IT資格道場

# AWS MLA-C02 学習用テキスト

予備知識ゼロから合格ラインへ —— 読んで理解する1冊目

無料サンプル(第1章まで)

AWS 公式 Exam Guide(MLA-C02・2026年9月29日取得)準拠
2026年9月版

## ご利用にあたって

- このテキストは「IT資格道場」(www.shikaku-doujou.com)の購入者限定テキストのうち、学習用テキストの第1章までを収録した**無料サンプル**です。どなたでもご覧いただけます。
- 無料サンプルも著作物です。転載・改変しての再配布・二次販売はできません。
- AI モデルの学習データとしてのご利用(学習目的の入力・データセット化を含む)はできません。
- 製品版(学習用テキスト・頻出度別ノート・暗記用ノートの3点)は、ご購入者を識別できる透かし入りのPDFとして提供されます。
- 本PDFはスマートフォン・タブレットでの閲覧に合わせたA5版面・文字サイズで組版しています。

# 目次

3 冊の使い方は、ダウンロード欄の案内を参照してください。

このテキストの骨組みは、AWS が公開している **AWS Certified Machine Learning Engineer - Associate (MLA-C02) 試験ガイド** です。試験ガイドは範囲を 4 つのコンテンツ分野、12 のタスク、107 のスキルで決めています。章はコンテンツ分野の順、節はタスクの順、小節はスキルの順に並べました。小節の見出しにある `1.1.1` はスキルの番号、見出しの末尾にある `[1.1]` はタスクの番号です。どちらも試験ガイドの番号そのままなので、公式のガイドと並べて読めます。

各小節の冒頭には、試験ガイド日本語版のスキルの文を引用しました。その下に、そのスキルの問題を解くのに要る知識を書いています。

## 試験の骨組みと、このテキストの読み方

### 試験の基本情報

| 項目 | 内容 |
| --- | --- |
| 試験名 | AWS Certified Machine Learning Engineer - Associate（MLA-C02） |
| 問題数 | **65 問**。うちスコアに影響する設問が 50 問、影響しない採点対象外の設問が 15 問です。どれが採点対象外かは受験者には分かりません |
| 出題形式 | **択一選択問題**（正解 1 つ、不正解 3 つ）と **複数選択問題**（5 つ以上の選択肢のうち正解が 2 つ以上。正解をすべて選んだ時だけ得点）の 2 種類 |

| 項目 | 内容 |
| --- | --- |
| 採点 | 100〜1,000 の換算スコアで、**720 以上が合格**。補整スコアリングモデルなので、分野ごとの合格ラインはありません |
| 未解答 | 不正解として扱われます。推測で答えても減点はありません |
| ベータ版 | 2026 年 9 月 29 日から英語で配信。85 問・170 分・75 米ドル。統計評価用の追加設問を含みます |
| 日本語 | 日本語・韓国語・簡体字中国語は正式版（一般提供）から受験できます。一般提供の開始は、AWS の日本語ブログで 2027 年初頭と案内されています |
| 受験方法 | Pearson VUE のテストセンター、またはオンライン監督試験 |
| 有効期間 | 3 年 |

正式版の試験時間と受験料は、AWS 認定の受験ページが案内します。申し込む前に受験ページで最新の値を見てください。

## 4 つのコンテンツ分野と章の対応

| 分野 | 名前（試験ガイド日本語版） | 採点対象に占める割合 | 章 |
| --- | --- | --- | --- |
| 1 | ML と AI のためのデータの準備 | **28%** | 第 1 章 |
| 2 | ML モデルと基盤モデル (FM) の開発 | **24%** | 第 2 章 |

| 分野 | 名前（試験ガイド日本語版） | 採点対象に占める割合 | 章 |
|---|---|---|---|
| 3 | ML と AI のワークフローのデプロイとオーケストレーション | **24%** | 第 3 章 |
| 4 | ML と AI のソリューションの運用、モニタリング、保護 | **24%** | 第 4 章 |

AWS が公表している重みは分野の単位だけです。タスクやスキルの単位の出題数は公表されていません。

## MLA-C01 から何が変わったか

分野の構成とタスクの並びは MLA-C01 と同じです。AWS の比較ページは、C01 の 12 のタスクを C02 の 12 のタスクに 1 対 1 で対応させています。配分は分野 2 が 26% から 24% に下がり、分野 3 が 22% から 24% に上がりました。

大きく変わったのはスキルの中身です。C02 では 45 のスキルが追加されました。多くは基盤モデル（FM）と生成 AI の運用に関わるもので、例えば次の 8 つです。

- 検索拡張生成（RAG）とベクトルデータベース
- 埋め込みモデル
- Amazon Bedrock での FM の選択とカスタマイズ
- エージェントと Amazon Bedrock AgentCore
- Amazon Bedrock Guardrails
- Amazon Bedrock のプロンプトマネジメント

- LLM の評価（BLEU、ROUGE、LLM-as-a-judge など）
- FM の推論コストとトークンの管理

反対に、C01 にあった次の 7 項目は範囲から外れました。

- 学習用データを Amazon EFS や Amazon FSx に置いて学習リソースへ読み込ませる設定
- Amazon Bedrock や SageMaker JumpStart で独自データを使って事前学習済みモデルをファインチューニングする項目（C01 の 2.2 にあったもの）
- データ型の変更や枝刈りによるモデルサイズの縮小
- SageMaker Neo によるエッジデバイス向けの最適化
- SageMaker での独自コンテナの持ち込み（BYOC）
- Amazon EventBridge のイベントによるインフラの監視
- プロビジョニングされた同時実行数、サービスクォータ、自動スケーリングに関わる容量の問題のトラブルシューティング（C01 の 4.2 にあったもの）

FM のファインチューニングそのものは、C02 では分野 1 と分野 2 の新しいスキル（1.2.9、2.1.2、2.2.8）として書き直されています。

## 試験が問う力

試験ガイドの受験対象者は、Amazon SageMaker AI や Amazon Bedrock を使った ML エンジニアリングの経験が 1 年以上ある人です。従来の ML と生成 AI の両方の経験も求めています（試験ガイド「受験対象者について」）。一方で、試験の範囲外の職務として、エンドツーエンドの AI と ML のソリューション全体の設計や、ML 戦略の主導が挙がっています。

ここから分かるのは、問われるのが「要件に合う部品を選び、正しく設定し、動き続けるように運用する力」だという点です。設問の多くは、場面の要件（レイテンシー、コスト、運用の手間、セキュリティ）を読み、選択肢の中から要件を最も少ない手間で満たすものを選ぶ形です。暗記した名前だけでは選べません。

選択肢を絞る時は、次の 3 つの順で考えると迷いが減ります。

1. どの工程の話か（データの準備、モデルの開発、デプロイ、運用と保護）
2. 要件が何を最優先しているか（レイテンシー、スループット、コスト、運用の手間）
3. 最小の権限と最小の露出で満たせているか

## サービス名は英語のまま覚える

試験の画面に出るのは英語のサービス名と機能名です。本文でも、SageMaker AI の機能名や API 名、設定項目名は英語のまま書きました。Feature Store、Clarify、Model Monitor、Inference Recommender、Model Registry などがその例です。Amazon SageMaker は 2024 年 12 月に機械学習の部分の名前が **Amazon SageMaker AI** に変わりました。古い資料の「Amazon SageMaker」は、このテキストの SageMaker AI と同じものを指します。

# 第1章 ML と AI のためのデータの準備（分野 1・採点対象の 28%）

分野 1 は 4 つの分野の中で最も重みが大きく、採点対象の 28% を占めます。扱うのは、データを集めて保存し（1.1）、変換して特徴量や埋め込みにし（1.2）、品質とバイアスを確かめる（1.3）までの工程です。C02 では、従来の表形式のデータに加えて、RAG 用の文書、埋め込み、FM のファインチューニング用データの準備が加わりました。

