金融サービスにおける機械学習のベストプラクティス

Amazon SageMaker を使用したセキュリティおよび

機械学習のガバナンスプラクティスの概要

2020 年 7 月

注意

お客様は、この文書に記載されている情報を独自に評価する責任を負うものとします。この文書に含まれる情報は、例示を目的としたものです。法律、コンプライアンス、規制に関する助言を提供することを目的としたものではありません。お客様に適用される法律については、お客様ご自身で確認する必要があります。本書は、(a) 情報提供のみを目的としており、(b) AWS の現行製品とプラクティスを説明していますが、予告なしに変更されることがあり、(c) AWS およびその関連会社、サプライヤーまたはライセンサーからの契約義務や確約を意味するものではありません。お客様に対する AWS の責任は、AWS 契約により規定されます。本書は、AWS とお客様の間で締結される一切の契約の一部ではなく、その内容を修正することもありません。

目次

要約

このホワイトペーパーには、機械学習のアプリケーションを使用している金融機関を対象とした、セキュリティとモデルガバナンスに関する考慮事項の概要が記されています。AWS 上に安全な機械学習環境を作成する方法、および組織のリスク耐性、既存ガバナンスとの統合、規制における期待に基づいてモデルガバナンスのベストプラクティスを使用する方法を例示しています。

はじめに

現在、ビジネスをトランスフォームするために機械学習 (ML) モデルを構築する金融機関がますます増えています。こうした金融機関は、たとえば次のような幅広いユースケースに ML を適用しています。

- 不正行為検出
- 市場監視
- デューディリジェンス
- ポートフォリオの最適化
- カスタマーエクスペリエンスのエンハンス
- 取引実行の最適化
- 定量的戦略の開発
- アルゴリズムトレード

ML モデルの学習能力により、より正確な予測を支援します。ただし、こうしたモデルが規制要件に沿って使用され、安全かつ適切に管理されるようにするためには、注意を怠らないことが必要です。

金融サービス業界がガイダンスとして使用できる文献は、数多く公開されています。例:

- Guidance on Model Risk Management は、連邦準備制度と OCC によって共同で作成されたものです。この文献では、Federal Reserve SR 11-7, OCC 2011-12 に記載されているガイダンスを含む、効果的なモデルガバナンスの必要性が説明されています。
- European Central Bank (ECB) guide to internal models では、安全かつ適切に管理された機械学習環境を実装する方法が説明されています。

ML のライフサイクル: ML ワークロードのプロビジョニング、管理、オペレーショナライズの方法

本セクションでは、エンドツーエンドの機械学習パイプラインを構成しているさまざまな要素のハイレベルな概要を説明します。本セクションはすべてを網羅したガイドではありませんが、金融サービス向けの機械学習ワークフローのセキュリティ、ガバナンス、運用に関する当社の考察を確認する一助となります。図 1 は、機械学習のライフサイクルを解説しています。

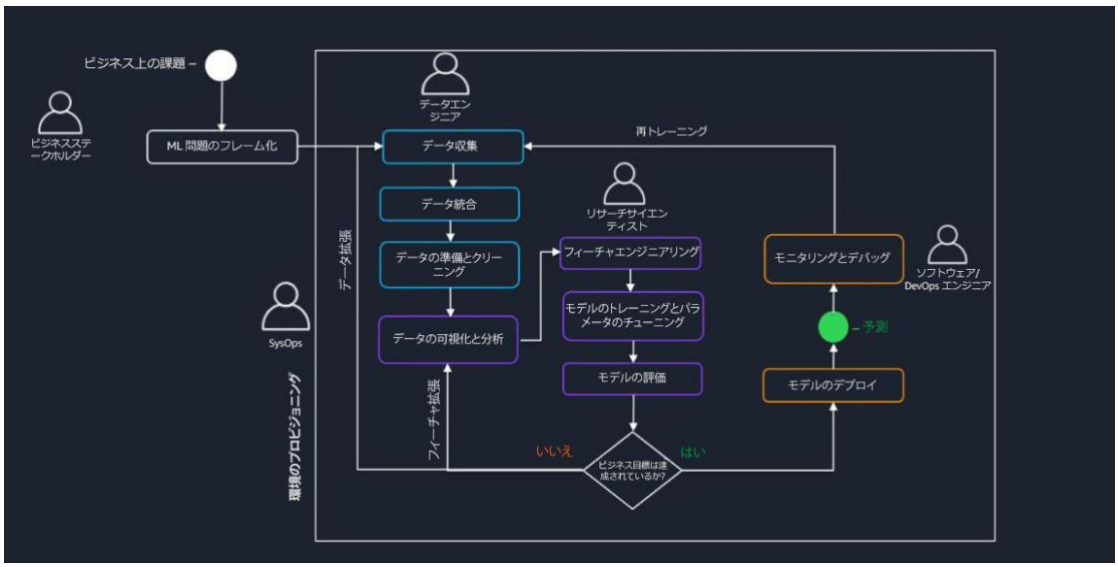


図 1 - 一般的な ML ライフサイクル

ML の一般的なワークフローには、複数のステークホルダーが関与します。このワークフローを適切に管理し運用するには、チームを横断した協力が不可欠です。ステークホルダーは通常、ビジネス上のハイレベルな問題をフレーム化します。問題が特定されたら、明確に定義されたインプットとアウトプットを使って、問題を変換し、機械学習の問題として構成します。ユースケースを運用する際の複雑性を把握するために、このプロセスには通常はデータサイエンスのセンターオブエクセレンス (CoE) チームに IT チームとともに参加しているビジネスステークホルダーと、リスクおよびコンプライアンス関連のパートナーが関与します。

この時点で、関連する成功メトリクスまたは主要業績評価指標 (KPI) を収集する必要があります。これはモデルの観点、および本番稼動ソリューションにおける非機能要件の観点 (ソリューションの有効性に影響する規制またはコンプライアンス上のニーズなど) の両方から行います。たとえば、規制によっては、下された決定の理由を説明する際に特定の要件が課されることがあります。それによって、データサイエンスチームが調査できるアルゴリズムのタイプが決まる可能性があります。こうしたことを決定するのは、通常、規制やコンプライアンスの専門家、ビジネスステークホルダーです。

次のステージでは、さまざまなソースから入力データを収集します。この役割は、多くの場合、データのインジェスト、抽出、変換、ロード (ETL) を行うためのビッグデータツールに精通しているエンジニアリングチームが担当します。データがバージョン管理され、データ系列が監査およびコンプライアンス向けに追跡されていることを保証することが重要になります。詳細は、Establish ML Governance を参照してください。

データが収集されると、データサイエンティストは通常、そのデータから適切に選択されたサブセットを調べてデータセットのスキーマを理解し、データセットをクリーンアップして欠落した値を削除または代入し、モデル構築に適した数値形式にデータを変換するために必要になる変換 (外れ値の削除、入力に依存しない変数のスケーリング、次元削減、カテゴリ変数のエンコードなど) を識別します。このプロセスは「特徴量エンジニアリング」と呼ばれ、通常、データサイエンティストの大半の時間が費やされますが、予測に寄与して、かつ従属変数と相関しているシグナルを含む、主要な特徴量を識別するための重要なステップとなります。

特徴量セットが選択されると、データサイエンティストは通常、どのフレームワークを使用するか、およびどのモデルで試すかを決定します。そして、モデル間でトレーニングイテレーションを開始します。識別された KPI に対するモデリングのパフォーマンスを改善するには、ハイパーパラメータ (トレーニングに先だって最初に値を定義する必要があるパラメータ) をチューニングするか、またはラベル付けされたトレーニングデータをより多く組み込んでモデルがさらに多くのインスタンスを参照し、さらに学習できるよ

うにします。モデルのトレーニングが完了したら、モデルのアーティファクトと対応するコードを適切にバージョニングし、一元化されたコードリポジトリまたはアーティファクト管理システムに保存します。本プロセスにおけるこのステージは実験的なものであり、モデルパフォーマンスが一貫して低ければ、データサイエンティストは特徴量エンジニアリングのステージやデータ収集のステージにまで戻ることもよくあります。データサイエンティストはまた、モデルパフォーマンスを向上させる上で重要な特徴量を洗い出すために多数の方法を使用します。

次に、DevOps エンジニアは通常、トレーニング済みのモデルをピックアップし、本番環境へのデプロイに向けた準備を行います。この時点で、モデルは、既存のパイプラインおよびビジネスワークフローと統合してテストし、過去のデータで検証し、推論を大規模に生成するためにパフォーマンスをテストし、過去のモデルと比較して品質をテストする必要があります。すべての依存関係、モデルアーティファクト、関連するメタデータを管理し、保存することが必要です。このステージでは通常、手動 (“human-in-the-loop”) プロセスにより、モデルがビジネス上の目標ならびに規制またはコンプライアンス上の要件を満たしていることを保証します。モデルが本番稼動向けにデプロイされると、モデルの再トレーニングや廃止 (これらはビジネスニーズの発展に伴って必要となるケースがあります) をトリガーするために、モデルは定期的にパフォーマンスの低下、データドリフト、コンセプトドリフトがモニタリングされます。これは、多くの場合ベストプラクティスですが、金融機関にモデルの定期的なレビューを要求している規制においても推奨されています。

