

使ってわかった AWSのAI

まるごと試せば視界は良好
さあ **Python** ではじめよう！

井上研一 [著]

SageMaker／Forecast／
Deep Learning AMI／
Rekognitionなどを
丁寧に解説！



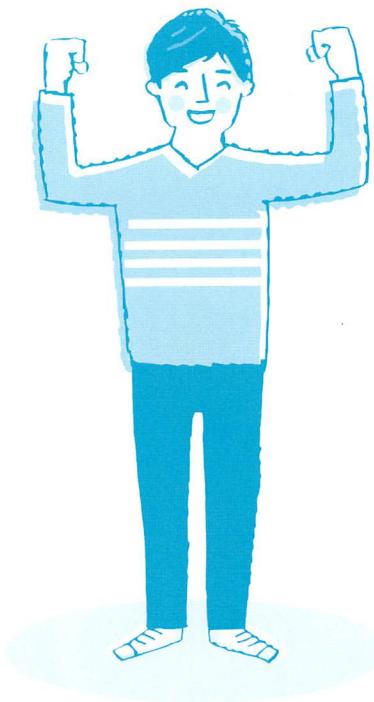
リックテレコム

本書は、AWSが提供するAIのサービスについて、その全体像をまるごと、ざっくりつかんで頂くことを目的としています。SageMaker、Forecast、Deep Learning AMI、Rekognition等々、さまざまなサービスがあり、全貌をつかむのも一苦労です。だからこそ、本書のナビゲートに沿って皆さん自身が操作し、サービスの内容を確認していくことが大切なのです。体感してこそ身に付きます。それでは、さっそくはじめましょう！

使ってわかった AWSのAI

まるごと試せば視界は良好
さあ Python ではじめよう！

井上研一 [著]

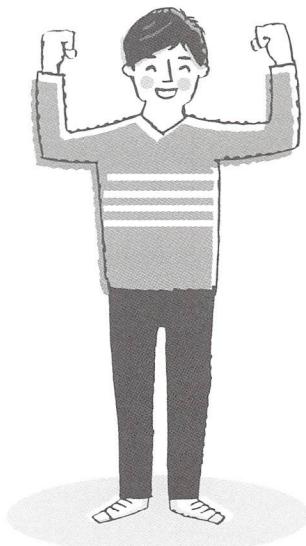


リックテレコム

使ってわかった AWS の AI

まるごと試せば視界は良好
さあ Python ではじめよう！

井上研一 [著]



リックテレコム

●ダウンロードのご案内

本書に掲載しているソースコードを、リックテレコムの Web サイトに zip 形式で圧縮した形でアップしています。本書をお買い上げの方は、ここからダウンロードしてご利用頂けます。

<http://www.ric.co.jp/book/index.html>

上記 Web サイトの左欄「総合案内」から「データダウンロード」ページに進み、「使ってわかった AWS の AI」の所から該当する zip ファイルをダウンロードしてください。その際には、以下の書籍 ID とパスワード、お客様のお名前等を入力して頂く必要がありますので、予めご了承ください。

書籍 ID : ric12461
パスワード: cai12461

本書に掲載しているソースコードは、下記の Web サイトでもご覧頂けます。

<https://inoccu.com/docs/awsai/>

●本書刊行後の補足情報

本書の刊行後、記載内容の補足や更新が必要となった場合、下記に読者フォローアップ資料を掲示する場合があります。必要に応じて参照してください。

http://www.ric.co.jp/book/contents/pdfs/12461_support.pdf

- ・本書は、2020 年 3 月時点の情報をもとにしています。本書に記載された内容は、将来予告なしに変更されることがあります。また、本書で紹介しているサービスおよびソフトウェアのバージョンアップに伴い、画面構成、操作手順等が変更され、その結果、本書の解説と一致しない部分が生じることが予想されます。予めご了承願います。
- ・本書の記述は、筆者の見解にもとづいており、Amazon Web Services, Inc. およびその関連会社とは一切の関係がありません。
- ・本書の記載内容にもとづいて行われた作業やその成果物がもたらす影響については、本書の著者、発行人、発行所、その他関係者のいずれも一切の責任を負いませんので、予めご了承願います。
- ・本書に記載されている会社名、製品名、サービス名等は、一般に各社の商標または登録商標であり、特にその旨明記がなくても本書は十分にこれらを尊重します。なお、本文中では、™、®マーク等は明記していません。

はじめに

皆さんの中には、これまで何らかの形で AWS を使ったことがある方も多いと思います。仕事でシステム構築の際に用いるだけでなく、個人的に勉強のために使っている方も多いでしょう。私自身も、クライアント企業の業務システムや、自分の会社（株式会社ビビンコ）で作っている IoT サービスなどで、EC2 や Lambda といった AWS のサービスを活用しています。

私は 2016 年から 2018 年にかけて、『初めての Watson』『ワトソンで体感する人工知能』（いずれもリックテレコム刊）という書籍を執筆しました。クラウドでの AI サービスの老舗ともいえる IBM Watson は、実際のシステム構築やセミナーの講師といった形でたくさん触れてきました。

そうした中、IBM Watson と同様、クラウド市場を牽引する AWS には、一体どのような AI サービスがあるのか？と好奇心がわいてきました。AWS には古くから Amazon Machine Learning というサービスがあり、分類や回帰のモデルを簡単に作ることができました。ただ、それ以外については、まだまだという印象が私にはありました。

しかし、ここ数年間に、AWS は矢継ぎ早にさまざまな AI サービスを追加しました。本書は、他の AI サービスをいろいろと使ってきた私が、AWS における AI サービスについて実際に手を動かして調べ、その調査結果をまとめたものです。本書を執筆している間にも次々と新しいサービスがリリースされ、それにできるだけ追随せねば…という使命感と苦労も感じましたが、それだけ AI の分野は技術の進歩が速く、AWS が全力を挙げて取り組んでいることを示しています。

また、AWS の母体である Amazon は皆さんご存じのとおり、世界一のショッピングサイトです。Amazon の内部ではさまざまな AI が使われていることは想像に難くなく、そこで生まれた予測やパーソナライズといった AI が、AWS のサービス（本書の第 3 章で取り上げる Amazon Forecast や Amazon Personalize）としても提供されています。

いまや AWS における AI サービスは多岐に渡り、全貌をつかむのも一苦労です。だからこそ、本書のナビゲートに沿って皆さん自身が手を動かし、どのようなサービスが提供されているのかを確認していくことが、実はとても大切なことなのです。私の執筆活動や講師としてのポリシーは、実際に使ってもらうことです。体感してこそ身に付きます。私が手を動かしてさまざまなサービスを試していったことを、是非、追体験してください。その体験はきっと楽しいものだと思います。

ところで、ここまで無造作に「AI サービス」という言葉を使ってきましたが、これについて少し説明します。AWS では、AI に関するサービスをまとめて「AWS での機械学習」と表していて、「AI サービス」という言葉は、AI や機械学習を深く知らなくても簡単に使えるサービスについてのみ使っているようです（本書の第 3 章）。AWS ではこの他、機械学

習について一定の知識がある人を対象に Amazon SageMaker（第4章）や、AWS Deep Learning AMI（第5章）といったサービスを提供しており、これらを「MLサービス」などと呼んでいます。本書では、ここに挙げたすべてについて、それぞれ章を設けて取り上げました。また、第1章では、機械学習やディープラーニングといった、現在のAIを構成する技術について基礎的な部分を説明しています。さらに、第2章では、AWSでの機械学習に関するサービスや、それと関係の深いデータ収集・蓄積といったサービスの概略をまとめています。第3章以降で実際に操作する際に、時折、前の章に戻って読んで頂くと理解が深まるでしょう。

本書の目的は、AIに関してAWSがどのようなサービスを提供しているか、その全体像をまるごと、ざっくりつかむことです。そのためには、サービスを実際に使ってみて、何ができるのか、どのように進めればよいのかを体感していきましょう。そうすると、視界は良好！ 適切なサービスを選択し、構築を始めることができるようになるはずです。実際に構築作業を行うと、別途AWSのリファレンスなどを参照しなければならない場合もあると思います。しかし、本書での体験があれば、リファレンスを使いこなして構築作業を進めていくことができるでしょう。

最後に、本書に取り組むにあたって注意事項を述べておきます。繰り返しになりますが、第3章以降は実際に操作してみましょう。その際、Pythonについての基本的な知識が必要となります。もちろん、パソコンも必要です。Pythonを実行する環境はWindowsでもmacOSでも構築できるので、いずれかのOSが動作するパソコンであれば基本的には問題ないでしょう（Linuxでも構築できますが、本書ではLinuxでの環境構築については説明しません）。なお、パソコン上にPythonとJupyter Notebookの動作環境を構築する際に、Anacondaという統合環境をインストールするので、それが動作する程度の性能は必要です。特に、Anacondaは5～6ギガバイト程度のディスク容量を消費するのでご留意頂けたらと思います。

それでは、本書とご自身のパソコンを並べて、さっそくはじめましょう！

著者 井上 研一



目次

はじめに	3
------------	---

第 1 章 人工知能とは何か

11

1.1 第 3 次人工知能ブーム	12
1.1.1 第 3 次人工知能ブームとは何か	12
1.1.2 人工知能とは何か	13
1.1.3 第 1 次人工知能ブーム	13
1.1.4 第 2 次人工知能ブーム	14
1.1.5 人工知能とは、ちょっと進んだ IT	15
1.2 機械学習	16
1.2.1 機械学習とは何か	16
1.2.2 機械学習の仕組み	17
1.2.3 機械学習とプログラミング	19
1.2.4 機械学習で何ができるのか	20
1.2.5 機械学習は間違うことがある	20
1.3 機械学習の代表的な手法	22
1.3.1 線形回帰	22
1.3.2 機械学習とディープラーニング	24
1.3.3 置き込みニューラルネットワーク	25

第 2 章 AWS の機械学習サービス

29

2.1 機械学習をどう使うか	30
2.1.1 機械学習を使う・作る	30

2.1.2 プログラミングでモデルを作る	30
2.1.3 Web サービスでモデルを作る	31
2.1.4 作成済みのモデルを使う	32
2.2 AWS で機械学習を使うには.....	34
2.2.1 Amazon AI.....	34
2.2.2 Amazon SageMaker	37
2.2.3 Amazon EC2 と AWS Deep Learning AMI	37
2.2.4 データレイクとデータ分析系サービス	38
2.3 機械学習をシステムで使うには	39
2.3.1 機械学習を使ったシステムとは	39
2.3.2 機械学習のワークフロー	40
2.4 AWS で機械学習のワークフローを作る	43
2.4.1 データの収集と蓄積	43
2.4.2 データの分析と前処理	44
2.4.3 どの機械学習サービスを使うか	45
2.4.4 システムへの組み込み	46
2.5 AWS のアカウントを作る	47
2.5.1 アカウントの作成手順.....	47

第 3 章 AI サービス

53

3.1 AI サービスとは	54
3.1.1 AI サービスとコグニティブサービス	54
3.1.2 AI サービスの概要.....	56
3.1.3 AI サービスの課金について	57
3.2 SDK の使用準備	58
3.2.1 AI サービスと AWS SDK	58
3.2.2 Jupyter Notebook の導入 (Windows)	58
3.2.3 Jupyter Notebook の導入 (macOS)	60
3.2.4 IAM でユーザーの追加と権限の付与を行う	61
3.2.5 認証情報の保存.....	65
3.2.6 既存のユーザーへの権限の付与	66

3.2.7	Jupyter Notebook で AWS SDK for Python を使う	68
3.2.8	S3 バケットの作成とファイルのアップロード	70
3.3	Amazon Rekognition.....	73
3.3.1	Amazon Rekognition とは.....	73
3.3.2	画像を用いたモノの認識	74
3.3.3	画像を用いた顔の認識.....	76
3.4	Amazon Comprehend	79
3.4.1	Amazon Comprehend とは	79
3.4.2	自然言語を識別する.....	80
3.4.3	エンティティを抽出する	81
3.5	Amazon Textract	83
3.5.1	Amazon Textract とは	83
3.5.2	英語の書類画像からデータを取得する	84
3.6	Amazon Translate	87
3.6.1	Amazon Translate とは	87
3.6.2	カスタム用語を使わない翻訳	88
3.6.3	カスタム用語を使った翻訳	89
3.7	Amazon Transcribe	93
3.7.1	Amazon Transcribe とは	93
3.7.2	日本語の音声ファイルの認識	94
3.8	Amazon Polly.....	99
3.8.1	Amazon Polly とは	99
3.8.2	日本語のテキストを音声に変換する	100
3.9	Amazon Lex	103
3.9.1	Amazon Lex とは	103
3.9.2	チャットボットを実現するための技術	103
3.9.3	AWS Lambda との連携	104
3.9.4	LexModelBuildingService と LexRuntimeService	104
3.9.5	Lex における会話の組み立て	105
3.9.6	BookTrip サンプルで動作を確認する	106
3.9.7	Response の追加.....	110
3.9.8	チャットボットの公開.....	112

3.10	Amazon Forecast	117
3.10.1	Amazon Forecast とは	117
3.10.2	Amazon Forecast で使用するデータセット	117
3.10.3	Amazon Forecast へのデータのインポート (データセットグループの作成)	119
3.10.4	予測子 (Predictor) の作成.....	127
3.10.5	予測 (Forecast) の作成と結果の確認	130
3.10.6	予測子メトリクスの参照.....	136
3.11	Amazon Personalize	139
3.11.1	Amazon Personalize とは	139
3.11.2	Amazon Personalize で使用するデータセット	139
3.11.3	Amazon Personalize へのデータのインポート (データセットグループの作成)	141
3.11.4	ソリューション (ソリューションバージョン) の作成.....	149
3.11.5	キャンペーンの作成.....	152
3.11.6	レコメンデーションの取得	153

第 4 章 Amazon SageMaker

157

4.1	SageMaker とは何か	158
4.1.1	SageMaker でできること	158
4.1.2	SageMaker の使用開始	158
4.1.3	SageMaker の機能と画面構成	159
4.1.4	Ground Truth (ラベリング)	160
4.1.5	ノートブック	161
4.1.6	トレーニング (学習)	161
4.1.7	推論	162
4.1.8	S3 バケットの準備と IAM ロールの作成	162
4.1.9	SageMaker の課金体系	165
4.2	SageMaker のノートブックを使う	166
4.2.1	SageMaker のノートブック	166
4.2.2	ノートブックインスタンスの作成	166
4.2.3	ノートブックを使う	169

4.2.4 ノートブックインスタンスの停止	171
4.3 SageMaker の組み込みアルゴリズムでモデルを作る	172
4.3.1 SageMaker の組み込みアルゴリズムを用いたモデルの作成	172
4.3.2 SageMaker の組み込みアルゴリズムとは	175
4.3.3 トレーニングデータの準備	175
4.3.4 トレーニングデータの加工	178
4.3.5 ハイパープラメータの設定と S3 へのデータのアップロード	180
4.3.6 トレーニングジョブの作成	184
4.3.7 精度の評価	186
4.3.8 モデルの作成	189
4.3.9 エンドポイント設定の作成	191
4.3.10 エンドポイントの作成	193
4.3.11 エンドポイントの動作確認	196
4.3.12 エンドポイントの削除	198
4.3.13 バッチ変換ジョブ	198
4.4 SageMaker のさまざまな組み込みアルゴリズム	201
4.4.1 組み込みアルゴリズムのカタログ	201
4.4.2 線形学習 (LinearLearner)	204
4.4.3 XGBoost	205
4.4.4 主成分分析 (PCA)	208
4.4.5 K-Means	209
4.5 SageMaker Studio と SageMaker Autopilot	213
4.5.1 SageMaker Studio	213
4.5.2 SageMaker Autopilot	216

第 5 章 AWS Deep Learning AMI 223

5.1 EC2 環境でのディープラーニング	224
5.1.1 より柔軟な環境が必要になるケース	224
5.1.2 EC2 と AMI	224
5.1.3 DLAMI と基本 DLAMI	226

5.2 DLAMI を使う	227
5.2.1 AMI による EC2 インスタンスの構築	227
5.2.2 DLAMI の Jupyter Notebook を開く	231
5.2.3 TensorFlow と Keras によるモデル構築.....	234
5.2.4 構築したモデルのデプロイ	238
5.2.5 EC2 インスタンスの停止または終了.....	243
索引	244

第 1 章

人工知能とは何か

AWS の機械学習サービスを使う前に、そもそも人工知能（AI）とは何か、人工知能と機械学習の関係はどうなっているのかを理解しておきましょう。

一般的なプログラミングと機械学習の違いについて知っておくことも重要です。また、機械学習とディープラーニングの基本的な内容についても説明していますので、第 3 章以降で実際に機械学習サービスを使った後で、再び本章に戻って頂くのも良いと思います。

1.1

第3次人工知能ブーム

1.1.1

第3次人工知能ブームとは何か

人工知能（AI：Artificial Intelligence）ブームが始まって何年経つでしょうか。詳しくは後述しますが、現在のブームは「第3次人工知能ブーム」といわれています。その起点をどこに置くかは難しいですが、例えば、ディープラーニングが登場して画像認識の精度（画像に何が写っているかを当てられる割合）が人間を超えたのが2015年、Googleのグループ会社であるDeepMind社が「AlphaGo（アルファ碁）」で囲碁のトップ棋士に勝利したのが2016年です。一方で、IBMのWatsonが「Jeopardy!（ジェパディ！）」というアメリカの有名なクイズ番組に登場し、人間のチャンピオンに勝利したのが2011年なので、現在のブームはもっと早くから始まっていたという見方もあるでしょう。

ブームの起点をどこに置くかはさておき、AlphaGoが勝利を収めたのを機に、人工知能がより一層大きな注目を集めていることは間違ひありません。

人工知能の活用領域はとても幅広く、いくつか例を挙げると、不審な人物の検出や、車の自動運転、絵画などの作品の創作が挙げられます。また、Watsonが特殊な白血病に効く薬を発見したことでも記憶に新しいところです。さらに、Amazonの「Echo（エコー）」、「Echo Dot（エコードット）」、「Echo Show（エコーショー）」をはじめとするスマートスピーカーは、人間が話しかけた内容を理解し、家電の操作やオンライン購入した商品の配送状況確認など、さまざまなことを行ってくれます。スマートスピーカーはAmazonだけでなく、GoogleやApple、LINEなどからも発売されています。

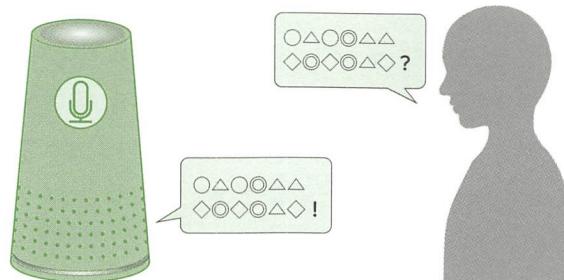


図 1-1-1 スマートスピーカー

自動運転にせよ、薬の発見^{*1}にせよ、スマートスピーカーにせよ、従来のITではなく、人工知能だからこそできることといえます。

1.1.2 人工知能とは何か

人工知能とは何でしょうか。この問いに答えるのはとても難しいです。人工知能の研究者がどのように人工知能を捉えているか、説明しているかを見ても、人それぞれとしか言いようがありません。「人工知能とは何か」だけでなく、「人工知能をどうやって作るか」、「そもそも知能とは何か」など、問い合わせを呼び、なかなか答えを見出せないのではないかと思います。

先ほど、現在の人工知能ブームは第3次、つまり3回目のブームであるといいました。「人工知能」に決まった定義があるのなら、第1次、第2次という過去の2回のブームと共に通するテーマがあるはずです。それは何なのでしょうか。

1.1.3 第1次人工知能ブーム

第1次人工知能ブームは1950年代に始まりました。1956年にアメリカで開催されたダートマス会議で「AI (Artificial Intelligence)」という言葉が生まれたといわれています。ちなみに、黎明期のコンピュータの代表格であるENIAC^{*2}が登場したのが1946年ですから、わずかその10年後には人工知能がブームになっていたわけであり、コンピュータの歴史と人工知能の歴史は同じ道を歩んでいるのではないかと思います。

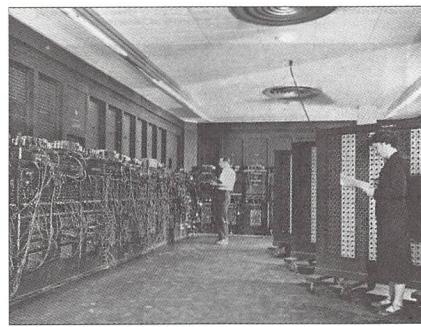


図1-1-2 ENIAC^{*3}

*1 Watsonは新薬を開発したわけではなく、特殊な白血病に効き目があると思われる薬の論文を見つけたのです。論文は膨大にあり、1人の人間がすべてを読み通すことは困難なので、Watsonの存在価値があったのです。

*2 ENIAC (Electronic Numerical Integrator and Computer、エニアック)は、アメリカで開発された黎明期のコンピュータです。1946年2月14日に完成し、翌日から弾道計算などの用途で使用が開始されました。

*3 Wikipedia Commons (<https://commons.wikimedia.org/wiki/File:Eniac.jpg>)

第1次ブームでの人工知能は、「探索」と「推論」でした。探索は、いわば力尽くで迷路を解くような技術です。迷路を解くには、通りうるすべてのルートをしらみつぶしに試していきます^{*4}。そうすれば、いずれゴールにたどり着きます。実際には探索はもっと賢い方法を採りますが^{*5}、基本的にコンピュータは単純なことを高速にやり続けることが最も得意なので、探索はそれを活かした技術といえるでしょう。一方、推論は、例えば「すべての人間は死ぬ」という大前提と「ソクラテスは人間である」という小前提から、「ソクラテスは死ぬ」という結論を得るようなもので、これは数学的に解くことが可能です。こちらも、やはりコンピュータが得意なことをそのまま活かした技術といえます。つまり、探索にしろ、推論にしろ、コンピュータがもともと得意とする領域で力を発揮しただけといえるのですが、それが当時の人工知能だったのです。

しかし、探索も推論も、今ではコンピュータ科学の基礎として扱われるようになっており、これらを人工知能と呼ぶ人はいません。

1.1.4 第2次人工知能ブーム

第2次人工知能ブームは1980年代に起きています。第1次ブームから約30年後、また現在の第3次ブームの約30年前ということで、人工知能ブームの周期のようなを感じます。

第2次ブームは、エキスパートシステムや、日本では通商産業省（現・経済産業省）が主導した第5世代コンピュータが発端といえるでしょう。第1次ブームでは実際のビジネスなどに用いるという視点が乏しかったので、第2次ブームでは実用になるものが求められました。エキスパートシステムの事例として、伝染病患者に薬を処方する Mycin（マイシン）や住宅ローンの審査などを挙げることができます。

当時の人工知能に複雑な事柄を処理させるためには、「If ~ Then ~（もし～だったら、～する）」の形で、人間があらかじめ知識をきちんと整理して人工知能に教える必要がありました。それがエキスパートシステムでは極めて大変だったため、ブームは終焉を迎えてしまいます。

第5世代コンピュータも芳しい成果があったとは思われていませんが、「日本語をどのように処理するか」という自然言語に関する技術が進んだことは間違ひありません。自然言語処理は、今でも人工知能の研究における重要な分野ですが、その源流はこの頃にあったといえるでしょう。

*4 これをしらみつぶし探索や、力任せ探索と呼ぶこともあります。

*5 しらみつぶし探索は、迷路のすべてのルート（これを探索空間といいます）をあたってゴール（解）を見つけ出しますが、その際、探索空間の構造に関する知識を活用して探索にかかる時間を短縮することができます。

COLUMN ディープラーニングの潮流

最近、「ディープラーニング」という言葉を見聞きすることが多くなりました。主に画像認識の分野で使われているディープラーニングは、人間の頭脳にあるニューロンの仕組みを活用した技術で、今、最も注目されているといっても過言ではないでしょう。しかし、ディープラーニングの研究は今回のブームで始まったわけではありません。

先ほど説明した第1次ブームのときには既に、「パーセプトロン」という人工ニューロンを活用した基礎的な技術がありました。第2次ブームではさらに発展して、現在のディープラーニングによる画像認識技術と似た「ネオコグニトロン」という理論が発表されました。しかし、この理論を本格的に活用するために必要となるコンピュータの性能や、膨大な「トレーニングデータ（後述）」が出揃うには、第3次ブームを待たなければなりませんでした。

1.1.5 人工知能とは、ちょっと進んだIT

このように第1次、第2次の人工知能ブームを振り返ったとき、人工知能の歴史はコンピュータの歴史ではないかと感じます。第1次ブームの探索や推論は、今ではコンピュータ科学の基礎となっており、また、日本で仮名漢字変換が実用化されたのは、ちょうど第2次ブームの自然言語処理の研究と同期しています。

そう考えると、人工知能はその時代に合わせた「ちょっと進んだIT」といえるのではないかでしょうか。少なくとも筆者はそのように捉えています。まず間違いなく、第3次ブームでもてはやされている画像認識や自然言語理解は、数年後には当たり前の技術になっているでしょう。その頃、第4次や第5次のブームがあるのかはわかりませんが、より進んだ技術を指して「人工知能」と呼んでいるのではないかと思うのです。

本書でこれから説明する Amazon Web Services (AWS) では、人工知能という言葉をサービスの紹介であまり使っていないようです。AWS では、機械学習を行う環境として「Amazon SageMaker」を提供したり、音声認識や自然言語認識といった人工知能を活用した機能を「AI サービス (Amazon AI)」として提供したりしています。機械学習やディープラーニングといった技術が人工知能と呼ばれなくなったとしても、重要なサービスとして提供され続けることは間違いないでしょう。

1.2

機械学習

1.2.1 機械学習とは何か

前節で述べたように人工知能がその時代の「ちょっと進んだIT」だとすると、本書執筆時点におけるその技術とは、一体、何でしょうか。それは、第3次人工知能ブームの鍵を握る技術である「機械学習」と「ディープラーニング」といって良いでしょう。第3章で詳しく説明しますが、Amazon AI を使うと、画像に何が写っているかや、顔が写っている画像から、その表情が笑顔なのかどうかなどの認識結果を得ることができます。

なぜ、このようなことができるのでしょうか。それは、AWSが膨大な数の画像を使って、「この画像には猫が写っている」とか「この顔の表情は笑顔である」というように画像に何が写っているかを認識するために、あらかじめコンピュータ（機械）に学習させているからです。コンピュータは自ら、猫（または笑顔）が写っている画像から共通点を見つけ出します。このようにコンピュータが、人間が与えた教材（データ）を用いて自ら学習しているので、「機械学習」と呼ばれているのです。

多くの場合、機械学習には、膨大な数のデータと極めて高性能なコンピュータが必要です。画像データはインターネット上に膨大に存在しますが、それらの画像一つ一つに、コンピュータに認識させたい（例えば「猫」や「笑顔」といった）答え（これを「教師データ」といいます）を与えるのは大変です。しかし、Amazon AIではあらかじめ機械学習が行われているので、データの準備や機械学習を利用者が行う必要はありません。

プログラマが機械学習を使うための方法として最も身近なものは、ここで紹介したAmazon AIのほか、IBMのWatson、MicrosoftのAzure Cognitive Servicesのようなクラウド上のAIサービスを活用する方法です。こうしたサービスを使えば、機械学習の成果だけを活用して、自分のプログラムやアプリに人工知能的な機能を搭載することができます。

ところで、自分で準備したデータを使って、独特な認識を行うことはできないのでしょうか。AIサービスの一種である画像認識サービスでは（Amazon AIではAmazon Rekognitionという名前で提供されています）、犬や猫といった一般的なモノ（犬や猫はモノではありませんが…）の認識は可能です。しかし、例えば企業内の情報システムにおいて、その企業の独自の製品や特殊な部品、状態（品質上の問題や不具合、鉄塔に付着したサビの状況など）を画像認識させることはできません。そうした要件に対応するためには、企業が自らデータを準備し、機械学習を行

う必要があります。このような独特な認識を行うための機能・サービスとして、画像認識を行う Amazon Rekognition のカスタムラベル機能や、Amazon SageMaker、Amazon EC2 (Elastic Compute Cloud)^{*6} の AWS Deep Learning AMI (Amazon Machine Images)^{*7} などが提供されています。

COLUMN AWS 以外の機械学習サービス

クラウド上のAIサービスはAWS以外の企業も提供しています。例えばIBMのWatsonやWatson Machine Learning、MicrosoftのAzure Cognitive ServicesやAzure Machine Learningのほか、Googleや富士通などのサービスが挙げられます。その中には、WatsonのVisual Recognition(画像認識サービス)のようにカスタムモデルを簡単に作成できるサービスもあります。

1.2.2 機械学習の仕組み

機械学習の仕組みをもう少し詳しく見てみましょう。画像認識を行う Amazon Rekognition や、テキストを音声に変える Amazon Polly を使って、自分のデータを認識させるだけの場合は、機械学習の仕組みについてそれほど意識する必要はありません。しかし、カスタムモデルを作成するために Amazon SageMaker や AWS Deep Learning AMI を使う場合は、知っておいた方が良いでしょう。本書では、第4章と第5章でこれらの内容を取り上げますので、その際に、本項をあらためて読み返しても良いかもしれません。

*6 Amazon EC2 は、AWS で提供されている仮想サーバーのサービスです。

*7 AWS Deep Learning AMI は、Amazon EC2 の自分の仮想サーバー上に、ディープラーニングを行うための環境を簡単に構築できるサービスです。

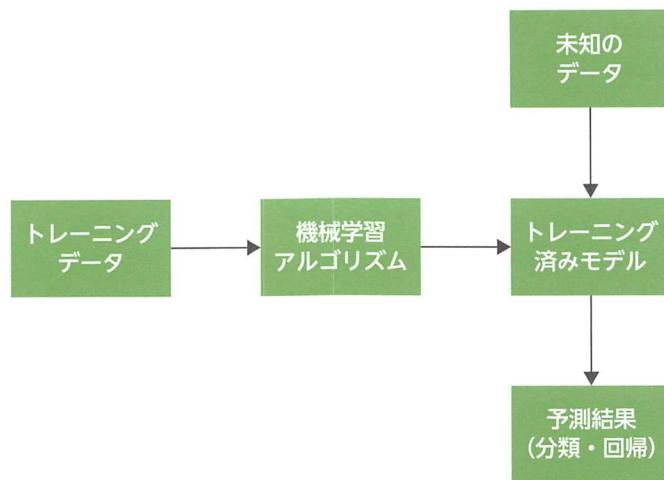


図 1-2-1 機械学習の仕組み

機械学習を簡単に図解すると、図 1-2-1 のようになります。機械学習では、トレーニングデータを元に、さまざまなアルゴリズムを用いてコンピュータ（機械）に学習を行わせます。学習するためのアルゴリズムは、私たちがあらかじめ作成しておかなければなりません。このとき作成したアルゴリズムのことを「モデル」といいます。

作成したモデルは、そのままでは実用的な働きをしないので、トレーニングデータを用いてモデルをトレーニングする必要があります。それが完了したモデルを「トレーニング済みモデル」といいます。

例えば、ある画像に犬が写っていることを認識するモデルを作りたい場合は、画像認識に適したアルゴリズムを使ってモデルを作成します。次に、あらかじめ準備しておいたトレーニングデータ（例えば、犬やそれ以外の動物などが写った画像データ）を用いて、モデルのトレーニングを行います。トレーニングデータは、一般的には数千枚や数万枚といった大量の画像と、それが何かを示すラベル（教師データ）です。そうした大量のトレーニングデータを用いて、犬の画像に共通する特徴などを見出し、モデルのさまざまなパラメータを調整してトレーニング済みモデルを作り出します。この一連の作業を「学習（トレーニング）フェーズ」といいます。

トレーニング済みモデルは、トレーニングデータが適切であれば相応の結果を返すようになります。例えば、犬の画像を与えたたら、犬が写っているという結果（ラベル）を返してくれたりするわけです。これを学習フェーズに対して「推論フェーズ」といいます。

なお、精度の高い（トレーニング済みの）モデルがすぐにできるとは限らないので、アルゴリズムを見直したり、トレーニングデータを増やしたりするなどの工夫を重ねて、徐々に精度の高いモデルに仕上げていく必要があります。

第3章で説明する Amazon Rekognition という画像認識のための AI サービスでは、API で画

像データを入力すると、その画像に犬が写っているといった結果を出力します（図 1-2-2）。しかも、なかなか高い精度で正しく回答してくれます。これは、優れたアルゴリズムと膨大なデータを用いたトレーニング済みのモデルが、AWS 上でサービスとして提供されているからです。Amazon AI サービスでは、このようなトレーニング済みモデルが提供されているので、原則として私たちがアルゴリズムを考えたり、トレーニングデータを準備したりする必要はありません。

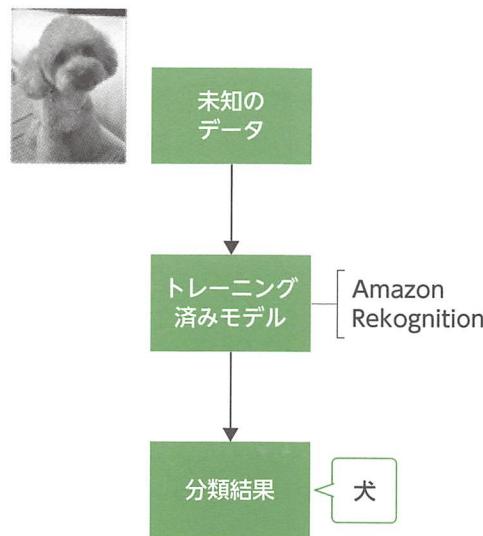


図 1-2-2 トレーニング済みモデルの活用

1.2.3 機械学習とプログラミング

従来からのプログラミングと機械学習は何が違うのか、気になるかもしれません。その違いは、問題（例えば、ある画像に犬が写っていると判定すること）の解法を誰が作るか、という点です。従来からのプログラミングでは人間自身が解法を考え、プログラミング言語を使ってプログラムを作成します。一方、機械学習では、コンピュータがアルゴリズムとトレーニングデータを用いて学習し、解法を得ていきます。多くの場合、機械学習でもプログラミングが必要になりますが、それは、機械学習アルゴリズムを使ってモデルを実装したり、トレーニングデータを使って学習フェーズを実行したりするためです。また、学習済みモデルを活用（推論）する際にもプログラミングは必要となります。

1.2.4 機械学習で何ができるのか

ここまで、「機械学習とはどのようなものか」を説明してきました。次に、「機械学習で何ができるか」について説明します。

機械学習でできることの1つは、「分類 (Classification)」です。AWSのAIサービスで提供されている、画像や音声、自然言語などの認識は、この分類の手法を用いて実現しています。認識とは、ある事象を、自分の知っている事柄のうち最も近いものに分類することです。例えば画像認識においては、コンピュータの画像は画素ごとの色情報として数値化されるので、その数値データから得られる特徴と何が写っているかの情報（教師データ）を紐付けて学習すれば、分類できるようになります。同様に、音声は波形で表現して数値化することができますし、自然言語のテキストは単語ごとに数値のラベルを付けるなどして数値化します。扱うデータが何であっても数値データに変換することができれば、分類を行うためのモデルを作成できます。

もう1つは、「回帰 (Regression)」です。活用例として、売上予測や、機械の消耗品交換時期の予測が挙げられます。回帰では、売上や消耗品の消費量に関係しそうなデータと、過去の実績値から、モデルを作成します。回帰も分類と同様に、数値化されたデータをもとに結果を返します。ただ、分類では、犬か猫かといったあらかじめ定義しておいた事柄のうち、どれに最も当てはまるかという結果を返すのに対し、回帰では、売上がいくらになりそうかといった数値そのものを返すという違いがあります。

Amazon AIをはじめとするAWSの機械学習に関するサービスを使いこなすには、分類と回帰の違いをきちんと理解しておくことが重要です。

1.2.5 機械学習は間違うことがある

機械学習を使うと、トレーニングデータを準備して分類や回帰を行うモデルを作成できますが、注意すべき点が1つあります。それは、機械学習によるモデルが出す分類や回帰は、間違った結果を返す場合があることです。前述したように、いわゆるプログラミングでは、人間が問題（要件）に対する解法を考えて、プログラムを作成します。コンピュータはプログラムに書かれているとおりに処理を行うので、解法そのものが誤っていなければ、間違った結果を出すことはありません。解法が誤っている場合は設計上のバグとして、また、解法どおりにプログラミングされていなければプログラム上のバグとして、改修の対象になります。つまり、コンピュータプログラムでは、間違った結果はバグであり、改修すれば間違わないようになるのです。

一方、機械学習のモデルでは、犬の写真を猫と分類するような間違いが起きても、それはバグではありません。ただ、分類や回帰の精度が低いだけです。

もちろん、精度を高めるためにトレーニングデータの量や質を改善したり、機械学習のアルゴリズムを見直したりすることはできますが、絶対に間違わないようなモデルを作ることはできないのです。

1.3

機械学習の代表的な手法

1.3.1 線形回帰

機械学習の代表的な手法として、「線形回帰」が挙げられます。線形回帰は、売上金額などの数値を予測するための手法です。ある変数 x が大きくなれば、別の変数 y が大きく（または小さく）なるという関係を活かしてモデルを作成します。ここで、ある変数 x を説明変数（または独立変数）、説明変数の変化によって値が変化する変数 y を目的変数（または従属変数）といいます。

図 1-3-1 のグラフを見てください。

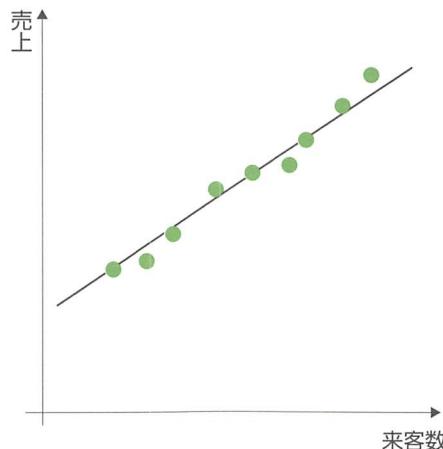


図 1-3-1 線形回帰

この図では、1日ごとの来客数と売上の関係を点でプロットしています。線形回帰でモデルを作成すると、ばらついた点のちょうど真ん中あたりに直線を描くことができます。この直線は右肩上がりになっており、来客数が多いほど売上が大きくなることを意味しています。先ほど説明した言葉で表すと、来客数が説明変数、売上が目的変数となります。

このような直線は、 $y = ax + b$ という式で示すことができます。ここで a は直線の傾き、 b は切片を示します。機械学習では、 $y = w_0 + w_1x$ という式で示すことが多いので、ここでもそのようにします。 w_0 は切片 b と同じ、 w_1 は傾き a と同じことを示しています。 w_1 のことを重み（ウェイト）ともいいます。この重みと切片の値を求めることが、機械学習における学習フェーズの

中身です。つまり、 $y = w_0 + w_1x$ が線形回帰というアルゴリズムを用いたモデルであり、トレーニングデータを用いて重み w_1 と切片 w_0 の値を求めたものがトレーニング済みモデルとなります。では、具体的にどのようにして求めるのでしょうか。

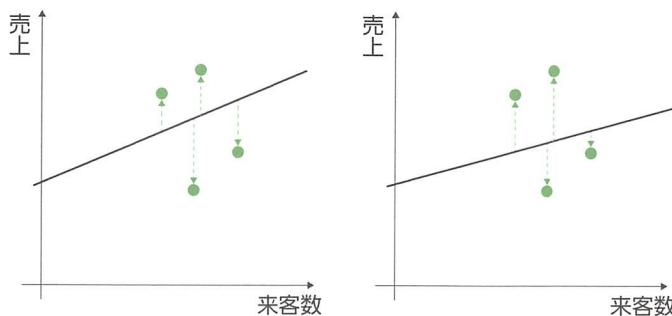


図 1-3-2 線形回帰による学習

この左右の図で、点がプロットされている位置は同じですが、直線の位置が異なっています。直線の位置が異なるということは、重みと切片の値が異なることを意味します。左図の直線は右図のそれと比べると、点との距離が均等になっているように見えます。左図において、もう少し切片の値を小さくして直線の位置を下げると、点との距離はより均等になるかもしれません。

直線と点との距離は、誤差です。線形回帰によるモデルに来客数の値(これが x です)を入れた際、予測結果として返される売上の値 (y) は直線上にある値ですから、実測値である点との誤差が大きいということは、それだけ予測精度が低いことになります。この誤差の計算方法として、平均二乗誤差を使います。平均二乗誤差は、直線とそれぞれの点との誤差を二乗したものを加算し、その平均をとったものです。誤差を二乗する理由は、プラスの誤差とマイナスの誤差がお互いを打ち消し合って、平均値がゼロになるのを避けるためです。誤差ができるだけ小さくなるような重みと切片の値を求めることができれば、トレーニング済みモデルの精度は高くなります。

ここまで説明では、売上を予測するための目的変数は来客数 1 つだけですが、現実には、設定した価格や他店の動きなどさまざまな要因があるでしょう。線形回帰では、目的変数が複数あっても構いません。ただ、目的変数が増えれば増えるほど、誤差を最小化する計算は複雑になっていきます。第 4 章で紹介する Amazon SageMaker では、この複雑な計算を簡単に実行するために、線形回帰が組み込みアルゴリズムとして提供されています^{*8}。なお、線形回帰は広く用いられている手法ですが、データの特性によっては線形回帰が向かないこともあります。そうした場合は、線形回帰とは別のアルゴリズムを使用する必要があります。Amazon SageMaker の組み込みアルゴリズムにはさまざまなものが用意されていますので、第 4 章を参照してみてください。

*8 線形回帰による予測は、Amazon SageMaker の組み込みアルゴリズム以外にも Python の機械学習ライブラリである scikit-learn などで提供されているほか、Excel でも使用可能です。

COLUMN**ロジスティック回帰**

機械学習でのわかりやすい例として、「アヤメの花の品種分類」がよく取り上げられます。アヤメの花の品種分類では、「花びらの長さ」と「がくの長さ」によって、例えばAという品種なのかBという品種なのかを分類します。

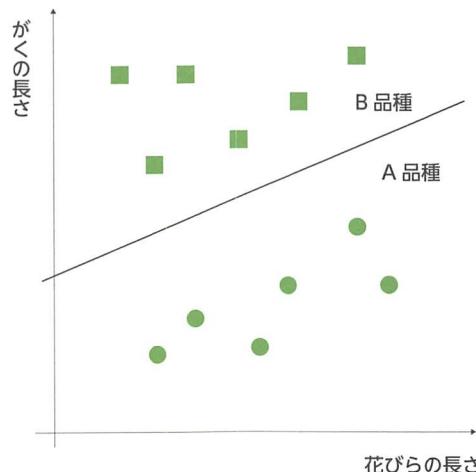


図 1-3-3 アヤメの花の品種分類

図1-3-3のように、アヤメの花の「花びらの長さ」と「がくの長さ」について、A品種を●で、B品種を■でグラフ上に表した場合、その間には1本の直線を引くことができます。このとき、それぞれの●から直線までの距離の合計と、それぞれの■から直線までの距離の合計が等しくなるように直線を引いています。こうすることで、新たに測定したアヤメの花の花びらの長さとがくの長さが、直線より下側に位置づけられれば、その花はA品種に分類することができるわけです。

この手法は、本文で説明した線形回帰を分類問題に応用した「ロジスティック回帰」というものです。第4章で説明するAmazon SageMakerでは、ロジスティック回帰は線形学習という組み込みアルゴリズムで提供されています。

1.3.2 機械学習とディープラーニング

機械学習とともに現在の人工知能ブームを生み出している技術が、ディープラーニングです。機械学習という言葉は聞いたことがないけれど、ディープラーニングなら聞いたことがある、という方もいらっしゃるかもしれません。それほど、ディープラーニングは現在の人工知能技術の代名詞になっています。

機械学習とディープラーニングは並列の関係ではなく、包含関係にあります。ディープラーニングは機械学習を行うための1つの技術に過ぎず、機械学習という大きなジャンルの一部であるといった方が妥当です。しかし、ディープラーニングの技術があるからこそ、学習モデルの精度が圧倒的に高まつたのは事実で、やはりブームの立役者といって間違いないでしょう。

一般的な（ディープラーニングを用いない）機械学習では、トレーニングデータから共通点などを学び取る方法として統計学的な手法が用いられます。そのため、人工知能技術と数学（統計学）は親和性が非常に高いです。ディープラーニングでも数学は必要ですが、人間の脳の仕組みの一部（ニューロン）の模倣から生まれた技術が用いられています。また、画像認識にディープラーニングを用いる際は、人間の目（視覚）の仕組みも参考にされています。



図 1-3-4 Amazon Rekognition (<https://aws.amazon.com/jp/rekognition/>)

Amazon AIでもディープラーニングは導入されています。例えば、画像認識サービスであるAmazon Rekognitionには、「深層学習に基づくイメージ認識サービス」という説明が付されています。深層学習とディープラーニングは同じことですので、少なくともAmazon Rekognitionにはディープラーニングが導入されているといえます。

1.3.3 畳み込みニューラルネットワーク

前述したように、機械学習の一手法としてディープラーニングがありますが、そのディープラーニングの中にも、さらにいくつかの手法があります。それらのうち最も広く使われているのが、主に画像認識に用いられる畳み込みニューラルネットワークです。

まず、ディープラーニングを構成する基礎技術であるニューラルネットワークについて簡単に説明しておきます。ニューラルネットワークは、図1-3-5のように示されることが多いです。図中の○が一つ一つのニューロンであり、そのニューロンが組み合わさってネットワークのようになっているので、ニューラルネットワークと呼ばれています。

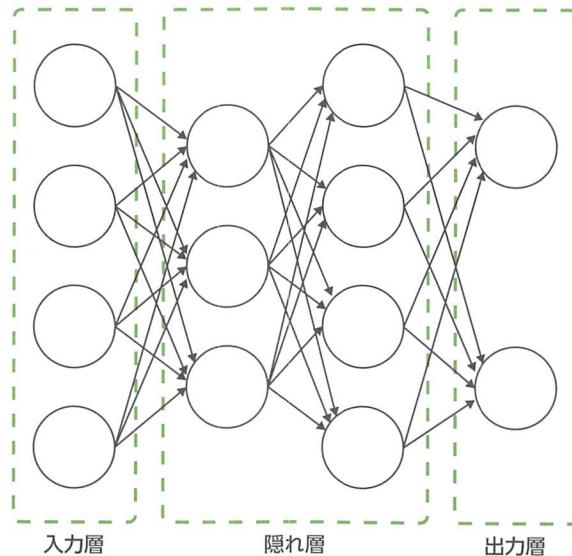


図 1-3-5 ニューラルネットワークの例

一つ一つのニューロンには、重み（ウエイト）と閾値（バイアス）という値があります。入力層のニューロンに、ある値が入力されると、それぞれのニューロンは自らの重みと閾値の値に従って計算を行い、次のニューロンに値を伝達します。ここで重みは入力値に掛ける値、閾値は次のニューロンに値をどう与えるかを決める値です。例えば、あるニューロンに(1, 2)という入力が行われ、その重みが(3, 1)だとすると、 $1 \times 3 + 2 \times 1$ という計算が行われ、計算結果は5になります。そのニューロンの閾値が6だとすると、 $5 < 6$ なので、次のニューロンへの値の伝達は抑制されます。

このような処理が一つ一つのニューロンで行われます。その際、重みと閾値の値がニューロンごとに異なると、そのニューラルネットワークに同じ値を入力しても得られる結果は大きく異なることになります。なお、ニューラルネットワークの構造は人間が作りますが、各ニューロンの重みと閾値の値を決めるのは機械学習の学習フェーズです。使用するアルゴリズムが線形回帰かニューラルネットワークかという違いはあるものの、学習フェーズでは同じようなことを行う、ということがわかるでしょう。機械学習は、人間が作成したモデルの重みと閾値（切片）の値（これをパラメータといいます）を求める作業を、トレーニングデータをもとに機械（コンピュータ）に自ら行わせる技術といえます。

さて、画像を扱う畳み込みニューラルネットワークでは、上記のようなニューラルネットワークの入力層に与える値を画像から取り出す処理を行います。図 1-3-6 のように、与えられた画像データについて畳み込み層とプーリング層を繰り返し処理し（特微量抽出）、そこで得られたデータ（特微量）をニューラルネットワーク（全結合層）の入力値とします。全結合層では、入力さ

れた値が何に分類できるか（もしくはどのような値を予測できるか）を推論します。このニューラルネットワーク全体の構造が、画像に何が写っているかを認識するものであった場合、ニューラルネットワークの出力層からは「人間が写っている」とか「犬が写っている」といった回答が得られます。

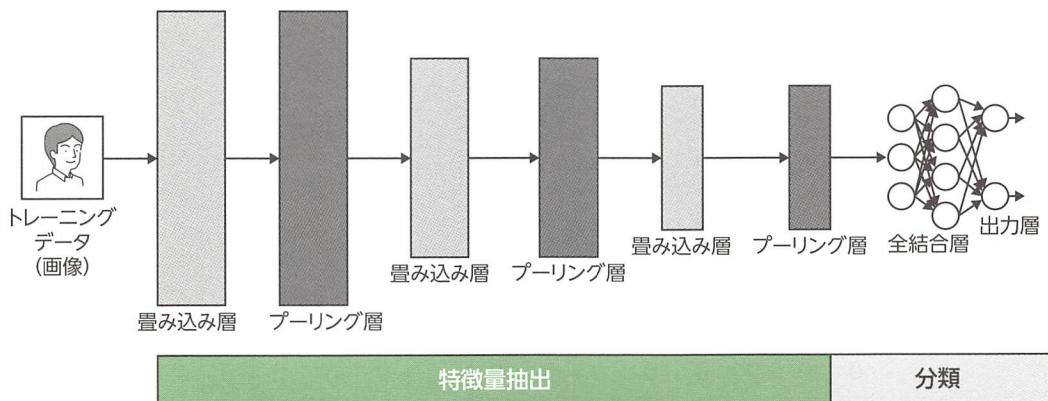


図 1-3-6 畠み込みニューラルネットワーク

では、畠み込み層とプーリング層は何を行っているのでしょうか。コンピュータ上の画像は、図 1-3-7 の左上にある「1」という数字のように、ピクセルごとの色情報（ここでは白黒画像のため 0 か 1。カラー画像の場合は赤・緑・青それぞれの 0 ~ 255 の値）の集合です。まず、畠み込み層では、画像を例えば 3×3 ピクセルずつフィルタにかける処理を行い、特徴マップを作成します。フィルタは、縦方向の直線に強く反応するものや、色調の変化に強く反応するものなどさまざまであり、そうした切り口で画像の特徴をあぶり出していくきます。

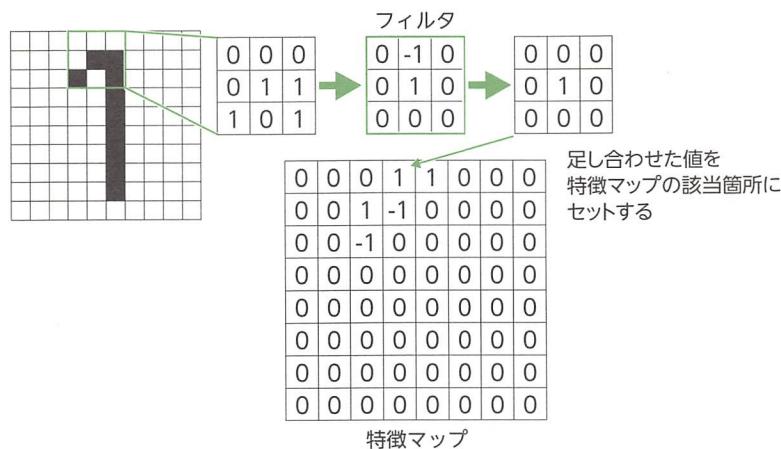


図 1-3-7 畠み込み層の処理

次にプーリング層では、特徴マップのサイズを縮小します。図1-3-8は、MaxPoolingという方法で特徴マップを 2×2 の領域に区切り、各領域の最大値をプーリング層の出力とします。プーリング層で処理を行うことにより、画像内で絵柄の位置が多少変動してもそれを吸収できるようになります。データ量が小さくなるので、学習フェーズにかかる時間を短縮できるというメリットがあります。

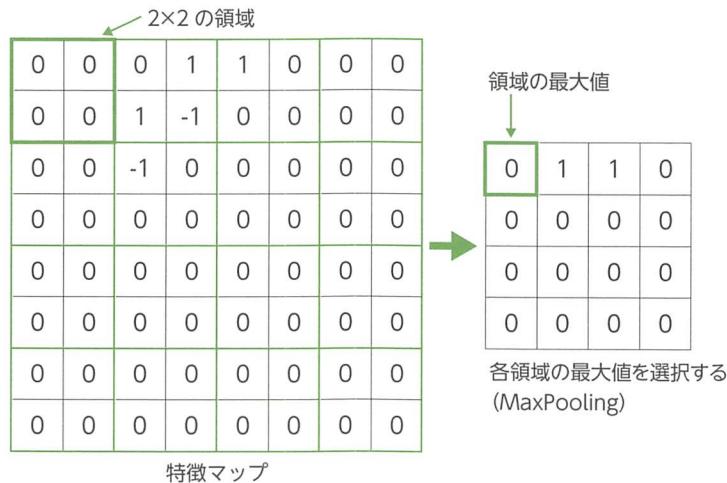


図1-3-8 プーリング層の処理

このような畳み込みニューラルネットワークは、第5章で AWS Deep Learning AMI の作例として紹介しています。

人工知能の基本的な説明はこの程度にして、次章からは Amazon AI の具体的な内容を見ていくことにしましょう。

第 2 章

AWS の機械学習 サービス

AWS では、機械学習に関するさまざまなサービスが提供されています。機械学習を行うにあたっては、データの取り扱いが重要です。データがあってこそ機械学習モデルを作ることができます。AWS ではデータを収集・蓄積し、分析するためのサービスも豊富に提供されています。

本章では、これらのサービスをどのように組み合わせて機械学習を実システムに取り込んでいくかを学びましょう。

2.1

機械学習をどう使うか

2.1.1 機械学習を使う・作る

第1章では人工知能の歴史を振り返りながら、現在の第3次ブームの中心となる技術である機械学習やディープラーニングについて説明しました。では、機械学習やディープラーニングは、どのように使えば良いのでしょうか。

そもそも、機械学習を「使う」ために、機械学習そのものを「作る」必要があるのでしょうか？作らなくても使うことができるのであれば、これほど簡単なことはありません。AWSでは、機械学習を「作る」ためのサービスだけでなく、機械学習を作らずに単に「使う」ためのサービスも提供されています。作らなくても使うことができるなら、なぜ作るためのサービスが必要なのでしょうか。

本章では、機械学習を使う・作ることのメリットなどを確認しながら、AWSで提供されているサービスについて概観していきましょう。

2.1.2 プログラミングでモデルを作る

「機械学習やディープラーニングを使う」というと、Python や R といった、機械学習や統計分析に適したプログラミング言語を使って、自分でモデル^{*1}を作ることが思い浮かぶでしょう^{*2}。書店には、Python や R を使って機械学習を行うための技術書がたくさんありますし（本書もその1つといえるかもしれません）、インターネット上でもそれらの情報を多く見かけます。

本書では第4章および第5章で、Python による機械学習やディープラーニングを使って自分

*1 第1章で説明したように、機械学習やディープラーニングを使って分類や回帰を行うためには「モデル」を作成し、トレーニングデータを用いて学習を行う必要があります。トレーニングデータを用いて学習フェーズを終えたモデルのことを「トレーニング済みモデル」といいます。本書では、特にトレーニング済みであることを強調する必要がある場合を除いて、単に「モデル」と記述します。

*2 プログラミングでモデルを作るといっても、プログラミングだけでモデルが完成するわけではありません。プログラミングによって作られたモデルは学習が済んでいない状態なので、分類や予測はほとんどできません。プログラミングで作成したモデルに対して、トレーニングデータを使って機械学習を行うことによって、比較的精度の高い分類や回帰ができるモデルにしていくのです。

でモデルを作成します。ディープラーニングに適したライブラリはオープンソースで数多く公開されていますが、第5章ではGoogleが公開しているTensorFlowとKerasといったライブラリを活用することにしましょう。機械学習に関する最新の研究成果はこれらのライブラリにすぐに反映されるので、新しい技術に触ることができ、自分で思いどおりのモデルを作れるという魅力があります。もちろん、トレーニングデータとして独自のデータを用いることも可能です。機械学習やディープラーニングを使う方法としては、こうしたプログラミングによるものが代表的といえるでしょう。

2.1.3 Webサービスでモデルを作る

最近は、プログラミングを行わずに、Webブラウザの操作だけでモデルを作成できるサービスも増えてきています。例えば、MicrosoftのAzure Machine Learning(Azure ML)は、Webブラウザ上で部品をドラッグ&ドロップするだけで、トレーニングデータの加工や各種アルゴリズムを用いた機械学習、さらに作成したモデルの評価も行うことができます。

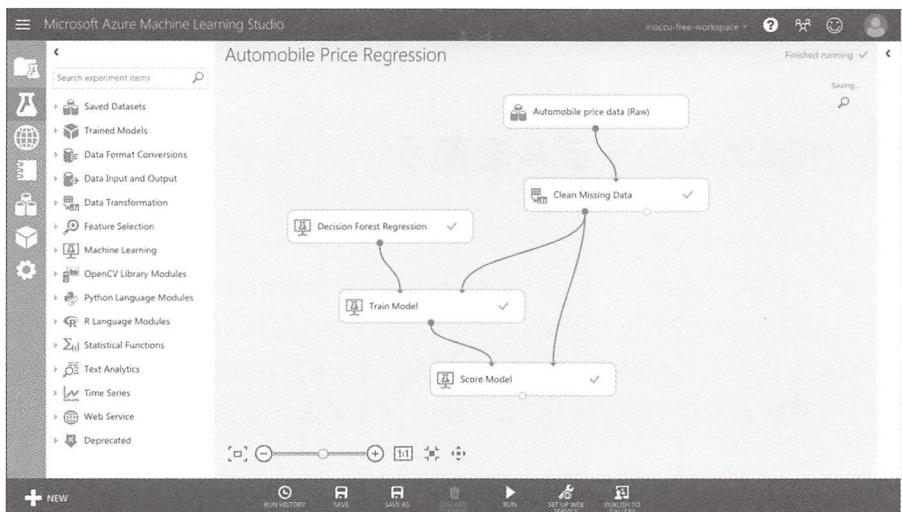


図 2-1-1 Microsoft Azure Machine Learning の画面例

また、SonyのNeural Network Consoleも、WebブラウザやWindows用のデスクトップアプリ上でのドラッグ&ドロップにより、ディープラーニングで用いるニューラルネットワークの構造を作り、学習フェーズを経てモデルを作成することができます。

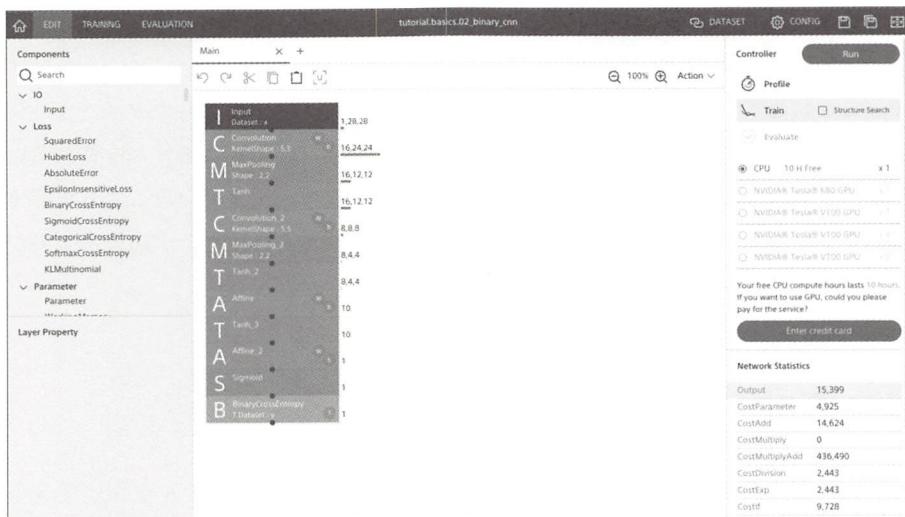


図 2-1-2 Sony Neural Network Console の画面例

他にも、Google は Cloud Auto ML、IBM も Watson Machine Learning(Watson ML) や Watson Studio というサービスを提供しています。もちろん、AWS にも第4章で解説する Amazon SageMaker というサービスがあります。

2.1.4 作成済みのモデルを使う

自分でモデルを作らなくても、機械学習やディープラーニングを用いて実現された機能を使うこともできます。IBM Watson や Microsoft Azure Cognitive Services では、画像認識や音声認識といった、一般的に機械学習などの技術を用いて実現する機能を、自分で開発したアプリから簡単に呼び出せる API として提供しています。画像認識や音声認識で用いるモデルはあらかじめトレーニング済みなので、自分でトレーニングデータを準備する必要もありません^{*3}。

^{*3} IBM Watson では、トレーニング済みのモデルを使用する方法のほか、独自のトレーニングデータを準備して独自のモデルを作る方法も提供されています。したがって IBM Watson は、作成済みのモデルを使う方法と Web サービスでモデルを作る方法のハイブリッド型といえるかもしれません。

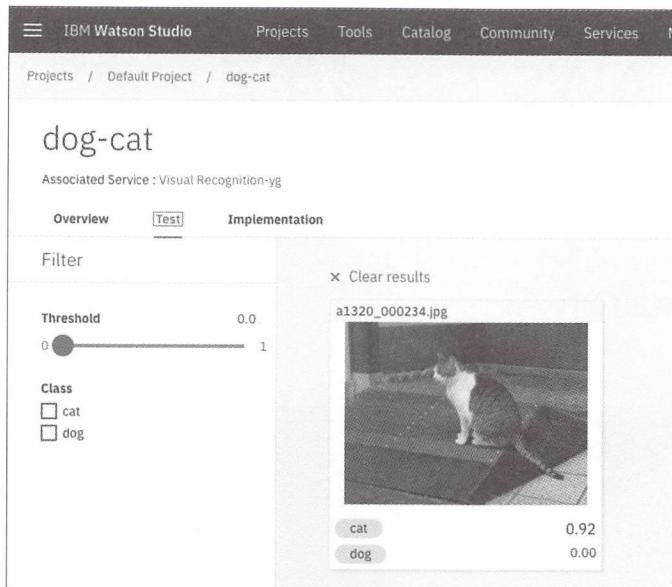


図 2-1-3 IBM Watson Visual Recognition の画面例

このようなサービスは、コグニティブ（認識）サービスまたはコグニティブ API と呼ばれることがあります。IBM は Watson を、コグニティブコンピューティングを実現するためのプラットフォームと説明しています。一方、AWS には Amazon AI というサービスがあり、画像認識や音声認識、チャットボットなど、さまざまなサービスを提供しています。

2.2 AWSで機械学習を使うには

2.2.1 Amazon AI



図 2-2-1 Amazon AI サービスの解説 (<https://aws.amazon.com/jp/machine-learning/what-is-ai/>)

ここでは、AWSで提供されているサービスを見ていくことにしましょう。

Amazon AIは、基本的にはあらかじめ作成済みのモデルを用いて、画像認識や音声認識といった機能を使うためのサービスであり、コグニティブサービス（右ページの「COLUMN」参照）に分類されます。機械学習アルゴリズムを使って自分でモデルを作ったりトレーニングデータを準備しなくとも、機械学習ならではといえるサービスを使えるので、自分のアプリや企業の業務システムなどのさまざまなシステムにAI的な機能を組み込む最も簡単な方法といえるでしょう。

