
  "Version": "2012-10-17",
  "Statement":
  [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action":
      [
        "comprehend:*",
        "ds:AuthorizeApplication",
        "ds:CheckAlias",
        "ds:CreateAlias",
        "ds:CreateIdentityPoolDirectory",
        "ds:DeleteDirectory",
        "ds:DescribeDirectories",
        "ds:DescribeTrusts",
        "ds:UnauthorizeApplication",
        "iam:AttachRolePolicy",
        "iam:CreatePolicy",
        "iam:CreatePolicyVersion",
        "iam:CreateRole",
        "iam:DeletePolicyVersion",
        "iam:DeleteRole",
        "iam:DetachRolePolicy",
        "iam:GetPolicy",
        "iam:GetPolicyVersion",
```

```
        "iam:GetRole",
        "iam:ListAccountAliases",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy",
        "iam:ListPolicies",
        "iam:ListPolicyVersions",
        "iam:ListRoles",
        "quicksight:*",
        "s3:*",
        "tag:GetResources"
      ],
      "Resource": "*"
    },
    {
      "Action":
      [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource":
      [
        "arn:aws:iam::*:role/*Comprehend*"
      ]
    }
  ]
}
```

前述のポリシーを使用して IAM ポリシーを作成し、グループまたはユーザーに追加します。IAM ポリシーの作成については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。IAM ポリシーのアタッチに関する詳細については、「IAM ユーザーガイド」の「IAM ID のアクセス許可の追加および削除」を参照してください。

- IAM グループまたはユーザーにアタッチされた管理ポリシー。前述のポリシーに加えて、次の AWS 管理ポリシーをグループまたはユーザーに追加する必要があります。
  - `AWSGlueConsoleFullAccess`
  - `AWSQuicksightAthenaAccess`

  これらの管理ポリシーは、AWS Glue、、Amazon Athenaおよび Amazon を使用するアクセス許可を付与します QuickSight。IAM ポリシーのアタッチに関する詳細については、「IAM ユーザーガイド」の「IAM ID のアクセス許可の追加および削除」を参照してください。

ステップ 1: Amazon S3 にドキュメントを追加する

Amazon Comprehend 分析ジョブを開始するには、カスタマーレビューのサンプルデータセットを Amazon Simple Storage Service (Amazon S3) に保存しておく必要があります。Amazon S3 は、バケットと呼ばれるコンテナにデータを保存します。Amazon Comprehend は、バケットに保存されているドキュメントを分析し、その分析結果をバケットに送信します。このステップでは、S3 バケットを作成して、バケットに入出力フォルダを作成し、バケットにサンプルデータセットをアップロードします。

トピック

- 前提条件
- サンプルデータをダウンロードする
- Amazon S3 バケットを作成する
- フォルダーを作成する (コンソールのみ)
- 入力データをアップロードする

前提条件

この手順を開始するにあたっては、チュートリアル:Amazon Comprehend を使用してカスタマーレビューからインサイトを分析する を確認して前提条件を完了しておいてください。

サンプルデータをダウンロードする

次のサンプルデータセットには、より大きなデータセット「Amazon reviews-Full」から取得した Amazon レビューが含まれています。このデータセットは、「Character-level Convolutional Networks for Text Classification」(Xiang Zhang その他、2015 年) という記事と共に公開されたものです。データセットをコンピュータにダウンロードします。

サンプルデータを取得する

1. zip ファイル tutorial-reviews-data.zip をコンピュータにダウンロードします。
2. コンピューター上の zip ファイルを解凍します。2 つのファイルがあります。ファイル `THIRD_PARTY_LICENSES.txt` は Xiang Zhang その他が公開したデータセットのオープンソースライセンスです。ファイル `amazon-reviews.csv` は、チュートリアルで分析するデータセットです。

Amazon S3 バケットを作成する

サンプルデータセットをダウンロードしたら、入出力データを保存するための Amazon S3 バケットを作成します。S3 バケットは、Amazon S3 コンソールまたは AWS Command Line Interface (AWS CLI) を使用して作成できます。

Amazon S3 バケットを作成する (コンソール)

Amazon S3 コンソールで、すべて AWSにおいて一意の名前でバケットを作成します。

S3 バケットを作成する (コンソール)

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/s3/ で Amazon S3 コンソールを開きます。
2. [Buckets] (バケット) で、[Create bucket] (バケットの作成) を選択します。
3. [バケット名] では、バケットの目的を説明するグローバルに一意の名前を入力します。
4. リージョン で、バケットを作成する AWS リージョンを選択します。選択するリージョンは Amazon Comprehend に対応している必要があります。レイテンシーを減らすには、Amazon Comprehend でサポートされている地理的な場所に最も近い AWS リージョンを選択します。Amazon Comprehend に対応しているリージョンについては、『グローバル・インフラストラクチャー・ガイド』の 「リージョン表」 を参照してください。
5. [Object Ownership]、[Bucket settings for Block Public Access]、 [Bucket Versioning] および [Tags] にデフォルトの設定を使用します。
6. [Default encryption] (デフォルトの暗号化) には、[Disable] (無効) を選択します。

> **Tip**
>
> このチュートリアルでは暗号化を使用しませんが、重要なデータを分析する場合は暗号化を使用することもできます。 end-to-end 暗号化では、バケット内の保管中のデータを暗号化したり、分析ジョブを実行したりすることもできます。による暗号化の詳細については AWS、「 AWS Key Management Service デベロッパーガイド」の「 とは AWS Key Management Service」を参照してください。

7. バケットの設定を確認して、[バケットの作成]を選択します。

Amazon S3 バケットを作成する (AWS CLI)

を開いた後 AWS CLI、 `create-bucket` コマンドを実行して、入力データと出力データを保存するバケットを作成します。

Amazon S3 バケットを作成するには (AWS CLI)

1. バケットを作成するには、 AWS CLIで次のコマンドを実行します。DOC-EXAMPLE-BUCKET を、すべての で一意のバケットの名前に置き換えます AWS。

   ```
   aws s3api create-bucket --bucket DOC-EXAMPLE-BUCKET
   ```

   デフォルトでは、 `create-bucket` コマンドは `us-east-1` AWS リージョンにバケットを作成します。`us-east-1` 以外の AWS リージョン でバケットを作成するには、`LocationConstraint` パラメーターを追加してリージョンを指定します。たとえば、次のコマンドは `us-west-2` リージョンにファイルシステムを作成しています。

   ```
   aws s3api create-bucket --bucket DOC-EXAMPLE-BUCKET
   --region us-west-2 --create-bucket-configuration LocationConstraint=us-west-2
   ```

   Amazon Comprehend に対応しているリージョンは限られていることに注意してください。Amazon Comprehend に対応しているリージョンについては、『グローバル・インフラストラクチャー・ガイド』の 「リージョン表」 を参照してください。
2. バケットが正常に作成されたことを確認するには、次のコマンドを使用します。このコマンドは、アカウントに関連付けられているすべての S3 バケットを一覧表示します。

   ```
   aws s3 ls
   ```

フォルダーを作成する (コンソールのみ)

次に S3 バケットに 2 つのフォルダを作成します。最初のフォルダは入力データ用です。2 つ目のフォルダは、Amazon Comprehend が分析結果の送信先になる場所です。Amazon S3 コンソールを使用する場合は、フォルダを手動で作成する必要があります。を使用する場合は AWS CLI、サンプルデータセットをアップロードしたり、分析ジョブを実行したりするときにフォルダを作成できます。このため、ここではコンソールユーザー専用にフォルダを作成する手順を説明します。 AWS CLI を使用する場合は、入力データをアップロードする および ステップ 3: Amazon S3 上のドキュメントに対する分析ジョブの実行 にフォルダを作成します。

S3 バケットにフォルダーを作成する (コンソール)

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. [バケット] のバケットリストからバケットを選択します。
3. [概要] タブで [フォルダーの作成] を選択します。
4. 新しいフォルダ名に、input を入力します。
5. 暗号化設定では、[なし (バケット設定を使用)] を選択します。
6. [保存] を選択します。
7. ステップ 3 ~ 6 を繰り返して分析ジョブの出力用の別のフォルダーを作成します。ただし、ステップ 4 では新しいフォルダー名として output を入力します。

入力データをアップロードする

バケットを作成しましたから、これでサンプルデータセット amazon-reviews.csv をアップロードできます。Amazon S3 コンソールまたは AWS CLIを使用して、S3 バケットにデータをアップロードできます。

サンプルドキュメントをバケットにアップロードする(コンソール)

Amazon S3 コンソールで、サンプルデータセットファイルを入力フォルダにアップロードします。

サンプルドキュメントをアップロードする (コンソール)

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. [バケット] のバケットリストからバケットを選択します。
3. input フォルダーを選択し、[アップロード] を選択します。
4. [ファイルを追加] を選択して、コンピューター上のファイル amazon-reviews.csv を選択します。
5. その他の設定はデフォルト値のままにしておきます。
6. [アップロード] を選択します。

サンプルドキュメントをバケットにアップロードする (AWS CLI)

S3 バケットに入力フォルダを作成し、cp コマンドを使用してデータセットファイルをそのフォルダにアップロードします。

サンプルドキュメントをアップロードする (AWS CLI)

1. バケット内の新しいフォルダにamazon-reviews.csvファイルをアップロードするには、次の AWS CLI コマンドを実行します。DOC-EXAMPLE-BUCKET の部分はバケットの名前に置き換えます。Amazon S3 は末尾にパス /input/ を追加することで、バケットに自動的に input という名前の新しいフォルダを作成し、そのフォルダにデータセットファイルをアップロードします。

   ```
   aws s3 cp amazon-reviews.csv s3://DOC-EXAMPLE-BUCKET/input/
   ```

2. バケットが正常にアップロードされたことを確認するには、次のコマンドを使用します。このコマンドは、バケットの input フォルダー内容を一覧表示します。

   ```
   aws s3 ls s3://DOC-EXAMPLE-BUCKET/input/
   ```

これで、input という名前のフォルダに amazon-reviews.csv ファイルを含む S3 バケットが作成されました。コンソールを使用した場合は、バケットに output フォルダも作成されます。を使用した場合は AWS CLI、Amazon Comprehend 分析ジョブの実行時に出力フォルダを作成します。

ステップ 2: (CLI のみ) Amazon Comprehend 用の IAM ロールを作成する

このステップは、 AWS Command Line Interface （ AWS CLI) を使用してこのチュートリアルを完了している場合にのみ必要です。Amazon Comprehend コンソールを使用して分析ジョブを実行する場合は、「ステップ 3: Amazon S3 上のドキュメントに対する分析ジョブの実行」にスキップしてください。

Amazon Comprehend で分析ジョブを実行するには、サンプルデータセットを含み、ジョブの出力を保存する Amazon S3 バケットへのアクセスが必要です。IAM ロールを使用すると、 AWS のサービスまたはユーザーのアクセス許可を制御できます。このステップでは、Amazon Comprehend 用の IAM ロールを作成します。次に、Amazon Comprehend に S3 バケットへのアクセスを許可するリソースベースのポリシーを作成し、このロールにアタッチします。このステップが完了すると、Amazon Comprehend は、入力データへのアクセス、出力の保存、感情分析とエンティティ分析ジョブの実行に必要な権限を取得できます。

IAM での Amazon Comprehend 使用の詳細については、「Amazon Comprehend と IAM 連携の仕組み」を参照ください。

トピック

- 前提条件
- IAM ロールを作成する
- IAM ポリシーを IAM ロールにアタッチする

前提条件

開始する前に、以下を実行します。

- ステップ 1: Amazon S3 にドキュメントを追加する を完了します。
- JSON ポリシーを保存し、Amazon リソースネーム (ARN) を追跡するためのコードまたはテキストエディタを用意してください。

IAM ロールを作成する

Amazon Simple Storage Service (Amazon S3) バケットにアクセスするには、Amazon Comprehend が AWS Identity and Access Management (IAM) ロールを引き受ける必要があります。IAM ロールは Amazon Comprehend を信頼できるエンティティとして宣言します。Amazon Comprehend がロールを引き受けて信頼できるエンティティになると、Amazon Comprehend にバケットアクセス権限を付与できます。このステップでは、Amazon Comprehend を信頼できるエンティティとしてラベル付けするロールを作成します。ロールは、 AWS CLI または Amazon Comprehend コンソールを使用して作成できます。コンソールを使用するには、「ステップ 3: Amazon S3 上のドキュメントに対する分析ジョブの実行」にスキップしてください。

Amazon Comprehend コンソールでは、ロール名に「Comprehend」が含まれ、信頼ポリシーに comprehend.amazonaws.com が含まれるロールを選択できます。コンソールにロールを表示させたい場合は、これらの基準を満たすように CLI で作成したロールを設定します。

Amazon Comprehend の IAM ロールを作成するには (AWS CLI)

1. コンピュータ上のコードまたはテキストエディタで、次の信頼ポリシーを `comprehend-trust-policy.json` という JSON ファイルとして保存します。この信頼ポリシーは、Amazon Comprehend を信頼できるエンティティとして宣言し、IAM ロールを引き受けることを許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "comprehend.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. IAM ロールを作成するには、次の AWS CLI コマンドを実行します。このコマンドは、AmazonComprehendServiceRole-access-role という名前のIAMロールを作成し、そのロールに信頼ポリシーを追加します。*path/* をローカルコンピュータの JSON ドキュメントパスに置き換えます。

