実習科目研修
クラウドコンピューティングA

AWS　AIサービスAPIを使用したAIシステムの開発

第３回

Comprehend（文章から話題や感情を抽出）サービス

Forecast（予測）サービス

Lex（対話型エージェント）サービス

機械学習マネージドサービス

これらのマネージドサービスに ML の経験は不要。

コンピュータビジョン

Amazon Rekognition

Amazon Textract

チャットボット

Amazon Lex

音声

Amazon Polly

Amazon Transcribe

予測

Amazon Forecast

言語

Amazon Comprehend

Amazon Translate

レコメンデーション

Amazon Personalize

Amazon Comprehend サービス

Amazon Comprehend

ドキュメント内のテキストから価値あるインサイトを導き出し、理解する

Amazon Comprehend の使用を開始する　お問い合わせ

ドキュメント、カスタマーサポートチケット、製品レビュー、電子メール、ソーシャルメディアフィードなどに含まれるテキストから、価値あるインサイトを発見します。

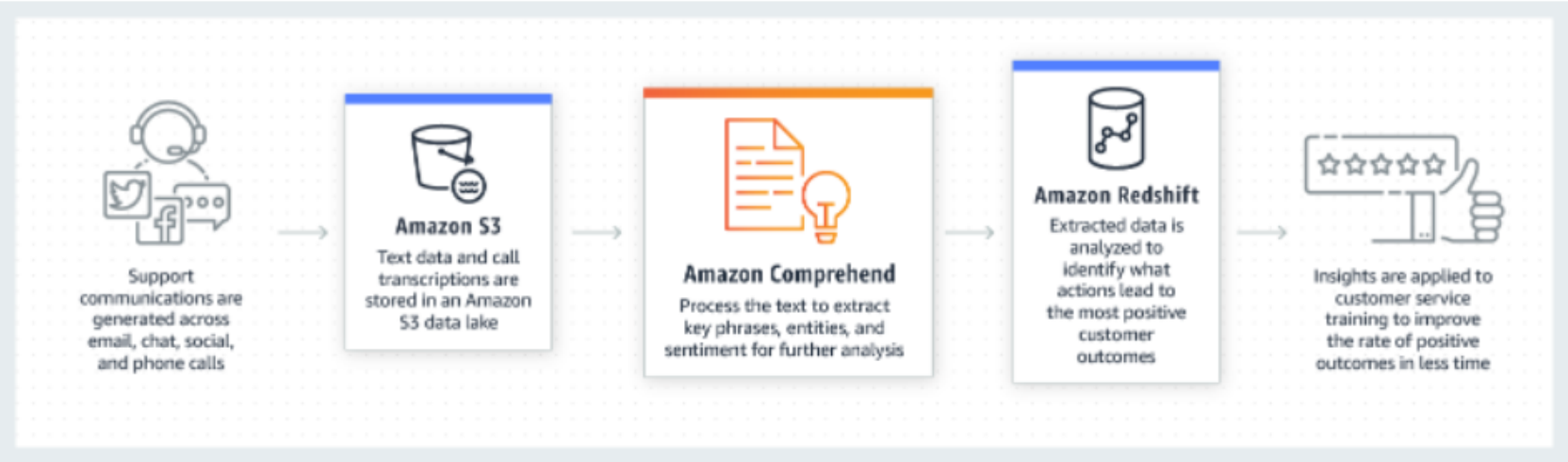
保険金請求書などのドキュメントから、テキスト、キーフレーズ、トピック、センチメントなどを抽出することで、ドキュメント処理のワークフローを簡素化します。

機械学習の経験の必要はなく、モデルをトレーニングしてドキュメントを分類し、言葉を識別することで、ビジネスを差別化します。

ドキュメントから個人を特定できる情報 (PII) を識別して再編集することで、機密データへのアクセス権を保護および管理します。

仕組み

Amazon Comprehend は、機械学習を使用して、テキストからインサイトや関係性を発見するための自然言語処理 (NLP) サービスです。



Comprehendとは（P266～P268）

言語検出	文章がどの言語で書かれているのかを判定する 判定できる言語の種類は１００種類
感情分析	文章の話者が、どんな感情を持っているのかを推測する。 結果はポジティブ（肯定的）、ネガティブ（否定的）、 ニュートラル（中立的）、ミックス（混合）の４種理
キーフレーズ抽出	文章から「重要そうな言い回し」を抽出する。 その文章が何を話題にしているのかを、 大まかに知ることができる。
エンティティ認識	文章から人名、地名、社名といった固有名詞や、日付などの 情報を抽出する。
構文解析	文章の構文を解析する。 文章を構成する各単語について、名詞や動詞といった品詞の 情報が得られる。

Comprehendの利用場面

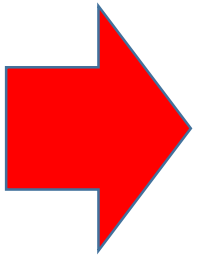
多数の文章から大まかな傾向を調べたいとき役に立つ

ある商品のレビューが大量にあるときに、文章から感情を推測することによって、全体として肯定的なレビューが多いのか、否定的なレビューが多いのかを調べることができる。

Comprehendが対応する言語

言語コード	言語名
de	ドイツ語
en	英語
es	スペイン語
fr	フランス語
it	イタリア語
pt	ポルトガル語

テキスト出版時

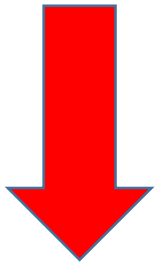


言語コード	言語名
de	ドイツ語
en	英語
es	スペイン語
fr	フランス語
it	イタリア語
pt	ポルトガル語
ja	日本語
zh-TW	中国語（繁体字）
zh	中国語（簡体字）
ko	韓国語
hi	ヒンディー語
ar	アラビア語

2022年時

言語を検出する（P269～P271）

I'm looking forward to visiting Japan next summer.
（来年の夏に日本を訪れるのを楽しみにしています。）



detect_dominant_language
メソッド


```
{
  "Languages": [
    {
      "LanguageCode": "en",
      "Score": 0.9679933786392212
    }
  ],
  "ResponseMetadata": {
    "RequestId": "71f3e3cf-17d4-4961-bd3e-8b5d08f0d607",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "71f3e3cf-17d4-4961-bd3e-8b5d08f0d607",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "64",
      "date": "Mon, 23 May 2022 08:31:29 GMT"
    },
    "RetryAttempts": 0
  }
}
```


キー	型	値の内容
Languages	辞書のリスト	検出した言語の一覧
LanguageCode	文字列	言語コード
Score	数値	検出の信頼度を表すスコア