また、2018年から2019年にかけてAmazon AIのサービスが追加され、コグニティブサービスにあたらないものも出てきています。例えばAmazon Forecastは、過去のデータなどをトレーニングデータとして、さまざまな予測を行うモデルを簡単に作成できるサービスです。次項以降で触れるAmazon SageMakerなどでは、自由にモデルを作成できる分だけ、機械学習に関する高度な知識を必要とします。しかし、Amazon AIに分類されるサービスは、ある程度決まった

用途のモデルしか作れない代わりに、そうした知識は要しません。

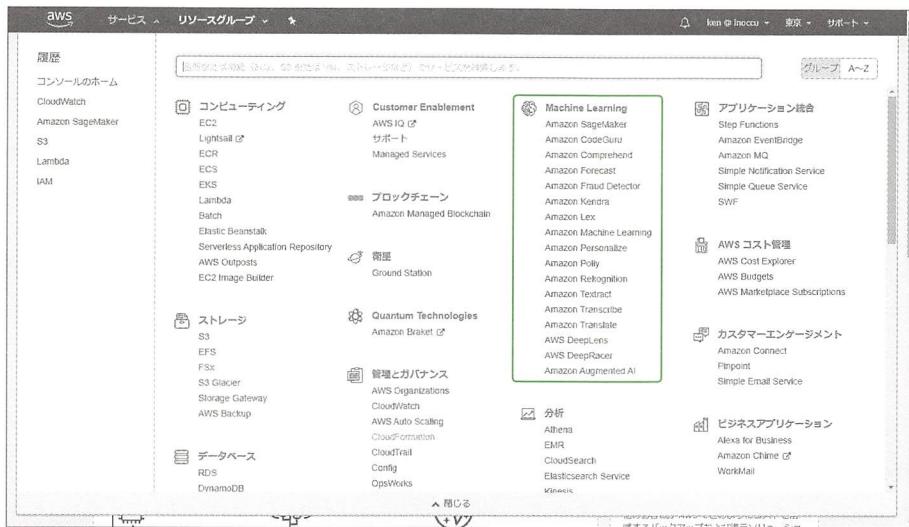


図 2-2-2 AWS コンソールでの機械学習サービスの一覧^{*4}

COLUMN

コグニティブサービス

コグニティブ (Cognitive) は、「認識」や「認知」という意味を持つ単語です。具体的には、人間が、目で見たものが何であるかを感じ取ったり、耳で聞いた言葉が何を意味しているかを理解したりすることをいいます。

また、コグニティブサービスとは、そのような機能を機械学習やディープラーニングなどの技術を活用してコンピュータで実現し、それをAPIサービスとしてクラウド上で提供しているものをいいます。本書で取り上げるAmazon AI以外にも、多くのクラウドサービス事業者がコグニティブサービスを提供しています。

- IBM Watson

IBMが提供しているWatsonは、同社が提唱するコグニティブコンピューティングの中核サービスです。画像認識サービスであるVisual Recognition、自然言語を分類するNatural Language Classifier、チャットボットなどで会話の流れを制御するWatson Assistant、大量の文書を蓄積して話し言葉などで検索できるDiscoveryなどのサービスが提供されています。

- Microsoft Azure Cognitive Services

Microsoftは、同社のクラウドサービスであるAzureで、Cognitive Servicesというサービスを提

^{*4} 機械学習サービスの一覧のうち、Amazon SageMakerとAmazon Machine Learning以外はコグニティブサービスに分類されます。一覧を見ると、コグニティブサービスが豊富に取り揃えられていることがわかります。

供しています。画像認識を行う Computer Vision API や検索エンジンの Bing に関するサービスなどがあります。

- Google Cloud Platform

Googleも同社のクラウドサービスである Google Cloud Platform 上で、コグニティブサービスを提供しています。特にコグニティブサービス単独のブランド名はありませんが、画像認識の Cloud Vision API や、自然言語に関する機能を提供する Cloud Natural Language API などを提供しています。

日本企業も、富士通の Zinrai やリクルートの A3RT などでコグニティブサービスを提供しており、今後もさらに参入企業が増えて、サービス間の競争が激しくなることが予想されます。

さて、画像認識を例にとると、画像に何が写っているかを認識したり、画像に写っている顔の表情を分析したりすることができます。何が写っているかは、あらかじめ学習されているものしか認識できません。それで用が足りるなら、自らトレーニングデータを準備して独自のモデルを作成する場合よりも、認識精度は一般的に良好なため、Amazon AI を使うメリットは大きいといえるでしょう。本書では第3章で、Amazon AI の各サービスについて説明します。

COLUMN Amazon Machine Learning (Amazon ML)

AWSがAIに関する機能を提供し始めた頃から用意されているサービスとして、Amazon Machine Learning (以下、Amazon ML) があります。Amazon MLは、Webブラウザの操作でトレーニングデータをアップロードすることで、独自のモデルを作成できるサービスです。数値や単語程度の文字列から成るデータをインプットして、分類したり数値を予測したりする学習モデルを作ることができます。ただし、本書執筆時点において新たに Amazon ML を使い始めようとすると、このサービスがいずれ廃止される旨の警告メッセージが表示されます。

Amazon MLでは、過去の売上データをもとに売上予測を行ったり、工場の生産装置の稼働データから装置の故障や消耗品の交換時期を予測するなどの用途が想定されていました。これらの用途には、新たに提供された Amazon Forecast や Amazon Personalize を使うことができます。また、Amazon SageMakerを使えば、より柔軟にモデルを作成できます。

2.2.2 Amazon SageMaker

Amazon SageMaker（以下、SageMaker）は、機械学習アルゴリズムを用いたモデル作成からデプロイまで、一貫して対応するサービスです。モデルの作成にあたっては、Pythonでの開発時によく使われる Notebook を使用できます。

SageMaker では Notebook を使用することで、Web ブラウザ上で Python のプログラムを書き、それを逐次実行して動作を確認しながら効率的にモデルを開発できます。また、SageMaker では、独自のライブラリや機械学習アルゴリズム（組み込みアルゴリズム）も提供されています。なお、機械学習の学習フェーズでは高性能なコンピュータ環境が必要となりますが、それを AWS のクラウド上で実現できることは大きなメリットです。さらに、完成したモデルを SageMaker 上の操作だけで API として公開することも可能です。このように SageMaker は機械学習に関するオールマイティな機能を有しているため、AWS 上でのモデル開発において大変便利なサービスです。 SageMaker については第 4 章で詳しく説明します。

2.2.3 Amazon EC2 と AWS Deep Learning AMI

機械学習やディープラーニングを行う際、AWS でお馴染みの IaaS（Infrastructure as a Service）環境である Amazon EC2（Amazon Elastic Compute Cloud。以下、EC2）を使用することもできます。EC2 では GPU 搭載のインスタンスが提供されているので、特にディープラーニングにおいて有用でしょう^{*5}。

なお、EC2 では、Linux などの基本ソフトのみが導入されたコンピューティング環境が提供されます。このため、一般的に機械学習やディープラーニングを行うための環境（Python や TensorFlow などのライブラリ）を構築するのは手間がかかります。しかし、そのような環境があらかじめ構成された AWS Deep Learning AMI（Amazon Machine Images）も提供されているので、機械学習のプログラミング作業にすぐに取りかかることができます。

本書では第 5 章で、AWS Deep Learning AMI を用いた EC2 の環境構築と、簡単な学習モデルの作成について説明します。

*5 ディープラーニングでは、GPU を搭載した高性能なコンピュータを用いることで学習にかかる時間を節約できます。

2.2.4 データレイクとデータ分析系サービス

機械学習を行うためには、一般的に、大量のデータすなわちビッグデータを準備する必要があります。また、個人でモデルを開発する場合を除いて、開発メンバーの間でそのデータを共有する必要があります。そのため、データをクラウド上に保管し、隨時使えるようにしておくことが求められます。そのための環境を一般的にデータレイクといいます。AWS ではデータレイクを構築するためのサービスとして、オブジェクトストレージである Amazon S3 (Amazon Simple Storage Service。以下、S3) や、NoSQL データベースである Amazon DynamoDB (以下、DynamoDB) を提供しています。

また、機械学習モデルを開発する際には、学習に用いるデータの特徴をあらかじめ分析しておくことも必要です。分析には、データウェアハウスの Amazon Redshift や、ビジネスインテリジェンス (BI) サービスの Amazon QuickSightなどを用います。なお、アドホック（その場限りの臨時的）な分析を行うために、S3 に蓄積したデータについて SQL で集計・検索などを行う Amazon Athena を使う場合もあるでしょう。

2.3

機械学習をシステムで使うには

2.3.1 機械学習を使ったシステムとは

ここでは、「機械学習を使ったシステム」とは、一体どのようなシステムなのか考えてみたいと思います。

機械学習を使ったシステムは、大きく分けて、モデルそのもの、そのモデルを使って業務要件などを処理するロジック、画面などのユーザーインターフェースで構成されます。システムには何らかの実現したい要件があり、一般的なシステムではユーザーインターフェースで入力を受け付け、ロジックで処理を行い、その結果をユーザーインターフェースで表示します。モデルは、主にロジックの部分から呼び出されて一部の処理を肩代わりする、というわけです。モデルだけあれば要件を満たせるということはほとんどありません。

また、モデルを作成するためにはデータを使った学習が必要ですが、その学習は一度で終わるわけではありません。基本的には、システムの本番運用を通してデータを収集し、そのデータを使ってモデルを洗練させる作業も必要です。したがって、そのための仕組みも、機械学習を使ったシステムは備えておく必要があります。

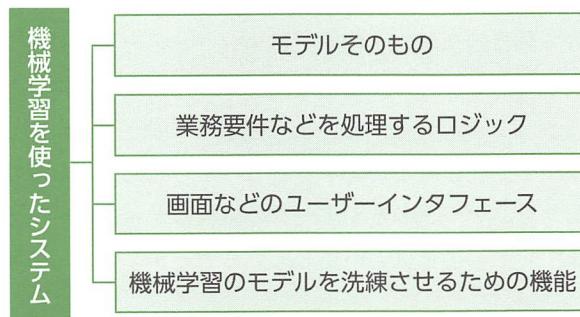


図 2-3-1 機械学習を使ったシステム

2.3.2 機械学習のワークフロー

機械学習のモデルは、図2-3-2のような手順で作成および学習・評価を行い、システムに実装します。

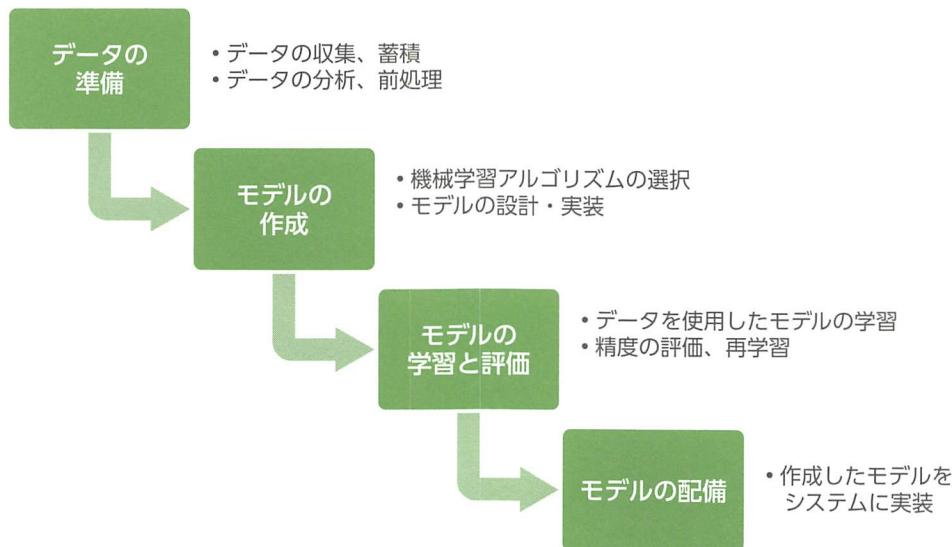


図2-3-2 機械学習のワークフロー

機械学習を使ったモデルを作成するためには、基本的にはデータが必要です。Amazon AIのようなAIサービスでは作成済みのモデルが提供されているので、自らデータを準備する必要はありません。一方、自分で独自のモデルを作成する場合は、データが必要です。そのデータはどこにあるのでしょうか。既存のシステムで蓄積しているデータや、公開されたオープンデータを使用できるなら、それを使えば良いでしょう。しかし、そのようなデータが存在しない場合は、まずデータを収集し、蓄積する仕組みを構築するところから始めなければなりません。なお、既にデータが存在する場合でも、データの量やデータの項目数が足りないことがモデルの作成時に判明するケースがあります。その場合は、不足しているデータを収集・蓄積する方法を考えることになります。

データの収集・蓄積ができたら、次にそのデータを分析し、モデルの作成工程に入りますが、ただデータがあるだけではモデルを作成できません。どのような機械学習アルゴリズムを使用し、どのようなデータを投入してトレーニングを行い、モデルを作成するかを考える必要があります。蓄積したデータをそのまま投入するのではなく、前処理として何らかの加工を施す必要があります。そうした前処理によってモデルの精度は向上するのです。

COLUMN データの前処理

機械学習に使用するデータは前処理を行う必要があります。データの前処理として行われるものには、以下のような事柄があります。

● 欠損値の補完

特定のデータにおいて、特徴量として使用する項目の値がセットされていないような状態を欠損値といいます。欠損値があると機械学習が行えないため、できるだけ学習や予測に影響を与えないような値で補完します。数値項目では、平均値や中央値といった値が用いられます。

● 外れ値のあるデータの除去

データ全体の中で大きく外れた値（外れ値）を持つデータは、機械学習モデルの予測結果に大きな影響を及ぼすことがあります。そのため、外れ値のあるデータを除去します。

● 多重共線性の回避

特徴量として用いる項目間で、強い相関を示す項目ペアがある場合、機械学習モデルの予測結果に大きな影響を及ぼすことがあります。そのため、どちらかの項目を除去します。

● データの変換

機械学習では基本的に、数値化されたデータしか使用できません。そこで、性別の項目なら男性を0、女性を1といった数値に変換します。また、同一の事柄を指しているのに別の用語が用いられている場合は名寄せして、同じ値として示されるように変換します。

● 自然言語データの数値データ化

自然言語のデータは、そのままでは機械学習で使用できないため、BoW^{*6} や Doc2Vec^{*7} などの手法を用いて数値データに変換します。

データの準備ができたら、使用する機械学習のアルゴリズムを決定し、モデルの設計と実装を始めます。さらにデータを使ってモデルのトレーニングをすれば、トレーニング済みのモデルができるになります。一度のトレーニングで必要十分な精度を持つモデルができるケースは少なく、たいていの場合はデータの準備に戻ったり、モデルの設計を見直したりして、再度トレーニングを行います。このような試行錯誤を繰り返して精度を高めていき、システムに組み込める程度の

*6 BoW (Bag of Words) は、文章の中で、ある単語がどの程度現れるかを数値化し、その数値によって文章の特徴を示す手法です。ただし、文章内での単語の出現順序は考慮されません。

*7 Doc2Vec は、文章をベクトル（ひとまとまりの数値を並べたもの）化する手法で、BoW と同様に機械学習で自然言語を扱う際に使用されます。BoW とは異なり、Doc2Vec は単語の出現順序が考慮されるため、文章の意味が表現されやすいという特徴があります。

精度を持つモデルを完成させます。

完成したモデルはシステムに組み込まれます。基本的にはモデルを使用するための API を作成し、それをシステムから呼び出すことになるでしょう。モデル作成のワークフローはこれで完了ですが、システムの本番運用を開始した後、さらにデータが蓄積されるので、それを使ってモデルの精度をより一層高めるためのトレーニングやモデルの設計見直しなどを行います。

2.4

AWSで機械学習のワークフローを作る

2.4.1 データの収集と蓄積

機械学習のワークフローはなかなか煩雑です。また、モデルの精度を向上させるための取り組みは、そのモデルを使い続ける限り継続して行う必要があります。

しかし、AWS のサービスを利用すれば、ワークフローをスムーズに進めることができます。例えばデータの収集に関しては、Web サイトのアクセスログや IT システムの動作ログのようなパーセンタルデータが収集の対象であれば、Amazon Kinesis Data Streams や AWS Data Pipeline などを利用できます。また、工場の設備稼働データや人間の身体データのようなリアルデータの場合は、AWS IoT Core などの IoT サービスを利用することになるでしょう。

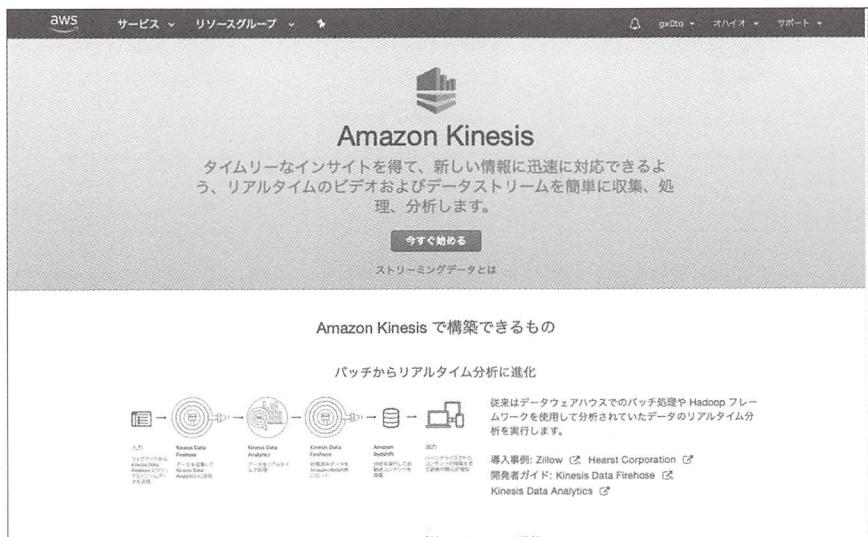


図 2-4-1 Amazon Kinesis

収集したデータは、S3 や DynamoDB といったデータを蓄積するサービスに保存します。この後で使用する Amazon ML や SageMaker などの機械学習サービスでは、S3 に蓄積したデータを主に使用するようになっているので、まずは S3 に保存すると良いでしょう。

2.4.2 データの分析と前処理

データの分析に関しては、前述したように Amazon Athena や Amazon Redshift といった分析系のサービスがあります。これらのサービスを利用するにあたって、まずは機械学習で解決したい課題について仮説を立てて、その仮説がデータで実証できるかを確認、検証する仮説検証のプロセスが必要です。設定した課題について、どの項目が影響を与えていているか^{*8}、どの程度の精度で予測や分類ができそうかといったデータの構造が見えれば、AWS の機械学習サービスや機械学習のアルゴリズムのうち、どれを使えば良いのかがわかるでしょう。

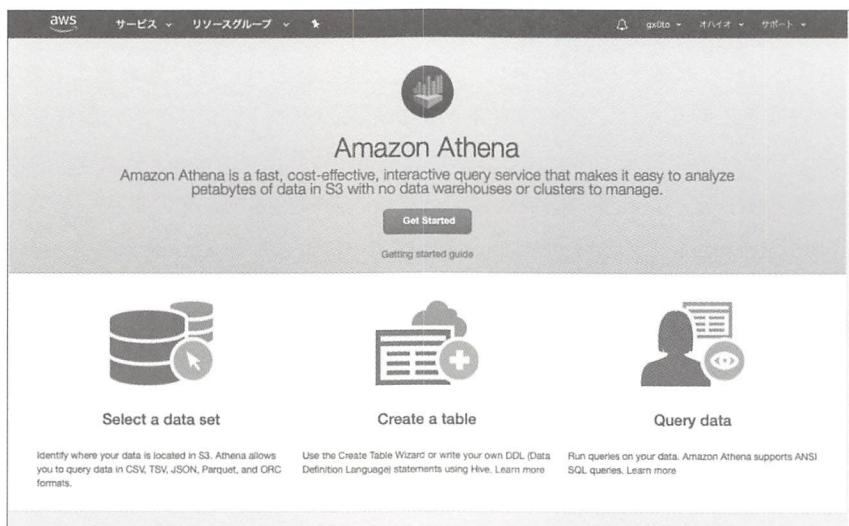


図 2-4-2 Amazon Athena

機械学習を行う際の前処理として、P41 の「COLUMN」で紹介したように、データの加工が必要になる場合があります。例えば、準備したデータに欠損値^{*9}があれば、そこに代わりの値などをセットします。なお、異常値^{*10}があれば、そのデータは無視した方が良いかもしれません。データの加工については、あらかじめ Excel などで CSV データを編集することもありますし、SageMaker で提供されている Notebook を使って Python のプログラムで処理することもあります。

*8 どのような項目が結果に影響を与えるかがわかれれば、モデルを作成する際の特徴量として活用することができます。

*9 すべてのデータ（表形式では行）において、すべての項目（列）の値がセットされていれば良いのですが、IoT データなどではそうでない場合もあります。このように値がセットされていない（欠損した）データのことを、欠損値のあるデータといいます。

*10 準備したデータ全体を見通すと、ある項目（列）の値が他の行の値に比べて大きく外れていることがあります。例えば、IoT センサーで室温のデータを蓄積している場合、80°Cというデータがあったならセンサーの故障が疑われます。外れ値の中でも、このように原因がわかつているデータのことを異常値と呼んでいます。

2.4.3 どの機械学習サービスを使うか

データを準備できたら、次は使用する機械学習のサービスを決めます。AWSではさまざまな方法で機械学習を使うことができるので、その選択が重要になります。

コグニティブサービスを使う

- ・データを準備する必要がない（または最低限で良い）
- ・使用できるのは、あらかじめ提供されているモデルのみ

Amazon Rekognitionカスタムラベルや、 Amazon Forecastのようなサービスを使う

- ・アルゴリズムを検討しなくてもモデルを作成できる
- ・学習の詳細までは制御できない

SageMakerやEC2上で開発する

- ・どのようなモデルでも作成できる

図 2-4-3 AWSでの機械学習サービスの選択

画像認識や音声認識といった、コグニティブサービスとして提供されていることを実装したい場合は、Amazon Rekognition や Amazon Transcribe 等のサービスの API をシステムに組み込むことを最初に検討することになるでしょう。このような、AWSで学習済みモデルが提供されているサービスのモデルの精度は比較的高いので、自らデータを準備して独自のモデルを作成する必要は基本的にはありません。独自のモデルが必要な場合は、Amazon Rekognition のカスタムラベル機能を使うほか、SageMaker（第4章）や AWS Deep Learning AMI（第5章）などを活用し、TensorFlowなどのフレームワークで画像認識モデルを自作することを検討します。

なお、売上予測や機械の故障予測などを行いたい場合は、Amazon Forecast を最初に検討すべきでしょう。Amazon Forecast では、決められたフォーマットのデータさえ準備すれば、画面の指示に従って操作するだけでモデルを作成できます。

Amazon Forecast で作ったモデルの精度に満足できない場合は、SageMakerなどを使って、モデルの作成に使用する機械学習アルゴリズムの選択や、きめ細かな設定を行うことになるでしょう。また、前述のように、SageMaker では Notebook を使ってデータの前処理を行えるので、使用する機械学習アルゴリズムだけでなく、投入するデータの観点でも見直しを行うことができます。

```
In [1]: from sagemaker import get_execution_role
role = get_execution_role()
bucket='sagemaker-inocu'

In [2]: %time
import pickle, gzip, numpy, urllib.request, json
# Load the dataset
urllib.request.urlretrieve("http://deeplearning.net/data/mnist/mnist.pkl.gz", "mnist.pkl.gz")
with gzip.open('mnist.pkl.gz', 'rb') as f:
    train_set, valid_set, test_set = pickle.load(f, encoding='latin1')
CPU times: user 936 ms, sys: 268 ms, total: 1.2 s
Wall time: 7.01 s

In [3]: %matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (2,10)

def show_digit(img, caption=''):
    if subplot is None:
        (subplot) = plt.subplots(1,1)
    img = img.reshape((28,28))
    subplot.axis('off')
    subplot.imshow(img, cmap='gray')
    plt.title(caption)

show_digit(train_set[0][30], 'This is a 3')

```

図 2-4-4 SageMaker で Notebook を使用する

2.4.4 システムへの組み込み

機械学習モデルをシステムに組み込む際は、基本的に API を使用します。Rekognition などのコグニティブサービスでは、もともと API としてサービスが提供されているので、それをシステムの中から呼び出す形になります。Amazon Forecast や SageMaker で作成したモデルは、API 化をそれぞれのサービスの中で行うことができます。また、システムのオンライン処理の中でリアルタイムに呼び出すだけでなく、バッチ処理を行わせることも可能になっています。

このように、AWS ではデータを収集・蓄積し、モデルを作成し（または提供されたモデルを使用し）、システムに組み込むためのサービスが豊富に提供されています。これらのサービスを適宜組み合わせて、最適なワークフローを作っていきます。

2.5 AWSのアカウントを作る

2.5.1 アカウントの作成手順

読者の皆さんには、既に AWS のアカウント登録を済ませている方が多いと思いますが、念のため簡単に触れておきたいと思います。

AWS のユーザー登録を行うには、まず、<https://aws.amazon.com/jp/> にアクセスして、アカウントを作成するためのボタンをクリックします。

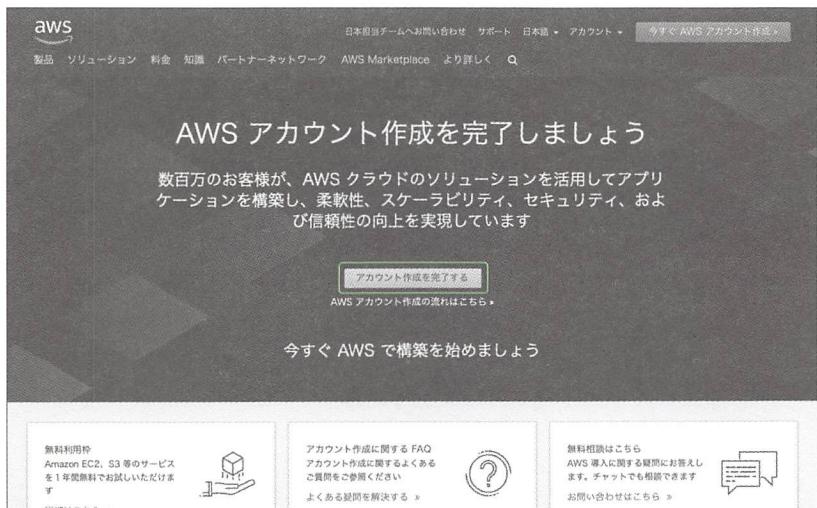


図 2-5-1 AWS のトップページ (<https://aws.amazon.com/jp/>)

AWS アカウントの作成画面が開くので、E メールアドレス、パスワード、AWS アカウント名を入力し、「続行」ボタンをクリックします。

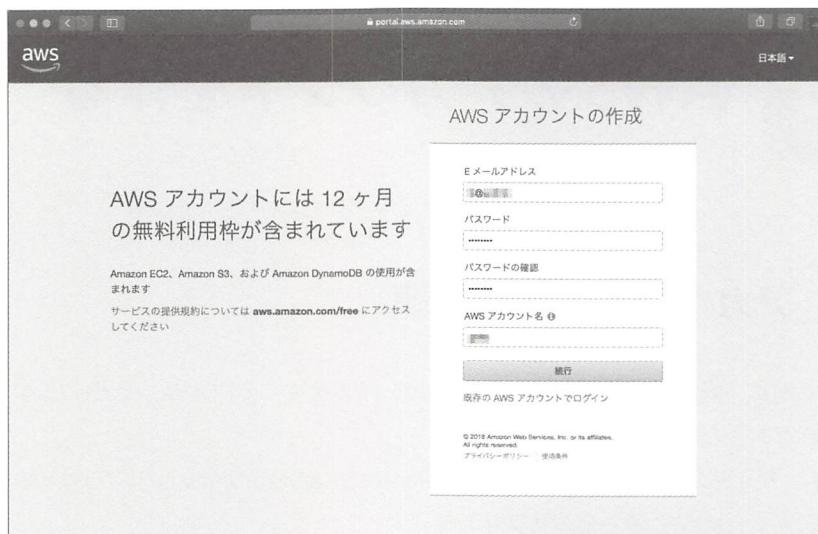


図 2-5-2 AWS アカウントの作成

次に、連絡先情報を設定します。個人で登録する場合は、アカウントの種類として「パーソナル」を選択します。

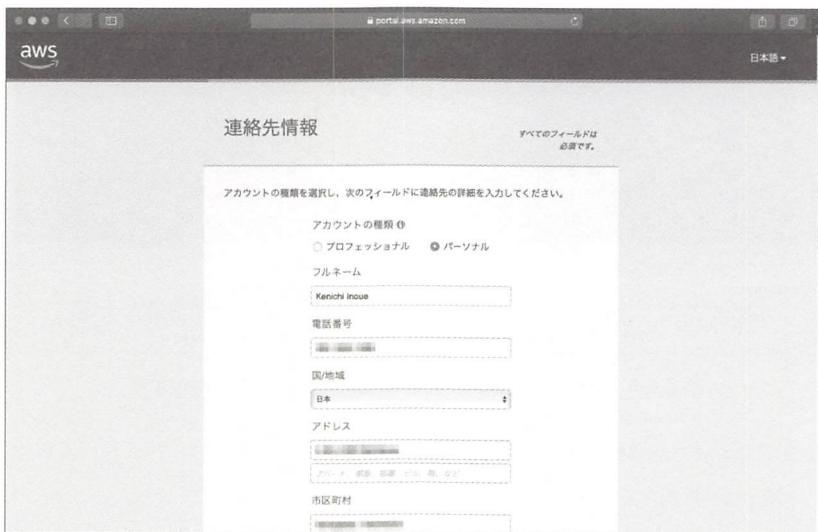


図 2-5-3 連絡先情報

AWS では、アカウントを作成する際に支払情報の入力が必要です。アカウントの作成後、12か月間はサービスごとに一定の無料枠がありますが、それを超えて使用する場合や、無料枠が提供されていないサービスを使用する場合は課金が発生します。

The screenshot shows a form titled "支払情報" (Payment Information). It includes fields for credit/debit card number, expiration date (09 2023), cardholder's name (KENICHI INOUE), and shipping address. There are two radio button options: "連絡先住所を使用する" (Use contact address) and "新しい住所を使用する" (Use new address), with the former selected. A "セキュアな送信" (Secure transmission) button is at the bottom.

図 2-5-4 支払情報

AWS のアカウント作成を完了するためには、電話による確認が必要です。画像によるセキュリティチェックを入力して「すぐに連絡を受ける」ボタンをクリックすると、画面が遷移して4桁の数字が表示されます。そして、AWS から自動音声の電話がかかってくるので、その数字を電話機のプッシュボタンで入力すると確認が完了します。

The screenshot shows a form titled "電話による確認" (Phone Verification). It instructs users to enter the 4-digit code sent via phone. Fields include "国/地域コード" (Country/Region Code) set to "日本 (+81)", "電話番号" (Phone Number) with an "内線" (Extension) field, and a "セキュリティチェック" (Security Check) section with a CAPTCHA input field containing "mm7fdb" and two verification buttons. A "すぐに連絡を受ける" (Receive a call now) button is at the bottom.

図 2-5-5 電話による確認

電話による確認が完了すると、図 2-5-6 のような画面に自動的に遷移します。

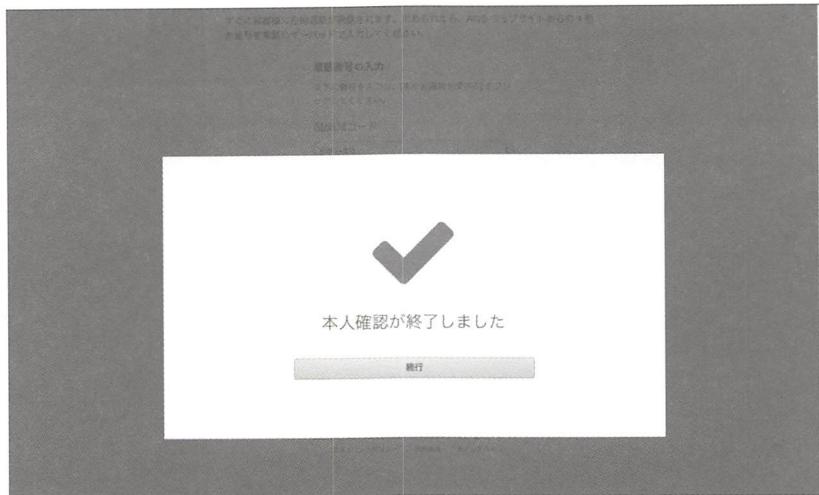


図 2-5-6 本人確認の終了

さらにサポートプランの選択を行います。本書で説明している内容を検証する程度なら、無料のベーシックプランを選択すると良いでしょう。

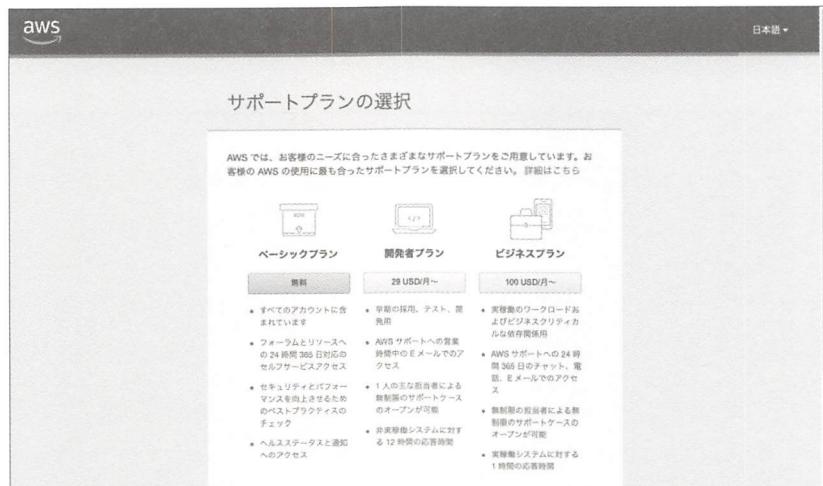


図 2-5-7 サポートプランの選択

これで AWS のアカウントが作成できました。

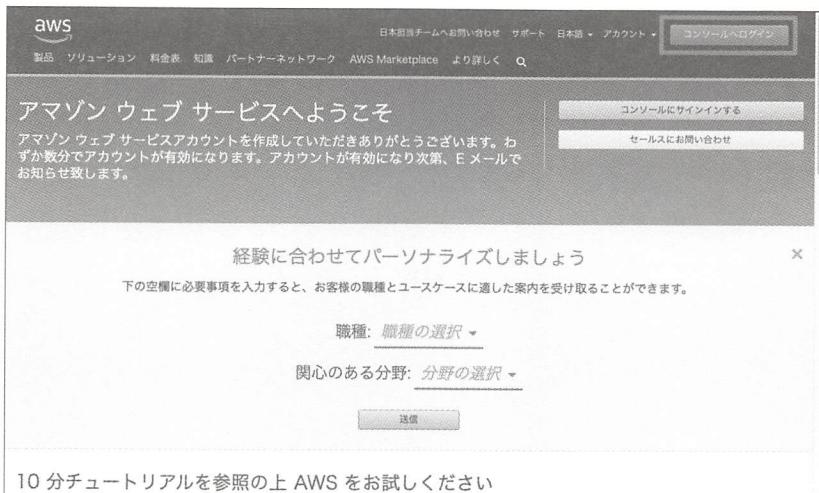


図 2-5-8 アカウント作成の完了

作成したアカウントでログインすると、AWS マネジメントコンソールが表示されます。AWS のサービスは、画面上部のメニューにある「サービス」をクリックすることで使用を開始することができます。



図 2-5-9 AWS マネジメントコンソール

第3章

AI サービス

いよいよ AWS の機械学習サービスを使っていきます。まずは画像認識（Rekognition）や音声認識（Transcribe）といった、「いかにも AI」な機能を気軽に使えるサービスを実際に操作していきます。また、トレーニングデータを自分で準備して予測モデルを作成する Forecast や Personalize といったサービスも AI サービスに分類されているので、それらも本章で説明します。

3.1 AIサービスとは

3.1.1 AIサービスとコグニティブサービス

AWS では機械学習（ML）サービスを、AI サービス、ML サービス、フレームワーク、インフラストラクチャの 4 つに大きく分類しています。

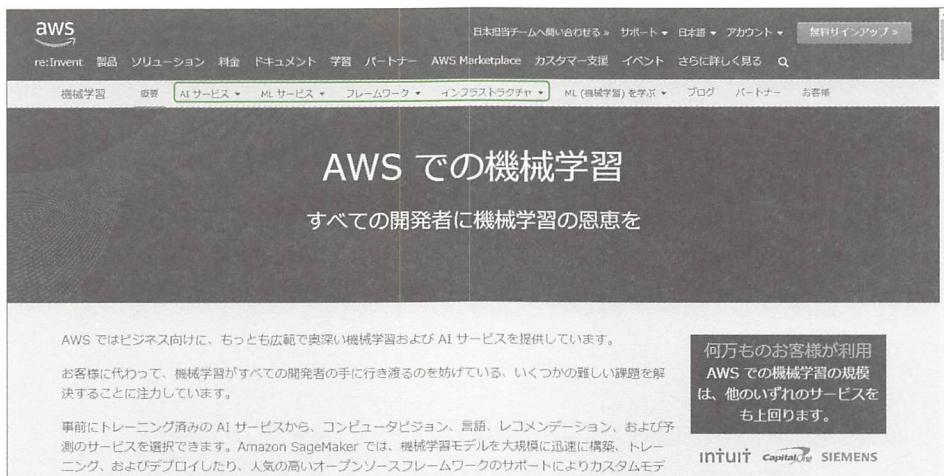


図 3-1-1 AWS での機械学習 (<https://aws.amazon.com/jp/machine-learning/>)

これらのうち AI サービスは、機械学習の仕組みをあまり意識することなく、そのメリットを活用できるサービスです。例えば、画像認識を行う Amazon Rekognition や、チャットボットを開発できる Amazon Lex といった、主に機械学習済みのモデルを API から呼び出すタイプのサービスがあります。Amazon Rekognition では、猫が写った画像に「猫」というラベルを付けるような、一般的な物体の画像認識などができる機械学習モデルが、学習済みの状態で提供されています。私たちは、それを API で呼び出して使用することができます。その際、事前に猫の画像を大量に準備して機械学習を行う必要はありません。

Amazon Lex では、チャットボットを開発することができます。ユーザーからどのような問い合わせが行われるか、その問い合わせに対してどのように返答するかといった設定が必要ですが、機械学習の学習アルゴリズムや、会話の制御を行うプログラムそのものを開発する必要はありません。

ところで、画像認識を行う Amazon Rekognition やチャットボットを作成する Amazon Lex といった、人間の目や耳などの代わりをしたり、会話のような人間ならではの能力の代わりを行うサービスを、コグニティブサービスと呼ぶことがあります。コグニティブとは、日本語では「認知」や「認識」を意味します。例えば、IBM の Watson は自らをコグニティブサービスと呼んでいますし、Microsoft の Azure で同様の機能を提供するサービスには Cognitive Services という名前が付けられています。AWS ではコグニティブという表現はあまり使われませんが、それに類するサービスは、AI サービスとして位置付けられています。

なお、2018 年から 2019 年にかけて正式に提供されるようになった、Amazon Forecast や Amazon Personalize といったサービスでは、トレーニングデータを準備して機械学習を行う必要があります。ただ、機械学習のモデルそのものは提供されたものを使用し、トレーニングデータを用いてモデルのトレーニングのみを行うので、機械学習の中身を意識する必要はほとんどありません。このため、Amazon Forecast や Amazon Personalize は AI サービスの 1 つとして位置付けられているのだと思われます。

COLUMN 機械学習を自ら行う必要性

コグニティブなサービスが AWS のようなクラウドベンダー^{*1}で提供されているなら、私たちは機械学習について学ぶ必要はないと思うかもしれません。しかし、それは違います。クラウドベンダーが準備したトレーニングデータの量は膨大ですが、あくまで一般的に知られているものに限られます。例えば、ある企業の特殊な製品や、機械の内部構造、農家での野菜や果物の選別といった独特なノウハウにもとづくデータはユーザーが自ら準備するしかなく、アルゴリズムの構築やトレーニングも自ら行うしかないのでです。

AWS では、機械学習そのものを自ら行うための環境もサービスとして提供されています。機械学習は行うものの、その中身についてはあまり考慮する必要のないサービスとして、本章で説明する Amazon Forecast と Amazon Personalize があります。また、機械学習の中身も含めて自由に開発できるサービスとして、第 4 章で説明する Amazon SageMaker と、第 5 章で説明する AWS Deep Learning AMI^{*2}があります。

*1 クラウドでの AI（コグニティブ）サービスは、AWS 以外にも Google Cloud や Microsoft Azure、IBM Cloud などで提供されています。

*2 AWS の EC2 で使用できるマシンイメージを AMI（Amazon Machine Images）といいます。AMI については第 5 章で詳しく説明します。

3.1.2 AI サービスの概要

AI サービスに分類されているサービスには、本書執筆時点では以下のものがあります^{*3}。

- Amazon Rekognition (画像解析・動画分析)
- Amazon Comprehend (テキスト内のインサイトや関係性を検出)
- Amazon Comprehend Medical (Amazon Comprehendのサービスを医療向けに特化したもの)
- Amazon Textract (画像化された文書からテキストとデータを抽出)
- Amazon Translate (テキスト翻訳)
- Amazon Transcribe (音声認識)
- Amazon Polly (テキストを音声に変換)
- Amazon Lex (チャットボット)
- Amazon Forecast (トレーニングデータから時系列予測モデルを作成)
- Amazon Personalize (トレーニングデータからパーソナライズと推奨のモデルを作成)

本章では、これらの AI サービスについて実際に動作を試していきたいと思いますが、残念ながら一部のサービスは日本語に対応していません。Amazon Lex や Amazon Comprehend のような自然言語系のサービスを日本で使用するには、日本語対応は必須といえますが、本書執筆時点では未対応です。こうした日本語対応が行われていないサービスについては、本章では簡単に触れる程度に留めておくことをご了承ください。

また、Amazon Comprehend Medical は、医療向けの文書に特化したモデルが採用されているという違いはありますが、機能としては概ね Amazon Comprehend と同じなので、本書では説明を省略しています。

*3 <https://aws.amazon.com/jp/machine-learning/>

COLUMN**さらに追加される AI サービス**

2019年12月に米国・ラスベガスで開催されたAWSのイベント「re:Invent 2019」において、AWSの各種サービスに関するさまざまな発表が行われました。AIサービスについては、本項で紹介した10個のサービス以外に、下記のサービスの追加が発表されました。

- Amazon CodeGuru（機械学習モデルを活用してプログラムのソースコードレビューを行う）
- Amazon Fraud Detector（機械学習モデルを活用してオンライン支払詐欺や偽のアカウント作成を発見する）
- Amazon Kendra（自然言語による質問でFAQなどのナレッジを検索する）

本書執筆時点では、いずれのサービスもPreviewリリースとして提供されており、Fraud Detectorについては、事前の申し込みを受け付けてから順番にサービスに招待される仕組みになっています（他の2つのサービスは、AWSアカウントさえあれば通常どおり使い始めることができます）。

CodeGuruとFraud Detectorは、これまでAmazonが社内で活用していたサービスを外部に提供するものであり、あらかじめトレーニング済みのモデルを簡単に使うことができます。特にFraud Detectorは、膨大なユーザー数とオンライン取引数を持つAmazonのデータが活用されていると思われるため、高い精度が期待できるサービスといえます。

一方、Kendraは、ユーザー企業などが保持するHTMLやWord、PowerPoint、PDFなどのドキュメントを蓄積し、そのドキュメントを自然言語で検索するサービスであるため、企業のコンタクトセンターなどでの活用が考えられます。ただ、現時点では英語版のみしか提供されておらず、日本企業で使用できるケースは限られたものになるでしょう。日本語の対応が待たれるところです。

3.1.3 AI サービスの課金について

AIサービスの課金は、基本的にAPIを呼び出すたびに行われます。例えば、Amazon Rekognitionで学習済みモデルを使用した画像認識を行う場合、本書執筆時点で画像1枚あたり0.001USDがかかります。また、1か月単位の画像認識数が100万枚超から1,000万枚までは、画像1枚あたり0.008USDとなっています。

Amazon Forecastのような独自のデータを用いたモデルのトレーニングが行われる場合は、トレーニングに関する課金に加えて、完成したモデルを使用する際にも課金されます。本書執筆時点で、トレーニングに関する課金は、トレーニングにかかった時間について1時間あたり0.24USD、トレーニングに使用したデータの保存について1GBあたり0.088USDです。また、モデルを使用した予測を行う際に、1,000件の予測ごとに0.6USDの課金が行われます。

本書のサンプルコードを試す程度であれば、数円～数百円程度の課金が予想されますが、サービスによって課金体形は異なりますので、使用する前に確認するようにしてください。

3.2

SDKの使用準備

3.2.1 AI サービスと AWS SDK

AWS の AI サービスは、基本的には API で機能を呼び出します。例えば、あるプログラムの中で画像認識を行うためには、認識させたい画像データといくつかのパラメータを AWS の Rekognition API に引き渡し、認識結果を得ます。パラメータや認識結果のデータは原則として JSON 形式になっています。

API は、何らかのプログラミング言語で使うためのインターフェースなので、動作を試すには、そのための環境を整えなければなりません。AWS では、プログラムの中で簡単に API を呼び出すための SDK が提供されており、Java、Python、PHP といったプログラミング言語向けのものを選んで使用することができます。本書では、プログラミング言語として Python を使用し、AWS SDK for Python を使います。

さて、本書では Windows と macOS での環境準備について説明します。

3.2.2 Jupyter Notebook の導入 (Windows)

Windows では、標準で Python は導入されていません。本書では Python 3 系と Jupyter Notebook を用いて解説を行うため、Anaconda ディストリビューションをインストールします。

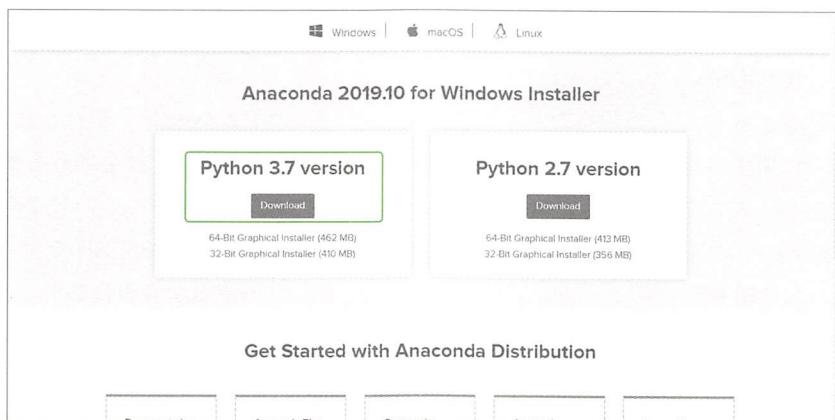


図 3-2-1 Anaconda のダウンロードページ (<https://www.anaconda.com/download/>)

Anaconda のダウンロードページ (<https://www.anaconda.com/download/>) を開き、Windows 用の Python 3.7 バージョンの Anaconda ディストリビューションをダウンロードします。

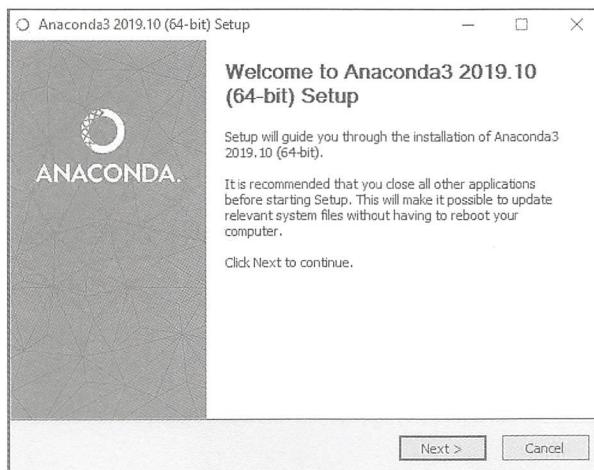


図 3-2-2 Anaconda のインストールウィザード

ダウンロードしたファイルを実行し、ウィザードの指示に従って導入します。

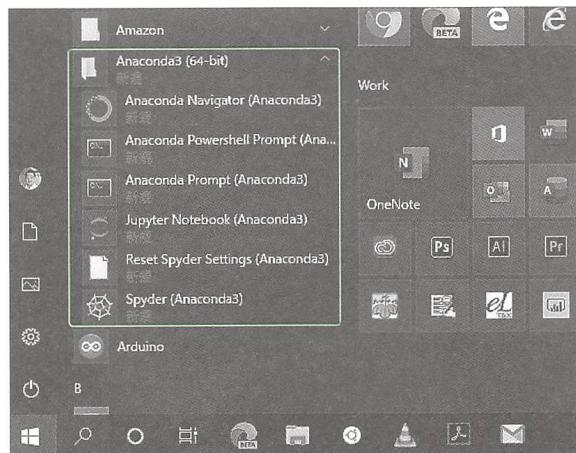


図 3-2-3 Windows のスタートメニュー

インストールが完了すると、Windows のスタートメニューに Anaconda3 (64-bit) というフォルダが表示されます。その中にある Jupyter Notebook を起動すると、自動的に Web ブラウザが開き、Jupyter Notebook が使用可能な状態になります。

3.2.3 Jupyter Notebook の導入 (macOS)

Mac (macOS) では、標準で Python 2.7 系が導入されています。AWS SDK for Python は 2.65 以上／2.7／3.3 以上の Python に対応しており、標準の Python で使用することができます。しかし、本書では Python 3 系で説明を行い、Jupyter Notebook を用いて Web ブラウザ上で のプログラミングと動作確認を進めていきます。そのため、Python 3 系と Jupyter Notebook の環境を簡単に構築できる Anaconda ディストリビューションを導入します。

まず、Anaconda のダウンロードページ (<https://www.anaconda.com/download/>) を開き、macOS 用の Python 3.7 バージョンの Anaconda ディストリビューションをダウンロードします。

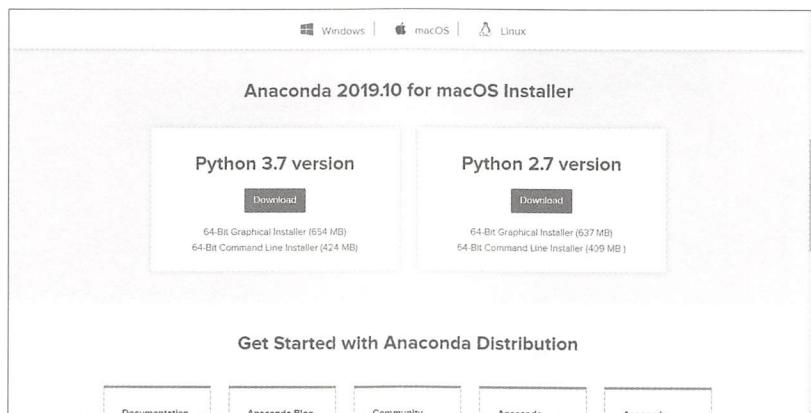


図 3-2-4 Anaconda のダウンロードページ (<https://www.anaconda.com/download/>)

次に、ダウンロードしたファイルを実行し、ウィザードの指示に従います。

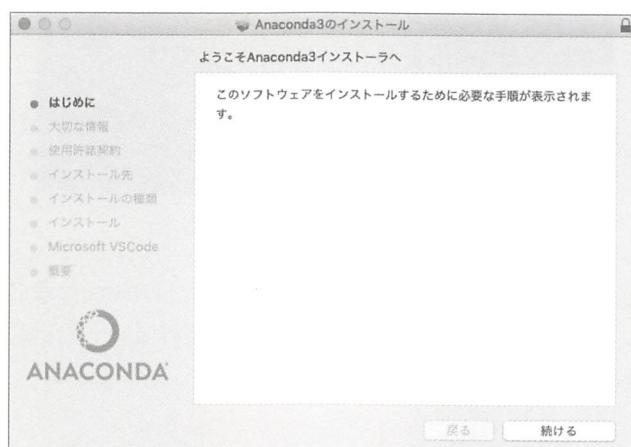


図 3-2-5 Anaconda3 のインストーラ

インストールが完了すると、macOS の Launchpad に Anaconda Navigator のアイコンが追加されます。それを実行して表示されたメニューから、Jupyter Notebook を起動することができます。

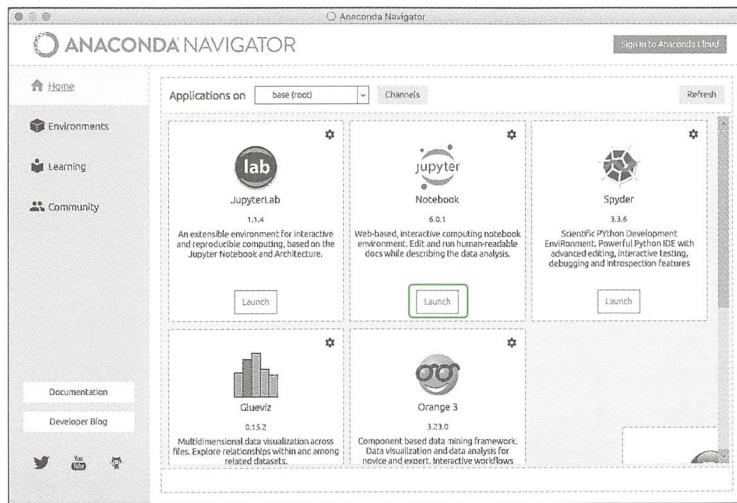


図 3-2-6 Anaconda Navigator から Jupyter Notebook を起動

3.2.4 IAM でユーザーの追加と権限の付与を行う

SDK から AWS のサービスを使用するために、あらかじめアクセスキー ID とシークレットアクセスキーを発行しておく必要があります。Web ブラウザで AWS コンソール (<https://console.aws.amazon.com/>) にログインし、認証に関するデータを管理するサービスである IAM を開きます。

IAM のダッシュボード画面が開いたら、画面左にあるメニューから「ユーザー」をクリックします。



図 3-2-7 IAM のダッシュボード

次に、「ユーザーを追加」ボタンをクリックします。

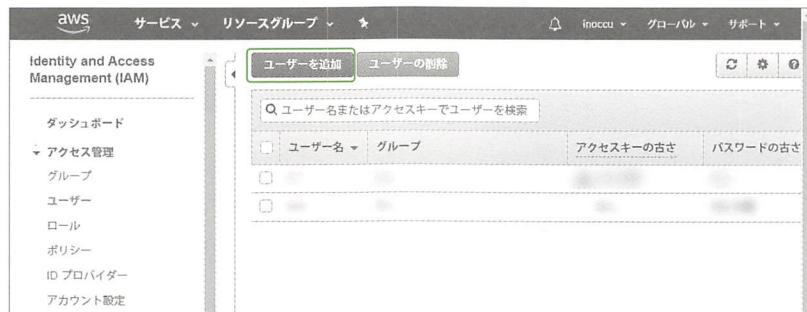


図 3-2-8 ユーザーの追加

ユーザー名を任意で設定します。ここでは、apiuser というユーザー名にしました。アクセスの種類で「プログラムによるアクセス」を選択し、画面右下にある「次のステップ：アクセス権限」ボタンをクリックします。

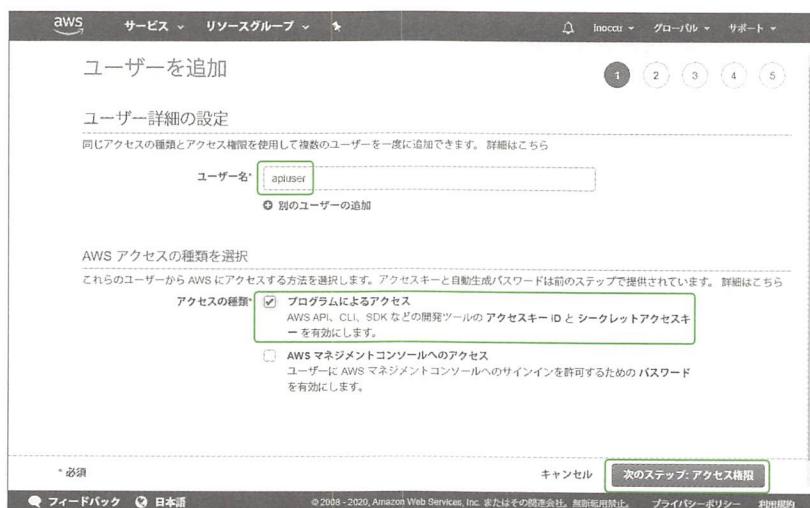


図 3-2-9 ユーザー名とアクセスの種類を設定

次に、作成するユーザーに付与するアクセス権限を指定します。権限は任意に作成することができますが、ここではサービスごとにいくつかの種類の権限をまとめて定義している「既存のポリシー」を使うことにしましょう。「既存のポリシーを直接アタッチ」を選択して、ユーザーに権限をまとめて付与します^{*4}。本章では、まず画像認識の Amazon Rekognition を使用するので、

^{*4} 本番環境では、権限を付与するサービスだけでなく、リソースも限定したポリシーを作成してユーザーにアタッチすべきでしょう。

Amazon Rekognition に関するすべての権限を持つ「AmazonRekognitionFullAccess」にチェック(✓)を入れて、「次のステップ：タグ」ボタンをクリックします。

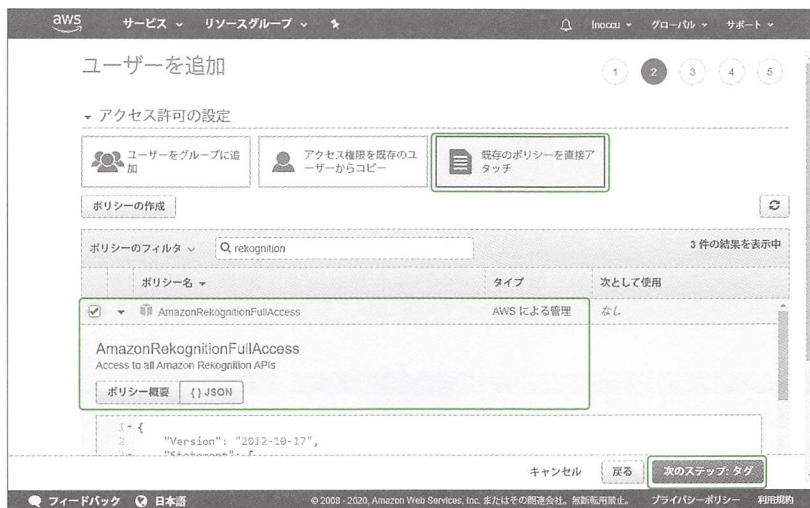


図 3-2-10 ポリシーのアタッチ

タグの設定は任意のため、特に指定する必要はありません。「次のステップ：確認」ボタンをクリックします。

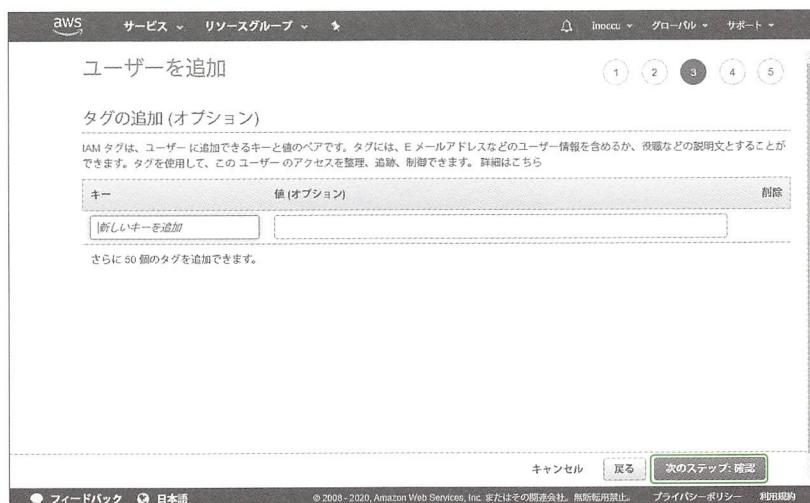


図 3-2-11 タグの追加

次の画面は、ここまで指定した内容の確認です。問題がなければ「ユーザーの作成」ボタンをクリックします。



図 3-2-12 確認

ユーザーの作成が完了すると、アクセスキー ID とシークレットアクセスキーが表示されます。アクセスキー ID は最初から画面に表示されており、いつでも再確認することができますが、シークレットアクセスキーは、最初はマスク表示となっており、表示リンクをクリックすることで表示されます。なお、シークレットアクセスキーが表示されるのは、この時点のみであり、後で再確認することはできません。必ず、メモを取るか、次項で説明する認証情報の設定の操作を行ってください。

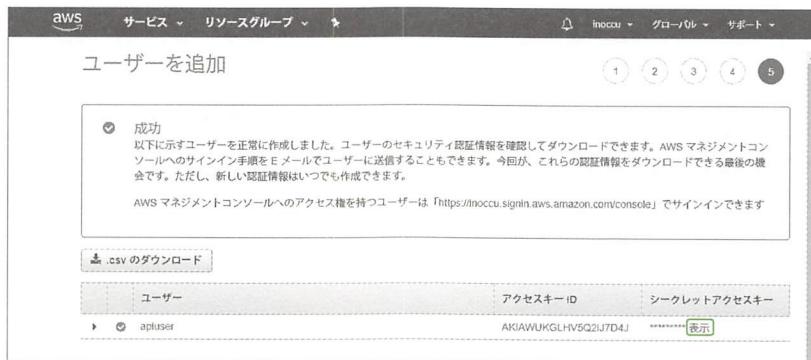


図 3-2-13 ユーザー追加の完了

3.2.5 認証情報の保存

前項で行ったユーザーの追加で最後に表示されたアクセスキー ID とシークレットアクセスキーを保存します^{*5}。まず、ホームディレクトリ（Windows では C:\Users\<ユーザー名>、macOS では /Users/<ユーザー名>）の直下に、.aws というフォルダを作成します。

次に、任意のテキストエディタを開きます。そして、.aws フォルダの中に、以下の内容のファイルを credentials というファイル名で保存します。

```
[default]
aws_access_key_id = <アクセスキーID>
aws_secret_access_key = <シークレットアクセスキー>
```

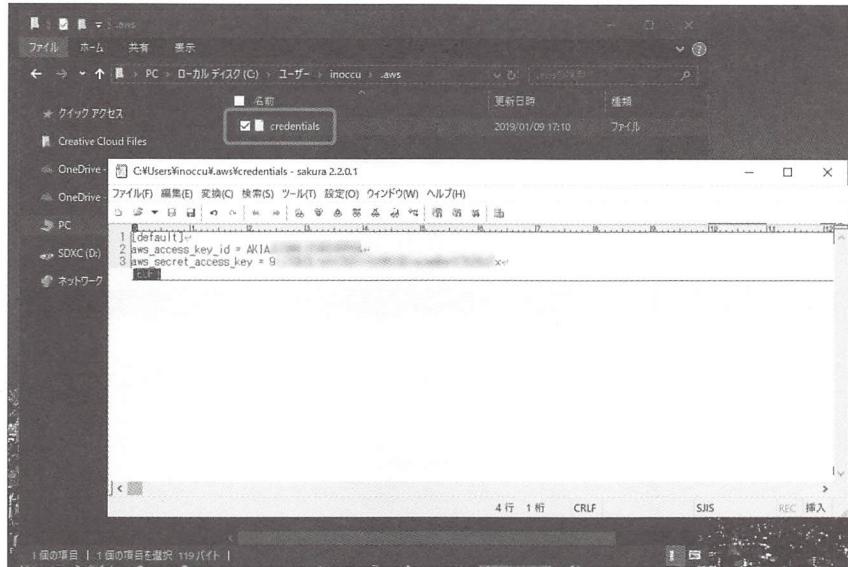


図 3-2-14 credentials ファイルの作成

これで、SDK で操作した際に AWS の認証が自動的にパスできるようになりました。

^{*5} AWS の認証情報の保存は、一般的には AWS CLI (Command Line Interface) をインストールして行いますが、本書では他の箇所で AWS CLI を使用しないため、直接テキストファイルを作成する方法で説明しました。