```
aws iam create-role --role-name AmazonComprehendServiceRole-access-role
--assume-role-policy-document file://path/comprehend-trust-policy.json
```

> **Tip**
>
> パラメータ解析エラーメッセージが表示された場合は、JSON 信頼ポリシーファイルへのパスが間違っている可能性があります。ホームディレクトリに基づいて、ファイルの相対パスを指定します。

3. Amazon リソースネーム (ARN) をコピーし、テキストエディタに保存します。ARN は、*arn:aws:iam::123456789012:role/AmazonComprehendServiceRole-access-role* のような形式です。この ARN は Amazon Comprehend 分析ジョブを実行するために必要です。

IAM ポリシーを IAM ロールにアタッチする

Amazon S3 バケットにアクセスするには、Amazon Comprehend にリスト、読み取り、書き込みの許可が必要です。Amazon Comprehend に必要なアクセス許可を付与するには、IAM ポリシーを作成して、IAM ロールに追加します。IAM ポリシーにより、Amazon Comprehend はバケットから入力データを取得し、分析結果をバケットに書き込むことができます。ポリシーを作成したら、それを IAM ロールにアタッチします。

IAM ポリシー (AWS CLI) を作成するには

1. 以下のポリシーを comprehend-access-policy.json という名前の JSON ドキュメントとしてローカルに保存します。Amazon Comprehend に指定された S3 バケットへのアクセスを付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

2. S3 バケットアクセスポリシーを作成するには、次の AWS CLI コマンドを実行します。*path/* をローカルコンピュータの JSON ドキュメントパスに置き換えます。

```
aws iam create-policy --policy-name comprehend-access-policy
```

```
--policy-document file://path/comprehend-access-policy.json
```

3. アクセスポリシー ARN をコピーし、テキストエディタに保存します。ARN は、*arn:aws:iam::123456789012:policy/comprehend-access-policy* のような形式です。この ARN は、アクセスポリシーを IAM ロールに追加するために必要です。

IAM ポリシーを IAM ロール (AWS CLI) にアタッチするには

- 以下のコマンドを実行します。*policy-arn* を、前のステップでコピーしたアクセスポリシー ARN に置き換えます。

```
aws iam attach-role-policy --policy-arn policy-arn
--role-name AmazonComprehendServiceRole-access-role
```

これで、Amazon Comprehend の信頼ポリシーと Amazon Comprehend に S3 バケットへのアクセスを付与するアクセスポリシーを含む、AmazonComprehendServiceRole-access-role という名前の IAM ロールができました。また、IAM ロールの ARN がテキストエディタにコピーされました。

ステップ 3: Amazon S3 上のドキュメントに対する分析ジョブの実行

Amazon S3 にデータを保存すると、Amazon Comprehend 分析ジョブの実行を開始することができます。感情分析ジョブでは、ドキュメントの全体的な雰囲気 (肯定的、否定的、中立、混在) が判定されます。エンティティ分析ジョブでは、ドキュメントから現実世界のオブジェクトの名前が抽出されます。そうしたオブジェクトとして、人物や場所、役職、イベント、日付、数量、商品、組織などがあります。このステップでは、2 つの Amazon Comprehend 分析ジョブを実行して、サンプルデータセットから感情とエンティティを抽出します。

トピック

- 前提条件
- 感情とエンティティを分析する

前提条件

開始する前に、以下を実行します。

- ステップ 1: Amazon S3 にドキュメントを追加する を完了します。

- (オプション) を使用している場合は AWS CLI、 を完了ステップ 2: (CLI のみ) Amazon Comprehend 用の IAM ロールを作成するし、IAM ロール ARN を準備します。

感情とエンティティを分析する

最初のジョブでは、サンプルデータセット内の各カスタマーレビューの感情を分析します。2 つ目のジョブでは、各カスタマーレビューのエンティティを抽出します。Amazon Comprehend の分析ジョブは、Amazon Comprehend コンソールまたは AWS CLIを使用して実行できます。

> **Tip**
>
> Amazon Comprehend をサポートする AWS リージョンにいることを確認します。詳細については、『グローバルインフラストラクチャガイド』の「リージョン」の表を参照してください。

感情とエンティティを分析する (コンソール)

Amazon Comprehend コンソールを利用すると、一度に 1 つのジョブを作成します。感情分析ジョブとエンティティ分析ジョブの両方を実行するには、次の手順を繰り返す必要があります。最初のジョブでは IAM ロールを作成しますが、2 つ目のジョブでは最初のジョブの IAM ロールを再利用できます。IAM ロールは、同じ S3 バケットとフォルダを使用する限り再利用できます。

感情分析ジョブとエンティティ分析ジョブを実行する (コンソール)

1. Amazon Simple Storage Service (Amazon S3) バケットを作成したリージョンと同じリージョンにいることを確認します。別のリージョンにいる場合は、ナビゲーションバーで、 AWS リージョンセレクタ から S3 バケットを作成したリージョンを選択します。
2. Amazon Comprehend コンソール (https://console.aws.amazon.com/comprehend/) を開きます。
3. [Launch Amazon Comprehend] (Amazon Comprehend の起動) を選択します。
4. ナビゲーションペインで、[分析ジョブ]] を選択します。
5. [Create job] (ジョブの作成) を選択します。
6. [Job settings] (ジョブの設定) セクションで、以下の操作を行います。
    a. [Name] (名前)に `reviews-sentiment-analysis` と入力します。
    b. 分析タイプで、時系列 を選択します。
    c. [Language] (言語) で、[English] (英語) を選択します。

d. [Job 暗号化] 設定は無効のままにしておきます。

7. [Input data] (入力データ) セクションで、以下の操作を行います。

   a. [Data source] (データソース) で、[My documents] (マイドキュメント) を選択します。
   b. [S3 上の場所]で、[S3 を参照]を選択し、バケットのリストからバケットを選択します。
   c. S3 バケットの [オブジェクト] で、`input` フォルダを選択します。
   d. `input` フォルダでサンプルデータセット `amazon-reviews.csv` を選択し、[選択] を選択します。
   e. [入力形式]で、[ファイルあたり 1 つのドキュメント]を選択します。

8. [Output data] (出力データ) セクションで、以下の操作を行います。

   a. [S3 上の場所]で、[S3 を参照]を選択し、バケットのリストからバケットを選択します。
   b. S3 バケットの [オブジェクト] で、`output` フォルダを選択し、[選択] を選択します。
   c. [暗号化] は無効のままにしておきます。

9. [Access permissions] (アクセス許可) セクションで、以下の操作を行います。

   a. [IAM role] (IAM ロール) で、[Create an IAM role] (IAM ロールの選択) を選択します。
   b. [Permissions to access] (アクセスの許可) で、[Input and Output S3 buckets] (S3 バケットの入力と出力) を選択します。
   c. [Name suffix] (サフィックスに名前を付ける) で、`comprehend-access-role` と入力します。このロールは、Amazon S3 バケットへのアクセスを提供します。

10. [ジョブの作成] を選択します。
11. ステップ 1 ~ 10 を繰り返して、エンティティ分析ジョブを作成します。以下の変更を加えます。

    a. [Job 設定] の [名前] で `reviews-entities-analysis` と入力します。
    b. [Job 設定] の [分析タイプ]で [エンティティ] を選択します。
    c. [アクセス許可] で、[既存の IAM ロールを使用] を選択します。[ロール名] では、`AmazonComprehendServiceRole-comprehend-access-role` (感情ジョブ用に作成したロールと同じロール) を選択します。

感情とエンティティを分析する (AWS CLI)

感情分析ジョブとエンティティ分析ジョブの実行には、`start-sentiment-detection-job` および `start-entities-detection-job` コマンドを利用します。各コマンドを実行すると、 は出力

S3 の場所など、ジョブの詳細にアクセスできるJobId値を持つ JSON オブジェクト AWS CLI を表示します。

感情分析ジョブとエンティティ分析ジョブを実行するには (AWS CLI)

1. 感情分析ジョブを開始するには、 AWS CLIで次のコマンドを実行します。*arn:aws:iam::123456789012:role/comprehend-access-role* の部分は、以前にテキストエディタにコピーした IAM ロールの ARN に置き換えます。デフォルトの AWS CLI リージョンが Amazon S3 バケットを作成したリージョンと異なる場合は、 --regionパラメータを含めて、 をバケットが存在するリージョン*us-east-1*に置き換えます。

   ```
   aws comprehend start-sentiment-detection-job
   --input-data-config S3Uri=s3://DOC-EXAMPLE-BUCKET/input/
   --output-data-config S3Uri=s3://DOC-EXAMPLE-BUCKET/output/
   --data-access-role-arn arn:aws:iam::123456789012:role/comprehend-access-role
   --job-name reviews-sentiment-analysis
   --language-code en
   [--region us-east-1]
   ```

2. ジョブを送信したら、JobId をコピーしてテキストエディタに保存します。分析ジョブの出力ファイルを見つけるには、JobId が必要です。
3. 感情分析ジョブを開始するには、 次のコマンドを実行します。

   ```
   aws comprehend start-entities-detection-job
   --input-data-config S3Uri=s3://DOC-EXAMPLE-BUCKET/input/
   --output-data-config S3Uri=s3://DOC-EXAMPLE-BUCKET/output/
   --data-access-role-arn arn:aws:iam::123456789012:role/comprehend-access-role
   --job-name reviews-entities-analysis
   --language-code en
   [--region us-east-1]
   ```

4. ジョブを送信したら、JobId をコピーしてテキストエディタに保存します。
5. ジョブのステータスを確認します。ジョブの JobId を追跡することで、ジョブの進行状況を確認できます。

   感情分析ジョブの進捗状況を追跡するには、次のコマンドを実行します。*sentiment-job-id* の部分は、感情分析の実行後にコピーした JobId に置き換えます。

   ```
   aws comprehend describe-sentiment-detection-job
   --job-id sentiment-job-id
   ```

エンティティ分析ジョブを追跡するには、次のコマンドを実行します。*entities-job-id* の部分は、エンティティ分析を実行した後にコピーし JobId に置き換えます。