図 2 は、機械学習のライフサイクル全体にわたって関与する、さまざまなテクニカルステークホルダーおよびビジネスステークホルダーの典型的な例を示しています。

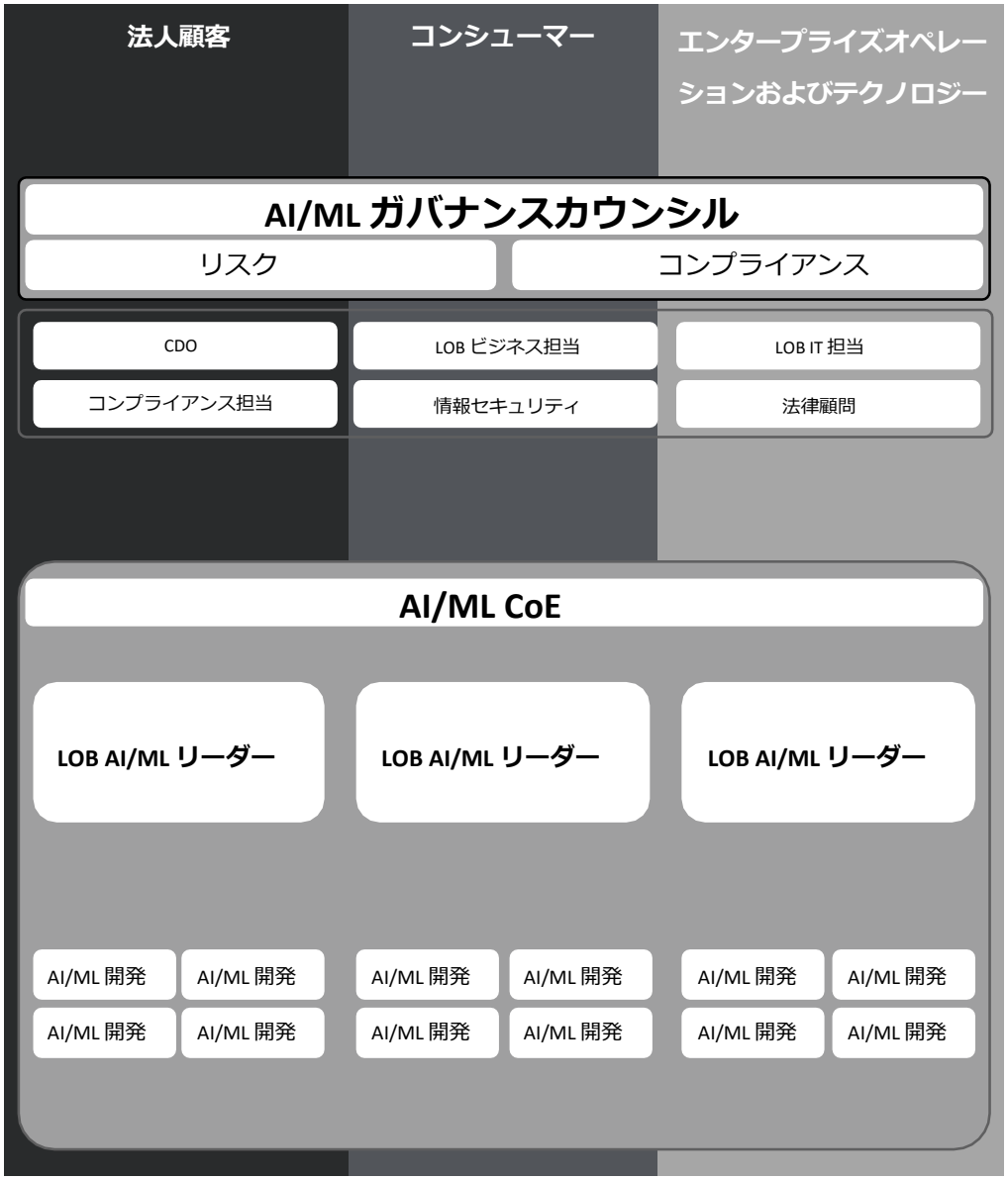


図 2 - ML ライフサイクルにおける一般的なステークホルダーとその役割

上図が示すとおり、ML ライフサイクルの各ステージでは、アクセスコントロール、セキュリティ、ガバナンスおよびモニタリングのレイヤーを適切に保つ必要があります。

本ホワイトペーパーの残りの部分では、ML ライフサイクル全体で Amazon SageMaker と他の AWS ツールを併用する方法を説明、例示していきます。具体的には、規制を受ける予測モデルを実行しているお客様の経験をもとに、次のトピックについて取り上げます。

A. **安全な ML 環境のプロビジョニング:**

- **コンピューティングとネットワークの分離:** SageMaker をインターネット接続なしでお客様のプライベートネットワークにデプロイする方法。
- **認証と承認:** 制御された形でユーザーを認証し、IAM アクセス許可に基づいてマルチテナンシーなしでユーザーを承認する方法。
- **データ保護:** 顧客が用意した暗号化キーを使って、転送時および保管時のデータを暗号化する方法。
- **可監査性:** 所定の時点で誰が何をしたかを監査、検出、防止して、悪意のあるアクティビティを早期にブロックまたは検出する方法。

B. **ML ガバナンスの確立:**

- **トレーサビリティ:** データ準備、モデル開発、トレーニングイテレーションから ML モデルの系列を追跡する方法、および所定の時点で誰が何をしたかを監査する方法。
- **説明可能性と解釈可能性:** トレーニング済みモデルの説明や解釈、および特徴量重要度の取得に役立つ方法。
- **モデルモニタリング:** 本番稼動中のモデルをモニタリングしてデータドリフトから保護する方法、および自分で定義したルールに自動的に反応する方法。
- **再現性:** モデルの系列および保存されたアーティファクトによって ML モデルを再現する方法。

C. **ML ワークロードのオペレーショナライズ:**

- **モデル開発ワークロード:** 開発環境で自動と手動の両方のレビュープロセスを構築する方法。
- **本番稼動前のワークロード:** AWS Code* Suiteおよび AWS Step Functions を使用して自動化された CI/CD パイプラインを構築する方法。

- **本番稼動中のおよび継続的なモニタリングワークロード:** 継続的デプロイと自動化されたモデルモニタリングを組み合わせる方法。
- **追跡とアラート:** モデルメトリクス (オペレーション可能かつ統計的) を追跡する方法、および問題が発生した場合に適切なユーザーにアラートを送信する方法。

A. 安全な ML 環境のプロビジョニング

金融機関にとって、機械学習環境のセキュリティは最優先事項です。不正アクセス、権限昇格攻撃、データ漏洩は、必ず防止する必要があります。安全な機械学習環境をセットアップするときの考慮事項には、一般に次の 4 つが挙げられます。

- コンピューティングとネットワークの分離
- 認証と承認
- データの暗号化
- 可監査性

次セクションでこれらの考慮事項について説明します。

コンピューティングとネットワークの分離

適切に管理された安全な機械学習ワークフローは、プライベートかつ分離されたコンピューティングおよびネットワーク環境を構築することから始まります。SageMaker とその関連コンポーネント (Jupyter ノートブック、トレーニングインスタンス、ホスティングインスタンスなど) をホストする仮想プライベートクラウド (VPC) は、インターネットに接続していないプライベートネットワークにデプロイする必要があります。また、これらの SageMaker のリソースを顧客の VPC 環境に関連付けると、SageMaker リソースへのアクセス管理や VPC 環境に出入りするデータの送受信を制御するセキュリティグループなど、ネットワークレベルの管理を適用できます。SageMaker と、Amazon Simple

Storage Service (Amazon S3) などさまざまな AWS のサービスとの接続は VPC エンドポイントを使って確立する必要がありますが、AWS PrivateLink も使用できます。さらに、VPC エンドポイントを作成するときは、エンドポイントポリシーをアタッチし、接続している特定のリソースへのアクセスも制御することが推奨されます。次の図 3 は、SageMaker ノートブックインスタンスの推奨されるデプロイを示しています。

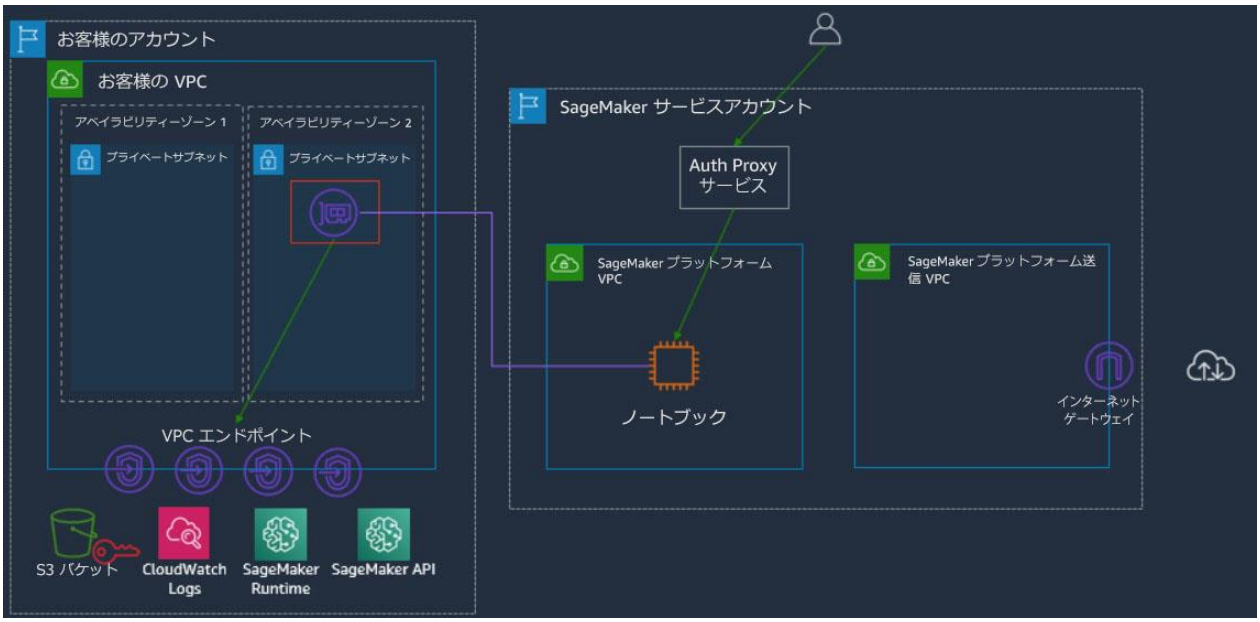


図 3 - Amazon SageMaker ノートブックインスタンスの推奨デプロイアーキテクチャ

同様に、トレーニングジョブとホストされたモデルの場合、SageMaker EC2 インスタンスは Amazon S3 からトレーニングデータを取得するときに VPC を介して Amazon S3 と通信します。SageMaker はこの作業を、指定された VPC 内に Elastic Network Interface (ENI) を作成し、これをサービスアカウントの Amazon EC2 インフラストラクチャにアタッチすることで行います。このパターンを使用することで、SageMaker で実行しているサービスへのネットワークレベルのアクセスを制御できます。同様に、SageMaker インスタンスがホスティングのためにトレーニング済みモデルを取得するとき、インスタンスは VPC を介して Amazon S3 と通信します。SageMaker のリソースとのネットワーク通信のガバナンスと制御は、このような方法で維持されます。上に示すようなプライベートネットワーク環境の設定の詳細については、Amazon SageMaker Workshop for Secure Networking を参照し