3.2.6 既存のユーザーへの権限の付与

AWS では、サービスを新たに使用するたびに、ユーザーに権限を付与する操作が必要となります。本書では、新たなサービスを使用する際にユーザーにアタッチすべきポリシーを、その都度説明します。先ほどは、新規のユーザーを作成して権限を付与する方法を説明しました。ここでは、既存のユーザーに権限を追加する操作について説明します。

先ほど作成した apiuser ユーザーに、よく使用するオブジェクトストレージサービスである S3 の権限を追加します。まず、IAM のユーザー画面で権限を付与したいユーザーをクリックし、詳細情報を表示させます。

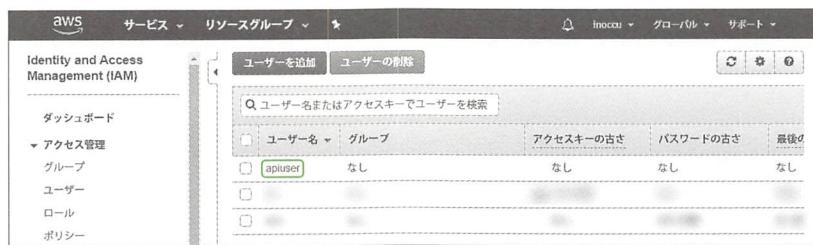


図 3-2-15 ユーザーの選択

アクセス権限タブにある「アクセス権限の追加」ボタンをクリックします。



図 3-2-16 アクセス権限の追加

ここから先の操作は、ユーザー作成時と概ね同様です。「既存のポリシーを直接アタッチ」を

選択し、S3 に関するすべての操作を行える「AmazonS3FullAccess」にチェック (✓) を入れます。そして、「次のステップ：確認」ボタンをクリックします。



図 3-2-17 ポリシーのアタッチ

確認画面が表示されるので、問題がなければ「アクセス権限の追加」ボタンをクリックします。



図 3-2-18 追加するアクセス権限の確認

3.2.7 Jupyter Notebook で AWS SDK for Python を使う

次に、Jupyter Notebook 上で AWS SDK for Python を使ってみましょう。まずは、SDK のインストールです。Jupyter Notebook で新規ノートブックを作成します。画面右上にある「New」ボタンをクリックし、「Python 3」を選択します。

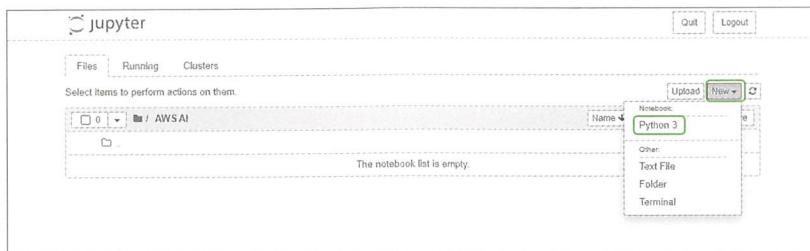


図 3-2-19 新規ノートブックの作成

Notebook には、セルと呼ばれるプログラムコードを入力する領域があり、ここに入力したコードをセルの単位で実行していきます。セルでの実行結果は次のセルでも有効なため、少しづつコードを書いて実行しながら動作を確認していくことが可能です。AWS SDK for Python には Boto3 という名前が付いています。この Boto3 を以下のコマンドを実行して導入します^{*6}。Notebook では、行の最初に “！” を付加すると、プログラムコードではなくコマンドとして認識されます。セルの実行は、画面の「Run」ボタンまたはセルの左側に表示されている「Run this cell」ボタンをクリックするか、Shift + Enter キーを押します。

```
!pip install --upgrade boto3
```

*6 Boto3 の詳細は、<https://aws.amazon.com/jp/sdk-for-python/> をご参照ください。

```
In [1]: pip install --upgrade boto3
Collecting boto3
  Downloading https://files.pythonhosted.org/packages/03/f9/98c5221d45b637ae1f42f0e0467e3bdfc3af46709b6bc7a
  29d93b2ecf6/boto3-1.10.46-py2.py3-none-any.whl (120B)
Collecting jmespath<1.0.0,>=0.7.1 (from boto3)
  Downloading https://files.pythonhosted.org/packages/03/94/7179c3932a6d45b266ddb2aac32ae01967fdb11f42f137
  71d27f225bb/jmespath-0.7.1-py2.py3-none-any.whl (120B)
Collecting s3transfer<0.3.0,>=0.2.0 (from boto3)
  Downloading https://files.pythonhosted.org/packages/74/17/40f9fb50219e8a24fd9c09c909d26f074d4a2fc83b68065df
  f3770ac6bb/botocore-1.13.46-py2.py3-none-any.whl (5.9MB)
Collecting s3transfer<0.3.0,>=0.2.0 (from boto3)
  Downloading https://files.pythonhosted.org/packages/16/8a/1fc3dbaa0c4923c2a78e1ff0d52b305c44806da63f718d143
  231e29c5fb0/s3transfer-0.2.1-py2.py3-none-any.whl (700B)
Requirement already satisfied: skipping upgrade: python-dateutil<3.0.0,>=2.1; python_version >= "2.7" in c:\users\inocu\anaconda3\lib\site-packages (from botocore<1.14.0,>=1.13.46->boto3) (2.8.0)
Requirement already satisfied: skipping upgrade: docutils<0.16,>=0.10 in c:\users\inocu\anaconda3\lib\site-packages (from botocore<1.14.0,>=1.13.46->boto3) (0.15.2)
Requirement already satisfied: skipping upgrade: urlib3<1.26,>=1.20; python_version >= "3.4" in c:\users\inocu\anaconda3\lib\site-packages (from botocore<1.14.0,>=1.13.46->boto3) (1.24.2)
Requirement already satisfied: skipping upgrade: idna<2.15,>=2.10 in c:\users\inocu\anaconda3\lib\site-packages (from botocore<1.14.0,>=1.13.46->boto3) (1.12.0)
Installing collected packages: jmespath, botocore, s3transfer, boto3
Successfully installed boto3-1.10.46 botocore-1.13.46 jmespath-0.9.4 s3transfer-0.2.1
```

図 3-2-20 Boto3 のインストール

Boto3 のインストール完了は、実行結果の最後に表示される「Successfully installed boto3…」というメッセージで確認できます。次のセルで、以下のプログラムを実行します。

```
import boto3

client = boto3.client('s3', region_name='ap-northeast-1')
client.list_buckets()
```

図 3-2-21 のように、エラーが出さずに結果が表示されれば、SDK のインストールおよび認証情報の保存が完了し、SDK を使って API を操作できます。

```
In [1]: import boto3
client = boto3.client('s3', region_name='ap-northeast-1')
client.list_buckets()

Out[1]: {'ResponseMetadata': {'RequestId': '0121AE911210F3E3',
  'HostId': 'PJM00QWpLC18NOMXA2z013bTbHb8WlgwN33c10xvjbvTxah7Iu0t8RY8wvc/VCMsPbHqj060=',
  'HTTPStatusCode': 200,
  'HTTPHeaders': ['x-amz-id-2': 'PJM00QWpLC18NOMXA2z013bTbHb8WlgwN33c10xvjbvTxah7Iu0t8RY8wvc/VCMsPbHqj060=',
    'x-amz-request-id': '0121AE911210F3E3',
    'date': 'Sat, 29 Feb 2020 04:55:30 GMT',
    'content-type': 'application/xml',
    'transfer-encoding': 'chunked',
    'server': 'AmazonS3'],
  'RetryAttempts': 0},
```

図 3-2-21 SDK の準備完了の確認

3.2.8 S3 バケットの作成とファイルのアップロード

AWS のサービスを利用する際には、合わせて S3 を使用することが多いです。例えば、Amazon Transcribe は音声データをテキストに変換するサービスですが、音声データはファイルサイズが大きくなりがちなため、いったんファイルを S3 にアップロードして、Amazon Transcribe の API には S3 上の URL をパラメータとしてセットする、というようなことを行います。

S3 にはバケットという領域を作成して、そこにファイルをアップロードしていくますが、バケットの作成時にはリージョンを指定します。Amazon Transcribeなどのサービスで S3 上のファイルを使用する際には、利用するサービスのリージョンと、ファイルをアップロードしたバケットのリージョンを一致させておく必要があります。基本的には東京リージョン（ap-northeast-1）のバケットを 1 つ作成しておけば事足りますが、AI 系のサービスには東京リージョンで展開されていないものがいくつかあり、その場合は、そのサービスが提供されているリージョンに合わせたバケットも必要になります。

S3 でバケットを作成するには、Web ブラウザで AWS コンソールを開き、S3 バケットの画面を表示します。

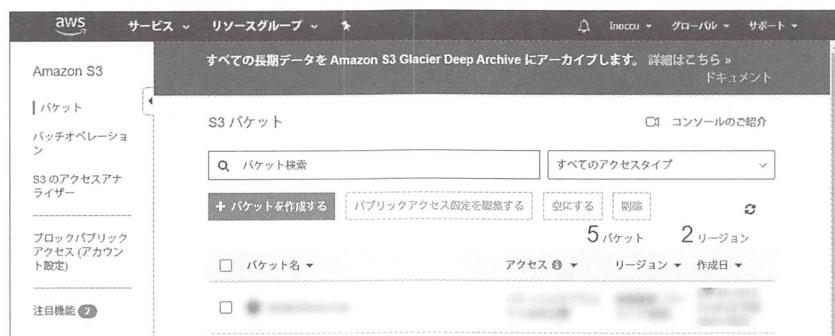


図 3-2-22 S3 バケットの作成

「バケットを作成する」ボタンをクリックすると、ウィザードが開きます。ここでは、任意のバケット名を付け、リージョンとしてアジアパシフィック（東京）を指定します。後は、ウィザードのデフォルトどおりに進めていくとバケットが作成されます。

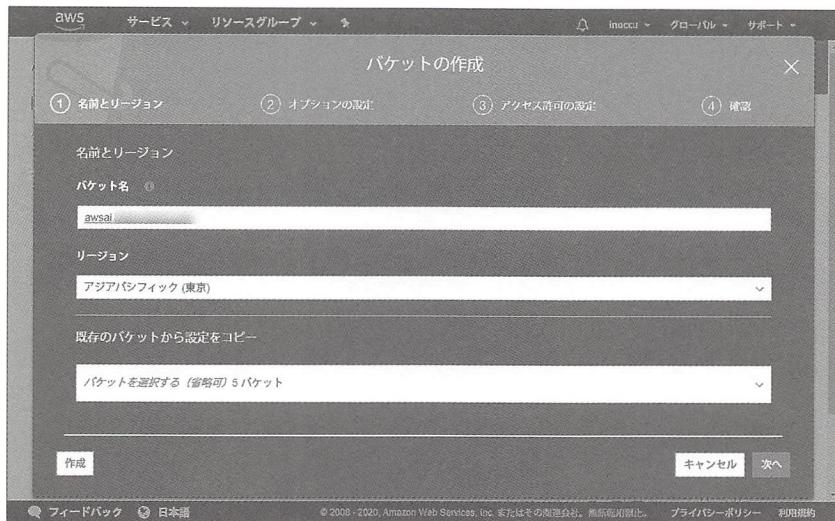


図 3-2-23 パケット名とリージョンの指定

パケットが作成されたら、ファイルをアップロードできます。作成したパケットの概要画面を開き、「アップロード」ボタンをクリックします。



図 3-2-24 パケットの概要画面

ファイルをアップロードするためのウィザードが表示されるので、アップロードする音声ファイルを指定します。後は、ウィザードのデフォルトどおりに進めます。

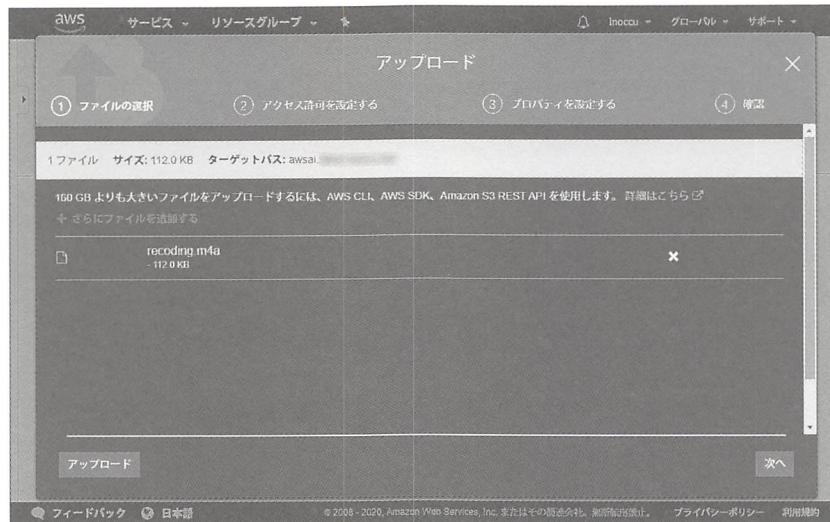


図 3-2-25 ファイルのアップロード

アップロードが完了すると、バケットの概要画面に戻ります。今アップロードしたファイルが表示されているので、それをクリックするとオブジェクト URL が表示されます。これが、API のパラメータとして指定する S3 上の URL です。



図 3-2-26 オブジェクト URL の確認

3.3

Amazon Rekognition

3.3.1 Amazon Rekognition とは

Amazon Rekognition（以下、Rekognition）^{*7} は、画像や動画の分析を行うためのサービスです。画像向けのサービスを特に Rekognition Image、動画向けのサービスを Rekognition Video と呼び、それらのサービスを総称したものが Rekognition です。Rekognition では、あらかじめ学習済みのモデルが提供されています。そのため、自分で機械学習用のデータを準備することなく、画像のラベル付け（モノとシーンの認識）や顔認識などの機能を使用することができます。

Rekognition で行えることを以下に示します。

- 画像や動画におけるモノとシーンの認識
- 画像や動画における顔の認識と分析（あらかじめ登録しておいた顔の認識を含む）
- 動画における人物の追跡
- 画像や動画における有名人の認識
- 画像や動画におけるアダルトコンテンツの認識
- 画像に含まれるテキストの認識

また、Amazon Rekognition Custom Labels を使用すれば、自ら準備した画像をトレーニングデータとして、独自の画像認識モデルを作成することができます。認識対象とする画像や動画は、あらかじめ S3 に保存しておいたものを使用するか、ローカルファイルをバイ二進データとしてアップロードし、使用することができます。動画については、保存済みの動画を認識するほか、ストリーミング動画の認識を行うことも可能です。

本節では、Rekognition の認識機能のうち、画像を用いたモノの認識や顔の認識について試してみることにしましょう。

^{*7} <https://aws.amazon.com/jp/rekognition/>

3.3.2 画像を用いたモノの認識

まず、画像を用いたモノの認識から始めましょう。ここでは、図 3-3-1 の犬（トイプードル）の画像（dog.jpg）を認識させてみます。

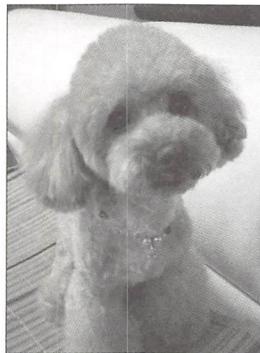


図 3-3-1 認識させる画像 (dog.jpg)

Jupyter Notebook で新規のノートブックを作成し、以下のコードを実行します。

```
import boto3
import json

client = boto3.client('rekognition', region_name='ap-northeast-1')

with open('dog.jpg', 'rb') as image_file:
    bytes_data = image_file.read()
    response = client.detect_labels(Image={'Bytes': bytes_data})

    print(json.dumps(response, indent=2))
```

ノートブックを作成したときと同じフォルダに画像（dog.jpg）を配置し、Python の open 関数で画像ファイルを開きます。AWS SDK（boto3）の Rekognition API では、認識させたいファイルを指定する際に Base64 エンコードのバイトデータを必要とするので、image_file.read() のようにしてバイトデータを取得します。実際に Rekognition を使って画像認識させているのは detect_labels() の部分です。認識結果は JSON 形式で得られるので、それを print 関数で表示しています。

```
In [3]: import boto3
import json

client = boto3.client('rekognition', region_name='ap-northeast-1')

with open('dog.jpg', 'rb') as image_file:
    bytes_data = image_file.read()
    response = client.detect_labels(Image={'Bytes': bytes_data})

print(json.dumps(response, indent=2))
```

[{"Labels": [{"Name": "Pet", "Confidence": 97.42623901367188, "Instances": [], "Parents": [{"Name": "Animal"}]}, {"Name": "Poodle", "Confidence": 97.42623901367188, "Instances": [], "Parents": [{"Name": "Pet"}]}]}

図 3-3-2 画像認識の実行結果

実行結果を見てみると、さまざまな出力が行われています。まず、この画像に付けられたラベル (Labels) を見ると、最初に Pet という値があります。つまり、この画像にペット (Pet) が写っていると認識しています。また、Confidence の数値は確信度を示しており、動物が写っているという認識結果にどの程度の確信を持っているかがわかります。0 ~ 100 の値で、数値が高いほど確信度が高いので、97.4 という値はかなり高い確度といえます。

次に、プードル (Poodle) という犬種についての結果も出ています。そこでは、Parents という属性に Pet という値が入っています。これはラベルの包含関係で、ペット (Pet) の中でもプードル (Poodle) であるという意味です。

{ "Name": "Dog", "Confidence": 97.42623901367188, "Instances": [{ "BoundingBox": { "Width": 0.9314640760421753, "Height": 0.9211114048957825, "Left": 0.03209564462304115, "Top": 0.07750921696424484 }, "Confidence": 96.78496551513672 }], "Parents": [{ "Name": "Animal" }] }

図 3-3-3 画像認識の実行結果 (続き)

ラベルの続きを見てみると、犬 (Dog) という値も見つかります。まさしく、この画像には犬が写っています。一部のラベルには、Instances 属性に、そのモノ (ここでは犬) が写っている

画像上の領域が BoundingBox の値として含まれています。BoundingBox の値が output されている場合は、その領域についての確信度 (Confidence) が、ラベル付けの確信度とは別に出力されます。

他にも、さまざまなラベルが付けられていますが、それらについては皆さんのが実際に認識させた画像で見てみると良いでしょう。

3.3.3 画像を用いた顔の認識

次に、顔認識も試してみましょう。認識させるのは図 3-3-4 の画像 (persons.jpg) です。



図 3-3-4 顔認識させる画像 (persons.jpg)

ノートブックの次のセルで、以下のコードを実行してみましょう。

```
import boto3
import json

client = boto3.client('rekognition', region_name='ap-northeast-1')

with open('persons.jpg', 'rb') as image_file:
    bytes_data = image_file.read()
    response = client.detect_faces(Image={'Bytes': bytes_data}, Attributes=['ALL'])

    print(json.dumps(response, indent=2))
```

コードのほとんどは先ほどのモノの認識の場合と同じですが、顔認識の場合は detect_faces() を使用します。また、引数として Attributes の値に ALL か DEFAULT のいずれかを指定します。DEFAULT では、顔の位置や顔の中の目や耳の位置が結果となります。一方、ALL では、年齢や笑顔か否かといったさまざまな項目まで認識します。また、1枚の画像に複数の人が写っている場合は、100 人までが認識されます。100 人以上の人気が写っている場合は、画像上に大きく写っ

ている人が優先されます。

```
In [1]: import boto3
import json

client = boto3.client('rekognition', region_name='ap-northeast-1')

with open('persons.jpg', 'rb') as image_file:
    bytes_data = image_file.read()
    response = client.detect_faces(Image={'Bytes': bytes_data}, Attributes=['ALL'])

print(json.dumps(response, indent=2))

{
  "FaceDetails": [
    {
      "BoundingBox": {
        "Width": 0.1288720667362213,
        "Height": 0.3454272150993347,
        "Left": 0.3725495934486389,
        "Top": 0.10323391109704971
      },
      "AgeRange": {
        "Low": 35,
        "High": 52
      },
      "Smile": {
        "Value": true,
        "Confidence": 58.496402740478516
      }
    }
  ]
}
```

図 3-3-5 顔認識の実行結果

実行結果を見てみましょう。前述のように複数の人が写っていても認識できるので、FaceDetails の値は配列になっています。今回は 2 人が写っているので、配列の要素数は 2 つです。その中身を見てみると、BoundingBox 属性で顔が写っている位置を示しています。図 3-3-5 には表示されていませんが、Landmarks 属性には右目、左目、口、鼻といった顔のパーツの位置が認識されています。また、今回は呼び出し時の Attributes 引数で ALL を指定しているので、さまざまな項目が認識されています。FaceDetails の 1 つ目の認識結果は、画像の左側の男性(筆者)です。年齢 (AgeRange) は 35 ~ 52 歳となっています。実際には 39 歳なので、正確に認識されているといえますが、少し高めの年齢になってしまったようです。また、Smile は true なので笑顔の写真といえますが、その確信度 (Confidence) は 58.49 です。たしかに微笑という程度の写真であり、強い確信を持って「笑顔」といえるほどではない、ということでしょう。

COLUMN Custom Labelsによる独自の画像認識モデルの作成

2019年12月、RekognitionにCustom Labelsというサービスが追加されました。Custom Labelsは、ユーザーが自ら持つ画像データをトレーニングデータとして、独自の画像認識モデルを作成できるサービスです。それまでRekognitionでは、AWSが提供するトレーニング済みのモデルを使って、一般的な物体のラベル付けなどができるのみで、例えばユーザー企業などが保有する特殊な製品の画像などを特有の名称でラベル付けすることはできませんでした。そのため、AWSで一般的でない物体のラベル付けを行うには、AWS Deep Learning AMIの環境上で、TensorFlowなどのディープラーニング用フレームワークを使って画像認識モデルを構築し、その上で画像を用いたトレーニングを行う必要がありました。しかし、Custom Labelsを使えば、そのような難易度の高い作業は不要となり、Rekognitionのコンソール操作か、APIを呼び出すことによって、モデルのトレーニングなどを行えます。

このようなAPIによる独自の画像認識モデルの作成については、IBM WatsonやMicrosoft Azureがサービス提供を既に行っており、AWSも先行サービスに追い付いた形といえます。Custom Labelsでは、モデルを作成・使用するAPIだけでなく、モデルの精度評価を行うAPIなども提供されており機能が充実しています。

3.4

Amazon Comprehend

3.4.1 Amazon Comprehend とは

Amazon Comprehend（以下、Comprehend）^{*8} は、自然言語のテキストからさまざまな情報を抽出するためのサービスです。例えば、そのテキストが日本語なのか英語なのかといったことや、テキストに含まれるキーフレーズなどを抽出します。Comprehend できることを以下に示します。

- 自然言語の識別（日本語か英語かなど）
- キーフレーズの抽出
- 場所、人物、ブランド、イベントといったエンティティの抽出
- 感情（センチメント）の抽出（肯定的か否定的か）

Comprehend では、抽出できる内容により対応言語が異なっています。日本語には自然言語の識別と、エンティティ、キーフレーズ、センチメントが対応しています。

ここでは、自然言語の識別と、エンティティの抽出を試してみることにしましょう。なお、Comprehend を使用する前に、IAM で Comprehend へのアクセス権限をユーザーに付与してください。既存のポリシーから「ComprehendFullAccess」をアタッチしておくと良いでしょう。

*8 <https://aws.amazon.com/jp/comprehend/>

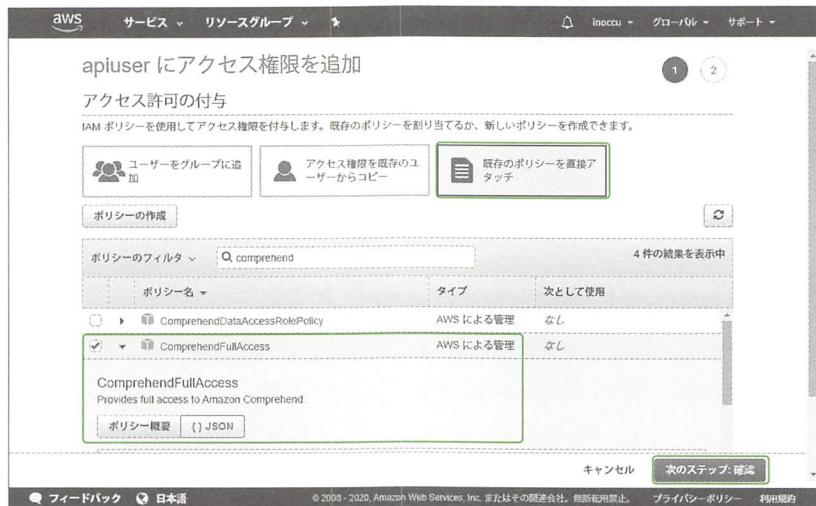


図 3-4-1 Comprehend へのアクセス権限の付与

3.4.2 自然言語を識別する

まず、自然言語の識別を行います。この機能は日本語に対応しているので、「貴社の記者が汽車で帰社した」というテキストを識別させることにしましょう。Jupyter Notebook のセルで以下のコードを実行します。

```
import boto3
import json

client = boto3.client('comprehend', region_name='us-east-1')

text = '貴社の記者が汽車で帰社した'

response = client.detect_dominant_language(Text=text)
print(json.dumps(response, indent=2))
```

自然言語の識別は、`detect_dominant_language()`で行います。Comprehend は、本書執筆時点では東京リージョン（ap-northeast-1）で公開されていないため、リージョンはバージニア北部（us-east-1）を指定していることに注意してください。引数の `Text` には識別させたいテキストを指定します。実行結果は図 3-4-2 のとおりです。

```
In [16]: import boto3
import json

client = boto3.client('comprehend', region_name='us-east-1')

text = '貴社の記者が汽車で帰社した'

response = client.detect_dominant_language(Text=text)
print(json.dumps(response, indent=2))

{
    "Languages": [
        {
            "LanguageCode": "ja",
            "Score": 0.999142587184906
        }
    ],
}
```

図 3-4-2 自然言語の識別の実行結果

認識結果は Languages の配列に入っています。この文章では ja（日本語）が識別されており、その確信度を示すスコア（Score）は 0.99 と極めて高い値を示しています。Comprehend が、日本語であると確信を持って答えていることがわかります。

3.4.3 エンティティを抽出する

次に、エンティティを抽出してみましょう。例文として、Amazon Comprehend の Web サイトに掲載されている紹介文を使うことにします^{*9}。

```
import boto3
import json

client = boto3.client('comprehend', region_name='us-east-1')
```

text = 'Amazon Comprehend は、機械学習を使用してテキスト内でインサイトや関係性を検出する自然言語処理 (NLP) サービスです。機械学習の経験は必要ありません。'

構造化されていないデータには膨大な量の潜在的な宝物があります。お客様の E メール、サポートチケット、製品レビュー、ソーシャルメディア、広告コピーが、ビジネスの役に立つ顧客感情のインサイトを表します。問題はそれをどのようにして手に入れるかです。このように、機械学習は、膨大な数のテキスト内の特定の関心項目（アナリストレポートで会社名を見つけるなど）を正確に特定することに特に優れており、言語の中に隠された感情（マイナスのレビュー や カスタマーサービスエージェントと顧客の積極的なのやりとりの特定）をほぼ無限の規模で学習することができます。'

*9 AWS のサイトにある Comprehend の説明の冒頭にあるテキストです。
(<https://aws.amazon.com/comprehend/>)

```
response = client.detect_entities(Text=text, LanguageCode='ja')
print(json.dumps(response, indent=2, ensure_ascii=False))
```

実行するコードはこのようなものです。エンティティの抽出には `detect_entities()` を使用します。抽出させたいテキストを `Text` 引数で指定するほか、言語を `LanguageCode` で指定します。今回は日本語のテキストなので、`ja` を指定しました。

```
In [9]: import boto3
import json

client = boto3.client('comprehend', region_name='us-east-1')

text = 'Amazon Comprehend は、機械学習を使用してテキスト内でインサイトや関係性を検出する自然言語処理 (NLP) サービスです'
response = client.detect_entities(Text=text, LanguageCode='ja')
print(json.dumps(response, indent=2, ensure_ascii=False))

[{"Entities": [{"Score": 1.0, "Type": "COMMERCIAL_ITEM", "Text": "Comprehend", "BeginOffset": 7, "EndOffset": 17}, {"Score": 0.7831317782402039, "Type": "OTHER", "Text": "自然言語", "BeginOffset": 49, "EndOffset": 59}]}]
```

図 3-4-3 エンティティの抽出の実行結果

実行結果は図 3-4-3 のようになりました。「Comprehend」、「自然言語」、「NLP」、「膨大な量の」などのエンティティが抽出され、「Comprehend」は商品（COMMERTIAL_ITEM）、「膨大な量の」は量（QUANTITY）であることも合わせて示されます。また、それぞれの抽出結果に対する確信度（Score）と、テキスト上の位置（BeginOffset と EndOffset）が結果に含まれています。

このように、Comprehend ではテキストからさまざまな情報を取り出すことができます。Comprehend を実際のサービスやシステムに組み込めば、人間がテキストをすべて読み込まなくても、必要な情報が含まれるテキストのみを読むといった機能を実現できるでしょう。また、Twitter や Instagram などの SNS 上の書き込みから自社の商品への評価を判断するようなこともできるかもしれません。

3.5 Amazon Textract

3.5.1 Amazon Textract とは

Amazon Textract（以下、Textract）^{*10} は、書類などを撮影した JPEG 形式または PNG 形式の画像から、テキストなどのデータを抽出するサービスです。単純に文字を認識するだけでなく、定型書類などのフォームからフィールドを認識し、そこに記載された値や、テーブル（表組）を認識して、その値を抽出することも可能です。また、抽出されたテキストが記載されていた画像上の座標データも取得できます。

一般的に、フォームからフィールドを認識するには、座標データなどをあらかじめ設定しておかなければなりませんが、Textract ではトレーニング済みの機械学習モデルが提供されているので、事前の準備は不要です。

書類を撮影した画像は、Base64 エンコーディングで Textract API のパラメータとして引き渡すか、S3 のバケットに保存されたファイルを指定します。取得されたデータは、JSON 形式の戻り値となります。

Textract を使用するためには、IAM でアクセス権限を付与しておく必要があります。既定のポリシーから「AmazonTextractFullAccess」を選択し、アタッチしておきましょう。また、ここでは S3 上のファイルを使って Textract の動作を確認するため、「AmazonS3FullAccess」というポリシーもアタッチしておきましょう。

*10 <https://aws.amazon.com/jp/textract/>

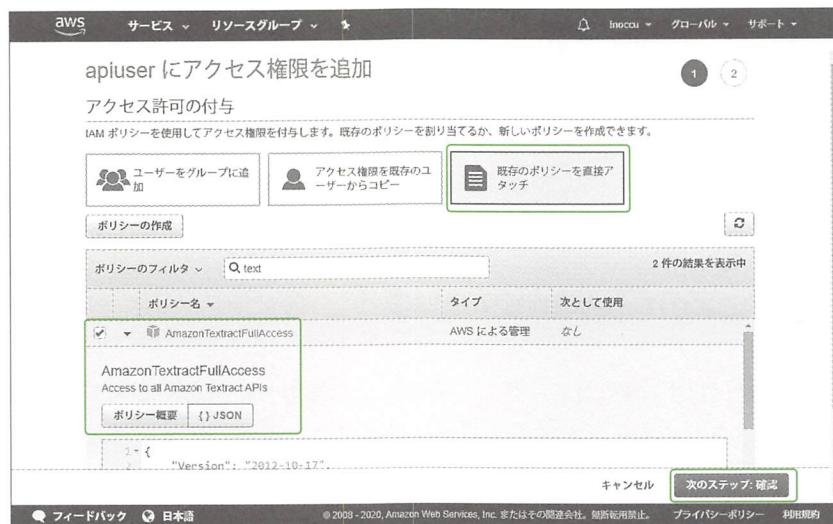


図 3-5-1 Textract へのアクセス権限の付与

3.5.2 英語の書類画像からデータを取得する

Extract は、本書執筆時点では残念ながら日本語に対応していません。そのため、英語の書類画像からテキストなどのデータを取得してみることにしましょう。ここでは、図 3-5-2 のような手書きが混ざった書類を認識させます。

Identity Verification Form and Affidavit

This form must be validated by a notary public to be effective.

In compliance with my request for access to the web services provided by Amazon Web Services, Inc., or continuation of those services, I, Kenichi Inoue, the undersigned, do hereby state and declare the following:

1. This affidavit concerns the following account (* indicates required):

Email Address of Account: ken@███████████

AWS Account # (if known): 11██████████8

Name on Account*: Kenichi Inoue

Billing Address*:

City: State: Zip*: Nakano, Tokyo, 164-0012 JP

Phone #: +81.3.██████████

You agree that we may contact you at this email address or phone number if we have additional questions.

2. I am the owner of the account referenced above and am requesting that the Multi-Factor Authentication be removed from my account.
3. I understand that knowingly making any false or fraudulent statement or representation may constitute a violation of federal, state, or local criminal statutes, and may result in imposition of a fine or imprisonment or both.

Signature: Kenichi Inoue

Date: Dec 5 2017

図 3-5-2 Textract に認識させる書類

Notebook で以下のようなコードを実行します。SDK では、テキストなどのデータを取得するために analyze_document() が用意されています。書類画像は引数の Document として指定し、S3 上のファイル^{*11} を指定する場合は S3Object という KEY で値をセットします。また、FeatureTypes には配列をセットし、フォームを認識させたい場合は FORMS、テーブルを認識させたい場合は TABLES という文字列を指定します。

```
import boto3
import json

client = boto3.client('textract', region_name='us-east-1')

response = client.analyze_document(
    Document={
        'S3Object': {
            'Bucket': '<格納先のS3バケット名>',
            'Name': '<格納先のS3オブジェクト名>',
            #'Version': '<S3でバージョン管理を行っている場合は、バージョン値>'
        }
    },
    FeatureTypes=['<フォームを認識させる場合はFORMS、テーブルを認識させる場合はTABLES>']
)

print(json.dumps(response, indent=2))
```

実行結果は図 3-5-3 のようになりました。

*11 ここでは米国東部（バージニア北部）リージョン（us-east-1）の Textract を使用するため、書類画像のファイルは同じリージョンの S3 バケットに格納されたものを使用します。

```

4 client = boto3.client('textract', region_name='us-east-1')
5
6 response = client.analyze_document(
7     Document={
8         'S3Object': {
9             'Bucket': 'awsai.████████',
10            'Name': 'page1.png',
11            '#Version': '<S3でバージョン管理を行なっている場合、バージョン値>'
12        }
13    },
14    FeatureTypes=['FORMS']
15 )
16
17 print(json.dumps(response, indent=2))
,
{
    "BlockType": "LINE",
    "Confidence": 99.27120971679688,
    "Text": "Identity Verification Form and Affidavit",
    "Geometry": {
        "BoundingBox": {
            "Width": 0.2985993027687073,
            "Height": 0.012637101113796234,
            "Left": 0.35433071851730347,
            "Top": 0.07879970222711563
        },
        "Polygon": [
            {
                "X": 0.35433071851730347,
                "Y": 0.07879970222711563
            },
            {
                "X": 0.6529300212860107,
                "Y": 0.07879970222711563
            }
        ]
    }
}

```

図 3-5-3 書類画像からデータを取得する

データは、JSON データとして取得されます。まず、書類画像上の要素はブロックとして認識され、Blocks という要素の値として配列がセットされます。ブロックには、PAGE や LINE といった種別（ブロックタイプ）があります。最初は PAGE という種別のブロックがあり、その領域は Geometry の値となります。ブロックの親子関係も認識されており、Relationships の値としてブロックの ID がセットされます。この PAGE ブロックには、多くの子ブロックがセットされます。

図 3-5-2 の書類画像のタイトル部分 (“Identity Verification Form and Affidavit”) の文字列が、図 3-5-3 の出力を見ると LINE ブロックとして認識され、その Text の値にセットされていることがわかります。画像上の文字をテキストとしてどの程度の確信度で認識できたかは、Confidence の値としてセットされます。Confidence は 0 ~ 100 の値で、数値が大きいほど確信度が高くなり、正しく認識されている可能性が高いことを示します。LINE ブロックにも PAGE ブロックと同様に Geometry の値がセットされるため、文字列が書類画像上のどこに書かれていたか、座標を知ることができます。

このように Textract では、手書きを含む書類画像からテキストと、そのテキストが記載されていた領域を簡単に抽出することができます。紙の書類がまだ多く残る日本では、特に必要性の高いサービスと思われるだけに、早期の日本語対応が期待されるところです。

3.6

Amazon Translate

3.6.1 Amazon Translate とは

Amazon Translate（以下、Translate）^{*12}は、その名のとおり自然言語の翻訳を行うサービスです。50を超える言語に対応しており、もちろん日本語も使用できます。また、日本語は、ソース言語としてもターゲット言語としても、Translateが対応しているすべての言語との間で翻訳が可能です。

Translateには、カスタム用語という機能があります。標準の翻訳機能では対応していないブランド名やキャラクター名といった独自の用語を、カスタム用語としてあらかじめ指定しておけば、その指定どおりに翻訳することができます。本節では、まず標準の（カスタム用語を使わない）翻訳を行い、その後でカスタム用語を使った翻訳を行います。そして、これら2つの翻訳結果にどのような差が出るかを試してみます。

これまでと同様に、あらかじめ Translateへのアクセス権限を付与しておきましょう。既定のポリシーから、「TranslateFullAccess」をアタッチします。



図 3-6-1 Translateへのアクセス権限の付与

*12 <https://aws.amazon.com/jp/translate/>

3.6.2 カスタム用語を使わない翻訳

まず、カスタム用語を使わずに翻訳してみます。使用するテキストは以下のとおりです。

Sukiyaki is a song by Japanese crooner Kyu Sakamoto, first released in Japan in 1961. The song topped the charts in several countries, including on the Billboard Hot 100 in 1963. The song has grown to become one of the world's best-selling singles of all time, having sold over 13 million copies worldwide.

これは、坂本九さんが歌った「上を向いて歩こう」の英語版 Wikipedia での説明^{*13}を、少し改変したものです。この曲は、日本では「上を向いて歩こう」というタイトルですが、海外では「Sukiyaki」というタイトルで知られています。

```
import boto3
import json

client = boto3.client('translate', region_name='us-east-1')

text = 'Sukiyaki is a song by Japanese crooner Kyu Sakamoto, first released in Japan in 1961. The song topped the charts in several countries, including on the Billboard Hot 100 in 1963. The song has grown to become one of the world's best-selling singles of all time, having sold over 13 million copies worldwide.'

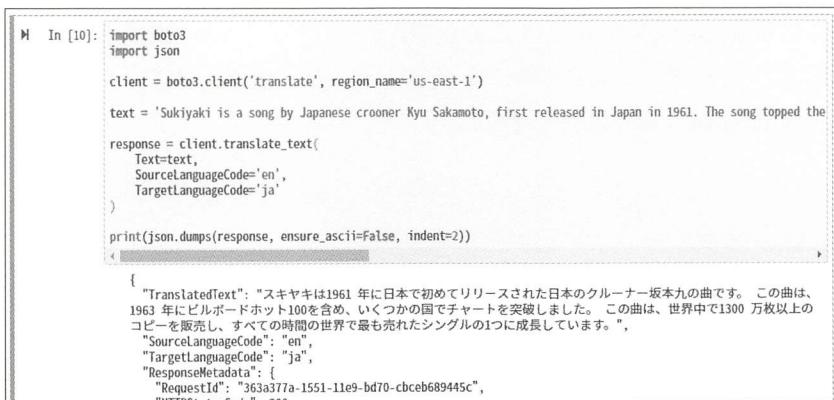
response = client.translate_text(
    Text=text,
    SourceLanguageCode='en',
    TargetLanguageCode='ja'
)

print(json.dumps(response, ensure_ascii=False, indent=2))
```

このようなコードを Jupyter Notebook で実行してみましょう。Translate は東京リージョン (ap-northeast-1) では提供されていないため、バージニア北部 (us-east-1) を指定しています。Python SDK では、translate_text() を使って Translate に翻訳させます。翻訳したい文章

^{*13} [https://en.wikipedia.org/wiki/Sukiyaki_\(song\)](https://en.wikipedia.org/wiki/Sukiyaki_(song))

は引数の Text で指定し、ソース言語コード (SourceLanguageCode) とターゲット言語コード (TragetLanguageCode) を指定します。今回は英語の文章を日本語に翻訳するので、ソース言語コードは en、ターゲット言語コードは ja となります。なお、ソース言語コードに auto と指定すると、Translate 側で自動的に言語が識別されます。



```

In [10]: import boto3
import json

client = boto3.client('translate', region_name='us-east-1')

text = 'Sukiyaki is a song by Japanese crooner Kyu Sakamoto, first released in Japan in 1961. The song topped the
        charts in Japan and worldwide, becoming a global hit. It features a simple melody and lyrics about a
        sukiyaki meal. The song has sold millions of copies worldwide and is still popular today.'

response = client.translate_text(
    Text=text,
    SourceLanguageCode='en',
    TargetLanguageCode='ja'
)

print(json.dumps(response, ensure_ascii=False, indent=2))

```

図 3-6-2 Translate の実行結果

実行結果は図 3-6-2 のとおりで、以下のような文章に翻訳されました。

スキヤキは 1961 年に日本で初めてリリースされた日本のクルーナー^{*14} 坂本九の曲です。この曲は、1963 年にビルボードホット 100 を含め、いくつかの国でチャートを突破しました。この曲は、世界中で 1300 万枚以上のコピーを販売し、すべての時間の世界で最も売れたシングルの 1 つに成長しています。

日本語として少しおかしな点もありますが、文意は正しく理解できる翻訳結果といえるでしょう。曲のタイトルはそのまま「スキヤキ」となっています。きちんとカタカナ表記になっていて、料理のすき焼きと混同することはないようです^{*15}。

3.6.3 カスタム用語を使った翻訳

前項の翻訳結果で気になるのは、「スキヤキ」というタイトルです。日本では「上を向いて歩こう」で知られているので、翻訳結果もそのようにしたいと思います。そこで、カスタム用語を使った

*14 クルーナーとは、低い声で感傷的に歌う歌手のことです。

*15 翻訳結果は API のバージョンにより異なるため、ここに掲載した翻訳結果とは異なる可能性があります。

翻訳を試してみることにしましょう。

カスタム用語は、CSV 形式のファイルや XML 形式の TMX (Translation Memory eXchange) ファイルで定義します。CSV 形式のファイルで定義する場合は、以下のようなフォーマットのファイルを作成します^{*16}。ここでは my_terminology.csv というファイル名で、UTF-8 形式のテキストファイルとして保存しました。ファイルの 1 行目に言語コードをセットします。そして、2 行目以降は、1 行目の言語コードの並びにそってカスタム用語を定義していきます。また、1 列目にはソース言語、2 列目以降にはターゲット言語を指定します。ターゲット言語として 2 つ以上の言語を指定することが可能です。

```
en,ja
Sukiyaki,上を向いて歩こう
```

カスタム用語の CSV ファイルを保存したら、import_terminology() を使って Translate にインポートします。引数には Name にインポートするカスタム言語の名前を付け、MergeStrategy には同じ Name で上書きするため OVERWRITE を指定します（本書執筆時点では OVERWRITE 以外の MergeStrategy には対応していません）。また、作成した CSV ファイルはバイトデータとして読み込み、TerminologyData で指定します。

```
import boto3
import json

client = boto3.client('translate', region_name='us-east-1')

with open('my_terminology.csv', 'rb') as mt:
    bytes_data = mt.read()
    response = client.import_terminology(
        Name='my_terminology',
        MergeStrategy='OVERWRITE',
        TerminologyData={
            'File': bytes_data,
            'Format': 'CSV'
        }
    )
    print(response)
```

^{*16} CSV データなので、カンマの前後にスペースを入れないようにしましょう。特に 1 行目の言語コード指定でスペースが入ると、対応していない言語コードとみなされ、エラーになります。

実行結果は図 3-6-3 のようになります。

```

In [9]: import boto3
import json

client = boto3.client('translate', region_name='us-east-1')

with open('my_terminology.csv', 'rb') as mt:
    bytes_data = mt.read()
    response = client.import_terminology(
        Name='my_terminology',
        MergeStrategy='OVERWRITE',
        TerminologyData={
            'File': bytes_data,
            'Format': 'CSV'
        }
    )
print(response)
{'TerminologyProperties': {'Name': 'my_terminology', 'Arn': 'arn:aws:translate:us-east-1:119310811178:terminology/my_terminology/LATEST', 'SourceLanguageCode': 'en', 'TargetLanguageCodes': ['ja'], 'SizeBytes': 40, 'TermCount': 1, 'CreatedAt': datetime.datetime(2019, 1, 16, 8, 56, 33, 696000, tzinfo=tzlocal()), 'LastUpdatedAt': datetime.datetime(2019, 1, 16, 9, 14, 17, 959000, tzinfo=tzlocal())}}, 'ResponseMetadata': {'RequestId': 'c7f32045-1923-11e9-b663-d15343a094be', 'HTTPStatusCode': 200, 'HTTPHeaders': {'content-type': 'application/x-amz-json-1.1', 'date': 'Wed, 16 Jan 2019 00:15:07 GMT', 'x-amzn-requestid': 'c7f32045-1923-11e9-b663-d15343a094be', 'content-length': '286', 'connection': 'keep-alive'}, 'RetryAttempts': 0}}

```

図 3-6-3 カスタム言語のインポート

インポートされているカスタム言語は、list_terminologies() で確認することができます。

```
print(client.list_terminologies())
```

実行結果は図 3-6-4 のとおりです。

```

In [14]: print(client.list_terminologies())
{'TerminologyPropertiesList': [{'Name': 'my_terminology', 'Arn': 'arn:aws:translate:us-east-1:119310811178:terminology/my_terminology/LATEST', 'SourceLanguageCode': 'en', 'TargetLanguageCodes': ['ja'], 'SizeBytes': 40, 'TermCount': 1, 'CreatedAt': datetime.datetime(2019, 1, 16, 8, 56, 33, 696000, tzinfo=tzlocal()), 'LastUpdatedAt': datetime.datetime(2019, 1, 16, 9, 14, 17, 959000, tzinfo=tzlocal())}], 'ResponseMetadata': {'RequestId': 'c7f32045-1923-11e9-b663-d15343a094be', 'HTTPStatusCode': 200, 'HTTPHeaders': {'content-type': 'application/x-amz-json-1.1', 'date': 'Wed, 16 Jan 2019 00:15:07 GMT', 'x-amzn-requestid': 'c7f32045-1923-11e9-b663-d15343a094be', 'content-length': '286', 'connection': 'keep-alive'}, 'RetryAttempts': 0}}

```

図 3-6-4 インポート済みのカスタム言語の表示

それでは、インポートしたカスタム言語を使って翻訳を試してみることにしましょう。使用するのは先ほどと同じ translate_text() ですが、引数に TerminologyNames を追加し、カスタム言語の Name を配列でセットします。

text = 'Sukiyaki is a song by Japanese crooner Kyu Sakamoto, first released in Japan in 1961. The song topped the charts in several countries, including on the Billboard Hot 100 in 1963. The song has grown to become one of the world's best-selling singles of all time, having sold over 13 million copies worldwide.'

```
response = client.translate_text(

```

```
    Text=text,
```

```
TerminologyNames=['my_terminology'],
SourceLanguageCode='en',
TargetLanguageCode='ja'

)
print(json.dumps(response, ensure_ascii=False, indent=2))
```

実行結果は図 3-6-5 のとおりです。先ほどは「スキヤキ」と翻訳されていた部分が、「上を向いて歩こう」に変わっていることがわかります。

In [16]:

```
text = 'Sukiyaki is a song by Japanese crooner Kyu Sakamoto, first released in Japan in 1961. The song topped the
response = client.translate_text(
    Text=text,
    TerminologyNames=['my_terminology'],
    SourceLanguageCode='en',
    TargetLanguageCode='ja'
)
print(json.dumps(response, ensure_ascii=False, indent=2))
```

{ "TranslatedText": "上を向いて歩こうは1961 年に日本で初めてリリースされた日本のクルーナー坂本九の曲です。この曲は、1963 年にビルボードホット100 を含め、いくつかの国でチャートを突破しました。この曲は、世界中で1300 万枚以上のコピーを販売し、すべての時間の世界で最も売れたシングルの1つに成長しています。", }

「スキヤキ」が「上を向いて歩こう」に変わっている

図 3-6-5 カスタム言語を使用した翻訳結果

3.7

Amazon Transcribe

3.7.1 Amazon Transcribe とは

Amazon Transcribe（以下、Transcribe）^{*17} は、音声を認識してテキストに変換するサービスです。FLAC、MP3、MP4、WAV のいずれかの形式で、最長 2 時間の音声ファイルを使用することができます。これまでに紹介した他のサービスとは異なり、同期型での処理を行うことができず、すべて非同期型^{*18}での処理となります。音声ファイルはあらかじめ S3 にアップロードしておき、その URL を Transcribe の呼び出し時に指定します。

Transcribe では、音声をテキストに変換するだけでなく、話者の識別を行うこともできます。また、カスタム語彙を設定して、特殊な用語を正確に変換させることも可能です。英語やフランス語などのほか、日本語にも対応しています。

あらかじめ Transcribe へのアクセス権限を付与しておきましょう。既定のポリシーから、「AmazonTranscribeFullAccess」をアタッチします。また、S3 へのアクセス権限も必要なため、「3.2.6 既存のユーザーへの権限の付与」を参考にして「AmazonS3FullAccess」をアタッチしておきます。

*17 <https://aws.amazon.com/jp/transcribe/>

*18 何らかの処理を呼び出し、制御が戻った際に処理が完了して結果を得ることができる処理を同期処理といいます。逆に、制御が戻ってきてても処理自体が終わったわけではなく、処理結果は別に取得する必要のある処理を非同期処理といいます。非同期処理は、時間のかかる処理を行う場合によく用いられます。

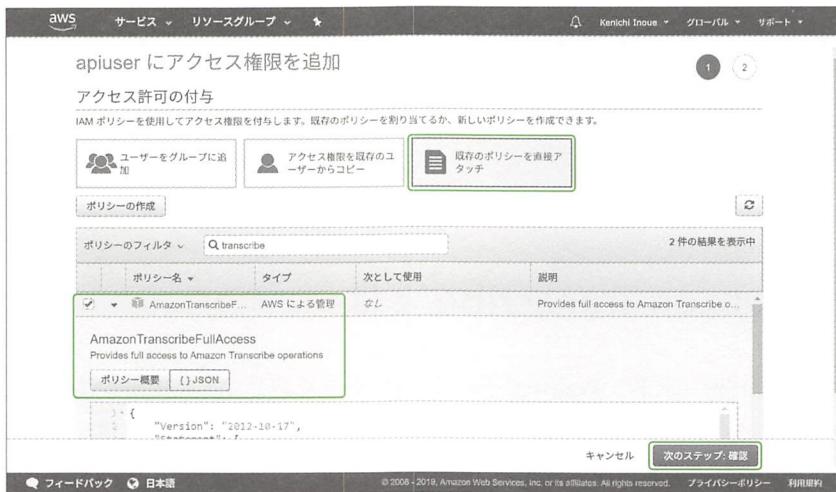


図 3-7-1 Transcribeへのアクセス権限の付与

3.7.2 日本語の音声ファイルの認識

ここでは、音声ファイルを認識させてみます。ご自身でお持ちの音声ファイルがあれば、それを使って試してみてください。お持ちでない方は、Windows 10 では標準で導入されているボイスレコーダーで、macOS では Quick Time Player を用いて自分の声などを録音してみても良いでしょう。

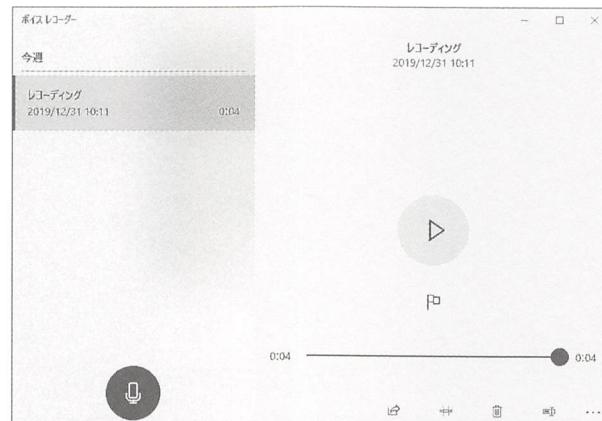


図 3-7-2 Windows10 のボイスレコーダーで録音

適当な音声ファイルを準備したら、S3 にアップロードします。Transcribe で使用する音声ファイルは、Transcribe が稼働しているのと同じリージョン内の S3 バケットにあるものでなければなりません。Transcribe は東京リージョン (ap-northeast-1) でも提供されているので、S3 バケットも東京リージョンに準備しましょう。

The screenshot shows the AWS S3 console interface. At the top, there's a navigation bar with 'aws' logo, 'サービス' (Services), 'リソースグループ' (Resource Groups), and user information 'Kenichi Inoue'. Below the navigation is a search bar with placeholder 'バケット検索' (Bucket search) and a dropdown for 'すべてのアクセスタイプ' (All access types). A large green button labeled '+ バケットを作成する' (Create a bucket) is prominently displayed. To its right, there are filters for 'バッリックアクセス設定を適用する' (Apply public access settings), '空にする' (Empty), and '削除' (Delete). The status shows '1 リージョン' (1 Region) and '9 バケット' (9 Buckets). The main table lists four existing buckets, each with a delete icon and a link to its details page. The first bucket, 'transcribe-test-bucket', is highlighted. It has a status of 'オブジェクトは公開可能' (Object is public), 'アジアパシフィック (東京)' (Asia Pacific (Tokyo)), and was created on '9月 14, 2017' at '5:57:17 午後' (5:57:17 PM) with 'GMT+0900'. The second bucket was created on '3月 25, 2017' at '4:55:03 午後' (4:55:03 PM) with 'GMT+0900'. The third bucket was created on '3月 25, 2017' at '3:36:50 午後' (3:36:50 PM) with 'GMT+0900'. The fourth bucket was created on '12月 12, 2018' at '12:55:47 午後' (12:55:47 PM) with 'GMT+0900'. At the bottom, there are sections for 'オペレーション' (Operations) with '0 進行中' (0 in progress), '2 成功' (2 successful), and '0 エラー' (0 errors); and 'フィードバック' (Feedback) and '日本語' (Japanese) buttons.

