aws

での本番稼働用 ML パイプラインの作成 AWS

AWS 規範ガイダンス

AWS 規範ガイダンス: での本番稼働用 ML パイプラインの作成 AWS

Table of Contents

序章 1
..... 1
..... 3
..... 4
処理ジョブを使用する 4
メインガード説を使用する 5
ユニットテスト 6
..... 8
Step Functions SDK を使用する 8
Step Functions SDK の拡張 9
..... 10
を使用した実装 AWS CloudFormation 10
Step Functions SDK からの出力の変更 11
..... 12
..... 14
さまざまな自動化レベル 14
ML ワークロード用のさまざまなプラットフォーム 14
パイプラインオーケストレーション用のさまざまなエンジン 15
..... 17
その他のリソース 18
ドキュメント履歴 19
..... xx

での本番稼働用 ML パイプラインの作成 AWS

Josiah Davis、Verdi March、Yin Song、Baichuan Sun、Chen Wu、および Wei Yih Yap、Amazon Web Services (AWS)

2021 年 1 月 (ドキュメント履歴)

機械学習 (ML) プロジェクトでは、ビジネス価値をもたらし、現実世界の問題を解決するために、モデリング、実装、制作を含む多段階にわたる多大な作業が必要です。各ステップには数多くの代替手段やカスタマイズオプションがあり、リソースと予算の制約の中で本番環境用の ML モデルを準備することがますます難しくなっています。Amazon Web Services (AWS) では、過去数年間、データサイエンスチームがさまざまな業界セクターと協力して ML イニシアチブに取り組んできました。組織的な問題と技術的な課題の両方から生じる多くの AWS お客様が共有する問題点を特定し、本番環境対応の ML ソリューションを提供するための最適なアプローチを開発しました。

このガイドは、ML パイプラインの実装に携わるデータサイエンティストと ML エンジニアを対象としています。本稼働環境に対応した ML パイプラインを提供するための当社のアプローチについて説明しています。このガイドでは、ML モデルをインタラクティブに (開発中に) 実行することから、ML ユースケースのパイプラインの一部として (本番環境で) デプロイする方法に移行する方法について説明します。この目的のために、カスタム ML ソリューションを本番環境に迅速に提供するためのサンプルテンプレート (「 ML Max プロジェクトプロジェクト」を参照) も開発しました。これにより、設計をあまり選択しなくてもすぐに開始できます。