てください。また、Vanguard's 2019 re:Invent presentation on Security for ML environments with Amazon SageMaker も参考になります。

認証と承認

分離されたプライベートネットワーク環境を作成したら、次のステップでは、承認されたユーザーのみが適切な AWS のサービスにアクセスできるようにします。AWS Identity and Access Management (IAM) は、SageMaker のリソースへのアクセス、Amazon S3 のデータへのアクセス、API エンドポイントへのアクセスなど、機械学習環境のさまざまな側面に対する予防的統制を実現する一助となります。AWS には RESTful API を使用してアクセス可能で、すべての API コールは IAM によって承認されます。ユーザーには IAM ポリシードキュメントによって、明示的なアクセス権限が付与されます。ポリシードキュメントによって、プリンシパル (ユーザー)、アクション (API コール)、許可されたリソース (S3 オブジェクトなど)、アクセス権限が付与される条件が指定されます。

したがって、SageMaker Jupyter ノートブックインスタンスへのアクセスも IAM によって管理されます。各データサイエンティストには独自の SageMaker ノートブックインスタンスが提供され、その特定のノートブックにのみアクセスできるようにすることが推奨されます。ノートブックインスタンスへのルートアクセスは、無効にしておきます。Jupyter ノートブックインスタンスを開くには、ユーザーが CreatePresignedNotebookInstanceUrl の API コールにアクセスする必要があります。この API コールによって、Jupyter ノートブックサーバーへのウェブ UI アクセスを取得する際に使用される署名付き URL が作成されます。このインターフェイスを保護するには、IAM ポリシーステートメントを使って、誰がどの IP アドレスからその API を呼び出せるかといった API の呼び出しに関する条件をラッピングします。

各組織によって認証とアクセスの要件はさまざまだと思われますが、アクセス権限は IAM ベストプラクティスと内部ポリシーおよびコントロールに従って、最小限の権限を付与するか、または特定タスクの実行に必要なアクセス権限のみを付与して、設定する

必要があります。ロールベースのアクセスコントロール (RBAC) は、金融サービスのお客様がよく使用している手法で、承認された当事者のみに、希望するシステムコントロールへのアクセスを可能にするものです。職能ポリシーに基づくロール作成、およびユーザーにアタッチされた IAM ポリシーの AWS Config を使用したモニタリングと定期監査は、経時変化する設定を表示するためのベストプラクティスです。Millennium Management: Secure Machine Learning using Amazon SageMaker では、Millennium Management が IAM を活用してコンプライアンスを強化した詳細をご紹介しています。AWS における金融サービス向けのセキュリティのベストプラクティスについての詳細は、Financial Services Industry Lens を参照してください。

データの暗号化

機械学習環境には、機密データや知的財産が含まれていることがあります。安全な ML 環境のための第 3 の考慮事項はデータの暗号化です。保管時と転送時の両方で、データの暗号化を有効にしておくことが推奨されます。保管時の暗号化では、たとえば Amazon S3 に保存されたデータや、SageMaker ノートブック内の Amazon Elastic Block Store (Amazon EBS) ボリュームに保存されたデータの場合、ほとんどの金融サービスのお客様は、カスタマーマネージド型のカスタマーマスターキー (CMK) の使用が必要になります。AWS KMS での CMK の詳細については、AWS Key Management Service concepts を参照してください。また、新しいオブジェクトを Amazon S3 バケットに暗号化して保存されるように、Amazon S3 のデフォルトの暗号化を有効にしておくことが推奨されます。さらに、Amazon S3 バケットポリシーを使用して、暗号化されていないオブジェクトがアップロードされないようにしておくことも推奨されます。転送時のデータについては、転送時のすべてのネットワーク間のデータ通信において TLS 1.2 による暗号化をサポートしています。最後に、分散トレーニング中にインスタンス間で送信されるデータについては、コンテナ間のトラフィックの暗号化を有効にすることが可能です。ただし、それによって、特に分散型深層学習アルゴリズムでは、トレーニング時間が長くなる場合があるので注意が必要です。

可監査性

適切に管理された安全な ML 環境のための第 4 の考慮事項は、モデル設定やハイパーパラメータの変更など、データやモデルへのすべてのアクセスおよび変更をログに記録する堅牢かつ透過的な監査証跡を維持することです。AWS CloudTrail は、AWS インフラストラクチャでのアクションに関連したアカウントアクティビティをログに記録し、継続的にモニタリングし、保持するサービスです。CloudTrail では、あらゆる AWS API コールをログに記録し、AWS マネジメントコンソール、AWS SDK、コマンドラインツールなどの AWS のサービスを通じて実行されたアクションを含む AWS アカウントアクティビティのイベント履歴が作成されます。さらに AWS Config を使用すると、AWS リソースの設定変更を継続的に監視、保持できます。

さらに広義では、AWS はログ記録機能と監査機能に加え、セキュリティに対する多重防御アプローチで、アプリケーションと環境のあらゆるレベルにセキュリティを適用することを推奨します。AWS CloudTrail と AWS Config は、セキュリティ上の潜在的な脅威やインシデントを検出する発見的コントロールとして使用できます。発見的統制によって潜在的驚異を検出すると、セキュリティインシデントの潜在的な影響に対応して緩和するための修正コントロールを設定することができます。Amazon CloudWatch は、CloudWatch Events のトリガーとなってセキュリティ対応を自動化できる AWS リソースのモニタリングサービスです。発見的統制および修正コントロールの詳細については、Amazon SageMaker Security Workshop を参照してください。

B. ML ガバナンスの確立

ML ガバナンスには 4 つの重要な側面があります。

- トレーサビリティと可監査性
- 説明可能性

- リアルタイムのモデルモニタリング
- 再現性

金融サービス業界では、コンプライアンスや規制に関するさまざまな義務を負います。これらの義務は、ML のプロセスに関わっている法律顧問、コンプライアンス担当者、さまざまなステークホルダーと共同でレビューし、理解する必要があります。たとえば、Jane Smith 氏が銀行から融資を断られた場合、貸手は規制要件を満たすために、この決定が下された経緯について説明を求められることがあります。貸手が融資審査の一環として ML を使用している場合は、上記の要件を満たすために ML モデルが行った予測の解釈または説明が必要になることがあります。一般に、ML モデルの解釈可能性または説明可能性とは、予測に到達するまでにモデルが使用したプロセスを理解して説明できる能力のことをいいます。多くの ML モデルは回答それ自体ではなく、あり得る回答の予測を行っているという点にも注意する必要があります。こうしたモデルによる予測は、アクションが実行される前に、トレーニングを受けた人によるレビューを追加することが妥当な場合があります。また、予測の根拠となったデータが変更されたときは新しいデータによって定期的に再トレーニングされるように、モデルをモニタリングすることが必要になることもあります。最後に、ML モデルは、アウトプットに至るステップを再トレースしたときにアウトプットが変化しないように再現可能である必要があります。次セクションでは、ML のガバナンス体系を構築する際に考慮すべき事例を紹介し、選択したガバナンス体系をオペレーショナライズする方法について解説します。

トレーサビリティ

効果的なモデルガバナンスには、すべてのモデル開発イテレーションを継続的に追跡することに加え、モデリングプロセスで使用されるデータおよびデータ変換を、詳しく理解していることが要求されます。どのデータセットが使用されたか、どのタイプの変換が行われたか、どのタイプのモデルが構築されたか、そしてそのハイパーパラメータについて把握しておくことが重要です。さらに、後処理ステップ (バッチ推論中に予測から

バイアスを削除するバッチジョブなど) を追跡する必要があります。モデルを利用してオンラインで推論する場合は、モデルパフォーマンスとモデル予測の追跡が必要になります。

機械学習ワークフロー内では、さまざまなステップで生データに対して、またはデータレイクの一部であるデータセットに対して変換が適用されています。この変換は、まずデータ探索から始まり、データポイントの欠落部分、重複部分、一貫性がない部分が特定され、処理されます。続いて、データの前処理および特徴量エンジニアリングでは、新しい特徴量の作成、正規化、標準化の処理をします。最後に、さまざまなフレームワーク、ML モデル、パラメータ化によるイテレーションに先だって、前処理されたデータを 3 つのセット (トレーニング、テスト、検証) に分割して終了します。データセット全体の不均衡性の有無によって、層化サンプリング、オーバーサンプリング、アンダーサンプリングなどの考慮事項が追加で必要になる場合があります。適切に管理された機械学習ワークフローでは、データに適用されたすべての変換および変換後のデータ出力の系列が明確にわかります。下図では、データとモデルの系列を追跡するための主な構成要素が示されています。

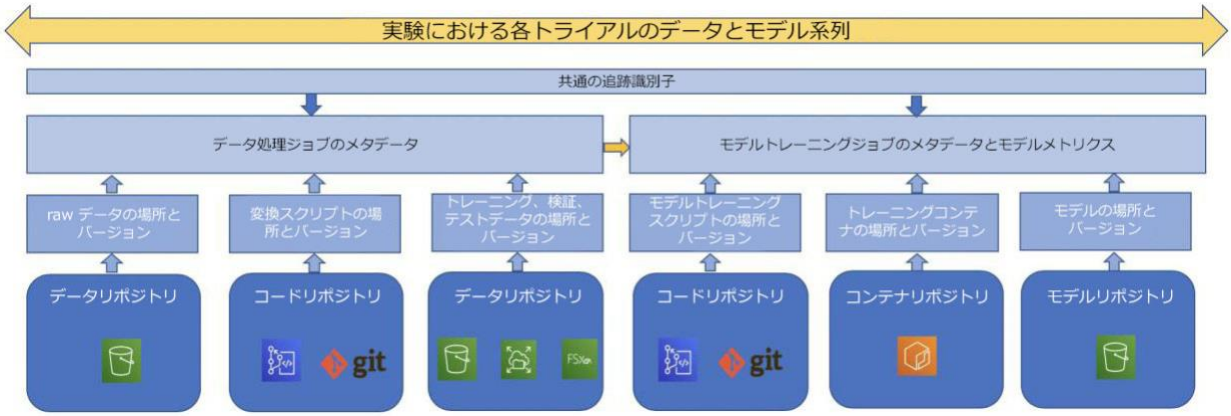


図 4 - データとモデルの系列

Amazon SageMaker Experiments を使用すると、ML アーティファクトを追跡し、データセット、アルゴリズム、ハイパーパラメータ、そしてメトリクスまでの系列を明確化でき、データの事前処理やトレーニング後のモデル評価といったその他の ML ワークフロー