図 3-7-3 S3 バケットの作成

音声ファイルの S3 へのアップロードが完了したら、新規の Notebook を作成して次ページのコードを実行します。Transcribe のクライアントを作成する際に、リージョンとして ap-northeast-1 を指定していることに注意してください。S3 バケットの作成時に説明したように、Transcribe のリージョンと S3 バケットのリージョンを一致させておく必要があります。

Transcribe で音声認識を行うには、`start_transcription_job()` を使用します。引数は、`TranscriptionJobName` に適当なジョブの名前を指定します。なお、ジョブの名前は、1 つの AWS アカウントの中で重複することはできないので注意が必要です。`LanguageCode` は音声ファイルの言語コードです。今回は日本語で収録した音声を使用するので、ja-JP を指定しました。`MediaFormat` は、mp3 / mp4 / wav / flac のいずれかを指定します。Windows 10 のボイスレコーダーで録音した音声ファイルの拡張子は m4a になっていますが、`MediaFormat` には mp4 を指定してください。最後に、`Media` 引数で辞書型の値をセットし、その中の `MediaFileUri` に音声ファイルの S3 でのオブジェクト URL を指定します。また、この引数で `OutputBucketName` を指定することにより、処理結果を、指定したバケットに格納することもできます。

```

import boto3
import json

client = boto3.client('transcribe', region_name='ap-northeast-1')

response = client.start_transcription_job(
    TranscriptionJobName='<ジョブ名>',
    LanguageCode='ja-JP',
    MediaFormat='<mp3 / mp4 / wav / flacのいずれか>',
    Media={
        'MediaFileUri': '<音声ファイルのオブジェクトURL>'
    }
)

print(response)

```

実行結果は図 3-7-4 のようになりました。



```

In [3]: import boto3
import json

client = boto3.client('transcribe', region_name='ap-northeast-1')

response = client.start_transcription_job(
    TranscriptionJobName='transcribe_2020010401',
    LanguageCode='ja-JP',
    MediaFormat='mp4',
    Media=[
        {'MediaFileUri': 'https://s3-ap-northeast-1.amazonaws.com/awsal.../recoding.m4a'}
    ]
)
print(response)

[{"TranscriptionJob": {"TranscriptionJobName": "transcribe_2020010401", "TranscriptionJobStatus": "IN_PROGRESS", "LanguageCode": "ja-JP", "MediaFormat": "mp4", "Media": [{"MediaFileUri": "https://s3-ap-northeast-1.amazonaws.com/awsal.../recoding.m4a"}], "StartTime": datetime.datetime(2020, 1, 4, 15, 43, 25, 35100, tzinfo=tzlocal()), "CreationTime": datetime.datetime(2020, 1, 4, 15, 43, 25, 295000, tzinfo=tzlocal()), "ResponseMetadata": {"RequestId": "288093cf-366b-4f11-961a-1leledadclf6", "HTTPStatusCode": 200, "HTTPHeaders": [{"Content-Type": "application/x-amz-json-1.1", "Date": "Sat, 04 Jan 2020 06:43:24 GMT", "X-Amzn-Request-Id": "288093cf-366b-4f11-961a-1leledadclf6", "Content-Length": "315", "Connection": "keep-alive"}, {"Retry-Attempts": 0}]}]

```

図 3-7-4 音声認識の実行

本章でこれまで紹介してきたサービスとは異なり、音声認識の結果がすぐに返ってくるわけではありません。その代わりに、TranscriptionJobStatus として IN_PROGRESS がセットされています。これは、音声認識ジョブの状態が進行中であることを示しています。

このように、Transcribe による音声認識処理は非同期で行われます。処理が完了しているかは、get_transcription_job() を行うことで確認できます。引数の TranscriptionJobName には、start_transcription_job() の実行時にセットしたものと同じ値を指定します。

3

```
response = client.get_transcription_job(  
    TranscriptionJobName='<ジョブ名>'  
)  
print(response)
```

同じ Notebook の次のセルに上記のコードを入力し、しばらく待ってから実行すると、図 3-7-5 のように TranscriptionJobStatus が COMPLETED となります。処理が完了すると、音声認識の結果にアクセスするための URL (TranscriptFileUri) が発行されます。

```
In [4]: response = client.get_transcription_job(
    TranscriptionJobName='transcribe_2020010401'
)
print(response)

{'TranscriptionJob': {'TranscriptionJobName': 'transcribe_2020010401', 'TranscriptionJobStatus': 'COMPLETED', 'LanguageCode': 'ja-JP', 'MediaSampleRateHertz': 48000, 'MediaFormat': 'mp4', 'Media': {'MediaFileUri': 'https://s3-ap-northeast-1.amazonaws.com/awssamples-recoding/m4a'}, 'Transcript': {'TranscriptFileUri': 'https://s3.ap-northeast-1.amazonaws.com/aws-transcribe-ap-northeast-1-prod/455950941563/transcribe_202010401/fle997a-825d-4e2d-a22f-c2215af3486d/asrOutput.json?X-Amz-Security-Token=IQoJb3JpZ2lJUx2VjEN7%2F%2F%2FX2F2', 'TranscriptFileUrl': 'https://s3.ap-northeast-1.amazonaws.com/aws-transcribe-ap-northeast-1-prod/455950941563/transcribe_202010401/fle997a-825d-4e2d-a22f-c2215af3486d/asrOutput.json?X-Amz-SignedHeaders=host&X-Amz-Expires=800&X-Amz-Credential=ASIAQHH062CNUQZ77JUY%2F20200104%2Fap-northeast-1%2Fs3%2Faws4_request&X-Amz-Signature=9dcab065810e9a47eb9887103aa254ef1fd2bc277c4a70eabb20bcff5093590b'}, 'StartTime': datetime.datetime(2020, 1, 4, 15, 43, 25, 351000, tzinfo=tzlocal()), 'CreationTime': datetime.datetime(2020, 1, 4, 15, 43, 25, 295000, tzinfo=tzlocal()), 'CompletionTime': datetime.datetime(2020, 1, 4, 15, 43, 59, 710000, tzinfo=tzlocal()), 'Settings': {'ChannelIdentification': False, 'ShowAlternatives': False}, 'ResponseMetadata': {'RequestId': 'f663efd7-4f3a-4f1e-bee3-518fcfa3c331a', 'HTTPStatusCode': 200, 'HTTPHeaders': {'content-type': 'application/x-amz-json-1.1', 'date': 'Sat, 04 Jan 2020 06:48:17 GMT', 'x-amzn-requestid': 'f663efd7-4f3a-4f1e-bee3-518fcfa3c331a', 'content-length': '2079', 'connection': 'keep-alive'}, 'RetryAttempts': 0}}
```

図 3-7-5 音声認識処理の完了

音声認識の結果を参照してみましょう。ここでは、Python コードを Notebook 上で実行することで参照していますが、TranscriptFileUri には、アクセスするための認証に必要なトークン (X-Amz-Security-Token) が URL パラメータとして含まれているので、Web ブラウザなどで直接アクセスすることも可能です。

```
import urllib.request

r = urllib.request.urlopen(response['TranscriptionJob']['Transcript']
                           ['TranscriptFileUri'])
json_data = json.loads(r.read())
print(json.dumps(json_data, indent=2, ensure_ascii=False))
```

図 3-7-6 のように、音声認識の結果は JSON 形式のファイルとなっています。ここでは、transcript の値として音声認識の結果がセットされています。

```
In [6]: import urllib.request  
r = urllib.request.urlopen(response['TranscriptionJob']['Transcript'][  
    'TranscriptFileUri'])  
json_data = json.loads(r.read())  
print(json.dumps(json_data, indent=2, ensure_ascii=False))  
  
{  
    "jobName": "transcribe_2020010401",  
    "accountId": "455950941563",  
    "results": [  
        "transcripts": [  
            {  
                "transcript": "吾輩は猫で、ある名前はまだない"  
            }  
        ],  
        "items": [  
            {  
                "start_time": "1.02",  
                "end_time": "1.62",  
                "alternatives": [  
                    {  
                        "confidence": "0.8296",  
                        "content": "吾輩"  
                    }  
                ]  
            }  
        ]  
    ]  
}
```

図 3-7-6 音声認識の結果ファイル

3.8 Amazon Polly

3.8.1 Amazon Polly とは

Amazon Polly（以下、Polly）^{*19} は、テキストを音声に変換するサービスです。前節の Transcribe は音声をテキストに変換するサービスでしたが、Polly はその逆を行います。本書執筆時点で 28 の言語に対応しており、そこには日本語も含まれています。また、多くの言語では男性の声と女性の声の両方への変換が可能であり、言語によっては男性、女性それぞれで複数の種類の声が提供されています。日本語の場合、男性の声として Takumi、女性の声として Mizuki があり、こうした人の名前のような声の識別子をボイス ID といいます。なお、単語に特殊な発音がある場合などは、発音レキシコン^{*20}を使用することでカスタマイズできます。

変換された音声は、S3 のバケットに保存されるか、ストリーミングデータとして取得できます。ここでは、S3 バケットへの保存を試してみることにしましょう。Polly を使用するためには、IAM でアクセス権限を付与しておく必要があります。既定のポリシーから「AmazonPollyFullAccess」を選択し、アタッチしておきましょう。また、音声ファイルの格納先となる S3 への書き込み権限も必要です。「3.2.6 既存のユーザーへの権限の付与」を参考にして「AmazonS3FullAccess」をアタッチしておきます。

*19 <https://aws.amazon.com/jp/polly/>

*20 例えば IEEE という単語は、米国電気電子学会を指すなら「アイ・トリプル・イー」と発声しますが、別の意味で「アイ・イー・イー・イー」と発声することも考えられます。このような場合、発音レキシコンを使用して、発音の方法をコントロールすることができます。詳細は、https://docs.aws.amazon.com/ja_jp/polly/latest/dg/managing-lexicons.htmlをご参照ください。



図 3-8-1 Polly へのアクセス権限の付与

3.8.2 日本語のテキストを音声に変換する

Polly は日本語に対応しているので、日本語のテキストを音声ファイルに変換してみましょう。Notebook で以下のようなコードを実行します。SDK では `start_speech_synthesis_task()` が、音声ファイルへの変換を行うために準備されています。これは、前節の Transcribe と同様に `start_` から始まる関数であり、非同期で処理が行われることを示しています。引数の `OutputFormat` は `mp3`、`ogg_vorbis`、`pcm` のいずれかを指定できます。また、`OutputS3BucketName` は、音声ファイルを保存する S3 バケット名で、Polly のクライアントを作成する際に指定したものと同じリージョンにある S3 バケットを指定します。

音声に変換したいテキストは引数の `Text` として指定し、使用する音声（ボイス ID）は `VoiceId` の値として指定します。言語コード（`LanguageCode`）を引数に指定することもできますが、指定したボイス ID がバイリンガルでない場合は、指定する必要はありません^{*21}。

```
import boto3
import json

client = boto3.client('polly', region_name='ap-northeast-1')
```

*21 Aditi というボイス ID が、インド英語とヒンディー語の両方に対応するバイリンガルとして提供されています。

```

text = 'こんにちは。私はAmazon Pollyのミズキです。'

response = client.start_speech_synthesis_task(
    OutputFormat='<mp3 / ogg_vorbis / pcmのいずれか>',
    OutputS3BucketName='<格納先のS3バケット名>',
    Text=text,
    VoiceId='Mizuki'
)

print(response)

```

実行結果は図 3-8-2 のようになりました。

```

In [1]: import boto3
import json

client = boto3.client('polly', region_name='ap-northeast-1')

text = 'こんにちは。私はAmazon Pollyのミズキです。'

response = client.start_speech_synthesis_task(
    OutputFormat='mp3',
    OutputS3BucketName='awsai.',
    Text=text,
    VoiceId='Mizuki'
)

print(response)

```

['ResponseMetadata': {'RequestId': '4ff15f4f-5546-4575-b43b-54d1ee5f0388', 'HTTPStatusCode': 200, 'HTTPHeaders': {'Content-Type': 'application/json', 'Date': 'Sat, 04 Jan 2020 07:06:45 GMT', 'X-Amzn-Requestid': '4ff15f4f-5546-4575-b43b-54d1ee5f0388', 'Content-Length': '461', 'Connection': 'keep-alive'}, 'RetryAttempts': 0}, 'SynthesisTask': {'TaskId': '7c0d7357-bbb0-4890-9857-475877c16820', 'TaskStatus': 'scheduled', 'OutputUri': 'https://s3.ap-northeast-1.amazonaws.com/awsa1/7c0d7357-bbb0-4890-9857-475877c16820.mp3', 'CreationTime': datetime.datetime(2020, 1, 4, 16, 6, 45, 71000, tzinfo=tzlocal()), 'RequestCharacters': 27, 'OutputFormat': 'mp3', 'TextType': 'text', 'VoiceId': 'Mizuki'}}]

TaskId と TaskStatus が戻り値にセットされている

図 3-8-2 音声ファイルへの変換

音声ファイルに変換する場合、その変換処理はタスクとして非同期で実行されます。そのため、戻り値には TaskId と TaskStatus がセットされています。TaskId は自動的に発行され、Polly でのタスクの処理状況を確認するために使用します。一方、TaskStatus は、タスクの処理状況を示しますが、start_speech_synthesis_task() を実行した直後は scheduled になっています。

タスクの処理状況は、以下のコードで確認できます。

```

r = client.get_speech_synthesis_task(TaskId=response['SynthesisTask']['TaskId'])
print(r)

```

処理が完了すると、TaskStatus が completed となります。

```
In [2]: r = client.get_speech_synthesis_task(TaskId=response['SynthesisTask']['TaskId'])
print(r)

[{"ResponseMetadata": {"RequestId": "a1688642-5157-43bf-aa4b-441585720579", "HTTPStatusCode": 200, "HTTPHeaders": {"Content-Type": "application/json", "Date": "Sat, 04 Jan 2020 07:07:52 GMT", "x-amzn-requestid": "a1688642-5157-43bf-aa4b-441585720579", "Content-Length": "461", "Connection": "keep-alive"}, "RetryAttempts": 0}, "SynthesisTask": {"TaskId": "7c0d7357-bbb0-4890-9957-475877c16820", "TaskStatus": "completed", "OutputUri": "https://s3.ap-northeast-1.amazonaws.com/awsal-test/7c0d7357-bbb0-4890-9957-475877c16820.mp3", "CreationTime": "2020-01-04T07:07:52Z", "Text": "こんにちは。私はAmazon Pollyのミズキです。", "VoiceId": "Mizuki"}]}
```

図 3-8-3 処理状況の確認

処理が完了したら、音声ファイルの保存先として指定した S3 バケットを見てみましょう。`start_speech_synthesis_task()` および `get_speech_synthesis_task()` の戻り値にある `OutputUri` で示されたファイルが保存されていると思います。

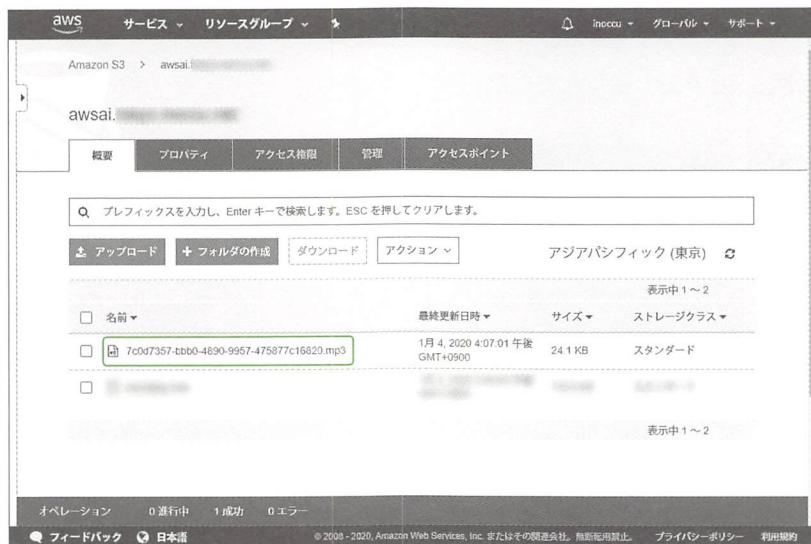


図 3-8-4 音声ファイルが保存された S3 バケット

保存された音声ファイルをダウンロードし、再生してみると、女性の声で「こんにちは。私は Amazon Polly のミズキです。」と発声しました。書籍では音を聞くことができませんが、皆さんも是非試してみてください。

3.9 Amazon Lex

3.9.1 Amazon Lex とは

Amazon Lex（以下、Lex）^{*22} は、音声やテキストを用いた対話型インターフェースを作成するためのサービスです。チャットボットを作るためのサービスといったほうが、わかりやすいかもしれません。AmazonにはAlexaというAIアシスタントが存在しており、Amazon Echoなどのスマートスピーカーに搭載されています。Lexは、Alexaと同じ技術を用いて開発されています。また、Lexを用いてAlexa用のスキル^{*23}を開発することも可能です。

本書執筆時点において、Lexは日本語に対応していません。そのため、本書ではあらかじめ用意されているLexのサンプルのチャットボットを使いながら、機能を説明していきます。

3.9.2 チャットボットを実現するための技術

チャットボットは、ユーザーとコンピュータの間で会話を実現するための技術ですが、そこでのようなAI技術が用いられているのでしょうか。

よく、会話はキャッチボールといわれます。ある人が何らかの言葉を相手に投げかけ、その相手はまた何らかの言葉を返します。チャットボットでいえば、（人間の）ユーザーがコンピュータに対して何らかの言葉を投げかけると、コンピュータがユーザーにその返事をしてくれる、というわけです。このとき、コンピュータがやらなければならないことは、ユーザーが投げかけてきた言葉を理解することです。ユーザーが何を言ったのかがわからなければ、チャットボットは返事をすることができません。また、ユーザーの言葉を理解できたとしても、適切な返事の仕方を知らなければ、チャットボットはユーザーに言葉を投げ返すことができません。

コンピュータがユーザーの言葉を理解するためには、自然言語理解（NLU：Natural Language Understanding）の技術が必要です。また、ユーザーが音声で言葉を投げかける場合（Alexaのような音声チャットボットの場合）は、自動音声認識の技術も必要となります。本章でここまで

*22 <https://aws.amazon.com/jp/lex/>

*23 特定の内容について会話できるチャットボットの能力のことを、Alexaではスキルと呼んでいます。

紹介してきた AWS の AI サービスの中では、Comprehend が自然言語理解の技術を搭載しています。また、Transcribe が音声認識のサービスです。このように、チャットボットを実現するためにはさまざまな技術が必要となります、Lex は、AWS の他の AI サービスが持っている技術なども統合し、ワンストップで使えるようにしたサービスといえるかもしれません。

ちなみに、ユーザーから投げかけられた言葉を理解した後、どのような返事をすべきかは Lex の中で定義することができ、その返事を音声でしなければならない場合は Polly の音声発話サービスを使用します。Lex そのものには音声発話の機能はないので注意が必要です。

3.9.3 AWS Lambda との連携

Lex では、チャットボットにおける自然言語理解などの技術を用いて会話のフローを制御することができます。しかし、会話の目的がレストランの予約だった場合、「いつ誰が来るのか」、「何人で来るのか」といった予約についての会話が流暢にできれば、それで満足といえるでしょうか。答えは「ノー」です。「いつ、誰が、何人で」という情報を記録しておかないとけませんし、そもそも空席があるか否かの確認も必要です。つまり、チャットボットがきちんと役目を果たすためには、会話だけできいてもダメだということです。

データの記録や空席の確認のような会話以外の機能（いわゆるビジネスロジック）については、Lex 自体では実現できず、AWS の別のサービスである AWS Lambda（以下、Lambda）との連携が必要です。Lambda は、FaaS（Function as a Service）というカテゴリのサービスであり、Python や Node.js といったプログラム言語で作成されたロジックを AWS に登録しておき、必要に応じて実行することができるものです。ビジネスロジックを Lambda で実装し、それを適宜 Lex から呼び出すことで、実際に役立つチャットボットを開発することができます。

3.9.4 LexModelBuildingService と LexRuntimeService

SDK（boto3）に関しては、ここまで説明してきた AI サービスでは、それぞれの AI サービスに SDK 上のサービスが 1 つずつ提供されています。一方、Lexにおいては、チャットボットを開発するための LexModelBuildingService と、使用するための LexRuntimeService で、SDK 上のサービスが 2 つ提供されています。

Lex でのチャットボットの開発は、AWS の画面を用いて簡単に行うことができます。そのため、LexModelBuildingService を使ってプログラムでチャットボットを作ることはほとんどない、と思うかもしれません。しかし、ユーザーが投げかけた文章をきちんと理解するためには、サンプルとなる文章（サンプル発話）を設定することが必要です。チャットボットの実際のユーザー

は、開発者が想定していたものとは違う問い合わせを行う（例えばレストランを予約したい場合、「予約したい」という人もいれば「押さえたい」とか「行っても良いですか」という人もいる）ことが多々あり、それに対応するためにはサンプル発話を定期的にアップデートする必要があるでしょう。その際、常に AWS の画面を使うよりも、プログラムを使って自動化したいというニーズが存在します。そうした場合に、LexModelBuildingService が効果を發揮するでしょう。

3.9.5 Lex における会話の組み立て

Lex におけるチャットボットは、図 3-9-1 のような構成で会話を組み立てます。

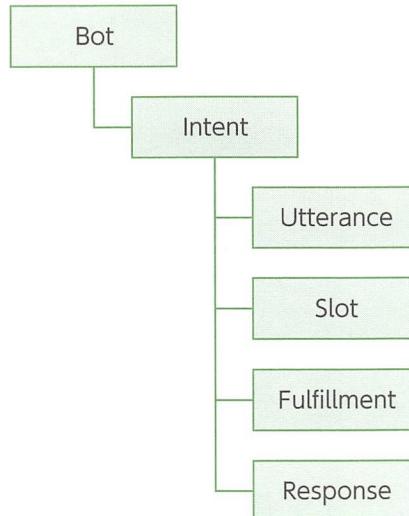


図 3-9-1 Lex におけるチャットボットの構成

Bot はチャットボットそのものであり、1つの AWS アカウントで複数の Bot を作成することができます。Bot は、複数の Intent で構成されます。Intent は、ユーザーが会話したい目的です。例えば、あるレストランがチャットボットを作る場合、まず Bot を1つ作成します。次に、その Bot の下に「レストランの予約」、「店舗場所の説明」、「出前の注文」という Intent を作ることになるでしょう。

Intent は Utterance、Slot、Fulfillment、Response といった項目で定義します。Utterance は発話という意味で、ユーザーがチャットボットに問い合わせた言葉（発話）です。レストランの予約という Intent を作るなら、「予約したい」、「席を取りたい」といった発話を登録することにな

るでしょう^{*24}。Slotは、予約を取る際に聞いておかなければならぬ事項です。例えば、名前、人数、来店日時という項目は必要でしょう。Lexでは、このような項目をSlotに登録しておき、それを順に聞いていくことができます。Slotに定義した項目をすべて聞き終わると、Fulfillmentに定義した動作（例えばLambdaの呼び出し）が行われます。

3.9.6 BookTrip サンプルで動作を確認する

それでは、Lexの画面と動作を実際に確認していきましょう。LexではBookTrip、OrderFlowers、ScheduleAppointmentという3つのサンプルが用意されていますが、ここではBookTrip（旅行予約）サンプルを使用します。

AWSコンソールでLexの画面を開きます。本書執筆時点において、Lexは東京リージョンでは提供されていないので、米国東部（バージニア北部）リージョンを指定します。

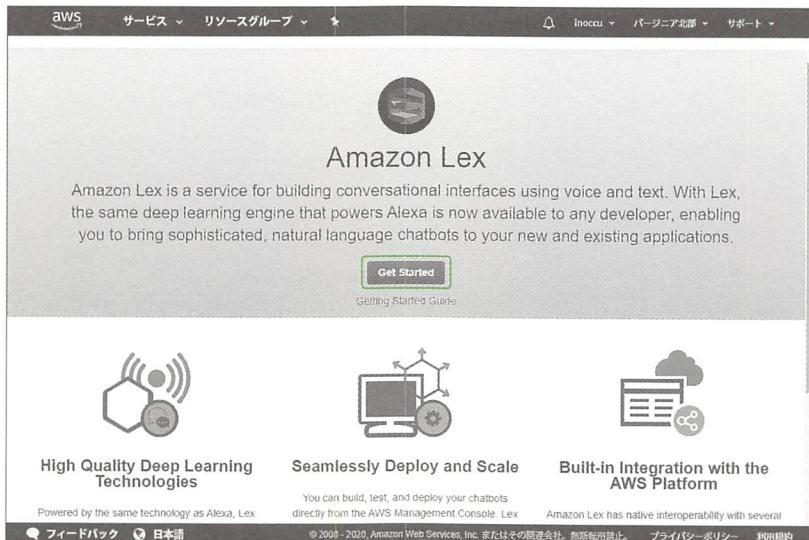


図 3-9-2 Lex のトップ画面

Lexのトップ画面で「Get Started」ボタンをクリックして、Botの作成画面を開きます。この画面でBookTripサンプルを選択すると、Botの名前(Bot name)が自動的に設定されるでしょう。

^{*24} Lexは日本語に対応していないので、Utteranceは英語で登録する必要があります。

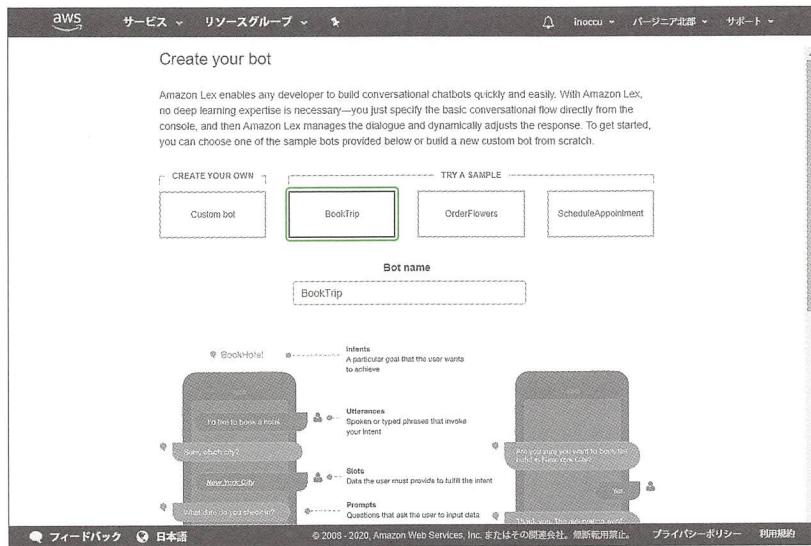


図 3-9-3 Bot の作成

画面下部に、IAM Role と COPPA の設定項目があります。IAM Role は、Lexにおいて、Bot が Polly による音声発話を使うための権限設定に関するもので、特に操作する必要はありません。COPPA は、米国の児童オンラインプライバシー保護法 (Children's Online Privacy Protection Rules) の対象となる Bot か否かを指定するものです。商用サイトが 12 歳以下の児童の個人情報を収集する場合に対象となるので、ここでは No を選択すれば良いでしょう。最後に、「Create」ボタンをクリックすると、Bot が作成されます。

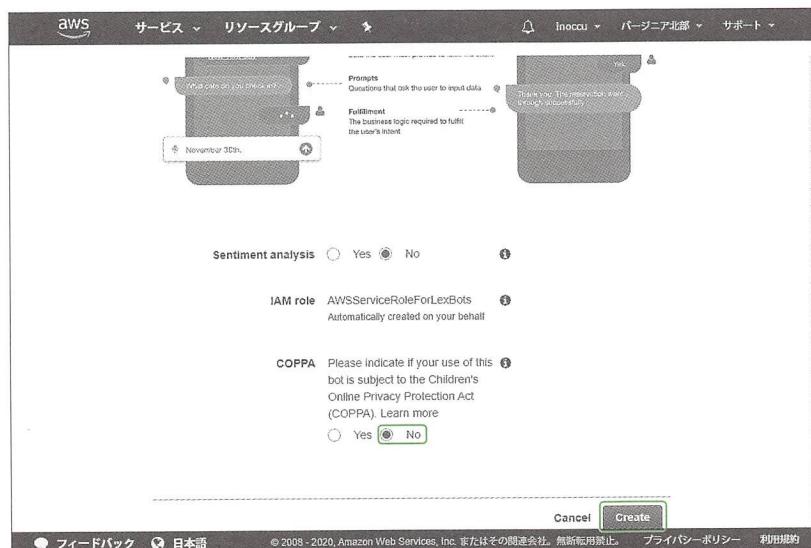


図 3-9-4 Bot の作成 (画面下部)

Bot の作成が完了すると、Editor 画面が表示され、自動的にビルドが行われます。ビルドが完了すると、画面右の Test bot という部分でチャットを試すことができます。

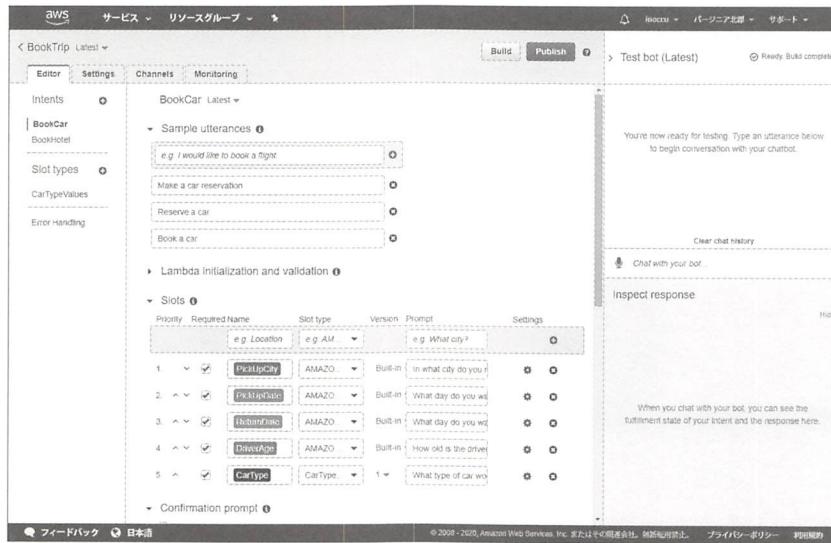


図 3-9-5 Lex の Editor 画面

この Editor 画面について見てきましょう。まず、画面左上に Intents の表示があります。1つのBotに複数のIntentを作成できますが、BookTripというBotでは、BookCar(レンタカー予約)とBookHotel(ホテル予約)の2つのIntentが存在しています。

次に画面中央を見ると、現在選択されているBookCarというIntentを起動するためのSample utterances(サンプル発話)が3つ設定されています。“Make a car reservation”、“Reserve a car”、“Book a car”という問い合わせがユーザーから行われると、このBookCarのIntentが起動するのです。実際に、このIntentの動作を確認してみましょう。

Test bot の Chat with your bot…という部分に、“book a car”と入力してみてください。

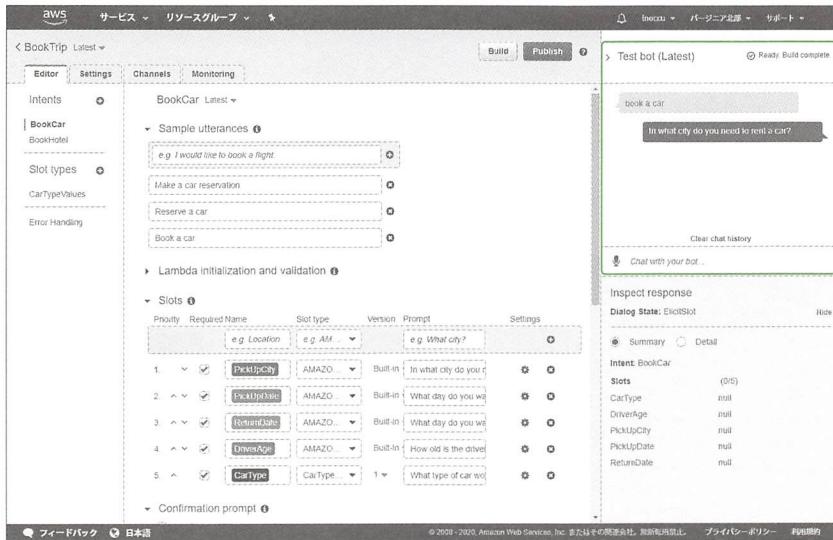


図 3-9-6 Bot との会話

Bot から “In what city do you need to rent a car?” という返事がきました。どの都市でレンタカーを借りたいのかを聞いています。これについて少し説明します。まず、“book a car” というユーザーからの問い合わせに対し、BookCar という Intent が認識されています。次に、その Intent で定義されている Slots（画面中央）の中から、1 番目の PickUpCity という Slot にもとづく質問が行われたのです。たしかに、PickUpCity という Slot の右に、“In what city…” という Bot からの返事が設定されています。

しばらく会話を続けてみましょう。

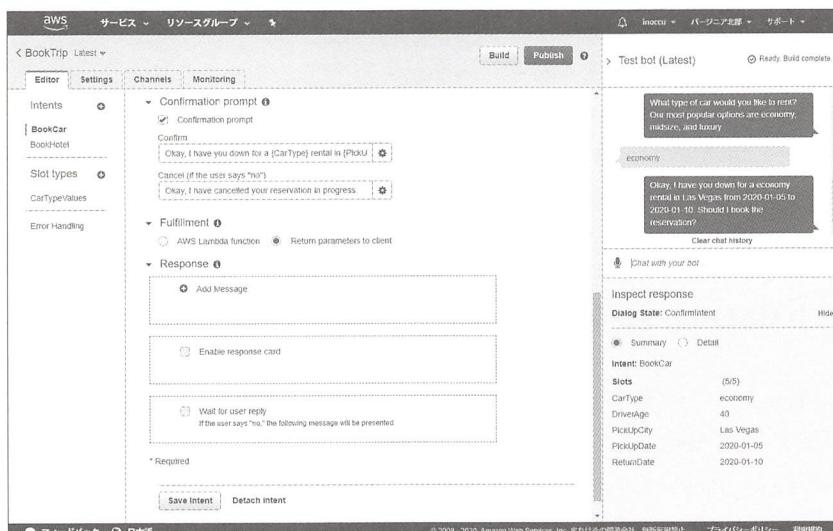


図 3-9-7 Bot による復唱（確認）

レンタカーを借りたい都市、借入日、返却日、運転手の年齢、車種の質問が順次行われます。これはSlotsとして定義したとおりです。すべての質問に答え終わると、Botからの質問に対するユーザーが返答した内容が、Botで認識した形で復唱されます。この復唱は、IntentのConfirmation promptの設定を有効にしているために行われた予約内容の確認です。“ok”と返事してみましょう。

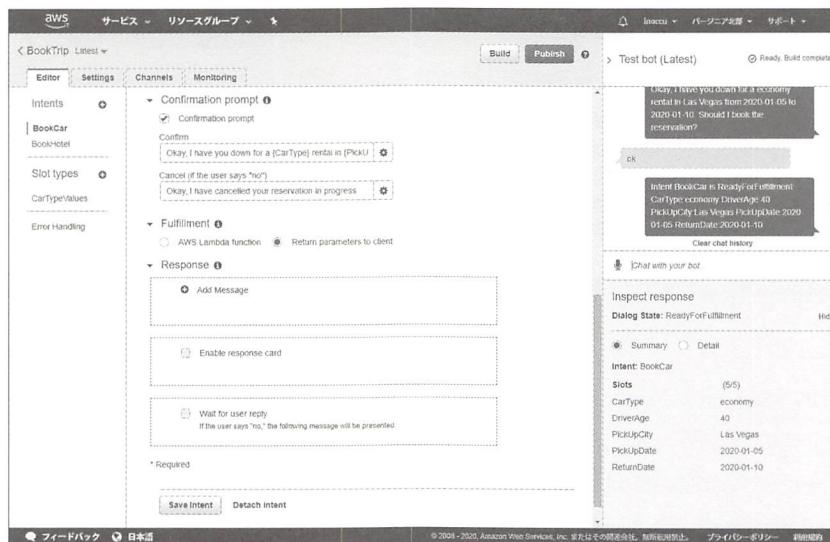


図 3-9-8 Botとの会話の終了

Botからは、“Intent BookCar is ReadyForFulfillment…”という無味乾燥な返事がきました。これは、Responseの項目でメッセージがセットされていないからです。後で、もう少し人間味のあるメッセージをセットすることにしましょう。

Botとの会話はこれで終わりましたが、実際にこのチャットボットを使う場合は、ユーザーから聞き出したレンタカーの予約情報を外部の予約管理システムなどに連携しなければなりません。IntentのFulfillmentの設定として連携を定義します。Fulfillmentとしては、AWSのLambda関数を使用することができます。

3.9.7 Response の追加

それでは、Botからの最後の返答であるResponseを定義してみましょう。

Response欄でAdd Messageをクリックし、メッセージを編集可能な状態にします。入力欄に“We are waiting in |PickUpCity| on |PickUpDate|.”と入力して、右にあるプラス(+)のアイコン

ンをクリックすると、メッセージを追加できます。Slot の名前を {PickUpCity} のようにメッセージに含めると、Bot が認識した Slot の入力内容がセットされます。

画面下にある「Save Intent」ボタンをクリックして、追加したメッセージを保存した後、画面上部の「Build」ボタンをクリックしてビルトの完了を待つと、Test bot で試すことができるようになります。

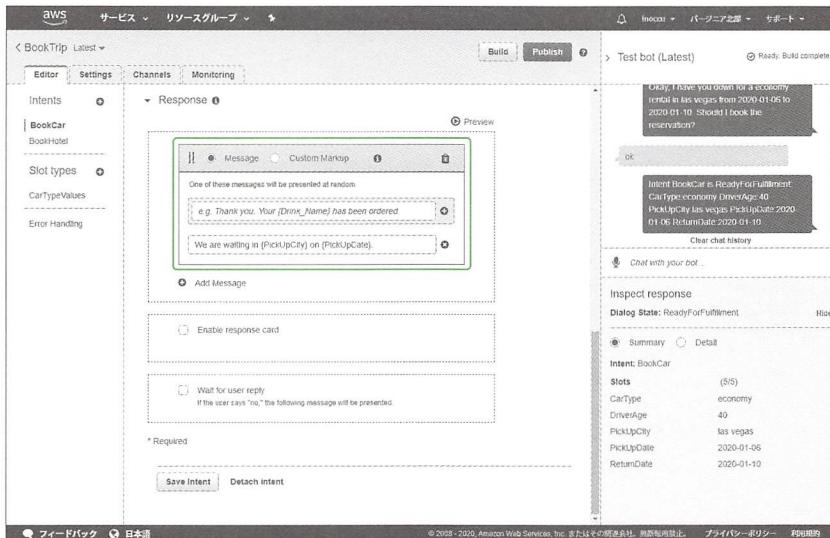


図 3-9-9 Response の設定

再び最初から会話をを行うと、図 3-9-10 のように、定義したメッセージによる返答が行われました。

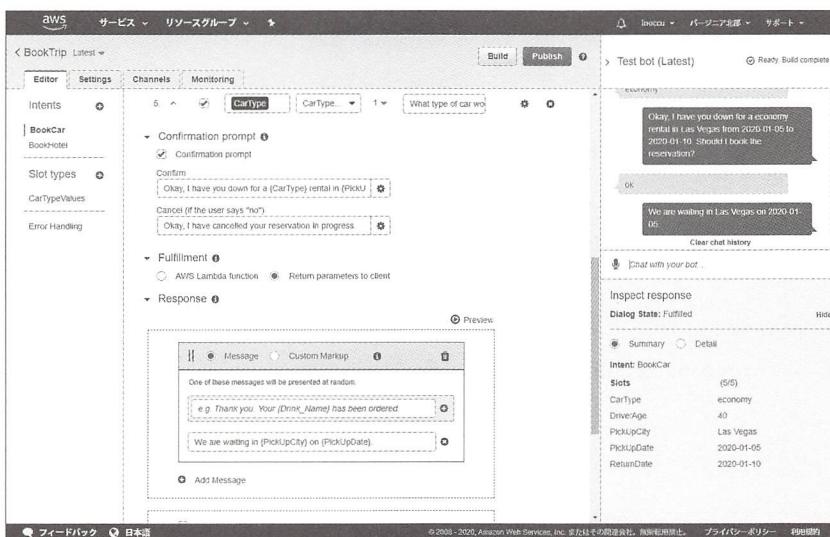


図 3-9-10 定義したメッセージによる返答

3.9.8 チャットボットの公開

Lex の Editor で作成したチャットボットは、Test bot として動作を確認することができます。ただし、公開されているわけではないので、一般ユーザーからの会話を受け付けることはできません。チャットボットを公開するには、2つの方法があります。1つ目は、Editor と同様の AWS の画面上で、Channels の設定を行う方法です。Facebook Messenger、Kik^{*25}、Slack といったメッセンジャーサービスや、Twilio 社の API を活用したショートメッセージングサービス (SMS) を使ってチャットボットを公開することを、Channels 画面で設定することができます。

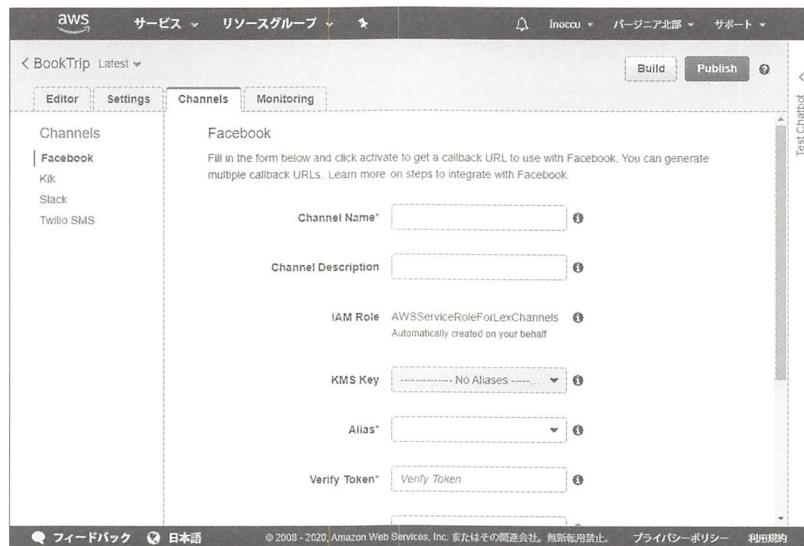


図 3-9-11 Channels 画面

もう1つの方法は、前述の LexRuntimeService を SDK から使用することです。例えば作成したチャットボットを LINE で公開したい場合は、Channels では対応していないため、LINE の API と LexRuntimeService の両方を取り持つサーバープログラムを開発する必要があるでしょう。ここでは、まず、LexRuntimeService の動作を Jupyter Notebook の画面上で確認してみることにしましょう。

LexRuntimeService を使用するためには、これまでと同様に権限を付与する必要があります。IAM の画面で「AmazonLexFullAccess」というポリシーをアタッチしておきましょう。

*25 Kik は、米国でよく利用されているメッセンジャーサービスです。



図 3-9-12 Lex へのアクセス権限の付与

また、作成したチャットボットを公開するには、Alias の設定をして Publish しておく必要があります。Alias は Lex における Bot のバージョン管理の単位で、バージョンごとに適当な名前を付けることができます。

まず、Bot の Settings 画面を開き、Aliases タブで Alias を設定します。ここでは、Dev という名前の Alias を作成し、その Bot version として Latest (最新) を設定しました。その後で、「Publish」ボタンをクリックします。



図 3-9-13 Alias の設定と Publish

どの Alias を Publish するかを指定する画面が表示されるので、いま作成した Dev という Alias を選択して、「Publish」ボタンをクリックします。

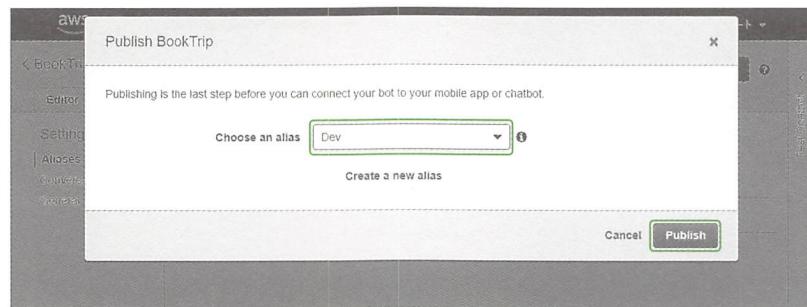


図 3-9-14 Publish する Alias の指定

次に、Jupyter Notebook で新しい Notebook を開き、以下のコードを実行します。

```
import boto3
import json

client = boto3.client('lex-runtime', region_name='us-east-1')

text = 'book a car'

response = client.post_text(
    botName='BookTrip',
    botAlias='Dev',
    userId='user01',
    inputText=text
)

print(response)
```

Python SDK では、post_text() 関数を使ってユーザーの発言を Lex に引き渡します。引数は、botName に Bot の名前、botAlias に先ほど作成した Alias の名前、userId にユーザーを識別する任意の値（後述）、inputText にユーザーの発言をセットします。実行結果は図 3-9-15 のとおりです。

```
In [1]: M import boto3
import json

client = boto3.client('lex-runtime', region_name='us-east-1')

text = 'book a car'

response = client.post_text(
    botName='BookTrip',
    botAlias='Dev',
    userId='user01',
    inputText=text
)

print(response)

[{"ResponseMetadata": {"RequestId": "24ba8257-1f6d-4652-ba89-8d7b47e7cbe3", "HTTPStatusCode": 200, "HTTPHeaders": {"Content-Type": "application/json", "Date": "Sat, 04 Jan 2020 07:34:58 GMT", "x-amzn-requestid": "24ba8257-1f6d-4652-ba89-8d7b47e7cbe3", "Content-Length": "373", "Connection": "keep-alive", "RetryAttempts": 0}, "IntentName": "BookCar", "Slots": {"CarType": None, "DriverAge": None, "PickUpCity": None, "PickUpDate": None, "ReturnDate": None}, "Message": "In what city do you need to rent a car?", "MessageFormat": "PlainText", "DialogState": "ElicitSlot", "SlotToElicit": "PickUpCity", "SessionId": "2020-01-04T07:34:59.163Z-0MmzouNJ"}]
```

図 3-9-15 post_text の実行結果

Lex からのレスポンスにある message の値に、"In what city do you need to rent a car?" とセットされています。AWS のコンソールで Test bot を行ったときと同様のメッセージが返っています。また、使用される Intent である "BookCar" が intentName にセットされていますし、Slots の状況などもわかります。

では、会話を続けてみましょう。Test bot の画面ではそのまま次の発言を入力すれば良いのですが、API を使用する場合は考慮すべき点が 1 つあります。それは、今回作成したチャットボット (BookTrip という Bot の、Dev という Alias で公開されているもの) が同時に複数のユーザーと会話しなければならない場合、1 回の API 呼び出しでは、会話のキャッチボールが 1 回行われただけで接続が切れてしまうので、2 回目以降の会話のキャッチボールでは、それが誰と会話しているかがわからないのではないか、という点です。つまり、チャットボットは同時に何人のユーザーと会話をするので、1 人 1 人のユーザーとどのような会話をしているか覚えておかないといけないし、誰から話しかけられたか識別する必要もあるのです。

そのために使用するのが userId の値です。最初に Bot に話しかけるときに任意で決めて、post_text() 関数の引数としてセットした userId の値を、2 回目以降の post_text() 関数の呼び出しても同じくセットすれば、会話が続いていることを示すことになります。

では、続けて API を操作してみましょう。Notebook の次のセルで、以下のコードを実行します。

```
text = 'New York'

response = client.post_text(
    botName='BookTrip',
    botAlias='Dev',
    userId='user01',
```

```
    inputText=text
)
print(response)
```

どの都市でレンタカーを借りるか質問されていたので、“New York”という返事を行いました。実行結果は図 3-9-16 のとおりです。

```
In [2]: text = 'New York'

response = client.post_text(
    botName='BookTrip',
    botAlias='Dev',
    userId='user01',
    inputText=text
)
print(response)
```

```
[{'ResponseMetadata': {'RequestId': '9da01235-45af-42c7-89d8-1d2008da5a8c', 'HTTPStatusCode': 200, 'HTTPHeaders': {'Content-Type': 'application/json', 'Date': 'Sat, 04 Jan 2020 07:35:44 GMT', 'X-Amzn-Requestid': '9da01235-45af-42c7-89d8-1d2008da5a8c', 'Content-Length': '382', 'Connection': 'keep-alive'}, 'RetryAttempts': 0}, 'intentName': 'BookCar', 'slots': {'CarType': None, 'DriverAge': None, 'PickUpCity': 'New York', 'PickUpDate': None, 'ReturnDate': None}, 'message': "What day do you want to start your rental?", 'messageFormat': 'PlainText', 'dialogState': 'ElicitSlot', 'slotToElicit': 'PickUpDate', 'sessionId': '2020-01-04T07:34:59.163Z-DMnzouNj'}
```

図 3-9-16 post_text の実行結果 (2 回目)

Lex からのレスポンスを見ると、message が “What day do you want to start your rental?” となり、質問が借入日に変わっています。どうやら会話が継続しているようです。また、slots の値を見ると、PickUpCity として “New York” がセットされています。

このように、Lex を使うとチャットボットを簡単に作成でき、また、公開することもできます。Lex の機能はまだたくさんあり、本書ではそのさわり程度しか紹介できませんでしたが、何よりも残念なのは日本語版がまだ提供されていないことです。日本のユーザーとしては、まずは Lex がどのようなものかを把握し、日本語版が提供されたときにすぐに使えるように準備をしておくと良いでしょう。

3.10 Amazon Forecast

3.10.1 Amazon Forecast とは

Amazon Forecast^{*26} は、時系列データを用いた予測モデルを構築し、使用するためのサービスです。このサービスは、例えば来店客数や売上、コールセンターに電話がかかってくる件数、消耗品の交換頻度など、さまざまな分野での予測に活用できます。

一般的に、時系列データを用いた予測モデルの構築では、さまざまなアルゴリズムの中から最適なものを選択する必要があります。Amazon Forecast では使用するアルゴリズムを自ら選択することもできますが、それぞれのアルゴリズムについての知識がなければ、適したものを見つけることは難しいでしょう。Amazon Forecast は、最適なアルゴリズムを自動的に選択してくれる AutoML オプションも提供しているため、機械学習の経験が少ない人でも時系列予測モデルを活用することができます。

また、こうした予測モデルは、第 4 章で解説する Amazon SageMaker を使って作成することも可能ですが、Amazon Forecast には、時系列データに特化した予測モデルの作成だけではなく、活用（他のシステムからの呼び出し）に関する機能も備わっています。したがって Amazon Forecast は、上記の用途などで活用するモデルを作成する際にまず最初に検討してみたいサービスといえるでしょう。

3.10.2 Amazon Forecast で使用するデータセット

Amazon Forecast で予測モデルを作成するには、データセットグループを作成する必要があります。データセットグループとは、あらかじめ定義されたデータセットタイプの形式にしたトレーニング用データをまとめたものです。

データセットグループには、少なくとも TARGET_TIME_SERIES データセットタイプのデータセットが必要です。TARGET_TIME_SERIES は、時系列データであり、目的変数（モデルで予測したい項目。例えば売上金額）と説明変数（予測に活用できそうな項目。例えば来客数や曜日）を含みます。

*26 <https://aws.amazon.com/jp/forecast/>

その他のデータセットはオプションであり、TARGET_TIME_SERIES に含めない説明変数を提供するためのものです。RELATED_TIME_SERIES データセットタイプは時系列データ、ITEM_METADATA データセットタイプは時系列データ以外のデータセットです。

それぞれのデータセットタイプでどのようなデータを準備するかは、予測モデルを活用する領域によって異なります。以下のとおり、領域ごとにデータセットドメインが定義されています。活用したい領域に適合するデータセットドメインがある場合はそれを使用し、適合するものがなければCUSTOM ドメインを使用すると良いでしょう。

- RETAIL ドメイン (小売の需要予測)
- INVENTORY_PLANNING ドメイン (サプライチェーンと在庫の計画)
- EC2 CAPACITY ドメイン (Amazon EC2 のキャパシティの予測)
- WORK_FORCE ドメイン (従業員の計画)
- WEB_TRAFFIC ドメイン (Web トラフィックの見積り)
- METRICS ドメイン (収益、キャッシュフローの予測)
- CUSTOM ドメイン (その他の時系列予測)

例えば RETAIL ドメインでは、それぞれのデータセットタイプに以下のような項目をセットします。

- TARGET_TIME_SERIES データセットタイプ
一つ、どの商品が、いくつ売れたかを示すデータ（「いくつ売れたか」が目的変数）
- RELATED_TIME_SERIES データセットタイプ
商品の価格や在庫数、Web ページのヒット数などの時系列データ
- ITEM_METADATA データセットタイプ
商品に関する属性情報（時系列データではない）

データセットタイプはデータの項目定義書にあたるもので、データセットドメインごとに異なる項目が定義されています。上記の RETAIL ドメインの例を見るとわかるように、売上予測に必要になりそうなデータ項目が定義されています。データ項目には必須のものとオプションのものがあります。一般的には、オプションの項目にも値をセットした方が予測精度の高いモデルができるでしょう。

3.10.3 Amazon Forecast へのデータのインポート (データセットグループの作成)

Amazon Forecast を使ってモデルを構築し、実際に予測を実施してみましょう。Amazon Forecast では、構築した予測モデルを「予測子」と呼びます。予測子の作成と使用（実際の予測）は、AWS コンソールを操作するか、あるいは API を用いて行います。本書では、AWS コンソールを操作して行ってみます。

使用するデータは、UCI（カリフォルニア大学アーバイン校）の Machine Learning Repository で公開されている Online Retail II Data Set です。詳細の確認とデータのダウンロードは、<http://archive.ics.uci.edu/ml/datasets/Online+Retail+II> から行うことができます。このデータは、英国を拠点とするオンライン小売店において 2009 年 12 月 1 日から 2011 年 9 月 12 日までに発生したすべての売上を含んでいます。データは Excel 形式 (.xlsx) で配布されており、上記 URL の画面から Data Folder というリンクをクリックすると、online_retail_II.xlsx というファイルをダウンロードすることができます。

The screenshot shows the UCI Machine Learning Repository homepage with the title "Machine Learning Repository" and a sub-section for "Online Retail II Data Set". Below the title, there are download links for "Data Folder" and "Data Set Description". A summary table provides details about the dataset, including its characteristics (Multivariate, Sequential, Time-Series, Text), number of instances (1067371), area (Business), and date donated (2019-09-21). The table also lists attribute characteristics (Integer, Real), number of attributes (8), and missing values (Yes). The "Source" section credits Dr. Daqing Chen from London South Bank University. The "Data Set Information" section notes that the dataset contains transactions from a UK-based company selling unique giftware between 2009 and 2011. The "Attribute Information" section provides detailed descriptions for each column: InvoiceNo (Invoice number), StockCode (Product code), Description (Product name), Quantity (Quantity per transaction), InvoiceDate (Invoice date and time), UnitPrice (Unit price in sterling), and CustomerID (Customer number).

図 3-10-1 Online Retail II Data Set

ダウンロードしたファイルを Excel で開くと、「Year 2009-2010」と「Year 2010-2011」の 2 つのシートがあり、それぞれ表 3-10-1 の項目がセットされています。

表 3-10-1 online_retail_II.xlsx にセットされている項目

項目名	説明	データ例
Invoice	請求 No	489434
StockCode	商品コード	85048
Description	商品名	15CM CHRISTMAS GLASS BALL 20 LIGHTS
Quantity	注文数	12
InvoiceDate	請求日時	2009/12/1 7:45
Price	価格	6.95
CustomerID	顧客 ID	13085
Country	国名	United Kingdom

今回は TARGET_TIME_SERIES データセットタイプのみをセットしてみます。TARGET_TIME_SERIES データセットタイプには、item_id、timestamp、demand の 3 つのデータ項目が必要です。ここでは、item_id として StockCode、timestamp として InvoiceDate、demand として Quantity の値を用いることにします。

まず、Excel 上で列を StockCode、Quantity、InvoiceDate の 3 列に絞り（今回は Year 2009-2010 のシートのみを使います）、Quantity と InvoiceDate の列を並べ替えます。次に、1 行目のタイトル行を TARGET_TIME_SERIES データセットタイプにそろえます。また、InvoiceDate 列（timestamp）の書式を「yyyy-mm-dd hh:mm:ss」にしておきます。

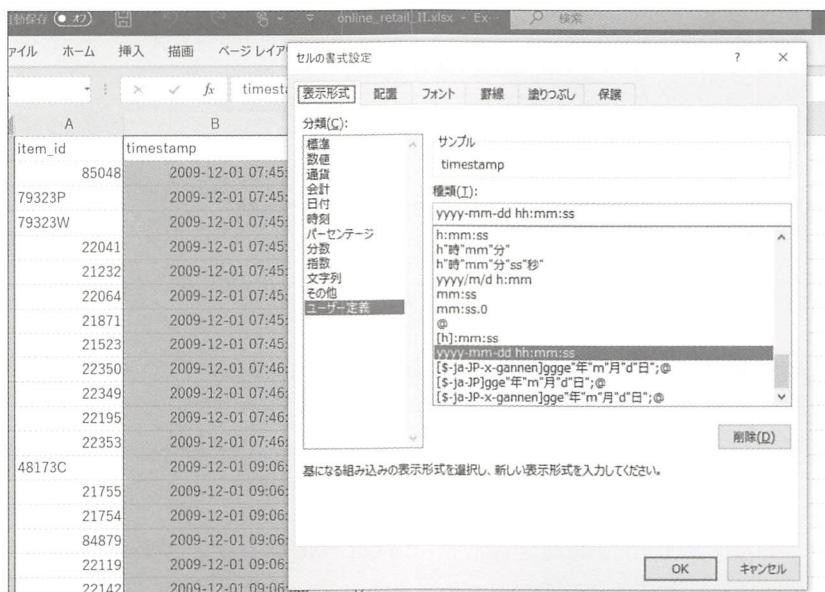


図 3-10-2 InvoiceDate 列 (timestamp) の書式を変更

図 3-10-3 のような形になったら、そのシートを CSV 形式のデータとして保存します。

A	B	C	D	E	F	G
1	item_id	timestamp	demand			
2	85048	2009-12-01 07:45:00	12			
3	79323P	2009-12-01 07:45:00	12			
4	79323W	2009-12-01 07:45:00	12			
5	22041	2009-12-01 07:45:00	48			
6	21232	2009-12-01 07:45:00	24			
7	22064	2009-12-01 07:45:00	24			
8	21871	2009-12-01 07:45:00	24			
9	21523	2009-12-01 07:45:00	10			
10	22350	2009-12-01 07:46:00	12			
11	22349	2009-12-01 07:46:00	12			

図 3-10-3 編集後の Excel データ

Amazon Forecast にトレーニング用のデータを引き渡すためには、あらかじめ S3 に CSV 形式のファイルをアップロードして、S3 上のパスとして指定する必要があります。そこで、Excel で保存した図 3-10-4 のような CSV ファイルを、S3 の適当なバケットにアップロードします。

```
1 item_id,timestamp,demand
2 85048,2009-12-01 07:45:00,12
3 79323P,2009-12-01 07:45:00,12
4 79323W,2009-12-01 07:45:00,12
5 22041,2009-12-01 07:45:00,48
6 21232,2009-12-01 07:45:00,24
7 22064,2009-12-01 07:45:00,24
8 21871,2009-12-01 07:45:00,24
9 21523,2009-12-01 07:45:00,10
10 22350,2009-12-01 07:46:00,12
11 22349,2009-12-01 07:46:00,12
12 22195,2009-12-01 07:46:00,24
13 22353,2009-12-01 07:46:00,12
14 48173C,2009-12-01 09:06:00,10
15 21755,2009-12-01 09:06:00,18
16 21754,2009-12-01 09:06:00,3
```

図 3-10-4 S3 にアップロードする CSV ファイル

ここまで準備ができたら、Amazon Forecast の画面を開き、「Create dataset group」ボタンをクリックして作業を始めます。



図 3-10-5 Amazon Forecast のトップ画面

まず、データセットグループの名前 (Dataset group name) と、ドメイン (Forecasting domain) を指定します。データセットグループの名前は任意のもので構いませんが、ドメインは Retail を指定します。これは、今回使用するデータがオンライン店舗の売上履歴であり、予測したい値（目的変数）が売上個数だからです。

これらを指定したら、「Next」ボタンをクリックします。

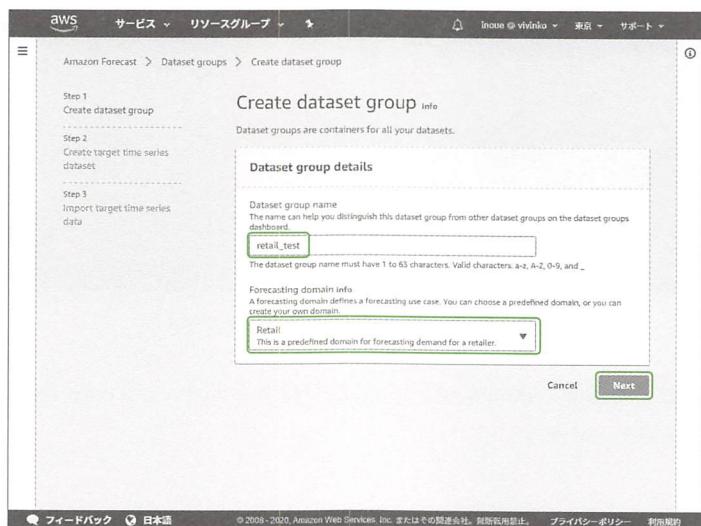


図 3-10-6 Create dataset group 画面

次に、データセットの詳細を設定します。まずは必須の TARGET_TIME_SERIES データセットタイプについて行い、それ以外のデータセットタイプを使用する場合は後の画面で行います。この画面では、データセットの名前(Dataset name)、データの周期(Frequency of your data)、データスキーマ (Data schema) を指定します。重要なのは、データの周期とデータスキーマです。

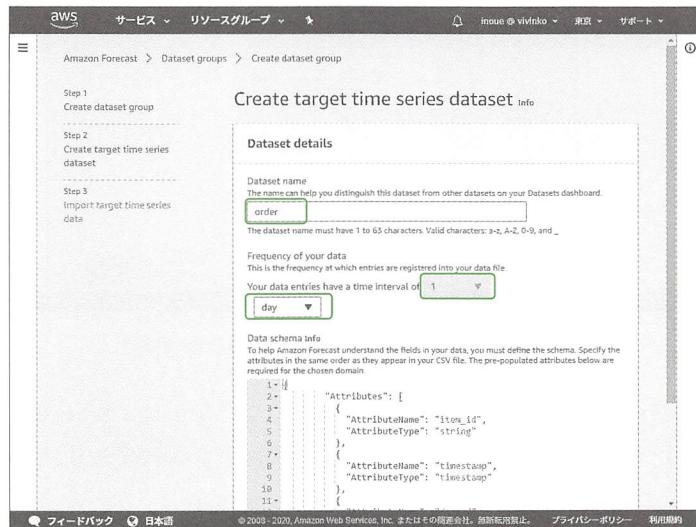


図 3-10-7 Create target time series dataset 画面①

データの周期では、日や時といった時間の単位と数値を指定します。例えば、1日単位でデータをまとめて予測したい場合、時間の単位は Day (日)、数値は 1 を指定します。ここで指定したデータの周期よりも大きい単位で予測を行うことはできますが、小さい単位ではできないので注意しましょう。つまり、1日単位でデータをまとめて予測子を作った場合、1か月単位での売上予測は行えますが、1時間単位での予測はできません。また、ここで指定したデータの周期と、実際に与えるデータの timestamp の単位が一致している必要はありません。例えば、データの周期として 1 日を指定し、実際のデータは今回使用するもののように売上の都度（1 日単位に集約されていない）でも構いません。指定されたデータの周期に合わせて、Amazon Forecast が自動的にデータを集約してくれます。

データスキーマはデフォルト値がセットされているので、そのままでも構いませんが、今回のデータは demand の値が整数値であるため、AttributeType を float から integer に変更しました。次に、「Next」ボタンをクリックします。

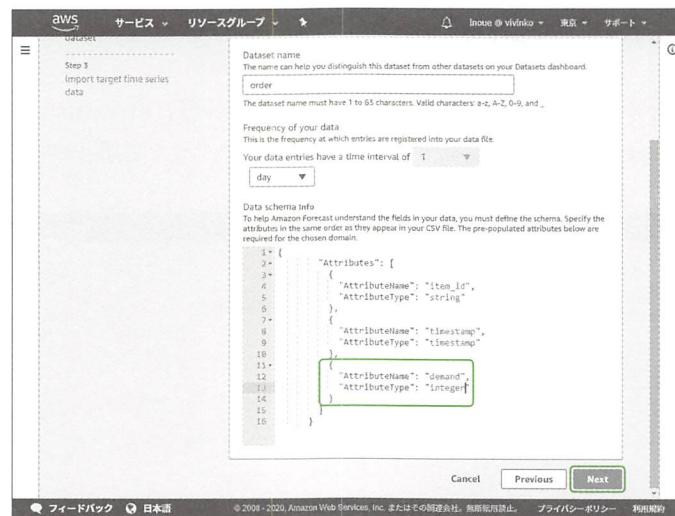


図 3-10-8 Create target time series dataset 画面②

続いて Dataset import details を指定します。ここでは、Amazon Forecast にインポートして使用するデータセットの詳細を指定します。インポートするデータセットの名前 (Dataset import name) は任意のものを指定します。Timestamp format は、データの timestamp の値の表記形式です。デフォルトでは、yyyy-MM-dd HH:mm:ss が指定されています。先ほど Excel 上で同じ形式に変換したので(日時形式の指定方法は Amazon Forecast と Excel とで異なりますが、得られる日時文字列は同じです)、そのまま構いません。

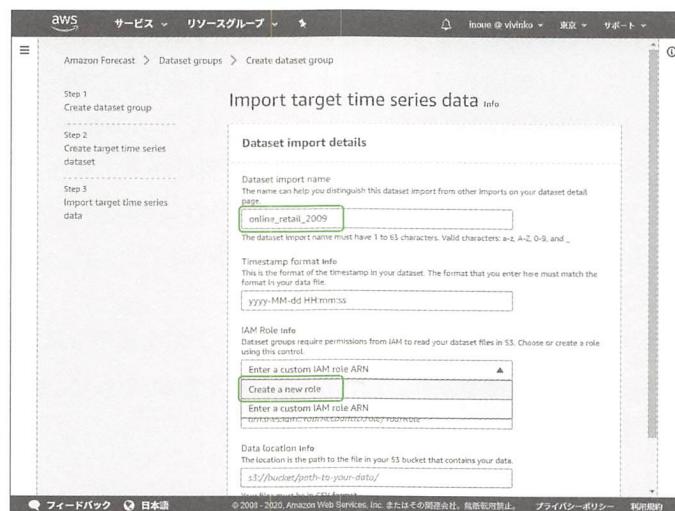


図 3-10-9 Import target time series dataset 画面①

Amazon Forecast から S3 にアクセスするためには、IAM Role (IAM ロール) が必要です。IAM Role とは、IAM で付与できるさまざまな権限の組み合わせを定義したものです。

IAM Role は、既存のものを指定するか、この画面で新規作成することができます。新規作成する場合は、IAM Role のプルダウンメニューで「Create a new role」を選択します。図 3-10-10 の画面が表示されるので、特定の S3 バケットのみにアクセスできるようにするか、同じ AWS アカウント配下にあるすべての S3 バケットにアクセスできるようにするかを指定します。ここでは、先ほどアップロードした CSV ファイルが格納されている S3 バケットが参照可能になるようにしてください。

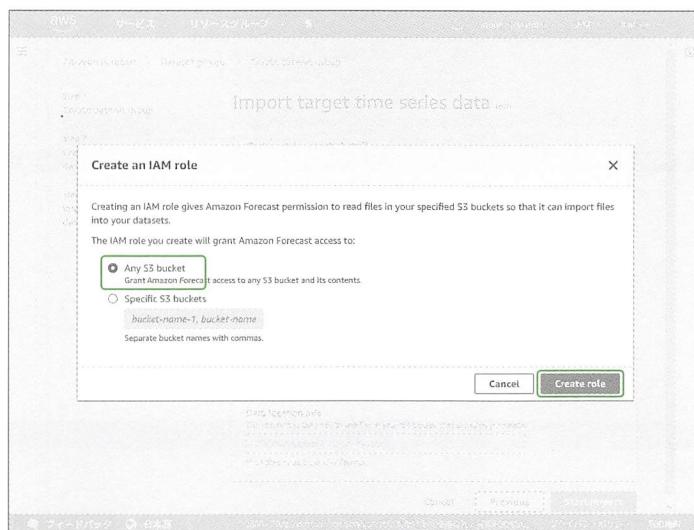


図 3-10-10 Import target time series dataset 画面②

最後に、CSV ファイルの S3 上のパス(s3:// <バケット名> / <オブジェクト名>)を指定して、「Start import」ボタンをクリックします。

ここまで操作を行うと、トレーニング用のデータが Amazon Forecast にインポートされます。少し時間がかかるので、Amazon Forecast のダッシュボードなどで状況を確認しましょう。Active と表示されれば、インポートは完了です。

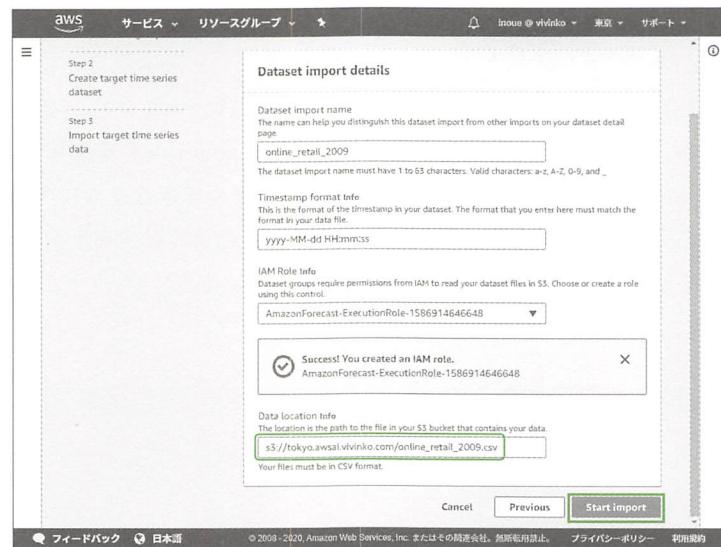


図 3-10-11 Import target time series dataset 画面③

本項で扱ったTARGET_TIME_SERIES以外のデータセットタイプのデータを、データセットグループにインポートする場合は、Amazon Forecast の Datasets 画面を開き、同様の操作を行います。

Amazon Forecast		Datasets (3) info					
		View details Delete Upload dataset					
		Dataset name	Dataset type	Status	Latest import status	Domain	Created
▼	retail_test	order	TARGET_TIME_SERIES	Active	Active	RETAIL	Wed, 15 Apr 2020 01:36:31 GMT
		-	ITEM_METADATA	Not created	Not uploaded	-	-
		-	RELATED_TIME_SERIES	Not created	Not uploaded	-	-

図 3-10-12 Datasets 画面

3.10.4 予測子 (Predictor) の作成

データセットグループの作成が完了したら、次は予測子 (Predictor) を作成します。Amazon Forecast の予測子は、一般的にはトレーニング済みモデルにあたるもので、先ほどインポートしたデータ（作成したデータセットグループ）を使って、予測アルゴリズムは、自ら選択するか、自動的に選択されたものを用いてトレーニングを行います。

Amazon Forecast のダッシュボードで、Train a predictor の部分にある「Start」ボタンをクリックします（「Start」ボタンは、データセットグループの作成が完了している場合に表示されます）。

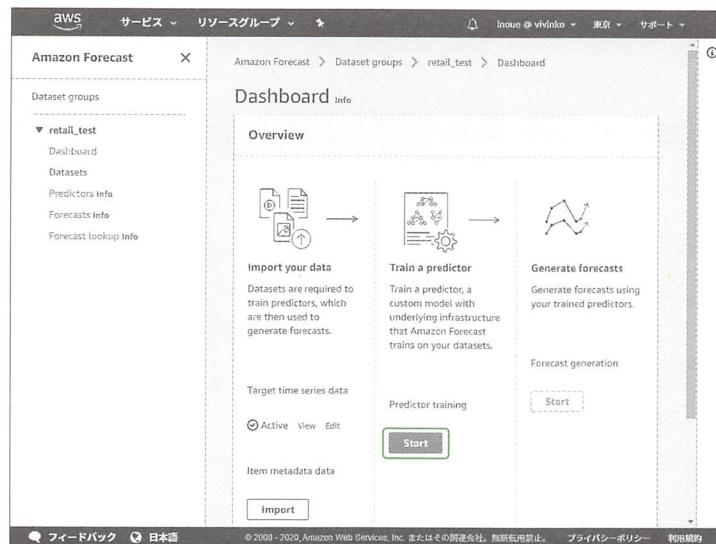


図 3-10-13 予測子の作成

Predictor details の画面が表示されるので、予測子の名前 (Predictor name)、予測の期間 (Forecast horizon)、予測の周期 (Forecast frequency)、予測アルゴリズム (Algorithm selection) などを指定します。ここで重要なのは、予測の期間、周期、予測アルゴリズムです。