文字列の言語を検出するプログラム（P270）

```python
# 各種ライブラリーのインポート
import boto3
import json

# Comprehend サービスクライアントを作成
comprehend = boto3.client('comprehend', 'us-east-1')
# 処理対象の文字列を設定
text = "I'm looking forward to visiting Japan next summer."
# 言語を検出
result = comprehend.detect_dominant_language(Text=text)
# 結果を整形して表示
print(json.dumps(result, indent=2))
```

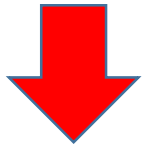
Comprehend　detect_domain_languageメソッド　P271

機能：文字列の主要な言語を検出する
使い方：detect_domain_language(Text= **文字列**)
戻り値：検出した言語の情報を含む辞書

CSVファイルの言語を検出する（P272～P274）

CSVファイル（comp_language.csv）

```
20200105,"Bear","It was good that shipment was quick."
20200216,"Bär","Ich möchte, dass die Versandkosten günstiger sind."
20200327,"Ours","Je veux plus de variations de couleurs."
20200408,"クマ","また注文したいです。"
```



detect_dominant_language
メソッド

```
$> python comp_language.py
It was good that shipment was quick.
en 0.998666524887085

Ich möchte, dass die Versandkosten günstiger sind.
de 0.9996805787086487

Je veux plus de variations de couleurs.
fr 0.9963231086730957

また注文したいです。
ja 1.0


$>
```

感情を分析する（P275～P277）

I'm looking forward to visiting Japan next summer.
（来年の夏に日本を訪れるのを楽しみにしています。）

detect_sentiment
メソッド

```
$> python comp_sentiment_dump.py
{
  "Sentiment": "POSITIVE",
  "SentimentScore": {
    "Positive": 0.9842482209205627,
    "Negative": 0.00042805029079318047,
    "Neutral": 0.015279133804142475,
    "Mixed": 4.46327576355543e-05
  },
  "ResponseMetadata": {
    "RequestId": "134e8fb5-d1be-47f5-ac5e-3ac052b41ca5",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "134e8fb5-d1be-47f5-ac5e-3ac052b41ca5",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "165",
      "date": "Wed, 08 Jun 2022 08:43:08 GMT"
    },
    "RetryAttempts": 0
  }
}

$>
```

分析した感情の情報

キー	型	値の内容
Sentiment	文字列	推測した感情
SentimentScore	辞書	感情ごとのスコアのリスト
Positive	数値	肯定的な感情のスコア
Negative	数値	否定的な感情のスコア
Neutral	数値	中立的な感情のスコア
Mixed	数値	混じった感情のスコア

文字列の感情を分析するプログラム（P276～P277）comp_sentiment_dump.py

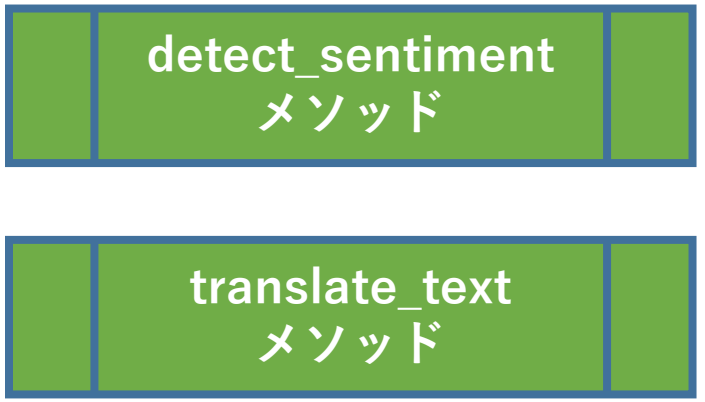
```
# 各種ライブラリのインポート
import boto3
import json

# Comprehendサービスクライアントを作成
comprehend = boto3.client('comprehend', 'us-east-1')
# 処理対象の文字列
text = "I'm looking forward to visiting Japan next summer."
# 感情を分析
result = comprehend.detect_sentiment(Text=text, LanguageCode='en')
# 結果を整形して表示
print(json.dumps(result, indent=2))
```

Comprehend　detect_sentimentメソッド　P277

機能：文字列の感情を分析する
使い方：detect_sentiment(
　　Text= **文字列**,
　　LanguageCode = **言語コード**）
戻り値：分析した言語の情報を含む辞書

CSVファイルの感情を分析する（P278～P281）

	A	B	C
1	20200105	Bear	It was good that shipment was quick.
2	20200216	Bär	Ich möchte, dass die Versandkosten günstiger sind.
3	20200327	Ours	Je veux plus de variations de couleurs.
4	20200408	クマ	また注文したいです。




```
$> python comp_sentiment.py
It was good that shipment was quick.
( It was good that shipment was quick. )
( 発送が早くてよかったです。 )
POSITIVE
  Positive    0.9995189905166626
  Negative    0.00011006184649886563
  Neutral     0.00016486212552990764
  Mixed       0.0002060992701444775

Ich möchte, dass die Versandkosten günstiger sind.
( I would like the shipping costs to be cheaper. )
( 送料をもっと安くしたいのですが。 )
NEGATIVE
  Positive    0.036663614213466644
  Negative    0.9371935725212097
  Neutral     0.020754767581820488
  Mixed       0.005388014484196901

Je veux plus de variations de couleurs.
( I want more color variations. )
( もっとカラーバリエーションが欲しい。 )
NEUTRAL
  Positive    0.3016678988933563
  Negative    0.32277435064315796
  Neutral     0.33251243829727173
  Mixed       0.04304526373744011

また注文したいです。
( I would like to order again. )
( また注文したいです。 )
POSITIVE
  Positive    0.9979466795921326
  Negative    0.0005012791370972991
  Neutral     0.0014936564257368445
  Mixed       5.83338005526457e-05


$>
```

CSVファイルの感情を分析するプログラム（P279～P281）comp_sentiment.py

```
# boto3インポート
import boto3
# csvインポート
import csv
# リージョンを設定
region = 'us-east-1'
# Translateサービスクライアントを生成
translate = boto3.client('translate', region)
# Comprehendサービスクライアントを生成
comprehend = boto3.client('comprehend', region)
# CSVファイルの読み込み
with open('comp_sentiment.csv', 'r', encoding='utf-8') as file:
    # 行単位で読み込む
    for row in csv.reader(file):
        # ３列目の文字列を英語に翻訳
        result_en = translate.translate_text(
            Text=row[2],
            SourceLanguageCode='auto',
            TargetLanguageCode='en')
        # ３列目の文字列を日本語に翻訳
        result_ja = translate.translate_text(
            Text=row[2],
            SourceLanguageCode='auto',
            TargetLanguageCode='ja')
        # 英語に翻訳された列の文字列の感情を分析
        result_comp = comprehend.detect_sentiment(
            Text=result_en['TranslatedText'],
            LanguageCode='en')
        # 原文を表示
        print(row[2])
        # 翻訳された英文を表示
        print('(', result_en['TranslatedText'], ')')
        # 翻訳された日本文を表示
        print('(', result_ja['TranslatedText'], ')')
        # 感情を表示
        print(result_comp['Sentiment'])
        # 各感情の情報を取得
        for key, value in result_comp['SentimentScore'].items():
            # 各感情の名前とスコアを表示
            print('  {:10} {}'.format(key, value))
        print() # 改行
```