ステップにも対応できます。Git リポジトリを使用すると、最新のコミット、ユーザー IP、コミット ID、各モデルバージョンのタイムスタンプを取得できるため、実験にモデルのバージョニングを追加できます。イミュータブルなスナップショットを保持し、それによって将来の後方互換性を確保するために、必要に応じてすべてのライブラリの依存関係を Amazon S3 にダウンロードし、バージョニングされるようにすることも可能です。さらに、SageMaker Experiments を使えば、すべての系統のメタデータが暗号化された方法で保存されるため、安全な仕方で現在および過去の実験を検索、閲覧し、実験を比較したり、最優秀モデルを識別したりすることが可能になります。

アーティファクトを追跡するプロセスは単純明快です。まず、以下のコードスニペットに示すとおり、ML ワークフローのすべてのコンポーネントおよびその内部のイテレーションを追跡するための実験を作成します。SageMaker Experiments は、データサイエンスの作業を整理する手段として捉えます。実験を作成すると、次のモデル開発作業をすべて組織化できます。

- 対応中のビジネスユースケース (下の例に示す「CreditCardDefault」という名前の実験を作成するなど)
- その実験を所有しているデータサイエンスチーム (「CreditCardTeam-ProjectName」という名前の実験を作成するなど)
- 特定のデータサイエンスと ML プロジェクト。これは「ファイル」を整理するための「フォルダ」と考えてください。

次に、SageMaker で実験を作成するスクリプトの例を紹介します。

```
cc_experiment = Experiment.create(
    experiment_name=f"CreditCardDefault-{int(time.time())}",
    description="Predict credit card default from payments data",
    sagemaker_boto_client=sm)
print(cc_experiment)
```

これで、前処理/特徴量エンジニアリングの各ステップから、トレーニングデータの場所、検証データの場所、トレーニング/テストの分割比率などのパラメータの追跡を開始できます。後者では、正規化平均、正規化標準偏差、データがスケーリングやワンホットエンコーディングなどの特徴量エンジニアリングステップを通ったか否かを示すさまざまな指標を追加することも可能です。

次は、前処理パイプラインで使用されるパラメータを追跡するためのスクリプトの例です。

```
with Tracker.create(display_name="Preprocessing", sagemaker_boto_client=sm) as
tracker:
    tracker.log_parameters({
        "train_test_split_ratio": 0.2
    })
    # we can log the s3 uri to the dataset we just uploaded
    tracker.log_input(name="ccdefault-raw-dataset", media_type="s3/uri",
value=raw_data_location)
    tracker.log_input(name="ccdefault-train-dataset", media_type="s3/uri",
value=train_data_location)
    tracker.log_input(name="ccdefault-test-dataset", media_type="s3/uri",
value=test_data_location)
```

続いて、トライアルを作成して各トレーニングジョブの追跡を開始します。前処理ジョブから得たパラメータでトライアルを強化するには、トライアルのコンポーネントをインスタンス化したトラッカー上で作成し、それをトライアルに追加する必要があります。次のコードスニペットでは、トライアルに **cc-fraud-training-job-** という名前を付け、タイムスタンプを加えて、TrialComponent として「前処理」ジョブおよびそのパラメータを追加しています。各トレーニングのトライアルでは次に示すとおり、model fit 中に、使用されたモデルの Docker イメージ、モデルアーティファクトの場所の出力バケット、ハイパーパラメータが SageMaker Experiments によって追跡されます。

```
account = sess.boto_session.client('sts').get_caller_identity()['Account']
image = '{}.dkr.ecr.{}.amazonaws.com/sagemaker-xgboost:latest'.format(account,
region)
```

```
output_bucket = 'output-bucket-name'
preprocessing_trial_component = tracker.trial_component

trial_name = f"cc-fraud-training-job-{int(time.time())}"
cc_trial = Trial.create(
       trial_name=trial_name,
       experiment_name=cc_experiment.experiment_name,
       sagemaker_boto_client=sm
    )

cc_trial.add_trial_component(preprocessing_trial_component)
cc_training_job_name = "cc-training-job-{}".format(int(time.time()))

xgb = sagemaker.estimator.Estimator(image,
                                    role,
                                    train_instance_count=1,
                                    train_instance_type='ml.m4.xlarge',
                                    train_max_run=86400,

output_path='s3://{}/{}/models'.format(output_bucket, prefix),
                                    sagemaker_session=sess,
                                    train_use_spot_instances=True,
                                    train_max_wait=86400,
                                    subnets = ['subnet-0f9914042f9a20cad'],
                                    security_group_ids = ['sg-089715a9429257862'],
                                    train_volume_kms_key=cmk_id,
                                    encrypt_inter_container_traffic=False) # set to
true for distributed training

xgb.set_hyperparameters(max_depth=5,
                        eta=0.2,
                        gamma=4,
                        min_child_weight=6,
                        subsample=0.8,
                        verbosity=0,
                        objective='binary:logistic',
                        num_round=100)

xgb.fit(inputs = {'training':'s3://' + train_data_location},
      job_name=cc_training_job_name,
       experiment_config={
           "TrialName": cc_trial.trial_name,
           "TrialComponentDisplayName": "Training",
       },
```

```
        wait=True,
    )
time.sleep(2)
```

トライアル内のすべてのコンポーネントとパラメータをレビューするために、次のコード例に示すとおり、SageMaker Experiments によってトライアル名に基づいて検索を行うように指示されます。

```
# Present the Model Lineage as a dataframe
sagemaker.session import Session
sess = boto3.Session()
lineage_table = ExperimentAnalytics(
    sagemaker_session=Session(sess, sm),
    search_expression={
        "Filters":[{
            "Name": "Parents.TrialName",
            "Operator": "Equals",
            "Value": trial_name
        }]
    },
    sort_by="CreationTime",
    sort_order="Ascending",
)
lineagedf= lineage_table.dataframe()

lineagedf
```

ユーザーは特徴量エンジニアリングおよびトレーニングプロセスの各コンポーネントの詳細を追跡し、両者を次のデータフレームのように表示できます。すべてのワークフローアーティファクトが Amazon S3 に保存されたら、上書きやトレーサビリティの消失を回避するため、Amazon S3 のオブジェクトロックとバージョニングも有効にします。

	TrialComponentName	DisplayName	train_test_split_ratio	SourceArn	SageMaker.ImageUri	SageMaker.InstanceCount	SageMaker.InstanceType	SageMaker.VolumeSizeInGB	eta	gamma	max_depth	min_child_weight	num_round	objective
0	TrialComponent-2020-03-16-201539-jwhc	Preprocessing	0.2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	cc-training-job-1584589765-aws-training-job	Training	NaN	arn:aws:sagemaker:us-east-1:[redacted]:training-job/cc-training-job-1584589765	[redacted].dkr.ecr.us-east-1.amazonaws.com/sagemaker-xgboost:latest	1.0	ml.m4.xlarge	30.0	0.2	4.0	5.0	6.0	100.0	binary:logistic

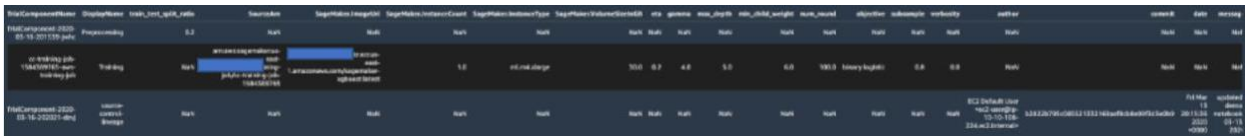
同様に、モデル追跡機能を使うと、特定のデータセットが監査のトレーニングジョブで使用されたことを検証したり、コンプライアンスを検証したりできます。これを実行するには、Amazon S3 内の場所を示す URL を検索します。モデル追跡機能によって、指定し

たデータセットを使用しているトレーニングジョブが返されます。検索でデータセットが返されなかった場合 (結果が空の場合)、そのデータセットはトレーニングジョブで使用されていません。結果が空の場合は、たとえば提示されたデータセットが使用されていないことが確定します。

```
def get_codecommit(commit_id):
    codecommitclient = boto3.client('codecommit')

    reponame =
codecommitclient.list_repositories()['repositories'][0]['repositoryName']

    return codecommitclient.get_commit(repositoryName=reponame,
    commitId=commit_id
    )

# Below you will need to navigate to CodeCommit to obtain the corresponding commit
IDs if you choose to commit your code.
# If you only commit your code once, then use the same repo name and CommitIDs for
sclineage and endpointlineage.

sclineage = get_codecommit('1796a0a6972c8df97fd2d279557a6f94cfe91eae') #
Enter your CommitID here

endpointlineage = get_codecommit('a6df1799849f52295864754a8f1e30e604fefd00')
# Enter your CommitID here

with Tracker.create(display_name="source-control-lineage",
sagemaker_boto_client=sm) as sourcetracker:
    sourcetracker.log_parameters({
        "commit": sclineage['commit']['commitId'],
        "author":sclineage['commit']['author']['email'],
        "date":sclineage['commit']['author']['date'],
        "message":sclineage['commit']['message'].replace('-', '_').split('\n')[0]
    })


with Tracker.create(display_name="prod-endpoint-lineage", sagemaker_boto_client=sm)
as endtracker:
    endtracker.log_parameters({
        "commit": endpointlineage['commit']['commitId'],
        "author":endpointlineage['commit']['author']['email'],
        "date":endpointlineage['commit']['author']['date'],
        "message":endpointlineage['commit']['message'].replace('-',
'_').split('\n')[0]
```

```
    })
    endtracker.log_input(name="endpoint-name", value='cc-training-job-
    1583551877')

cc_trial.add_trial_component(sourcetracker.trial_component)
cc_trial.add_trial_component(endtracker.trial_component)

# Present the Model Lineage as a dataframe
from sagemaker.session import Session
sess = boto3.Session()
lineage_table = ExperimentAnalytics(
    sagemaker_session=Session(sess, sm),
    search_expression={
        "Filters":[{
            "Name": "Parents.TrialName",
            "Operator": "Equals",
            "Value": trial_name
        }]
    },
    sort_by="CreationTime",
    sort_order="Ascending",
)
lineagedf= lineage_table.dataframe()

lineagedf
```