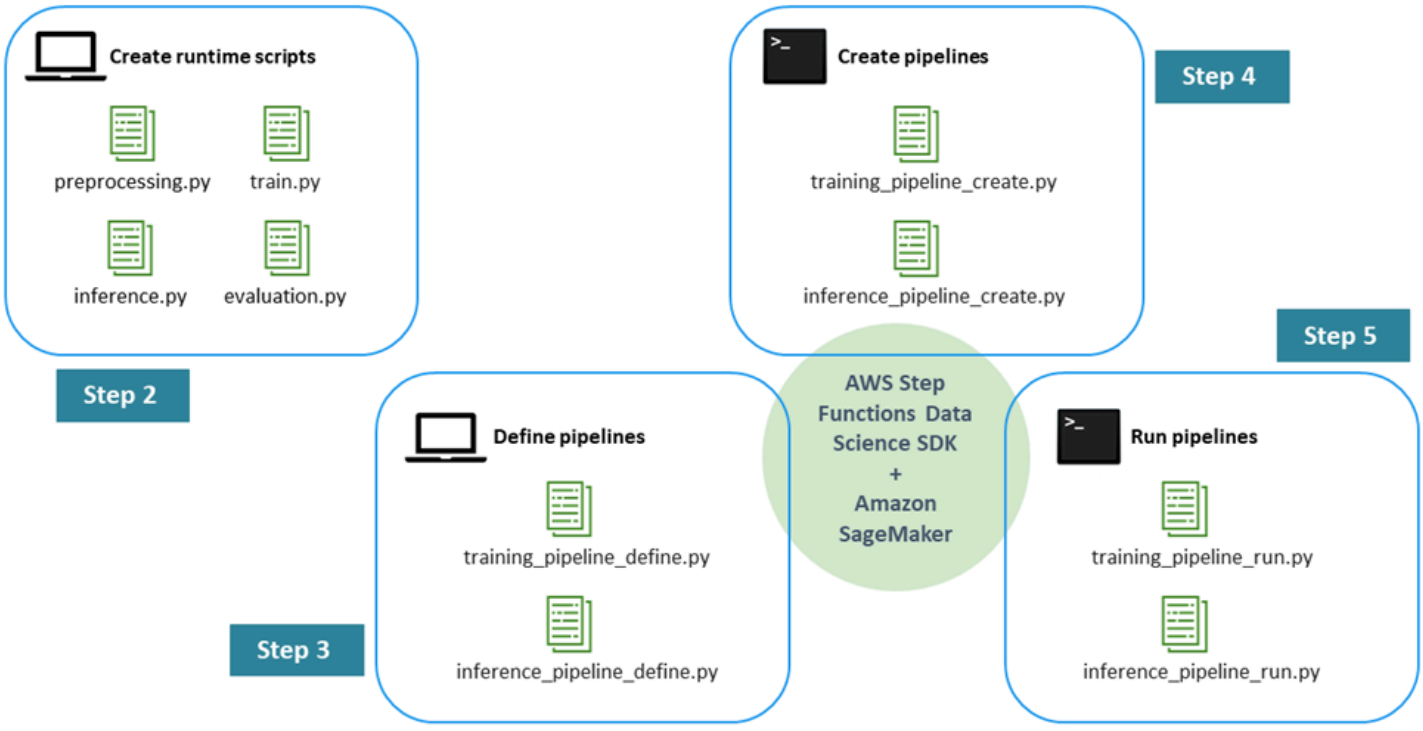
概要

本番環境に対応した ML パイプラインを作成するプロセスは、以下のステップで構成されています。

- ステップ 1。EDA の実行と初期モデルの開発 — データサイエンティストは、Amazon Simple Storage Service (Amazon S3) で生データを利用できるようにし、探索的データ分析 (EDA) を実行し、初期 ML モデルを開発して、その推論性能を評価します。これらのアクティビティは Jupyter Notebook を通じてインタラクティブに実行できます。
- ステップ 2。ランタイムスクリプトの作成 — モデルをランタイム Python スクリプトと統合して、ML フレームワーク (この場合は Amazon SageMaker AI) で管理およびプロビジョニングできるようにします。これは、スタンドアロンモデルのインタラクティブな開発から本番環境に移行するための第一歩です。具体的には、前処理、評価、トレーニング、推論のロジックを個別に定義します。

- ステップ 3。パイプラインの定義 — パイプラインの各ステップの入力と出力のプレースホルダーを定義します。これらの具体的な値は、後のランタイム (ステップ 5) に提供されます。トレーニング、推論、相互検証、バックテストのためのパイプラインに焦点を当てます。
- ステップ 4。パイプラインの作成 – を使用して、 AWS Step Functions ステートマシンインスタンスを含む基盤となるインフラストラクチャを自動 (ほぼワンクリック) で作成します AWS CloudFormation。
- ステップ 5。パイプラインの実行 — ステップ 4 で定義したパイプラインを実行します。また、ステップ 3 で定義した入出力プレースホルダーの具体的な値を入力するためのメタデータとデータまたはデータの場所を準備します。これには、ステップ 2 で定義したランタイムスクリプトとモデルのハイパーパラメータが含まれます。
- ステップ 6。パイプラインの拡張 — 継続的インテグレーションと継続的デプロイ (CI/CD) プロセス、自動再トレーニング、スケジュールに基づく推論など、パイプラインの拡張を実装します。

次の図は、このプロセスの主な手順を示しています。



Steps for creating the training and inference pipeline

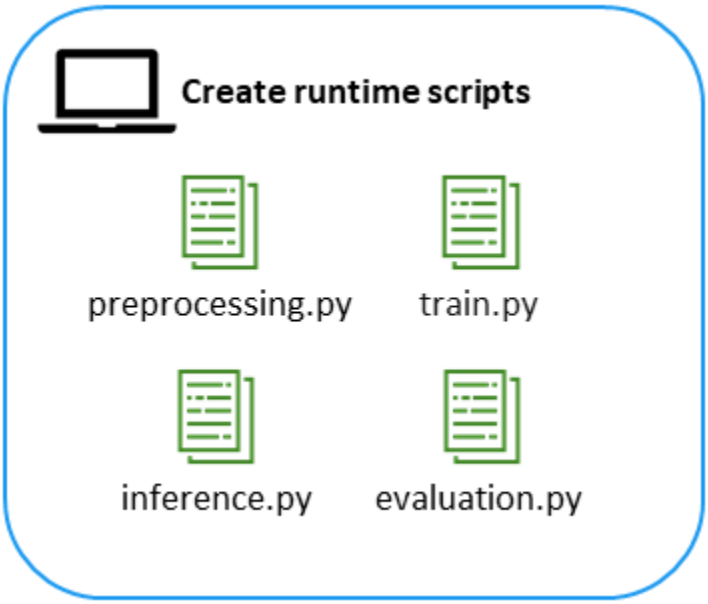
ステップ 1. EDA を実施し、初期モデルを開発する

このステップでは、データサイエンティストは ML のユースケースとデータを理解するために、探索的データ分析 (EDA) を行います。そして、与えられたユースケースにおける問題を解決するための ML モデル (例えば、分類モデルや回帰モデル) を開発します。モデル開発中、データサイエンティストは、データ形式、データライフサイクル、中間出力の場所など、入力と出力について推測することがよくあります。これらの前提条件は、ステップ 2 のユニットテストでの検証に使用できるように文書化する必要があります。