```
aws comprehend describe-entities-detection-job
--job-id entities-job-id
```

JobStatus が COMPLETED と表示されるまで、数分かかります。

感情とエンティティの分析ジョブが完了しました。次のステップに進むには、両方のジョブを完了している必要があります。ジョブが完了するまでに数分かかることがあります。

ステップ 4: データ可視化用に Amazon Comprehend 出力を準備する

データ可視化を作成する際に感情分析ジョブとエンティティ分析ジョブの結果を準備するには、AWS Glue と Amazon Athenaを使用します。このステップでは、Amazon Comprehend の結果ファイルを抽出します。次に、データを調べ、自動的に AWS Glue Data Catalogの表にカタログ化する AWS Glue クローラーを作成します。その後、サーバーレスでインタラクティブなクエリサービスである を使用して Amazon Athena、これらのテーブルにアクセスして変換します。このステップが完了すると、Amazon Comprehend の結果はクリーンになり、可視化できる状態になります。

PII エンティティ検出ジョブの場合、出力ファイルは圧縮されたアーカイブではなくプレーンテキストです。出力ファイル名は入力ファイルと同じで、末尾に .out が付加されます。出力ファイルを抽出する手順は必要ありません。をスキップして にデータをロードします AWS Glue Data Catalog。

トピック

- 前提条件
- 出力をダウンロードする
- 出力ファイルを抽出する
- 抽出したファイルをアップロードする
- データを AWS Glue Data Catalogに読み込む
- 分析するデータを準備する

前提条件

始める前に、ステップ 3: Amazon S3 上のドキュメントに対する分析ジョブの実行 を完了します。

出力をダウンロードする

Amazon Comprehend は Gzip 圧縮を使用して出力ファイルを圧縮し、tar アーカイブとして保存します。出力ファイルを抽出する最も簡単な方法は、`output.tar.gz` アーカイブをローカルにダウンロードすることです。

このステップでは、感情とエンティティの出力アーカイブをダウンロードします。

出力ファイルをダウンロードする (コンソール)

各ジョブの出力ファイルを見つけるには、Amazon Comprehend コンソールの分析ジョブに戻ります。　分析ジョブは出力用の S3 の場所を提供し、そこに出力ファイルをダウンロードできます。

出力ファイルをダウンロードするには (コンソール)

1. Amazon Comprehend コンソールのナビゲーションペインで、[分析ジョブ]に移動します。
2. 感情分析ジョブ `reviews-sentiment-analysis` を選択します。
3. [Output] (出力) で、[Output data location] (出力データの場所) の隣に表示されるリンクをクリックします。これにより、S3 バケットの `output.tar.gz` アーカイブにリダイレクトします。
4. [Overview] (概要) タブで、[Download] (ダウンロード) を選択します。
5. コンピュータ上で、アーカイブの名前を `sentiment-output.tar.gz` に変更します。出力ファイルにはすべて同じ名前が付いているので、感情ファイルとエンティティファイルを追跡しやすくなります。
6. ステップ 1 ~ 4 を繰り返して、 `reviews-entities-analysis` ジョブの出力を検索してダウンロードします。コンピュータ上で、アーカイブの名前を `entities-output.tar.gz` に変更します。

出力ファイルをダウンロードする (AWS CLI)

各ジョブの出力ファイルを検索するには、分析ジョブの `JobId` を使用して出力の S3 ロケーションを検索します。次に、`cp` コマンドを使用して出力ファイルをコンピュータにダウンロードします。

出力ファイルをダウンロードするには (AWS CLI)

1. 感情分析ジョブの詳細を表示するには、以下のコマンドを実行します。*`sentiment-job-id`* を、保存した感情 `JobId` と置き換えます。

```
aws comprehend describe-sentiment-detection-job --job-id sentiment-job-id
```

JobId を見失った場合は、以下のコマンドを実行してすべての感情ジョブを一覧表示し、ジョブを名前でフィルタリングすることができます。

```
aws comprehend list-sentiment-detection-jobs
--filter JobName="reviews-sentiment-analysis"
```

2. OutputDataConfig オブジェクト内の S3Uri 値を検索します。S3Uri 値の形式は次のようになります: *s3://DOC-EXAMPLE-BUCKET/.../output/output.tar.gz* この値をテキストエディタにコピーします。
3. 感情出力アーカイブをローカルディレクトリにダウンロードするには、以下のコマンドを実行します。S3 バケットパスを、前の手順でコピーした S3Uri に置き換えます。*path/* を、ローカルディレクトリへのフォルダパスと置き換えます。名前 sentiment-output.tar.gz は元のアーカイブ名に置き換わるため、感情ファイルやエンティティファイルを追跡しやすくなります。

```
aws s3 cp s3://DOC-EXAMPLE-BUCKET/.../output/output.tar.gz
path/sentiment-output.tar.gz
```

4. エンティティ分析ジョブの詳細を表示するには、以下のコマンドを実行します。

```
aws comprehend describe-entities-detection-job
--job-id entities-job-id
```

JobId が不明な場合は、次のコマンドを実行してエンティティジョブをすべて一覧表示し、ジョブを名前でフィルタリングします。

```
aws comprehend list-entities-detection-jobs
--filter JobName="reviews-entities-analysis"
```

5. エンティティのジョブの説明内の OutputDataConfig オブジェクトから、S3Uri 値をコピーします。
6. エンティティ出力アーカイブをローカルディレクトリにダウンロードするには、以下のコマンドを実行します。　S3 バケットパスを、前の手順でコピーした S3Uri に置き換えます。*path/* を、ローカルディレクトリへのフォルダパスと置き換えます。名前 entities-output.tar.gz は元のアーカイブ名に置き換わります。

```
aws s3 cp s3://DOC-EXAMPLE-BUCKET/.../output/output.tar.gz
path/entities-output.tar.gz
```

出力ファイルを抽出する

Amazon Comprehend の結果にアクセスする前に、感情とエンティティのアーカイブを解凍します。ローカルファイルシステムまたはターミナルのいずれかを使用してアーカイブを解凍できます。

出力ファイルを抽出する (GUI ファイルシステム)

macOS を使用している場合は、GUI ファイルシステムのアーカイブをダブルクリックして、アーカイブから出力ファイルを抽出します。

Windows を使用している場合は、7-Zip などのサードパーティ製のツールを使用して、GUI ファイルシステムの出力ファイルを抽出できます。　Windows で、アーカイブ内の出力ファイルにアクセスするには 2 つの手順を実行する必要があります。　最初にアーカイブを解凍し、次にアーカイブを抽出します。

出力ファイルを区別できるように、感情ファイルの名前を `sentiment-output` に、エンティティファイルの名前を `entities-output` に変更します。

出力ファイルを抽出する (ターミナル)

Linux または macOS を使用している場合は、標準のターミナルを使用できます。　Windows を使用している場合、tar コマンドを実行するには Cygwin などの Unix スタイルの環境にアクセスできる必要があります。

感情出力ファイルを感情アーカイブから抽出するには、ローカルターミナルで次のコマンドを実行します。

```
tar -xvf sentiment-output.tar.gz --transform 's,^,sentiment-,'
```

`--transform` パラメーターは、アーカイブ内の出力ファイルにプレフィックス `sentiment-` が追加され、ファイル名が `sentiment-output` に変更されることに注意してください。これにより、感情とエンティティの出力ファイルを区別し、上書きを防ぐことができます。

エンティティ出力ファイルをエンティティアーカイブから抽出するには、ローカルターミナルで次のコマンドを実行します。