モデル系列を追跡するときは、関連するエンドポイントからモデルの名前を追跡します。トライアルコンポーネントのいずれかの名前を使用することもできます。次に、いずれかの名前を使って、上記のコードスニペットの最後に記されているとおり、TrialComponentName API で、デプロイされたエンドポイントに含まれるトレーニング、パラメータ、前処理ジョブ、モデルのバージョンを検索します。以下は、系列のさまざまなコンポーネントのすべてを含むデータフレームです。重要な注意点として、最終行には、デプロイされたモデルバージョンのコミット ID、タイムスタンプ、作成者、コミットメッセージが表示されています。



再現性プロセスのシンプル化には、Amazon SageMaker Studio が役立ちます。新たにノートブック共有機能が追加され、各チームメンバーは簡単に作業を共有でき、チームメートはモデルをすばやく再現できるようになりました。

説明可能性と解釈可能性

ロジスティック回帰やデシジョンツリーなどの一部のモデルはきわめてシンプルかつ明快で、モデルがどのようにしてそのアウトプットを見つけだしたかをより簡単に理解できます。しかし、特徴量がさらに追加されたり、より複雑な機械学習モデルが使用されると、解釈可能性はさらに難しくなります。金融サービスにおける意思決定に関連したアルゴリズムの予測を使用する場合は、モデルがどの特徴量を考慮したか、または考慮しなかったかを知ることが重要です。場合によっては、アクションの実行前に、適切な人がその予測をレビューすることが重要になります。金融サービス業界においては、モデルユーザーは、モデルの結果に寄与した要因を理解するなど、各モデルの限界、意図、アウトプットを理解しておく必要がある場合があります。

機械学習モデルの説明に役立つ方法はいろいろあります。Partial Dependency Plot (部分従属プロット)、データセット全体 (グローバル) またはデータのサブセット (ローカル) のいずれかでの代理モデルの構築、Quantitative Input Influence (量的入力の影響)、LIME (Local Interpretable Model-Agnostic Explanation)、SHAP (SHapley Additive exPlanations) などがその一例です。ニューラルネットワークモデルの説明可能性は現在も重要な研究テーマであり、このテーマの学習を継続することが重要です。

機械学習における説明可能性については今でも研究がアクティブに行われていますが、SHAP は特徴量重要度を抽出するためのポピュラーな統一的方法として登場しました。ここでは、データセットの文脈から詳しく説明します。SHAP の目標は、特徴量を含むものと含まないものの 2 つのモデルを生成して予測に対する各特徴量の寄与を計算することで、予測を説明することにあります。所定のサンプルにおける予測の差には、特徴量重要度が関係しています。SHAP 値は、特徴量がモデルに追加される順序のすべての配

列を平均化し、特徴量の相互依存性を考慮することによって、系列モデルまたはローカルの独立した特徴量を超えて、この考え方を拡張します。

SageMakerを使用すると、LightGBMやXGBoostなどの一般的なモデルや単純なディープラーニングモデルのSHAP値を取得することができます。まず、SHAP ライブラリが、ローカルの Jupyter 作業環境にあらかじめインストールされていることを確認します。このライブラリは、ライフサイクル設定スクリプトを使って、最初からデータサイエンティスト向けにプロビジョニングすることが可能です。データサイエンティストがモデルトレーニングと開発に使用する開発 VPC がインターネットアクセスできないように設定されている場合には、ライブラリのダウンロードはローカルの PyPI サーバーからダウンロードするか、共有サービスアカウントなどインターネットアクセスが有効になっている別の VPC から pip ミラーを使用します。

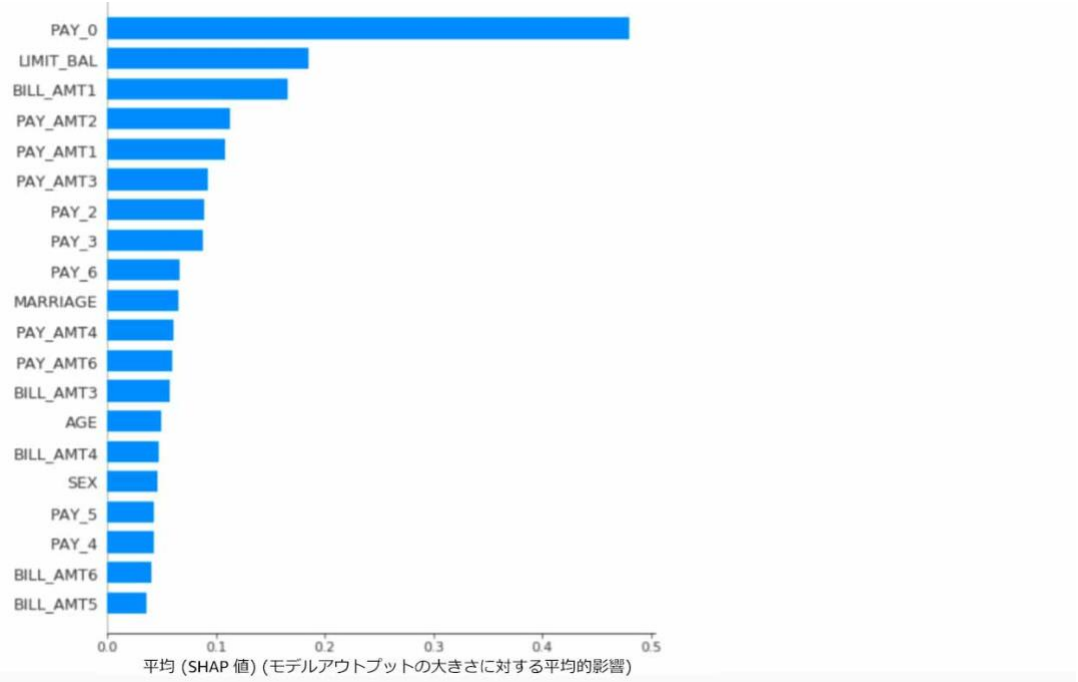
インポートが完了すると、トレーニングされたモデルオブジェクトを Amazon S3 からローカル環境にコピーできるようになります。次の関数は、SageMaker Experiments Trail を呼び出して、pickle 化されたモデルオブジェクトを解凍します。トレーニングコンテナがここで、トレーニング済みの XGBoost モデルオブジェクトを xgboost.pkl という pickle ファイルとして保存します。

```
def download_artifacts(job_name, local_fname):
    ''' Given a trial name in a SageMaker Experiment, extract the model
file and download it locally'''
    sm_client = boto3.Session().client('sagemaker')
    response =
sm_client.describe_trial_component(TrialComponentName=job_name)
    model_artifacts_full_path =
response['OutputArtifacts']['SageMaker.ModelArtifact']['Value']

    p = re.compile('(?<=s3://).*?/')
    s = p.search(model_artifacts_full_path)
    object_name_start = s.span()[1]
    object_name = model_artifacts_full_path[object_name_start:]
    bucket_name = s.group()[:-1]
    s3 = boto3.client('s3')
    s3.download_file(bucket_name, object_name, local_fname)
```

```
def unpack_model_file(fn):
    # Unpack model file
    _tar = tarfile.open(fn, 'r:gz')
    _tar.extractall()
    _fil = open('xgboost-model', 'rb')
    _model = joblib.load(_fil)
    print(_model)

    return _model
```

モデルオブジェクトが抽出されたら、トレーニングデータセット全体またはデータセットのサブセットで、SHAP ライブラリを使って特徴量重要度を計算し、プロットできます。以前のアプローチでは、データセット全体にわたるグローバルな特徴量重要度を取得していますが、ローカル重要度の抽出も可能であり、リスク管理とモデルガバナンスの詳細レポートの一部として公開できます。たとえば、顧客がクレジットカードの支払いを履行しない可能性が高いとモデルが予測した場合、この判断の根拠となった特徴量を識別してモデルの判断を破棄し、必要であれば修正措置を講じなければならない場合があります。

```
shap.summary_plot(shap_values, traindata.drop(columns =['Label']))
```



Gradient Boosted Trees のようなツリーベースのモデルでは、ある特徴量による特定の分割がノード上で分割された観測数によって重み付けされ、モデルの予測精度に影響しているか否かの要因と見なすことによって、特徴量の相対的重要度がわかります。このジニ不純度メトリクスも、すべてのツリーで平均化し、特徴量重要度の初期メトリクスとして使用できます。特徴量重要度を計算するために、複数の異なるアプローチを報告することは珍しくありません。このメトリクスは XGBoost API からネイティブに取得できます。SHAP 値のように他のツリーベース以外のモデルに一般化されることはないため、これ以上の説明は省きます。

SageMaker で最近リリースされたフルマネージド型の Amazon SageMaker Debugger では、ポストトレーニングの特徴量重要度を識別する SHAP 値を手動で使用することに加え、深層学習モデルのテンソルの可視化によってトレーニング中のモデルをデバッグしたり、トレーニングジョブをチェックするための組み込みルールやカスタムルールを評価したり、エラーを発見したりできます。SageMaker Debugger を使用すれば、モデルトレーニングのイテレーション中に特徴量重要度や SHAP 値をログに記録できるため、Amazon SageMaker Studio を使って可視化も可能です。SageMaker Debugger の詳細については How Debugger Works を参照してください。これまで運用の観点から SHAP について説明してきましたが、説明可能性は非常に新しい発展途上の領域であるため、SHAP のようなモデルの説明可能性における有効性は今も議論が続いているということを最後に申し上げておきます。