このステップはモデル開発に重点を置いているが、データサイエンティストはしばしば、前処理、トレーニング、評価、推論のための最低限のヘルパーコードを書かなければなりません。データサイエンティストは、開発環境でこのコードを実行できるはずです。また、オプションのランタイム引数を指定して、このヘルパーコードを手動で大幅に変更しなくても他の環境でも実行できるように動的に設定できるようにすることをお勧めします。これにより、ステップ 2 および 3 でモデルとパイプラインの統合が加速されます。例えば、生データを読み取るコードは、一貫した方法でデータを前処理できるように、関数にカプセル化されるべきです。

ML モデルとそのヘルパーコードを開発するには、「scikit-learn」、「XGBoost」、「PyTorch」、「Keras」、「TensorFlow」などのフレームワークから始めることをお勧めします。たとえば、scikit-learn は Python で書かれた無料の ML ライブラリです。オブジェクトに統一された API 規則を定めており、Estimator、Predictor、Transformer、model という 4 つの主要なオブジェクトが含まれています。これらのオブジェクトは軽量データ変換に対応し、ラベルや特徴量エンジニアリングをサポートし、前処理とモデリングのステップをカプセル化します。これらのオブジェクトは、ボイラープレートコードの拡散を防ぎ、検証データやテストデータがトレーニングデータセットに漏れるのを防ぐのに役立ちます。同様に、すべての ML フレームワークには重要な ML アーティファクトが独自に実装されているため、ML モデルを開発するときは、選択したフレームワークの API 規則に従うことをお勧めします。

ステップ 2. ランタイムスクリプトを作成する



このステップでは、ステップ 1 で開発したモデルとそれに関連するヘルパーコードを ML プラットフォームに統合して、本番環境ですぐに使えるトレーニングと推論を行います。具体的には、モデルを SageMaker AI に組み込むためのランタイムスクリプトの開発が含まれます。これらのスタンドアロン Python スクリプトには、定義済みの SageMaker AI コールバック関数と環境変数が含まれています。これらは Amazon Elastic Compute Cloud (Amazon EC2) インスタンスでホストされている SageMaker AI コンテナ内で実行されます。Amazon SageMaker AI Python SDK ドキュメントには、これらのコールバックと補助セットアップがどのように連携してトレーニングと推論を行うかについての詳細な情報が記載されています。以下のセクションでは、 AWS お客様と協働した経験に基づいて ML ランタイムスクリプトを開発する際のその他の推奨事項を説明します。

処理ジョブを使用する

SageMaker AI には、バッチモードでのモデル推論を実行するための 2 つのオプションがあります。SageMaker AI 処理ジョブまたはバッチ変換ジョブを使用できます。各オプションにはメリットとデメリットがあります。

処理ジョブは、SageMaker AI コンテナ内で実行される Python ファイルで構成されます。処理ジョブは、Python ファイルに入力したすべてのロジックで構成されます。これには次のような利点があります。

- トレーニングジョブの基本的なロジックを理解していれば、処理ジョブは簡単に設定でき、理解しやすくなります。これらはトレーニングジョブと同じ抽象概念を共有しています (たとえば、インスタンス数やデータ分散のチューニング)。
- データサイエンティストと ML エンジニアは、データ操作オプションを完全に制御できます。
- データサイエンティストは、使い慣れた読み取り/書き込み機能以外は I/O コンポーネントのロジックを管理する必要がありません。
- SageMaker AI 以外の環境でファイルを実行する方がいくらか簡単であり、迅速な開発とローカルテストに役立ちます。
- エラーが発生した場合、スクリプトが失敗すると同時に処理ジョブも失敗し、予期せぬ再試行待ちが発生することはありません。

一方、バッチ変換ジョブは SageMaker AI エンドポイントの概念を拡張したものです。ランタイムに、これらのジョブはコールバック関数をインポートし、その I/O を処理してデータの読み取り、モデルの読み込み、予測を行います。バッチ変換ジョブには次の利点があります。