```
tar -xvf entities-output.tar.gz --transform 's,^,entities-,'
```

`--transform` パラメーターは、出力ファイル名にプレフィックス `entities-` を追加します。

> **Tip**
>
> Amazon S3 のストレージコストを節約するために、アップロードする前に Gzip でファイルを再度圧縮できます。は tar アーカイブからデータを自動的に読み取ることができないため、元のアーカイブ AWS Glue を解凍することが重要です。ただし、 AWS Glue は Gzip 形式のファイルから読み取ることができます。

抽出したファイルをアップロードする

ファイルを抽出したら、バケットにアップロードします。がデータを正しく AWS Glue 読み取るには、感情とエンティティの出力ファイルを別々のフォルダに保存する必要があります。バケットに、抽出されたセンチメント結果用のフォルダと、抽出されたエンティティ結果用の 2 つ目のフォルダを作成します。フォルダは、Amazon S3 コンソールまたは AWS CLIで作成できます。

抽出したファイルを Amazon S3 にアップロードする (コンソール)

S3 バケットに、抽出された感情結果用のフォルダ 1 つと、エンティティ結果ファイル用のフォルダ 1 つを作成します。次に、抽出した結果ファイルをそれぞれのフォルダにアップロードします。

抽出したファイルを Amazon S3 にアップロードするには (コンソール)

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. [バケット] でバケットを選択し、[フォルダを作成] を選択します。
3. 新しいフォルダ名に、`sentiment-results` を入力して、[保存] を選択します。このフォルダには、抽出された感情出力ファイルが含まれます。
4. バケットの [概要] タブのバケットコンテンツのリストから、新しいフォルダ `sentiment-results` を選択します [アップロード] を選択します。
5. [ファイル追加] を選択し、ローカルコンピュータから `sentiment-output` ファイルを選択して、[次へ] を選択します。
6. デフォルトとして、「ユーザーの管理」、「他の のアクセス AWS アカウント」、「パブリックアクセス許可の管理」のオプションはそのままにしておきます。[次へ] をクリックします。

7. [ストレージクラス]で、[スタンダード]を選択します。[暗号化] 、[メタデータ] 、[タグ] のオプションはデフォルトのままにします。[次へ] をクリックします。
8. アップロードオプションを確認し、[アップロード] を選択します。
9. ステップ 1 ~ 8 を繰り返して `entities-results` という名前のフォルダを作成し、その `entities-output` フォルダにファイルをアップロードします。

抽出したファイルを Amazon S3 にアップロードする (AWS CLI)

cp コマンドを使用してファイルをアップロードすると S3 バケットにフォルダを作成できます。

抽出したファイルを Amazon S3 にアップロードするには (AWS CLI)

1. 感情フォルダを作成し、以下のコマンドを実行して感情ファイルをアップロードします。*path/* を、抽出した感情出力ファイルのローカルパスに置き換えます。

```
aws s3 cp path/sentiment-output s3://DOC-EXAMPLE-BUCKET/sentiment-results/
```

2. エンティティ出力フォルダを作成し、以下のコマンドを実行してエンティティファイルをアップロードします。*path/* を、抽出したエンティティ出力ファイルのローカルパスに置き換えます。

```
aws s3 cp path/entities-output s3://DOC-EXAMPLE-BUCKET/entities-results/
```

データを AWS Glue Data Catalogに読み込む

結果をデータベースに取り込むには、 AWS Glue クローラー を使用できます。 AWS Glue クローラーはファイルをスキャンし、データのスキーマを検出します。次に、データを AWS Glue Data Catalog (サーバーレスデータベース) のテーブルに配置します。クローラーは、 AWS Glue コンソールまたは を使用して作成できます AWS CLI。

データを AWS Glue Data Catalog に読み込む (コンソール)

`sentiment-results` フォルダと `entities-results`フォルダを個別にスキャンする AWS Glue クローラーを作成します。 AWS Glue の IAM ロールは、S3 バケットへのアクセス許可をクローラーに付与します。この IAM ロールはクローラーの設定時に作成します。

データを にロードするには AWS Glue Data Catalog（コンソール）

1. をサポートするリージョンにいることを確認します AWS Glue。別の地域にいる場合は、ナビゲーションバーの [地域セレクター] からサポートされている地域を選択します。をサポートするリージョンのリストについては AWS Glue、 グローバルインフラストラクチャガイドのリージョン表を参照してください。
2. https://console.aws.amazon.com/glue/ で AWS Glue コンソールを開きます。
3. ナビゲーションペインで、[クローラー]、[クローラーの追加] の順に選択します。
4. [クローラー名] に「`comprehend-analysis-crawler`」と入力し、[次へ] を選択します。
5. [クローラーソースタイプ] で、[データストア]、[次へ] の順に選択します。
6. [データストアの追加]で、次の操作を行います。
   a. [Choose a data store (データストアの選択)] で [S3] を選択します。
   b. [条件]は空白にしておきます。
   c. [データをクロールする]で、[自分のアカウントで指定したパス]を選択します。
   d. [パスを含める]には、感情出力フォルダ `s3://DOC-EXAMPLE-BUCKET/sentiment-results` の S3 フルパスを入力します。
   e. [次へ] をクリックします。
7. [別のデータストアの追加]で [はい]、[次へ]の順に選択します。ステップ 6 を繰り返しますが、エンティティ出力フォルダの完全な S3 パス (`s3://DOC-EXAMPLE-BUCKET/entities-results`) を入力します。
8. [別のデータストアの追加]で、[いいえ]、[次へ]の順に選択します。
9. [IAM ロールの選択]で、以下のいずれかの操作を行います。
   a. [IAM ロールの作成]を選択します。
   b. [IAM ロール] には、`glue-access-role` と入力して [次へ] を選択します。
10. [このクローラーのスケジュールを作成] で [オンデマンドで実行]、[次へ] の順に選択します。
11. [クローラーの出力を設定する]で、次の操作を行います。
    a. [データベース]で、[データベースを追加]を選択します。
    b. [Database name (データベース名)] に「`comprehend-results`」と入力します。このデータベースには、Amazon Comprehend の出力テーブルが保存されます。
    c. 他のオプションをデフォルト設定のままにして、[次へ] をクリックします。
12. クローラー情報を確認してから、[終了]を選択します。

13. Glue コンソールの [クローラー] で comprehend-analysis-crawler を選択し、[クローラーを実行] を選択します。クローラーが終了するまでに数分かかることがあります。

データを AWS Glue Data Catalog に読み込む (AWS CLI)

S3 バケットへのアクセス許可を付与 AWS Glue する の IAM ロールを作成します。次に、 AWS Glue Data Catalogにデータベースを作成します。最後に、データベース内のテーブルにデータをロードするクローラーを作成して実行します。

データを にロードするには AWS Glue Data Catalog（ AWS CLI ）

1. の IAM ロールを作成するには AWS Glue、次の手順を実行します。

   a. 次の信頼ポリシーを glue-trust-policy.json いう JSON ドキュメントとしてコンピュータに保存します。

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "glue.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
       }
     ]
   }
   ```

   b. IAM ロールを作成するには、次のコマンドを実行します。*path/* をローカルコンピュータの JSON ドキュメントパスに置き換えます。

   ```
   aws iam create-role --role-name glue-access-role
   --assume-role-policy-document file://path/glue-trust-policy.json
   ```

   c. が新しいロールの Amazon リソースナンバー (ARN) を AWS CLI 一覧表示したら、コピーしてテキストエディタに保存します。

   d. 次の IAM ポリシーを glue-access-policy.json という JSON ドキュメントとしてコンピュータに保存します。このポリシーは、結果フォルダをクロールする AWS Glue アクセス許可を付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/sentiment-results*",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/entities-results*"
            ]
        }
    ]
}
```

e. IAM ポリシーを作成するには、次のコマンドを実行します。*path/* をローカルコンピュータの JSON ドキュメントパスに置き換えます。

```
aws iam create-policy --policy-name glue-access-policy
--policy-document file://path/glue-access-policy.json
```

f. がアクセスポリシーの ARN を AWS CLI 一覧表示したら、コピーしてテキストエディタに保存します。

g. 次のコマンドを実行して、新しいポリシーを IAM ロールにアタッチします。*policy-arn* を、前のステップでコピーした IAM ポリシー ARN に置き換えます。

```
aws iam attach-role-policy --policy-arn policy-arn
--role-name glue-access-role
```

h. 次のコマンドを実行してAWSGlueServiceRole、 AWS マネージドポリシーを IAM ロールにアタッチします。

```
aws iam attach-role-policy --policy-arn
arn:aws:iam::aws:policy/service-role/AWSGlueServiceRole
--role-name glue-access-role
```

2. 次のコマンドを実行して、 AWS Glue データベースを作成します。

```
aws glue create-database
```

```
--database-input Name="comprehend-results"
```

3. 次のコマンドを実行して、新しい AWS Glue クローラーを作成します。を AWS Glue IAM ロールの ARN *glue-iam-role-arn*に置き換えます。

```
aws glue create-crawler
--name comprehend-analysis-crawler
--role glue-iam-role-arn
--targets S3Targets=[
{Path="s3://DOC-EXAMPLE-BUCKET/sentiment-results"},
{Path="s3://DOC-EXAMPLE-BUCKET/entities-results"}]
--database-name comprehend-results
```

4. 次のコマンドを使用して、クローラーを起動します。

```
aws glue start-crawler --name comprehend-analysis-crawler
```

クローラーが終了するまでに数分かかることがあります。

分析するデータを準備する

これで、Amazon Comprehend の結果が入力されたデータベースができました。ただし、結果はネストされています。ネストを解除するには、 でいくつかの SQL ステートメントを実行します Amazon Athena。 Amazon Athena は、標準 SQL を使用して Amazon S3 内のデータを簡単に分析できるインタラクティブなクエリサービスです。Athena はサーバーレスであるため、管理するインフラストラクチャがなく、 pay-per-query 料金モデルがあります。このステップでは、分析と視覚化に使用できる、クリーンアップされたデータを含む新しいテーブルを作成します。Athena コンソールを使用してデータを準備します。

データを準備するには

1. https://console.aws.amazon.com/athena/ で Athena コンソールを開きます。
2. クエリエディタで、[設定] を選択し、[管理] を選択します。
3. [クエリ結果の場所] には、`s3://DOC-EXAMPLE-BUCKET/query-results/` と入力します。これにより、実行する Amazon Athena クエリの出力を保存する という新しいフォルダがバケット`query-results`に作成されます。[保存] を選択します。
4. クエリエディタで、[エディタ]を選択します。
5. データベース で、`comprehend-results`作成した AWS Glue データベースを選択します。

6. [テーブル] セクションには、sentiment_results と entities_results という 2 つのテーブルがあります。テーブルをプレビューして、クローラーがデータを読み込んだことを確認します。各テーブルのオプション (テーブル名の横にある 3 つの点) で、[テーブルのプレビュー]を選択します。ショートクエリは自動的に実行されます。[結果] ウィンドウをチェックして、テーブルにデータが含まれていることを確認します。

   > **Tip**
   >
   > テーブルにデータがない場合は、S3 バケット内のフォルダを確認してみてください。エンティティ結果用のフォルダが 1 つと、感情結果用のフォルダが 1 つあることを確認します。次に、新しい AWS Glue クローラーを実行してみてください。

7. sentiment_results テーブルのネストを解除するには、次のクエリをクエリエディタに入力し、[実行]を選択します。

```
CREATE TABLE sentiment_results_final AS
SELECT file, line, sentiment,
sentimentscore.mixed AS mixed,
sentimentscore.negative AS negative,
sentimentscore.neutral AS neutral,
sentimentscore.positive AS positive
FROM sentiment_results
```

8. エンティティテーブルのネスト解除を開始するには、クエリエディタに次のクエリを入力し、[実行]を選択します。