説明可能性に加え、公正性と安全性の考慮事項も同じように重要です。これについて次に簡単に説明します。公平性は非常に複雑かつ広範囲なテーマであり、完全な議論を行うことは本ホワイトペーパーの範囲を越えています。本書の文脈では、バイアスの評価およびモデル内のバイアスの理解という観点で公平性について説明します。トレーニングデータが公平かつ一般的であることを保証するには、相当な労力が必要です。トレーニングに使用する各データセットは、バイアス検証が必要になります。企業は、モデルバイアスポリシーとバイアスチェックの方法を運用し、定期的に結果をモニタリングします。バイアスにはさまざまなタイプがあります。以下に例を紹介します。

- **サンプリングバイアス** - トレーニングデータが現実のシナリオを正確に表していないときに発生します。サンプルバイアスは、すべての潜在シナリオ上でモデルをトレーニングすれば削減、除去できます。
- **除外バイアス** - 通常はデータのクリーニングのもとで、データセットから一部の特徴量が除外された結果として発生します。これは、デベロッパーによるデータ理解によって特徴量が削除された結果として発生します。除外バイアスは、特徴量を除去する前に適切な調査を行えば削減、除去できます。また、除去が予定されている特徴量については、ビジネスの内容領域専門家 (SME) の同意を得ておけば、削減、除去できます。
- **文化的またはステレオタイプのバイアス** - 外見、社会階層、ステータス、性別などの問題に関連するバイアスです。ある種のタイプのバイアスは、文化的、ステレオタイプ的に偏りのある結果を把握して回避することで、減らせる可能性があります。前の例では、職業と性別との統計的関連性を無視することが含まれている可能性があります。こうしたバイアスを減らす重要な要素としては、これらの問題を学習して認知度の高い多様性のあるチームを使うことや、文化的、ステレオタイプ的バイアスをミニマイズする適切なデータセットでモデルをトレーニングすることなどが含まれます。
- **測定バイアス** - 観察または測定に使用したデバイスに問題がある場合に発生します。系統的な値の歪みは、デバイスの問題に起因して発生します。測定バイアスを削減、除去するには、複数のデバイス使用によるデバイスの歪みの回避や、人によるデータチェックまたは Amazon Mechanical Turk によるデータチェックを導入してデバイスのアウトプットを比較するなどが考えられます。

組織のリスク、法務、コンプライアンスのチームと協力し、ML システムの構築と使用に関連した、法律、倫理、規制、コンプライアンスの要件および影響を評価することが必要になります。これには、システム内のデータとモデルのいずれか、または両方を使用する法的権利の審査、プライバシー、生体認証、差別防止、金融サービスのユースケース固有のさまざまな考慮事項といった問題をカバーする法律の適用可能性の判断などが含まれます。

モデルモニタリング

モデルモニタリングとは機械学習モデルを本番環境にデプロイした後に続く次のフェーズで、データサイエンスプロジェクトのライフサイクルに関するものです。モデルモニタリングでは実際の結果に照らしてモデルパフォーマンスが測定されるため、デプロイされたモデルが実際の環境で期待どおりに機能していることを保証する一助となります。モデルモニタリングには、関連する規制ガイドラインが存在することがあります。モデルパフォーマンスのデータは、監査レポートでよく使用されます。また、適切な緊急時対応策を実施できるようにするための早期警戒信号としても使用されることがあります。モデルの再トレーニングやモデル廃止のための、ガイドラインとして使用することも可能です。

テスト環境ですでに検証済み、実証済みであるのに、なぜモデルモニタリングが必要なのかと疑問に思う方がいるかもしれません。その理由は、平均値、中間値や相関関係といったデータの統計的性質が徐々に変化したり、独立変数と従属変数の統計的関係が経時変化したりしたとき、機械学習モデルのパフォーマンスが低下する可能性があるためです。前者はデータドリフト、後者はコンセプトドリフトと呼ばれます。ドリフトはさまざまなソースから発生するため、その検出方法もさまざまです。その 1 つが、トレーニングデータと新規データ間でスキーマと計算された統計分布を比較する方法です。平均偏差シフトや標準偏差シフトといった変化、またはデータフィールドタイプの変化は、データドリフトの指標となります。予測分布は、モデルのドリフトを検出する方法として、経時的に監視および計算することも可能です。さらに、正解 のラベルを新しいデータで収集できる場合は、予測結果を 正解 と比較することも、モデルパフォーマンスの低下をモニタリングする方法となります。

SageMaker によるデータドリフト検出

データドリフト検出は、エンジニアリングとサイエンスの難しい課題です。モデルが受け取ったデータをキャプチャし、あらゆる種類の統計分析を実行してそのデータをトレーニングセットと比較し、検出のルールを定義して、ドリフトが発生したときはア

ラートを送信することが必要になります。そして新しいモデルをリリースするたびに、これらのステップを繰り返す必要があります。SageMaker Model Monitor を使うと、次のステップ/機能により、データドリフトの検出をシンプル化できます。

1. **受信リクエストデータとモデルアウトプットのキャプチャ** - SageMaker Model Monitor によって、受信リクエストデータと予測出力がキャプチャされます。キャプチャされたデータは Amazon S3 バケットに保存され、コンテンツタイプやタイムスタンプなどのメタデータが付与されます。ユーザーは、キャプチャするデータの量を設定できます。

```
from sagemaker.model_monitor import DataCaptureConfig
endpoint_name = 'DEMO-xgb-churn-pred-model-monitor-' + strftime("%Y-%m-%d-%H-%M-
%S", gmtime())
print("EndpointName={}".format(endpoint_name))

data_capture_config = DataCaptureConfig(
                        enable_capture=True,
                        sampling_percentage=100,
                        destination_s3_uri=s3_capture_upload_path)

predictor = model.deploy(initial_instance_count=1,
                instance_type='ml.m4.xlarge',
                endpoint_name=endpoint_name,
                data_capture_config=data_capture_config)
```

2. **モニタリングベースラインの作成** - SageMaker Model Monitor によって、モデルのトレーニングに使用されたデータを使ってベースラインを作成できます。各特徴量の種類や完全性といった入力データのスキーマを推測し、ベースライン計算の一環として特徴量統計を計算します。

```
from sagemaker.model_monitor import DefaultModelMonitor
from sagemaker.model_monitor.dataset_format import DatasetFormat

my_default_monitor = DefaultModelMonitor(
    role=role,
    instance_count=1,
    instance_type='ml.m5.xlarge',
    volume_size_in_gb=20,
```

```
    max_runtime_in_seconds=3600,
)

my_default_monitor.suggest_baseline(
    baseline_dataset=baseline_data_uri+'/training-dataset-with-header.csv',
    dataset_format=DatasetFormat.csv(header=True),
    output_s3_uri=baseline_results_uri,
    wait=True
)
```

3. **エンドポイントのモニタリング** - SageMaker Model Monitor を使用すると、モニタリングスケジュールを設定して、収集されたデータをベースラインに照らしてインスペクションできます。結果に適用される違反チェックなど、組み込みルールが多数あり、その結果は Amazon S3 にプッシュされます。

結果には、最新のタイムフレーム中に受信されたデータに関する統計およびスキーマ情報のほか、検出された違反が含まれています。また、SageMaker Model Monitor によって Amazon CloudWatch に特徴量ごとのメトリクスを出力されるため、アラートの設定に使用できます。サマリーメトリクスは SageMaker Studio でも提供されており、入手は簡単です。レポートとアラートは、データ品質問題をすばやく検出する一助となるため、問題があれば必要な修正措置を講じて対処することが可能です。

SageMaker によるモデルドリフト検出

データドリフト検出は、モデルパフォーマンスの低下に対抗する防御の一番手です。新しいデータで 正解 のラベルを収集できる場合は、SageMaker バッチ変換を使ってモデルメトリクスの計算用にバッチ推論を定期的に実行できます。次に、SageMaker バッチ変換を使ってモデルメトリクスを実行および計算するステップについて説明します。

1. **データの準備** - ラベル付きのデータセットを使ってモデル特徴量ベクトルを準備し、入力から実際のラベルを削除して、リクエストファイルを Amazon S3 バケットに保存します。
2. **バッチ変換の実行** - SageMaker のターゲットモデルに対して SageMaker バッチ変

換を実行します。出力ファイルは Amazon S3 バケットに保存されます。

3. **コンピューティングメトリクス** - 出力ファイルとオリジナルの 正解 ラベルを使ってモデル評価スクリプトを実行し、新しいメトリクスを計算してオリジナルの検証メトリクスと比較します。モデルパフォーマンスメトリクスの変化が定義済みのしきい値を超えている場合は、修正を検討する必要があります。

以上のステップを実行する方法として、さまざまな選択肢が多数存在します。たとえば、AWS Lambda 関数を使用すれば、データの準備、バッチ変換の実行、モデルメトリクスの計算、つまり潜在的なモデルドリフトの計算が可能です。

再現性

再現性とは、プロセスに関わるランダム性を減らしながら、同じ ML モデルを後日再作成するか、または異なるステークホルダーによって再作成する能力を指します。お客様は規制要件を満たすために、エンドツーエンドの開発プロセスを再現および検証できて、なおかつ最終モデルの作成に使用された必要なアーティファクトおよび方法を提出できることが必要になる場合があります。モデルの再現性に関する詳細な解説は本書の範囲を越えています。ここでは、SageMaker を使用する際にモデル開発に使用されるソースデータ、実際のモデル選択、モデルのパラメータ、最終モデルのアーティファクトを簡単に検出する方法に絞ります。SageMaker Experiments とお使いのバージョン管理システム (Git、CodeCommit、Artifactory) では、トレーサビリティのセクションで示されたとおり、デプロイされたモデルバージョンのすべての系列を追跡することが可能であるため、同じ前処理パイプラインを同じデータ上で再実行し、同じハイパーパラメータを持つ同じモデルを使って、そのモデルとほぼ同じものが取得できます。確率的勾配降下法を実装したオプティマイザーに必然的に伴うランダム性により、同じアーティファクト上のまったく同じプロセスで、わずかに異なるモデルが生成され、わずかに異なる予測結果が生じる場合があることは重要です。このケースでは、ランダム性はモデルのトレーニングを除くすべてのステップで完全に除去されていますが、まったく同じ入力、モデルタイプ、パラメータを使用しても、オプティマイザーのシードシーケンスと確率的特