- トレーニングジョブで使用される抽象化とは異なるデータ分散抽象化を使用します。
- バッチ推論とリアルタイム推論の両方に同じコアファイルと関数構造を使用しているため、便利です。
- リトライベースのフォールトトレランスメカニズムが組み込まれています。たとえば、レコードのバッチでエラーが発生した場合、ジョブは失敗として終了する前に複数回リトライします。

その透明性、複数の環境での使いやすさ、トレーニングジョブとの抽象化が共通していることから、このガイドで説明するリファレンスアーキテクチャのバッチ変換ジョブの代わりに処理ジョブを使用することにしました。

Python ランタイムスクリプトは、クラウドにデプロイする前にローカルで実行する必要があります。具体的には、Python スクリプトを構造化してユニットテストを行う際には、メインのガード節 (guard clause) を使用することをおすすめします。

メインガード説を使用する

モジュールのインポートをサポートし、Python スクリプトを実行するには、メインガード節を使用してください。Python スクリプトを個別に実行すると、ML パイプラインの問題をデバッグして切り分けるのに役立ちます。次のステップを推奨します。

- Python 処理ファイル内の引数パーサーを使用して、入出力ファイルとその場所を指定します。

- 各 Python ファイルのメインガイドとテスト関数を提供します。
- Python ファイルをテストしたら、 AWS Step Functions モデルを使用しているか SageMaker AI 処理ジョブを使用しているかにかかわらず、ML パイプラインのさまざまなステージに組み込みます。
- テストとデバッグを容易にするために、スクリプトの重要なセクションには Assert ステートメントを使用してください。たとえば、アサートステートメントを使用すると、読み込み後にデータセット特徴量の数が一定になるようにすることができます。

ユニットテスト

パイプライン用に作成されたランタイムスクリプトのユニットテストは、ML パイプライン開発ではしばしば無視される重要なタスクです。これは、機械学習とデータサイエンスは比較的新しい分野であり、ユニットテストなどの確立されたソフトウェアエンジニアリング手法を採用するのが遅れているためです。ML パイプラインは運用環境で使用されるため、ML モデルを実際のアプリケーションに適用する前に、パイプラインコードをテストすることが不可欠です。

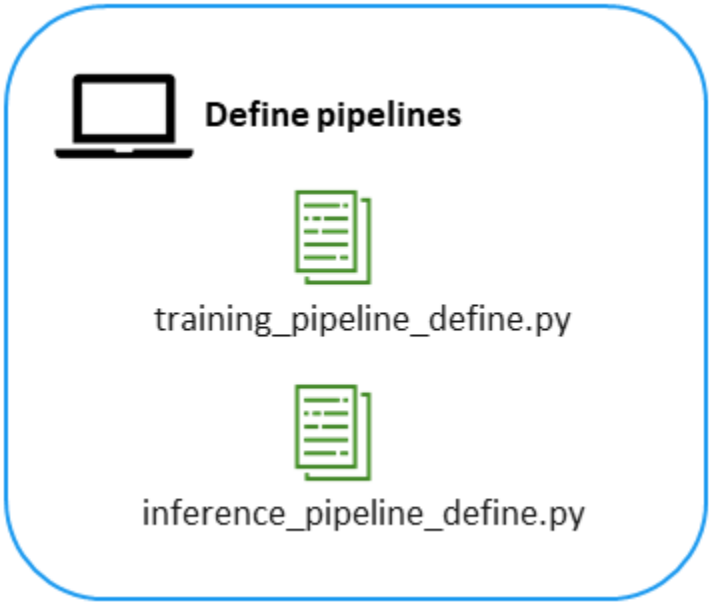
ランタイムスクリプトをユニットテストすることには、ML モデルに次のような独自の利点もあります。

- これにより、予期しないデータ変換が防止されます。ほとんどの ML パイプラインには多数のデータ変換が含まれるため、これらの変換が期待どおりに実行されることが不可欠です。
- これにより、コードの再現性を検証します。コード内のランダム性は、さまざまなユースケースでユニットテストを行うことで検出できます。
- コードのモジュール性が強化されます。ユニットテストは通常、特定のテストスイート (テストケースの集まり) がプログラムのソースコードをどの程度実行するかを示すテストカバレッジ指標と関連付けられます。大量のコードを関数やクラスに分解しないとユニットテストを書くのは難しいので、高いテストカバレッジを実現するために、開発者はコードをモジュール化します。
- これにより、品質の低いコードやエラーが本番環境に導入されるのを防ぐことができます。

ユニットテストケースを書くために、pytest のような成熟したユニットテストフレームワークを使うことを推奨します。フレームワークの中で広範囲なユニットテストを管理する方が簡単だからです。