キーフレーズを抽出する（P282～P284）

I tried to use the 20% OFF coupon, ¥
but only 10% discount and I was unable to
place an order.
（20%のクーポンを使用しようとしたのですが、10%
しか割引がならず、注文できませんでした。）

detect_key_phrases
メソッド

```
$> python comp_phrase_dump.py
{
  "KeyPhrases": [
    {
      "Score": 0.9782768487930298,
      "Text": "the 20% OFF coupon",
      "BeginOffset": 15,
      "EndOffset": 33
    },
    {
      "Score": 0.9186369180679321,
      "Text": "only 10% discount",
      "BeginOffset": 39,
      "EndOffset": 56
    },
    {
      "Score": 0.9999603033065796,
      "Text": "an order",
      "BeginOffset": 83,
      "EndOffset": 91
    }
  ],
  "ResponseMetadata": {
    "RequestId": "1f133fc9-f2d4-4fde-a9df-bfd8bbfe9f66",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "1f133fc9-f2d4-4fde-a9df-bfd8bbfe9f66",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "272",
      "date": "Wed, 08 Jun 2022 09:32:03 GMT"
    },
    "RetryAttempts": 0
  }
}
```

抽出したキーフレーズの情報

キー	型	値の内容
Keyphrases	文字列	キーフレーズのリスト
Score	辞書	検出の信頼度を表すスコア
Text	数値	キーフレーズの文字列
BiginOffset	数値	文字列内のキーフレーズの開始位置
EndOffset	数値	文字列内のキーフレーズの終了位置

キーフレーズを抽出するプログラム（P284）comp_phrase_dump.py

```
# boto3インポート
import boto3
# jsonインポート
import json
# Comporehendサービスクライアントを生成
comprehend = boto3.client('comprehend', 'us-east-1')
# 処理対象の文字列
text = "I tried to use the 20% OFF coupon, ¥
but only 10% discount and I was unable to place an order."
# キーフレーズを抽出
result = comprehend.detect_key_phrases(Text=text, LanguageCode='en')
# 結果をJSON形式に整形して表示
print(json.dumps(result, indent=2))
```

Comprehend　detect_key_phrasesメソッド　P284

機能：文字列のキーフレーズを抽出する
使い方：detect_key_phrases(
　　Text= 文字列,
　　LanguageCode = 言語コード）
戻り値：抽出したキーフレーズの情報を含む辞書

テキストファイルのキーフレーズを抽出する（P285～P288）

Thank you for using our software.
Your question is probably due to a different version of Python.

Is your environment Windows or macOS or Linux?
Use the "python" command on Windows or "python3" command on macOS or Linux.
If you use the "python" command on macOS or Linux, the software may not work as expected because Python 2 starts instead of Python 3.

I hope the above measures solve the problem.
Thank you for your continued use of our software.

HigPen Works

```
$> python comp_phrase.py
0.9999865293502808 Your question
0.9999830126762390 our software
0.9999808669090271 a different version
0.9999800920486450 the problem
0.9999700188636780 the software
0.9999427199363708 your continued use
0.9999057054519653 the "python" command
0.9998526573181152 Python
0.9996860623359680 the above measures
0.9996262192726135 Python 3
0.9993065595626831 Python 2
0.9989516735076904 your environment
0.9972262978553772 Windows
0.9661321043968201 macOS or Linux
0.9094141125679016 "python3" command
```

detect_key_phrases
メソッド

テキストファイルのキーフレーズを抽出するプログラム（P286～P287）comp_phrase.py

```
# boto3 のインポート
import boto3
# Comprehendサービスクライアントを作成
comprehend = boto3.client('comprehend', 'us-east-1')
# テキストファイルを開く
with open('comp_phrase.txt', 'r', encoding='utf-8') as file:
    # 文章からキーフレーズを抽出
    result = comprehend.detect_key_phrases(
        Text=file.read(),
        LanguageCode='en'
    )
    # レポート作成用の辞書を初期化
    report = {}
    # キーフレーズを順番に処理
    for phrase in result['KeyPhrases']:
        # 文字列とスコアを取得
        text, score = phrase['Text'], phrase['Score']
        # レポートにスコアと文字列を登録
        report[text] = '{:<018} {}'.format(score, text)
    # レポートをソートしてから表示
    for line in sorted(report.values(), reverse=True):
        print(line)
```

エンティティを認識する（P289～P293）

I'm looking forward to visiting Japan next summer.
（来年の夏に日本を訪れるのを楽しみにしています。）

detect_entities
メソッド

認識したエンティティの情報

キー	型	値の内容
Entities	辞書のリスト	エンティティのリスト
Score	数値	検出の信頼度を表すスコア
Type	文字列	エンティティのタイプ
Text	文字列	エンティティの文字列
BeginOffset	整数	文字列内のエンティティの開始位置
EndOffset	整数	文字列内のエンティティの終了位置

```
$> python comp_entity_dump.py
{
    "Entities": [
        {
            "Score": 0.9993908405303955,
            "Type": "LOCATION",
            "Text": "Japan",
            "BeginOffset": 32,
            "EndOffset": 37
        },
        {
            "Score": 0.9923950433731079,
            "Type": "DATE",
            "Text": "next summer",
            "BeginOffset": 38,
            "EndOffset": 49
        }
    ],
    "ResponseMetadata": {
        "RequestId": "29a71bc6-1ec2-4f4d-b140-3b764215799b",
        "HTTPStatusCode": 200,
        "HTTPHeaders": {
            "x-amzn-requestid": "29a71bc6-1ec2-4f4d-b140-3b764215799b",
            "content-type": "application/x-amz-json-1.1",
            "content-length": "204",
            "date": "Thu, 09 Jun 2022 08:11:41 GMT"
        },
        "RetryAttempts": 0
    }
}

$>
```

文字列内のエンティティを認識するプログラム（P286～P287）comp_entity_dump.py

```
# boto3インポート
import boto3
# jsonインポート
import json
# comprehendサービスクライアントを生成
comprehend = boto3.client('comprehend', 'us-east-1')
# エンティティ認識対象文字列の初期化
text = "I'm looking forward to visiting Japan next summer."
# エンティティを認識
result = comprehend.detect_entities(Text=text,
LanguageCode='en')
# 結果をJSON形式で整形表示
print(json.dumps(result, indent=4))
```