```
CREATE TABLE entities_results_1 AS
SELECT file, line, nested FROM entities_results
CROSS JOIN UNNEST(entities) as t(nested)
```

9. エンティティテーブルのネスト解除を開始するには、クエリエディタに次のクエリを入力し、[クエリを実行]を選択します。

```
CREATE TABLE entities_results_final AS
SELECT file, line,
nested.beginoffset AS beginoffset,
nested.endoffset AS endoffset,
nested.score AS score,
nested.text AS entity,
nested.type AS category
```

```
FROM entities_results_1
```

sentiment_results_final テーブルは以下のようになり、[ファイル] 、[行] 、[感情] 、[ミックス] 、[ネガティブ] 、[ニュートラル] 、[ポジティブ] という名前の列があるはずです。テーブルには、1 セルごとに値が 1 つ含まれている必要があります。[感情] 列には、特定のレビューについて最も可能性の高い全体的な感情が表示されます。[ミックス] 、[ネガティブ] 、[ニュートラル] 、[ポジティブ] の各列には、感情のタイプごとにスコアが表示されます。

Results

	file	line	sentiment	mixed	negative	neutral	positive
1	amazon-reviews.csv	6	MIXED	0.9862896203994751	0.0015502438182011247	1.6660270921420306E-4	0.0119935879483
2	amazon-reviews.csv	8	POSITIVE	0.0012987082591280341	0.01186690479516983	0.174478679895401	0.8123556375503
3	amazon-reviews.csv	11	POSITIVE	6.536858109029708 4E-6	0.0013866390800103545	0.007405391428619623	0.9912014007568
4	amazon-reviews.csv	13	POSITIVE	4.7155481297522783E-4	0.24615342915058136	0.017713148146867752	0.7356618046760
5	amazon-reviews.csv	14	POSITIVE	1.5821871784282848E-5	0.06828905642032623	0.014075091108679771	0.9176200628280
6	amazon-reviews.csv	16	MIXED	0.9864791035652161	8.548551704734564E-4	1.0789262159960344E-4	0.0125581491738
7	amazon-reviews.csv	20	NEGATIVE	1.1621621524682269E-4	0.9815887212753296	0.004688907880336046	0.0136061981320
8	amazon-reviews.csv	21	POSITIVE	4.663573781726882E-5	0.009533549658954144	0.0015825830632820725	0.9888372421264
9	amazon-reviews.csv	23	POSITIVE	1.7699007003102452E-4	0.40269607305526733	0.0018250439316034317	0.5953019261360
10	amazon-reviews.csv	25	POSITIVE	1.8434448065818287E-6	1.15832663141191E-4	0.0010993879986926913	0.9987829327583

entities_results_final テーブルは以下のようになり、[ファイル] 、[行] 、[開始オフセット] 、[終了オフセット] 、[スコア] 、[エンティティ] 、[カテゴリ] という名前の列があります。テーブルには、1 セルごとに値が 1 つ含まれている必要があります。[スコア] 列には、検出したエンティティにおける Amazon Comprehend の信頼度が示されます。[カテゴリ] は、Comprehend が検出したエンティティの種類を示します。

Results

	file	line	beginoffset	endoffset	score	entity	category
1	amazon-reviews.csv	0	15	22	0.9885989378545348	English	OTHER
2	amazon-reviews.csv	2	24	28	0.9699371997593782	2 me	QUANTITY
3	amazon-reviews.csv	2	94	95	0.6523066984191679	2	QUANTITY
4	amazon-reviews.csv	2	125	126	0.713791396412543	2	QUANTITY
5	amazon-reviews.csv	4	30	36	0.9957169942979278	kindle	COMMERCIAL_ITEM
6	amazon-reviews.csv	5	1	10	0.9979111763962706	Hawthorne	PERSON
7	amazon-reviews.csv	5	135	142	0.5065408081314243	Puritan	OTHER
8	amazon-reviews.csv	5	143	148	0.7702269458801602	Salem	LOCATION
9	amazon-reviews.csv	5	211	229	0.999675563687763	The Scarlet Letter	TITLE
10	amazon-reviews.csv	5	233	236	0.8944631322676461	one	QUANTITY

これで Amazon Comprehend 結果が表にロードされたので、データから有意義なインサイトを可視化して抽出することができます。

ステップ 5: Amazon で Amazon Comprehend 出力を視覚化する QuickSight

Amazon Comprehend の結果をテーブルに保存したら、Amazon を使用してデータに接続して視覚化できます QuickSight。Amazon QuickSight は、データを視覚化するための AWS マネージドビジネスインテリジェンス (BI) ツールです。Amazon QuickSight では、データソースへの接続と強力なビジュアルの作成が容易になります。このステップでは、Amazon QuickSight をデータに接続し、データからインサイトを抽出するビジュアライゼーションを作成し、ビジュアライゼーションのダッシュボードを公開します。

トピック

- 前提条件
- Amazon QuickSight にアクセス権を付与する
- データセットのインポート
- 感情のビジュアライゼーションを作成する
- エンティティのビジュアライゼーションを作成する
- ダッシュボードの公開
- クリーンアップ

前提条件

始める前に、ステップ 4: データ可視化用に Amazon Comprehend 出力を準備する を完了します。

Amazon QuickSight にアクセス権を付与する

データをインポートするには、Amazon Simple Storage Service (Amazon S3) バケットと Amazon Athena テーブルへのアクセス QuickSight が必要です。Amazon にデータ QuickSight へのアクセスを許可するには、 QuickSight 管理者としてサインインし、リソースのアクセス許可を編集するアクセス許可を持っている必要があります。以下の手順を完了できない場合は、概要ページ チュートリアル:Amazon Comprehend を使用してカスタマーレビューからインサイトを分析する で IAM の前提条件を確認してください。

Amazon にデータ QuickSight へのアクセスを許可するには

1. Amazon QuickSight コンソール を開きます。
2. Amazon を初めて使用する場合は QuickSight、コンソールで E メールアドレスを指定して新しい管理者ユーザーを作成するように求められます。[E メールアドレス] には、自分の AWS アカウントと同じ E メールアドレスを入力します。[Continue] を選択します。
3. サインインしたら、ナビゲーションバーでプロファイル名を選択し、 の管理 QuickSightを選択します。管理 QuickSightオプションを表示するには、管理者としてサインインする必要があります。
4. [セキュリティとアクセス許可] を選択します。
5. QuickSight AWS のサービスにアクセスするには、 を追加または削除を選択します。
6. [Amazon S3] を選択します。
7. [Amazon S3 バケットの選択] から、[S3 バケット] と Athena ワークグループの書き込み権限の両方に使用する S3 バケットを選択します。
8. [Finish] を選択します。
9. [更新] を選択します。

データセットのインポート

視覚化を作成する前に、感情データセットとエンティティデータセットを Amazon に追加する必要があります QuickSight。これは Amazon QuickSight コンソールで行います。ネストされていない感情テーブルとネストされていないエンティティテーブルを からインポートします Amazon Athena。

データセットをインポートするには

1. Amazon QuickSight コンソール を開きます。
2. ナビゲーションバーの [データセット] で、[新規データセット] を選択します。
3. [データセット作成] で [Athena] を選択します。
4. データソースの名前で `reviews-sentiment-analysis` を入力し、[データソースを作成] を選択します。
5. [Database] (データベース) で、データベース `comprehend-results` を選択します。
6. [テーブル] では、感情シート `sentiment_results_final` を選択し、[選択] を選択します。
7. [SPICEにインポートして解析を高速化する]を選択し、[視覚化] を選択します。SPICE は QuickSightのインメモリ計算エンジンで、ビジュアライゼーションの作成時に直接クエリするよりも高速な分析を提供します。
8. Amazon QuickSight コンソールに戻り、データセット を選択します。ステップ 1 ~ 7 を繰り返してエンティティデータセットを作成しますが、以下の変更を行います。
   a. [データソース名]には、`reviews-entities-analysis` を入力します。
   b. [テーブル] では、エンティティテーブル `entities_results_final` を選択します。

感情のビジュアライゼーションを作成する