この章の設問は、場面の要件を読んで「どのサービスで」「どの形式で」「どの設定で」を選ばせる形が中心です。サービスの名前だけでなく、上限の値や設定の名前まで知っていると選択肢を切れます。

## 1.1 データを収集して保存する [1.1]

タスク 1.1 は、データを取り出し、取り込み、保存するまでの 9 つのスキルで構成されます。C02 で加わったのは、ベクトルデータベースの設定（1.1.7）と、テキストや画像や音声といった多様なデータの取り込み（1.1.8）の 2 つです。

### 1.1.1 データソースからデータを抽出する [1.1]

> 試験ガイド 1.1.1：データソース (Amazon S3、Amazon EBS、Amazon EFS 、Amazon RDS 、Amazon DynamoDB 、Amazon OpenSearch Service など) からデータを抽出する。

ML のデータは、いろいろな保管先に散らばっています。抽出の設問は「どの保管先から、どの方法で取り出すと、本番の業務を邪魔せず、速く、安く取り出せ

るか」を問います。保管先ごとの標準の取り出し方と、詰まった時の打ち手を表にしました。

| 保管先 | 標準の取り出し方 | 遅い、または本番に影響する時の打ち手 |
| --- | --- | --- |
| **Amazon S3** | SDK や CLI の `GetObject`、Amazon Athena の SQL、AWS Glue や Spark からの読み込み | マルチパートでの並列転送、プレフィックスの分散、S3 Transfer Acceleration（遠距離のアップロード） |
| **Amazon EBS** | EC2 インスタンスにアタッチしてファイルとして読む | gp3 の IOPS とスループットを個別に上げる、io2 Block Express に変える |
| **Amazon EFS** | NFS でマウントして読む | スループットモードを見直す（Elastic、プロビジョンド） |
| **Amazon RDS** | SQL で抽出して S3 へ書き出す | リードレプリカから抽出する、スナップショットを S3 へエクスポートする、Glue の JDBC 接続で並列に読む |
| **Amazon DynamoDB** | `Query` と `Scan` | S3 へのエクスポート機能を使う（テーブルの読み込みキャパシティを消費しない） |
| **Amazon OpenSearch Service** | 検索 API、スクロールでの一括取得 | スナップショットを S3 に取る、Glue の OpenSearch 接続で読む |

表の中で取り違えやすいのは、DynamoDB と RDS の抽出です。DynamoDB の `Scan` はテーブルの全項目を読むので、読み込みキャパシティを大きく消費します。学習用に全件が欲しい場面では、ポイントインタイムリカバリーを有効にしたうえで **S3 へのエクスポート** を使います。この機能はテーブルのキャパシティを使わず、エクスポートの間もテーブルへの読み書きを止める必要はあ

りません。出力は DynamoDB JSON か Amazon Ion 形式で S3 に置かれ、Athena や Glue から読めます。

RDS も同じ考え方です。本番のプライマリに重いクエリを投げると、業務の応答が遅くなります。抽出はリードレプリカに向けるか、スナップショットを **Parquet 形式で S3 へエクスポート** します。スナップショットのエクスポートは本番のインスタンスに負荷をかけません。

S3 から読む時の速さは、読み方で大きく変わります。Athena や Spark で読むなら、列指向の形式（1.1.5）とパーティション分割が効きます。一方で SageMaker AI の学習ジョブが S3 から読む時は、入力モードが効きます（1.1.3）。

EBS と EFS の違いも押さえてください。EBS は 1 台の EC2 インスタンスにアタッチするブロックストレージです（io1 と io2 のマルチアタッチを除く）。EFS は複数のインスタンスから同時にマウントできるファイルストレージです。「複数の処理ノードから同じファイルを同時に読みたい」なら EFS、「1 台のインスタンスの作業領域を速くしたい」なら EBS を選びます。

### 1.1.2 コスト、性能、データ構造、コンプライアンスでストレージを決める [1.1]

> 試験ガイド 1.1.2：コスト、パフォーマンス、データ構造、データコンプライアンスに基づいて、ストレージに関する考慮事項を決定し、ストレージサービスを設定する。

ストレージの選び方は、4 つの軸で決まります。コスト、パフォーマンス、データ構造、データコンプライアンスです。先にデータ構造で候補を絞り、次に性能と

コストで 1 つに決め、最後にコンプライアンスの設定を足す順で考えると迷いません。

| 条件 | 選ぶもの | 理由 |
| --- | --- | --- |
| 構造化、半構造化、非構造化のデータを大量に貯め、分析にも学習にも使う | **Amazon S3**（データレイク） | 安く、ほぼすべての AWS の分析サービスと ML サービスから読める |
| キーで 1 件ずつ一桁ミリ秒で引く | **Amazon DynamoDB** | キーバリューのアクセスが速い |
| SQL の結合とトランザクションが要る業務データ | **Amazon RDS** | リレーショナルデータベース |
| 大量のデータに SQL の集計をかける | **Amazon Redshift** | 列指向の MPP データウェアハウス |
| 全文検索、ログ分析、ベクトル検索 | **Amazon OpenSearch Service** | 検索インデックス |
| 複数のインスタンスから POSIX で共有する | **Amazon EFS** | 共有ファイルシステム |
| 大量の読み出しを数百 GB/秒の単位で行う | **Amazon FSx for Lustre** | 高スループットの並列ファイルシステム |
| ほぼ読まないが長く残す | **S3 Glacier Deep Archive** | 最も安い。読む前に復元が要る |

S3 の中では、ストレージクラスの選び方が問われます。判断の手がかりは「どのくらいの頻度で読むか」「読む時にどれだけ待てるか」「アクセス頻度が分かっているか」の 3 つです。

| ストレージクラス | 向くデータ | 注意点 |
| --- | --- | --- |
| S3 Standard | 頻繁に読む | 最小保存期間なし |

| ストレージクラス | 向くデータ | 注意点 |
| --- | --- | --- |
| S3 Intelligent-Tiering | アクセス頻度が分からない、変わる | 取り出し料金なし。オブジェクトごとに監視料金 |
| S3 Standard-IA | 月 1 回程度しか読まないが、読む時はすぐ欲しい | 取り出しに GB 単位の料金。最小保存期間 30 日 |
| S3 One Zone-IA | 作り直せるデータ | 1 つのアベイラビリティーゾーンだけに保存 |
| S3 Glacier Instant Retrieval | 四半期に 1 回程度、読む時はミリ秒で欲しい | 最小保存期間 90 日 |
| S3 Glacier Flexible Retrieval | 年に 1 回程度、数分から数時間待てる | 復元してから読む。最小保存期間 90 日 |
| S3 Glacier Deep Archive | 年に 1 回未満、数時間待てる | 最も安い。最小保存期間 180 日 |

アクセス頻度が読めないのに Standard-IA を選ぶと、取り出し料金で Standard より高くつくことがあります。この場面の答えは Intelligent-Tiering です（取り出し料金がかからないため）。古くなったデータを自動で安いクラスへ移すには、**S3 ライフサイクル設定** を使います。

性能の軸で覚えておく値もあります。S3 は、1 つのプレフィックスあたり毎秒 3,500 件の書き込み系リクエスト（PUT、COPY、POST、DELETE）と、毎秒 5,500 件の読み込み系リクエスト（GET、HEAD）を処理できます。プレフィックスの数に上限はありません。分けた数だけ並列に処理できます。レイテンシーが最優先で 1 つのアベイラビリティーゾーンに置いてよいなら、**S3 Express One Zone**（ディレクトリバケット）が一桁ミリ秒の応答を出します。