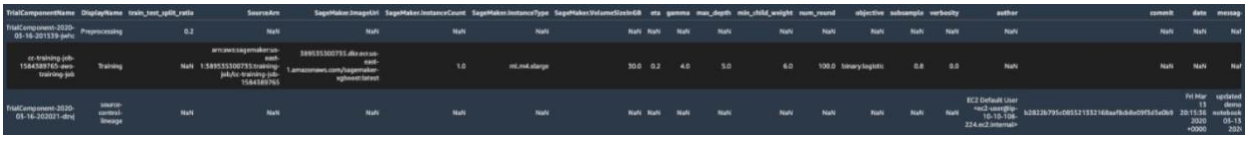
性により、最終モデルの最適な重みにわずかな差異が認められることがあります。

以下は、Git からのコミットを追跡するスクリプトの一例です。

```
def get_codecommit(commit_id):
    codecommitclient = boto3.client('codecommit')

    reponame =
    codecommitclient.list_repositories()['repositories'][0]['repositoryName']

    return codecommitclient.get_commit(repositoryName=reponame,
    commitId=commit_id
    )

# Below you will need to navigate to CodeCommit to obtain the corresponding
commit IDs if you choose to commit your code.
# If you only commit your code once, then use the same repo name and
CommitIDs for sclineage and endpointlineage.

sclineage = get_codecommit('1796a0a6972c8df97fd2d279557a6f94cfe91eae') #
Enter your CommitID here

endpointlineage = get_codecommit('a6df1799849f52295864754a8f1e30e604fefd00')
# Enter your CommitID here

with Tracker.create(display_name="source-control-lineage",
sagemaker_boto_client=sm) as sourcetracker:
    sourcetracker.log_parameters({
        "commit": sclineage['commit']['commitId'],
        "author":sclineage['commit']['author']['email'],
        "date":sclineage['commit']['author']['date'],
        "message":sclineage['commit']['message'].replace('-',
'_').split('\n')[0]
})


with Tracker.create(display_name="prod-endpoint-lineage",
sagemaker_boto_client=sm) as endtracker:
    endtracker.log_parameters({
        "commit": endpointlineage['commit']['commitId'],
        "author":endpointlineage['commit']['author']['email'],
```

```
        "date":endpointlineage['commit']['author']['date'],
        "message":endpointlineage['commit']['message'].replace('-',
'_').split('\n')[0]
    })
    endtracker.log_input(name="endpoint-name", value='cc-training-job-
1583551877')

cc_trial.add_trial_component(sourcetracker.trial_component)
cc_trial.add_trial_component(endtracker.trial_component)

# Present the Model Lineage as a dataframe
from sagemaker.session  import Session
sess = boto3.Session()
lineage_table = ExperimentAnalytics(
    sagemaker_session=Session(sess, sm),
    search_expression={
        "Filters":[{
            "Name": "Parents.TrialName",
            "Operator": "Equals",
            "Value": trial_name
        }]
    },
    sort_by="CreationTime",
    sort_order="Ascending",
)
lineagedf= lineage_table.dataframe()

lineagedf
```

上記のコードスニペットに示すとおり、モデル系列とモデル再現能力を獲得したいときは、前処理パラメータとコードバージョニングの両方で Tracker API の使用を実行する必要があります。モデル、パラメータ、データセットをネイティブに追跡する TrialComponent API に加え、お客様は、Tracker API が他のすべてのステップで常にログに記録されるようにしておくことが必要です。次に、Git/CodeCommit のベストプラクティスを検討し、分岐、コミット、適切なコメントに関連して Tracker に作成者、コミット ID、日付、コミットのメッセージを正確に含めるようにします。下図の右下のセクションを参照してください。



C. ML ワークロードのオペレーショナライズ

この最後のセクションでは、FSI ML ワークロードの運用に関するいくつかのベストプラクティスを説明します。最初におおまかな説明から始め、次に AWS ネイティブのツールやサービスを活用している具体的なアーキテクチャを紹介します。規制対象業界で ML モデルを本番環境にデプロイすることは、モデルのデプロイプロセス、つまり従来のソフトウェアデプロイで CI/CD (継続的インテグレーション/継続的デプロイ) と呼ばれていたものに加えて、運用の観点からさらなる影響を及ぼす場合があります。

下図は、企業の ML プラットフォームで対処の必要なガバナンス、監査、ログ記録、レポートに関するガイドラインのおおまかな要件の一部を示しています。

- 生データおよび関連するメタデータを管理するためのデータレイク
- 機械学習の特徴量および関連するメタデータを管理するための特徴量ストア (生データから、ワンホットエンコーディング、スケーリング変換といった生成された特徴量へのマッピングなど)
- トレーニング済みのモデルアーティファクトと関連するメタデータを含むモデルとコンテナレジストリ (ハイパーパラメータ、トレーニング時間、依存関係など)
- ソースコードの管理とバージョニングのためのコードリポジトリ (Git/CodeCommit/Artifactory など)
- トレーニングパイプラインおよびデプロイパイプラインをバージョニング、管理するためのパイプラインレジストリ
- アクセスログを管理するためのログ記録ツール
- 本番稼動のモニタリングとパフォーマンスのログ
- 監査とレポート報告のためのツール

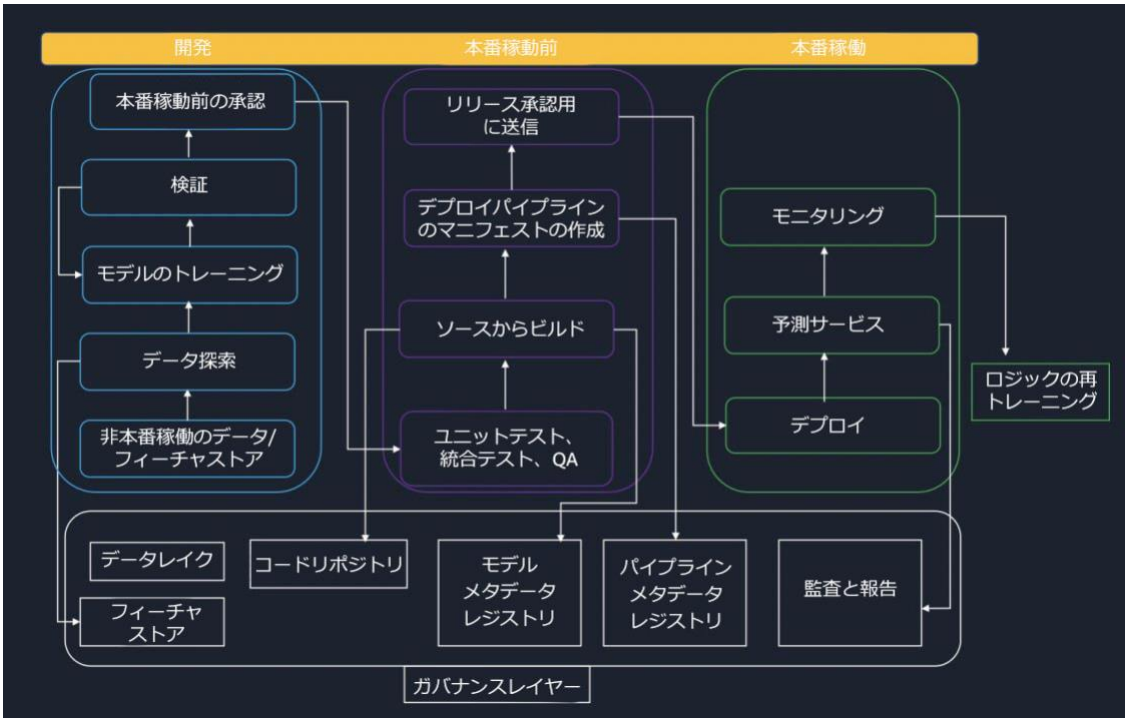


図 5 - ML DevOps ワークフロー

下図は、AWS ネイティブのツールとサービスを活用した具体的な実装を示しています。マーケットには Airflow や Jenkins といったスケジューリングツールやオーケストレーションツールが流通していますが、残りの部分では説明に具体性を持たせるために、AWS Step Functions に絞ります。

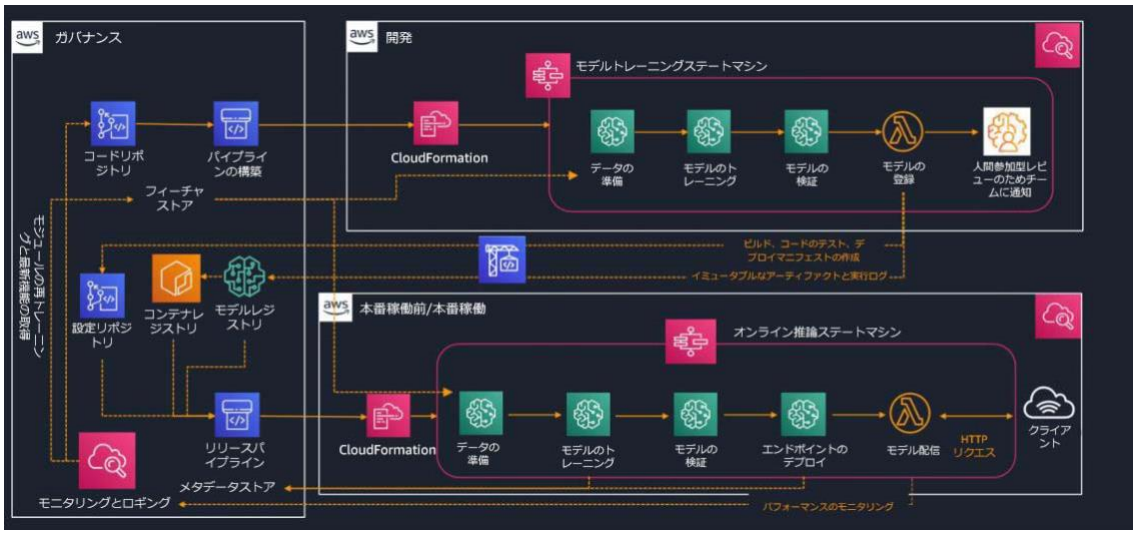


図 6 - AWS ネイティブツールを使用した ML DevOps

モデル開発のワークロード

まず開発環境から始めましょう。データサイエンティストはここでは、生データを分析および検証し、特徴量を生成し、トレーニング済みモデルを使って実験を行い、当該モデルをテストシナリオで検証します。これらは、ユースケースの具体的な特性に応じて、本番環境のデータを使っても使わなくても実行できます。本プロセスは直線的なものとして示されていますが、しばしば高度に循環的かつ実験的なプロセスとなります。