Amazon のデータにアクセスできるようになったので QuickSight、視覚化の作成を開始できます。Amazon Comprehend の感情データを使用して円グラフを作成します。円グラフには、ポジティブ、ニュートラル、ミックスとネガティブの割合が示されます。

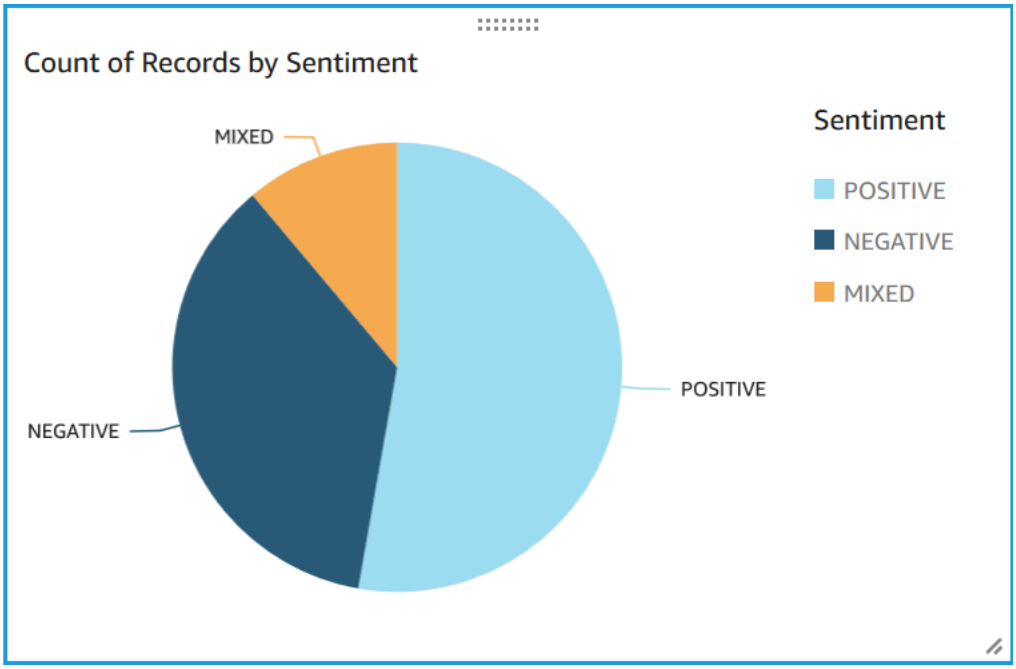
感情データを可視化するには

1. Amazon QuickSight コンソールで、分析 を選択し、新しい分析 を選択します。
2. [Your Data Sets] から感情データセット `sentiment_results_final` を選択し、[分析を作成]を選択します。
3. ビジュアルエディターの[フィールドリスト] で、[感情]を選択します。

> **Note**
>
> [フィールドリスト] 値は、 Amazon Athenaのテーブルの作成に使用した列名によって異なります。SQL クエリで指定した列名を変更した場合、[フィールドリスト]の名前は、これらのビジュアライゼーションの例で使用されている名前とは異なります。

4. [ビジュアルタイプ] には、[円グラフ] を選択します。

ポジティブ、ニュートラル、ミックス、ネガティブのセクションを含む次のような円グラフが表示されます。セクションの数とパーセンテージを確認するには、そのセクションにカーソルを合わせます。



エンティティのビジュアライゼーションを作成する

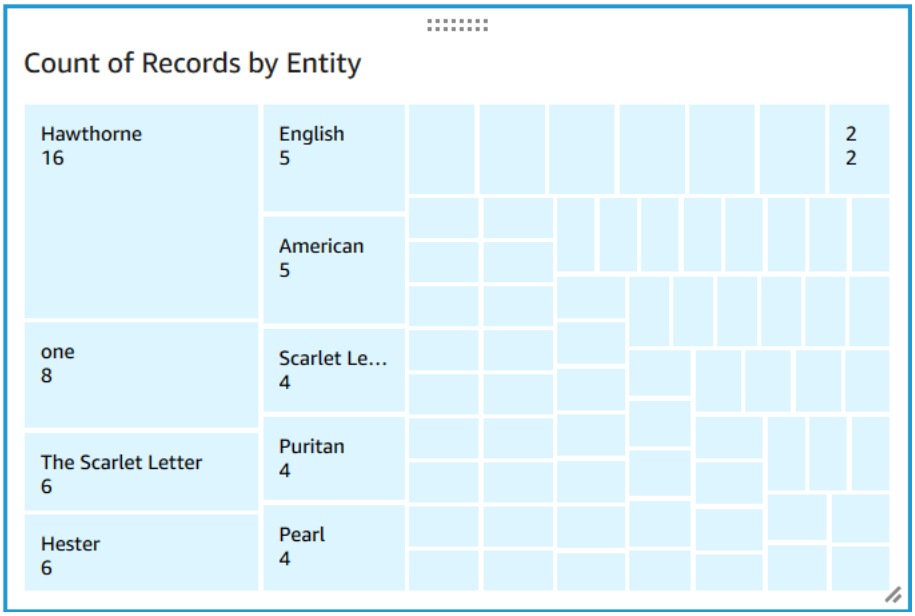
次に、エンティティデータセットを使用して 2 つ目のビジュアライゼーションを作成します。データ内の個別エンティティのツリーマップを作成します。ツリーマップ内の各ブロックは 1 つのエンティティを表し、ブロックのサイズは、そのエンティティがデータセットに表示される回数に関係します。

エンティティデータを可視化するには

1. [Visualize] コントロールペインの [データセット] の横にある [データセットの追加、編集、置換、削除] アイコンを選択します。

2. [Add data set] (データセットを追加) を選択します。
3. [追加するデータセットを選択] で、データセットリストからエンティティデータセット `entities_results_final` を選択し、[選択] を選択します。
4. [可視化] コントロールペインで、[データセット] ドロップダウンメニューを選択し、エンティティデータセット `entities_results_final` を選択します。
5. [フィールドリスト] で [エンティティ] を選択します。
6. [ビジュアルタイプ] には、[ツリーマップ] を選択します。

円グラフの横に、次のようなツリーマップが表示されます。特定のエンティティの数を確認するには、ブロックにカーソルを合わせます。



ダッシュボードの公開

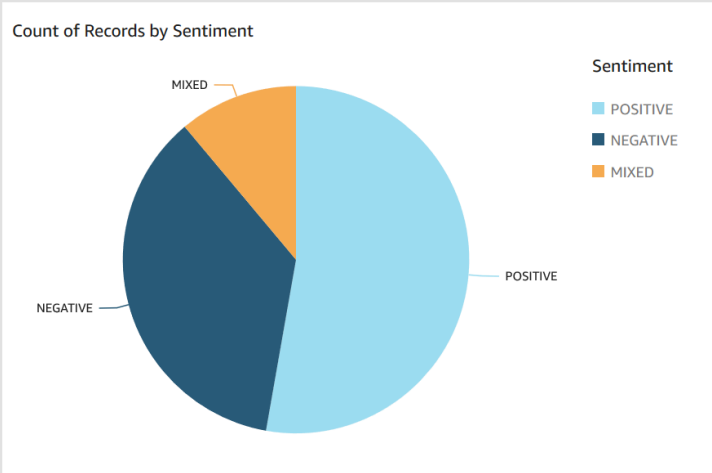
ビジュアライゼーションを作成すると、ダッシュボードとして公開できます。ダッシュボードを使用して、 のユーザーとの共有、PDF としての保存 AWS アカウント、レポートとしての E メール送信 (Amazon の Enterprise Edition に限定) など、さまざまなタスクを実行できます QuickSight。このステップでは、ビジュアル効果をアカウント内のダッシュボードとして公開します。

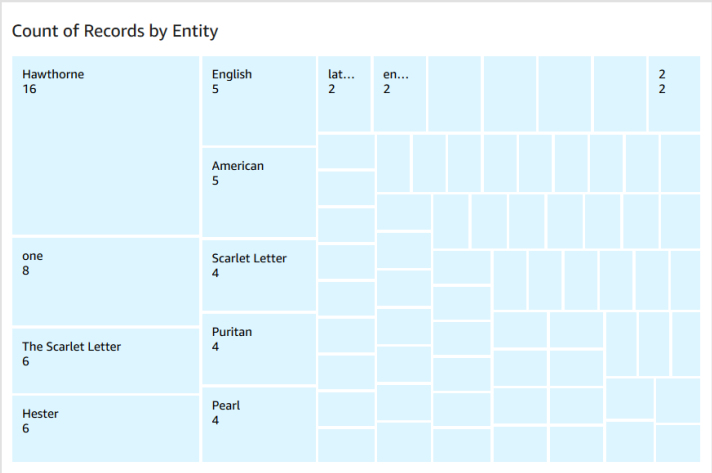
ダッシュボードを公開するには

1. ナビゲーションバーで [共有]を選択します。
2. [ダッシュボードの公開] を選択します。

3. [新規ダッシュボードに名前を付けて公開] を選択し、 ダッシュボードの名前 `comprehend-analysis-reviews` を入力します。
4. [ダッシュボードの公開] を選択します。
5. 右上の閉じるボタンを選択して、[ダッシュボードをユーザーと共有する] ペインを閉じます。
6. Amazon QuickSight コンソールのナビゲーションペインで、Dashboards を選択します。新しいダッシュボード `comprehend-analysis-reviews` のサムネイルが [ダッシュボード] の下に表示されます。ダッシュボードを選択して表示させます。

これで、次の例のような、感情とエンティティのビジュアライゼーションを含むダッシュボードができました。





> **Tip**
>
> ダッシュボードのビジュアライゼーションを編集したい場合は、[分析] に戻り、更新したいビジュアライゼーションを編集してください。次に、そのダッシュボードを新しいダッシュボードとして、または既存のダッシュボードの代わりとして再度公開します。

クリーンアップ

このチュートリアルを完了したら、使用しなくなった AWS リソースをクリーンアップできます。アクティブな AWS リソースには、引き続き アカウントで料金が発生する可能性があります。

次の操作を行うことで、継続的な課金の発生を防ぐことができます。

- Amazon QuickSight サブスクリプションをキャンセルします。Amazon QuickSight は毎月のサブスクリプションサービスです。サブスクリプションをキャンセルするには、「Amazon ユーザーガイド」の「サブスクリプションのキャンセル」を参照してください。 QuickSight
- Amazon S3 バケットを削除します。Amazon S3 はストレージの料金を請求します。Amazon S3 リソースをクリーンアップするには、バケットを削除してください。削除に関する詳細については、[Amazon Simple Storage Service ユーザーガイド] の[S3バケットを削除する方法]をご覧ください。バケットを削除する前に、重要なファイルをすべて保存してください。
- AWS Glue Data Catalogを消去してください。ストレージに対して毎月請求される AWS Glue Data Catalog 料金。データベースを削除することで、継続的な課金の発生を防ぐことができます。 AWS Glue Data Catalog データベースの管理の詳細については、「 AWS Glue デベロッパーガイド」の「 AWS Glue コンソールでのデータベースの使用」を参照してください。データベースやテーブルを消去する前に、必ずデータを出力してください。

個人を特定できる情報 (PII) のための Amazon S3 Object Lambda アクセスポイントの使用

個人を特定できる情報 (PII) 用の Amazon S3 Object Lambda アクセスポイントを使用して、Amazon S3 バケットからドキュメントを取得する方法を構成します。PII を含むドキュメントへのアクセスを制御し、ドキュメントから PII を編集できます。Amazon Comprehend がドキュメント内の PII を検出する方法の詳細については、「PII エンティティの検出」を参照してください。Amazon S3 Object Lambda アクセスポイントは、AWS Lambda 関数を使用して、標準 Amazon S3 GET リクエストの出力を自動的に変換します。詳細については、Amazon Simple Storage Service ユーザーガイドの「S3 ObjecLambda でのオブジェクトの変換」を参照してください。

PII 用の Amazon S3 Object Lambda アクセスポイントを作成すると、ドキュメントは Amazon Comprehend Lambda 関数を使用して処理されます。これにより、PII を含むドキュメントへのアクセスを制御し、ドキュメントの PII を編集します。

PII 用 Amazon S3 Object Lambda アクセスポイントを作成すると、ドキュメントは次の Amazon Comprehend Lambda 関数を使用して処理されます。

- `ComprehendPiiAccessControlS3ObjectLambda` - S3 バケットに保存された PII を含むドキュメントへのアクセスの制御。この Lambda 関数の詳細については、 にサインインAWS Management Consoleして、 で ComprehendPiiAccessControlS3ObjectLambda 関数を表示します AWS Serverless Application Repository。

- `ComprehendPiiRedactionS3ObjectLambda` - Amazon S3 バケット内のドキュメントから PII を編集します。この Lambda 関数の詳細については、 にサインインAWS Management Consoleして、 で ComprehendPiiRedactionS3ObjectLambda 関数を表示しますAWS Serverless Application Repository。

AWS Serverless Application Repository からサーバーレスアプリケーションをデプロイする方法の詳細については、AWSサーバーレスアプリケーションレポジトリデベロッパーガイドの「アプリケーションのデプロイ」を参照してください。

トピック

- 個人を特定できる情報 (PII) を含むドキュメントへのアクセスの制御
- ドキュメントから個人を特定できる情報 (PII) を編集する

個人を特定できる情報 (PII) を含むドキュメントへのアクセスの制御

Amazon S3 Object Lambda アクセスポイントを使用すると、個人を特定できる情報 (PII) を含むドキュメントへのアクセスの制御を行うことができます。

Amazon S3 バケットに保存されている PII を含むドキュメントに、権限のあるユーザーだけがアクセスできるようにするには、`ComprehendPiiAccessControlS3ObjectLambda` 関数を使用します。この Lambda 関数は、ドキュメントオブジェクトに対する標準の Amazon S3 GET リクエストを処理するときに ContainsPiiEntitiesオペレーションを使用します。

たとえば、S3 バケットにクレジットカード番号や銀行口座情報などの PII を含むドキュメントがある場合、`ComprehendPiiAccessControlS3ObjectLambda` 関数を構成してこれらの PII エンティティタイプを検出し、許可されていないユーザーへのアクセスを制限できます。サポートされる PII タイプについては、「PII ユニバーサルエンティティタイプ」を参照してください。

この Lambda 関数の詳細については、 にサインインAWS Management Consoleして、 で ComprehendPiiAccessControlS3ObjectLambda 関数を表示しますAWS Serverless Application Repository。

Amazon S3 Object Lambda アクセスポイントを作成してドキュメントへのアクセスを制御する

次の例では、Amazon S3 Object Lambda アクセスポイントを作成して、社会保障番号を含むドキュメントへのアクセス制御を行います。

AWS Command Line Interface を使用してAmazon S3 Object Lambda アクセスポイントを作成する

Amazon S3 Object Lambda アクセスポイント構成を作成し、config.json という名前のファイルに構成を保存します。