コンプライアンスの軸では、設定の名前で答えます。

- **暗号化**: S3 は新しいオブジェクトを既定で SSE-S3 で暗号化します。鍵の利用を監査したい、鍵を自分で管理したい時は SSE-KMS を選ぶ
- **削除の防止**: S3 オブジェクトロック（コンプライアンスモードとガバナンスモード）とバージョニング
- **保存場所**: データを置くリージョンを選ぶ。レプリケーションの宛先もリージョンで決まります
- **機密データの発見**: Amazon Macie が S3 の中の個人情報を見つけます
- **アクセスの制限**: バケットポリシー、S3 ブロックパブリックアクセス、VPC エンドポイントポリシー

「規制で、指定した期間は誰も削除できないようにする」はオブジェクトロックのコンプライアンスモードです。ガバナンスモードは、特別な権限を持つユーザーなら保持期間中でも削除できます。

### 1.1.3 容量とスケーラビリティの問題を切り分ける [1.1]

> 試験ガイド 1.1.3：容量とスケーラビリティに関係するデータ取り込みとストレージの問題をトラブルシューティングおよびデバッグする。

取り込みと保存のトラブルは、症状、見る指標、原因、打ち手の順に並べると解けます。先に「どの指標が上限に張り付いているか」を決めてから選択肢を読みます。指標を見ずに打ち手を選んではいけません。

| 症状 | 見る指標 | ありがちな原因 | 打ち手 |
| --- | --- | --- | --- |
| Kinesis Data Streams への書き込みが `ProvisionedThroughputExceededException` で弾かれる | `WriteProvisionedThroughputExceeded` | シャードあたりの上限（毎秒 1 MB または 1,000 レコード）を超えた。パーティションキーが偏ってホットシャードになった | シャードを増やす、オンデマンドモードに切り替える、パーティションキーを散らす |
| Kinesis のコンシューマーの処理が遅れていく | `GetRecords.IteratorAgeMilliseconds` | 複数のコンシューマーがシャードあたり毎秒 2 MB の読み出しを取り合っている | 拡張ファンアウトの登録コンシューマーにする、コンシューマーを並列化する |
| S3 のリクエストが `503 Slow Down` になる | S3 のリクエストメトリクス | 1 つのプレフィックスにリクエストが集中した | プレフィックスを分ける、指数バックオフで再試行する |
| DynamoDB の書き込みがスロットリングされる | `ThrottledRequests`、`WriteThrottleEvents` | プロビジョンドキャパシティ不足、特定のパーティションキーへの集中 | オンデマンドモードにする、パーティションキーの設計を見直す |
| SageMaker AI の学習ジョブがディスク不足で止まる | 学習ジョブのログ | File モードはデータセット全体をインスタンスのボリュームにコピーする | FastFile モードや Pipe モードに変える、ボリュームを大きくする |
| 学習ジョブの開始までが長い | ジョブの開始時刻と `DownloadingTrainingImage` などの状態 | File モードで大きなデータを全部コピーしている | FastFile モードで S3 から直接ストリーミングする |

| 症状 | 見る指標 | ありがちな原因 | 打ち手 |
| --- | --- | --- | --- |
| AWS Glue のジョブがメモリ不足で止まる | Glue のジョブメトリクス、Spark UI | データの偏り、小さなファイルが大量にある、ワーカーが足りない | ワーカーの数とタイプを上げる、小さなファイルをまとめる |
| EBS の I/O が頭打ちになる | `VolumeQueueLength`、`BurstBalance` | gp2 のバーストクレジットを使い切った | gp3 に変えて IOPS とスループットを個別に上げる |

Kinesis Data Streams の上限は、この表の根拠です。書き込みはシャードあたり毎秒 1 MB または 1,000 レコード、読み出しはシャードあたり毎秒 2 MB です。1 レコードの上限は 10 MiB です（base64 エンコードの前の大きさ）。データの保持期間は既定で 24 時間、最大で 365 日まで延ばせます。

シャードの読み出し帯域は、そのシャードを読む全コンシューマーで共有されます。コンシューマーを増やしても、シャードの読み出し帯域は増えません。「複数のアプリケーションが同じストリームを低遅延で読む」なら、拡張ファンアウトを使います。拡張ファンアウトでは、登録したコンシューマーごとにシャードあたり毎秒 2 MB が割り当てられます。

書き込みエラーの設問で「Amazon Data Firehose に切り替える」という選択肢が出ることがあります。これは書き込み側のスループット不足の解決になりません。シャードの追加、オンデマンドモード、パーティションキーの分散のどれかが答えです。

必要なシャードの数は、書き込みの量から計算できます（プロビジョンドモードの場合）。例えば、1 件 2 KB のレコードが毎秒 3,000 件届くとします。データの量は毎秒 6 MB なので、1 MB の上限から 6 シャードが要ります。件数では毎秒 1,000 件の上限から 3 シャードで足ります（件数の方が緩い）。2 つのう

ち大きい方を取るので、答えは 6 シャードです。余裕を見て少し多めにし、パーティションキーが偏らないように設計します。

学習ジョブの入力モードも容量の問題に直結します。

| 入力モード | 動き | 向く場面 |
| --- | --- | --- |
| File | 開始前にデータセット全体をインスタンスのボリュームにコピーする | 小さなデータセット。何度も同じファイルを読む |
| FastFile | S3 のオブジェクトをファイルとして見せ、読んだ部分だけを取りに行く | 大きなデータセット。開始を早くしたい。コード変更なし |
| Pipe | S3 から学習コンテナへストリーミングする | 大きなデータを順に読む。対応するアルゴリズムやコードが要る |

### 1.1.4 ストリーミングデータを取り込む [1.1]

> 試験ガイド 1.1.4：AWS のストリーミングデータソース (Amazon Kinesis、Apache Flink、Apache Kafka など) を使用してデータを取り込む。

ストリーミングの設問では、Amazon Kinesis、Apache Flink、Apache Kafka の名前が並びます。この 3 つは同じ役割ではありません。Kinesis と Kafka はデータを保持して配る役、Flink はストリームの上で計算する役です。

| 名前 | 役割 | AWS での姿 | 選ぶ場面 |
| --- | --- | --- | --- |
| Amazon Kinesis Data Streams | レコードを順序付きで保持し、複数のコンシューマーに配る | マネージドサービス | 1 件ずつ低遅延で処理する。複数のアプリケーションが同じデータを読む |

| 名前 | 役割 | AWS での姿 | 選ぶ場面 |
|---|---|---|---|
| Amazon Data Firehose | バッファしてから宛先に届ける | サーバーレスのマネージドサービス | コードを書かずに S3、Redshift、OpenSearch Service へ貯める |
| Apache Flink | ウィンドウ集計、結合、異常検知などをストリームの上で行う | Amazon Managed Service for Apache Flink | 時間の窓で集計した特徴量をリアルタイムに作る |
| Apache Kafka | データを保持して配る（オープンソース） | Amazon MSK | 既存の Kafka のコードやエコシステムをそのまま使う |
| Amazon Kinesis Video Streams | 映像のストリームを取り込んで保存する | マネージドサービス | カメラ映像を画像系の ML に渡す |

Firehose の動きは設定の名前で問われます。Firehose は **バッファサイズ（MB）** と **バッファ間隔（秒）** のどちらかに達した時点で宛先へ届けます。遅延を短くしたいならバッファ間隔を小さくし、小さなファイルが大量にできるのを防ぎたいならバッファサイズを大きくします。配送の前に AWS Lambda で変換でき、Parquet や ORC への **レコード形式の変換** もできます（スキーマは AWS Glue Data Catalog から取ります）。**動的パーティショニング** を使えば、レコードの中の値でS3 のプレフィックスを分けられます。

Firehose はバッファしてから届けるので、数秒以上の遅れが入ります。「1 件ずつすぐにカスタム処理したい」には向きません。その場合は Kinesis Data Streams と Lambda の組み合わせ、集計が要るなら Managed Service for Apache Flink を選びます。