データサイエンティストは、AWS Step Functions Data Science SDK によって実験を高速化できます。このオープンソースライブラリを使用すると、新しい特徴量またはデータがトレーニングデータリポジトリにプッシュされたとき、または新しいモデルが追加されたときにトリガーされるトレーニングパイプラインを作成できます。AWS Step Functions は、パイプラインをデプロイするためのフルサーバーレスなオーケストレーションツールです。Data Science SDK は、ステートマシンの作成に使用できる Python SDK で、機械学習のパイプラインオーケストレーションのための便利で使いやすい API が用意されています。SDK は、AWS Lambda 関数を自動的に呼び出して、モデルコンテナのデプロイ、当該データを使ったモデルのトレーニング、エンドポイントの作成、バッチ変換ジョブによるオフラインテストの使用を実行し、モデルパフォーマンスをテストします。

図 7: 開発環境のエンドポイントの作成ワークフロー

次のコードには、Data Science SDK を使ってモデルトレーニングジョブを開始し、トレーニングアーティファクトからモデルを作成する方法が示されています。

```
training_step = steps.TrainingStep(
    'Train Step',
    estimator=xgb,
    data={
        'train': sagemaker.s3_input(train_s3_file, content_type='csv'),
        'validation': sagemaker.s3_input(validation_s3_file, content_type='csv')},
    job_name=execution_input['JobName']
)

model_step = steps.ModelStep
    ('Save
    Model',
    model=training_step.get_expected_model(),
    model_name=execution_input['ModelName']).
    result_path='$.ModelStepResults'
)
```

これまでのセクションで説明したとおり、SageMaker Experiments は、トレーニングに関連するメタデータとモデル系統の取得に使用できます。実験が完了したら、金融サービス関係の多くの企業では、手動または人間参加型プロセスを設けて、トレーニングされたモデルを検証、レビューし、それらがモデルの説明可能性および解釈可能性に関する要件を満たしていること、モデルリスクがあれば適切に記録されていることを確認し、さらには開発プロセスが適切に記録されているかどうかを内部監査で確認します。このレビューは、実際のモデル開発の実務に従事していない独立した立場の者が実施しなければならない場合もあります。

セキュリティとアクセス管理の観点からすると、開発環境にはほぼ手作業による人の介入が必要ですが、本番稼動のデータが使用されないため、制限は緩くなることがあります。ただし、開発環境で本番稼動データへのアクセスが必要となる場合は特に注意が必要です。アプローチの 1 つとしてマイクロアカウントを作成すると、承認されたチームとユーザーのみがアクセスを許可され、厳密な境界が適用されます。さらに、モデル開発のために本番稼動データにアクセス権を付与する前に、特別な承認ワークフローの作成が必要になることもあります。

本番稼動前のワークロード

モデルが適切にレビューされたら、DevOps チームによる本番前または本番稼動のデプロイに向けてモデルを渡す準備が整ったことになります。わかりやすくするために、ここでは本番稼動前環境と本番稼動環境を組み合わせていますが、これらは分けることも可能です。

DevOps チームは次のことを行います。

- モデルを既存のビジネスワークフローに組み込めること、およびモデルが必要な処理ロジックを構築できることを保証するための統合テスト。
- パフォーマンステストなどの QA テスト。過去の本番稼動データ (バックテスト) でのモデルパフォーマンスとトレーニングデータでのモデルパフォーマンスの比較などのQAテスト、または以前にデプロイされたモデルでの A/B テスト。
- モデルをさまざまなシナリオでテストするストレステスト。たとえば、取引アルゴリズムではモデルリスクを理解するために市場変動率が低いときと高いときとでテストする。

最後に、トレーニングされたモデルのソースコード、モデルの特徴量、モデルのアーティファクトと関連メタデータ、コンテナや関連するメタデータなどのビルドを、適切なレジストリに保存します。

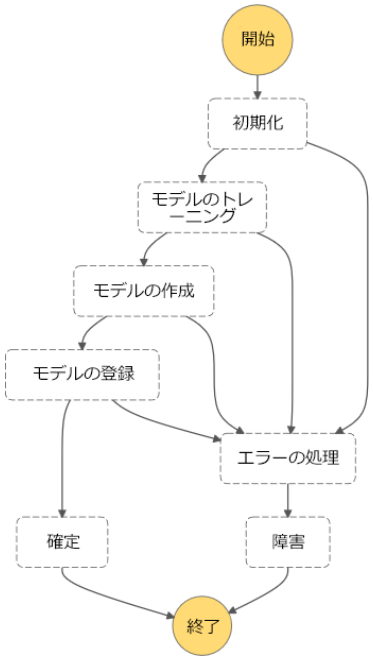


図 8 - モデルのテストワークフロー

```
check_accuracy_step = steps.states.Choice(
    'Accuracty > 90%'
)
```

AWS CodePipeline は、テスト済みモデルをピックアップするマニフェストの作成に使用できます。モデルを本番稼動環境に移行させるには人による承認フローが必要です。AWS CodePipeline によって、AWS CloudFormation などの Infrastructure as Code ツールと統合し、モデルのデプロイに必要なリソースがプロビジョニングされます。

本番稼動ワークロードおよび継続的モニタリングワークロード

本番稼動パイプラインの準備ができたら、AWS CodePipeline は、基盤となるインフラストラクチャの作成に必要な AWS CloudFormation スクリプト、およびモデルを本番稼動のエンドポイントにデプロイするために必要な AWS Step Functions ステートマシンを実行できます (図 9 参照)。AWS CodePipeline は、AWS CodeCommit からコードを取得することで、

AWS Step Functions ワークフローの起動にも使用できます。このワークフローは次に示すとおり、モデルを本番稼動環境にデプロイする際に必要になります。モデルが本番稼動前のステージで本番稼動用データを使用して検証済みの場合は、図 7 に示すとおりエンドポイントに直接デプロイできます。それ以外の場合は、本番稼動用データに対して 2 回目の再トレーニングステップが必要になる場合があります。これにより、トレーニングとデプロイされたモデルパフォーマンス間の不均衡を最小化できます。

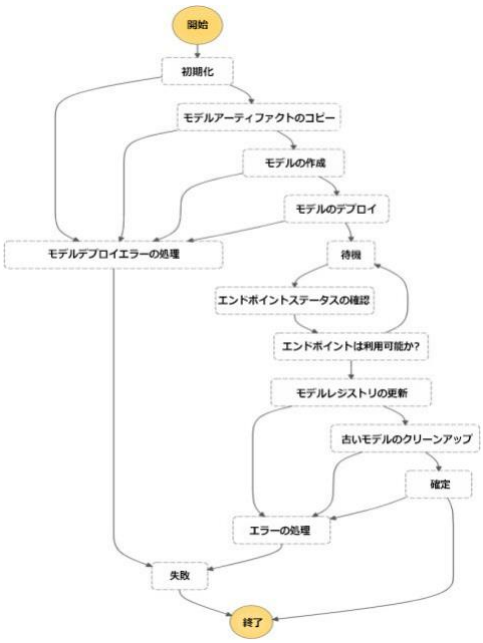


図 9 - AWS Step Functions ステートマシン

継続的なデプロイに加えて、MLワークフローのデプロイにおける重要要素はモニタリングです。これには、前半のセクションで説明したように、エンドポイントのパフォーマンスモニタリングと、モデル自体のドリフトのモニタリングの両方が含まれています。これまでのセクションでは、SageMaker Model Monitor を使って、デプロイされたエンドポイントに対してモデルモニタリングを確立する方法を説明しました。

継続的モニタリングがセットアップされたら、モデルを頻繁に再トレーニングし、モデルの再トレーニング用のトリガーを設定することが重要です。これには、データまたはラベル/コンセプトのドリフトのトリガー、モデルパフォーマンスのドリフトのトリガー、

特徴量ストアへの新規データまたは新規特徴量の追加などがあり、イベントドリブン型にも、スケジュール型にもできます。その後、AWS Lambda を使って上記の AWS Step Functions パイプラインをトリガーし、モデルを再トレーニングします。

お客様は、人間参加型ワークフローを使って、モデルアウトプットをレビュー担当者に定期的に送信することも可能です。レビュープロセスのアウトプットは、真の 正解 ラベルとして、トレーニングデータに再度組み込むことができます。SageMaker の Augmented AI 機能を使用すると、外部または内部のワークフォースを使って、人間参加型ワークフローを簡単に作成できます。

まとめ

本ホワイトペーパーでは、適切に管理された ML ワークフローをプロビジョニングし運用する際の考慮事項について説明してきました。ML ワークフローは、さまざまなステークホルダーが関わる組織的な取り組みです。組織内のコンプライアンスチームと協力し、特定の規制ガイドラインや会社独自のポリシーが、機械学習の使用にどのような影響をもたらすのかを評価する必要があります。

実装の詳細については、SageMaker デベロッパーリソースのビデオ Secure and compliant ML workflows with Amazon SageMaker を参照してください。

寄稿者

本ドキュメントの寄稿者は次のとおりです。

- Stefan Natu, Sr.、機械学習ソリューションアーキテクト
- David Ping、プリンシパル機械学習ソリューションアーキテクト
- Kosti Vasilakakis、ビジネス開発マネージャー、AWS Machine Learning

- Alvin Huang、資本市場ビジネス開発
- Paul Jeyasingh、アドバイザリーコンサルタント、AWS プロフェッショナルサービス

ドキュメント改訂履歴

日付	説明
2020 年 7 月	初版発行