Comprehend　detect_entitiesメソッド　P291

機能：文字列内のエンティティを認識する
使い方：detect_entities(Text= **文字列**, LanguageCode = **言語コード**)
戻り値：認識したエンティティの情報を含む辞書

エンティティのタイプ

タイプ名	意味
COMMERCIAL__ITEM	ブランド商品
DATE	日付、時刻
EVENT	イベント（祭り、コンサート、選挙など）
LOCATION	場所（国、都市、湖、建物など）
PERSON	人（個人、グループ、ニックネーム、架空のキャラクター
ORGANIZATION	組織
QUANTITY	数量（金額、パーセンテージ、数値、バイト数など）
TITLE	タイトル（映画、本、歌など）
OTHER	その他（上記のカテゴリに入らないもの）

構文を解析する（P294～P298）

I'm looking forward to visiting Japan next summer.
（来年の夏に日本を訪れるのを楽しみにしています。）

detect_syntax
メソッド

```
$> python comp_syntax_dump.py
{
  "SyntaxTokens": [
    {
      "TokenId": 1,
      "Text": "I",
      "BeginOffset": 0,
      "EndOffset": 1,
      "PartOfSpeech": {
        "Tag": "PRON",
        "Score": 0.9999825954437256
      }
    },
    {
      "TokenId": 2,
      "Text": "'m",
      "BeginOffset": 1,
      "EndOffset": 3,
      "PartOfSpeech": {
        "Tag": "AUX",
        "Score": 0.9918413162231445
      }
    },
    {
      "TokenId": 10,
      "Text": ".",
      "BeginOffset": 49,
      "EndOffset": 50,
      "PartOfSpeech": {
        "Tag": "PUNCT",
        "Score": 0.9999933838844299
      }
    }
  ],
  "ResponseMetadata": {
    "RequestId": "4ca83bc0-9e4e-447d-93b5-4678a5ddf8dc",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "4ca83bc0-9e4e-447d-93b5-4678a5ddf8dc",
      "content-type": "application/x-amz-json-1.1",
      "content-length": "1187",
      "date": "Thu, 09 Jun 2022 08:55:53 GMT"
    },
    "RetryAttempts": 0
  }
}
```

構文解析の情報（アルファベット順）

キー	型	値の内容
SyntaxTokens	辞書のリスト	トークンのリスト
TokenID	整数	トークンのID
Text	文字列	トークンの文字列
BeginOffset	整数	文字列内のトークンの開始位置
EndOffset	整数	文字列内のトークンの終了位置
PartOfSpeech	辞書	品詞
Tag	文字列	品詞を識別するタグ
Score	数値	品詞の判定の信頼度を表すスコア

文字列の構文を解析するプログラム（P296）comp_syntax_dump.py

```python
# boto3インポート
import boto3
# jsonインポート
import json
# comprehendサービスクライアントを生成
comprehend = boto3.client('comprehend', 'us-east-1')
# 構文解析対象文字列の初期化
text = "I'm looking forward to visiting Japan next summer."
# 構文を解析
result = comprehend.detect_syntax(Text=text,
LanguageCode='en')
# 結果を整形して表示
print(json.dumps(result, indent=2))
```

Comprehend　detect_syntaxメソッド　P296

機能：文字列の構文を解析する
使い方：detect_sysntax(Text= **文字列**, LanguageCode = **言語コード**)
戻り値：構文解析の情報を含む辞書

Amazon Forecast サービス

Amazon Forecast

機械学習を用いて、ビジネスの成果を簡単かつ正確に予測

Amazon Forecast の使用を開始する

10,000 件の時系列を予測
(AWS 無料利用枠 2 か月間)