典型の組み合わせを並べておきます。

1. センサーのデータを Kinesis Data Streams に書き込む
2. Managed Service for Apache Flink が 5 分の窓で平均や最大を計算する
3. 計算した特徴量を SageMaker Feature Store のオンラインストアへ書き込む（1.1.9）
4. 生のデータは Firehose が Parquet に変換して S3 に貯める

Kafka を選ぶ条件ははっきりしています。オンプレミスで Kafka を運用していて、プロデューサーとコンシューマーのコードを変えずに移したい時は Amazon MSK です。新しく作るなら、運用の手間が少ない Kinesis が第一の候補になります。

### 1.1.5 アクセスパターンでデータ形式を選ぶ [1.1]

> 試験ガイド 1.1.5：データアクセスパターンに基づいて適切なデータ形式 (Apache Parquet、JSON、CSV、ORC など) を使用し、データの取り込みと書き込みを行う。

データ形式は、読み方から逆に決めます。何百もある列のうち数列だけを集計するのか、1 行をまるごと読み書きするのか、人が中身を目で見るのかで答えが変わります。

| 読み方 | 選ぶ形式 | 理由 |
|---|---|---|
| 多くの列のうち数列だけを繰り返し集計する | **Apache Parquet**、**Apache ORC** | 列指向なので、必要な列だけを読む。スキャン量が減り、Athena の料金も下がる |

| 読み方 | 選ぶ形式 | 理由 |
| --- | --- | --- |
| レコード全体を順に書き込み、スキーマがよく変わる | **Apache Avro** | 行指向で、スキーマをファイルに持つ。スキーマの変更に強い |
| ほかのシステムと受け渡す、人が目で見る | **CSV** | どのツールでも読める。スキーマと型を持たない |
| 入れ子の半構造化データを保つ | **JSON、JSON Lines** | 階層を保てる。1 行 1 レコードの JSON Lines は分割して読める |
| SageMaker AI の組み込みアルゴリズムで大量の数値を学習する | **RecordIO-protobuf** | バイナリで小さく、Pipe モードと相性がよい |

Parquet と ORC は、列ごとに圧縮し、列ごとの最小値と最大値を持ちます（統計情報）。クエリの条件に合わないブロックを読み飛ばせるので、Athena のスキャン量が大きく下がります。さらに S3 のキーを `year=2026/month=09/` のようにパーティションで区切れば、関係のないプレフィックスそのものを読まずに済みます。

圧縮の形式も見落とせません。gzip で圧縮した CSV は途中から読めないので、Spark や Athena で 1 つのファイルを並列に読めません。Snappy で圧縮した Parquet は列の単位で分割して読めます。「Athena のクエリが遅く、スキャン量が多い」の答えは、列指向の形式への変換とパーティション分割の組み合わせです。

書き込みの側でも形式を選びます。Glue の ETL ジョブ、Firehose のレコード形式の変換、Athena の CTAS（CREATE TABLE AS SELECT）は、どれも Parquet や ORC で書き出せます。

### 1.1.6 複数のソースのデータをマージする [1.1]

> 試験ガイド 1.1.6：複数のソースからデータをマージする (プログラミング手法、AWS Glue、Apache Spark の使用などによる)。

マージの手段は、データの大きさと繰り返しの頻度で選びます。

| 手段 | 向く条件 |
| --- | --- |
| プログラミング手法（pandas の `merge`、SQL の `JOIN`） | データが小さい。探索の段階で 1 回だけ結合する |
| **AWS Glue** の ETL ジョブ | サーバーレスで定期的に回したい。Data Catalog とクローラーで複数のソースのスキーマを管理したい |
| **Apache Spark**（Amazon EMR） | データが大きい。既存の Spark のコードを使う。クラスターを細かく調整したい |
| SageMaker Data Wrangler | 画面の操作で結合と変換を試したい。そのまま Pipelines や Feature Store に書き出したい |
| Amazon Athena のフェデレーテッドクエリ | S3 と RDS などをまたいで SQL で結合し、結果だけを取り出す |

Glue では、クローラーがデータソースを調べ、Data Catalog にテーブルの定義を作ります。ETL ジョブはそのテーブルを読み、`Join` などの変換で結合します。Glue は内部で Spark を使っていて、DynamicFrame という形でスキーマが揃わないデータも扱えます（列の型がファイルごとに違う場合など）。

結合で気をつけることは 3 つあります。

- **キーの整合**: 型、大文字と小文字、前後の空白がずれていると、結合されない行が出る

- **粒度の一致**: 「1 顧客 1 行」と「1 顧客 N 行」を結合すると行が増える。先に集約するか、結合後に重複を除く
- **時点の一致**: 特徴量の時刻がラベルの時刻より後だと、未来の情報が混ざる（ターゲットリーケージ）

時点の一致は見落としやすい点です。例えば「解約したか」を当てるモデルで、解約の翌月の利用額を特徴量に入れると、学習では高い精度が出ても本番では使えません。SageMaker Feature Store のオフラインストアはイベント時刻を持つので、ある時点の特徴量だけを取り出す結合（ポイントインタイムの結合）が作れます。

### 1.1.7 AI アプリケーション用のベクトルデータベースを設定する [1.1]

> 試験ガイド 1.1.7：仕様に基づいて AI アプリケーション用にスケーラブルなベクトルデータベース (OpenSearch Service、pgvector を使用する Amazon RDS、Amazon S3 など) を設定する。

ベクトルデータベースは、埋め込みベクトル（1.2.5）を保存し、あるベクトルに近いものを速く探すためのデータベースです。RAG では、質問を埋め込みに変え、ベクトルデータベースから意味の近い文書の断片を探して FM に渡します。

近いものを探す方法には 2 つあります。全件と距離を比べる **完全一致の k-NN**（厳密検索）と、索引を使って近似的に探す **近似最近傍探索（ANN）** です。件数が多い時に全件と比べるのは速度の面で現実的ではありません。そこで ANN を使います。ANN の代表的な索引が **HNSW**（階層的なグラフ）と **IVF**（クラスタに分けて探す）です。HNSW は速くて精度も高い代わりにメモリを多く使います。IVF はメモリを抑えられる代わりに、学習の手順と精度の調整が要ります。

距離の測り方も決めなければなりません。**コサイン類似度**、**ユークリッド距離（L2）**、**内積** の 3 つです。どれを使うかは埋め込みモデルに合わせます（1.2.5）。正規化した埋め込みなら、コサイン類似度と内積は同じ順位を返します。

AWS で選べる主なベクトルストアを並べます。

| ベクトルストア | 特徴 | 選ぶ場面 |
|---|---|---|
| **Amazon OpenSearch Service**（マネージドクラスター） | k-NN プラグインで HNSW や IVF を使う。キーワード検索とのハイブリッド検索ができる | 大規模で低レイテンシー。キーワードと意味の両方で探したい。細かく調整したい |
| **Amazon OpenSearch Serverless**（ベクトル検索コレクション） | 容量を自動で調整する。クラスターの運用が要らない | 運用の手間を減らしたい。Amazon Bedrock ナレッジベースのクイック作成で選べる |
| **Amazon RDS for PostgreSQL**、**Amazon Aurora PostgreSQL**（pgvector） | PostgreSQL の拡張機能。SQL で業務データとベクトルを一緒に扱える | 既に PostgreSQL を使っている。メタデータを SQL で結合したい |
| **Amazon S3 Vectors** | S3 のベクトルバケットとベクトルインデックスにベクトルを保存し、API で類似検索する | 大量のベクトルを安く長く持ちたい。検索の頻度が高くない |
| **Amazon Neptune Analytics** | グラフとベクトルを一緒に扱う | 文書の中のつながりを使う GraphRAG |

pgvector では、テーブルに `vector` 型の列を作り、`CREATE EXTENSION vector` で拡張機能を有効にします。索引は HNSW か IVFFlat を選びます（索引を作