> **Important**
>
> ユニットテストでは、すべてのコーナーケースがテストされることを保証することはできませんが、モデルをデプロイする前にミスを未然に防ぐのに役立ちます。優れた運用性を確保するため、デプロイ後もモデルをモニタリングすることをお勧めします。

ステップ 3. パイプラインを定義する



このステップでは、パイプラインが実行するアクションの順序とロジックを定義します。これには、個別のステップだけでなく、それらのロジカルな入力と出力も含まれます。たとえば、パイプライン開始時のデータの状態はどうなっているのでしょうか。データは細分性の異なる複数のファイルから取得されるのか、それとも 1 つのフラットファイルからのものなのでしょうか。データが複数のファイルからのものである場合、前処理ロジックを定義するために、すべてのファイルに対して 1 つのステップが必要なのか、それともファイルごとに別々のステップが必要なのか。決定は、データソースの複雑さと前処理の範囲によって異なります。

今回のリファレンス実装では、サーバーレス関数オーケストレーターである AWS Step Functions を使用してワークフローステップを定義します。ただし、ML Max フレームワークは Apache AirFlow (「パイプラインオーケストレーション用のさまざまなエンジン」セクションを参照) などの他のパイプラインシステムやステートマシンシステムもサポートしており、ML パイプラインの開発とデプロイを推進します。

Step Functions SDK を使用する

ML パイプラインを定義するには、まずAWS Step Functions データサイエンス SDK (Step Functions SDK) が提供する高レベルの Python API を使用して、パイプラインの 2 つの主要コンポーネントであるステップとデータを定義します。パイプラインを有向非循環グラフ (DAG) と考えると、ステップはグラフ上のノードを表し、データは 1 つのノード (ステップ) を次のノード (ステップ) に接続す

る有向エッジとして表示されます。ML ステップの一般的な例には、前処理、トレーニング、評価などがあります。Step Functions SDK には、使用できる組み込みステップ (TrainingStep など) が多数用意されています。データの例としては、入力、出力、パイプラインのいくつかのステップによって生成される多くの中間データセットなどがあります。ML パイプラインを設計する場合、データ項目の具体的な値はわかりません。データ項目名とプリミティブデータ型のみを含むテンプレートとして機能するデータプレースホルダーを定義できます(関数パラメーターと同様)。この方法では、グラフ上を移動するデータの具体的な値を事前に知らなくても、完全なパイプラインブループリントを設計できます。この目的のために、Step Functions SDK のプレースホルダークラスを使用して、これらのデータテンプレートを明示的にモデル化できます。

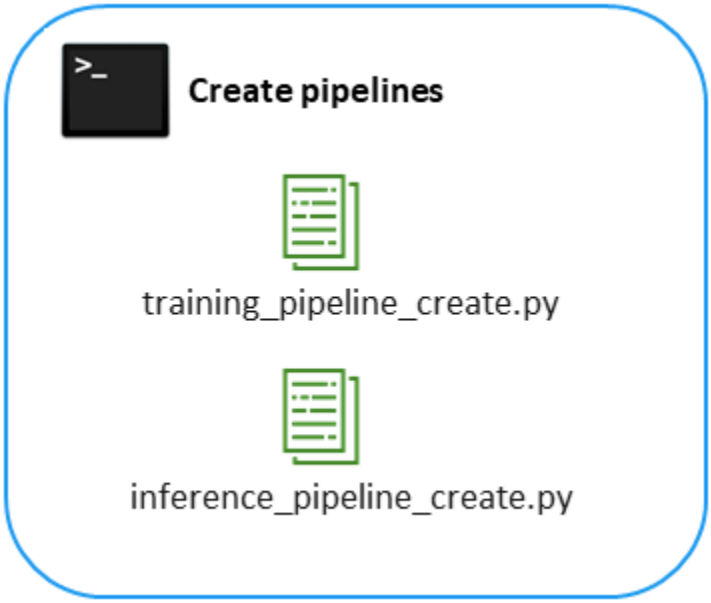
ML パイプラインには、各 ML ステップの動作をきめ細かく制御するための設定パラメーターも必要です。このような特別なデータプレースホルダーはパラメータープレースホルダーと呼ばれます。パイプラインを定義する時点では、その値の多くは不明です。パラメータプレースホルダーの例としては、パイプライン設計中に定義するインフラストラクチャ関連のパラメータ (AWS リージョン コンテナイメージ URL など) や、パイプラインの実行時に定義する ML モデリング関連のパラメータ (ハイパーパラメータなど) などがあります。