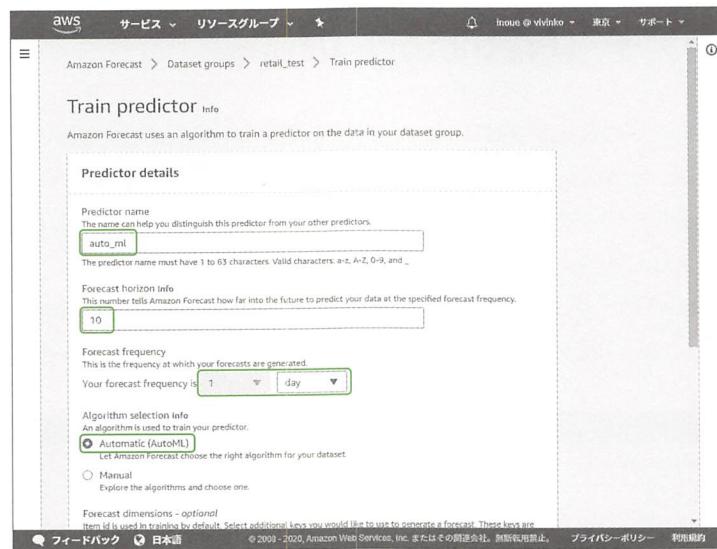


図 3-10-14 Train predictor 画面①

まず初めに、予測の周期について説明します。予測の周期は、データセットグループの作成時に指定したデータの周期と同様のもので、どの程度の時間単位で予測を行いたいかを指定します。先ほどデータセットグループの作成を行った際、データの周期を1日にしたので、予測の周期は1日以上にする必要があります。ここでは、明日の売上、明後日の売上というように1日単位の予測をしたいので、予測の周期を1日にします。

そして、予測の期間は、ここで指定した予測の周期の何個分を予測したいかを指定します。予測の周期が1日、予測の期間が10の場合、10日分の予測を1日単位で行うことになります。最大500までの値を指定できるので、予測の周期が1日なら1年半くらいの長期の予測も可能です。しかし、一般的には、トレーニングデータを入れ替えて予測子を作り替えた方が予測精度は良好になる可能性が高いので、そこまで大きな値を指定しなくとも良いでしょう。

次に、予測アルゴリズムを選択します。選択肢は、Automatic (AutoML) と Manual の2つです。Automatic (AutoML) は、データの内容に応じて Amazon Forecast が自動的に予測アルゴリズムを選択するもので、通常はこちらを選択しておけば問題ないでしょう。Manual を選択した場合は、実際に使用したい予測アルゴリズムを以下の候補から選びます。

- 自己回帰和分移動平均 (ARIMA)
- DeepAR+
- 指数平滑法 (ETS)
- ノンパラメトリック時系列 (NPTS)
- Prophet

1つのデータセットグループから複数の予測子を作成することもできます。この場合、Manualを指定して、複数の予測アルゴリズムを用いて予測子を作成します。複数の予測子を作成して予測精度を比較してみるのも良いかもしれません。

ここまで図 3-10-14 の画面の必須入力項目について説明しましたが、このほか、オプションの入力項目として以下のようないわゆるがあります。

- Forecast dimensions

必須項目である item_id 以外の項目を予測対象にしたい場合に指定します。

- Country for holidays

祝祭日は、売上などの値が通常よりも増減することが考えられます。祝祭日は国によって異なるため、この項目で、基準としたい国を指定します。

- Number of backtest windows

予測子の精度評価は、入力データの末尾から「Backtest window offset」で指定した分のデータを分割して行いますが、その分割回数を 1～5 で指定します。デフォルト値は 1 です。この場合、データを 1 回だけ分割して、その分のデータのみ精度評価が行われることになります。

- Backtest window offset

予測子の精度評価を行う際に、1 回の分割で取得する期間を指定します。デフォルト値は、予測の期間 (Forecast horizon) の値です。Forecast horizon が 10、Forecast frequency が日の場合、デフォルトでは入力データの末尾から 10 日分のデータを分割し、精度評価を行います。

最後に、「Train predictor」ボタンをクリックすると、予測子の作成（モデルのトレーニング）が行われます。指定した予測アルゴリズムやデータセットのサイズにもよりますが、長時間かかる可能性もあります。ダッシュボードや Predictors 画面にステータスが表示されるので、Active になるまで待ちましょう。

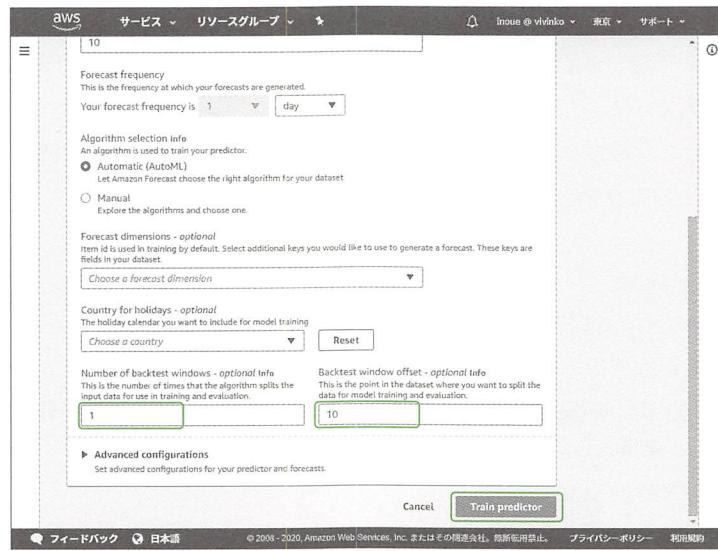


図 3-10-15 Train predictor 画面②

3.10.5 予測 (Forecast) の作成と結果の確認

Amazon Forecast のダッシュボードまたは Predictors の画面で、予測子のトレーニングステータス (Predictor training status) が Active になれば、トレーニングが終了して予測子が完成したことを意味します。次に、予測子を使って実際に予測を行ってみましょう。

予測を行うには、予測子 (Predictor) から予測 (Forecast) を作成します。Amazon Forecast のダッシュボードの右端に Generate forecasts と表示されています。その下にある「Start」ボタンをクリックすると、図 3-10-17 に示す Create a forecast 画面が開きます。なお、先ほどの Train a predictor と同様に、予測子のトレーニングステータスが Active になっていないと「Start」ボタンは表示されません。また、図 3-10-16 の画面にあるメニューから Predictors 画面を開き、「Create forecast」ボタンをクリックしても構いません。

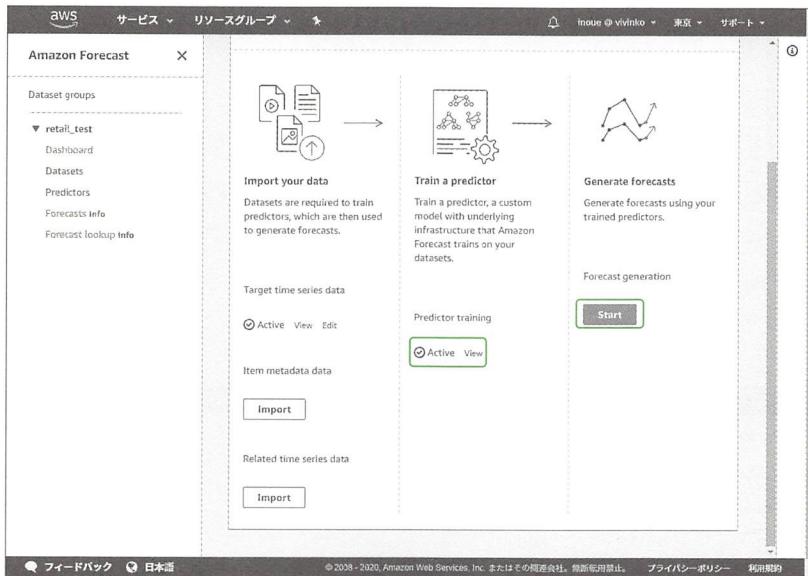


図 3-10-16 予測の作成

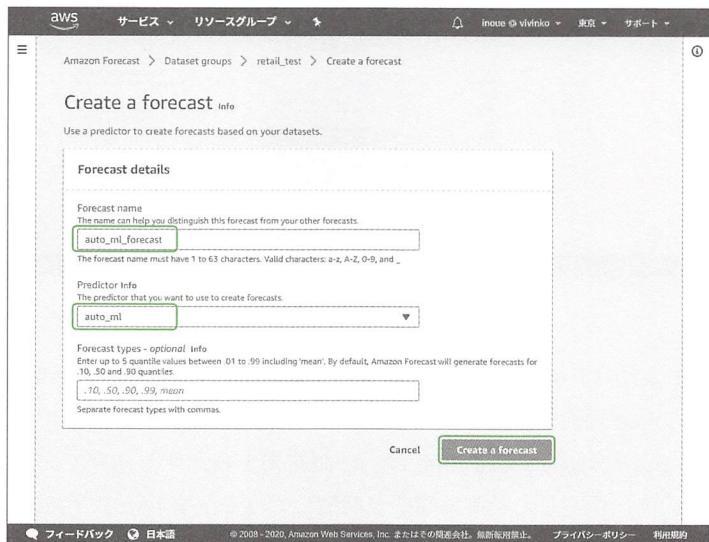


図 3-10-17 Create a forecast 画面

Create a forecast 画面では、予測の名前 (Forecast name) を任意で指定します。また、使用する予測子 (Predictor) として、先ほど作成した予測子を選択します。この画面では、ここまででの入力が必須です。任意の入力項目である Forecast types では、分位点回帰の指定を行います。今回の例のように売上個数の予測を行う場合、例えば「明日、A という商品が 20 個売れる」といっ

た予測ができますが、予測結果は上振れまたは下振れする可能性があるので、「15個～25個くらい売れる」というように範囲(分布)で回答した方が予測の的中率は高くなります。一般的には、分布の中央値が予測結果として用いられており、Forecast typesでは、0.5もしくはmeanを指定することができます。Amazon Forecast のデフォルトでは、0.5以外に、0.1と0.9が指定されており、分布の両裾（下振れした値から上振れした値まで）を求めることができます。

最後に「Create a forecast」ボタンをクリックし、予測を作成します。作成が完了すると、Amazon Forecast のダッシュボードに「Lookup forecast」ボタンが表示されます。また、Forecasts 画面では、作成した名前の予測のステータスが Active になります。

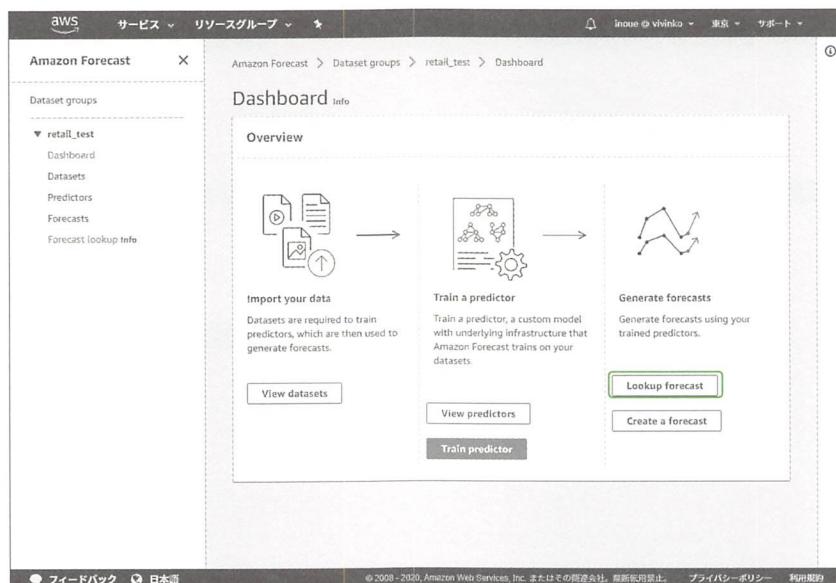


図 3-10-18 予測結果の参照

予測結果を参照するには、図 3-10-18 の画面で「Lookup forecast」ボタンをクリックします。すると、Forecast lookup 画面が表示されるので、まず、Forecast の値として、先ほど作成した予測を選択します。Start date と End date は、予測結果を表示する期間です。ここで指定できる値は、予測子の作成時に指定した予測の周期と期間によって決まります。今回使用しているデータセットには、2010年12月9日までのデータが含まれており、予測の周期として日、期間として10を指定したため、予測結果が表示できるのは2010年12月19日までです。また、Forecast key では、予測結果を表示したい item_id を指定します。予測子の作成時に Forecast dimensions を指定した場合は、その値を使うこともできます。

今回は、予測結果を表示する期間として2010年12月9日～2010年12月19日、item_id として22909を指定し、「Get Forecast」ボタンをクリックします。

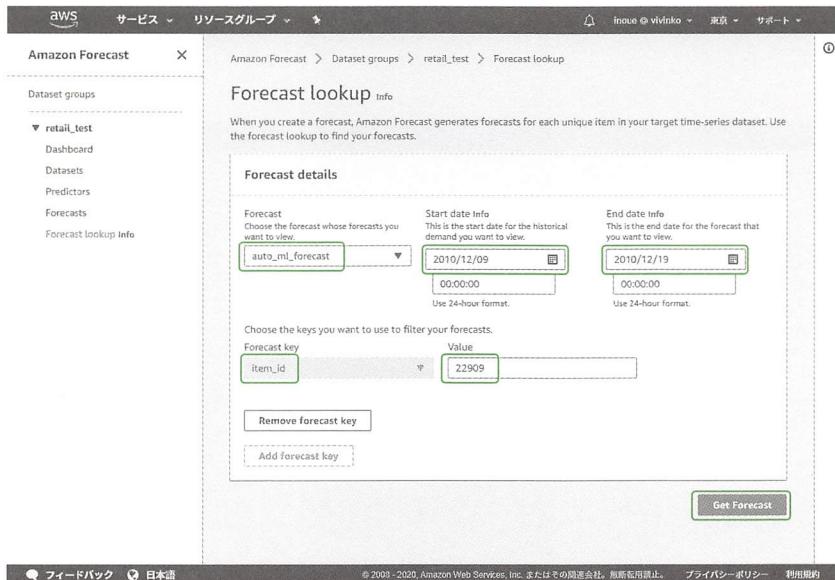


図 3-10-19 Forecast lookup 画面①

画面下部にグラフが表示されます。予測の作成時に指定した Forecast types に応じて複数のグラフが描かれます。今回、Forecast types はデフォルトになっているので、中央値が P50 forecast、その上が P90 forecast (Forecast type が 0.9 の予測結果)、下が P10 forecast (Forecast type が 0.1 の予測結果) のグラフとなります^{*27}。

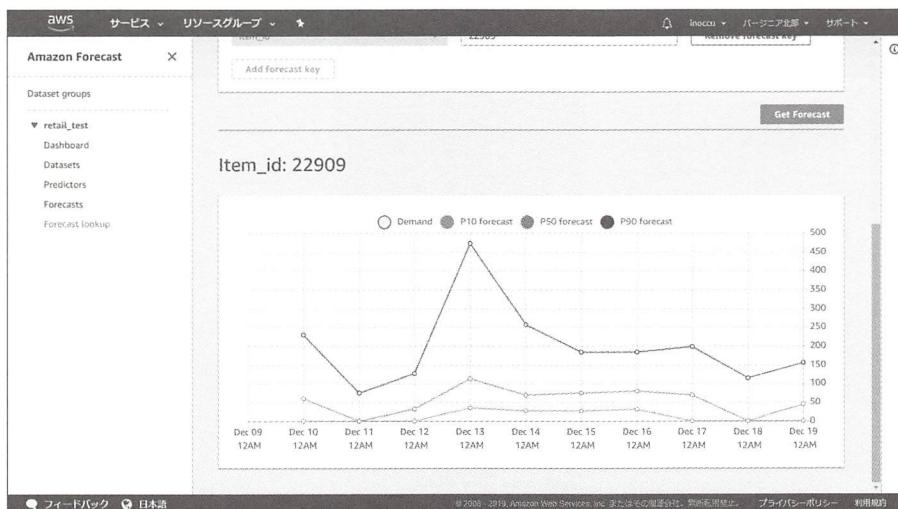


図 3-10-20 Forecast lookup 画面②

*27 同じデータを使用してもトレーニングデータとテストデータの分割などがランダムで行われるため、常に同じモデルが作成されるわけではありません。そのため、掲載した予測結果とは異なる予測が行われることがあります。

予測結果はS3にエクスポートすることもできます。予測(Forecast)を作成した後で、Forecasts画面で予測を選択し、「Create forecast export」ボタンをクリックします。

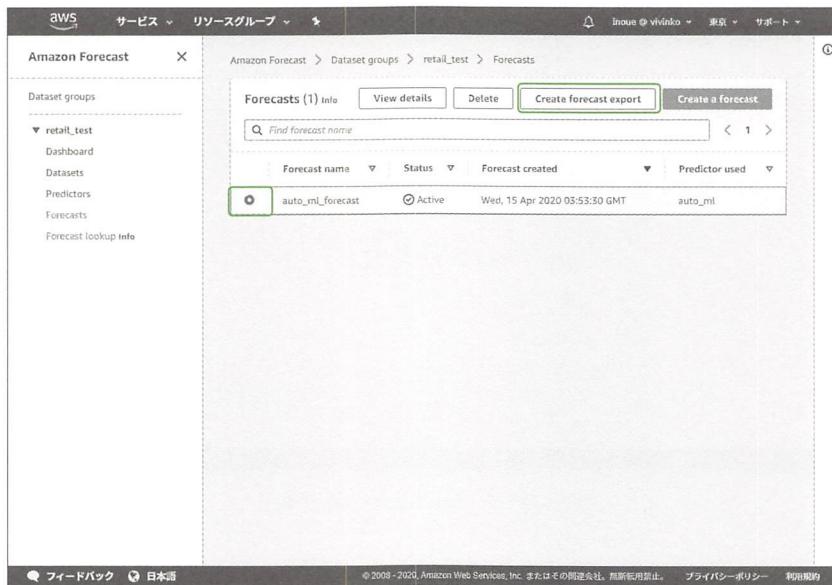


図 3-10-21 予測結果のエクスポート

S3にエクスポートするファイル形式は、CSV形式となります。Export nameに適当な名前を指定し、S3 forecast export locationにエクスポート先のS3バケット名とオブジェクト名を指定します。

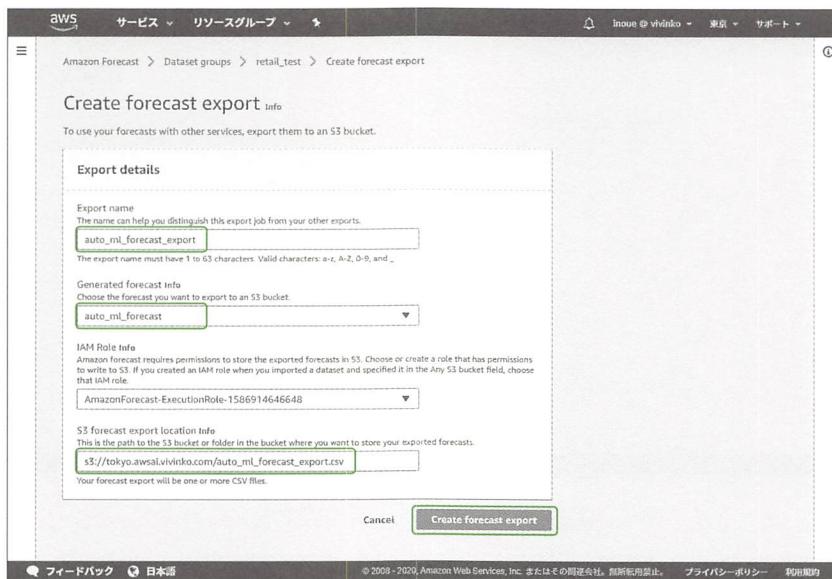


図 3-10-22 Create forecast export 画面

S3 にエクスポートされた CSV ファイルをダウンロードしてみると、図 3-10-23 のような内容が確認できます。商品を特定する item_id、予測対象の日である date と、グラフに描かれていた p10、p50、p90 のそれぞれの予測値がセットされています。

	item_id,date,p10,p50,p90
1	21894,2010-12-10T00:00:00Z,0.0,0.0,4.0
2	21894,2010-12-11T00:00:00Z,0.0,0.0,0.0
3	21894,2010-12-12T00:00:00Z,0.0,0.0,3.0
4	21894,2010-12-13T00:00:00Z,0.0,0.0,4.0
5	21894,2010-12-14T00:00:00Z,0.0,0.0,4.0
6	21894,2010-12-15T00:00:00Z,0.0,0.0,12.0
7	21894,2010-12-16T00:00:00Z,0.0,0.0,12.0
8	21894,2010-12-17T00:00:00Z,0.0,0.0,12.0
9	21894,2010-12-18T00:00:00Z,0.0,0.0,0.0
10	21894,2010-12-19T00:00:00Z,0.0,0.0,3.0
11	22254,2010-12-10T00:00:00Z,0.0,0.0,0.0
12	22254,2010-12-11T00:00:00Z,0.0,0.0,0.0
13	22254,2010-12-12T00:00:00Z,0.0,0.0,0.0
14	22254,2010-12-13T00:00:00Z,0.0,0.0,12.0
15	22254,2010-12-14T00:00:00Z,0.0,0.0,12.0

図 3-10-23 エクスポートした CSV ファイル

3.10.6 予測子メトリクスの参照

ここまで見てきたように、Amazon Forecast では、トレーニング用のデータセットを用いて簡単に時系列予測モデルを作成できますが、気になるのはできあがったモデル（予測子）が、どの程度の精度で予測できるようになっているかです。予測子を作成する際、精度の評価も自動的に行われており、画面上でその指標を参照することができます。

The screenshot shows the AWS Management Console interface for Amazon Forecast. The top navigation bar includes the AWS logo, service dropdown, resource group dropdown, and user information (Inoue @ vivintko, 東京, Support). The main title is "Amazon Forecast > Dataset groups > retail_test > Predictors". On the left sidebar, under "Dataset groups", the "retail_test" dataset is selected, showing "Predictors" as the active tab. Other tabs include "Forecasts" and "Forecast lookup info". The main content area is titled "Predictors (1) Info" and contains a table with one row of data. The table columns are Predictor name, Training status, wQL[0.5]/MAPE, wQL[0.9], wQL[0.1], and Created. The single row shows "auto_ml" as the Predictor name, "Active" as the Training status, and three numerical values: 0.8715, 0.8913, and 0.3822 respectively. The "Created" column shows the date and time as "Wed, 15 Apr 2020 01:48:22 GMT". Navigation buttons for "View details", "Delete", "Create a forecast", and "Train new predictor" are at the top of the table. A search bar "Find predictor name" is also present.

図 3-10-24 Predictors 画面

Predictors 画面では、作成した予測子の概要と精度指標が表示されます。まず、予測子の概要 (Predictor overview) を参照してみましょう。

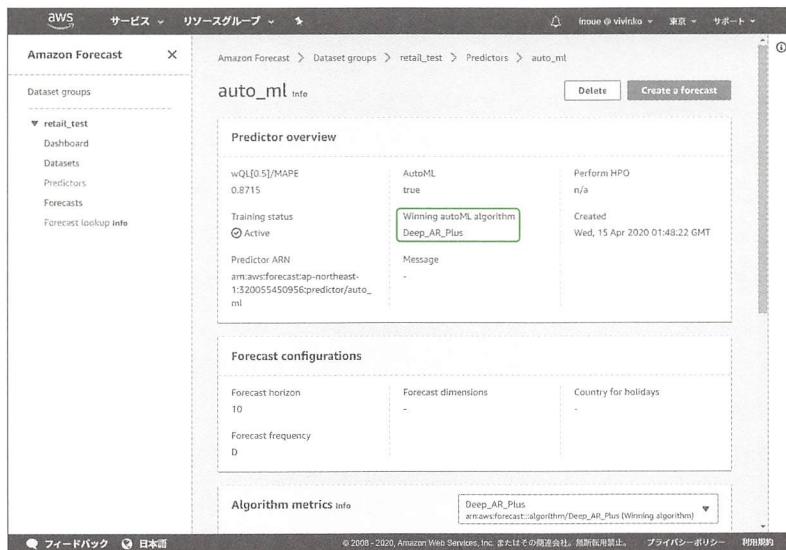


図 3-10-25 Predictor overview の確認

Predictor overview で注目したいのは Winning AutoML Algorithm です。今回は AutoML を指定して予測子を作成したので、Amazon Forecast が予測アルゴリズムを自動的に選定しています。ここでは、その選択された予測アルゴリズムが表示されます。図 3-10-25 を見ると、今回は DeepAR+ (Deep_AR_Plus) が選択されたことがわかります。なお、同じデータを使用してもトレーニングデータとテストデータの分割などがランダムに行われるため、常に同じモデルが作成されるわけではありません。そのため、異なる予測アルゴリズムが選定されることがあります。さて、図 3-10-26 に表示されている wQL[0.5] の値は、予測子の精度指標の値です。これは、次に説明する Algorithm metrics での値と同じです。

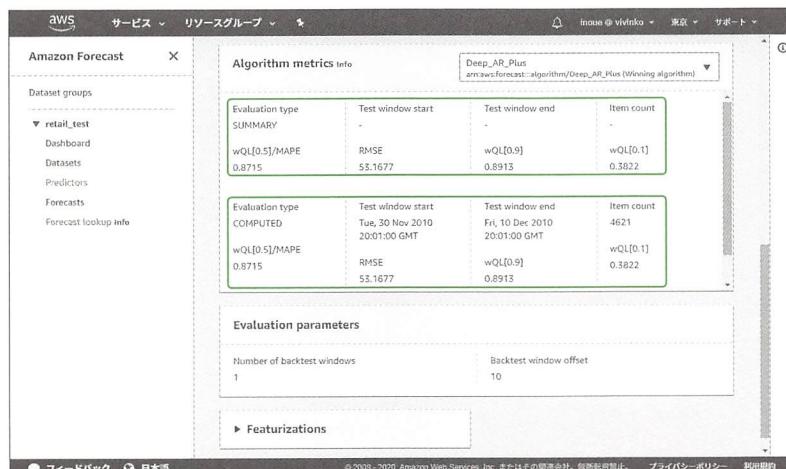


図 3-10-26 Algorithm metrics の確認

予測子の精度指標は Algorithm metrics に表示されます。AutoML を使用した場合、自動的に選択されたアルゴリズムごとに表示されます。Algorithm metrics 欄の右上にあるプルダウンメニューから、表示したいアルゴリズムを選ぶことができます（採用されたアルゴリズムは Winning algorithm として初期表示されています）。アルゴリズムの精度評価は、予測子の作成時に Number of backtest windows で指定した回数だけ行われます。精度指標には SUMMARY と COMPUTED の 2 種類の Evaluation type があり、SUMMARY は、精度評価全体の平均値が表示されます。また、COMPUTED は、Test window start と Test window end の期間でデータを分割して精度評価を行った際の指標が表示されます。なお、Number of backtest windows の値が 1 であった場合、SUMMARY の値と COMPUTED の値は同じになります。

精度指標は、RMSE と wQL の 2 つが表示されます。RMSE は二乗平均平方根誤差です。これは、予測子で予測した値と入力データとして与えた正解値の誤差を平均したもので、つまり、プラスの誤差とマイナスの誤差で打ち消し合わないように、誤差を二乗した後で平均を取り、二乗したことによって数値が大きくなつたことを打ち消すために最後に平方根した値です。この値が小さいほど、予測子の予測精度が良好であることを示します。また、wQL (wQuantileLoss) は、分位点回帰の 0.1 (wQL0.1)、0.5 (wQL0.5)、0.9 (wQL0.9) のそれぞれにおける予測精度を示し、値が小さいほど予測精度が良好であることを示します。

このように、Amazon Forecast により時系列データの予測モデルを簡単に作成し、予測結果を得ることができます。本節では、AWS のコンソールを使って予測子の作成や予測の実行を行いましたが、API を使って行うことも可能です。本番運用では、AWS の S3 や RDS などにあるさまざまなデータを集めて Amazon Forecast 用のデータセットを作成し、API を使ってデータセットのインポートや予測子の作成、予測の実行を行い、最後に S3 に予測結果をエクスポートするという流れになるでしょう。さまざまなビジネス領域の時系列データを扱い、予測を行うことができるので、皆さんも是非活用してみてください。

3.11

Amazon Personalize

3.11.1

Amazon Personalize とは

Amazon Personalize^{*28}は、さまざまな推薦（レコメンデーション）を行うための機械学習モデルを作成するサービスです。もともとオンライン書店としてスタートしたAmazonは、現在、多種多様な商品を扱うオンラインショッピングサイトを運営しています。そこではユーザーの過去の購買データなどを活用して、さまざまな商品の推薦が行われます。皆さんも、Amazonである商品をショッピングカートに入れたら、ほかの商品の購入も推薦されたことがあるでしょう。すべてのユーザーに同じ商品が推薦されているのではなく、あくまで各ユーザーそれぞれの購買履歴などに応じてパーソナライズされています。Amazon Personalizeは、Amazonがこれまで培ってきたパーソナライズの仕組みを取り入れ、AWSのAIサービスとして提供されているのです。

Amazon Personalizeは、ショッピングサイトだけでなくさまざまな用途が考えられます。例えばWebサービスを運営する場合、閲覧しているユーザーごとに表示するコンテンツを変えて、より興味を惹くものにすることは、パーソナライズの最たる例といえるでしょう。また、マーケティングや営業を行う場合、さまざまな顧客に対して、どのような商品やサービスを提案していくのかを決めるることは重要な仕事であり、やはりパーソナライズの例といえます。

3.11.2

Amazon Personalize で使用するデータセット

Amazon Personalizeを使用するためには、前節のAmazon Forecastと同様に、あらかじめ定義されたデータセットタイプのスキーマに合わせてデータセットを準備し、データセットグループを作成する必要があります。

データセットタイプには、ユーザーに関するデータである「User」、商品やサービスに関するデータである「Item」、過去のユーザーとアイテムの関係（例えば購買）に関する履歴データである「User-item interaction」の3つがあります。Amazon Personalizeは基本的に過去の行動（例

*28 <https://aws.amazon.com/jp/personalize/>

えば商品の購買)履歴をもとに、あるユーザーにどのようなアイテム(商品やサービス)を推薦するかを求めるサービスのため、過去の行動履歴である User-item interaction のデータは必須です。User と Item のデータについては、ユーザーやアイテムに関する詳細情報を提供するためのものであり、Amazon Personalize の予測精度を高めるために必要なデータといえます。

Amazon Personalize では、パーソナライズの用途に合わせて、表 3-11-1 に示す事前定義済みのレシピが提供されています。レシピとは機械学習のアルゴリズムに相当するもので、レシピに応じて必要なデータセットタイプが異なります。HRNN レシピから Popularity-Count レシピまでの 4 つのレシピは USER_PERSONALIZATION レシピ、SIMS レシピは RELATED_ITEMS レシピとも呼ばれています。

表 3-11-1 Amazon Personalize で提供されている事前定義済みのレシピ

レシピ名	説明	データセットタイプ(必須)
HRNN	ユーザーがやり取りするアイテムを予測します。ユーザーの行動が時間の経過とともに変化していく場合に適しています。	User-item interaction
HRNN-Metadata	HRNN レシピの派生で、ユーザーやアイテムのデータも活用します。	User-item interaction User Item
HRNN-Coldstart	HRNN-Metadata レシピに、新しいアイテムを含めたパーソナライズの対応が追加されたものです。	User-item interaction User Item
Popularity-Count	User-item interaction データセット内のアイテムに対するイベントのカウントにもとづいて、アイテムの人気度を計算します。	User-item interaction
Personalized-Ranking	パーソナライズされた recommendation のランキングを生成する場合などに使用します。	User-item interaction
SIMS	指定されたアイテムに類似したアイテムを取得します。	User-item interaction

3.11.3 Amazon Personalize へのデータのインポート (データセットグループの作成)

それでは、Amazon Personalize を使ってモデルを構築し、実際に推薦データを取得してみましょう。Amazon Personalize では、構築した推薦モデルをソリューションと呼びます。前節の Amazon Forecast での予測子と同様に、ソリューションの作成と使用（推薦データの取得）は AWS コンソールを操作するか、あるいは API を用いて行います。本書では、AWS コンソールを操作することにします。

前節と同様に、UCI（カリフォルニア大学アーバイン校）の Machine Learning Repository で公開されている Online Retail II Data Set をデータとして使用します。Online Retail II Data Set は、オンラインショップにおける購買履歴が時系列データとしてまとめたものです。前節の Amazon Forecast では購買の日時と量、商品コードを使って、ある商品が毎日どの程度売れそうかを予測しました。これに対し本節では、購買の日時と商品コード、顧客 ID を使って、それぞれの顧客 ID にどの商品を推薦すれば売れそうか、推薦データを得ようというわけです。

まずは、UCI のサイト (<http://archive.ics.uci.edu/ml/datasets/Online+Retail+II>) からデータセットをダウンロードします。ダウンロードの方法と、データセットの詳細は、「3.10.3 Amazon Forecast へのデータのインポート（データセットグループの作成）」の前半部分を参照してください。

さて、必須のデータセットタイプである User-item interaction データを作成します。このデータセットタイプは、USER_ID、ITEM_ID、TIMESTAMP の 3 つの項目が必須です。ダウンロードした Online Retail II Data Set を Excel で開き、下記の前処理を行います。

1. Year 2009-2010 のみを使用するため、他のシートを削除します。
2. StockCode、InvoiceDate、Customer ID 以外の列を削除し、残った 3 列のタイトルを次のように変更します。
 - StockCode → ITEM_ID
 - InvoiceDate → TIMESTAMP
 - Customer ID → USER_ID
3. 列を USER_ID、ITEM_ID、TIMESTAMP の順に並べ替えます。
4. TIMESTAMP 列を日時文字列から long 型の値に変更します（Excel で変換するには、「=(日時文字列 -25569)*86400」という計算を行います。計算を行った後は、その計算結果の値を別の行にコピーし、元の日時文字列は削除してください）。
5. USER_ID が空の行があるので、Excel のフィルター機能などを使用して削除します。

TIMESTAMP 列の計算は、図 3-11-1 のように行います。

The screenshot shows an Excel spreadsheet with four columns: A, B, C, and D. Column A is labeled 'USER_ID', column B is 'ITEM_ID', column C is 'InvoiceDate', and column D is 'TIMESTAMP'. Row 1 contains the headers. Rows 2 through 7 contain data. In row 2, the formula $=(C2-25569)*86400$ is entered in cell D2, and the value 1259653500 is displayed in cell D3. The other rows show various combinations of USER_ID and ITEM_ID, with their corresponding InvoiceDate and calculated TIMESTAMP values.

	A	B	C	D	E
1	USER_ID	ITEM_ID	InvoiceDate	TIMESTAMP	
2	13085	85048	2009/12/1 7:45	1259653500	
3	13085	79323P	2009/12/1 7:45	1259653500	
4	13085	79323W	2009/12/1 7:45	1259653500	
5	13085	22041	2009/12/1 7:45	1259653500	
6	13085	21232	2009/12/1 7:45	1259653500	
7	13085	22064	2009/12/1 7:45	1259653500	

図 3-11-1 Excel での TIMESTAMP 列の計算

保存した CSV ファイルは図 3-11-2 のようになりました。Amazon Personalize へのデータのインポートは S3 から行う必要があるので、事前に CSV ファイルを S3 バケットにアップロードしておきましょう。

The terminal window shows the path C:\> Users > inocu > OneDrive > RIC AWS AI > user_item_interactions.csv. The file contains 11 rows of data, each with three fields: USER_ID, ITEM_ID, and TIMESTAMP. The data is as follows:

	USER_ID	ITEM_ID	TIMESTAMP
1	13085	85048	1259653500
2	13085	79323P	1259653500
3	13085	79323W	1259653500
4	13085	22041	1259653500
5	13085	21232	1259653500
6	13085	22064	1259653500
7	13085	21871	1259653500
8	13085	21523	1259653500
9	13085	22350	1259653560
10	13085	22349	1259653560

図 3-11-2 保存された CSV ファイル

次に、Amazon Personalize のダッシュボードにある「New dataset group」の「Get started」ボタンをクリックします。

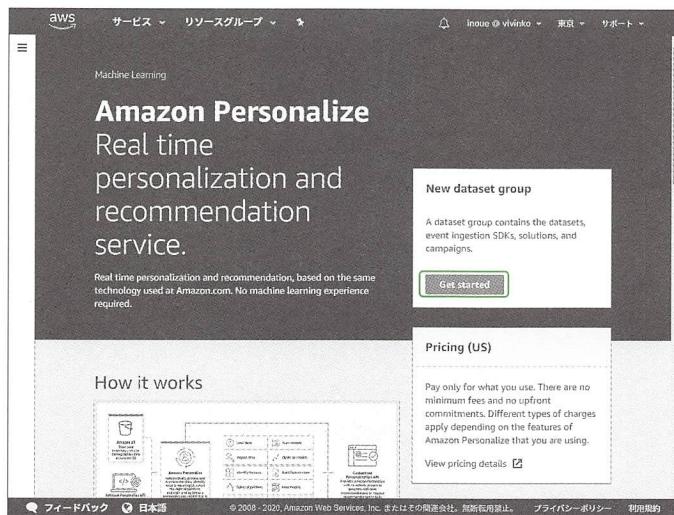


図 3-11-3 新規データセットグループの作成

Create dataset group 画面が開くので、適当なデータグループの名前を付けて、「Next」ボタンをクリックします。

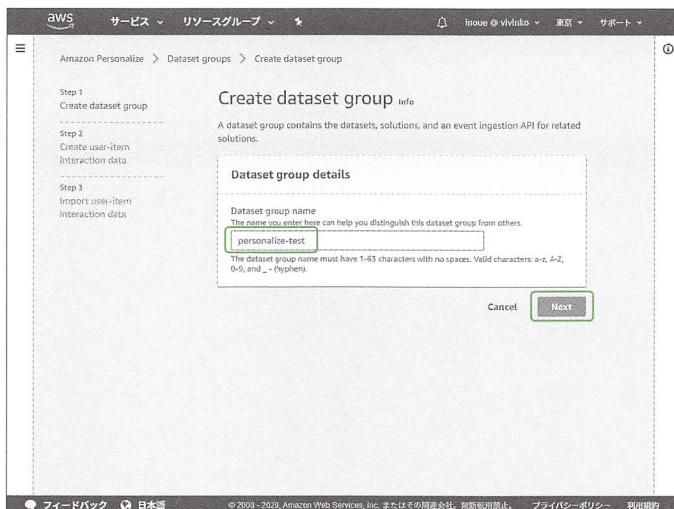


図 3-11-4 Create dataset group 画面

Create user-item interaction data 画面が開きます。まずは必須のデータセットである、User-item interaction データをインポートします。任意の名前を Dataset name 欄に入力し、スキーマの設定を行います。スキーマは、既に作成されていればそこから選択することができます。新規のスキーマを作成する場合は、「Create new schema」を選択します。

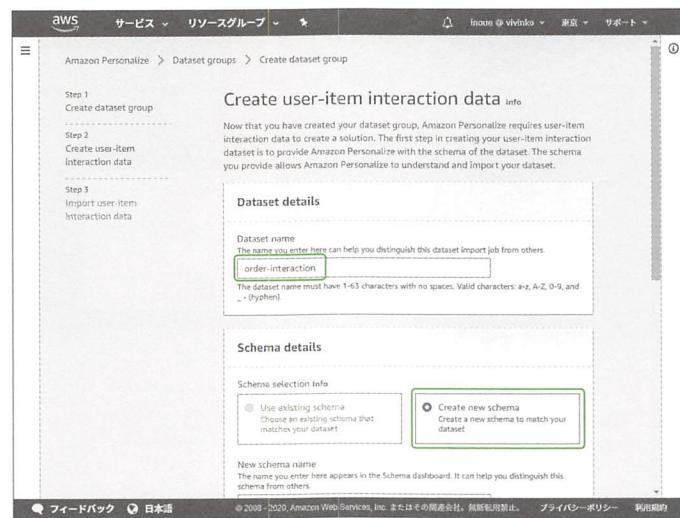


図 3-11-5 Create user-item interaction data 画面

スキーマを新たに作成する場合でも、User-item interaction データセットタイプの必須項目（USER_ID、ITEM_ID、TIMESTAMP という 3つの項目）が指定されたデフォルトのスキーマが Schema definition 欄に表示されており、そのスキーマをそのまま使用できます。今回は必須の 3 項目のみのデータをインポートするので、スキーマに任意の名前を付けて、「Next」ボタンをクリックします。

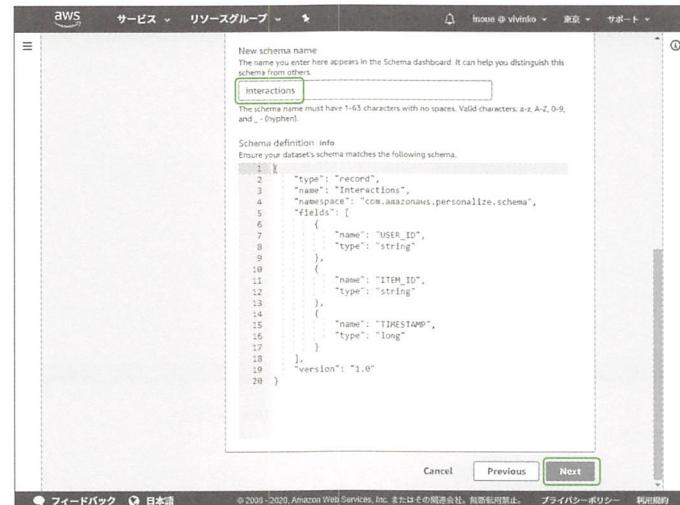


図 3-11-6 スキーマの設定

データセットをインポートするための Import user-item interaction data 画面が開きます。イ

ンポートは非同期で行われるため、ジョブを作成します。適当なジョブの名前を Dataset import job name 欄に入力します。インポートする CSV ファイルは、前述のように S3 バケットから行うので、Amazon Personalize から S3 バケットへのアクセス権限が必要です。アクセス権限の設定の一部は、この画面で行うことができます。IAM service role 欄で「Create a new role」を選択します。

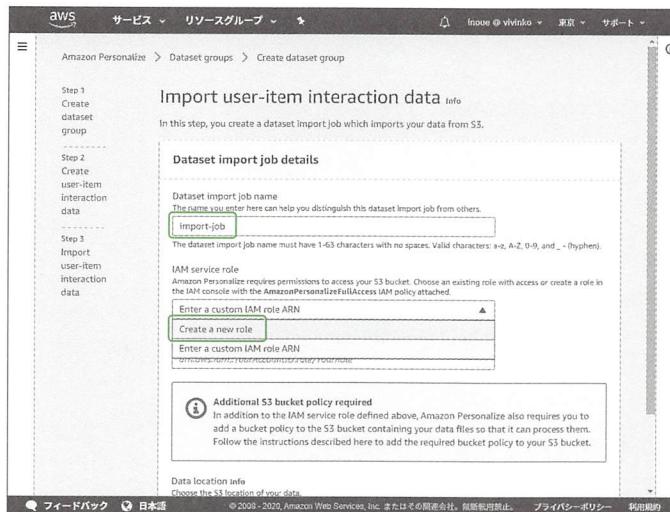


図 3-11-7 Import user-item interaction data 画面

自動的に Create an IAM role 画面が開くので、アクセスを許可する S3 バケットを指定するか、「Any S3 bucket」を選択してすべての S3 バケットへのアクセスを許可して、「Create role」ボタンをクリックします。

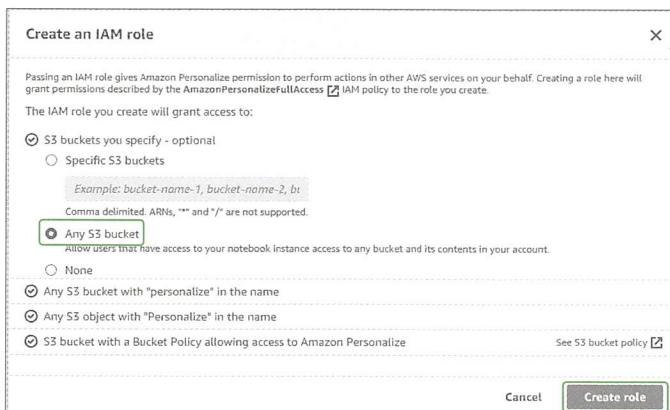


図 3-11-8 IAM Role の作成

先ほどの Import user-item interaction data 画面に戻ると、作成した IAM ロールが選択された状態になっています。前節の Amazon Forecast では、あとは S3 上の CSV ファイルのパスを指定すればよかったです、Amazon Personalize では S3 側でもアクセス権限の設定が必要です。CSV ファイルをアップロードした S3 バケットのアクセス権限画面を開き、バケットポリシーとして以下の内容を入力します。

```
{  
    "Version": "2012-10-17",  
    "Id": "PersonalizeS3BucketAccessPolicy",  
    "Statement": [  
        {  
            "Sid": "PersonalizeS3BucketAccessPolicy",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "personalize.amazonaws.com"  
            },  
            "Action": [  
                "s3:GetObject",  
                "s3>ListBucket"  
            ],  
            "Resource": [  
                "arn:aws:s3:::<バケット名>",  
                "arn:aws:s3:::<バケット名>/*"  
            ]  
        }  
    ]  
}
```

あとは「保存」ボタンをクリックすれば、S3 側のアクセス権限の設定は完了です。

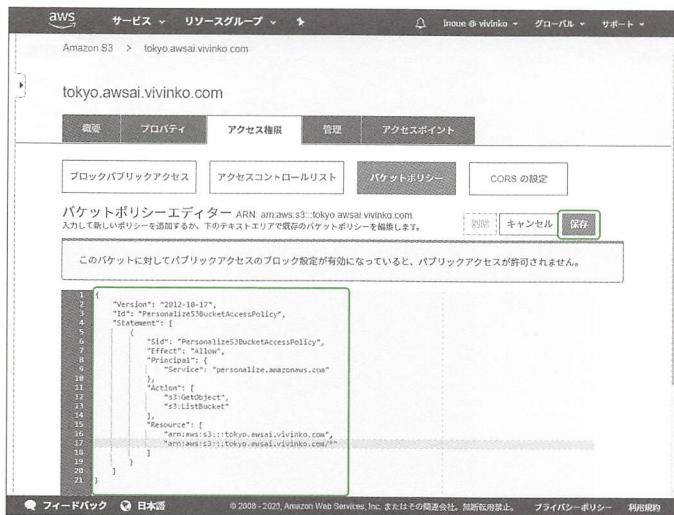


図 3-11-9 S3 のアクセス権限を追加

再び Import user-item interaction data 画面に戻り、Data location 欄で CSV ファイルの S3 上のパス (`s3://<バケット名>/<オブジェクト名>`) を指定します。最後に、「Finish」ボタンをクリックします。

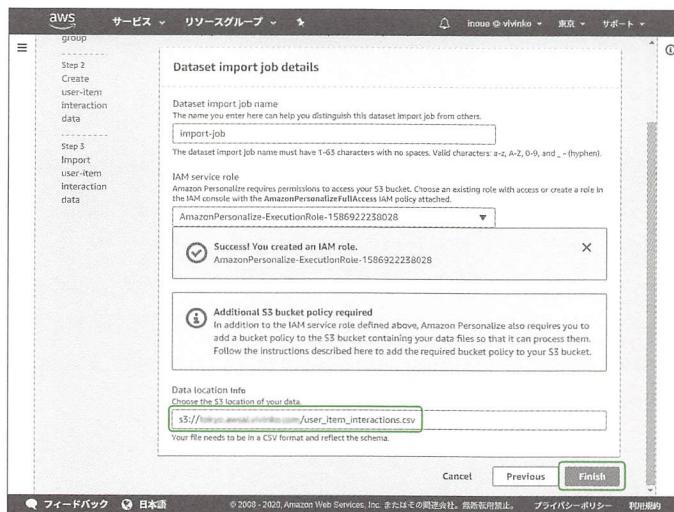


図 3-11-10 S3 上のパスの設定

ここまで操作を行うと、Amazon Personalize にトレーニング用のデータがインポートされます。少し時間がかかるので、Amazon Personalize のダッシュボードなどで状況を確認しましょう。「Active」と表示されれば、インポートは完了です。

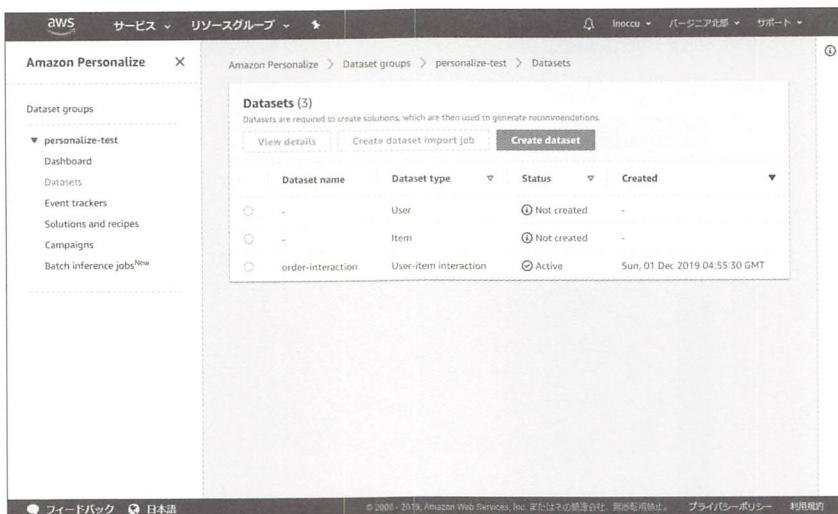


図 3-11-11 データセットのインポートを確認

Amazon Personalize のダッシュボードから Datasets 画面を開くと、データセットグループにインポート済みのデータセットが表示されます。本節では User-item interaction データセットタイプのデータのみを使用しますが、User データセットタイプと Item データセットタイプのデータもインポートする場合は、この画面からデータセットを追加することができます。

COLUMN イベントの記録

本節では既存の履歴データをインポートして、ソリューションの作成を行いますが、Amazon Personalize ではリアルタイムのイベントデータを用いることもできます。また、インポートしたデータとリアルタイムのイベントデータを組み合わせて使用することも可能です。

リアルタイムのイベントデータを使用するためには、イベントが発生する Web サイトなどに、Amazon Personalize イベント取り込み SDK を用いたイベント記録処理を追加します。詳細は、Amazon Personalize のドキュメント (https://docs.aws.amazon.com/ja_jp/personalize/latest/dg/recording-events.html) を参照してください。

3.11.4 ソリューション(ソリューションバージョン)の作成

データセットグループを作成したら、次はソリューションの作成です。Amazon Personalizeでは、データセットグループにインポートしたデータセットを用いてトレーニングしたトレーニング済みモデルを、ソリューションバージョンと呼びます。ソリューションを作成すると、ソリューションバージョンが自動的に作成されます。ソリューションバージョンの作成とは、機械学習の言葉でいうとモデルのトレーニング処理にあたります。

データセットのインポートが完了した後で Amazon Personalize のダッシュボードを開くと、Create solutions の「Start」ボタンがクリック可能な状態になっています。この「Start」ボタンをクリックしてみます。

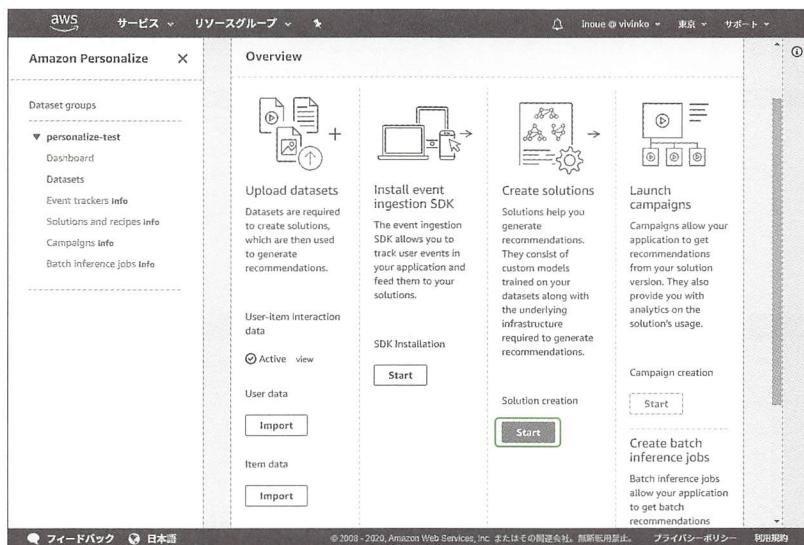


図 3-11-12 ソリューション(ソリューションバージョン)の作成

Create solution 画面が開きます。これから作成するソリューションの名前を Solution name 欄に入力し、次にレシピの選択を行います。レシピの選択は、Manual と Automatic (AutoML) の2つの方法があります。Manual を指定した場合は、画面下の Recipe のプルダウンメニューから、インポートしたデータセットタイプの種類に応じて、最適と思うレシピをユーザー自身が選択します。一方、Automatic を指定した場合は、インポートしたデータセットの内容に応じて Amazon Personalize が自動的に最適なレシピを選択してくれます。

最後に「Next」ボタンをクリックすると、ソリューションの作成が開始されます。

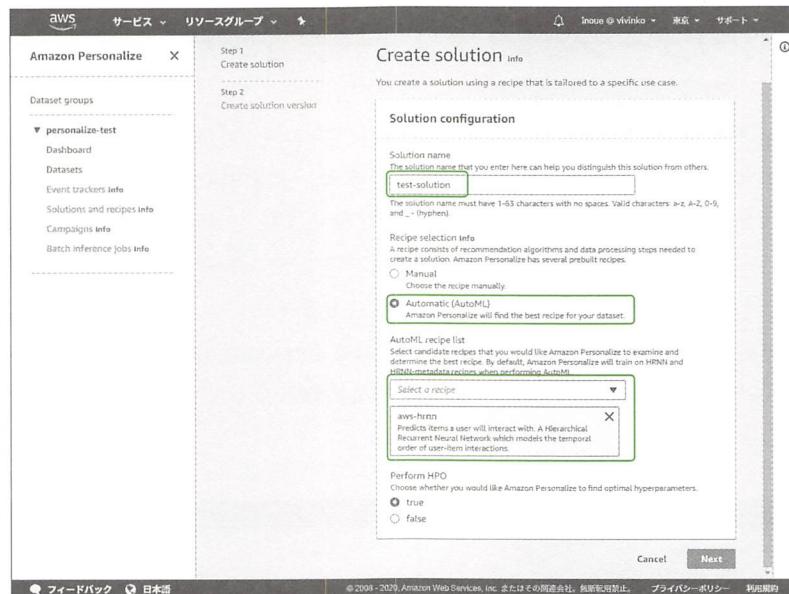


図 3-11-13 Create solution 画面

ソリューション画面を開くと、紐づくソリューションバージョンが自動的に作成されたことを確認できます。

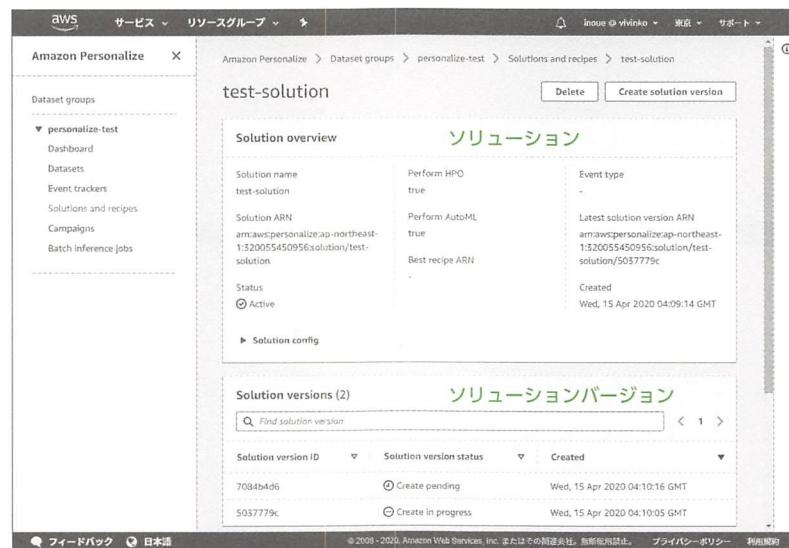


図 3-11-14 ソリューションとソリューションバージョン

前述のように、ソリューションバージョンの作成はモデルのトレーニング処理にあたるため、

ある程度の時間がかかります。処理が完了すると、ステータスが「Active」になります。

Normalized discounted cumulative	Precision	Mean reciprocal rank	Coverage
At 5 0.1631	At 5 0.0645	At 5 0.1929	0.5479
At 10 0.1905	At 10 0.0440		
At 25 0.2474	At 25 0.0313		

図 3-11-15 Solution version overview 画面

ソリューションバージョンの画面には、トレーニング済みモデルの精度評価指標にあたるメトリクスが表示されます。これらのメトリクスは、いずれもレコメンデーションでよく使用され、数値が大きいほど精度が高いことを示しています。それぞれのメトリクスの意味は以下のとおりです。

- Normalized discounted cumulative

モデルによる DCG を正解値の DCG で割ったもの。なお、DCG とは、推薦アイテムを降順に並べてマイナスの重み付けをし、点数を付けた値をいう。

- Precision

上位 K 個 (K の値は At の値で示されます) のモデルによる推薦アイテムのうち、正解値が含まれる割合。

- Mean reciprocal rank

モデルによる推薦アイテムのうち、正解値に含まれる最上位のアイテムの順位が r であるとき、 $1/r$ の値の平均。

それぞれのメトリクスは、At 5、At 10 のような単位で計算されますが、これは推薦順位の上位いくつまで (At 5 であれば上位 5 個) の値であるかを示します。

3.11.5 キャンペーンの作成

ソリューションおよびソリューションバージョンの作成が完了したら、次にキャンペーンを作成します。キャンペーンは、トレーニング済みのモデル（ソリューションバージョン）を API 化したものにあたります。レコメンデーションをリアルタイムで取得しなければならない場合は、キャンペーンの作成が必要です。なお、バッチ処理で構わない場合は、Batch interface job 機能を用いてレコメンデーションを取得することができます。

ソリューションバージョンの作成が完了すると、Amazon Personalize のダッシュボードに「Create new campaign」ボタンがクリック可能な状態で表示されるので、クリックしてキャンペーンの作成を始めます。

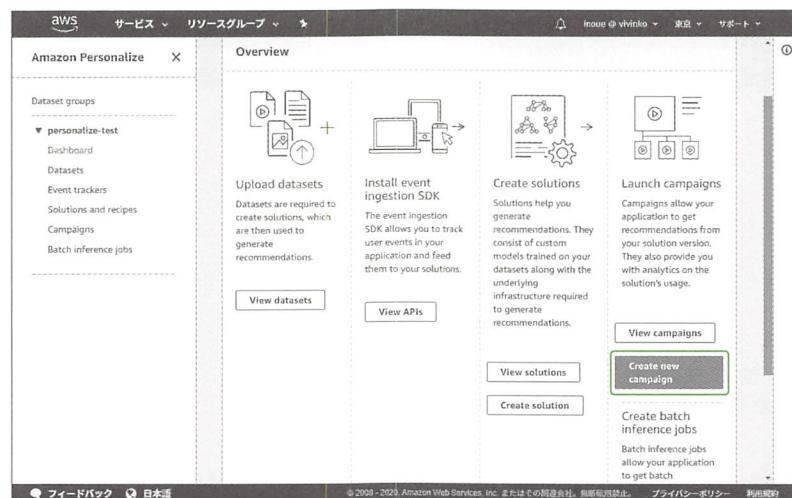


図 3-11-16 新規キャンペーンの作成

表示された Create new campaign 画面で、任意の名前を Campaign name 欄に入力し、使用するソリューションおよびソリューションバージョンを選択します。Minimum provisioned transactions per second（最小プロビジョニング TPS）は、1 秒間の最小トランザクション数であり、これは課金に関係します。課金は、実際に使用したレコメンデーションのトランザクション数に応じて行われますが、使用しなくとも最小プロビジョニング TPS の分は課金されるので注意が必要です。

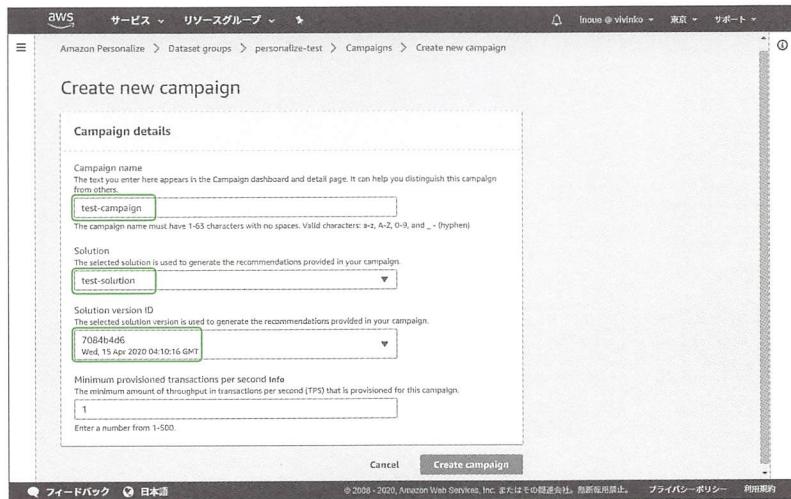


図 3-11-17 Create new campaign 画面

最後に「Create campaign」ボタンをクリックすると、キャンペーンが作成されます。

3.11.6 レコメンデーションの取得

キャンペーンの作成が完了した後、Amazon Personalize のダッシュボードからキャンペーンの画面を開きます。すると、API を呼び出す際に使用する ARN (Amazon Resource Name) と合わせて、Test campaign results 欄が表示され、テスト的にレコメンデーションを取得することができます。ここでは、レコメンデーションを取得するユーザー ID として「13085」を入力し、「Get recommendations」ボタンをクリックします。

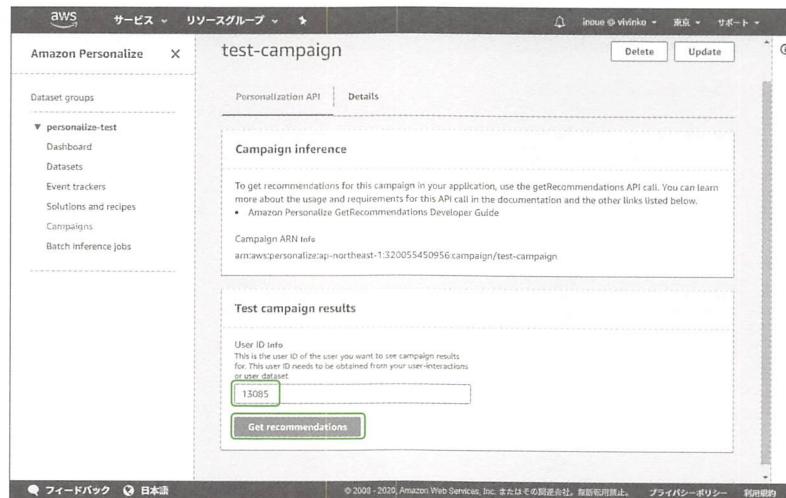


図 3-11-18 レコメンデーションの取得

画面下部に、図 3-11-19 のようなレコメンデーション結果（アイテム ID）が表示されました。実際の活用においては、API でこのようなレコメンデーションを取得し、Web サイトなどに推薦アイテムを掲載することなどが考えられます。