Amazon.com と同じ技術を使用して、数百万アイテムを予測することで、オペレーションをスケール。

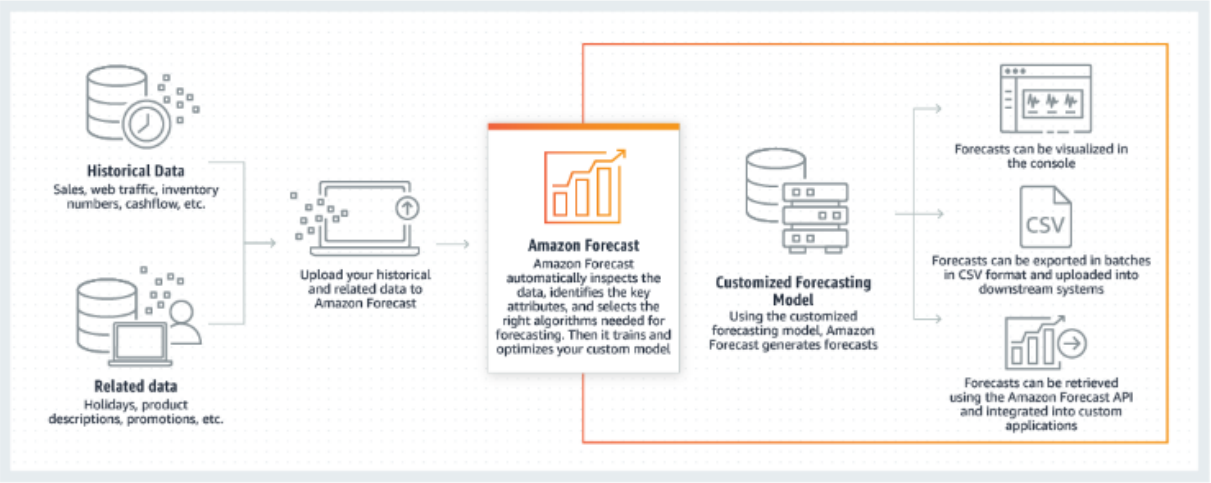
きめ細かなレベルでの正確な予測により、在庫を最適化し、無駄を省きます。

資本稼働率を向上させ、自信を持って長期的な意思決定を行うことができます。

需要レベルの変化に応じた最適な人員配置により、顧客の満足度を高めることができます。

仕組み

Amazon Forecast は、機械学習 (ML) をベースにした時系列予測サービスで、ビジネスメトリクス分析のために構築されています。



拡大イメージを見る

ユースケース

SaaS ソリューションに機械学習予測機能を追加

統合された機械学習ベースの予測により、複雑な需要関係を特定し、SaaS (Software as a Service) 製品の機能を強化します。

製品需要計画の最適化

過去の売上と需要のデータを、関連するウェブトラフィック、料金、製品カテゴリー、天候、休日の情報と組み合わせることで、個々の店舗における在庫ニーズを予測します。

リソースの効率的な管理

ほぼリアルタイムで正確なリソース要求予測を行い、稼働率と顧客の満足度を高めます。

Forecastとは（P356～P358）

時系列データ（時間とともに変化する値の集まり）に関して
過去の値から未来の値を予測するAIサービス

（1）データの登録

過去の値をForecastに登録する。このデータを「データセット」と呼ぶ。

（2）予測子の作成

データセットを使った機械学習を行い、予測子を構築する。
予測を行うための仕組みを予測子と呼ぶ。

（3）予測の取得

予測子を使って予測した、未来の値を取得する

使用するデータ（P359～P360）

Forecastに登録するデータは、CSV形式のデータ
CSVの各行は以下の構成になっている必要がある

タイムスタンプ（timestamp），数値（target_value），アイテム（item_id）

データの意味

「タイムスタンプ」における、「アイテム」の「数値」。
Forecastが予測するのはこの数値の部分

Forecastは以下のドメイン（Domain）に対応した予測を行う

ドメイン名	用途
CUSTOM	下記以外の時系列データを予測
EC2 CAPACITY	EC2の容量を予測
INVENTORY_PLANNING	原材料の需要を予測
METRICS	収益や売り上げを予測
RETAIL	小売の需要を予測
WEB_TRAFFIC	Webのアクセス数を予測
WORK__FORCE	労働力の需要を予測

本書で使用するデータ
CUSTOMドメインデータの予測
1970年から2018年までのデータのを学習して
2019年の月別平均気温を予測させる

```
1970-01-01 00:00:00,4.5,Tokyo
1970-01-01 00:00:00,4.8,Osaka
1970-02-01 00:00:00,6.0,Tokyo
1970-02-01 00:00:00,6.1,Osaka
1970-03-01 00:00:00,5.5,Tokyo
```

```
2018-10-01 00:00:00,19.7,Osaka
2018-11-01 00:00:00,14.0,Tokyo
2018-11-01 00:00:00,14.6,Osaka
2018-12-01 00:00:00,8.3,Tokyo
2018-12-01 00:00:00,9.4,Osaka
```

データを登録する（P361～P364）

（1）データセットの作成

①Forecast上にデータセットを作成する
②データ構造を定義するスキーマを設定する

（2）データセットグループの作成

実行結果

```
$> python fore_create_dataset.py
dataset ARN: arn:aws:forecast:us-east-1:440854864937:dataset/MyDataset
dataset group ARN: arn:aws:forecast:us-east-1:440854864937:dataset-group/MyGroup

$>
```

fore_create_dataset.py

```
# boto3インポート
import boto3
# jsonインポート
import json
# forecastサービスクライアントを作成
forecast = boto3.client('forecast')
# スキーマ定義
schema_json = {
    "Attributes": [
        # タイムスタンプ
        {
            "AttributeName": "timestamp",
            "AttributeType": "timestamp"
        },
        # 数値
        {
            "AttributeName": "target_value",
            "AttributeType": "float"
        },
        # アイテム
        {
            "AttributeName": "item_id",
            "AttributeType": "string"
        }
    ]
}
# データセットを作成
dataset_arn = forecast.create_dataset(
    Domain='CUSTOM', DatasetType='TARGET_TIME_SERIES',
    DatasetName='MyDataset', DataFrequency='M',
    Schema=schema_json)['DatasetArn']
print('dataset ARN:', dataset_arn)
# データセットグループを作成
group_arn = forecast.create_dataset_group(
    DatasetGroupName='MyGroup', Domain='CUSTOM',
    DatasetArns=[dataset_arn])['DatasetGroupArn']
print('dataset group ARN:', group_arn)
```

Forecast　create_datasetメソッド　P363

機能：データセットを作成する
使い方：create_dataset(
　　Domain=ドメイン,
　　DatasetType＝データセットのタイプ,
　　DatasetName=データセット名 ,
　　DataFrequency=時系列データの頻度,
　　Schema=スキーマのJSON)
戻り値：作成したデータセットのARNを含む辞書

Forecast　create_dataset_groupメソッド　P364

機能：データセットを作成する
使い方：create_dataset_group(
　　DatasetGroupName=グループ名,
　　Domain=ドメイン,
　　DatasetArns=[データセットのARN])
戻り値：作成したデータセットグループのARNを含む辞書

データセットのタイプ（DatasetType）

値	意味
'TARGET_TIME_SERIES'	予測の対象となる時系列データ
'RELATED_TIME_SERIES'	関連する時系列データ
'ITEM_METADATA'	アイテムに関するデータ

時系列データの頻度（DataFrequency）

値	意味
'Y'	年（Year）
'M'	月（Month）
'W'	週（Week）
'D'	日（Day）
'H'	時（Hour）
'30min'	30分（30minutes）
'15min'	15分（15minutes）
'10min'	10分（10minutes）
'5min'	5分（5minutes）
'1min'	1分（1minutes）

ロールを作成する（P365～P367）

時系列データの取り込み手順

（１）ロールの作成

Forecastに対して、S3にアクセスする許可を与える。
IAMを使ってロールを作成する。

（２）S3へのアップロード

S3上にデータをアップロードする

（３）データの取り込み

S3上にアップロードしたデータを、Forecastに取り込む

ロールを作成するプログラム（P365～P367）

```
# boto3インポート
import boto3
# jsonインポート
import json
# IAMサービスクライアントを作成
iam = boto3.client('iam')
# ロールの定義
role_json = {
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
        "Principal": {
          "Service": "forecast.amazonaws.com"
        },
      "Action": "sts:AssumeRole"
    }
  ]
}
# ロールの作成
role = iam.create_role(
    RoleName='ForecastRole',
    AssumeRolePolicyDocument=json.dumps(role_json))
print('role ARN:', role['Role']['Arn'])
```

```
# ポリシーの定義
policy_json = {
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:Get*",
        "s3:List*",
        "s3:PutObject"
      ],
      "Effect":"Allow",
      "Resource":[
        "arn:aws:s3:::*"
       ]
    }
  ]
}
# ポリシーを作成
policy = iam.create_policy(
    PolicyName='ForecastPolicy',
    PolicyDocument=json.dumps(policy_json))
print('policy ARN:', policy['Policy']['Arn'])
# ロールにポリシーを追加
iam.attach_role_policy(
    RoleName='ForecastRole',
    PolicyArn=policy['Policy']['Arn'])
```