```
{
    "SupportingAccessPoint": "s3-default-access-point-name-arn",
    "TransformationConfigurations": [
        {
            "Actions": [
                "s3:GetObject"
            ],
            "ContentTransformation": {
                "AwsLambda": {
                    "FunctionArn": "comprehend-pii-access-control-s3-object-lambda-
arn",
                    "FunctionPayload": "{\"pii_entities_types\": \"SSN\"}"
                }
            }
        }
    ]
}
```

次の例では、config.json ファイルに定義された構成に基づいて Amazon S3 Object Lambda アクセスポイントを作成します。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws s3control create-banner-access-point \
    --region region \
    --account-id account-id \
    --name s3-object-lambda-access-point \
    --configuration file://config.json
```

Amazon S3 Object Lambda アクセスポイントを呼び出してドキュメントへのアクセスを制御する

次の例では、Amazon S3 Object Lambda アクセスポイントを呼び出して、ドキュメントへのアクセスの制御を行います。

AWS Command Line Interface を使用してAmazon S3 Object Lambda アクセスポイントを呼び出す

次の例では、AWS CLI を使用して Amazon S3 Object Lambda アクセスポイントを呼び出します。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws s3api get-object \
    --region region \
    --bucket s3-object-lambda-access-point-name-arn \
    --key object-prefix-key output-file-name
```

ドキュメントから個人を特定できる情報 (PII) を編集する

Amazon S3 Object Lambda アクセスポイントを使用すると、ドキュメントから個人を特定できる情報 (PII) を編集できます。

S3 バケットに保存されているドキュメントから PII エンティティタイプを編集するには、ComprehendPiiRedactionS3ObjectLambda 関数を使用します。この Lambda 関数は、ドキュメントオブジェクトに対する標準の Amazon S3ContainsPiiEntities GET リクエストを処理するときに、 および DetectPiiEntitiesオペレーションを使用します。

クレジットカード番号や銀行口座情報などの情報を含むドキュメントが S3 バケットにある場合は、ComprehendPiiRedactionS3ObjectLambda 関数を構成して、PII を検出し、PII エンティティタイプがマスキングされたドキュメントのコピーを返すように構成できます。サポートされる PII タイプについては、「PII ユニバーサルエンティティタイプ」を参照してください。

この Lambda 関数の詳細については、 にサインインAWS Management Consoleして、 で ComprehendPiiRedactionS3ObjectLambda 関数を表示しますAWS Serverless Application Repository。

Amazon S3 Object Lambda アクセスポイントを作成して、ドキュメントから PII を編集する

次の例では、Amazon S3 Object Lambda アクセスポイントを作成して、ドキュメントからクレジットカード番号を取り消します。

AWS Command Line Interface を使用してAmazon S3 Object Lambda アクセスポイントを作成する

Amazon S3 Object Lambda アクセスポイントの構成を作成し、config.json というファイルに構成を保存します。

```
{
    "SupportingAccessPoint": "s3-default-access-point-name-arn",
    "TransformationConfigurations": [
        {
            "Actions": [
                "s3:GetObject"
            ],
            "ContentTransformation": {
                "AwsLambda": {
                    "FunctionArn": "comprehend-pii-redaction-s3-object-lambda-arn",
                    "FunctionPayload": "{\"pii_entities_types\": \"CREDIT_DEBIT_NUMBER
\"}"
                }
            }
        }
    ]
}
```

次の例では、config.json ファイルに定義された構成に基づいて Amazon S3 Object Lambda アクセスポイントの作成を示します。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws s3control create-access-point-for-object-lambda \
    --region region \
    --account-id account-id \
    --name s3-object-lambda-access-point \
    --configuration file://config.json
```

Amazon S3 Object Lambda アクセスポイントを呼び出して、ドキュメントから PII を編集します

次の例では、Amazon S3 Object Lambda アクセスポイントを呼び出して、ドキュメントから PII を編集します

AWS Command Line Interface を使用してAmazon S3 Object Lambda アクセスポイントを呼び出す

次の例では、AWS CLI を使用して Amazon S3 Object Lambda アクセスポイントを呼び出します。

例は、Unix、Linux、および macOS 用にフォーマットされています。Windows の場合は、各行末のバックスラッシュ (\) Unix 連結文字をキャレット (^) に置き換えてください。

```
aws s3api get-object \
    --region region \
    --bucket s3-object-lambda-access-point-name-arn \
    --key object-prefix-key output-file-name
```

解決策: Amazon Comprehend と を使用したテキストの分析 OpenSearch

AWS は、Amazon Comprehend と OpenSearch サービスを使用したテキスト分析のリファレンス実装を提供します。Amazon Comprehend はテキスト分析を提供し、ドキュメントのインデックス作成、検索、視覚化 OpenSearch を行います。

詳細については、「 OpenSearch と Amazon Comprehend を使用したテキストの分析」を参照してください。

API リファレンス

API リファレンスは独立したドキュメントになっています。詳細については、『Amazon Comprehend API リファレンス』を参照してください。

Amazon Comprehend のドキュメント履歴

以下の表は、Amazon Comprehend の今回のリリースのドキュメント内容をまとめたものです。

変更	説明	日付
ネイティブドキュメントによるカスタム分類子トレーニング	Amazon Comprehend は、ネイティブドキュメントによるカスタム分類子トレーニングをサポートするようになりました。詳細については、「Amazon Comprehend のトレーニング分類モデル」を参照してください。	2023 年 4 月 19 日
カスタムモデルを管理するためのフライホイール	Amazon Comprehend がフライホイールをサポートするようになり、カスタムモデルのモデルバージョンのトレーニングと追跡を管理しやすくなりました。詳細については、「Amazon Comprehend のフライホイール」を参照してください。	2023 年 2 月 28 日
IAM セキュリティ更新に関するトピック	IAM セキュリティトピックが更新され、フェデレーティッドアイデンティティが含まれるようになりました。詳細については、「Amazon Comprehend の Identity and Access Management」を参照してください。	2022 年 12 月 22 日
カスタムモデルによる推論のワンステップステップ処理	Amazon Comprehend は、カスタム分類またはカスタムエンティティ認識を実行する前	2022 年 12 月 1 日

	に、画像、PDF、または Word の入力ドキュメントのテキスト抽出を自動的に実行するようになりました。詳細については、「Amazon Comprehend のドキュメント処理」を参照してください。	
ターゲット感情のための同期 API	Amazon Comprehend は、ターゲット感情の同期 API とコンソールのリアルタイム分析をサポートするようになりました。ターゲット感情は、ドキュメント内の特定のエンティティに関連する感情を決定します。詳細については、「Amazon Comprehend のターゲット感情」を参照してください。	2022 年 9 月 21 日
レコグナイザをトレーニングするためのコメントの最小下限	Amazon Comprehend では、プレーンテキストの CSV 注釈ファイルを使用してレコグナイザーをトレーニングするための最小要件が低減されました。注釈付きドキュメントがわずか 3 つ、エンティティタイプ 1 つにつき 25 個以上の注釈を使用して、カスタムエンティティ認識モデルを構築できるようになりました。詳細については、「トレーニングデータの準備」を参照してください。	2022 年 8 月 3 日

リアルタイム API の入力ドキュメントサイズ増大	Amazon Comprehend は、ほとんどのリアルタイム API で最大 100 KB の入力ドキュメントをサポートするようになりました。詳細については、「ガイドラインとクォータ」を参照してください。	2022 年 7 月 18 日
追加の PII エンティティタイプ	Amazon Comprehend によって検出されるようになったその他の PII エンティティタイプ 詳細については、「Amazon Comprehend での PII エンティティの検出」を参照してください。	2022 年 5 月 20 日
目次の再編成	Amazon Comprehend デベロッパーガイドの目次を再構成したことで、ナビゲーションしやすくしました。詳細については、「Amazon Comprehend とは」を参照してください。	2022 年 4 月 7 日
ターゲット感情	Amazon Comprehend は、ドキュメント内の特定のエンティティに関連する感情を判断する、ターゲット感情分析をサポートするようになりました。詳細については、「Amazon Comprehend のターゲット感情」を参照してください。	2022 年 3 月 9 日

新機能	Amazon Comprehend では、画像を分析してカスタムエンティティを認識できるようになりました。詳細については、「Amazon Comprehend でのカスタムエンティティの検出」を参照してください。	2022 年 2 月 28 日
新機能	トレーニング済みのカスタムモデルを AWS アカウント 間でコピーできるようになりました。詳細については、「Amazon Comprehend のアカウント間でのカスタムモデルのコピー」を参照してください。	2022 年 2 月 2 日
新機能	AWS Trusted Advisor を使用して、Amazon Comprehend エンドポイントのコストとセキュリティを最適化するのに役立つ推奨事項を確認できるようになりました。詳細については、「Amazon Comprehend で Trusted Advisor を使用する」を参照してください。	2021 年 9 月 29 日