Amazon Personalize		Test campaign results																							
		User ID info																							
Dataset groups		This is the user ID of the user you want to see campaign results for. This user ID needs to be obtained from your user-interactions or user dataset.																							
		13085																							
		Get recommendations																							
		<table border="1"> <thead> <tr> <th>Item ID</th> <th>Score</th> </tr> </thead> <tbody> <tr> <td>M</td> <td>0.0778666</td> </tr> <tr> <td>22423</td> <td>0.0628204</td> </tr> <tr> <td>15056N</td> <td>0.0356072</td> </tr> <tr> <td>85014B</td> <td>0.0241641</td> </tr> <tr> <td>POST</td> <td>0.0198810</td> </tr> <tr> <td>21624</td> <td>0.0152912</td> </tr> <tr> <td>21514</td> <td>0.0143333</td> </tr> <tr> <td>22371</td> <td>0.0131508</td> </tr> <tr> <td>15036</td> <td>0.0122494</td> </tr> <tr> <td>33221</td> <td>0.0118810</td> </tr> </tbody> </table>		Item ID	Score	M	0.0778666	22423	0.0628204	15056N	0.0356072	85014B	0.0241641	POST	0.0198810	21624	0.0152912	21514	0.0143333	22371	0.0131508	15036	0.0122494	33221	0.0118810
Item ID	Score																								
M	0.0778666																								
22423	0.0628204																								
15056N	0.0356072																								
85014B	0.0241641																								
POST	0.0198810																								
21624	0.0152912																								
21514	0.0143333																								
22371	0.0131508																								
15036	0.0122494																								
33221	0.0118810																								

図 3-11-19 取得されたレコメンデーション

Amazon Personalize を使うと、過去の購買履歴データなどをもとに簡単にレコメンデーションを行うことができます。マーケティングの世界では、古くから存在する手法として、幅広い顧客（および顧客候補）に 1 つの商品・サービスを訴求していくマス・マーケティングが用いられていますが、今では、1 人の顧客にそれぞれパーソナライズした商品・サービスを提案する One

to One マーケティングの手法も必要とされ、注目を集めています。

パーソナライズのための IT システムや統計学、機械学習の活用は、日々研究が進められています。こうした研究の先駆けとなってきたのは Amazon であり、Amazon Personalize は、その研究成果を実際に使用できる形で一般公開したものといえます。Amazon Personalize は、One to One で商品やサービスの推薦を行う際に有用なサービスでしょう。

第 4 章

Amazon SageMaker

AI サービスを使えるようになったら、次は独自のモデルを自由に作ることができる Amazon SageMaker にチャレンジしましょう。

SageMaker では、モデルを作成する際、あらかじめ準備されている組み込みアルゴリズムだけでなく、独自のアルゴリズムを用いることができます。また、作成したモデルを AWS の環境上にデプロイして実システムから呼び出して使うことも可能です。SageMaker を使いこなして、AI をより本格的に活用できるようにしましょう。

4.1

SageMaker とは何か

4.1.1 SageMaker でできること

Amazon SageMaker（以下、SageMaker）では、トレーニングデータの加工や教師データの作成、機械学習アルゴリズムを用いたモデルの作成、モデルのトレーニング、トレーニング済みモデルのデプロイ（展開）といった機械学習の一連のプロセスを行う機能が提供されています。モデルは、SageMaker の組み込みアルゴリズムや、scikit-learn^{*1}、TensorFlow といった機械学習フレームワークで提供されている機械学習アルゴリズムなどを用いて作成します。また、サードパーティの構築済みアルゴリズムを AWS Marketplace で購入して使用することも可能です^{*2}。

4.1.2 SageMaker の使用開始

SageMaker を使用するには、AWS コンソールのサービス一覧から「SageMaker」をクリックします。SageMaker は、Web 画面上のコンソールやノートブックを使って操作するほか、第3章で説明した多くの AI サービスと同様に Python SDK などから API を操作して使用することもできます。

まずは、Web 画面上で SageMaker のコンソールを開いてみましょう。

*1 scikit-learn は、機械学習でよく使われる Python ライブラリです。分類や回帰などの処理を、使用するアルゴリズムは違っても同様の構文のプログラムで記述することができます。本書では、SageMaker の組み込みアルゴリズムを用いた機械学習モデルの作成について説明していますが、scikit-learn に慣れている方であれば、SageMaker 上で scikit-learn を使ってモデルを作成することも選択肢の 1 つになります。

*2 AWS Marketplace では、本書執筆時点で 61 個のアルゴリズムが販売されています。例えば、文章の類似度の判定や、需要予測といったモデルを構築するためのアルゴリズムなどがあります。



図 4-1-1 SageMaker のコンソール

4.1.3 SageMaker の機能と画面構成

SageMaker の機能は、画面左のメニューにあるように、Ground Truth、ノートブック、トレーニング、推論という 4 つのカテゴリに分類されています。

第 2 章で示した機械学習のワークフローの図に SageMaker の機能をマッピングすると、図 4-1-2 のようになります。準備したデータについて教師データを作成する「Ground Truth (ラベリング)」、データを加工したり機械学習アルゴリズムを選択または作成してモデルを作成する「ノートブック」、データを使用してモデルをトレーニングさせる「トレーニング (学習)」、学習済みのモデルをデプロイする「推論」といった SageMaker の機能を利用することで、機械学習をより深く活用できます。

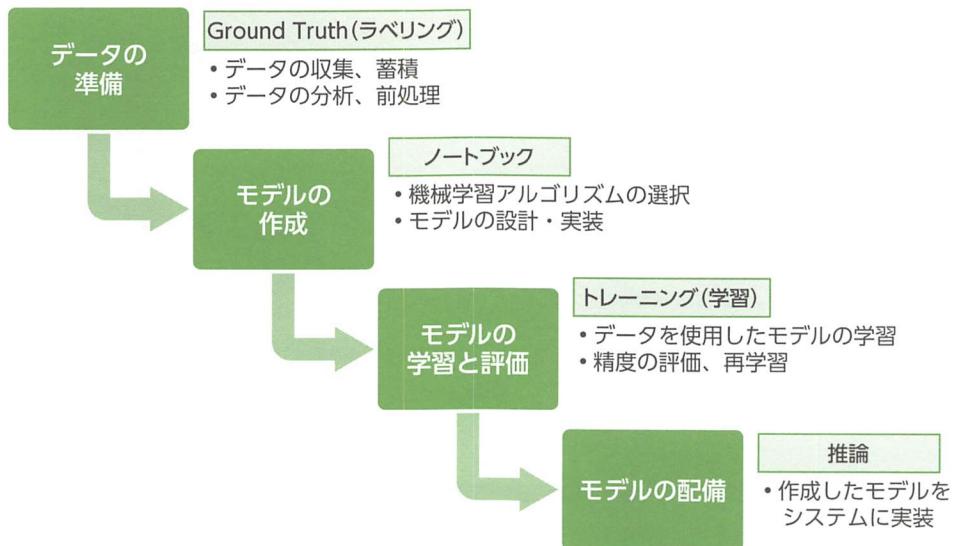


図 4-1-2 機械学習のワークフロー (SageMaker の機能をマッピング)

次項以降、それぞれのカテゴリに沿って機能を簡単に説明していきます。

4.1.4 Ground Truth (ラベリング)

機械学習は、「教師あり学習」と「教師なし学習」に大きく分けることができます（ここに強化学習を加えることもあります）。これらのうち、教師あり学習では、トレーニングデータと対になる教師データを準備する必要があります。例えば、手書きの0～9の数字を分類する機械学習モデルを作成する場合、0、1、2…といった手書き数字の画像データはトレーニングデータです。一方、手書きで0と書かれている画像データに対して、その画像は0と読み取れば正解であることを示す必要があり、その正解を示すデータを教師データやラベルなどといいます。

SageMaker の Ground Truth で提供されている機能は、この教師データを作成するためのものです。トレーニングデータの一つ一つにラベルを付けていく作業は、トレーニングデータそのものを準備する場合に比べて非常に手間がかかりますが、これらの作業を、ラベリングジョブとして SageMaker で管理して共同で遂行することができます。また、ラベリング労働力として、AWS で提供される外部リソース（人員）を用いることも可能です。

4.1.5 ノートブック

SageMaker では、Python を用いたデータサイエンスの現場でよく使われている Jupyter Notebook を使用することができます。SageMaker 上で Jupyter Notebook を使用すれば、ローカル PC などに環境を準備する必要がなく、また、多くの機械学習フレームワークがあらかじめ導入されているので便利です。さらに、ノートブックを GitHub などの Git リポジトリで管理するための機能も提供されています。

SageMaker でノートブックを使用するためには、ノートブックインスタンスを作成します。ノートブックインスタンスは複数作成することができ、また、1つのノートブックインスタンスには複数のノートブックを作成できます。

SageMaker での機械学習モデルの作成作業などは、SageMaker 上のノートブックを使用して行うケースが一般的です。ただ、多くの作業は、Python SDK などを用いてローカル PC 上の Jupyter Notebook を使用することもできますし、SageMaker のコンソール操作だけ（ノートブックを使わない）でモデルを作成することも可能なので、作業の内容や PC 環境などに応じて、適宜使い分けると良いでしょう。

4.1.6 トレーニング（学習）

トレーニングメニューは、SageMaker 上で機械学習（トレーニング）を管理するための機能です。具体的には、機械学習アルゴリズムの管理機能や、トレーニングデータを使用したモデルのトレーニング機能のほか、ハイパープラメータ^{*3}を自動的に調整し、最適なトレーニング済みモデルを作成する機能も提供されています。

SageMaker の組み込みアルゴリズムを使用する場合は、コンソールの操作だけで、CSV データを使ってトレーニングを行い、モデルを作成することができます。また、ノートブックを使って操作する場合も（ローカル PC 上の Jupyter Notebook から SageMaker を操作する場合を含む）、モデルのトレーニングはコンソール上で一元管理されます。

^{*3} ハイパープラメータとは、モデルのトレーニングに関してユーザーが自ら指定するパラメータのことです。また、モデルのトレーニングとは、モデル内部にあるパラメータを機械学習によって自動調整することをいいます。モデル内部にあるパラメータと区別するために、機械学習による自動調整の対象外となるパラメータを「ハイパープラメータ」と呼んでいます。SageMaker のハイパープラメータの調整ジョブは、このハイパープラメータを自動調整の対象にしています。

4.1.7 推論

推論メニューでは、SageMaker のトレーニング機能のほか、別の環境で作成したトレーニング済みのモデルをデプロイし、他のアプリケーションで使用するためのエンドポイントの作成機能や、大量のデータをバッチジョブとして推測処理する機能が提供されています。

4.1.8 S3 バケットの準備と IAM ロールの作成

SageMaker では、トレーニングデータやトレーニング済みモデルなどを保存するために S3 を多用します。そのため、第3章の演習で使用した S3 のバケットをそのまま使うか、あるいは本章での演習のために S3 バケットを新たに作成して、準備しておきましょう。

SageMaker は、東京 (ap-northeast-1) リージョンでも提供されています。東京リージョンを使用する場合は、S3 バケットも ap-northeast-1 に作成します (S3 バケットの作成は「3.2.8 S3 バケットの作成とファイルのアップロード」で説明していますので、参考にしてください)。

さて、SageMaker で作業を行う際には、「AmazonSageMakerFullAccess」という IAM ポリシーと、SageMaker で使用する S3 バケットへのアクセス権限を持つ「IAM ロール」^{*4} が必要です。まずは、これらを作成しておきましょう。IAM ロールの作成は、AWS のコンソールから IAM の画面を開いて行う方法もありますが、SageMaker のトレーニングジョブの作成画面などで自動的に作成することもできます。SageMaker の画面から作成した方が権限の漏れがないので、ここではその方法を説明します。

SageMaker のコンソールでトレーニングジョブの画面を開き、「トレーニングジョブの作成」ボタンをクリックします。

^{*4} IAM ロールは、IAM で付与できるさまざまな権限の組み合わせを定義したものです。ここではノートブックインスタンスに、S3 バケットへのアクセス権限を持つ IAM ロールを設定しています。これにより、ノートブックの実行時に S3 バケットにあるデータを読み込んだり、処理結果を S3 バケットに書き込むことができます。

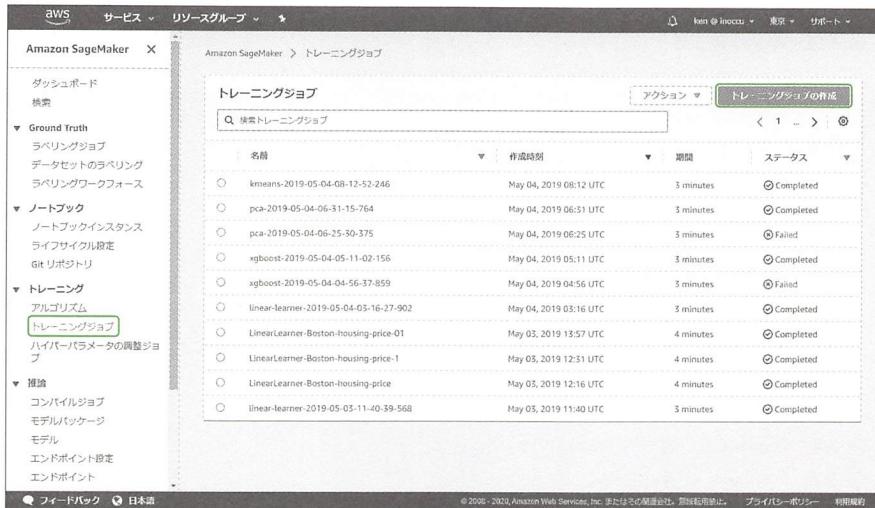


図 4-1-3 トレーニングジョブ画面

トレーニングジョブの作成画面が表示されるので、IAM ロールのプルダウンメニューを開き、「新しいロールの作成」を選択します。

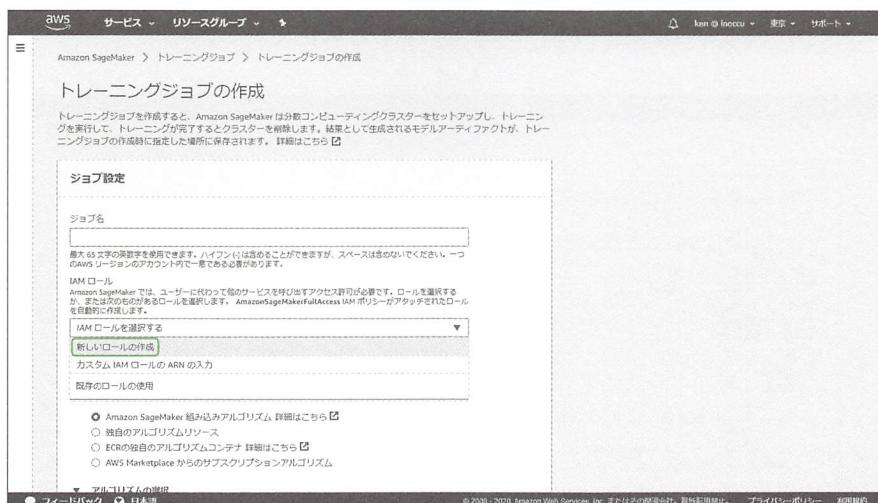


図 4-1-4 トレーニングジョブの作成画面

IAM ロールを作成する画面が開くので、SageMaker で使用する S3 バケットを、カンマ区切りで「特定の S3 バケット」欄に入力します。ここで指定した S3 バケットのほか、名前に「sagemaker」という文字列を含む S3 バケットなどへのアクセス権限が設定されます。次に、「ロールの作成」ボタンをクリックします。

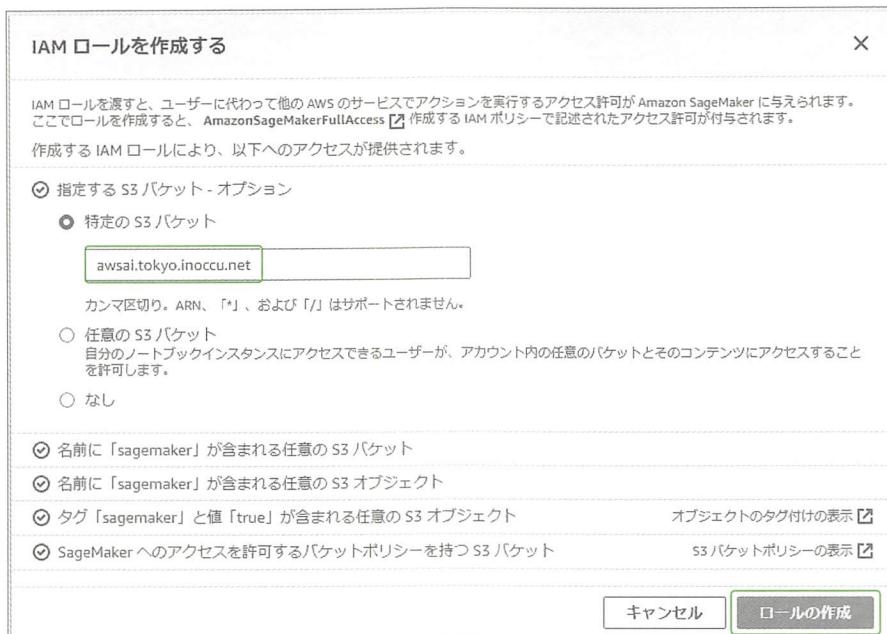


図 4-1-5 IAM ロールの作成

IAM ロール名が自動的に設定されて、IAM ロールの作成が完了します。

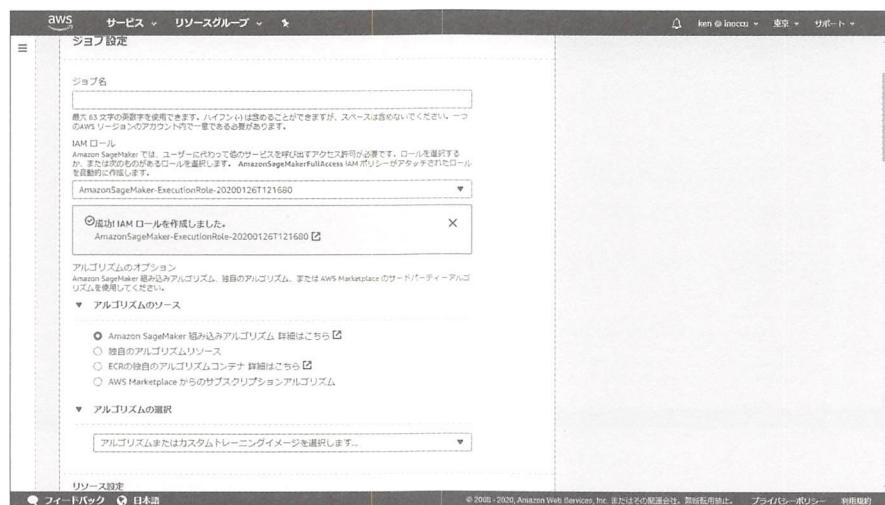


図 4-1-6 IAM ロールの作成完了

本項ではトレーニングジョブの作成画面を開きましたが、IAM ロールの作成は完了したので、この画面を閉じても問題ありません。

4.1.9 SageMaker の課金体系

第3章で説明したAIサービスは、基本的にAPIを呼び出すたびに課金が行われますが、本書のサンプルコードを試す程度であれば課金は数円～数百円程度です。一方、SageMakerは、インスタンスが起動していれば、実際には使用していなくても時間単位で課金されるものがあります。参考として、本書執筆時点におけるアジアパシフィック（東京）インスタンスでの料金目安を以下に示します。

4

- ノートブックインスタンス 0.0608USD／時間 (ml.t2.medium) →約 44USD／月
- オンデマンド ML トレーニング 0.174USD／時間 (ml.m5.large)
- リアルタイム推論用のオンデマンド ML ホスティングインスタンス 0.0851USD／時間 (ml.t2.medium) →約 61USD／月
- バッチ変換用のオンデマンド ML インスタンス 0.174USD／時間 (ml.m5.large)
- ML ストレージ プロビジョニングされたストレージについて 0.168USD／GB／月
- データ処理量（入力） 0.016USD／GB
- データ処理量（出力） 0.016USD／GB

これらのうち、ノートブックインスタンスと、リアルタイム推論用のオンデマンド ML ホスティングインスタンス（モデルのエンドポイントを作成した場合に課金される）は、インスタンスを作成・使用した後で、放置してしまいがちです。ノートブックは、特に必要がなければローカルPC上のJupyter Notebookを使っても問題ありません。また、本番システムで使わない限りはエンドポイントを動作させ続ける必要はありません。

ノートブックインスタンスとリアルタイム推論用のオンデマンド ML ホスティングインスタンスを稼働状態のまま放置してしまうと、月に1万円以上の課金が発生するので、不要になった場合は停止や削除などを行います。

4.2

SageMakerのノートブックを使う

4.2.1 SageMaker のノートブック

SageMaker のノートブックは、Web ブラウザ上で Python プログラムのコーディングや実行を行うためのサービスです。第3章で使用したローカル PC 上の Jupyter Notebook と同じものですが、特に機械学習では、重くなりがちな計算処理を AWS のクラウドを使って行うサービスと考えると良いでしょう。SageMaker では、広く用いられている Jupyter Notebook だけではなく、その後継として開発されている JupyterLab も使用することができます。

4.2.2 ノートブックインスタンスの作成

SageMaker でノートブックを使用するには、まず初めにノートブックインスタンスを作成します。SageMaker のコンソールにあるメニューから「ノートブックインスタンス」を選択し、開いた画面で「ノートブックインスタンスの作成」ボタンをクリックします。



図 4-2-1 ノートブックインスタンス

ノートブックインスタンスに任意の名前を付けて、画面下部の「ノートブックインスタンスの作成」ボタンをクリックします。

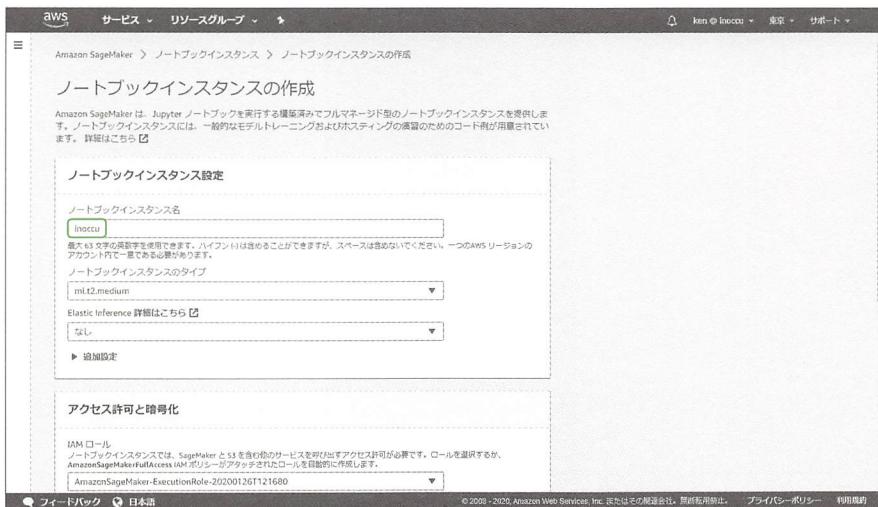


図 4-2-2 ノートブックインスタンスの作成

ノートブックインスタンスが作成されるまで少し時間がかかります。ノートブックインスタンスのステータスは、作成ボタンをクリックした直後は Pending になっていますが、InService になれば作成完了です。



図 4-2-3 ノートブックインスタンスの作成完了

ノートブックインスタンスのステータスが InService になったら、「Jupyter を開く」というリ

ンクをクリックします。すると、Jupyter Notebook の画面が表示されます。

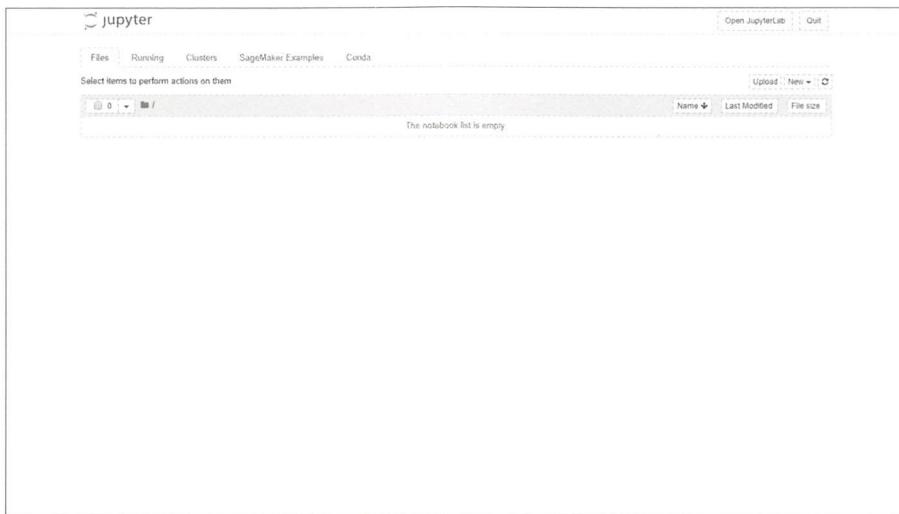


図 4-2-4 SageMaker 上の Jupyter Notebook

なお、JupyterLab を使用する場合は、「JupyterLab を開く」をクリックします。

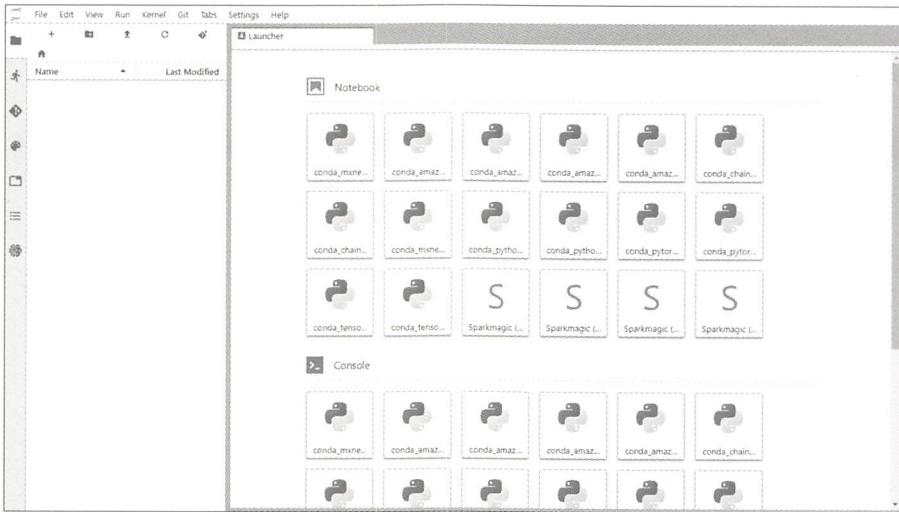


図 4-2-5 SageMaker 上の JupyterLab

SageMaker 上のノートブックでは Python2 および3 の環境だけでなく、TensorFlow、

Chainer^{*5}、PyTorch^{*6}といった機械学習フレームワークがあらかじめ導入された環境も用意されています。また、標準の Python2 および 3 の環境にも Numpy^{*7} や Pandas^{*8} といったデータ分析用のライブラリや、scikit-learn が導入済みですぐに使用できるようになっています。

4.2.3 ノートブックを使う

それでは実際にノートブックを使ってみましょう。今回は JupyterLab ではなく、Jupyter Notebook を使用します。前項で説明したとおり、ノートブックインスタンスの画面から「Jupyterを開く」をクリックして、Jupyter Notebook を開きます。次に、図 4-2-6 のように「New」のプルダウンメニューから「conda_python3」を選択します。

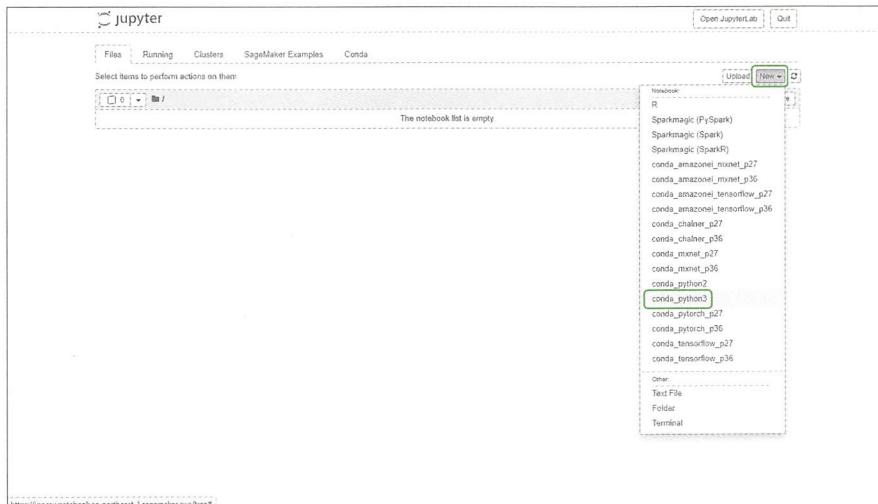


図 4-2-6 ノートブックの作成

- *5 Chainer は、日本の Preferred Networks 社が開発したディープラーニング向けのフレームワークです。Chainer 自体が Python で記述されており、また、ニューラルネットワークの構造を簡単かつ直感的に記述できるという特徴があります。
- *6 PyTorch は、もともと Facebook 社の人工知能研究グループによって開発されたディープラーニング向けのフレームワークです。主に研究目的で使用されています。
- *7 Numpy は、Python で数値計算を行うためのライブラリです。Numpy は C 言語や Fortran で記述されており、純粹に Python を使う場合よりも高速に数値計算することができます。機械学習（ディープラーニングを含む）では数値計算を大量に行う必要があり、Numpy は多くのケースで使用されています。
- *8 Pandas は、Python を用いてデータ分析を行う際に広く活用されているライブラリです。CSV や SQL を用いたデータの読み取りおよびデータ操作を高速に行えるデータフレームの作成、集計などの計算処理のほか、時系列データの取り扱い、Matplotlib 等のグラフ描画ライブラリと組み合わせたデータの可視化などを行うことができます。

図4-2-7のようにノートブックが開きます。ここでは、ローカルPCでのJupyter Notebookと同様に、Pythonによるコーディングや実行が可能です。次節では、このノートブックを使って、モデルの作成からデプロイまで順次行っていきます。

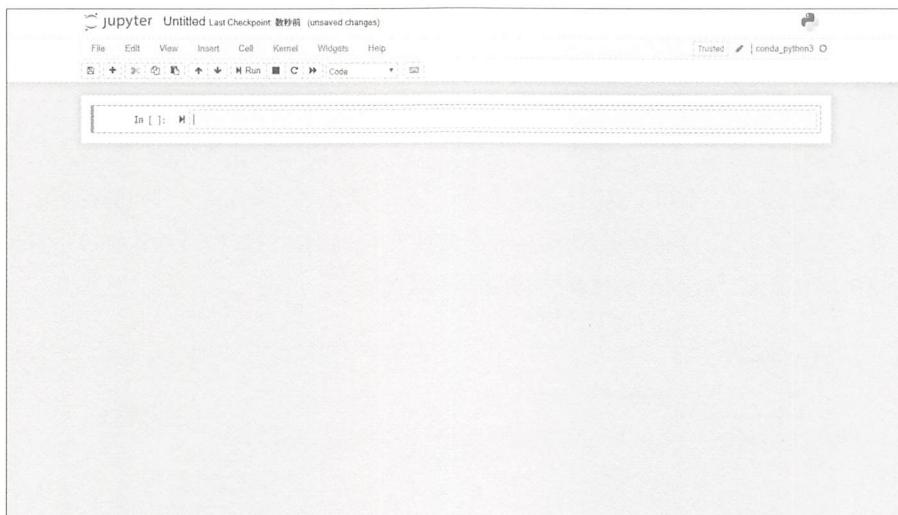


図4-2-7 ノートブック

4.2.4 ノートブックインスタンスの停止

ノートブックインスタンスは、実際には使用していないくとも、起動していれば課金の対象になります。そのため、ノートブックを使用しない場合はノートブックインスタンスを停止しておくと良いでしょう。SageMaker のコンソールでノートブックインスタンスの詳細画面を開き、「停止」ボタンをクリックします。



図 4-2-8 ノートブックインスタンスの停止

ノートブックを再び使用する場合は、同じ画面で「開始」ボタンをクリックしてノートブックインスタンスを起動します。

4.3

SageMakerの組み込みアルゴリズムでモデルを作る

4.3.1

SageMaker の組み込みアルゴリズムを用いたモデルの作成

機械学習モデルを作成するためには、一般的に、scikit-learn や TensorFlow といった機械学習フレームワークを使用することが多いでしょう。SageMaker でも、さまざまな機械学習フレームワークをノートブックですぐに使用できるようになっており、作成したモデルを SageMaker 上にデプロイすることも可能です。しかし、SageMaker でのモデル作成において最初に検討すべきことは、SageMaker にあらかじめ組み込まれているアルゴリズムを、一般的な機械学習フレームワークの代わりに使用することです。本節では、その組み込みアルゴリズムを使って、トレーニングやエンドポイントの作成といった SageMaker での一連の作業を体験していきましょう。

COLUMN

SageMaker のコンソールを使用したモデルの作成

SageMaker では、コンソールの操作だけでモデルを作成することができます。まず、コンソールのトレーニングジョブ画面を表示し、「トレーニングジョブの作成」ボタンをクリックします。次に、トレーニングジョブの作成画面が表示されるので、そこでアルゴリズムを選択し、また、ハイパーパラメータやトレーニングデータなどの指定を行います。

アルゴリズムは、SageMaker の組み込みアルゴリズムなどから選択して使用することができます。



図 4-3-1 アルゴリズムの選択

アルゴリズムを選択すると、そのアルゴリズムで指定可能なハイパーパラメータが表示されるので、編集します。



図 4-3-2 ハイパーパラメータの編集

トレーニングデータなどの入力データとして、S3 上にアップロードしたファイルを指定します。LinearLearner をはじめとする多くのアルゴリズムでは、CSV データを使用できます。なお、CSV データは、カラムの先頭に目的変数を置き、カラム名の行は作らない、というルールがあります。

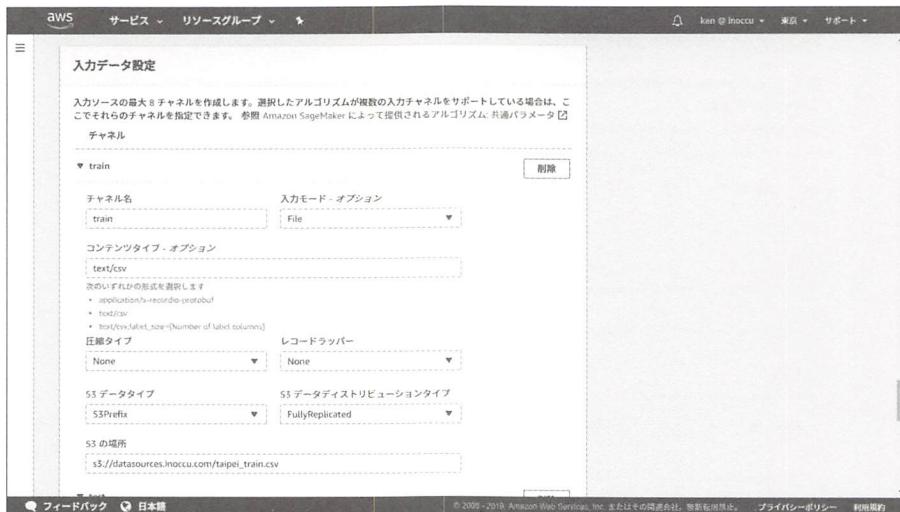


図 4-3-3 入力データの設定

次に、トレーニング済みのモデルを保存するS3出力パスを指定し、「トレーニングジョブの作成」ボタンをクリックします。すると、本章で説明しているノートブックを使用した方法と同様に、モデルのトレーニングが開始されます。

このようにSageMakerのコンソールを使えば、簡単な操作でモデルを作成できますが、多くのケースで必要となるトレーニングデータの加工は、ノートブックを使った方が便利です。本章で説明しているように、データの確認から加工、トレーニングジョブの作成までの操作はノートブックで一貫して行った方が良いかもしれません。

COLUMN SageMakerで独自のアルゴリズムを使用する場合

本章ではSageMakerの組み込みアルゴリズムを使用していますが、一般的に用いられている機械学習フレームワークをSageMaker上のノートブックで使用したり、作成したモデルをSageMaker上にデプロイすることもできます。SageMaker Python SDKでは、以下の機械学習フレームワークをサポートしています。

- Apache Spark
- PyTorch
- TensorFlow
- Chainer
- Apache MXNet
- SparkML Serving
- scikit-learn

詳細は、公式リファレンスの「Amazon SageMakerによるMachine Learningフレームワークの使用」(https://docs.aws.amazon.com/ja_jp/sagemaker/latest/dg/frameworks.html)を参照してください。

4.3.2 SageMaker の組み込みアルゴリズムとは

SageMaker では 17 種類のアルゴリズムが、組み込みアルゴリズムとして提供されています。汎用的なアルゴリズムとしては、教師あり学習で分類や回帰を行う線形学習（LinearLearner）や XGBoost、教師なし学習の K-Means や主成分分析（PCA）などがあります（これらについては次節で説明します）。さらに、画像や文章といった数値以外のデータを扱うアルゴリズムも提供されています。例えば、画像を扱うものとしては、イメージ分類やオブジェクト検出があります。また、文章を扱うものとしては、BlazingText（テキスト分類）やニューラルトピックモデル（文章を分類するためのトピックの特定）というアルゴリズムが提供されています。

本節では、SageMaker でのモデル作成の基本を理解するため、線形学習アルゴリズムを使って、ボストン市の住宅価格を予測する回帰モデルを作成します。その他のアルゴリズムについては次節で説明します。まずは本節で操作に慣れて頂き、読者の皆さんのが自身でモデルを作成する際は、適したアルゴリズムを使用されると良いでしょう。

4.3.3 トレーニングデータの準備

ボストン市の住宅価格に関するデータセット（Boston house-prices）^{*9} は、UCI（カリフォルニア大学アーバイン校）の Machine Learning Repository で提供されていますが、scikit-learn を使うと簡単に準備することができるので、ここではその方法を使います。ノートブックを開き、以下のコードを実行します。

```
from sklearn.datasets import load_boston
boston = load_boston()

print('data:', boston.data[0])
print('target:', boston.target[0])
print('feature_names:', boston.feature_names)
```

^{*9} Harrison, D. and Rubinfeld, D.L. (1978) Hedonic prices and the demand for clean air. *J. Environ. Economics and Management* 5, 81–102.

Belsley D.A., Kuh, E. and Welsch, R.E. (1980) *Regression Diagnostics. Identifying Influential Data and Sources of Collinearity*. New York: Wiley.

```

❷ from sklearn.datasets import load_boston
boston = load_boston()

print('data:', boston.data[0])
print('target:', boston.target[0])
print('feature_names:', boston.feature_names)

data: [6.320e-03 1.800e+01 2.310e+00 0.000e+00 5.380e-01 6.575e+00 6.520e+01
      4.090e+00 1.000e+00 2.960e+02 1.530e+01 3.969e+02 4.980e+00]
target: 24.0
feature_names: ['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
      'B' 'LSTAT']

```

図 4-3-4 データセットの取得と表示

`load_boston()` 関数でデータセットを取得します。説明変数は `data`、目的変数は `target` のリストにセットされています。

説明変数のカラム名が `feature_names` にありますが、これらの意味は表 4-3-1 のとおりです。また、目的変数の値は、オーナーが所有する住宅価格の中央値を 1,000 ドル単位で示したものです。

表 4-3-1 説明変数のカラム

カラム名	説明
CRIM	人口 1 人あたりの犯罪発生率
ZN	25,000 平方フィート以上の住居区画の占める割合
INDUS	小売業以外の商業が占める割合
CHAS	1：チャールズ川の周辺、0：それ以外
NOX	NOx（大気汚染の原因となる窒素酸化物）の濃度
RM	住居の平均部屋数
AGE	1940 年以前に建てられた物件の割合
DIS	ボストン市内にある 5 つの雇用施設からの距離を重み付けした値
RAD	環状高速道路へのアクセスの容易性
TAX	10,000 ドルあたりの不動産税率の総計
PTRATIO	町ごとの児童と教師の比率
B	町ごとの黒人の比率
LSTAT	給与の低い職業に従事する人口の割合

Pandas を使ってデータを表示してみましょう。

```

import pandas as pd
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df['PRICE'] = boston.target
df.head()

```

```
import pandas as pd
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df['PRICE'] = boston.target
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

図 4-3-5 Pandas を用いたデータの表示

次に、データ間の散布図行列を作成し、相関を見てみましょう。

```
%matplotlib inline
import matplotlib.pyplot as plt

pd.plotting.scatter_matrix(df, figsize=(15,15), range_padding=0.2)
```

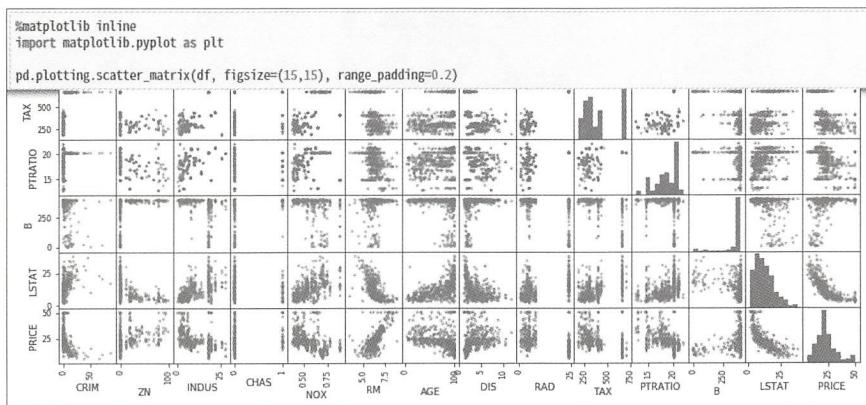


図 4-3-6 散布図行列の表示

目的変数である市場価格（PRICE）は、散布図行列の右端の列です。相関について見てみると、例えば、住居の平均部屋数（RM）との間では正の相関（平均部屋数が多いほど、住宅価格が高い）、給与の低い職業に従事する人口の割合（LSTAT）とは負の相関（給与の低い職業に従事する人口が少ないほど、住宅価格が高い）といったデータの傾向がつかめます。

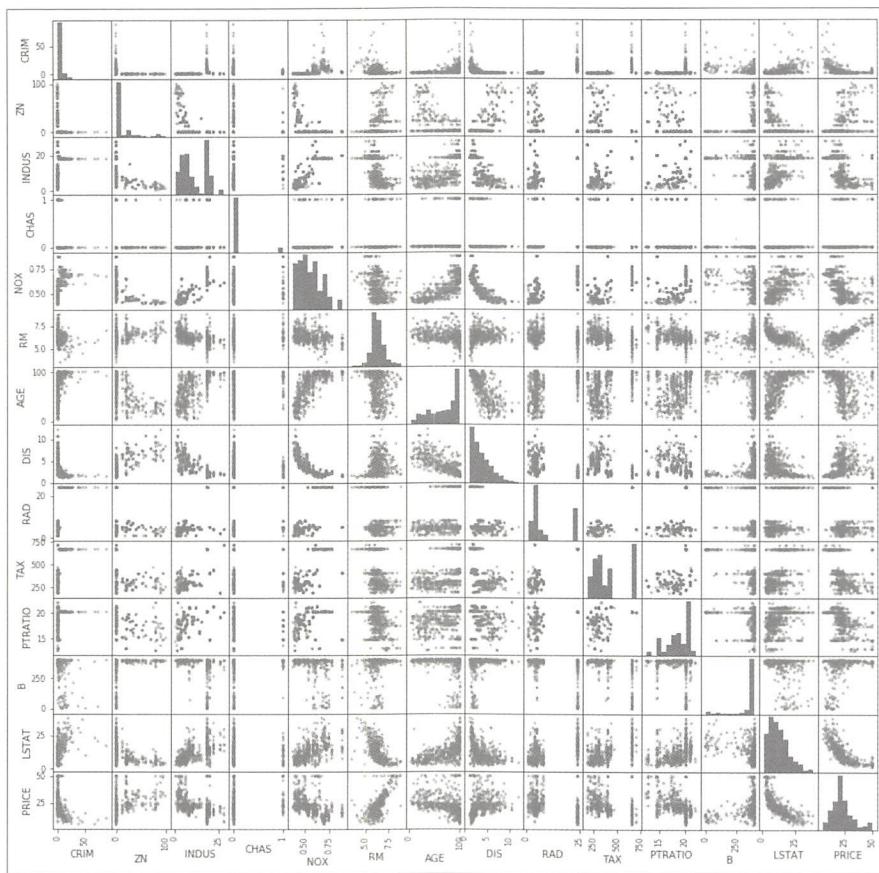


図 4-3-7 ボストン市の住宅価格の散布図行列

4.3.4 トレーニングデータの加工

データの傾向をつかんだら、次はトレーニングデータとして使用できる形に加工していきます。といっても、今回はすべてのデータをそのまま使うので、やるべきことは以下の2つです。

- データの型を float32 型に変換する
- データを、トレーニングデータ、検証データ、テストデータの3つに分割する

scikit-learn で取得したデータセットは float64 型で値がセットされていますが、SageMaker の入力データは float32 型に合わせる必要があるので、以下のコードで変換しておきます。

```
print('boston.data:', boston.data.dtype)
data = boston.data.astype('float32')
print('data:', data.dtype)

print('boston.target:', boston.target.dtype)
target = boston.target.astype('float32')
print('target:', target.dtype)
```

```
print('boston.data:', boston.data.dtype)
data = boston.data.astype('float32')
print('data:', data.dtype)

print('boston.target:', boston.target.dtype)
target = boston.target.astype('float32')
print('target:', target.dtype)
```

boston.data: float64
 data: float32
 boston.target: float64
 target: float32

図 4-3-8 データ型の変換

機械学習を行う際にはデータを、トレーニングデータ、検証データ、テストデータの3つに分割することが一般的です。機械学習においてモデルのトレーニングは、まずトレーニングデータを用いて行います。また、その精度を検証するために検証データ、最終的にできあがったデータをテストするためにテストデータを用います。

データの分割は、scikit-learn で提供されている関数を使用すると良いでしょう。以下のコードでは、まず、全体（506 件）の 20%（102 件）をテストデータとして確保します。次に、残った 80% をさらに分割し、その 30%（122 件）を検証データとします。その残りがトレーニングデータ（282 件）です。

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2)
print(len(X_train), len(X_test))

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.3)
print(len(X_train), len(X_valid))
```

```

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2)
print(len(X_train), len(X_test))

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.3)
print(len(X_train), len(X_valid))

404 102
282 122

```

図 4-3-9 データの分割

4.3.5

ハイパーパラメータの設定とS3へのデータのアップロード

次に、ハイパーパラメータを設定するとともに、準備したデータをS3へアップロードします。SageMakerでは、Python用のSDKがBoto3とは別に準備されており、それを使用してモデルの作成を行います。

まず、以下のコードを実行して、SageMakerで使用するIAMロールを取得します。「4.1.8 S3バケットの準備とIAMロールの作成」で作成したIAMロールが取得されるはずです。

```

import boto3
import sagemaker
from sagemaker import get_execution_role

role = get_execution_role()
print(role)

```

```

import boto3
import sagemaker
from sagemaker import get_execution_role

role = get_execution_role()
print(role)

```

1

図 4-3-10 IAMロールの取得

次に、今回使用する線形学習（LinearLearner）アルゴリズムのインスタンスを作成します。その際にさまざまな引数を指定することができます。最低限必要なのはIAMロール、トレーニングに使用するインスタンスの数（train_instance_count）とタイプ（train_instance_type）、そして、どのような予測を行いたいか（predictor_type）です。predictor_typeは、binary_

classifier（2値分類）、multiclass_classifier（多値分類）、regressor（回帰）のいずれかで指定します。また、output_pathは、モデルを保存するS3バケットとプレフィックスを指定しますが、必須ではありません。ハイパーパラメータの設定^{*10}もここで行いますが、デフォルトのままでトレーニングは可能です。

```
linear = sagemaker.LinearLearner(
    role,
    train_instance_count=<インスタンス数 例：1>,
    train_instance_type='<インスタンスタイプ 例：ml.m5.large>',
    output_path='<モデルを保存するS3バケットとプレフィックス>',
    predictor_type='regressor',
    epochs=<エポック数>,
    early_stopping_patience=<早期終了しないエポック数>
)
```

今回は、train_instance_countを1、train_instance_typeをml.m5.largeとして、predictor_typeはregressorを指定します。また、output_pathは、「4.1.8 S3バケットの準備とIAMロールの作成」で準備したS3バケットとプレフィックスを指定します。

LinearLearnerではデフォルトで、エポック数は15、トレーニングは早期終了（アーリーストッピング）するように設定されています。それを変更するには、引数のepochsとearly_stopping_patienceを設定します。early_stopping_patienceがepochsの値以上の場合は早期終了せず、epochsで指定した回数のトレーニングを行います。

```
linear = sagemaker.LinearLearner(
    role,
    train_instance_count=1,
    train_instance_type='ml.m5.large',
    output_path='s3://awsai.tokyo.inocu.net',
    predictor_type='regressor',
    epochs=100,
    early_stopping_patience=100
)
```

図 4-3-11 LinearLearner インスタンスの作成

次に、SageMakerの組み込みアルゴリズムで使用可能な形式^{*11}にデータを変換した上で、そ

*10 LinearLearnerのハイパーパラメータとしては、コード例に挙げたエポック数や早期終了しないエポック数、さらに、P184でfit関数の引数として指定しているミニバッチサイズなどがあります。なお、ハイパーパラメータの自動調整について、P186の「COLUMN」で説明していますのでご参照ください。

*11 SageMakerの組み込みアルゴリズムで使用可能なデータ形式は、アルゴリズムによって異なります。多くのアルゴリズムではCSVデータを使用できますが、本節のようにrecord_set関数でアップロードする場合はprotobuf形式になります。

のデータを S3 へアップロードします。LinearLearner ではトレーニングデータに train、検証データに validation、テストデータに test というチャネル名を付けることになっているので、それに従って channel 引数を指定しています。なお、戻り値は RecordSet 型の値です。

```
train = linear.record_set(X_train, labels=y_train, channel='train')
validation = linear.record_set(X_valid, labels=y_valid, channel='validation')
test = linear.record_set(X_test, labels=y_test, channel='test')
```

```
train = linear.record_set(X_train, labels=y_train, channel='train')
validation = linear.record_set(X_valid, labels=y_valid, channel='validation')
test = linear.record_set(X_test, labels=y_test, channel='test')
```

図 4-3-12 RecordSet の作成

S3 バケットを確認すると、「sagemaker」という文字列で始まるバケットが自動的に作成され、3つのフォルダができてデータがアップロードされていることがわかります（先ほど LinearLearner インスタンスを作成した際に指定した output_path に保存されるわけではありません。output_path は、この後で行うモデルのトレーニング結果を保存するパスとして用いられます）。

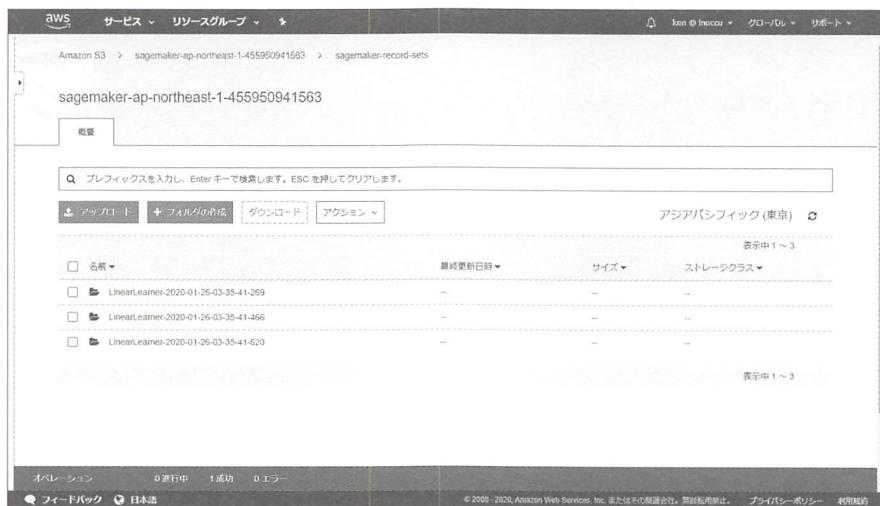


図 4-3-13 S3 コンソールを用いたデータ作成の確認

COLUMN**ローカル PC の Jupyter Notebook を使用する場合**

本節では、SageMakerのノートブックを使用する前提で、コードと、その実行例を説明しています。ローカルPCのJupyter Notebookを使用する場合は、以下の点に注意してください。

- SageMaker Python SDK のインストール

以下のコードをJupyter Notebook上で実行してインストールします。

```
!pip install requests==2.20.1
!pip install --upgrade sagemaker
```

- `get_execution_role()` が動作しない

`get_execution_role()`が動作しないため、IAM ロールはARN文字列を直接指定する必要があります。

- `sagemaker_session` を指定する

SageMaker Python SDKのLinearLearnerインスタンスを取得する際、本節のコード例では、引数に `sagemaker_session` を指定しなくても、ノートブックが動作している環境から自動的にセッションを取得してインスタンスが作成できています。しかし、ローカルPCのJupyter Notebookではセッションの自動取得ができないため、手動でセッションを取得して引数にセットします。

ローカルPCのJupyter Notebookでは、以下のようなコードでLinearLearnerインスタンスを作成できます (`epochs` や `early_stopping_patience` といった引数は、本節のコードと同様に指定することができます)。

```
import boto3
import sagemaker

role = '<IAMロールのARN文字列>'

boto_session = boto3.session.Session(region_name='ap-northeast-1')
sagemaker_session = sagemaker.Session(boto_session=boto_session)

linear = sagemaker.LinearLearner(
    role,
    train_instance_count=<インスタンス数>,
    train_instance_type='<インスタンスタイプ>',
```

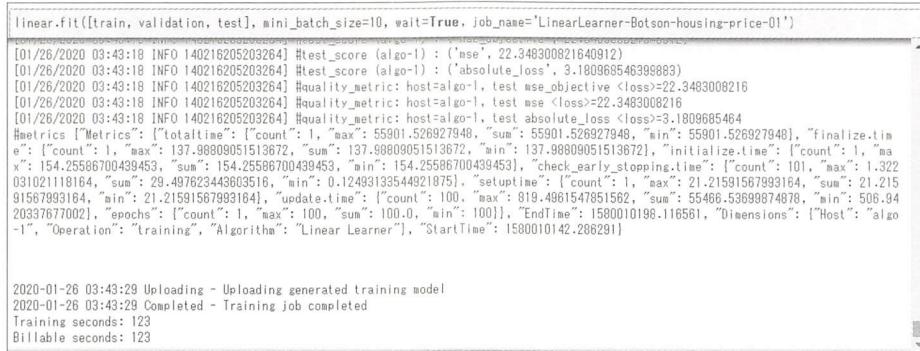
```
output_path='<モデルを保存するS3バケットとプレフィックス>',
predictor_type='regressor',
sagemaker_session=sagemaker_session
)
```

4.3.6 トレーニングジョブの作成

次に、fit関数でトレーニングジョブを作成し、モデルのトレーニングを実行させます。引数には、先ほど作成した3つのRecordSet（train、validation、test）をリストで指定します。また、ミニバッチ学習^{*12}を行うためのmini_batch_sizeの指定もここで行います。

```
linear.fit([train, validation, test], mini_batch_size=10, wait=True, job_name='<ジョブ名>')
```

トレーニングジョブは、完了するまで少し時間がかかります。wait引数をTrueに設定した場合、ノートブック上では、ジョブが完了するまで待ち状態となります。なお、job_nameは、指定しなかった場合、アルゴリズム名とタイムスタンプを含む名前が自動的にセットされます。



```
linear.fit([train, validation, test], mini_batch_size=10, wait=True, job_name='LinearLearner-Boston-housing-price-01')
[01/26/2020 03:43:18 INFO 140216205203264] #test_score [algo-1] : ('mse', 22.348300821640912)
[01/26/2020 03:43:18 INFO 140216205203264] #test_score [algo-1] : ('absolute_loss', 3.180968546399883)
[01/26/2020 03:43:18 INFO 140216205203264] #quality_metric: host:algo-1, test mse_objective <loss>=22.3483008216
[01/26/2020 03:43:18 INFO 140216205203264] #quality_metric: host:algo-1, test mse <loss>=22.3483008216
[01/26/2020 03:43:18 INFO 140216205203264] #quality_metric: host:algo-1, test absolute_loss <loss>=3.1809685464
Metrics [{"name": "Metrics", "values": [{"metric": "total_time", "min": 55901.526927948, "max": 55901.526927948, "sum": 55901.526927948}, {"metric": "finalize_time", "min": 154.25586700439453, "max": 154.25586700439453, "sum": 154.25586700439453}, {"metric": "initialize_time", "min": 137.98809051513672, "max": 137.98809051513672, "sum": 137.98809051513672}, {"metric": "check_early_stopping_time", "min": 101, "max": 101, "sum": 101}, {"metric": "setup_time", "min": 21.2591567993164, "max": 21.2591567993164, "sum": 21.2591567993164}, {"metric": "update_time", "min": 819.4961547851562, "max": 819.4961547851562, "sum": 819.4961547851562}, {"metric": "epochs", "min": 100, "max": 100, "sum": 100}, {"metric": "dimensions", "min": 118561, "max": 118561, "sum": 118561}], "dimensions": [{"Host": "algo-1", "Operation": "training", "Algorithm": "Linear Learner"}]}, {"time": "2020-01-26 03:43:29 Uploading - Uploading generated training model", "seconds": 123}, {"time": "2020-01-26 03:43:29 Completed - Training job completed", "seconds": 123}, {"time": "Training seconds: 123", "seconds": 123}, {"time": "Billable seconds: 123", "seconds": 123}]
```

図 4-3-14 トレーニングジョブの作成と実行

^{*12} モデルのトレーニングでは、トレーニングデータを用いて推論をいったん行い、教師データとの誤差を小さくするための調整を自動で行います。このとき、すべてのトレーニングデータを用いて推論を行った後で調整する方法をバッチ学習、一つ一つのトレーニングデータごとに調整する方法をオンライン学習といいます。ミニバッチ学習は、バッチ学習とオンライン学習の中間的な方法であり、mini_batch_sizeで指定したトレーニングデータの個数ごとに調整を行います。

ノートブック上のログを最後から少し戻すと、「test_score」、「quality_metric」という部分が見えます。ここに、モデルの精度指標である平均二乗誤差（MSE）と絶対値誤差（Absolute Loss）の値が出力されています。作成したモデルが予測した値と実際の値の誤差の平均値を確認したい場合は平均二乗誤差、中央値を確認したい場合は絶対値誤差の値を見ると良いでしょう。

```
#Metrics [{"Metrics": {"Max Batches Seen Between Resets": {"count": 1, "max": 11, "sum": 11.0, "min": 11}, "Number of Batches Since Reset": {"count": 1, "max": 11, "sum": 11.0, "min": 11}, "Number of Records Since Last Reset": {"count": 1, "max": 102, "sum": 102.0, "min": 102}, "Total Batches Seen": {"count": 1, "max": 11, "sum": 11.0, "min": 11}, "Total Records Seen": {"count": 1, "max": 102.0, "sum": 102.0, "min": 102}, "Reset Count": {1, "max": 1, "sum": 1.0, "min": 1}, "EndTime": "1580010198.11229", "Dimensions": {"Host": "algo-1", "Meta": "test_data_iter", "Op": "training", "Algorithm": "Linear Learner"}, "StartTime": "1580010198.102537"}]
[01/26/2020 03:43:18 INFO 140216205203264] #test_score (algo-1) : ('mse_objective', 22.348300821640912)
[01/26/2020 03:43:18 INFO 140216205203264] #test_score (algo-1) : ('mse', 22.348300821640912)
[01/26/2020 03:43:18 INFO 140216205203264] #test_score (algo-1) : ('absolute_loss', 3.180968546399883)
[01/26/2020 03:43:18 INFO 140216205203264] #quality_metric: host=algo-1, test mse_objective <loss>=22.3483008216
[01/26/2020 03:43:18 INFO 140216205203264] #quality_metric: host=algo-1, test mse <loss>=22.3483008216
[01/26/2020 03:43:18 INFO 140216205203264] #quality_metric: host=algo-1, test absolute_loss <loss>=3.180968546399883
#Metrics [{"Metrics": {"totaltime": {"count": 1, "max": 55901.526927948, "sum": 55901.526927948, "min": 55901.526927948}, "final":
```

図 4-3-15 トレーニング結果の確認

SageMaker のコンソールでトレーニングジョブの画面を開くと、ノートブック上で作成したトレーニングジョブが表示されます。ステータスが Completed になっていれば、トレーニングは完了しています。

The screenshot shows the Amazon SageMaker console with the 'Training Jobs' page selected. On the left, there's a sidebar with various navigation options like 'Ground Truth', 'Notebook', 'Training', and 'Algorithms'. The 'Training' section is expanded, showing a list of training jobs. One job, 'LinearLearner-Boston-housing-price-01', is highlighted with a green border. The table lists the following details for each job:

名前	作成時間	期間	ステータス
LinearLearner-Boston-housing-price-01	Jan 26, 2020 03:39 UTC	4 minutes	Completed
kmeans-2019-05-04-08-12-52-246	May 04, 2019 08:12 UTC	3 minutes	Completed
pca-2019-05-04-06-31-15-764	May 04, 2019 06:31 UTC	3 minutes	Completed
pca-2019-05-04-06-25-30-375	May 04, 2019 06:25 UTC	3 minutes	Failed
xgbtrain-2019-05-04-11-02-156	May 04, 2019 05:11 UTC	3 minutes	Completed
xgbtrain-2019-05-04-04-56-37-859	May 04, 2019 04:56 UTC	3 minutes	Failed
linear-learner-2019-05-04-03-16-27-902	May 04, 2019 03:16 UTC	3 minutes	Completed
LinearLearner-Boston-housing-price-01	May 03, 2019 12:57 UTC	4 minutes	Completed
LinearLearner-Boston-housing-price-1	May 03, 2019 12:31 UTC	4 minutes	Completed
LinearLearner-Boston-housing-price	May 03, 2019 12:16 UTC	4 minutes	Completed

図 4-3-16 コンソールでのトレーニングジョブの確認

COLUMN ハイパーパラメータの自動調整

モデルのトレーニング時に指定するハイパーパラメータは、基本的にはユーザーが自ら指定しますが、SageMakerには、ハイパーパラメータを自動調整する機能があります。SageMakerのコンソールから「ハイパーパラメータの調整ジョブ」を開き、自動調整するためのジョブを作成します。