データを取り込む（P368～P373）データを取り込むプログラム

```
# boto3, json, uuid, time インポート
import boto3
import json
import uuid
import time
# s3サービスクライアントを作成
region = 'ap-northeast-1'
s3 = boto3.client('s3', region)
# バケット名を生成
bucket = str(uuid.uuid1())
print('bucket:', bucket)
# バケットを作成
s3.create_bucket(
    Bucket=bucket,
    CreateBucketConfiguration={'LocationConstraint': region})
# バケットのポリシーを定義
policy_json = {
    "Version": "2012-10-17",
    "Id": "ForecastS3BucketAccessPolicy",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "forecast.amazonaws.com"
            },
            "Action": [
                "s3:Get*",
                "s3:List*",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::"+bucket,
                "arn:aws:s3:::"+bucket+"/*"
            ]
        }
    ]
}
# バケットにポリシーを追加
s3.put_bucket_policy(Bucket=bucket,
           Policy=json.dumps(policy_json))
```

```
# s3にファイルをアップロード
file_name = 'temperature.csv'
s3.upload_file(file_name, bucket, file_name)
# ロール（ForecastRole）のARNを取得
iam = boto3.client('iam')
role_arn = iam.get_role(RoleName='ForecastRole')['Role']['Arn']
# Forecastサービスクライアントを作成
forecast = boto3.client('forecast', region)
# データセット（MyDataset）を取得
for dataset in forecast.list_datasets()['Datasets']:
    if dataset['DatasetName'] == 'MyDataset':
        break
dataset_arn = dataset['DatasetArn']
# データ取り込みのジョブを開始
job_arn = forecast.create_dataset_import_job(
    DatasetImportJobName='MyJob',
    DatasetArn=dataset_arn,
    DataSource={'S3Config': {
        'Path': 's3://'+bucket+'/'+file_name, 'RoleArn': role_arn}},
    TimestampFormat='yyyy-MM-dd hh:mm:ss')['DatasetImportJobArn']
print('job ARN:', job_arn)
# データ取り込みのジョブの進捗を表示
start = time.time()
status = ''
while status in ['CREATE_PENDING', 'CREATE_IN_PROGRESS']:
    status = forecast.describe_dataset_import_job(
        DatasetImportJobArn=job_arn)['DatasetImportJob']['Status']
    time.sleep(10)
    print('{:7.2f} {}'.format(time.time()-start, status))
```

予測子と予測を作成する（P374～P378）予測子と予測を作成するプログラム

```
import time
# Forecastサービスクライアントを作成
forecast = boto3.client('forecast')
# データセットグループを取得
for group in forecast.list_dataset_groups()['DatasetGroups']:
    if group['DatasetGroupName'] == 'MyGroup':
        break
group_arn = group['DatasetGroupArn']
# 予測子を作成
predictor_arn = forecast.create_predictor(
    PredictorName='MyPredictor', ForecastHorizon=12,
    PerformAutoML=True,
    InputDataConfig={'DatasetGroupArn': group_arn},
    FeaturizationConfig={'ForecastFrequency': "M"})['PredictorArn']
print('predictor ARN:', predictor_arn)
# 予測し作成の進捗を表示
start = time.time()
status = ''
while status not in ['ACTIVE', 'CREATE_FAILED']:
    status = forecast.describe_predictor(
        PredictorArn=predictor_arn)['Status']
    time.sleep(10)
    print('{:7.2f} {}'.format(time.time()-start, status))
# 予測を作成
forecast_arn = forecast.create_forecast(
    ForecastName='MyForecast',
    PredictorArn=predictor_arn)['ForecastArn']
print('forecast ARN:', forecast_arn)
# 予測作成の進捗を表示
start = time.time()
status = ''
while status not in ['ACTIVE', 'CREATE_FAILED']:
    status = forecast.describe_forecast(
        ForecastArn=forecast_arn)['Status']
    time.sleep(10)
    print('{:7.2f} {}'.format(time.time()-start, status))
```

予測を取得する（P379～P383）予測を取得するプログラム

```
# boto3, sysインポート
import boto3
import sys
# コマンドライン引数の個数が不適切ならば使い方を表示して終了
if len(sys.argv) != 2:
    print('python', sys.argv[0], 'item-id')
    exit()
# Forecastサービスクライアントを作成
forecast = boto3.client('forecast')
# ForecastQueryサービスクライアントを作成
forecast_query = boto3.client('forecastquery')
# 予測（MyForecast）を取得
for fc in forecast.list_forecasts()['Forecasts']:
    if fc['ForecastName'] == 'MyForecast':
        break
forecast_arn = fc['ForecastArn']
# 指定したアイテムIDに関する予測値を取得
result = forecast_query.query_forecast(
    ForecastArn=forecast_arn,
    Filters={'item_id': sys.argv[1]})
# 予測値を分位点（P10, P50, P90）ごとに表示
for prediction in result['Forecast']['Predictions']:
    print(prediction+':')
    # タイムスタンプと予測値を表示
    for line in result['Forecast']['Predictions'][prediction]:
        print(line['Timestamp'], line['Value'])
    print()
```

不要になったリソースを削除する（P384～P385）予測子を削除するプログラム

```
# boto3インポート
import boto3
# Forecastサービスクライアントを作成
forecast = boto3.client('forecast')
# 予測を取得
for fc in forecast.list_forecasts()['Forecasts']:
    forecast_arn = fc['ForecastArn']
    print('forecast ARN:', forecast_arn)
    # 予測を削除
    forecast.delete_forecast(ForecastArn=forecast_arn)
# 予測子を取得
for predictor in forecast.list_predictors()['Predictors']:
    predictor_arn = predictor['PredictorArn']
    print('predictor ARN:', predictor_arn)
    # 予測子を削除
    forecast.delete_predictor(PredictorArn=predictor_arn)
```

不要になったリソースを削除する（P384～P385）ロールを削除するプログラム

```
# boto3インポート
import boto3
# IAMサービスクライアントを作成
iam = boto3.client('iam')
# ポリシー（ForecastPolicy）を取得
for policy in iam.list_policies()['Policies']:
    if policy['PolicyName'] == 'ForecastPolicy':
        break
policy_arn = policy['Arn']
print('policy ARN:', policy_arn)
# ロールからポリシーを分離
iam.detach_role_policy(
    RoleName='ForecastRole', PolicyArn=policy_arn)
# ポリシーを削除
iam.delete_policy(PolicyArn=policy_arn)
# ロールを削除
iam.delete_role(RoleName='ForecastRole')
```

不要になったリソースを削除する（P384～P385）データを削除するプログラム

```
# boto3インポート
import boto3
# Forecastサービスクライアントを作成
forecast = boto3.client('forecast')
# データセットグループの一覧を取得
for group in forecast.list_dataset_groups()['DatasetGroups']:
    group_arn = group['DatasetGroupArn']
    print('dataset group ARN:', group_arn)
    # データセットグループを削除
    forecast.delete_dataset_group(DatasetGroupArn=group_arn)
# データセットの一覧を取得
for dataset in forecast.list_datasets()['Datasets']:
    dataset_arn = dataset['DatasetArn']
    print('dataset ARN:', dataset_arn)
    # データセットを削除
    forecast.delete_dataset(DatasetArn=dataset_arn)
```

Amazon Lex サービス

Amazon Lex

会話型 AI で Chatbot を構築

Amazon Lex の使用を開始する

10,000 件のテキストリクエストと5,000件のスピーチリクエストが 12 か月間無料
(AWS 無料利用枠)