らなければ全件の厳密検索）。距離の演算子は、L2 距離が `<->`、内積が `<#>`、コサイン距離が `<=>` です。

S3 Vectors は、ベクトルを専用のベクトルバケットに置き、その中のベクトルインデックスに対して類似検索の API を呼ぶ仕組みです。サーバーを立てる必要はありません。大量のベクトルを安く持ちたい時に選びます。レイテンシーをさらに下げたい時は、よく使うベクトルを OpenSearch Service に出す組み合わせも取れます。

仕様から設定を決める時は、次の 5 つを先に確かめます。

- **次元数**: 埋め込みモデルの出力の次元と、索引の次元を一致させる
- **距離の種類**: 埋め込みモデルが前提とする距離に合わせる
- **件数と伸び方**: ANN の索引の種類、シャード数、容量を決める
- **フィルタ**: 部署や日付などのメタデータで絞るなら、メタデータの列やフィールドを用意する
- **更新の頻度**: 頻繁に追加や削除があるなら、索引の再構築のコストを見込む

次元数の不一致は典型的なエラーです。埋め込みモデルを Amazon Titan Text Embeddings V2 の 1,024 次元から 512 次元の設定に変えたのに、索引を作り直さないと書き込みがエラーになります。モデルや次元を変えた時は、全文書の埋め込みを作り直し、索引も作り直します。

### 1.1.8 テキスト、画像、音声などを取り込んで保存する [1.1]

> 試験ガイド 1.1.8：AI と ML のアプリケーションのために、さまざまなデータタイプ (テキスト、画像、音声など) を取り込んで保存する。

生成 AI のアプリケーションは、表形式のデータだけでなく、PDF、画像、音声、動画を扱います。基本の形は、元のファイルを S3 に置き、そこから取り出したテキストやメタデータ、埋め込みを別に持つことです。

データの種類ごとに、取り込みで使うサービスを対応させます（保存先も並べました）。

| データの種類 | 取り込みと変換 | 保存先 |
| --- | --- | --- |
| スキャンした書類、PDF、帳票 | **Amazon Textract** でテキスト、表、フォームのキーと値を取り出す | 元のファイルは S3。取り出したテキストは S3 やナレッジベースへ |
| 音声、通話の録音 | **Amazon Transcribe** で文字起こしする（個人情報を伏せる設定もある） | 音声は S3。文字起こしの結果は JSON で S3 |
| 画像 | **Amazon Rekognition** でラベルや文字を取り出す。マルチモーダルの埋め込みモデルでベクトルにする | 画像は S3。ベクトルはベクトルストア |
| 動画、カメラ映像 | **Amazon Kinesis Video Streams** で取り込む | 映像はストリームと S3 |
| 自由記述のテキスト | **Amazon Comprehend** でエンティティや感情、言語を取り出す | S3、ナレッジベース |
| ログ、クリックの記録 | Kinesis Data Streams や Firehose | S3 の Parquet |

文書、画像、音声が混ざるデータを RAG で使う場合、**Amazon Bedrock ナレッジベース** の解析の設定が役に立ちます。既定のパーサーはテキストを取り出すだけですが、**Amazon Bedrock Data Automation** や FM をパーサーに選ぶと、表や図を含む文書から構造を保ったまま中身を取り出せます。

保存の設計では、元のファイルと派生物を混ぜません。元のファイルを S3 に残しておけば、埋め込みモデルやチャンキングの方法を変えた時に作り直せます。画像や音声は S3 のオブジェクトとして置き、ベクトルストアにはベクトルと S3 のキー、メタデータを入れます。大きなバイナリをベクトルストアに直接入れる設計は取りません。

小さなファイルが大量にある場合は、読み出しのオーバーヘッドが大きくなります。学習で何百万枚もの画像を読むなら、ファイルをまとめた形式（RecordIO や TFRecord、WebDataset の tar）にすると読み出しが速くなります。

### 1.1.9 SageMaker Feature Store にデータを取り込む [1.1]

> 試験ガイド 1.1.9：SageMaker Feature Store にデータを取り込む。

**SageMaker Feature Store** は、特徴量を作って保存し、学習と推論で共有するための場所です。特徴量の集まりを **フィーチャーグループ** と呼びます。フィーチャーグループの各行はレコードで、次の 2 つを持ちます。

- **レコード識別子**（RecordIdentifier）：その行を一意に表すキー（顧客 ID など）
- **イベント時刻**（EventTime）：その値がいつの状態かを表す時刻

フィーチャーグループには、オンラインストアとオフラインストアの 2 つの保存先があります。

| 項目 | オンラインストア | オフラインストア |
| --- | --- | --- |
| 持つもの | レコードごとの最新の値だけ | すべての履歴（追記のみ） |

| 項目 | オンラインストア | オフラインストア |
| --- | --- | --- |
| 用途 | リアルタイム推論で低レイテンシーに読む | 学習、バッチ推論、データ探索 |
| 実体 | マネージドの低レイテンシーのストア | 自分の S3 バケットに Parquet で保存 |
| 読み方 | `GetRecord`、`BatchGetRecord` | Amazon Athena でクエリ |

取り込みの経路は 2 つです（ストリーミングとバッチ）。1 つは `PutRecord` **API** による取り込みです。レコードを 1 件ずつ同期で書き込み、オンラインストアに即座に反映されます。オフラインストアにも数分以内に書き込まれます（オフラインストアを有効にした場合）。Kinesis や MSK、Flink で作ったストリーミングの特徴量を入れる時はこの経路です。

もう 1 つは **バッチ取り込み** です。Data Wrangler からのエクスポート、SageMaker Processing ジョブ、Feature Store の Spark コネクタを使って、大量のレコードをまとめて書き込みます。オンラインストアを使わずオフラインストアだけのフィーチャーグループなら、S3 に直接まとめて書き込めます（オンラインストアを経由しません）。

レコードを消す時は `DeleteRecord` を呼びます。オンラインストアからは消えますが、オフラインストアは追記のみなので、削除を表すレコードが追加されます（履歴は消えません）。オンラインストアには **TTL**（有効期間）も設定でき、期限が過ぎたレコードを読まないようにできます。

Feature Store が解く問題は **学習とサービングのずれ**（training-serving skew）です。学習と推論で別々に特徴量を計算すると、計算の仕方が少しずつずれて、本番の精度が下がります。同じ定義の特徴量を同じストアから読めば、このずれが起きません。

取り違えやすいのは、オフラインストアをリアルタイム推論に使う選択肢です。オフラインストアは S3 の Parquet なので、ミリ秒単位の読み出しには向きません。反対に「過去のある時点の特徴量で学習し直したい」はオフラインストアの出番です。オンラインストアは最新の値しか持たないからです。