Step Functions SDK の拡張

リファレンス実装における要件の 1 つは、特定のパラメータ設定を使用して ML パイプライン定義を具体的な ML パイプラインの作成とデプロイから分離することでした。ただし、Step Functions SDK の組み込みステップの中には、これらすべてのプレースホルダーパラメーターを渡すことができないものもありました。その代わり、パラメータ値は、パイプライン設計時に SageMaker AI 設定 API 呼び出しを通じて直接取得されることが想定されていました。これは SageMaker AI の設計時環境が SageMaker AI ランタイム環境と同一であれば問題なく動作しますが、実際の設定ではそうなることはほとんどありません。パイプライン設計時間とランタイムがこれほど密結合していることと、ML プラットフォームインフラストラクチャが一定に保たれるという前提は、設計されたパイプラインの適用性を著しく妨げています。実際、ML パイプラインは、基盤となるデプロイプラットフォームに少しでも変更が加えられるとすぐに機能しなくなります。

この課題を克服し、堅牢な ML パイプライン (一度設計すればどこでも実行したい) を作成するために、TrainingStep、ModelStep、TransformerStep などの組み込みステップを拡張して、独自のカスタムステップを実装しました。これらの拡張は ML Max プロジェクトで提供されています。これらのカスタムステップのインターフェースは、パイプラインの作成時やパイプラインの実行時に具体的な値を入力できるパラメータプレースホルダーをはるかに多くサポートしています。

ステップ 4. パイプラインの作成



パイプラインを論理的に定義したら、パイプラインをサポートするインフラストラクチャを作成します。このステップには、少なくとも以下の機能が必要です。

- コード、モデルアーティファクト、トレーニングや推論の実行に使用されるデータなど、パイプラインの入出力をホストし、管理するためのストレージ。
- モデリング、推論、およびデータの前処理と後処理を行うコンピューティング (GPU または CPU)。
- 使用中のリソースを管理し、定期的な実行をスケジュールするオーケストレーション。たとえば、新しいデータが利用可能になったら、モデルを定期的に再トレーニングできます。
- パイプラインモデルの精度、リソースの利用状況、トラブルシューティングをモニタリングするためのログとアラート。

を使用した実装 AWS CloudFormation

使用したパイプラインを作成するために。これは AWS CloudFormation、Infrastructure as Code をデプロイおよび管理するための AWS サービスです。 AWS CloudFormation テンプレートには、前のステップで Step Functions SDK を使用して作成された Step Functions 定義が含まれています。このステップには、Step Functions ステートマシンと呼ばれる 管理の Step Functions インスタンスの作成が含まれます。トレーニングジョブと推論ジョブは、必要なときにのみ SageMaker AI ジョブと

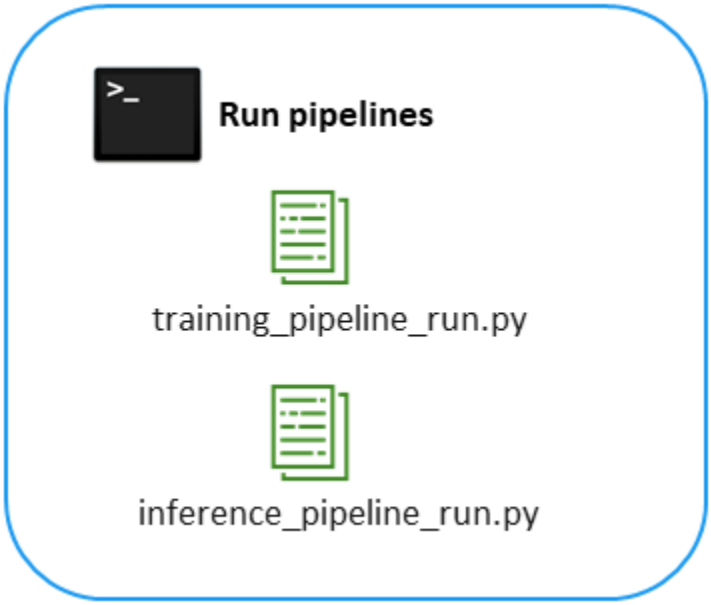
してオンデマンドで実行されるため、この段階ではトレーニングと推論のためのリソースは作成されません。このステップには、Step Functions の実行、SageMaker AI の実行、Amazon S3 からの読み取りと書き込みを行うための AWS Identity and Access Management (IAM) ロールの作成も含まれます。 Amazon S3

Step Functions SDK からの出力の変更

前のセクションの CloudFormation 出力にいくつかの小さな変更を加える必要がありました。単純な Python 文字列マッチングを使用して次のことを行いました。