意図を理解し、コンテキストを維持し、多くの言語で簡単なタスクを自動化する AI を簡単に追加できます。

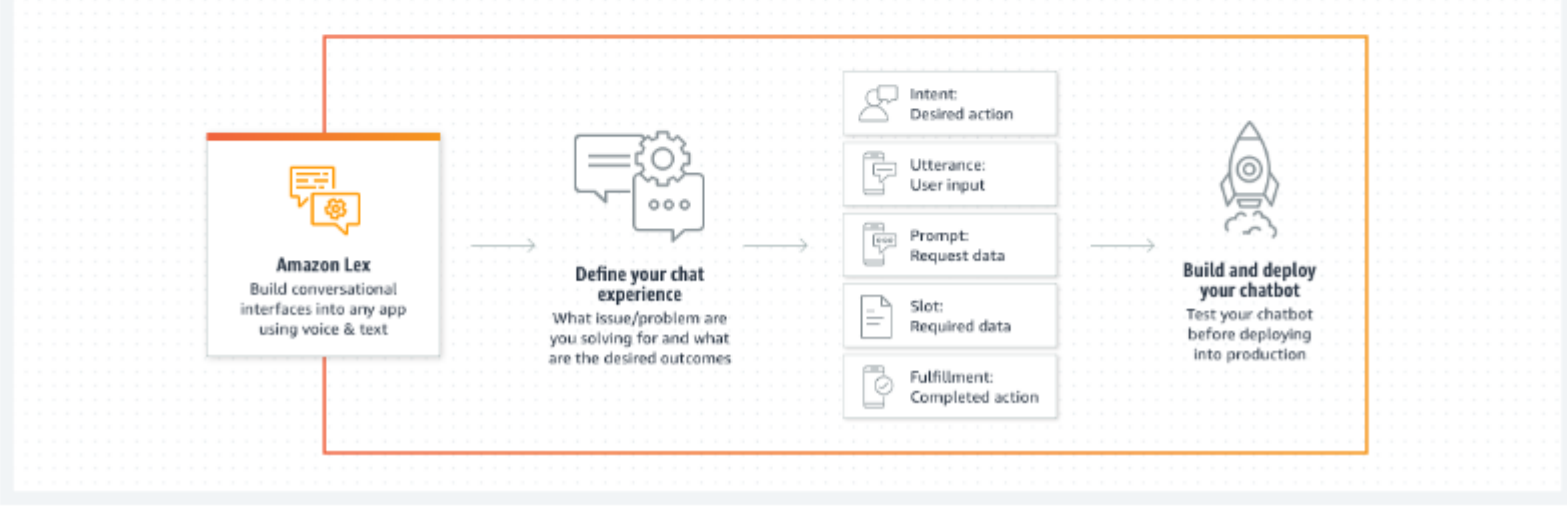
ハードウェアやインフラストラクチャを気にすることなく、ワンクリックでオムニチャネルの会話型 AI を設計およびデプロイできます。

他の AWS のサービスとシームレスに接続して、データのクエリ、ビジネスロジックの実行、パフォーマンスのモニタリングなどを行うことができます。

初期コストや最低料金はなく、音声やテキストのリクエストに対してのみお支払いいただきます。

仕組み

Amazon Lex は、アプリケーションに会話型インターフェイスを設計、構築、テスト、およびデプロイするための高度な自然言語モデルを備えた、フルマネージド型人工知能 (AI) サービスです。



これから作成するボットの会話例（P396）

```
$> python lex_chat_text.py
You: ice cream
Bot: Corn or cup?
You: corn
Bot: Vanilla, chocolate or strawberry?
You: choco
Bot: Vanilla, chocolate or strawberry?
You: choc
Bot: OK, chocolate ice cream in corn

Flavor   : chocolate
Container: corn

$>
```

ボットを作成する（P396～P409）

```
$> python lex_create_bot.py
slot type: FlavorSlotType
slot type: ContainerSlotType
intent: OrderIntent
bot: MyBot
  10.26 BUILDING
  20.50 BUILDING
  30.75 BUILDING
  40.94 READY
bot alias: MyBotAlias

$>
```

スロットタイプ（フレーバー）の作成

スロットタイプ（容器）の作成

インテントの作成

ボットの作成
ボットの構築

ボットエリアスの作成

ロールの作成（P397～P398）

```
# boto3インポート
import boto3
# timeインポート
import time
# IAMサービスクライアントを作成
iam = boto3.client('iam')
# サービスにリンクされたロールを作成
iam.create_service_linked_role(AWSServiceName='lex.amazonaws.com')
# Lexサービスクライアントを作成
lex = boto3.client('lex-models', 'us-east-1')
```

IAM　create_service_linked_roleメソッド　P398

機能：サービスにリンクされたロールを作成する
使い方：create_service_linked_role(
　　AWSServiceName＝AWSのサービス名)
戻り値：作成したロールの情報を含む辞書

スロットタイプの作成（P398～P401）

```
# フレーバーのスロットタイプを作成
flavor_slot_type = lex.put_slot_type(
  # フレーバースロットタイプの名前
  name='FlavorSlotType',
  # スロット値の設定
  enumerationValues=[
    # スロット値：vanilla
    {'value': 'vanilla'},
    # スロット値：chocolate，同義語：choc
    {'value': 'chocolate', 'synonyms': ['choc']},
    # スロット値：strawberry，同義語：berry
    {'value': 'strawberry', 'synonyms': ['berry']}
  ],
  # 値を選択する方策の設定
  # TOP_RESOLUTION：スロットタイプの中から、ユーザの言葉に近いものを返す
  valueSelectionStrategy='TOP_RESOLUTION',
  # スロットタイプのバージョンを作成する
  createVersion=True)
print('slot type:', flavor_slot_type['name'])
```

```
# フレーバーのスロットタイプを作成
flavor_slot_type = lex.put_slot_type(
  # フレーバースロットタイプの名前
  name='FlavorSlotType',
  # スロット値の設定
  enumerationValues=[
    # スロット値：vanilla
    {'value': 'vanilla'},
    # スロット値：chocolate，同義語：choc
    {'value': 'chocolate', 'synonyms': ['choc']},
    # スロット値：strawberry，同義語：berry
    {'value': 'strawberry', 'synonyms': ['berry']}
  ],
  # 値を選択する方策の設定
  # TOP_RESOLUTION：スロットタイプの中から、ユーザの言葉に近いものを返す
  valueSelectionStrategy='TOP_RESOLUTION',
  # スロットタイプのバージョンを作成する
  createVersion=True)
print('slot type:', flavor_slot_type['name'])
```