フィーチャーグループの名前、説明、タグには、個人を特定できる情報を入れません。名前やタグはアカウントの中で広く見えるメタデータだからです。

## 場面から選ぶ：データの収集と保存 [1.1]

タスク 1.1 の設問は、要件の言い回しで答えがほぼ決まります。よく見る言い回しと、選ぶもの、外す選択肢の理由を表にしました。

| 要件の言い回し | 選ぶもの | 外す選択肢と理由 |
| --- | --- | --- |
| アクセス頻度が分からない学習データを安く保存したい | S3 Intelligent-Tiering | Standard-IA は取り出し料金が読めない |
| アーカイブしたデータを、必要な時はミリ秒で読みたい | S3 Glacier Instant Retrieval | Flexible Retrieval と Deep Archive は復元を待つ |
| 規制で 7 年間は誰も削除できないようにしたい | S3 オブジェクトロックのコンプライアンスモード | ガバナンスモードは特権で消せる |
| 本番の DynamoDB の全件を学習に使いたい | S3 へのエクスポート | `Scan` はキャパシティを消費する |
| 本番の RDS に負荷をかけずに抽出したい | リードレプリカ、スナップショットの S3 エクスポート | プライマリへの直接のクエリ |
| 複数のアプリケーションが同じストリームを低遅延で読みたい | Kinesis の拡張ファンアウト | シャードを足しても 1 つあたりの帯域は共有のまま |

| 要件の言い回し | 選ぶもの | 外す選択肢と理由 |
| --- | --- | --- |
| コードを書かずにストリームを S3 に Parquet で貯めたい | Data Firehose のレコード形式の変換 | Lambda で自作するのは手間が大きい |
| 5 分の窓で集計した特徴量をリアルタイムに作りたい | Managed Service for Apache Flink | Firehose は集計しない |
| 既存の Kafka のコードを変えずに移したい | Amazon MSK | Kinesis は Kafka の API と互換ではない |
| Athena のスキャン量を減らしたい | Parquet への変換とパーティション分割 | gzip の CSV は分割して読めない |
| 学習ジョブがディスク不足で止まる | FastFile モード | File モードは全データをコピーする |
| リアルタイム推論で最新の特徴量を読みたい | Feature Store のオンラインストア | オフラインストアは S3 の Parquet |
| 過去の時点の特徴量で学習し直したい | Feature Store のオフラインストア | オンラインストアは最新の値だけ |
| 既存の PostgreSQL で業務データと一緒にベクトル検索したい | RDS または Aurora の pgvector | 別のストアに分けると結合が手間 |
| 大量のベクトルを安く持ちたい | Amazon S3 Vectors | 常に動くクラスターは費用がかさむ |
| 型番のキーワードと意味の両方で探したい | OpenSearch Service のハイブリッド検索 | ベクトル検索だけでは型番の区別が苦手 |

**場面 1**：ある小売の会社が、店舗の POS のデータを毎日 S3 に置いています。分析者は Athena で 300 列のうち 5 列だけを集計していて、クエリが遅く料金も高い状態です。この時に効くのは、データを Parquet に変換し、日付でパーティション分割する手です。Athena は必要な列と日付のファイルだけを読

むようになります（Glue の ETL ジョブや Athena の CTAS で変換できる）。インスタンスを大きくする、Redshift に移すといった選択肢は、スキャン量という原因を直していません。

**場面 2**：コールセンターが、通話の録音と文字起こしを RAG で検索できるようにしたい場面です（埋め込みモデルは今後変えるかもしれない）。この場合は、録音の元のファイルと文字起こしの JSON を S3 に残します。ベクトルストアには、ベクトルと S3 のキーとメタデータだけを入れます。埋め込みモデルを変えた時は、S3 の文字起こしから埋め込みを作り直せます。

## 1.2 データ変換、特徴量エンジニアリング、前処理を実行する [1.2]

タスク 1.2 は、集めたデータをモデルが使える形に変える 9 つのスキルです（1.2.1〜1.2.9）。前半の 4 つは表形式のデータの変換と特徴量エンジニアリング、後半の 5 つは埋め込み、テキストの前処理、RAG の文書準備、マスキング、FM のファインチューニング用データです。後半の 5 つはすべて C02 で加わりました。

### 1.2.1 AWS ツールでデータを変換する [1.2]

> 試験ガイド 1.2.1：AWS ツール (AWS Glue、AWS Glue DataBrew、Amazon EMR での Spark、SageMaker Data Wrangler など) を使用してデータを変換する。

変換のツールは、コードを書くかどうか、データの大きさ、運用の手間で選びます。

| ツール | 形 | 選ぶ場面 |
| --- | --- | --- |
| **AWS Glue**（ETL ジョブ） | サーバーレスの Spark。Python や Scala のコード、または Glue Studio の画面 | 定期的な ETL をサーバーの管理なしで回す |
| **AWS Glue DataBrew** | 画面で操作するデータ準備。250 以上の組み込みの変換を「レシピ」として保存する | コードを書かない分析者が、変換の手順を再利用したい |
| **Amazon EMR での Spark** | 自分で構成するクラスター（EMR Serverless も選べる） | 大規模なデータ。既存の Spark や Hive のコード。細かい調整が要る |
| **SageMaker Data Wrangler** | SageMaker AI（SageMaker Canvas）の画面でのデータ準備。変換の流れをデータフローとして持つ | ML の前処理を画面で試し、SageMaker Pipelines や Feature Store に書き出す |
| **SageMaker Processing** ジョブ | 自分のスクリプトをマネージドのコンテナで実行する | scikit-learn や Spark の前処理を学習の前段として決まった環境で回す |
| **AWS Lambda** | 小さな関数 | 1 件ずつの軽い変換。15 分以内に終わる処理 |

DataBrew と Data Wrangler は、どちらも画面で変換する道具です。違いは行き先にあります。DataBrew は分析や ETL の一般的なデータ準備で、結果を S3 などに書き出します。Data Wrangler は ML の前処理に特化していて、クイックモデルでの特徴量の重要度の確認や、ターゲットリーケージの分析まで画面でできます（学習につなげる機能が中心）。

Glue の ETL ジョブでは、ジョブブックマークで前回処理した位置を覚えられます。毎日新しいファイルだけを処理したい場面で使います（処理済みのファ

イルは読み直さない）。ワーカーの種類（G.1X、G.2X など）と数で処理能力を決めます。

EMR を選ぶのは、Spark の設定を細かく変えたい時や、Spark 以外のフレームワークも使いたい時です。クラスターの管理を避けたいなら EMR Serverless や Glue を選びます。

### 1.2.2 SageMaker Feature Store で特徴量を作成して管理する [1.2]

> 試験ガイド 1.2.2：AWS ツール (SageMaker Feature Store など) を使用して特徴量を作成および管理する。

1.1.9 は Feature Store への取り込みでした。ここでは、特徴量を定義し、管理し、再利用する側を扱います。

フィーチャーグループを作る時は、特徴量の名前と型（`String`、`Integral`、`Fractional`）、レコード識別子、イベント時刻、オンラインストアとオフラインストアの有無を決めます。作った後から特徴量の列を追加することもできます（`UpdateFeatureGroup`）。

特徴量を管理する機能は次の 4 つです。

- **特徴量の検索とメタデータ**: 特徴量に説明やパラメータを付けて、ほかのチームが探せるようにする
- **リネージ**: どのデータとどの処理から特徴量ができたかをたどる
- **ポイントインタイムの取り出し**: オフラインストアから、ある時刻より前の最新の値だけを集めて学習データを作る

- **暗号化とアクセス制御**: AWS KMS の鍵でオンラインとオフラインを暗号化し、IAM でフィーチャーグループごとに権限を分ける

ポイントインタイムの取り出しは、学習データを作る時の要です。ラベルの時刻ごとに、それより前の特徴量だけを結合します。これで未来の情報が混ざるのを防げます（1.1.6 のターゲットリーケージ）。

Feature Store を使う理由は、特徴量を一度作れば、学習と推論とほかのチームの別モデルで使い回せる点にあります。同じ計算を何度も書かずに済み、学習とサービングのずれも起きません。

オフラインストアのデータは、AWS Glue Data Catalog にテーブルとして登録されます。そのため、Athena の SQL でフィーチャーグループ同士を結合し、学習データを作れます（テーブルの形式は Glue の形式か Apache Iceberg を選ぶ）。Iceberg の形式にすると、小さなファイルをまとめる処理（コンパクション）で読み出しを速く保てます。

オンラインストアには、保存の種類が 2 つあります。標準の保存は、多くの特徴量を低レイテンシーで読む一般的な用途に向きます。インメモリの保存は、さらに低いレイテンシーが要る用途に向きます（料金は高い）。どちらを選ぶかは、推論のレイテンシーの目標で決めます。