- CloudFormation テンプレートの `Parameters`セクションを作成するロジックを追加しました。これは、2 つのロールを作成し、デプロイ環境とともにパイプライン名をパラメータとして定義したいからです。このステップでは、ステップ 6 で説明したように、追加で作成するリソースやロールについても説明します。
- デプロイプロセスの一環として動的に更新できるように、3 つのフィールドを必要な `!Sub` プレフィックスと引用符で再フォーマットしました。
  - ステートマシンに名前を付ける `StateMachineName` プロパティ。
  - ステートマシンを定義する `DefinitionString` プロパティ。
  - ステートマシンによって返される `RoleArn` プロパティ。

ステップ 5. パイプラインを実行する



このステップでは、ステップ 4 で CloudFormation スタックで作成されたトレーニングパイプラインまたは推論パイプラインを実行します。パイプラインは、内部のプレースホルダーパラメータに具体的な値が入力されるまで実行できません。プレースホルダーパラメータに値を割り当てるこのアクションは、ステップ 5 の主要なアクティビティです。プレースホルダーパラメータには以下が含まれます。

- 入力、出力、および中間データセットの場所
- ステップ 2 で開発されたランタイムスクリプトとその他の前処理コードまたは評価コード (トレーニングパイプラインの `sm_submit_url` など) が格納されている Amazon S3 の場所
- AWS リージョンの名前

パイプラインを実行する前に、これらのパス値が有効なデータまたはコードを指していることを確認する必要があります。たとえば、Python ランタイムスクリプトの Amazon S3 URL を表すプレースホルダーパラメータを入力する場合、それらのスクリプトをその URL にアップロードする必要があります。パイプラインを実行する担当者は、整合性チェックとデータのアップロードを担当します。パイプラインを定義または作成する人は、これについて何も心配する必要はありません。

パイプラインの成熟度によっては、このステップを定期的に (毎週または毎月) 実行するように自動化されている場合があります。オートメーションには堅牢なモニタリングも必要です。これは重要な領域ですが、本ガイドの対象外です。トレーニングパイプラインの実行には、評価メトリクスをモニ

タリングするのが適切でしょう。推論パイプラインでは、入力データの分布ドリフトを監視し、可能であれば定期的にラベルを収集して予測精度のドリフトを測定するのが適切でしょう。トレーニングと推論の実行によるこれらの記録は、後で分析できるようにデータベースに記録しておく必要があります。

ステップ 6。パイプラインの拡大

このガイドでは、具体的なアーキテクチャを使用して、AWS で ML パイプラインの構築を迅速に開始する方法について説明します。パイプラインを完成させるためには、メタデータの管理、実験の追跡、モニタリングなど、他にも考慮すべき点があります。これらは、このガイドの対象外である重要なトピックです。以下のセクションでは、パイプライン管理のもう 1 つの側面であるパイプラインの自動化について説明します。

さまざまな自動化レベル

SageMaker AI コンソールでトレーニングパイプラインを手動で設定することもできますが、実際には、ML モデルが一貫して繰り返し展開されるように、ML トレーニングパイプラインのデプロイにおける手動タッチポイントを最小限に抑えることをお勧めします。要件と対応するビジネス上の問題に応じて、半自動、完全自動、完全管理の 3 つのレベルでデプロイ戦略を決定し、実施することができます。

- 半自動 - 前のセクションで説明した手順は、CloudFormation テンプレートを使用してトレーニングと推論のパイプラインをデプロイするため、デフォルトでは半自動のアプローチを採用しています。これはパイプラインの再現性を確保し、変更や更新を容易にするのに役立ちます。
- 完全自動 - より高度なオプションは、開発環境、ステージング環境、本番環境への継続的インテグレーションと継続的デプロイ (CI/CD) を使用することです。トレーニングパイプラインのデプロイに CI/CD のプラクティスを組み込むことで、自動化にトレーサビリティと品質ゲートを確実に含めることができます。
- 完全管理 - 最終的には、シンプルなマニフェストセットを含む ML トレーニングパイプラインをデプロイできる完全マネージド型のシステムを開発でき、必要な AWS サービスをシステムが自己設定して調整できるようになります。

このガイドでは、具体的なアーキテクチャを紹介することにしました。ただし、検討できる代替技術もあります。次の 2 つのセクションでは、プラットフォームとオーケストレーションエンジンのいくつかの代替案について説明します。

ML ワークロード用のさまざまなプラットフォーム

Amazon SageMaker AI は、ML モデルのトレーニングと提供を行うための AWS マネージドサービスです。多くのユーザーは、ML ワークロードを実行するための幅広い組み込み機能と、多数のオプ

ションを高く評価しています。SageMaker AI は、クラウドに ML 実装を始めたばかりの場合に特に便利です。SageMaker AI の主な機能は次のとおりです。