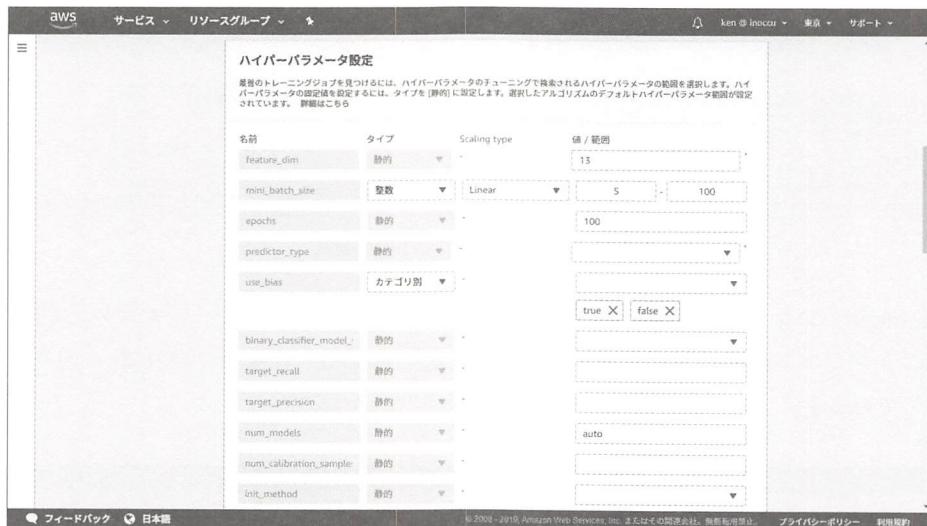


図 4-3-17 ハイパーパラメータの自動調整

自動調整が可能なハイパーパラメータは、アルゴリズムによって異なります。例えば線形学習アルゴリズムの場合は、mini_batch_size や learning_rate (学習率) などを自動調整できます。また、下限値と上限値を指定しておけば、その間で適した値を自動的に見つけ出して、精度の高いモデルを作成してくれます。

4.3.7 精度の評価

SageMaker では、モデルのトレーニングを行った際に、さまざまな精度指標が Amazon CloudWatch^{*13}（以下、CloudWatch）に記録されます。その一部は、トレーニングジョブの詳細画面に表示されます。先ほどトレーニングジョブのステータスが Completed になったこと

*13 Amazon CloudWatch は、AWS のさまざまなサービスと連携できるモニタリング（ログやリソース使用率などの監視）のサービスです。SageMaker のトレーニングジョブに関するモニタリングも CloudWatch で行うことができます。

を確認した画面で、そのトレーニングジョブ名をクリックし、詳細画面を表示してみます。すると、画面下部にモニタリングという項目があり、いくつかのグラフが表示されています。DiskUtilization や CPUUtilization といったグラフは、トレーニングで消費したディスク容量や CPU の使用量を示すもので、インスタンスマトリクスといいます。一方、train:objective_loss や test:objective_loss のような、train、validation、test のいずれかが名前の先頭に付くグラフは、モデルのトレーニング進捗や精度を示すもので、アルゴリズムメトリクスといいます。

「アルゴリズムメトリクスの表示」、「インスタンスマトリクスの表示」をクリックするとメトリクスが、また、「ログの表示」をクリックするとトレーニングに関するログが、それぞれ CloudWatch の画面に表示されます。

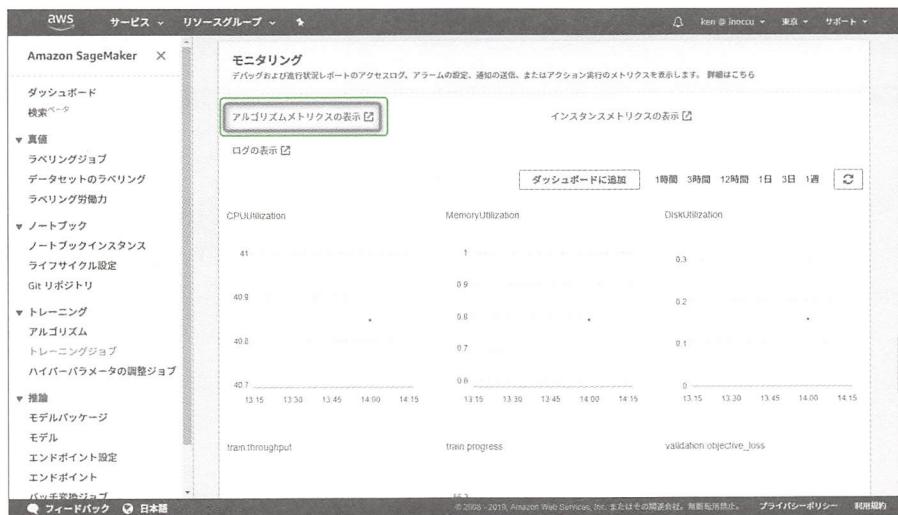


図 4-3-18 トレーニングジョブのモニタリング

画面をさらに下にスクロールして、test:objective_loss のグラフを見てみましょう。これはテストデータを用いた精度評価のグラフです。objective_loss には、LinearLearner で回帰モデルを作成した場合（predictor_type に regressor を指定した場合）、平均二乗誤差の値がセットされています。平均二乗誤差とは、モデルで予測を行った値と教師データの値を比較し、その差を二乗して平均した値です。予測値が教師データより大きい場合、誤差は正の値となり、逆の場合は負の値となります。そのまま平均すると正と負の差が打ち消し合って誤差が小さく見えてしまうため二乗してすべて正の値にしています。ここでは平均二乗誤差は 223 となっています。



図 4-3-19 平均二乗誤差の確認

「アルゴリズムメトリクスの表示」をクリックして CloudWatch の画面に移ってみましょう。トレーニングジョブ名で絞り込みが行われた状態になっているので、メトリクス名「train:object_loss」、「validation:object_loss」、「test:object_loss」の3つを選択してグラフを表示します。トレーニングデータに対する平均二乗誤差は 0.254 であり、トレーニングデータでのトレーニングはきちんと行われているようです。一方、検証データに対する平均二乗誤差は 27、テストデータに対する平均二乗誤差はトレーニングジョブの詳細画面で確認したのと同じ値である 22.3 となっています。



図 4-3-20 CloudWatch による平均二乗誤差の確認

4.3.8 モデルの作成

トレーニング済みのモデルは、LinearLearner のインスタンス作成時に指定した output_path の S3 バケットに保存されています。実際に S3 の画面で表示してみると、「トレーニングジョブ名 /output」というフォルダの中に、「model.tar.gz」というファイルができていることがわかります。



図 4-3-21 トレーニング済みモデルの出力の確認

このモデルを SageMaker 上にデプロイするためには、SageMaker 上のモデルとして登録しておく必要があります。SageMaker のコンソールでトレーニングジョブの詳細画面を開き、「モデルの作成」ボタンをクリックします。



図 4-3-22 トレーニングジョブの詳細画面

モデルの作成画面では、任意のモデル名を指定し、IAM ロールは「4.1.8 S3 バケットの準備」と IAM ロールの作成で作成済みのものを選択します。それ以外の項目はデフォルト（初期値）のままで構いません。最後に、画面下部にある「モデルの作成」ボタンをクリックします。

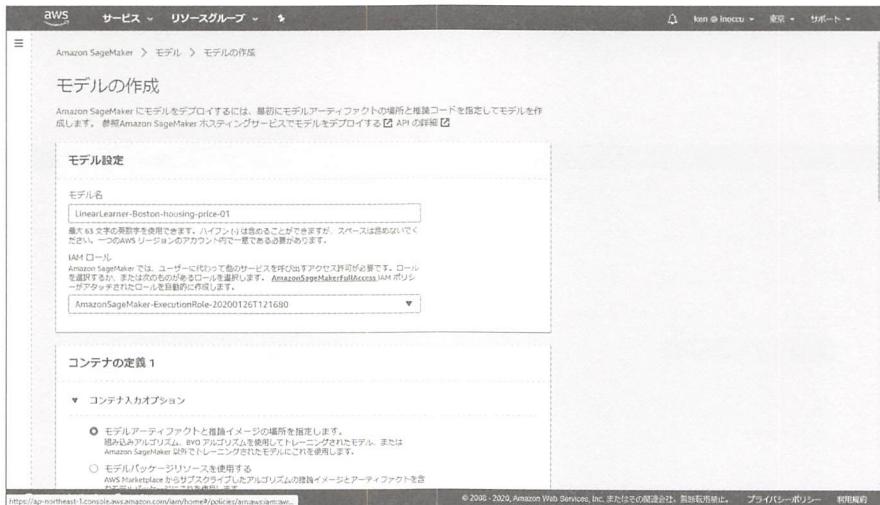


図 4-3-23 モデルの作成

これでモデルの作成は完了です。

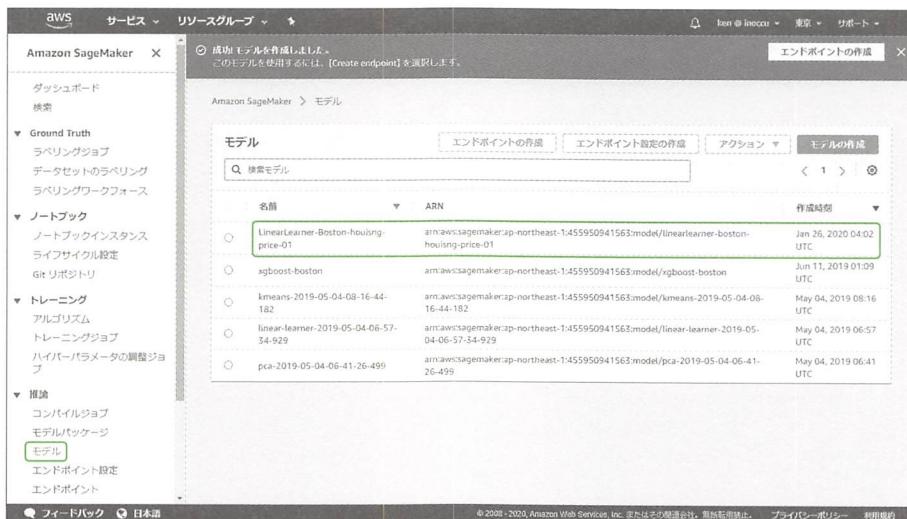


図 4-3-24 モデルの作成完了

4.3.9 エンドポイント設定の作成

次にエンドポイントを作成しますが、その前に、作成するエンドポイントに紐付ける「エンドポイント設定」の作成を行います。SageMaker のコンソールでエンドポイント設定画面を表示し、「エンドポイント設定の作成」ボタンをクリックします。



図 4-3-25 エンドポイント設定の作成

エンドポイント設定の作成画面が開くので、任意のエンドポイント設定名を入力し、「モデルの追加」をクリックします。

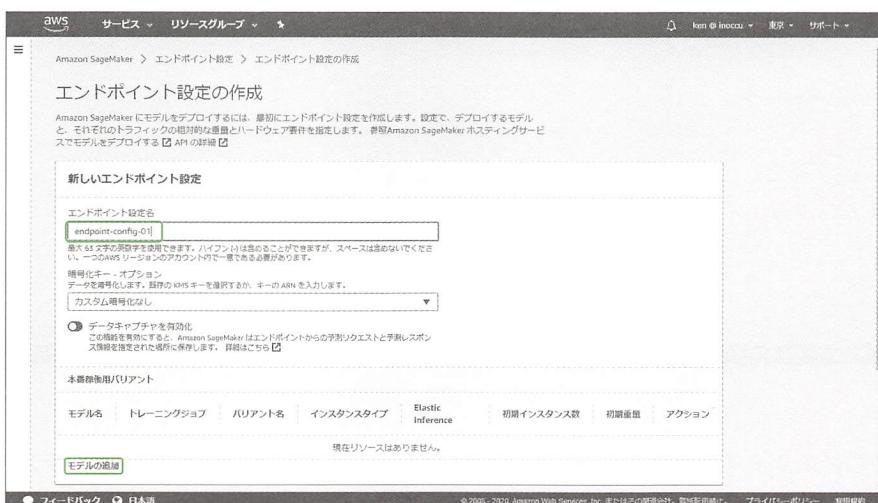


図 4-3-26 エンドポイント設定

SageMaker に登録されているモデルが表示されます。前項(P190)で作成したモデルを選択し、「保存」ボタンをクリックします。



図 4-3-27 モデルの追加

「本番稼働用バリエント」欄に、選択したモデルが表示されます。また、エンドポイントで使用するインスタンスタイプや初期インスタンス数などが自動設定されます。さて、ここでは「編集」をクリックして、インスタンスタイプを編集します。



図 4-3-28 エンドポイント設定に追加されたモデルの表示

インスタンスタイプの初期値は ml.m4.xlarge が指定されていますが、高性能なインスタンス

タイプは料金が高く、アカウントごとの使用可能数もデフォルトでは小さく設定されているため、ここでは ml.t2.medium に変更し、「保存」ボタンをクリックします。

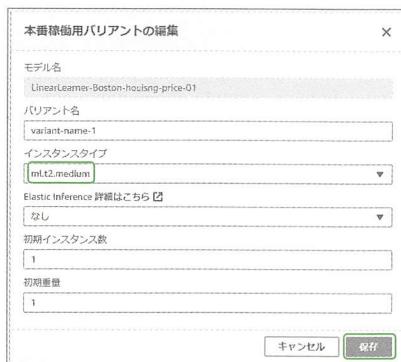


図 4-3-29 インスタンスタイプの編集

最後に、「エンドポイント設定の作成」ボタンをクリックします。

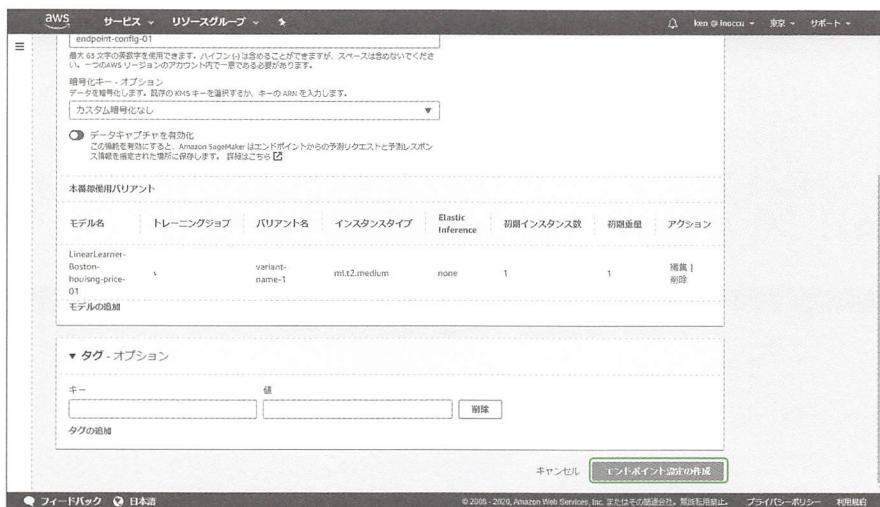


図 4-3-30 エンドポイント設定の作成（実行）

4.3.10 エンドポイントの作成

次に、エンドポイントを作成します。エンドポイントを作成すると、SageMaker の Python SDK を使用して予測を行ったり、Web API として使用したりすることができます。

SageMaker のコンソールでモデル画面を開き、前項（P190）で作成したモデルをクリックしてモデルの詳細画面を表示します。次に、画面上部にある「エンドポイントの作成」ボタンをクリックします。

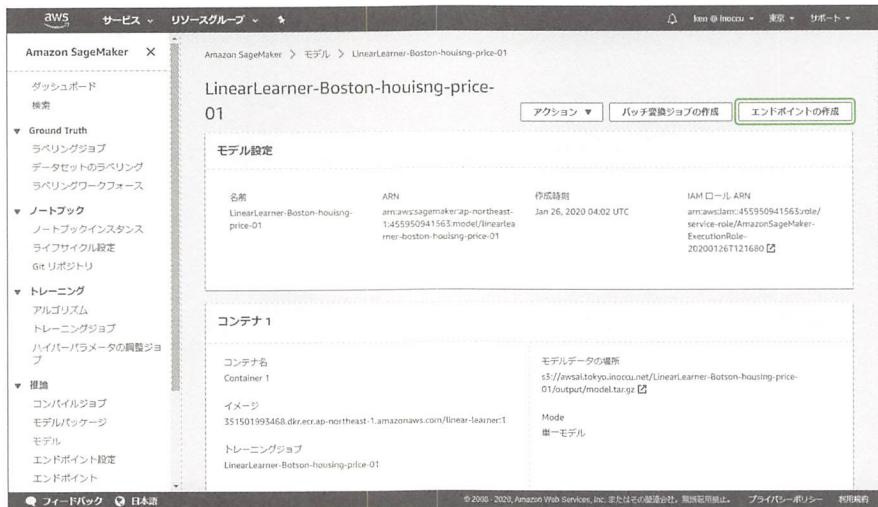


図 4-3-31 エンドポイントの作成

「エンドポイントの作成と設定」画面が開くので、まず、任意のエンドポイント名を指定します。次に、エンドポイント設定のアタッチを行います。ここでは、「既存のエンドポイント設定の使用」を選択します。

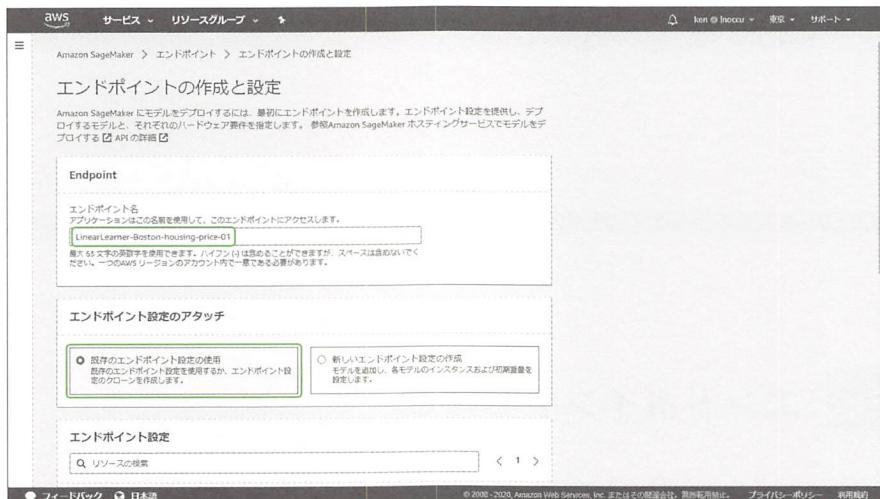


図 4-3-32 エンドポイント名の設定とエンドポイント設定のアタッチ

画面を下にスクロールし、先ほど作成したエンドポイント設定を選択して「エンドポイント設定の選択」ボタンをクリックします。



図 4-3-33 エンドポイント設定の選択

最後に、「エンドポイントの作成」ボタンをクリックします。

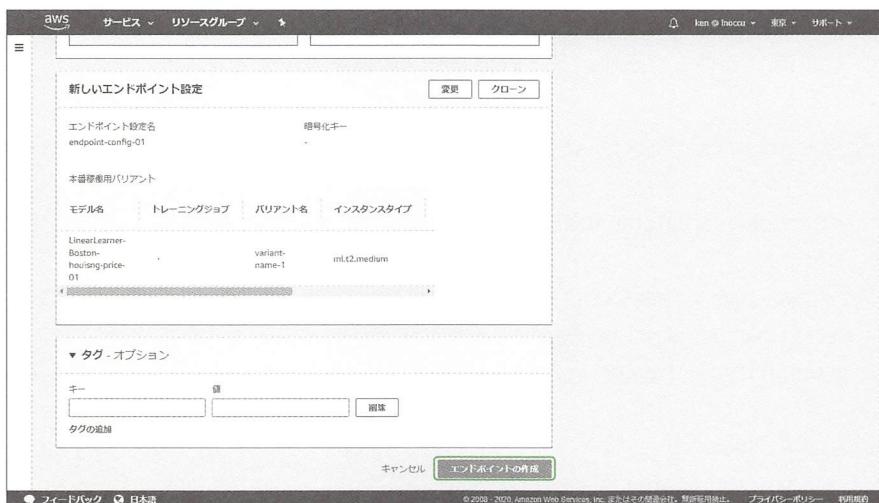


図 4-3-34 エンドポイントの作成 (実行)

エンドポイントの作成には少し時間がかかります。ステータスが InService に変われば作成完了です。

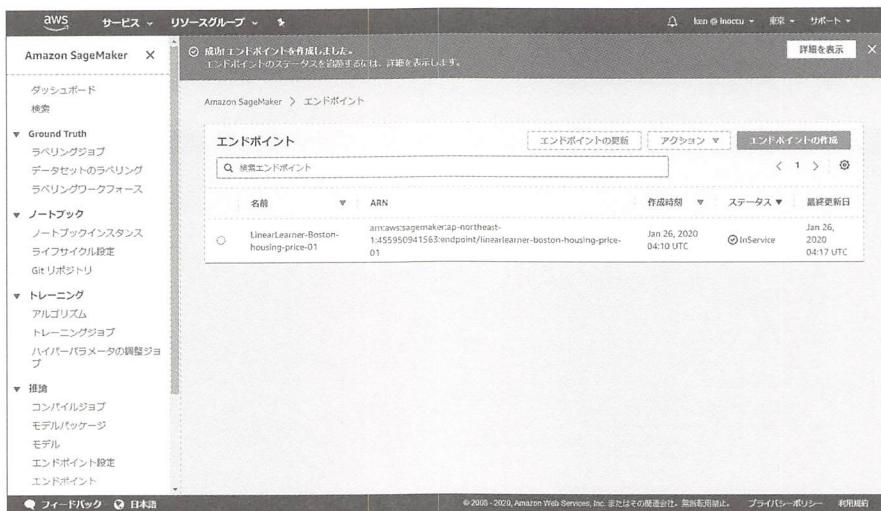


図 4-3-35 エンドポイントの作成完了

4.3.11 エンドポイントの動作確認

作成したエンドポイントの動作を確認してみましょう。再びノートブックに戻って、SageMaker の Python SDK からリアルタイムでの予測処理を実行してみます。ここでは、テストデータの先頭 10 件を使って予測結果と教師データの値を表示します。

```
from sagemaker.predictor import csv_serializer, json_deserializer

predictor = sagemaker.predictor.RealTimePredictor('<エンドポイント名>')

predictor.content_type = 'text/csv'
predictor.serializer = csv_serializer
predictor.deserializer = json_deserializer

for i in range(0, 10):
    result = predictor.predict(X_test[i])
    print(result, y_test[i])
```

```
from sagemaker.predictor import csv_serializer, json_deserializer
predictor = sagemaker.predictor.RealTimePredictor('LinearLearner-Boston-housing-price-01')
predictor.content_type = 'text/csv'
predictor.serializer = csv_serializer
predictor.deserializer = json_deserializer
for i in range(0, 10):
    result = predictor.predict(X_test[i])
    print(result, y_test[i])
{'predictions': [{'score': 1.308034896850586}]} 8.8
{'predictions': [{'score': 16.14173169555664}]} 13.6
{'predictions': [{'score': 34.33338165283203}]} 34.6
{'predictions': [{'score': 19.882999420166016}]} 24.3
{'predictions': [{'score': 31.023574829101562}]} 23.0
{'predictions': [{'score': 26.929702758789062}]} 22.3
{'predictions': [{'score': 42.05268859863281}]} 21.9
{'predictions': [{'score': 19.86287689208984}]} 19.5
{'predictions': [{'score': 18.927959442138672}]} 14.6
{'predictions': [{'score': 12.801763534545898}]} 13.5
```

図 4-3-36 エンドポイントの呼び出しによる予測の実行

1件目は、予測値が 1.30 に対して教師データの値は 8.8 となっていて誤差が大きいですが、2 件目は 16.14 に対して 13.6、3 件目は 34.33 に対して 34.6 と比較的誤差の小さな結果も出ていることがわかります^{*14}。このように、Python SDK 経由でリアルタイムでの予測処理を呼び出せるので、アプリケーションへの組み込みは簡単に行うことができるでしょう。

*14 モデルのトレーニング時に、トレーニングデータとテストデータの分割をランダムに行うため、予測結果が本書の例とは異なることがあります。

4.3.12 エンドポイントの削除

エンドポイントは、実際には使用していないても、使用可能な状態であれば課金の対象になります。そのため、本番運用するエンドポイントでなければ、動作確認が終わった時点で削除しておくと良いでしょう。SageMaker のコンソールでエンドポイントの詳細画面を開き、「削除」ボタンをクリックします。

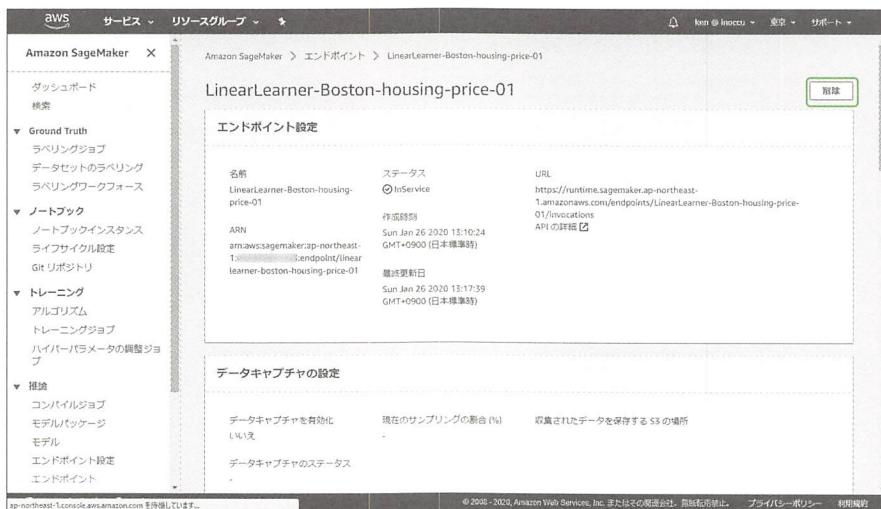


図 4-3-37 エンドポイントの削除

削除確認のダイアログが表示されるので、再度「削除」ボタンをクリックするとエンドポイントが削除されます。

4.3.13 バッチ変換ジョブ

SageMaker 上に作られたモデルは、エンドポイントを作成してリアルタイムに予測処理を行なう方法以外に、バッチ変換ジョブを作成して大量のデータをまとめて予測処理させる方法もあります。後者の方法は、バッチ変換ジョブの実行時にエンドポイントを作成しなくとも良いので、アプリケーションの中でリアルタイムに予測処理を実行する必要がなければ、無駄な課金は発生しません。

バッチ変換ジョブに引き渡すデータは、モデルを作成する際にトレーニングデータとして準備したデータと同じカラムの並びの CSV ファイルです。Pandas を使ってデータセットを CSV ファイルに変換してみましょう。ここでは、先頭行にカラム名の行を出力しないように、

header=False という引数を指定しています。

作成した CSV ファイルを S3 にアップロードします。

```
import pandas as pd
import boto3

df = pd.DataFrame(boston.data)
df.to_csv('boston_data.csv', header=False, index=False)

s3 = boto3.resource('s3')
s3.Bucket('<S3バケット名>').Object('boston_data.csv').upload_file('boston_data.csv')
```

次に、バッチ変換ジョブを作成します。model_name には、「4.3.8 モデルの作成」で指定したモデル名をセットします。

```
import sagemaker

transformer = sagemaker.transformer.Transformer(
    base_transform_job_name='BatchTransformer',
    model_name='<モデル名>',
    instance_count=<インスタンス数 例：1>,
    instance_type='<インスタンスタイプ 例：ml.m5.large>',
    output_path='<結果を出力するS3バケットと префикс>'
)

transformer.transform('<CSVファイルをアップロードしたS3パス>/boston_data.csv',
                     content_type='text/csv', split_type='Line')

# バッチ変換ジョブの完了を待つ場合（任意）
transformer.wait()
```

SageMaker のコンソールからバッチ変換ジョブ画面を見ると、作成したバッチ変換ジョブが表示されており、進捗状況を確認することができます。なお、バッチ変換ジョブはコンソール上で作成することも可能です。



図 4-3-38 作成したバッチ変換ジョブの表示

バッチ変換ジョブが完了すると、output_path で指定した S3 の出力先に、入力ファイル名に「.out」という拡張子の付いたファイルが保存されます。また、入力ファイルの行に合わせて、予測結果が出力されます。

```
boston_data.csv.out
1  [{"score": 30.264657974243164}
2  {"score": 25.314426422119141}
3  {"score": 31.757993598128117}
4  {"score": 29.630949920385747}
5  {"score": 29.287719513125}
6  {"score": 25.549446920654297}
7  {"score": 22.560011890869141}
8  {"score": 19.446836471557617}
9  {"score": 11.1727123166469865}
10 {"score": 18.690139541625976}
11 {"score": 19.298549652999699}
12 {"score": 21.168341215281738}
13 {"score": 20.694501724243164}
14 {"score": 19.856443634093203}
15 {"score": 19.706454766235152}
16 {"score": 19.475778570711914}
17 {"score": 20.934743881255865}
18 {"score": 17.361753463745117}
19 {"score": 16.09559562631105}
20 {"score": 18.386811123057227}
21 {"score": 12.525577273559571}
22 {"score": 17.979886840082831]
23 {"score": 16.541748386274414}
24 {"score": 14.109421985517578}
25 {"score": 16.047496795654297}
26 {"score": 11.4312744140625}
27 {"score": 15.6939697265625}
28 {"score": 15.349975385375}
29 {"score": 20.525012969670707}
30 {"score": 22.672538175854492}
31 {"score": 11.709448013385664}
32 {"score": 18.422193527221687}
33 {"score": 19.766555780328121}
34 {"score": 14.368999481281172}
35 {"score": 14.413316726684577}
36 {"score": 23.379869461689571}
```

図 4-3-39 予測結果ファイルの表示

4.4

SageMaker のさまざまな組み込みアルゴリズム

4.4.1 組み込みアルゴリズムのカタログ

4

前節では、SageMaker の組み込みアルゴリズムを使ってモデルを作成し、エンドポイントとしてデプロイするところまで説明しました。そこでは最も基礎的なアルゴリズムである線形学習(LinearLearner)を使いましたが、他にもさまざまな組み込みアルゴリズムが提供されています。データの特性や何を予測したいかによって、使用すべきアルゴリズムは異なるため、SageMaker を活用するにはその使い分けを行う能力が必要^{*15}になります。

SageMaker の組み込みアルゴリズムは、現時点で 17 種類ありますが、そのすべてが汎用的に用いられるわけではありません。本書では、アルゴリズムを汎用的なものと特定用途のものに分け、このうち汎用的なアルゴリズムについて、代表的なものを 4 つ（線形学習、XGBoost、主成分分析、K-Means）取り上げます。なお、その他については表で簡単に触れます。組み込みアルゴリズムの詳細については、AWS のリファレンス (https://docs.aws.amazon.com/ja_jp/sagemaker/latest/dg/algos.html) を参照してください。

*15 SageMaker の特徴の 1 つとして、組み込みアルゴリズムが存在することが挙げられます。ただし、SageMaker では、組み込みアルゴリズムを使わなくてもモデルを作成し、デプロイすることができます。また、クラウド上のノートブックとデプロイ環境の組み合わせは、他社のクラウドサービスでも提供されています。

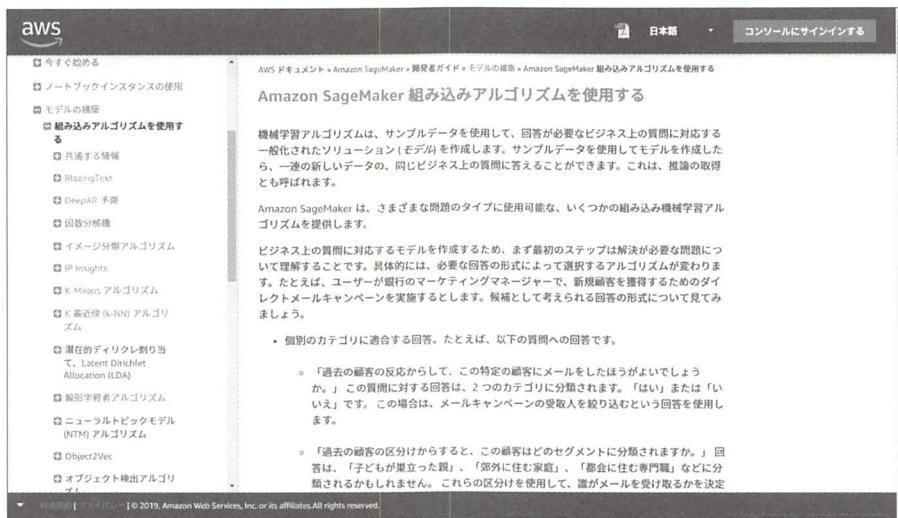


図 4-4-1 組み込みアルゴリズムに関するリファレンス

汎用的に用いられる組み込みアルゴリズムを表 4-4-1 に示します。汎用的なアルゴリズムは、トレーニングに教師データを必要とするか否かで「教師あり学習」と「教師なし学習」の 2 つに分類できます。

さらに、教師あり学習では、アルゴリズムを用いて作成したモデルが予測する値を、カテゴリなどの離散値である「分類」と、金額や気温のような連続値である「回帰」に分けることができます。一方、教師なし学習では、データセットを構成するカラム（次元）から、データの特徴を保ちながら次元を減らして表現する「次元削減」と、データ間の近似の構造を発見してグループ分けする「クラスタリング」という手法があります。

表 4-4-1 のアルゴリズムのうち、線形学習、XGBoost、主成分分析、K-Meansについて、動作例を含めて後で詳しく説明します。それ以外のものは、ここで簡単に説明しておきましょう。まず、因数分解機は、線形学習と同様に分類や回帰が可能なアルゴリズムです。特微量の次元数が多いデータの場合、線形学習では学習の時間が長くかかることがあります、因数分解機は行列因数分解という手法によって計算効率を大きく向上させています。また、K 近傍法は、比較的単純なアルゴリズムで、教師データの中で最も近いデータを見つけ出して分類や回帰を行います。モデルのトレーニングがほとんど不要で、線形学習に向かないデータセットでも使用できるというメリットがある反面、推論に時間がかかるというデメリットもあります。

表 4-4-1 汎用的に用いられる組み込みアルゴリズム

アルゴリズム名	学習	目的	備考
線形学習	教師あり	分類・回帰	分類の場合はロジスティック回帰、回帰の場合は線形回帰が行われます。
XGBoost	教師あり	分類・回帰	—
因数分解機	教師あり	分類・回帰	—
K 近傍法	教師あり	分類・回帰	—
主成分分析	教師なし	次元削減	—
K-Means	教師なし	クラスタリング	—

特定用途に用いられる組み込みアルゴリズムは、表 4-4-2 のとおりです。これらのうち潜在的ディリクレ割り当てとニューラルトピックモデルは、用途が同じです。このように、同じ用途に対して複数の異なるアルゴリズムが提供されています。使用するデータに対してより適した結果を返すアルゴリズムを使うと良いでしょう。

表 4-4-2 特定用途に用いられる組み込みアルゴリズム

アルゴリズム名	学習	説明
イメージ分類	教師あり	イメージ全体を対象に分類を行います。
オブジェクト検出	教師あり	イメージ内のオブジェクトの検出と分類を行います。
セマンティックセグメンテーション	教師あり	イメージ内のすべてのピクセルを分類します。
Sequence to Sequence	教師あり	一般的にニューラル機械翻訳に用いられます。
ランダムカットフォレスト	教師なし	データセット内の異常なポイントを発見します。
潜在的ディリクレ割り当て	教師なし	一連の文書のトピックを決定します。
ニューラルトピックモデル	教師なし	一連の文書のトピックを決定します。
BlazingText	教師なし・ 教師あり	Word2Vec による単語ベクトルの作成(教師なし)と、テキストの分類(教師あり)を行います。
DeepAR 予測	教師あり	再帰型ニューラルネットワークを用いて時系列データを予測します。
IP Insights	教師なし	IPv4 アドレスの使用パターンを学習してモデル化します。
Object2Vec	教師なし	さまざまなオブジェクトのベクトル化を行います。これは Word2Vec を一般化したものです。

4.4.2 線形学習 (LinearLearner)

線形学習アルゴリズムは、線形モデルを用いて分類または回帰を行う教師あり学習のアルゴリズムです。一般的に、線形モデルで分類を行う場合は「ロジスティック回帰」、回帰を行う場合は「線形回帰」というように分けて説明されることが多いのですが、SageMaker の組み込みアルゴリズムでは1つにまとめられており、ハイパーパラメータの設定によって使い分けています。今回は output_path を指定していないので、S3 上に自動作成されたバケットにトレーニング済みのモデルが保存されます。

線形学習アルゴリズムで連続値を予測する方法(線形回帰といいます)は、前節で説明しました。ここでは、分類を行うロジスティック回帰の例を紹介します。データセットとして有名な Iris データセット^{*16}を使用して、花びらのがくと花弁の長さ・幅から、アヤメの品種を分類します。

```
import sagemaker
from sagemaker import get_execution_role
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()
data = iris.data.astype('float32')
target = iris.target.astype('float32')

role = get_execution_role()

linear = sagemaker.LinearLearner(
    role,
    train_instance_count=1,
    train_instance_type='ml.m5.large',
    predictor_type='multiclass_classifier', #多値分類のためmulti_classifier
    num_classes=3 # 3種類に分類するため3
)

X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.3)
train = linear.record_set(X_train, labels=y_train)
test = linear.record_set(X_test, labels=y_test, channel='test')

linear.fit([train, test], mini_batch_size=10)
```

*16 <http://archive.ics.uci.edu/ml/datasets/Iris>

`fit()` 関数のログを見ると、テストデータでの精度 (accuracy) は 0.933 (93.3%)^{*17} となっており、比較的高い精度で分類できていることがわかります。

```
linear.fit([train, test], mini_batch_size=10)
[01/26/2020 04:42:37 INFO 140169918482240] #test_score (algo=1) : ('multiclass_top_k_accuracy_3', 1.0)
[01/26/2020 04:42:37 INFO 140169918482240] #test_score (algo=1) : ('dsc', 0.9753953297932942)
[01/26/2020 04:42:37 INFO 140169918482240] #test_score (algo=1) : ('macro_recall', 0.94110274)
[01/26/2020 04:42:37 INFO 140169918482240] #test_score (algo=1) : ('macro_precision', 0.937037)
[01/26/2020 04:42:37 INFO 140169918482240] #test_score (algo=1) : ('macro_f1_1.000', 0.9394902)
[01/26/2020 04:42:37 INFO 140169918482240] #quality_metric: host=algo=1, test_multiclass_cross_entropy_objective <loss>=0.110401797295
[01/26/2020 04:42:37 INFO 140169918482240] #quality_metric: host=algo=1, test_multiclass_accuracy <score>=0.933333333333
[01/26/2020 04:42:37 INFO 140169918482240] #quality_metric: host=algo=1, test_multiclass_top_k_accuracy_3 <score>=1.0
[01/26/2020 04:42:37 INFO 140169918482240] #quality_metric: host=algo=1, test_dcg <score>=0.975395329793
[01/26/2020 04:42:37 INFO 140169918482240] #quality_metric: host=algo=1, test_macro_recall <score>=0.941102743149
[01/26/2020 04:42:37 INFO 140169918482240] #quality_metric: host=algo=1, test_macro_precision <score>=0.937036991119
[01/26/2020 04:42:37 INFO 140169918482240] #quality_metric: host=algo=1, test_macro_f1_1.000 <score>=0.939490211964
```

図 4-4-2 線形学習アルゴリズムによる分類モデルの精度の確認

4.4.3 XGBoost

XGBoost アルゴリズムは、勾配ブーストツリー^{*18}というアルゴリズムをオープンソース実装した XGBoost を、SageMaker に組み込んだものです。教師あり学習の 1 つであり、ハイパーパラメータの指定により分類、回帰のいずれにも対応します。

XGBoost アルゴリズムは、これまで述べてきた線形学習アルゴリズムとは違って SageMaker Python SDK で直接サポートされていないため、実装方法が少し異なります。前節で扱ったボストン市の住宅価格のデータセットを用いて、回帰モデルを作成してみましょう。

```
import boto3
import sagemaker
from sagemaker.session import s3_input
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
import pandas as pd

boto_session = boto3.session.Session(region_name='ap-northeast-1')

# データの準備
boston = load_boston()
df = pd.DataFrame(boston.data, columns=boston.feature_names)
```

*17 実行の都度、トレーニングデータとテストデータをランダムに分割しているため、結果は毎回変わります。

*18 勾配ブーストツリーは、識別器に決定木を用いたアンサンブル学習手法です。

```

df['PRICE'] = boston.target
df = df.iloc[:, [13,0,1,2,3,4,5,6,7,8,9,10,11,12]] ——①

# トレーニングデータと検証データに分割し、CSVファイルを作成
train, valid = train_test_split(df.values, test_size=0.3)
train_df = pd.DataFrame(train)
valid_df = pd.DataFrame(valid)
train_df.to_csv('boston_train.csv', header=False, index=False) ——②
valid_df.to_csv('boston_valid.csv', header=False, index=False)

# トレーニングデータと検証データをS3にアップロード
b = boto_session.resource('s3').Bucket('<バケット名>')
b.Object('boston_train.csv').upload_file('boston_train.csv')
b.Object('boston_valid.csv').upload_file('boston_valid.csv')

train = s3_input('s3://<バケット名>/boston_train.csv', content_type='text/csv')
valid = s3_input('s3://<バケット名>/boston_valid.csv', content_type='text/csv')

# XGBoostインスタンスの作成
container = '501404015308.dkr.ecr.ap-northeast-1.amazonaws.com/xgboost:latest'
xgboost = sagemaker.estimator.Estimator(
    container,
    role,
    1, #インスタンス数
    'ml.m5.large', #インスタンスタイプ
) ——③

# ハイパーパラメータの設定
xgboost.set_hyperparameters(
    objective='reg:linear', #回帰を行うことを指定
    num_round=25 #トレーニングを実行するラウンド数
)

# トレーニングジョブの作成
xgboost.fit({'train': train, 'validation': valid}) #チャネル名をキーとした辞書型が引数

```

上記のコード中に付した①～③について、以下に説明します。

- ① XGBoost アルゴリズムの入力データは、CSV 形式か LIBSVM 形式のファイルです。今回は CSV 形式のファイルとしますが、そのフォーマットは、最初のカラムが目的変数として扱われます。そのため、boston.data（説明変数）をもとに、Pandas の DataFrame を作成した後で最後の列に boston.target（目的変数）を追加し、さらに列を並べ替えて目的変数を最初のカラムにしています。
- ② Pandas の DataFrame をいったん Numpy の ndarray に変換して、train_test_split() 関数でトレーニングデータと検証データに分割しています。その後で、再び Pandas の DataFrame に変換し、CSV ファイルを出力しています。XGBoost アルゴリズムの入力データには、列名の行を設けない、というルールがあるため、header=False という引数を指定しています。
- ③ SageMaker Python SDK では、XGBoost アルゴリズムが直接はサポートされていないため、第 1 引数 (container) で、アルゴリズムがホストされている Amazon ECR (Elastic Container Registry)^{*19} の Docker^{*20} イメージ名を指定しています。後は概ね線形学習の例と同じですが、ハイパーパラメータについては set_hyperparameters() 関数で指定します。

トレーニングジョブが完了した後は、前節と同様の方法でエンドポイントを作成し、予測を行うことができます。

```

1 from sagemaker.predictor import csv_serializer, json_deserializer
2
3 predictor = sagemaker.predictor.RealTimePredictor('xgboost-boston', sagemaker_session=sagemaker_session)
4
5 predictor.content_type = 'text/csv'
6 predictor.serializer = csv_serializer
7 predictor.deserializer = json_deserializer
8
9 for i in range(0, 10):
10     result = predictor.predict(boston.data[i])
11     print(result, boston.target[i])
12
13 24.5820388794 24.0
14 21.9684619904 21.6
15 34.5131340027 34.7
16 33.3270187378 33.4
17 36.7741775513 36.2
18 28.5402393341 28.7
19 22.0544662476 22.9
20 25.5722026825 27.1
21 16.6258506775 16.5
22 19.2616519928 18.9

```

図 4-4-3 XGBoost で作成したモデルによる予測の実行

*19 Amazon ECR は、Docker イメージを保存、管理できる AWS のサービスです。SageMaker の組み込みアルゴリズムは、Docker イメージとして提供されており、その保存先として Amazon ECR が活用されています。

*20 Docker は、OS レベルの仮想化環境を提供するソフトウェアです。ホスト OS 上で独立して動作する OS レベルの仮想化環境をコンテナといいます。また、コンテナのファイルシステムをひとまとめに保存して、環境を簡単に再現できるようにしたものをイメージといいます。

4.4.4 主成分分析（PCA）

主成分分析（PCA：Principal Component Analysis）アルゴリズムは、データの特徴を損なわずに少ない次元数でデータを表現するための手法です。教師なし学習の一種で、トレーニングデータを使ってモデルのトレーニングを行いますが、教師データは不要です。トレーニング済みのモデルに対してデータを与えると、モデルの作成時に指定した次元数に削減された状態のデータを出力します。次元削減をした後で線形学習アルゴリズムなどを使って回帰モデルを作成すると、精度が高まる場合があります。

これまでと同様に、ボストン市の住宅価格のデータセットを用います。このデータセットの説明変数は13個（13次元）ありますが、以下のコードで3個（3次元）まで削減します。主成分分析アルゴリズムはSageMaker Python SDKで直接サポートされているため、使い方は線形学習アルゴリズムと概ね同様です。algorithm_modeは、regularかrandomizedのいずれかです。データセットの次元数やデータ量が多い場合は、randomizedを指定すると良いでしょう。

```
import sagemaker
from sagemaker import get_execution_role
from sklearn.datasets import load_boston

boston = load_boston()
data = boston.data.astype('float32')

role = get_execution_role()
pca = sagemaker.PCA(
    role,
    train_instance_count=1,
    train_instance_type='ml.m5.large',
    num_components=3,
    algorithm_mode='regular'
)

train = pca.record_set(data)

pca.fit(train, mini_batch_size=10)
```

トレーニング済みのモデルでエンドポイントを作成した後、実際に動作を試すと、13次元のデータが3次元のデータに次元削減されていることがわかります。projectionの値としてセットされているリストを見ると、3つの主成分に射影された値が確認できます。

```

from sagemaker.predictor import csv_serializer, json_deserializer
pca_predictor = sagemaker.predictor.RealTimePredictor('pca-2019-05-04-06-31-15-764', sagemaker_session=sagemaker_session)
pca_predictor.content_type = 'text/csv'
pca_predictor.serializer = csv_serializer
pca_predictor.deserializer = json_deserializer
pca_predictor.predict(data[0])
{'projections': [[{'projection': [3.1726608276367188,
-5.55999755859375,
119.81883239746094]]}}]

```

図 4-4-4 主成分分析による次元削減の結果

4.4.5 K-Means

K-Means アルゴリズムは、データの構造を自動的に発見し、指定した数のグループに分類するための手法です。教師なし学習の一種で、トレーニングデータを使ってモデルのトレーニングを行いますが、教師データは不要です。トレーニング済みのモデルに対してデータを与えると、モデルの作成時に指定した数のグループのうち、どのグループに分類されるかを出力します。

「4.4.2 線形学習（LinearLearner）」で Iris データセットを使い、ロジスティック回帰でアヤメの品種を分類（3 品種のいずれかに分類）しました。本項では同じデータセットを使い、教師データ（目的変数）を渡さずに説明変数だけで 3 つに分類できるかを試してみましょう。

```

import sagemaker
from sklearn.datasets import load_iris

iris = load_iris()
data = iris.data.astype('float32')

role = get_execution_role()
kmeans = sagemaker.KMeans(
    role,
    train_instance_count=1,
    train_instance_type='ml.m5.large',
    k=3, # 分類したいクラス数
)

train = kmeans.record_set(data)

kmeans.fit(train, mini_batch_size=10)

```

トレーニング済みのモデルで分類を行ってみました。closest_clusterは、分類先のグループ($k=3$ なので $0.0 / 1.0 / 2.0$ の3種類が出力されます)を示しています。また、distance_to_clusterは分類されたグループ(クラスター)からの距離を示し、値が小さいほど距離が短いため確信度の高い回答となります。今回の実行結果では、品種0(setosa)については高い精度で分類できていましたが、品種1(versicolor)と品種2(virginica)は混同してしまっているようでした。ハイパーパラメータを調整して、どの程度精度が向上するか試してみたいところです。

```
from sagemaker.predictor import csv_serializer, json_deserializer
kmeans_predictor = sagemaker.predictor.RealTimePredictor('kmeans-2019-05-04-08-12-52-246', sagemaker_session=sagemaker_session)
kmeans_predictor.content_type = 'text/csv'
kmeans_predictor.serializer = csv_serializer
kmeans_predictor.deserializer = json_deserializer

for i in range(0, len(data)):
    print(kmeans_predictor.predict(data[i]), iris.target[i])

[{'predictions': [[{'distance_to_cluster': 0.13361430168151855, 'closest_cluster': 0.0}]] 0
[{'predictions': [[{'distance_to_cluster': 0.4397524893283844, 'closest_cluster': 0.0}]] 0
[{'predictions': [[{'distance_to_cluster': 0.40776365995407104, 'closest_cluster': 0.0}]] 0
[{'predictions': [[{'distance_to_cluster': 0.5243622064590454, 'closest_cluster': 0.0}]] 0
[{'predictions': [[{'distance_to_cluster': 0.18696007132530212, 'closest_cluster': 0.0}]] 0
[{'predictions': [[{'distance_to_cluster': 0.688403844833374, 'closest_cluster': 0.0}]] 0
[{'predictions': [[{'distance_to_cluster': 0.4154421389102936, 'closest_cluster': 0.0}]] 0
[{'predictions': [[{'distance_to_cluster': 0.0722392275929451, 'closest_cluster': 0.0}]] 0
[{'predictions': [[{'distance_to_cluster': 0.8035821914672852, 'closest_cluster': 0.0}]] 0
[{'predictions': [[{'distance_to_cluster': 0.3712890148162842, 'closest_cluster': 0.0}]] 0
[{'predictions': [[{'distance_to_cluster': 0.4852641224861145, 'closest_cluster': 0.0}]] 0
[{'predictions': [[{'distance_to_cluster': 0.26398274302482605, 'closest_cluster': 0.0}]] 0]
```

図 4-4-5 K-Meansによるクラスタリングの結果

COLUMN 公開されているデータセットを使用する

本章では、scikit-learnを使って入手できるBoston house-prices datasetと、Iris datasetを使用しましたが、他にも以下のようなデータセットを入手することができます。

- **Breast cancer wisconsin dataset**

乳がんの腫瘍が良性か悪性かを示すデータセットです。分類に向いています。

- **Diabetes dataset**

糖尿病患者の診断データです。回帰に向いています。

- **Digits dataset**

0～9の数字を手書きしたデータセットです。分類に向いています。

- **Linnerud dataset**

20人の成人男性に対してフィットネスクラブで測定した生理学的特徴と運動能力の関係を示すデータセットです。回帰に向いています。

scikit-learnに限らず、第5章で使用するKerasでも以下のようなデータセットを入手できます。

- **CIFAR10**

10 個のクラスに分類された、60,000 枚のカラー画像のデータセットです。分類に向いています。

- **CIFAR100**

100 個のクラスに分類された、60,000 枚のカラー画像のデータセットです。分類に向いています。

- **IMDB 映画レビュー 感情分類**

25,000 件の映画レビューのデータセットです。肯定／否定の感情でラベル付けされており、分類に向いています。

- **ロイターのニュースワイヤー トピックス分類**

46 個のトピックにラベル付けされた、11,228 個のロイターのニュースワイヤーのデータセットです。分類に向いています。

- **MNIST**

0～9 の数字を手書きしたデータセットです。分類に向いています。

- **Fashion-MNIST**

Tシャツやバッグといったファッショナブルアイテムで 10 個のクラスに分類された、60,000 枚の白黒画像のデータセットです。分類に向いています。

データセットを scikit-learn や Keras といったライブラリを経由して入手すると、概ね、どのデータセットでも同じような方法で操作できるので便利です。また、こうしたデータセットは、UCI (カリフォルニア大学アーバイン校) の UCI Machine Learning Repository (<http://archive.ics.uci.edu/>) や、機械学習に関するさまざまな情報が集まる Kaggle (<https://www.kaggle.com/>) などの Web サイトに集められ、公開されています。scikit-learn や Keras では入手できないデータセットも多数ありますので、このような Web サイトを見てみるのも良いでしょう。

公開されているデータセットは機械学習向けに整備された状態で配布されており、機械学習を学ぶためには便利です。しかし、実務で機械学習を活用する場合は、データを準備するところから自分で始めなければなりません。「データサイエンスや機械学習のプロジェクトは、アルゴリズムに投入できるデータセットを準備するところまでくれば仕事の8割は終わっている」といわれています。ある程度、機械学習に慣れてきたら、自分でデータを準備することにもチャレンジすると良いでしょう。

COLUMN**Amazon SageMaker Neo**

Amazon SageMaker Neoを使うと、SageMaker上でMXNet、TensorFlow、PyTorch、XGBoostといったフレームワークを使用して構築・トレーニングしたモデルを最適化し、精度を損なうことなく最大2倍の速度までパフォーマンスを向上できるとされています。また、モデルを使用した予測時に、フレームワークが必要とするメモリ量を10分の1に低減し、Intel、NVIDIA、ARMといった複数のハードウェアプラットフォームで同じ機械学習モデルを使用することができます。特にIoTなどのエッジ・コンピューティング^{*21}の環境で、機械学習モデルによる予測処理を行う際に有効といえるでしょう。

現在、SageMaker Neoのコードは、Neo-AIプロジェクトの下でオープンソース化されています。

***21** エッジ・コンピューティングとは、ユーザーやユーザーの端末の近くにサーバーを配置し、処理を行うことをいいます。特にIoTで機械学習モデルを活用する場合、AWSなどのクラウドで予測処理を行うと通信遅延が問題になるケースがありますが、エッジ・コンピューティングで機械学習モデルを用いた予測処理が可能になれば、この問題を解決することができます。

4.5

SageMaker Studio と SageMaker Autopilot

4.5.1 SageMaker Studio

4

2019年12月、機械学習の統合開発環境として「Amazon SageMaker Studio（以下、SageMaker Studio）」が発表されました。本書執筆時点では、米国東部（オハイオ）リージョンでプレビュー版として提供されています。

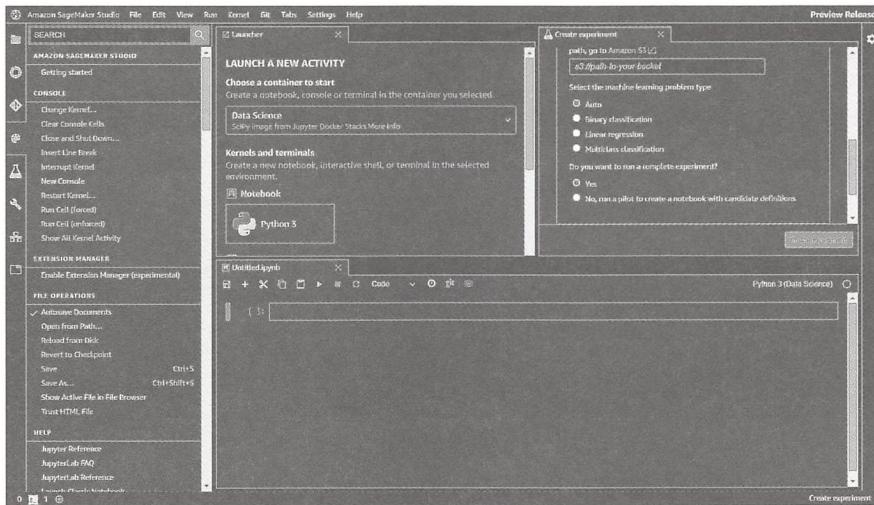


図 4-5-1 SageMaker Studio

SageMaker Studio では、前述したノートブックやトレーニング (Experiment)、モデルの配備といった機能に簡単にアクセスできます。また、モデルの学習時の複雑な処理のデバッグや、モデルの自動評価、デバッグデータの収集などを行い分析できる Debugger、配備したモデルの性能劣化の検知などを行う Model Monitorなどの機能が提供されています。さらに、Experiment では、アルゴリズムの選択、データ前処理、モデルのチューニングなどを自動的に行う Autopilot を使うこともできます。

SageMaker Studio を使用するには、AWS のコンソールの画面右上にあるリージョン選択で「オハイオ」を指定します。すると、SageMaker のコンソール上に Amazon SageMaker Studio というメニューが表示されるので、それをクリックします^{*22}。

^{*22} 本書執筆時点では、SageMaker Studio は Preview 版です。そのため、画面デザインや操作手順などが今後大きく変わることがあります。



図 4-5-2 SageMaker のコンソール (オハイオ)

図 4-5-3 のように、SageMaker Studio の環境を作成する画面が表示されます。環境作成は、Quick start と Standard setup の 2 つの方法がありますが、ここでは Quick start を選択することにしましょう。



図 4-5-3 SageMaker Studio の環境作成

SageMaker Studio を使用するには、User name と、S3 へのアクセス権限、および AmazonSageMaker

FullAccess というポリシーがアタッチされた IAM ロールが必要となります。この画面の Execution role で「新しいロールの作成」を選択して IAM ロールを作成するか、既存の IAM ロールを選択します。新しい IAM ロールの作成時には、図 4-5-4 のように、アクセス権限を付与する S3 バケットを指定して（ここでは「任意の S3 バケット」を選択します）、「ロールの作成」ボタンをクリックすると、図 4-5-3 の画面に戻ります。このようにして Execution role を指定した後、「送信」ボタンをクリックします。



図 4-5-4 新しい IAM ロールの作成

SageMaker Studio の環境が作成されるまで少し時間がかかりますが、ステータスが Ready になったら、使用できます。

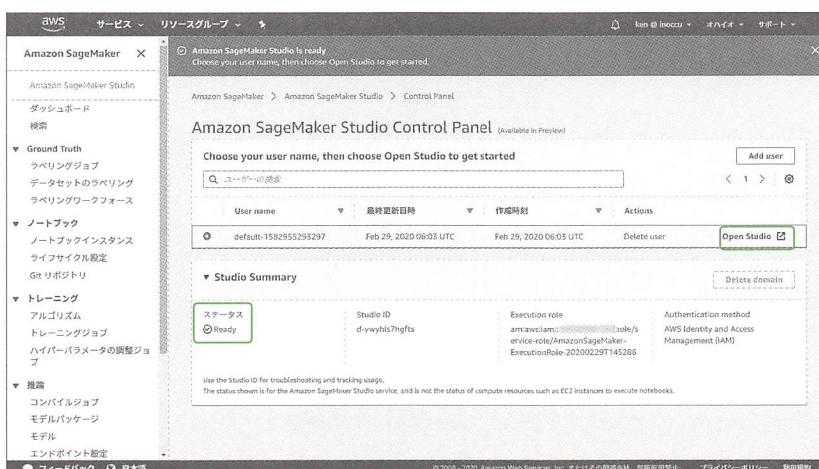


図 4-5-5 SageMaker のサマリー

SageMaker Studio は複数のユーザーを作成して使用することができますが、環境作成時に指定した User name で、最初のユーザーが既にできています。「Open Studio」リンクをクリックすると、SageMaker Studio が開きます。

初回は少し時間がかかりますが、図 4-5-6 のように SageMaker Studio が起動します。画面左にあるバーから、ノートブックの作成や、Git との連携、Experiment やエンドポイントの管理などを行うことができます。

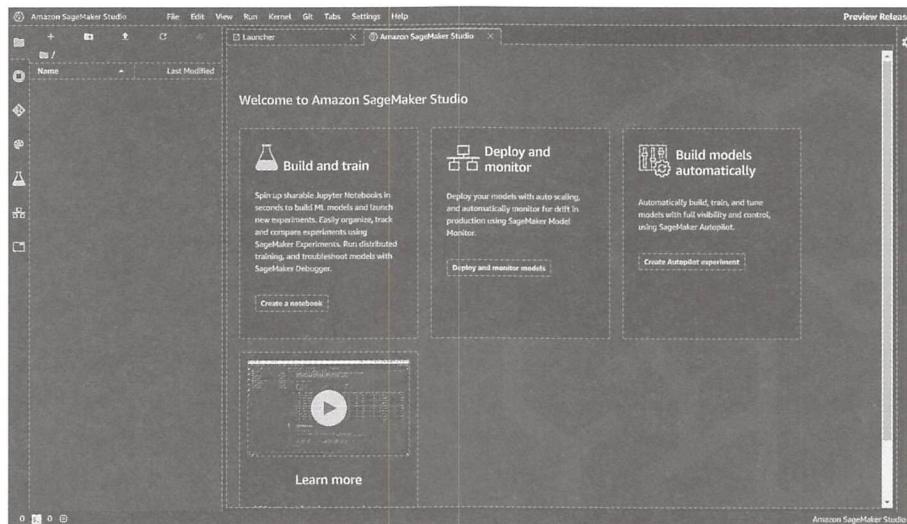


図 4-5-6 起動した SageMaker Studio

4.5.2 SageMaker Autopilot

前項では、機械学習の統合開発環境「SageMaker Studio」について紹介しました。この SageMaker Studio で提供されている特徴的な機能として「Amazon SageMaker Autopilot (以下、SageMaker Autopilot)」があります。SageMaker Autopilot では、入力したデータにもとづいて自動的に機械学習のアルゴリズムやハイパーパラメータの調整が行われ、最適なモデルが作成されます。本項では、SageMaker Autopilot を使用したモデルの自動作成を試してみましょう。

データとしては、scikit-learn を使って入手できる California Housing データセット^{*23}を使用します。このデータセットは、カリフォルニアの国勢調査ブロックグループ（米国国勢調査局が公

^{*23}Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions, Statistics and Probability Letters, 33 (1997) 291-297

開する最小の地理的単位で 600 ~ 3,000 人程度の人口を含む) ごとの収入、住宅の築年数、部屋数、寝室数、人口、世帯数などの情報と住宅価値を示しています。住宅価値を目的変数、それ以外の値を説明変数としてモデルを作成します。

SageMaker Autopilot には CSV 形式のデータをインポートできるので、Notebook 上で以下のコードを実行し、CSV ファイルを作成します。また、SageMaker Autopilot では 1,000 件以上のデータが必要ですが、California Housing データセットには 20,640 件のデータがあるので条件を十分満たしています。

```
from sklearn.datasets import fetch_california_housing
import pandas as pd

housing = fetch_california_housing()
df = pd.DataFrame(housing.data, columns=housing.feature_names)
df['HouseValue'] = housing.target
df.to_csv('housing.csv', index=False)
```

ちなみに、DataFrame の先頭部分を表示して、データの内容を確認しておきましょう。

```
df.head()
```

In [8]:	df.head()
Out [8]:	
	MedInc HouseAge AveRooms AveBedrms Population AveOccup Latitude Longitude HouseValue
0	8.3252 41.0 6.984127 1.023810 322.0 2.555556 37.88 -122.23 4.526
1	8.3014 21.0 6.238137 0.971880 2401.0 2.109842 37.86 -122.22 3.585
2	7.2574 52.0 8.288136 1.073446 496.0 2.802260 37.85 -122.24 3.521
3	5.6431 52.0 5.817352 1.073059 558.0 2.547945 37.85 -122.25 3.413
4	3.8462 52.0 6.281853 1.081081 565.0 2.181467 37.85 -122.25 3.422

図 4-5-7 California Housing データセット

作成した CSV ファイルは、「3.2.8 S3 バケットの作成とファイルのアップロード」を参考に S3 バケットにアップロードしておきます。現時点では、SageMaker Studio は米国東部（オハイオ）リージョンのみで提供されているため、S3 バケットも同じリージョンにしておく必要があります。

SageMaker Studio の画面で EXPERIMENTS タブを開いて、以下のように設定します。

- Experiment Name : 英数字とハイフンで任意の名前を付けます。

- S3 location of input data : S3 バケットにアップロードした CSV ファイルの場所を、s3:// <バケット名> / <ファイルパス> の形式で指定します。
- Target attribute name : 目的変数のカラム名を指定します。今回は HouseValue という列に目的変数をセットしたので、「HouseValue」と指定します。
- S3 location of output data : 実行結果を格納する S3 バケットとパスを指定します。
- Select the machine learning problem type : 作成するモデルが、Binary Classification (二値分類)、Linear Regression (線形回帰)、Multiclass Classification (多値分類) のうち、どれを行うかを選択します。また、データから自動的に指定する Auto を選択することもできます。今回は Auto を選択しましょう。
- Do you want to run a complete experiment : この操作で SageMaker Autopilot の処理をすべて進めてよいので、Yes を選択します。