特徴量を作る処理そのものを管理する機能もあります（Feature Processor）。Spark で書いた変換を登録し、スケジュールやイベントで実行して、結果をフィーチャーグループに書き込みます。特徴量の計算の定義がコードとして残るので、誰がどう計算したかをたどれます（1 か所で定義する）。

### 1.2.3 ストリーミングデータを変換する [1.2]

> 試験ガイド 1.2.3：ストリーミングデータを変換する (AWS Lambda、

> Spark の使用などによる)。

ストリーミングのデータは、届いた順に少しずつ変換します。手段の選び方は、処理が 1 件ごとで済むか、時間の窓で集計が要るかで分かれます。

| 変換の中身 | 手段 |
|---|---|
| 1 件ごとの形式の変換、項目の追加、不要な項目の削除 | **AWS Lambda**（Kinesis Data Streams のイベントソース、または Firehose のデータ変換） |
| 時間の窓での集計、ストリーム同士の結合、状態を持つ処理 | **Amazon Managed Service for Apache Flink** |
| マイクロバッチでの集計、既存の Spark のコードを使う | **Spark Structured Streaming**（Amazon EMR、AWS Glue のストリーミング ETL） |
| S3 に貯める前に Parquet や ORC に変える | **Amazon Data Firehose** のレコード形式の変換 |

Lambda を Kinesis のイベントソースにすると、Lambda はシャードからレコードをまとめて受け取ります。バッチサイズとバッチウィンドウで、1 回に受け取る量と待つ時間を決めます。処理がエラーで止まると、同じシャードの後ろのレコードは待たされます（順序を守るため）。この時は **関数のエラー時にバッチを二分割する設定** や、失敗したレコードを送る宛先（オンフェイラー）を使います。

Firehose のデータ変換では、Firehose が Lambda にレコードをまとめて渡し、Lambda が変換した結果を返します。Lambda は各レコードに `Ok`、`Dropped`、`ProcessingFailed` のどれかを付けて返します（`result` フィールド）。`Ok` は変換に成功したレコードで、宛先に届きます。`Dropped` は意図して捨て

るレコードで、宛先に届きません。`ProcessingFailed` は変換に失敗したレコードで、S3 のエラー用のプレフィックスに書き出されます。

Flink は、ウィンドウの種類（タンブリング、スライディング、セッション）とイベント時刻を使って集計します。遅れて届いたデータを扱うには、ウォーターマークで「どこまで待つか」を決めます。「過去 10 分の取引の回数と合計を 1 分ごとに計算する」はスライディングウィンドウの例です。

### 1.2.4 特徴量エンジニアリングを実行する [1.2]

> 試験ガイド 1.2.4：特徴量エンジニアリングを実行する (スケーリング、標準化、特徴量分割、ビニング、ログ変換、正規化など)。

特徴量エンジニアリングは、モデルが学びやすいように値の形を変える作業です。代表的な手法を、何を解決するかと合わせて表にしました（下の 6 つ）。

| 手法 | やること | 効く場面 |
| --- | --- | --- |
| **スケーリング**（最小最大スケーリング） | 値を 0 から 1 などの範囲に収める | 距離を使うアルゴリズム（k-NN、k-means）、ニューラルネットワーク |
| **標準化** | 平均 0、標準偏差 1 に変える（z スコア） | 値の範囲が特徴量ごとに大きく違う。線形モデル、勾配降下法 |
| **正規化** | ベクトルの長さを 1 にそろえる（L2 正規化） | テキストのベクトル、コサイン類似度を使う場面 |
| **ログ変換** | 値の対数を取る | 右に長く裾を引く分布（年収、アクセス数、価格） |

| 手法 | やること | 効く場面 |
| --- | --- | --- |
| **ビニング** | 連続値を区間に分けてカテゴリにする | 年齢を 10 歳ごとに分けるなど。外れ値の影響を抑える |
| **特徴量分割** | 1 つの列を複数の意味のある列に分ける | 日時から曜日と時間帯を取り出す。住所から都道府県を取り出す |

日本語の「正規化」は、最小最大スケーリングを指す文脈もあります。試験の英語では、scaling と standardization と normalization が別の選択肢として並ぶことがあるので、表の意味で区別してください。

外れ値が多いデータには、最小最大スケーリングが向きません。最小最大スケーリングは、最大値が 1 つ極端に大きいと、ほかの値が 0 の近くに押し込まれます。外れ値が多いなら、標準化や、中央値と四分位範囲を使うロバストスケーリング、ログ変換を選びます。

数字で確かめます（5 件の小さな例）。年収の列に 300 万、400 万、500 万、600 万、1 億の 5 件があるとします。最小最大スケーリングでは、1 億が 1、300 万が 0 になり、400 万から 600 万は 0.01 から 0.03 の間に詰め込まれます。ほとんどの人の違いが見えなくなります（差が 0.02 しかない）。対数を取ると、1 億の値は他と比べて極端には離れず、300 万から 600 万の違いも残ります。

木を使うモデル（XGBoost、ランダムフォレスト）は、値の大きさの順番だけを使って分割します。そのためスケーリングや標準化の影響をほとんど受けません。スケーリングが効くのは、距離や勾配を使うモデルです。

カテゴリの変数は、数値に変えてから使います（エンコーディング）。

| エンコーディング | やること | 注意点 |
| --- | --- | --- |
| **ワンホットエンコーディング** | カテゴリごとに 0 と 1 の列を作る | カテゴリの数が多いと列が増えすぎる |
| **ラベルエンコーディング** | カテゴリに整数を割り当てる | 順序のないカテゴリに使うと、線形モデルが大小の意味を読み取ってしまう |
| **順序エンコーディング** | 順序のあるカテゴリ（小、中、大）に順番どおりの整数を振る | 順序がある時だけ使う |
| **ターゲットエンコーディング** | カテゴリごとのラベルの平均で置き換える | リーケージを避けるため、学習データの中で分けて計算する |
| **特徴量ハッシュ** | ハッシュ関数で決まった数の列に落とす | カテゴリの数が非常に多い時。衝突が起きる |

最後に、変換の手順は学習データだけで作ります。平均や標準偏差を、検証データやテストデータを含めて計算すると、評価が実力より良く出ます。学習データで作った変換を、そのまま検証、テスト、本番の推論に当てます。

### 1.2.5 埋め込みモデルでテキストと画像を数値にする [1.2]

> 試験ガイド 1.2.5：埋め込みモデルを設定し使用して、テキストデータと画像データを数値表現に変換する。

**埋め込み**（embedding）は、テキストや画像を、意味の近さが距離の近さになるような数値のベクトルに変えたものです。「猫」と「ねこ」と「cat」は、文字は違っても近いベクトルになります。この性質を使うと、キーワードが一致しない文書も意味で探せます。RAG の検索、似た商品の推薦、文書の分類、重複の検出に使います。

Amazon Bedrock で使える埋め込みモデルの例は次の表のとおりです（Bedrock のモデル一覧から抜粋）。

| モデル | 入力 | 特徴 |
|---|---|---|
| **Amazon Titan Text Embeddings V2** | テキスト | 出力の次元を 1,024、512、256 から選べる。正規化した出力を返せる |
| **Amazon Titan Multimodal Embeddings G1** | テキストと画像 | テキストと画像を同じベクトル空間に置く。文章で画像を探せる |
| **Cohere Embed**（英語、多言語） | テキスト（モデルによって画像も） | 検索用の文書と検索の質問で入力の種類を指定する |