- 組み込みのトレーサビリティ (ラベル付け、トレーニング、モデルトラッキング、最適化、推論を含む)。
- Python と ML の経験が最小限で済むトレーニングと推論のためのワンクリックオプションが組み込まれています。
- 高度なハイパーパラメータチューニング。
- すべての主要な人工知能と機械学習 (ML/AI) フレームワークとカスタム Docker コンテナのサポート。
- 内蔵モニタリング機能。
- トレーニングジョブ、処理ジョブ、バッチ変換ジョブ、モデル、エンドポイント、検索性を含む履歴の追跡機能を内蔵。トレーニング、処理、バッチ変換など、一部の履歴は不変で、追記のみです。

SageMaker AI の使用に対する代替方法の 1 つは、AWS Batch です。AWS Batch は、コンピューティングとオーケストレーションの低レベルの制御をユーザーの環境に提供しますが、機械学習用にカスタムビルドされているわけではありません。いくつかの主な特徴は以下のとおりです。

- ワークロードに基づくコンピュートリソースのすぐに使える自動スケーリング。
- すぐにサポートできるジョブ優先度、リトライ、ジョブ依存性。
- リカレントジョブやオンデマンドジョブの構築をサポートするキューベースのアプローチ。
- CPU と GPU のワークロードのサポート。GPU は、特にディープラーニングモデルの場合、トレーニングプロセスを大幅にスピードアップできるため、ML モデルの構築に GPU を使用できることが不可欠です。
- コンピューティング環境用のカスタム Amazon マシンイメージ (AMI) を定義できる能力。

パイプラインオーケストレーション用のさまざまなエンジン

2 つ目の主要コンポーネントは、パイプラインオーケストレーションレイヤーです。AWS は完全に管理されたオーケストレーション体験のための Step Functions を提供します。Step Functions の代わりによく使われるのが、Apache Airflow です。この 2 つのいずれかを選択するには、以下の点を考慮してください。

- 必要なインフラストラクチャ - AWS Step Functions はフルマネージドサービスでサーバーレスであるのに対し、Airflow は独自のインフラストラクチャを管理する必要があり、オープンソースソフトウェアをベースにしています。その結果、Step Functions は設定なしですぐに高可用性を実現できますが、Apache Airflow の管理には追加の手順が必要です。
- スケジューリング機能 - Step Functions も Airflow も同等の機能を備えています。
- 視覚化機能と UI - Step Functions も Airflow も同等の機能を備えています。
- 計算グラフ内での変数の受け渡し - Step Functions が AWS Lambda 関数を使用するための限られた機能を提供するのに対し、Airflow は XCom インターフェイスを提供します。
- 使い方 - Step Functions は AWS カスタマーの間で非常に人気があり、Airflow はデータエンジニアリングコミュニティで広く採用されています。

結論

機械学習が研究分野から応用分野へと移行するにつれ、さまざまな業界で ML パイプラインの開発、デプロイ、運用が年間 25% の成長を遂げています。ML のビジネス価値は、日常的な ML 運用とパイプラインを通じて実現され、ひいては ML モデルとアルゴリズムの研究開発を促進します。とはいえ、ML を本番環境に導入すると、データ管理、処理、分析、モデリング、検証、セキュリティなど、さまざまなアクティビティやアーティファクトが非常に絡み合うため、多くの課題が伴います。データサイエンスチームは、AWS カスタマーとの数多くの AI/ML の取り組みを通じて、さまざまな ML DevOps アクティビティやアーティファクトを最適に融合または分離するためのテンプレートセットを提供するエンドツーエンドのワークフローがないことが重要な課題であることに気づきました。このガイドでは、この差し迫った問題に対処するための「ML Max ワークフロー」を紹介しました。ML Max は、ステップバイステップのガイドラインとプログラミングテンプレートのセットを提供します。目標は、インタラクティブなモデル開発段階から、生産準備が整った完全でスケーラブルな ML パイプライン構成への迅速かつ費用対効果の高い移行を可能にすることです。

その他のリソース

関連ガイドとパターン

- 深層学習システムにおける不確実性の定量化
- AWS デベロッパーツールを使用して ML ビルド、トレーニング、デプロイのワークロードを Amazon SageMaker AI に移行
- AWS 規範ガイダンスのウェブサイト

AWS のサービス

- AWS Batch
- AWS CloudFormation
- IAM
- Amazon SageMaker AI
- AWS Step Functions

AWS 一般的なリソース

- AWS「ドキュメント」
- AWS 参考文献
- AWS「用語集」

ドキュメント履歴

以下の表は、本ガイドの重要な変更点について説明したものです。今後の更新に関する通知を受け取る場合は、RSS フィード をサブスクライブできます。

変更	説明	日付
初版発行	—	2021 年 1 月 29 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。