新機能	Amazon Comprehend は Comprehend Custom 向けの一連の機能をを提供し、モデルの新しいバージョンの作成、特定のテストセットでの継続的なテスト、新しいモデルのエンドポイントへのライブマイグレーションを可能にすることで、モデルの継続的な改善を実現します。	2021 年 9 月 21 日
新機能	Amazon Comprehend では、PDF および Word ドキュメントを分析してカスタムエンティティを認識できるようになりました。PDF 形式と Word 形式の場合、ヘッダー、リスト、表を含むドキュメントから情報を抽出できます。	2021 年 9 月 14 日
新機能	Amazon Comprehend は、エンドポイントのグローバルビューを提供する新しいエンドポイント概要機能をリリースしました。エンドポイント概要ページでは、すべてのエンドポイントを 1 か所に表示して、エンドポイントの使用状況と実際のリソース使用状況を把握できます。	2021 年 8 月 24 日

新機能	Amazon Comprehend Medical で、インターフェイス VPC エンドポイントを作成することにより、Virtual Private Cloud (VPC) とのプライベート接続を確立できるようになりました。詳細については、「VPC endpoints(PrivateLink) 」を参照してください。	2021 年 6 月 13 日
言語拡張	Amazon Comprehend では、主要な言語機能として、ハウサ語 (ha)、ラオス語 (lo)、マルタ語 (mt)、オロモ語 (om) の 4 つの言語が追加されました。詳細については、「Amazon Comprehend でサポートしている言語」を参照してください。	2021 年 5 月 10 日
新機能	Amazon Comprehend では、カスタマーマネージドキー (CMK) を使用してカスタムモデルを暗号化できるようになりました。詳細については、「Amazon Comprehend での KMS の暗号化」を参照してください。	2021 年 3 月 31 日

新機能	Amazon S3 Object Lambda アクセスポイントを使用して、個人を特定できる情報 (PII) を含むドキュメントを Amazon S3 バケットから取得する方法を構成できるようになりました。PII を含むドキュメントへのアクセスを制御し、ドキュメントから PII を編集できます。詳細については、「個人を特定できる情報 (PII) のための Amazon S3 Object Lambda アクセスポイントの使用」を参照してください。	2021 年 3 月 18 日
新機能	ドキュメントに個人を特定できる情報 (PII) をラベル付けできるようになりました。Amazon Comprehend は、ドキュメント内にPIIが存在するかどうかを分析し、名前、住所、銀行口座番号、電話番号など、識別されたPIIエンティティタイプのラベルを返します。詳細については、「ドキュメントに PII をラベル付けする」を参照してください。	2021 年 3 月 11 日

新機能	Amazon Comprehend では、一連のドキュメント内のイベントを検出できるようになりました。非同期イベント検出ジョブを作成すると、Amazon Comprehend はサポートされている財務イベントタイプを検出できます。詳細については、「イベントの検出」を参照してください。	2020 年 11 月 24 日
新機能	Amazon Comprehend では、カスタムエンティティレコグナイザーエンドポイントに自動スケーリングを使用できるようになりました。自動スケーリングを使用すると、キャパシティのニーズに合わせてエンドポイントのプロビジョニングを自動的に設定できます。詳細については、「エンドポイントでの自動スケーリング」を参照してください。	2020 年 9 月 28 日

新機能	カスタム分類子またはエンティティレコグナイザーをトレーニングするために、Amazon SageMaker Ground Truth によって生成されるラベル付きデータセットである拡張マニフェストファイルを提供できるようになりました。これらのファイルの詳細と例については、「マルチクラスモード」、「マルチラベルモード」、および「注釈」を参照してください。	2020 年 9 月 22 日
チュートリアルの新規追加	Amazon Comprehend に、カスタマーレビューを分析し、分析結果を可視化するマルチサービスワークフローを順を追って説明するチュートリアルが追加されました。詳細については、「チュートリアル: レビューからのインサイトの分析」を参照してください。	2020 年 9 月 17 日

新機能	Amazon Comprehend を使用すると、住所、銀行口座番号、電話番号などの個人を特定できる情報 (PII) を含むテキスト内のエンティティを検出できるようになりました。Amazon Comprehend は、テキスト内の各 PII エンティティの場所を提供することも、PII が編集されたテキストのコピーを提供することもできます。詳細については、「個人を特定できる情報 (PII) の検出」を参照してください。	2020 年 9 月 17 日
新機能	以前は、最大 12 個 のカスタムエンティティで 1 つのモデルしかトレーニングできませんでした。Amazon Comprehend では、一度に最大 25 のカスタムエンティティでモデルをトレーニングできるようになりました。詳細については、「カスタムエンティティ認識」を参照してください。	2020 年 8 月 12 日

言語拡張	Amazon Comprehend では、カスタムエンティティ認識機能に、ドイツ語 (de)、スペイン語 (es)、フランス語 (fr)、イタリア語 (it)、ポルトガル語 (pt) の 5 つの言語が追加されました。詳細については、「Amazon Comprehend でサポートしている言語」を参照してください。	2020 年 8 月 12 日
新機能	Amazon Comprehend で、インターフェイス VPC エンドポイントを作成することにより、仮想プライベートクラウド (VPC) とのプライベート接続を確立できるようになりました。詳細については、「VPC エンドポイント (AWS PrivateLink)」を参照してください。	2020 年 8 月 11 日
新機能	Amazon Comprehend では、リアルタイム分析を実行することで、個々のテキストドキュメント内のカスタムエンティティをすばやく検出できるようになりました。詳細については、「Amazon Comprehend でのカスタムエンティティのリアルタイムでの検出」を参照してください。	2020 年 7 月 9 日

新機能の追加	Amazon Comprehend では、ドキュメントの非同期カスタム分類の 2 番目のモードがサポートされるようになりました。これにより、ドキュメントにカスタムクラスを適用する際の柔軟性が大幅に向上します。マルチクラスモードでは各ドキュメントに 1 つのクラスしか関連付けられませんが、新しいマルチラベルモードでは複数のクラスを関連付けることができます。たとえば、1 本の映画をSFとアクションの 2 つに同時に分類することができます。詳しくは、「カスタム分類におけるマルチクラスモードとマルチラベルモード」を参照してください。	2019 年 12 月 19 日
新機能の追加	Amazon Comprehend は、非構造化テキストを含むドキュメントのリアルタイムカスタム分類をサポートするようになりました。お客様は、リアルタイムのカスタム分類を使用して、独自のビジネスルールに基づいて情報の理解、タグ付け、ルーティングを同期化できます。詳細については、「カスタム分類によるリアルタイム分析」を参照してください。	2019 年 11 月 25 日

新しい言語が追加されました	Amazon Comprehend では、その機能のいくつかに対応するため、アラビア語 (ar)、ヒンディー語 (hi)、日本語 (ja)、韓国語 (ko)、簡体字中国語 (zh)、繁体字中国語 (zh-TW) の 6 つの言語が追加されました。これらの新しい言語は、感情の判定、キーフレーズの検出、およびカスタムではないエンティティ検出操作でのみサポートされています。さらなる詳細については、Supported languages を参照してください。	2019 年 11 月 6 日
新機能	以前は、1 つのカスタムエンティティでのみモデルをトレーニングできました。そのため、エンティティレコグナイザー操作ではその 1 つのエンティティしか検索できませんでした。Amazon Comprehend はこれを変更し、一度に最大 12 のカスタムエンティティでモデルをトレーニングできるようになりました。詳細については、「カスタムエンティティ認識」を参照してください。	2019 年 7 月 9 日

新機能	Amazon Comprehend では、カスタム分類子をトレーニングする際にメトリクスを分析する機能が追加されたマルチクラス混同マトリックスが提供されるようになりました。これは現在、APIを使用することのみによりサポートされています。詳細については、「Amazon Comprehend のリソースにタグ付けする」を参照してください。	2019 年 4 月 5 日
新機能	Amazon Comprehend には、カスタム分類子とカスタムエンティティレコグナイザー用のタグが用意されています。これらのタグをメタデータとして使用することで、リソースへのアクセスをこれまで以上に細かく整理、フィルタリング、制御できます。詳細については、「Amazon Comprehend のリソースにタグ付けする」を参照してください。	2019 年 4 月 3 日

新機能	Amazon S3 では、すでに入力ドキュメントを暗号化することができます。Amazon Comprehend は、これをさらに拡張します。独自の KMS キーを使用することで、ジョブの出力結果だけでなく、分析ジョブを処理するコンピュートインスタンスに追加されたストレージボリューム上のデータも暗号化できます。結果はセキュリティです end-to-end。詳細については、「Amazon Comprehend での KMS の暗号化」を参照してください。	2019 年 3 月 28 日
新機能	カスタムエンティティ認識により、Amazon Comprehend の機能が拡張され、あらかじめ設定された共通エンティティタイプの 1 つとしてサポートされていない新規エンティティタイプを識別できるようになります。つまり、ドキュメントを分析し、製品コードやビジネス固有のエンティティなど、特定のニーズに合ったエンティティを抽出できます。詳細については、「カスタムエンティティ認識」を参照してください。	2018 年 11 月 16 日

新機能	Amazon Comprehend を使用して、カスタム分類用の独自モデルを構築し、ドキュメントをクラスまたはカテゴリに割り当てることができます。詳細については、「ドキュメント分類」を参照してください。	2018 年 11 月 15 日
リージョンの拡張	Amazon Comprehend が欧州 (フランクフルト) (eu-central-1) で利用可能になりました。	2018 年 10 月 10 日
言語拡張	Amazon Comprehend では、英語とスペイン語のほかに、フランス語、ドイツ語、イタリア語、ポルトガル語のドキュメントも調べることができるようになりました。詳細については、「Amazon Comprehend でサポートしている言語」を参照してください。	2018 年 10 月 10 日
リージョンの拡張	Amazon Comprehend が、アジアパシフィック (シドニー) (ap-southeast-2) で利用可能になりました。	2018 年 8 月 15 日
新機能	Amazon Comprehend はドキュメントを解析して、ドキュメントの構文と各単語の品詞を検出するようになりました。詳細については、「Syntax」を参照してください。	2018 年 7 月 17 日

新機能	Amazon Comprehend は、言語、キーフレーズ、エンティティ、および感情を検出するための非同期バッチ処理をサポートするようになりました。詳細については「非同期バッチ処理」を参照してください。	2018 年 6 月 27 日
新しいガイド	これは、「Amazon Comprehend デベロッパーガイド」の最初のリリースです。	2017 年 11 月 29 日

AWS 用語集

AWS の最新の用語については、「AWS の用語集リファレンス」の「AWS 用語集」を参照してください。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。