上記の設定をした後、「Create Experiment」ボタンをクリックすると、モデルの作成が自動的に行われます。

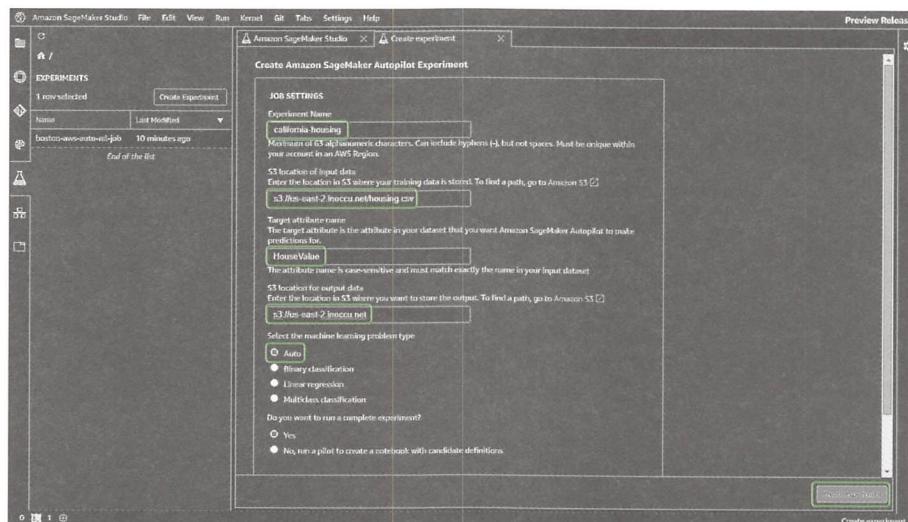


図 4-5-8 Create Experiment 画面

SageMaker Autopilot は、複数のモデルを作成して性能を比較し、モデルのチューニングを進めていくので時間がかかります。今回の例では約 2 時間かかりました。実行中は、図 4-5-9 のように進捗状況を確認できます。

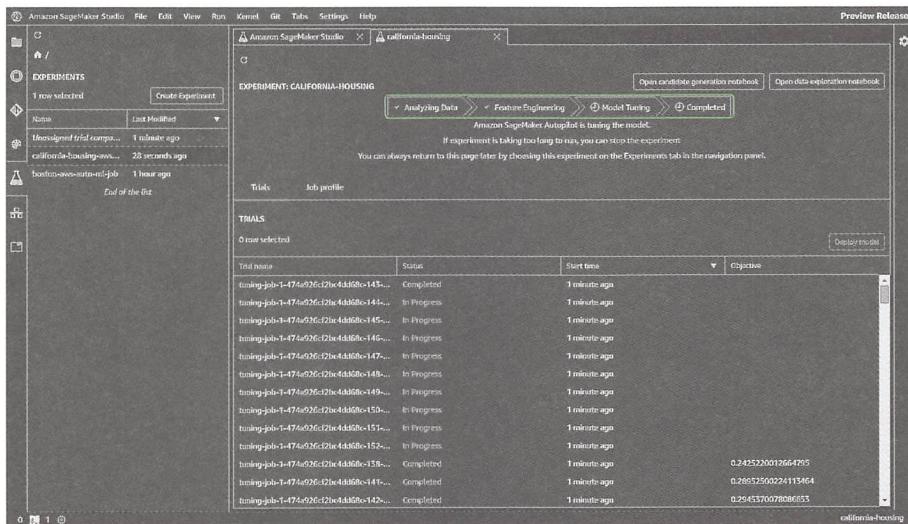


図 4-5-9 Trials 画面 (Model Tuning 実行中)

進捗状況の表示からわかるように、SageMaker Autopilot は、Analyzing Data（データの分析）、Feature Engineering（特徴量設計）、Model Tuning（モデルのチューニング）という順序で処理が進みます。画面右上にある「Open data exploration notebook」ボタンをクリックすると、データの分析結果として、カラムごとの平均値や中央値といった基本統計量を確認することができます。

また、Trials タブを開くと、モデルのトレーニング状況が表示されます。1 つのモデルを作成するためにハイパーパラメータの異なる多くのモデルが作成され、トレーニングが進んでいることがわかります。

SageMaker Autopilot の処理が完了すると、Job profile タブの Status の値が Completed になります。また、Problem type として Regression が表示されており、今回作成したモデルが数値を予測する回帰モデルとして作成されたことがわかります（Problem type は、Analyzing Data の処理が終わった時点で表示されます）。

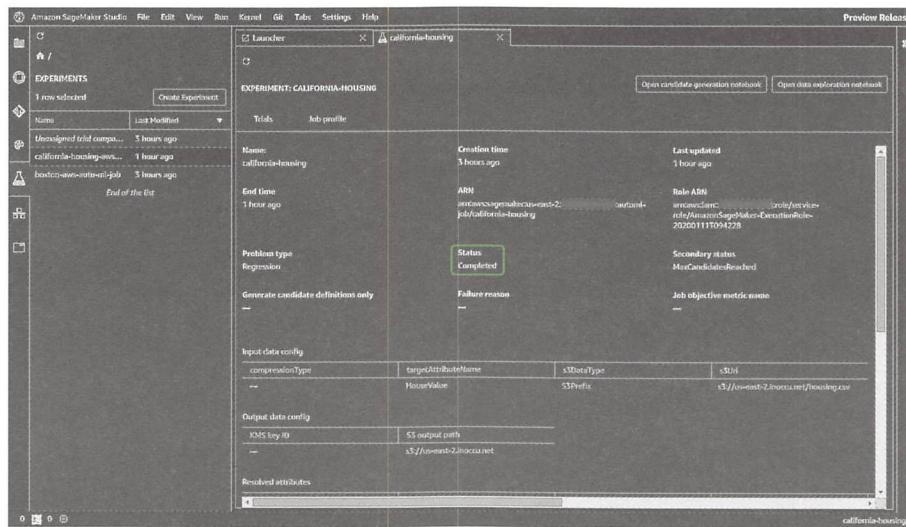


図 4-5-10 Job profile 画面

Status が Completed になった後で、Trials タブを開きます。SageMaker Autopilot ではモデルのトレーニングを何度も行いますが、その1回のトレーニングを Trial といいます。Trials タブには、Trial ごとの予測精度が Objective 列に表示され、比較することができます。最も精度の高い（ここでは、Objective 列の値は誤差を示すので最小のもの）Trial には Best と表示されています。したがって、この試行で作成されたモデルが、最も精度の高いモデルということになります。Trial を選択して画面右の「Deploy model」ボタンをクリックすると、モデルのデプロイを行うことができます。

Trial name	Status	Start time	Objective
* Best: training-job-v1-174a926c72bcAdd... training-job-v1-1-474a926c72bcAdd0x-095... training-job-v1-1-474a926c72bcAdd0x-210... training-job-v1-1-474a926c72bcAdd0x-248... training-job-v1-1-474a926c72bcAdd0x-191... training-job-v1-1-474a926c72bcAdd0x-184... training-job-v1-1-474a926c72bcAdd0x-214... training-job-v1-1-474a926c72bcAdd0x-154... training-job-v1-1-474a926c72bcAdd0x-245... training-job-v1-1-474a926c72bcAdd0x-113... training-job-v1-1-474a926c72bcAdd0x-155... training-job-v1-1-474a926c72bcAdd0x-231... training-job-v1-1-474a926c72bcAdd0x-216... training-job-v1-1-474a926c72bcAdd0x-155... training-job-v1-1-474a926c72bcAdd0x-155... training-job-v1-1-474a926c72bcAdd0x-102... training-job-v1-1-474a926c72bcAdd0x-102...	Completed	2 hours ago	0.19022259549328014

図 4-5-11 Trial 画面 (Autopilot の処理終了後)

Trial 画面で Trial を選択し、右クリックして表示されるメニューで Open in trial details をクリックすると、Trial の詳細情報を表示できます。詳細情報の Parameters タブを開くと、その Trial で指定されたハイパーパラメータを確認できます。今回、最も精度が高かった Trial について表示してみると、SageMaker の組み込みアルゴリズムのうち XGBoost が採用されていることや、ハイパーパラメータの値として何が指定されたかがわかります。

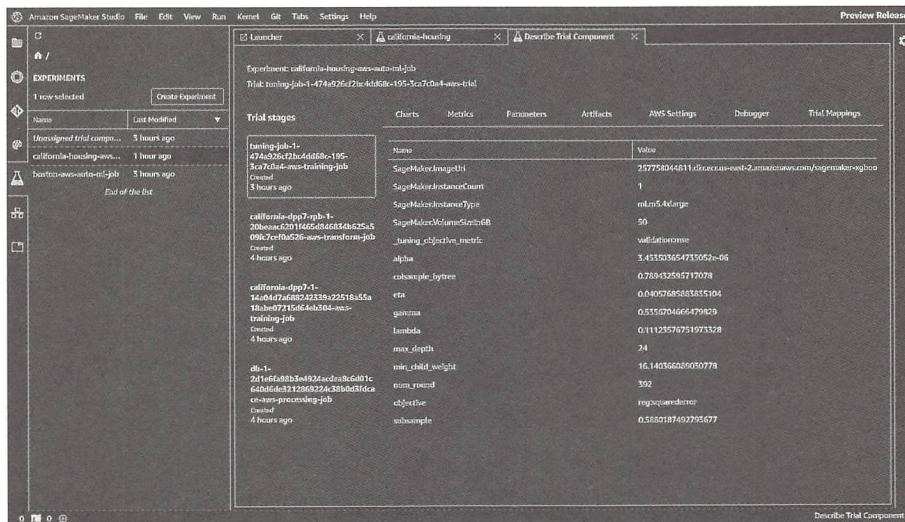


図 4-5-12 Describe Trial Component 画面

精度の高いモデルを作成するためには、本来、さまざまな機械学習アルゴリズムやハイパーパラメータをいろいろと試してみる必要があります。そのためには、時間や手間がかかりますし、機械学習に関する深い知識や経験が必要になるでしょう。しかし、SageMaker Autopilot を使えば、そのような手間が不要になり、また機械学習に関する知識がそれほどなくても、精度の高いモデルを得ることができます。

第 5 章

AWS Deep Learning AMI

独自のモデルを作成し、それを実システムで活用していくには、SageMaker で必要十分でしょう。しかし、先進的なアルゴリズムを率先して使いたい場合など、より柔軟な機械学習環境が必要になるケースでは、AWS Deep Learning AMI (DLAMI) が有用です。

本章では、DLAMI を使って EC2 環境上でディープラーニングモデルを作成する方法を中心説明します。

5.1

EC2環境での ディープラーニング

5.1.1

より柔軟な環境が必要になるケース

第4章では、SageMakerを使用して機械学習モデルを構築しました。SageMakerでは、モデルの構築に必要な環境が完全マネージド型で提供されているので、Jupyterや機械学習ライブラリのインストールなどを行わずに、構築作業をすぐに始めることができました。

モデルの構築にあたって第4章では、主にSageMakerの組み込みアルゴリズムを使用しましたが、scikit-learnやTensorFlowといった広く用いられている機械学習ライブラリを使うことも可能です。また、モデルのトレーニングやデプロイ時に使用するインスタンスは、処理に必要な性能に合ったものを選択することができ、さらにオートスケーリングも可能です。したがって、AWSにおいて機械学習モデルの構築を検討する場合は、簡単に作業を始められ、かつ一定の自由度も兼ね備えているSageMakerが、最初に選択すべきサービスといえるでしょう。

ただ、より柔軟に環境を構築したいケースもあります。例えば、既に手元のPCなどでモデルの開発を進めていて、より高性能な環境（GPUなど）が必要になった場合です。また、機械学習ライブラリとは別に独自のアルゴリズムを開発している場合も同様です。SageMakerでは、トレーニングやテストに用いるデータはS3にアップロードする必要があり、また、独自のアルゴリズムをSageMakerで使用するにはDockerコンテナを作成しなければならないので、手間がかかります。

このようにSageMakerよりも柔軟な環境が求められる場合は、AWS Deep Learning AMI（以下、DLAMI）を利用します。

5.1.2

EC2とAMI

DLAMIに触れる前に、AWSのEC2（Elastic Compute Cloud）とAMI（Amazon Machine Images）について再び説明しておきます。なぜなら、DLAMIはAWSの独立したサービスではなく、EC2というサービスを用いているからです。

EC2は、AWSの最も基本的なサービスであり、スケーラブルなコンピューティング能力として提供されています。具体的にはLinuxサーバーやWindowsサーバーといったコンピュータが、

AWS のデータセンターのサーバー内で仮想的に実現されており、私たちはハードウェアの準備や OS のインストールを行うことなく、あたかも 1 台のコンピュータを占有しているかのように使用することができます。

また、EC2 のコンピュータ性能は、使用目的に応じた「インスタンスタイプ」としてさまざまな種類のものが提供されています。利用者は EC2 の使用開始時に、それらの中から必要なものを適宜選択します。汎用的なインスタンスタイプでは GPU は搭載されておらず、CPU のコア数とメモリ容量に応じて複数の組み合わせがあります。また、ディープラーニング用途としては、GPU が搭載された高速コンピューティング^{*1} のインスタンスタイプが提供されています。

AMI は、EC2 で用いられる暗号化されたマシンイメージです。Linux や Windows などの OS や、さまざまなソフトウェアがコンピュータのルートドライブにインストールされ、環境構築済みの状態でイメージ化されています。AMI は、AWS がデフォルトで提供しているものだけでなく、コミュニティによって提供されているものや、AWS Marketplace で販売されているものなどがあります。さらに、プライベートで使用する AMI を独自に作成することもできます。なお、EC2 を使う場合は、最初に AMI を選択して環境構築が完了した状態にしておきます。

DLAMI は、ディープラーニング用の環境構築が行われた AMI です。EC2 環境上でディープラーニングを行う際に DLAMI を選択すれば、環境構築の手間なく、EC2 上でのディープラーニングの作業にすぐに取り掛かることができます。もちろん、DLAMI ではない一般的な AMI(例えば Amazon Linux や Ubuntu といった Linux ディストリビューションだけがインストールされた AMI) を用いて、自ら機械学習やディープラーニング向けの環境構築を行うことも可能です。

COLUMN Amazon Elastic Inference

Amazon Elastic Inference (以下、Elastic Inference) は、GPU のアクセラレーション (高速化) を自動的に行うための仕組みです。

機械学習では、モデルのトレーニング時に大量のデータを並列で処理するために、高性能なコンピューティング環境が長時間必要となります。しかし、モデルのデプロイ後は、リクエストに応じて推論処理が行われるだけなので、稼働していても実際には使用していない待ち状態の時間が長くなります。EC2 は、実際の使用状況に関わらず稼働時間単位での課金体系となっているので、高額な高速コンピューティングインスタンスを待ち状態のまま稼働させるのは、無駄なコストを発生させるだけです。

Elastic Inference を使用すれば、GPU の性能が必要になった時点で自動的に高速化が行われるため、コストを削減することができます。Elastic Inference は、EC2 だけでなく SageMaker でも使用可能で (現時点では、ディープラーニングのライブラリとして TensorFlow と Apache MXNet を使用した場合のみ使用できます)。

^{*1} GPU が搭載されたインスタンスタイプを「GPU インスタンス」と呼ぶこともあります。

5.1.3 DLAMI と基本 DLAMI

DLAMI には、Amazon Linux、Amazon Linux 2、Ubuntu、Windows のいずれかの OS と、GPU を使用するための環境（CUDA^{*2} と cuDNN^{*3}）、ディープラーニング用のライブラリがあらかじめインストールされています。DLAMI では Conda^{*4} を用いた Python 仮想環境が構築されているので、使用したいライブラリなどで（仮想の）環境を切り替えて、それぞれの環境を独立して運用することができます。

DLAMI で使用できるディープラーニング用のライブラリは、以下のとおりです。

- cuDNN6／CUDA8
Caffe
- cuDNN7／CUDA9
Apache MXNet、Caffe2、Chainer、CNTK、Keras、TensorFlow、Theano
- cuDNN7／CUDA10
PyTorch

AWS では、DLAMI だけでなく基本 DLAMI (AWS Deep Learning Base AMI) も提供されています。これは、CUDA と cuDNN のみがインストールされている AMI です。ディープラーニング用のライブラリを自らインストールしなければなりませんが、環境をより柔軟に構築できるという利点があります。

*2 CUDA (Compute Unified Device Architecture) は、半導体メーカーである NVIDIA 社が開発・提供している GPU 向けの汎用並列コンピューティングプラットフォームであり、C や Python といったプログラミング言語向けにライブラリが提供されています。CUDA を利用することで、NVIDIA 社製の GPU の性能を最大限に引き出すことができます。EC2 の高速コンピューティング向けのインスタンスでは、NVIDIA 社製の GPU が搭載されています。

*3 cuDNN は、CUDA を用いたディープラーニング向け基底ライブラリです。NVIDIA 社製の GPU を TensorFlow や Caffe、Chainer などで使用する際には、CUDA および cuDNN のインストールが必要となります。

*4 Conda は、Python パッケージのインストールや、Python 仮想環境の切り替えを行うための仕組みです。Python のディストリビューションの 1 つである Anaconda に導入されています。

5.2

DLAMIを使う

5.2.1 AMIによるEC2インスタンスの構築

それでは早速、DLAMIを使ってみましょう。前述のように DLAMI は EC2 用の AMI なので、まずは Web 画面上で EC2 のコンソールを開きます。



図 5-2-1 EC2 のコンソール

「インスタンスの作成」をクリックすると、AMI の選択画面が開くので、検索テキストボックスに Deep Learning と入力してみましょう。使用できる DLAMI が一覧表示されます。ここでは、OS に Ubuntu が使用された DLAMI である、「Deep Learning AMI (Ubuntu 18.04)」を選択します^{*5}。

*5 DLAMI は随时バージョンアップしています。本書では、執筆時点のバージョンで説明しています。読者の皆様は、「Deep Learning AMI (Ubuntu)」で表示されたバージョンを選択してみてください（大きくバージョンアップした場合は、コードの実行結果が本書とは異なる可能性があります）。



図 5-2-2 DLAMI の選択

次に、インスタンスタイプを選択します。AWSでは、アカウントを作成してから最初の1年間は無料使用枠があり、EC2では汎用のt2.micro（仮想CPU数1、メモリ1GB）が対象となっています。DLAMIにおいてGPUが搭載されたインスタンス（GPUインスタンス）を使用するか否かは任意です。しかし、GPUが非搭載で、かつ、CPUの仮想CPU数やメモリが少ない低性能（その分、低成本）なインスタンスタイプを選択した場合は、モデルのトレーニングに時間がかかったり、対象のデータを使用する際にメモリ不足で動作が停止したりする可能性があります。そのため、ある程度は性能に余裕のあるインスタンスタイプを選択することをお勧めします。ここでは、GPUは非搭載ですが機械学習向けとされているc5.large（仮想CPU数2、メモリ4GB）を使用することにしましょう^{*6}。

使用するインスタンスタイプを選択し、「確認と作成」ボタンをクリックします。

^{*6} GPUインスタンスを使用する場合は、比較的安価なp2.xlarge（GPU数1、仮想CPU数4、メモリ61GB）がお勧めです。ただし、AWSアカウントの初期状態では、使用できるGPUインスタンスの数が0個に制限されていることがあります。その場合は、<https://aws.amazon.com/contact-us/ec2-request>から、GPUインスタンスを1個まで作成できるようにするなど制限の緩和を依頼する必要があります。

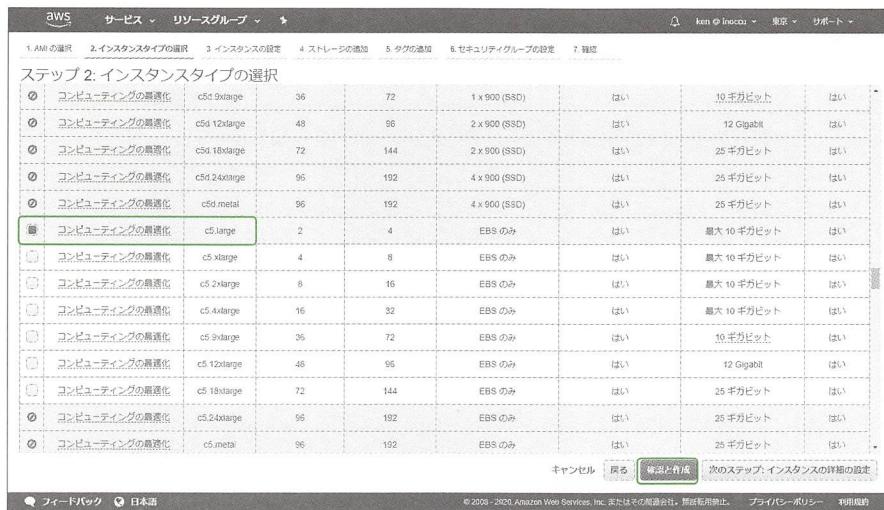


図 5-2-3 インスタンスタイプの選択

インスタンス作成の確認画面が表示されるので、「起動」ボタンをクリックします。



図 5-2-4 インスタンス作成の確認

次に、EC2 インスタンスへのログインに使用するキーペアを選択・作成する画面が表示されます。ここでは「新しいキーペアの作成」を選択し、「キーペア名」に任意のファイル名を指定します(拡張子を指定する必要はありません)。「キーペアのダウンロード」ボタンをクリックすると、拡張子が pem のファイルがダウンロードされます。キーペアのダウンロードが完了すると、「インスタンスの作成」ボタンがクリックできるようになるので、最後にこのボタンをクリックして

インスタンスを作成します。

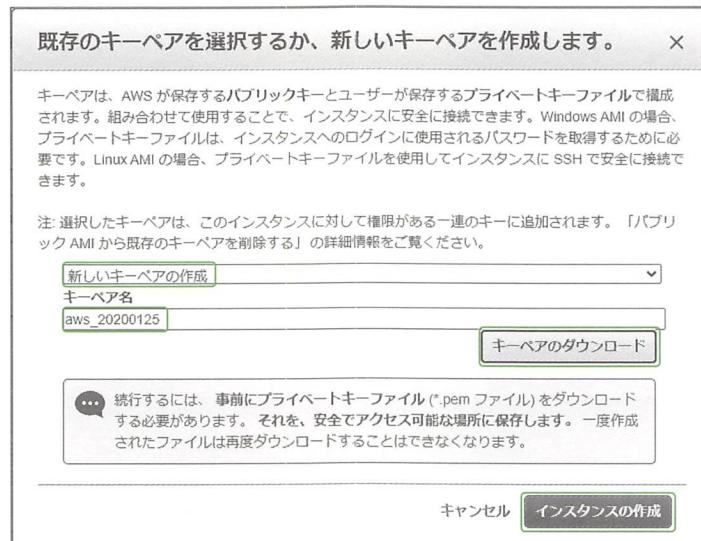


図 5-2-5 キーペアの選択・作成

インスタンスの作成が完了したら、インスタンス ID のリンクをクリックして EC2 のインスタンス一覧画面に戻ります。



図 5-2-6 作成ステータス

作成したインスタンスの状態が「running」になっていれば、そのインスタンスを使用することができます。

インスタンスにログインするには、画面下部に表示されているパブリック DNS の情報が必要なので、確認しておきましょう。



図 5-2-7 インスタンスの状態とパブリック DNS の確認

5.2.2 DLAMI の Jupyter Notebook を開く

前項で作成した DLAMI の EC2 インスタンスは、Ubuntu 環境に SSH ログインしてさまざまな方法で使用することができます。最も一般的な使用方法は、Jupyter Notebook を起動してローカル PC の Web ブラウザから操作し、ディープラーニングのモデルを作成することです。

まず、Ubuntu 環境にログインするには、先ほど確認したパブリック DNS を使用し、以下のコマンドで SSH ログインします。Windows PC では、PuTTy や Tera Term といった SSH に対応したターミナルアプリケーション、Windows 10 April 2018 Update 以降において標準で導入されている SSH コマンドをコマンドプロンプトまたは PowerShell で使用できます。一方、macOS では、標準のターミナルから SSH コマンドを使用できます。

SSH コマンドを使用する場合、先ほどダウンロードしたペアキー（pem ファイル）を用いて以下のようにログインします。

```
mv ~/Desktop/aws_20200125.pem .          ①
chmod 400 aws_20200125.pem                 ②
ssh -L localhost:8888:localhost:8888 -i <pemファイルのパス> ③
ubuntu@<パブリックDNS>
```

まず、①で、ダウンロードした pem ファイルを適当な場所に移動します（任意）。次に、SSH で使用する pem ファイルは読み取り専用のファイルにする必要があるため、②でアクセス属性を変更します。最後に、③でパブリック DNS を指定して EC2 インスタンスにログインします（ユーザー名は、Deep Learning AMI (Ubuntu) の場合は ubuntu）。後で Jupyter Notebook にアクセスするため、EC2 インスタンスの 8888 番ポートをローカル PC の 8888 番ポートでアクセスできるようにトンネルします。なお、初めてログインする場合は、サーバー（EC2 インスタンス）のフィンガープリントの確認のため「Are you sure you want to continue connecting (yes/no)?」というメッセージが表示されるので、「yes」と入力します。

ログインに成功すると、図 5-2-8 のようなメッセージが表示されます。

```

ubuntu@ip-172-31-10-11: ~ % 
inocu@acrophaus:~$ ssh -L localhost:8888:localhost:8888 -i aws_20200125.pem ubuntu@ec2-172-31-10-11.ap-northeast-1.compute.amazonaws.com
=====
 _l_ _l_ )
 _l ( _ / Deep Learning AMI (Ubuntu 18.04) Version 26.0
 _\_\_|_|
=====

Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-1054-aws x86_64v)

Please use one of the following commands to start the required environment with the framework of your choice:
for MXNet(+Keras2) with Python3 (CUDA 10.1 and Intel MKL-DNN) _____ source activate mxnet_p36
for MXNet(+Keras2) with Python2 (CUDA 10.1 and Intel MKL-DNN) _____ source activate mxnet_p27
for MXNet(+AWS Neuron) with Python3 _____ source activate aws_neuron_mxnet_p36
for TensorFlow(+Keras2) with Python3 (CUDA 10.0 and Intel MKL-DNN) _____ source activate tensorflow_p36
for TensorFlow(+Keras2) with Python2 (CUDA 10.0 and Intel MKL-DNN) _____ source activate tensorflow_p27
for Tensorflow(+AWS Neuron) with Python3 _____ source activate aws_neuron_tensorflow_p36
for TensorFlow 2(+Keras2) with Python3 (CUDA 10.0 and Intel MKL-DNN) _____ source activate tensorflow2_p36
for TensorFlow 2(+Keras2) with Python2 (CUDA 10.0 and Intel MKL-DNN) _____ source activate tensorflow2_p27
for PyTorch with Python3 (CUDA 10.1 and Intel MKL) _____ source activate pytorch

```

図 5-2-8 EC2 インスタンスへのログイン

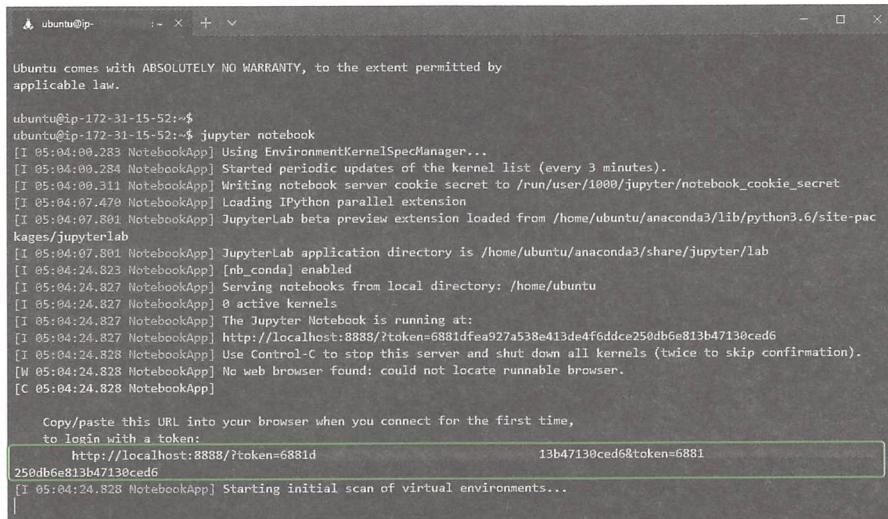
ログインメッセージに続いて、それぞれのディープラーニング用ライブラリを使用する際の、Python 仮想環境の切り替え方法が画面に表示されています。例えば、TensorFlow と Keras を Python3 で使用したい場合は、「source activate tensorflow_p36」というコマンドを実行すれば良いことがわかります。ただし、Jupyter Notebook を使用する場合は、ノートブックの作成時に、使用する仮想環境を指定できるため、ここで仮想環境の切り替えを行う必要はありません。

Jupyter Notebook を使用するには、ログインした SSH 環境で以下のコマンドを実行します。

jupyter notebook

Jupyter Notebook が起動すると、<http://localhost:8888?token=...> という URL が表示されます。この URL にある localhost は EC2 インスタンス自身を指しており、普通は外部から接続す

ることはできません。しかし、SSH 接続時に EC2 インスタンスの 8888 番ポートをローカル PC の 8888 番ポートにトンネルしているので、ローカル PC 上の Web ブラウザでも同様に、`http://localhost:8888?token=...` という URL で接続することができます。なお、この URL に含まれる token の値は、Jupyter Notebook の起動ごとに変わります。そのため、Web ブラウザのお気に入りなどにブックマークできず、起動するたびに値をコピーして使用します。



```

ubuntu@ip-172-31-15-52:~$ jupyter notebook
[I 05:04:00.283 NotebookApp] Using EnvironmentKernelSpecManager...
[I 05:04:00.284 NotebookApp] Started periodic updates of the kernel list (every 3 minutes).
[I 05:04:00.311 NotebookApp] Writing notebook server cookie secret to /run/user/1000/jupyter/notebook_cookie_secret
[I 05:04:07.479 NotebookApp] Loading IPython parallel extension
[I 05:04:07.801 NotebookApp] JupyterLab beta preview extension loaded from /home/ubuntu/anaconda3/lib/python3.6/site-packages/jupyterlab
[I 05:04:07.801 NotebookApp] JupyterLab application directory is /home/ubuntu/anaconda3/share/jupyter/lab
[I 05:04:24.823 NotebookApp] [nb_conda] enabled
[I 05:04:24.827 NotebookApp] Serving notebooks from local directory: /home/ubuntu
[I 05:04:24.827 NotebookApp] 0 active kernels
[I 05:04:24.827 NotebookApp] The Jupyter Notebook is running at:
[I 05:04:24.827 NotebookApp] http://localhost:8888/?token=6881dfea927a538e413de4f6ddce250db6e813b47130ced6
[I 05:04:24.828 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 05:04:24.828 NotebookApp] No web browser found; could not locate runnable browser.
[O 05:04:24.828 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://localhost:8888/?token=6881dfea927a538e413de4f6ddce250db6e813b47130ced6
13b47130ced6&token=6881
[I 05:04:24.828 NotebookApp] Starting initial scan of virtual environments...

```

図 5-2-9 Jupyter Notebook の起動

ローカル PC の Web ブラウザ（ここでは Google Chrome）で、コピーした URL を開きます。すると、Jupyter Notebook の画面が表示されます。この画面右上の「New」のプルダウンメニューを開くと、使用可能なディープラーニング用ライブラリの仮想環境が一覧表示されます。

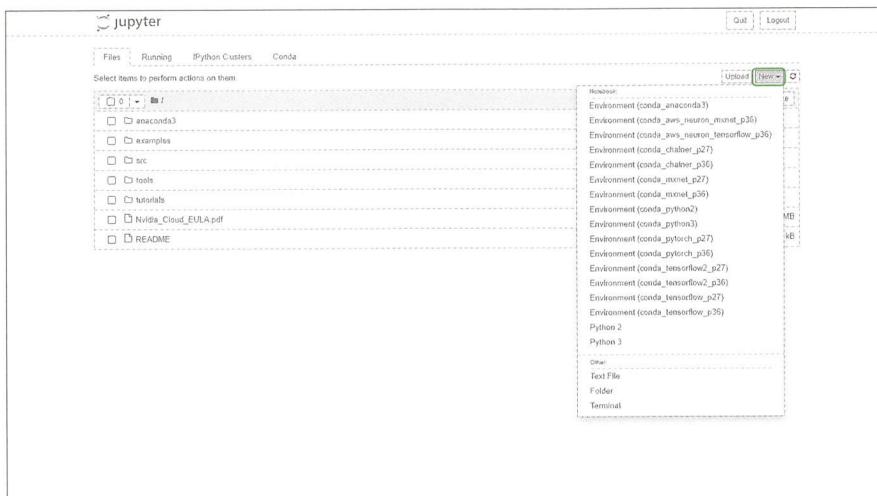


図 5-2-10 DLAMI の Jupyter Notebook

5.2.3 TensorFlow と Keras によるモデル構築

DLAMIで構築したEC2インスタンスで、モデルを構築してみましょう。ここでは、TensorFlowとKerasを用いることにします。前項で開いたJupyter Notebookの画面で新規ノートブックを作成します。画面右上の「New」のプルダウンメニューを開き、「Environment (conda_tensorflow_p36)」を選択します。

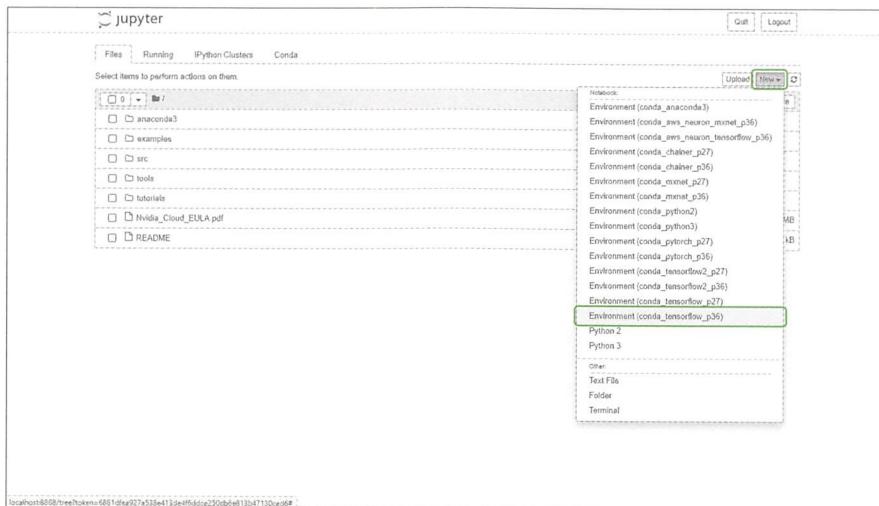


図 5-2-11 ノートブックの作成

ノートブックが開いたら、TensorFlowとKerasのインポートを行い、エラーが出ないことを確認します。

```
import tensorflow as tf
import keras
```

```
In [1]: import tensorflow as tf
        import keras
Using TensorFlow backend.
```

図 5-2-12 TensorFlowとKerasのインポート

ここでは、著名な CIFAR-10 データセット^{*7}を用いてディープラーニングによる分類モデルを作成してみましょう。CIFAR-10 データセットでは、 32×32 ピクセルのカラー画像が60,000枚提供されています。画像は、0:airplane（飛行機）、1:automobile（自動車）、2:bird（鳥）、3:cat（猫）、4:deer（鹿）、5:dog（犬）、6:frog（カエル）、7:horse（馬）、8:ship（船）、9:truck（トラック）の10種類のラベルのいずれかに分類されます。

まず、Keras を用いて、データセットのダウンロードと、ディープラーニングで使用するためデータの変換を行います。

```
from keras.datasets import cifar10
from keras.utils import np_utils

# データセットのダウンロード
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# One-Hot形式に変換
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)

# 標準化
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255
X_test = X_test / 255
```

In [2]:

```
from keras.datasets import cifar10
from keras.utils import np_utils

# データセットのダウンロード
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# One-Hot形式に変換
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)

# 標準化
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255
X_test = X_test / 255
```

Downloading data from <https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz>
170500096/170498071 [=====] - 22s 0us/step

図 5-2-13 データセットのダウンロードと変換

*7 Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.
<http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>

次に、モデルの構築を行います。畳み込みを2回行った後で全結合層を2層だけ接続した簡単なモデルです（畳み込みおよび全結合層については、第1章を参照してください）。

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D

model = Sequential()

# 1回目の畳み込みとプーリング
model.add(Conv2D(32, (3, 3), padding='same', input_shape=X_train.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 2回目の畳み込みとプーリング
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 全結合層
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

それでは、モデルのトレーニングを行います。今回はエポック数を10にしました。モデルの精度を上げるためにもう少し多めにしたいところですが、動作の様子を確認するには十分でしょう。

```
model.fit(X_train, y_train, batch_size=32, epochs=10, verbose=1)
```

```
In [4]: model.fit(X_train, y_train, batch_size=32, epochs=10, verbose=1)
WARNING:tensorflow:From /home/ubuntu/anaconda3/envs/tensorflow_p36/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.py:3066: to_in
t32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/10
50000/50000 [=====] - 56s 1ms/step - loss: 1.3244 - acc: 0.5287
Epoch 2/10
50000/50000 [=====] - 55s 1ms/step - loss: 0.9496 - acc: 0.6667
Epoch 3/10
50000/50000 [=====] - 54s 1ms/step - loss: 0.7624 - acc: 0.7319
Epoch 4/10
50000/50000 [=====] - 54s 1ms/step - loss: 0.6084 - acc: 0.7868
Epoch 5/10
50000/50000 [=====] - 55s 1ms/step - loss: 0.4703 - acc: 0.8367
Epoch 6/10
50000/50000 [=====] - 54s 1ms/step - loss: 0.3432 - acc: 0.8827
Epoch 7/10
50000/50000 [=====] - 54s 1ms/step - loss: 0.2364 - acc: 0.9178
Epoch 8/10
50000/50000 [=====] - 55s 1ms/step - loss: 0.1646 - acc: 0.9440
Epoch 9/10
50000/50000 [=====] - 54s 1ms/step - loss: 0.1216 - acc: 0.9591
Epoch 10/10
50000/50000 [=====] - 54s 1ms/step - loss: 0.1074 - acc: 0.9630
Out[4]: <keras.callbacks.History at 0x7f84d6312d30>
```

図 5-2-14 トレーニングの実行

モデルのトレーニングは1エポックあたり約1分かかり、10分程度で終わりました。

以下のコードを実行して、精度を表示してみましょう。

```
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Accuracy', '{:.2f}'.format(accuracy))
```

```
In [5]: loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Accuracy', '{:.2f}'.format(accuracy))
Accuracy 0.70
```

図 5-2-15 精度の表示

精度^{*8}として0.70と表示されました。これは、70%の確率で正答することができたことを示しています。もう少し精度を高めたいところですが、モデルのネットワーク構造を変えたり、トレーニング時のエポック数を増やしたり試行錯誤が必要なため、ここでは先に進めることにしましょう。

最後にモデルを保存します。lsコマンドを実行して、きちんと保存されたかを確認してみます。

```
model.save('cifar10-cnn.h5')
!ls -l
```

*8 ここで使用した精度指標であるAccuracy（正解率）は、予測結果全体における正答の割合を示しています。

```
In [6]: model.save('cifar10-cnn.h5')
!ls -l

total 21964
-rw-rw-r-- 1 ubuntu ubuntu 2839191 Nov 21 10:20 Nvidia_Cloud_EULA.pdf
-rw-rw-r-- 1 ubuntu ubuntu 2842 Nov 27 10:08 README
-rw-rw-r-- 1 ubuntu ubuntu 8887 Jan 25 05:28 Untitled.ipynb
drwxrwxr-x 24 ubuntu ubuntu 4096 Nov 21 12:50 anaconda3
-rw-rw-r-- 1 ubuntu ubuntu 19607720 Jan 25 05:29 cifar10-cnn.h5
drwxrwxr-x 9 ubuntu ubuntu 4096 Nov 21 12:28 examples
drwxrwxr-x 4 ubuntu ubuntu 4096 Nov 27 10:08 src
drwxrwxr-x 3 ubuntu ubuntu 4096 Nov 21 10:20 tools
drwxrwxr-x 5 ubuntu ubuntu 4096 Nov 21 12:27 tutorials
```

図 5-2-16 モデルの保存

5.2.4 構築したモデルのデプロイ

トレーニングが完了したモデルをデプロイします。DLAMI には TensorFlow Serving がインストールされています。TensorFlow Serving により、サーバー上にトレーニング済みのモデルを配置し、クライアントから送信されたデータを用いて推論した結果を返すことができます。ここでは、モデルの作成で使用した EC2 インスタンス上で TensorFlow Serving を動作させ、作成したモデルをデプロイしてみましょう。

前項までの作業で、Keras を用いて作成したトレーニング済みモデルが EC2 インスタンス上に保存されています。このモデルを以下のコードにより、TensorFlow Serving でデプロイできる SavedModel 形式に変換します。

```
from keras.models import load_model
from tensorflow.python.estimator.export import export

keras_model_path = 'cifar10-cnn.h5'
model = load_model(keras_model_path) —————①
estimator = tf.keras.estimator.model_to_estimator(keras_model_path=keras_model_path, ↴
model_dir='./') —————②

feature_spec = {'conv2d_1_input': model.input} —————③
serving_input_fn = export.build_raw_serving_input_receiver_fn(feature_spec) —————④
estimator._model_dir = './keras' —————⑤
estimator.export_savedmodel('cifar10-cnn', serving_input_fn) —————⑥
```

まず、①のコードで、保存された Keras のモデルをロードし、②で TensorFlow の Estimator

に変換します。③と④は、デプロイ後のモデルを呼び出す際にセットするパラメータ名と型の指定です。⑤は、②の処理で保存されるモデルの保存先ディレクトリを指定しています（これは常に「./keras」です）。最後に、⑥で SavedModel 形式に変換して保存します。この第 1 引数がモデルの保存先ディレクトリとなります。実行結果は図 5-2-17 のとおりです。

```
In [6]: from keras.models import load_model
from tensorflow.python.estimator.export import export

keras_model_path = 'cifar10-cnn.h5'
model = load_model(keras_model_path)
estimator = tf.keras.estimator.model_to_estimator(keras_model_path=keras_model_path, model_dir='./')

feature_spec = {'conv2d_1_input': model.input}
serving_input_fn = export.build_raw_serving_input_receiver_fn(feature_spec)
estimator._model_dir = './keras'
estimator.export_savedmodel('cifar10-cnn', serving_input_fn)

INFO:tensorflow:Using default config.
INFO:tensorflow:Loading models from cifar10-cnn.h5
INFO:tensorflow:Using config: {'_model_dir': './', '_tf_random_seed': None, '_save_summary_steps': 100, '_save_checkpoints_steps': None, '_save_checkpoints_secs': 600, '_session_config': allow_soft_placement: true
graph_options {
    rewrite_options {
        meta_optimizer_iterations: ONE
    }
}, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps': 100, '_train_distribute': None, '_device_fn': None, '_protocol': None, '_eval_distribute': None, '_experimental_distribute': None, '_service': None, '_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec object at 0x7f0a621415f8>, '_task_type': 'worker', '_task_id': 0, '_global_id_in_cluster': 0, '_master': '', '_evaluation_master': '', '_is_chief': True, '_num_ps_replicas': 0, '_num_worker_replicas': 1}
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Signatures INCLUDED in export for Classify: None
INFO:tensorflow:Signatures INCLUDED in export for Regress: None
INFO:tensorflow:Signatures INCLUDED in export for Predict: ['serving_default']
INFO:tensorflow:Signatures INCLUDED in export for Train: None
INFO:tensorflow:Signatures INCLUDED in export for Eval: None
INFO:tensorflow:Restoring parameters from ./keras/keras_model.ckpt
INFO:tensorflow:Assets added to graph.
INFO:tensorflow:No assets to write.
INFO:tensorflow:SavedModel written to: cifar10-cnn/temp-b'1559480647'/saved_model.pb

Out[6]: b'cifar10-cnn/1559480647'
```

図 5-2-17 SavedModel 形式に変換

次に、TensorFlow Serving のサーバーを、DLAMI の EC2 インスタンス上で起動します。ローカル PC でもう 1 つターミナルを起動し、以下のコマンドで SSH 接続します。

```
ssh -i <pemファイルのパス> ubuntu@<パブリックDNS>
```

EC2 インスタンスにログインしたら、以下のコマンドを実行します。

```
tensorflow_model_server --port=9000 --model_name=cifar10-cnn --model_base_path=/home/ubuntu/cifar10-cnn
```

ここで実行する tensorflow_model_server が、サーバーを起動するコマンドです。ここではポート番号として 9000、モデル名 (model_name) として「cifar10-cnn」、SavedModel 形式のモデルの保存先ディレクトリ (model_base_path) としてフルパスで「/home/ubuntu/cifar10-cnn」を指定しました。

```

ubuntu@ip-172-31-10-10: ~ % ubuntu@ip-172-31-10-10: ~ %
ififar10-cnn version: 1579930245}
2020-01-25 05:33:07.873878: I tensorflow_serving/core/loader_harness.cc:74] Loading servable version {name: cifar10-cnn
version: 1579930245}
2020-01-25 05:33:07.873899: I external/org_tensorflow/tensorflow/contrib/session_bundle/bundle_shim.cc:363] Attempting to load native SavedModelBundle in bundle-shim from: /home/ubuntu/cifar10-cnn/1579930245
2020-01-25 05:33:07.873919: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:31] Reading SavedModel from: /home/ubuntu/cifar10-cnn/1579930245
2020-01-25 05:33:07.875023: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:54] Reading meta graph with tags { serve }
2020-01-25 05:33:07.936034: I external/org_tensorflow/tensorflow/core/platform/cpu_feature_guard.cc:145] This TensorFlow binary is optimized with Intel(R) MKL-DNN to use the following CPU instructions in performance critical operations: AVX512F
To enable them in non-MKL-DNN operations, rebuild TensorFlow with the appropriate compiler flags.
2020-01-25 05:33:07.937770: I external/org_tensorflow/tensorflow/core/common_runtime/process_util.cc:115] Creating new thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads for best performance.
2020-01-25 05:33:08.150037: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:202] Restoring SavedModel bundle.
2020-01-25 05:33:08.982555: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:151] Running initialization op on SavedModel bundle at path: /home/ubuntu/cifar10-cnn/1579930245
2020-01-25 05:33:08.984928: I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:311] SavedModel load for tags { serve }; Status: success. Took 1111004 microseconds.
2020-01-25 05:33:08.985216: I tensorflow_serving/servables/tensorflow/saved_model_warmup.cc:105] No warmup data file found at /home/ubuntu/cifar10-cnn/1579930245/assets.extra/tf_serving_warmup_requests
2020-01-25 05:33:08.985377: I tensorflow_serving/core/loader_harness.cc:87] Successfully loaded servable version {name: cifar10-cnn
version: 1579930245}
2020-01-25 05:33:08.988850: I tensorflow_serving/model_servers/server.cc:353] [Running gRPC ModelServer at 0.0.0.0:9000].
[...]

```

図 5-2-18 TensorFlow Serving サーバーの起動

サーバーの起動に成功すると、最後に「Running gRPC ModelServer at 0.0.0.0:9000」のようなメッセージが表示されます。これでデプロイは完了です。

それでは、デプロイされたCIFAR-10のトレーニング済みモデルを呼び出して、推論を実行してみましょう。TensorFlow Serving のサーバーは9000番ポートで待ち受け状態ですが、EC2インスタンスの9000番ポートへの外部からのアクセスを許容していないため^{*9}、ここではDLAMI上で起動しているJupyter Notebook（先ほどまでモデルの作成作業を行っていたノートブック）を用いて呼び出し処理を実行してみます。

ノートブックで以下のようなコードを実行します。

```

import grpc -----①
import tensorflow as tf
from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_service_pb2_grpc

channel = grpc.insecure_channel('localhost:9000') -----②
stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)

request = predict_pb2.PredictRequest() -----③

```

^{*9} EC2のコンソールで、外部から9000番ポートへのアクセスを許容するようにセキュリティグループの設定を行えば、外部からの接続が可能になります。

```
request.model_spec.name = 'cifar10-cnn' ——④  
feature = X_test[0].reshape(1, 32, 32, 3) ——⑤  
request.inputs['conv2d_1_input'].CopyFrom(tf.contrib.util.make_tensor_proto(feature)) —⑥  
  
result = stub.Predict(request, 10.0) ——⑦  
print(result)
```

TensorFlow Serving では gRPC^{*10} を用いて通信するため、①でインポートを行います。接続先サーバーの指定は②で行います。ここでは上記のとおり、Jupyter Notebook が動作している EC2 インスタンス上でサーバーも起動しているので、「localhost」の 9000 番ポートを指定します。また、サーバーへのリクエストに用いるオブジェクトを③で作成します。④は、使用するモデル名の指定です。これは、サーバーを起動する際に指定した model_name と同じ値を指定します。⑤は、推論を行うデータの形状を変更しています。データとしては、モデルのトレーニング時のテストデータ（X_test）の最初の要素（X_test[0]）を使用します。X_test[0] は (32, 32, 3) の 3 次元のリストですが、モデルは (?, 32, 32, 3) という 4 次元のリストを必要としているため、reshapeを行っています。形状を変更したデータは⑥でリクエストオブジェクトにセットします。最後に、⑦でリクエストを実行します。第 2 引数の「10.0」はタイムアウト（秒）の指定です。

実行結果は図 5-2-19 のようになりました。

*10 gRPC は Google が開発した RPC (Remote Procedure Call) プロトコルです。

```
In [22]: import grpc
import tensorflow as tf
from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_service_pb2_grpc

channel = grpc.insecure_channel('localhost:9000')
stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)

request = predict_pb2.PredictRequest()
request.model_spec.name = 'cifar10-cnn'
feature = X_test[0].reshape(1, 32, 32, 3)
request.inputs['conv2d_1_input'].CopyFrom(tf.contrib.util.make_tensor_proto(feature))

result = stub.Predict(request, 10.0)
print(result)

outputs {
    key: "activation_4"
    value {
        dtype: DT_FLOAT
        tensor_shape {
            dim {
                size: 1
            }
            dim {
                size: 10
            }
        }
        float_val: 5.974068795211451e-09
        float_val: 8.015073049705279e-09
        float_val: 1.0526325695536798e-06
        float_val: 0.9757429957389832
        float_val: 1.4294363381850417e-07
        float_val: 0.024252979084849358
        float_val: 2.916763378379983e-06
        float_val: 1.5309710832411838e-08
        float_val: 4.299432809773407e-09
        float_val: 2.6766258542920696e-08
    }
}
```

図 5-2-19 TensorFlow Serving を用いた推論の実行結果

表示された戻り値の内容を見ると、outputs の中に float_val の値が 10 個並んでいます。これが推論（分類）を行った結果です。上から 4 番目の値（ラベル名は 0 から始まるので、ラベルは 3）が最も大きな値となっているので、「猫の画像である」と推論されたことになります^{*11}。

この推論が正しいかどうかは、X_test[0] のデータに対応する教師データ y_test[0] を表示するとわかります。

```
print(y_test[0])
```

In [23]:	print(y_test[0])
	[0. 0. 0. 1. 0. 0. 0. 0. 0.]

図 5-2-20 該当する教師データの表示

*11 P235 で述べたように、画像は、0 : airplane(飛行機)、1 : automobile(自動車)、2 : bird(鳥)、3 : cat(猫)、4 : deer(鹿)、5 : dog(犬)、6 : frog(カエル)、7 : horse(馬)、8 : ship(船)、9 : truck(トラック) の 10 種類のラベルのいずれかに分類されます。

4番目の値が「1」になっているので、推論どおり、猫が正答です。このデータについては、きちんと分類できたようです。

5.2.5 EC2 インスタンスの停止または終了

EC2 インスタンスは、特に使用していない起動時間に応じて課金されます。そのため、使用していないインスタンスは停止（stopped）または終了（terminated）して、課金が発生しない状態にしておきます。

インスタンスの停止または終了は、EC2 のダッシュボードからインスタンスの一覧を表示して操作します。まず、停止または終了したいインスタンスを選択します。次に、「アクション」のプルダウンメニューを開いて、「インスタンスの状態」で「停止」または「終了」をクリックします。

停止したインスタンスは、「インスタンスの状態」で「開始」をクリックすると、停止直前の状態から再開できます。一方、終了したインスタンスは再開できず、再び AMI を指定して EC2 インスタンスを作成する必要があるので注意してください。

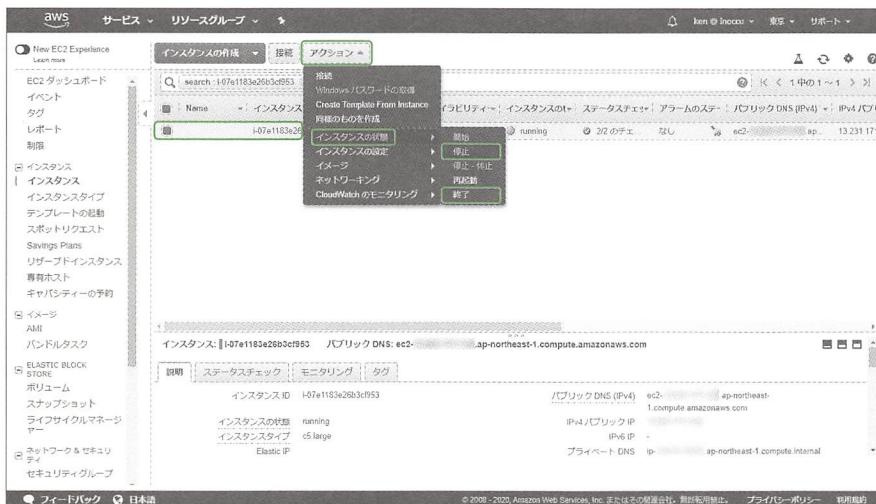


図 5-2-21 インスタンスの停止または終了

索引

A

Absolute Loss	185
Accuracy.....	237
AI.....	12
AI サービス.....	54
Algorithm metrics.....	137
Amazon AI.....	16, 34
Amazon Athena	38, 44
Amazon CloudWatch.....	186
Amazon CodeGuru.....	57
Amazon Comprehend.....	56, 79
Amazon Comprehend Medical.....	56
Amazon DynamoDB.....	38
Amazon EC2 (Elastic Compute Cloud)	37, 224, 227
Amazon ECR (Elastic Container Registry)	207
Amazon Elastic Inference	225
Amazon Forecast	45, 56, 117, 119
Amazon Fraud Detector	57
Amazon Kendra	57
Amazon Kinesis Data Streams	43
Amazon Lex.....	56, 103
Amazon Machine Images (AMI)	225, 227
Amazon Machine Learning (Amazon ML)	36
Amazon Personalize	56, 139
Amazon Polly	56, 99
Amazon QuickSight	38
Amazon Redshift.....	38, 44
Amazon Rekognition	25, 45, 56, 73
Amazon S3.....	38, 70, 162
Amazon SageMaker.....	37, 45, 158
Amazon SageMaker Autopilot	216
Amazon SageMaker Neo	212
Amazon SageMaker Studio	213
Amazon Textract	56, 83
Amazon Transcribe	56, 93
Amazon Translate	56, 87
Anaconda.....	58, 60
Apache MXNet.....	174
Apache Spark	174
AWS Data Pipeline.....	43
AWS Deep Learning AMI.....	37, 45, 226, 227
AWS Deep Learning Base AMI.....	226

AWS IoT Core	43
AWS Lambda.....	104
AWS SDK.....	58
AWS SDK for Python	68

B

BlazingText	203
Bot	105
Boto3.....	68
BoW	41

C

Chainer	169, 174
CIFAR-10 データセット	235
Classification	20
Conda.....	226
CUDA	226
cuDNN	226
Custom Labels	78

D

DeepAR 予測	203
DLAMI	226, 227
Doc2Vec	41
Docker	207

E

EC2	37, 224, 227
EC2 インスタンス	227
ENIAC	13

F

fit 関数	184
Forecast	130
Fulfillment	106

G

Google Cloud Platform	36
GPU インスタンス	225, 228
Ground Truth	160
gRPC	241

I	
IAM	61, 162
IAM ロール	162, 180, 215
IBM Watson	17, 32, 35
Intent	105
IP Insights	203
Item	140
ITEM_METADATA	118
J	
Jupyter Notebook ...	58, 60, 68, 161, 169, 183, 231
K	
K-Means	209
Keras	234
K 近傍法	202
L	
LexModelBuildingService	104
LexRuntimeService	104, 112
LinearLearner	175, 180, 204
M	
Microsoft Azure	55
Microsoft Azure Cognitive Services	17, 32, 35
Microsoft Azure Machine Learning (Azure ML)	17, 31
MSE	185
N	
NLU (Natural Language Understanding)	103
Notebook	37, 44
Numpy	169
O	
Object2Vec	203
P	
Pandas	169, 176
PCA (Principal Component Analysis)	208
Predictor	127
Predictor overview	137
PyTorch	169, 174
R	
RecordSet	182, 184
Regression	20
S	
SageMaker	37, 45, 158
SageMaker Autopilot	216
SageMaker Studio	213
scikit-learn	158, 169, 174
Sequence to Sequence	203
Slot	106
Sony Neural Network Console	31
SparkML Serving	174
T	
TARGET_TIME_SERIES	117
TensorFlow	158, 168, 174, 234
TensorFlow Serving	238
U	
User	140
User-item interaction	140
Utterance	105
X	
XGBoost	205
あ	
アカウントの作成	47
アルゴリズム	18, 175, 201
い	
異常値	44
イベント	148
イメージ分類	203
因数分解機	202
う	
ウエイト	26
え	
エキスパートシステム	14
エッジ・コンピューティング	212
エンティティの抽出	81
エンドポイント	191, 193, 196, 198
お	
オブジェクト検出	203

重み	26
音声認識	94
オンライン学習	184

か

回帰	20
顔認識	76
学習	161
学習（トレーニング）フェーズ	18
隠れ層	26
カスタム用語	87
画像認識	16, 74

き

機械学習	16, 30
基本 DLAMI	226
キャンペーン	152
教師あり学習	202
教師データ	16
教師なし学習	202

く

組み込みアルゴリズム	172, 175, 201
------------	---------------

け

欠損値	44
権限の付与	61, 66
検証データ	179

こ

勾配ブーストツリー	205
コグニティブサービス	35, 55
コンソール	172

さ

散布図行列	177
-------	-----

し

閾値	26
自然言語の識別	80
自然言語理解	103
主成分分析	208
出力層	26, 27
人工知能	12
人工知能グーム	12

す

推論	14, 162
推論フェーズ	18
スマートスピーカー	12

せ

正解率	237
精度	237
絶対値誤差	185
セマンティックセグメンテーション	203
線形回帰	22
線形学習	180, 204
全結合層	27, 236
潜在的ディリクレ割り当て	203

そ

ソリューション	149
ソリューションバージョン	149

た

畳み込み	236
畳み込み層	27
畳み込みニューラルネットワーク	25
探索	14

ち

チャットボット	103, 112
---------	----------

て

ディープラーニング	15, 24
データ型	178
データの前処理	41
データセット	117, 139, 210
データレイク	38
テキスト分類	175
テストデータ	179

と

同期処理	93
特徴マップ	28
特微量抽出	27
トレーニング	161, 236
トレーニングジョブ	162, 184
トレーニング済みモデル	18
トレーニングデータ	18, 178

に	
ニューラルトピックモデル	203
ニューラルネットワーク	26
入力層	26
ニューロン	26
認証情報の保存	65
ね	
ネオコグニトロン	15
の	
ノートブック	161, 166
ノートブックインスタンス	166
は	
パーセプトロン	15
バイアス	26
ハイパーパラメータ	161, 180, 186
バケット	70
発音レキシコン	99
バッチ学習	184
バッチ変換ジョブ	198
ひ	
非同期処理	93
ふ	
プーリング	236
プーリング層	27
プログラミング	19
分類	20
へ	
平均二乗誤差	185, 187
み	
ミニバッチ学習	184
め	
メトリクス	151
も	
モデル	18, 30, 40
モデルの作成	172, 189
モデルのデプロイ	238
モデルのトレーニング	161
よ	
ユーザーの追加	61
予測	130
予測アルゴリズム	128
予測子	127
予測子メトリクス	136
予測の期間	128
予測の周期	128
ら	
ラベリング	160
ラベル	75
ランダムカットフォレスト	203
れ	
レコメンデーション	153
レシピ	140
ろ	
ロジスティック回帰	24
わ	
ワークフロー	40, 43

【著者プロフィール】

井上 研一 (いのうえ けんいち)

IT エンジニア／経済産業省推進資格 IT コーディネータ。

株式会社ビビンコ代表取締役、Tech Garden School 講師。

福岡県北九州市出身。20 年を超える業務システムの開発経験の中で、コールセンターへの AI 導入プロジェクトに参画したことをきっかけに、2016 年に初の著書『初めての Watson ~ API の用例と実践プログラミング』(リックテレコム) を執筆。AI・IoT に強い IT コーディネータとして活動するようになる。2017 年には、北九州市主催のビジネスコンテスト「北九州で IoT」に応募したアイディアが入選。そのメンバーと一緒に株式会社ビビンコを北九州市小倉北区に創業し、IoT ソリューションの開発・導入や、画像認識モデルを活用したアプリの開発などを行っている。

近著に『ワトソンで体感する人工知能』(リックテレコム)、『現場で使える！ Watson 開発入門』(共著、翔泳社)。日本全国でセミナー・研修講師としての登壇も多数。

[プロフィールの詳細と最新情報]

<https://inoccu.com>

つか 使ってわかったAWSのAI ため し かい りょうこう バイソン

まるごと試せば視界は良好 さあPythonではじめよう！

© 井上研一 2020

2020年 6月2日 第1版第1刷発行

著　者 井上研一
発　行　人 新関卓哉
企画担当 蒲生達佳
編集担当 古川美知子
発　行　所 株式会社リックテレコム
〒113-0034 東京都文京区湯島3-7-7
振替　　00160-0-133646
電話　　03(3834)8380(営業)
　　　　03(3834)8427(編集)
URL　　<http://www.ric.co.jp/>

本書の全部または一部について、無断で複写・複製・転載・電子ファイル化等を行うことは著作権法で定める例外を除き禁じられています。

装　丁　長久雅行
組　版　株式会社トップスタジオ
印刷・製本　シナノ印刷株式会社

●訂正等

本書の記載内容には万全を期しておりますが、万一誤りや情報内容の変更が生じた場合には、当社ホームページの正誤表サイトに掲載しますので、下記よりご確認ください。

*正誤表サイトURL

http://www.ric.co.jp/book/seigo_list.html

●本書の内容に関するお問い合わせ

本書の内容等についてのお尋ねは、下記の「読者お問い合わせサイト」にて受け付けております。
また、回答に万全を期すため、電話によるご質問にはお答えできませんのでご了承ください。

*読者お問い合わせサイトURL

<http://www.ric.co.jp/book-q>

●その他のお問い合わせは、弊社サイト「BOOKS」のトップページ <http://www.ric.co.jp/book/index.html> 内の左側にある「問い合わせ先」リンク、またはFAX：03-3834-8043にて承ります。

●乱丁・落丁本はお取り替え致します。

ISBN978-4-86594-246-0

使ってわかった

AWS の AI

まるごと試せば視界は良好
さあ **Python** ではじめよう！



9784865942460



1923055026001

ISBN978-4-86594-246-0
C3055 ¥2600E

定価(本体 2,600円+税)

B8-02 人工知能・機械学習

〈本書の主な内容〉

第1章 人工知能とは何か

第2章 AWS の機械学習サービス

第3章 AI サービス

第4章 Amazon SageMaker

第5章 AWS Deep Learning AMI