フレーバーのスロットタイプ（FlavorSlotType）

値	同義語
vanilla（バニラ）	なし
chocolate（チョコレート）	choc（チョコ）
strawberry（ストロベリー）	berry（ベリー）

容器のスロットタイプ（ContainerSlotType）

値	同義語
corn（コーン）	なし
cup（カップ）	なし

Lex　put_slot_typeメソッド　P400

機能：スロットタイプを作成する
使い方： `put_slot_type(`
`name = スロットタイプ名,`
`enumerationValues={ スロットが取り得る値 },`
`valueSelectionStrategy=値を選択する方策,`
`createVersion=True)`
戻り値：作成したスロットタイプの情報を含む辞書

・enumerationValues（スロットが取り得る値）

◆ スロットが取り得る値の記法

同義語	記法
なし	{ 'value' : 値 }
あり	{ 'value' : 値, 'synonyms' : [同義語] }

・valueSelectionStrategy（値を選択する方策）

◆ valueSelectionStrategy に指定する値

値	意味
'TOP_RESOLUTION'	スロットタイプの中から、ユーザの言葉に近いものを探す（遠い場合には何も返さない）
'ORIGINAL_VALUE'	ユーザの言葉がスロットタイプに登録した値に近い場合、そのユーザの言葉を返す

インテントの作成（P402～P406）

```
# インテントの作成
# インテント（ユーザとボットの会話の台本）
intent = lex.put_intent(
  # インテントの名前
  name='OrderIntent',
  # スロットの設定
  slots=[
    {
      # スロットの名前
      'name': 'Flavor',
      # スロットの制約：必須
      'slotConstraint': 'Required',
      # スロットタイプ
      'slotType': 'FlavorSlotType',
      # スロットタイプのバージョン
      'slotTypeVersion': '1',
      # スロットの値をユーザーから聞き出すためのプロンプト
      #  （促進するセリフ）
      'valueElicitationPrompt': {
        # セリフを表す辞書のリスト
        'messages': [{
          # セリフのタイプ：PlainText
          'contentType': 'PlainText',
          # セリフの内容
          'content': 'Vanilla, chocolate or strawberry?'
        }],
        # 聞き返す回数の上限
        'maxAttempts': 3
      }
    },
    {
      'name': 'Container',
      'slotConstraint': 'Required',
      'slotType': 'ContainerSlotType',
      'slotTypeVersion': '1',
      'valueElicitationPrompt': {
        'messages': [{
          'contentType': 'PlainText',
          'content': 'Corn or cup?'
        }],
        'maxAttempts': 3
      }
    }
  ],
  # 発話例の設定
  sampleUtterances=[
        'I want {Flavor} ice cream in {Container}',
        '{Flavor} ice cream {Container}',
        'ice cream'
  ],
  # 完了時のセリフの設定
  conclusionStatement={
    'messages': [{
      'contentType': 'PlainText',
      'content': 'OK, {Flavor} ice cream in {Container}'
    }],
  },
  # 完了時の動作
  # ReturnIntent : インテントを返す
  # CodeHook : Lambdaを呼び出す
  fulfillmentActivity={'type': 'ReturnIntent'},
  # インテントのバージョンを設定
  createVersion=True)
print('intent:', intent['name'])
```

Lex　put_intentメソッド　P403

機能：インテントを作成する
使い方：put_intent(
　　name = インテント名,
　　slots={ スロット },
　　sampleUtterances={ 発話例 },
　　conclusionStatement=完了時のセリフ,
　　fulfillmentActivity=完了時の動作)
戻り値：作成したインテントの情報を含む辞書

ボットの作成（P406～P409）

```
# ボットの作成
bot = lex.put_bot(
  # ボット名
  name='MyBot',
  # 言語と地域
  locale='en-US',
  # ボットが子供向けであるかどうか
  childDirected=False,
  # インテントの情報を含む辞書のリスト
  intents=[
    {
      # インテント名
      'intentName': 'OrderIntent',
      # インテントのバージョン
      'intentVersion': '1'
    }
  ],
  # 中止時のセリフを表す辞書のリスト
  abortStatement={
    'messages': [
      {
        'contentType': 'PlainText',
        'content': 'Please try again.'
      }
    ]
  },
  # Polly の 音声ID
  voiceId='Joanna',
  createVersion=True)
print('bot:', bot['name'])
```

```
# ボット作成の進捗表示
start = time.time()
status = ''
while status not in ['READY', 'FAILED']:
  # ボットを取得
  status = lex.get_bot(name='MyBot', versionOrAlias='1')['status']
  time.sleep(10)
  print('{:7.2f} {}'.format(time.time()-start, status))
# 作成に失敗した場合は理由を表示
if status == 'FAILED':
  print(lex.get_bot(
      name='MyBot', versionOrAlias='1')['failureReason'])

# ボットエリアスを作成
bot_alias = lex.put_bot_alias(
    name='MyBotAlias', botName='MyBot', botVersion='1')
print('bot alias:', bot_alias['name'])
```

Lex　put_botメソッド　P407

機能：ボットを作成する
使い方：put_bot(
　　name = ボット名,
　　locale=言葉と地域,
　　childDirected=False,
　　intents={インテント},
　　abortStatement=中止時のセリフ,
　　voiceId=音声ID,
　　createVersion=True)
戻り値：作成したボットの情報を含む辞書

Lex　get_botメソッド　P409

機能：ボットを取得する
使い方：get_bot(
　　name = ボット名,
　　versionOrAlias=バージョンまたはボットエリアス名)
戻り値：ボットの情報を含む辞書

Lex　put_bot_aliasメソッド　P409

機能：ボットエイリアスを作成する
使い方：put_bot_alias(
　　name = エイリアス名,
　　botName=ボット名,
　　botVersion=ボットのバージョン)
戻り値：作成したボットエイリアスの情報を含む辞書

ボットと文字で会話する（P410～P414）

```python
# boto3インポート
import boto3
# uuidインポート
import uuid
# LexRuntimeサービスクライアントを作成
lex_runtime = boto3.client('lex-runtime', 'us-east-1')
# ユーザIDを生成
user = str(uuid.uuid1())
# インテントが完了するまで続ける
state = ''
while state != 'Fulfilled':
    # ボットに文字列を送信する
    result = lex_runtime.post_text(
        botName='MyBot', botAlias='MyBotAlias',
        userId=user, inputText=input('You: '))
    # ボットの応答を表示
    print('Bot:', result['message'])
    # 会話の状態を取得
    state = result['dialogState']
# 取得したスロットの値を表示
print()
print('Flavor   :', result['slots']['Flavor'])
print('Container:', result['slots']['Container'])
```

LexRuntime　post_textメソッド　P414

機能：ボットに文字列を送信する
使い方：post_text(
　　botName＝ボット名,
　　botAlias=ボットエリアス名,
　　userId=ユーザID,
　　inputText=送信する文字列)
戻り値：ボットの応答や会話の状態を含む辞書