埋め込みモデルを設定する時に決めることは 4 つあります。

- **次元数**: 次元を小さくすると、保存のコストと検索の時間が減る代わりに、精度が少し下がる
- **正規化**: 出力の長さを 1 にそろえる。コサイン類似度と内積の結果が一致する
- **入力の上限**: 1 回に渡せるトークン数には上限がある。長い文書はチャンクに分けてから埋め込む（1.2.7）
- **言語と分野**: 日本語の文書なら多言語に対応したモデルを選ぶ

埋め込みの呼び出しは、Bedrock の `InvokeModel` API で行います。大量の文書を一度に埋め込むなら、**バッチ推論** を使うと 1 件ずつ呼ぶより安くなります。ストリーミングの応答は最初の文字を早く見せる手段で、プロビジョンドスループットは使わない時間も料金がかかる手段です。どちらも急がない大量の処理を安くする手段ではありません。Amazon Bedrock ナレッジベースを

使う場合は、同期（取り込みジョブ）の中でナレッジベースが埋め込みモデルを呼びます。自分で呼ぶ必要はありません。

守るべき決まりが 1 つあります。**索引に入れる文書と、検索の質問は、同じ埋め込みモデルで同じ設定のまま埋め込む** ことです。モデルや次元が違うベクトルを比べても、距離に意味はありません。モデルを変えたら、すべての文書を埋め込み直します。

SageMaker AI で埋め込みを作ることもできます。SageMaker JumpStart から埋め込みモデルをエンドポイントにデプロイし、自分のデータで呼びます。独自の分野に合わせた埋め込みモデルが要る時や、データを Bedrock に渡したくない事情がある時の選択肢です。

### 1.2.6 トークン化とドメイン固有の拡張でテキストを前処理する [1.2]

> 試験ガイド 1.2.6：高度なテキスト前処理手法 (トークン化、ドメイン固有の拡張など) を適用する。

テキストをモデルに渡す前の処理を前処理と呼びます。従来の NLP では、小文字化、記号の除去、ストップワードの除去、語幹化（stemming）、見出し語化（lemmatization）が定番でした。FM や埋め込みモデルは文脈を読むので、これらを強くかけると逆に意味が失われることがあります。前処理は強くかけるほど良いわけではありません。何を残すかは、使うモデルに合わせて決めます。

**トークン化** は、テキストをモデルが扱う単位（トークン）に切る処理です。単語ごとに切る方法、文字ごとに切る方法、その中間の **サブワード** に切る方法があります。今の FM の多くは BPE（Byte Pair Encoding）や SentencePiece に

よるサブワードのトークン化を使います。サブワードなら、辞書にない新しい単語も部品に分けて表せます（未知語が出ない）。

トークン化で知っておくことは 3 つです。

- **モデルごとにトークナイザーが違う**: 同じ文でもトークン数が変わる。上限や料金はそのモデルのトークン数で決まる
- **日本語は英語よりトークンが多くなりやすい**: 文字数あたりのトークン数がモデルによって違う
- **最大の長さを超えると切り捨てられる**: 学習やファインチューニングのデータは、上限に収まるように切るか分ける

**ドメイン固有の拡張**（domain-specific augmentation）は、専門分野のデータが少ない時に、学習データを人工的に増やす方法です。例として次の 5 つがあります。

- **同義語の置き換え**: 分野の辞書を使って、専門用語を同じ意味の別の言い方に置き換える
- **逆翻訳**: 日本語を英語に訳し、また日本語に戻して言い回しを変える
- **テンプレートでの生成**: 「{製品名} の {部品} が {症状}」のような型に値を入れて文を作る
- **FM での言い換え**: FM に元の文の言い換えや、似た質問を作らせる
- **ノイズの追加**: 実際の入力に近いタイプミスや略語を混ぜる

拡張で気をつけるのはラベルの保存です。言い換えで意味が変わってしまうと、ラベルと中身が合わないデータが増えます。拡張したデータは一部を人が確かめてから使います。拡張したデータを検証やテストのデータに入れてはいけません（評価が実力より良く出るため）。

分野の専門用語が多い場合、トークナイザーが専門用語を細かく切りすぎることがあります。継続的な事前トレーニング（1.2.9）で分野の文書を読ませると、その分野の言葉への理解が深まります。

### 1.2.7 RAG のための文書を準備する [1.2]

> 試験ガイド 1.2.7：検索拡張生成 (RAG) アプリケーションのためのドキュメント (チャンキング戦略、メタデータ抽出など) を準備する。

**検索拡張生成（RAG）** は、質問に関係する文書の断片を先に探し、それを FM へのプロンプトに添えて答えを作らせる方法です。FM を学習し直さずに、社内の文書や最新の情報に基づいた答えを返せます。RAG の答えの質は、文書をどう切り、どんな情報を添えて索引に入れたかで大きく変わります（検索で拾えない情報は答えに使われない）。

文書を切る単位を **チャンク**、切り方を **チャンキング戦略** と呼びます。Amazon Bedrock ナレッジベースで選べる戦略は次のとおりです。

| 戦略 | 切り方 | 向く文書 |
|---|---|---|
| **既定のチャンキング** | 約 300 トークンごとに切る。文の区切りを尊重する | まず試す時 |
| **固定サイズのチャンキング** | 指定したトークン数ごとに切り、前後のチャンクと重なり（オーバーラップ）を持たせる | 構造がそろった文書。サイズを自分で決めたい時 |
| **階層チャンキング** | 大きな親チャンクと小さな子チャンクを作る。子で検索し、答えには親を渡す | マニュアルや規程のような、節ごとの文脈が要る長い文書 |

| 戦略 | 切り方 | 向く文書 |
| --- | --- | --- |
| **セマンティックチャンキング** | 埋め込みの近さで、意味の切れ目を見つけて切る | 話題が途中で変わる文書、自然な文章 |
| **チャンキングなし** | 1 ファイルを 1 チャンクとして扱う | 前もって自分で切ってある文書。FAQ の 1 問 1 答 |

独自の切り方が要る時は、**AWS Lambda でのカスタム変換** を使います。取り込みの途中で Lambda を呼び、自分のロジックでチャンクを作ったり、メタデータを付けたりできます（途中の結果は指定した S3 に置かれる）。

チャンクの大きさは、検索の精度と答えの質のバランスで決めます。小さすぎると、1 つのチャンクに答えに要る文脈が収まりません。大きすぎると、関係のない内容が混ざって検索の精度が下がり、プロンプトのトークンも増えます（料金とレイテンシーが上がります）。オーバーラップは、文の途中で切れた情報を前後のチャンクで拾うためのものです。

大きさの感覚を数字で持っておきます。約 6 万トークンのマニュアルを 300 トークンのチャンクに切ると、約 200 のチャンクができます（オーバーラップを付けるとその分だけ増える）。検索で上位 5 件を渡すと、プロンプトに約 1,500 トークンが加わります。チャンクを 1,000 トークンにすれば数は約 60 に減りますが、上位 5 件で約 5,000 トークンになり、1 回の呼び出しの料金が上がります。

**メタデータ** は、チャンクに付ける属性です。文書の種類、部署、作成日、製品名、言語などを付けておくと、検索の時にフィルタで絞れます。ナレッジベースでは、S3 の文書と同じ場所に `文書名.metadata.json` というファイルを置いて属性を付けます（メタデータファイル）。例えば「2026 年度の規程だけから答える」「質問した人の部署の文書だけを見る」を実現できます。

メタデータを取り出す方法は、文書の中身によって選びます。

- **ファイルの属性**: パス、ファイル名、更新日から付ける
- **文書の構造**: 見出し、ページ番号、章の番号を取り出す
- **サービスでの抽出**: Amazon Textract で表やフォームを取り出す。Amazon Comprehend でエンティティ（製品名、組織名）を取り出す
- **FM での抽出**: FM に要約やキーワード、分類を作らせる

文書の前処理も答えの質に効きます。ヘッダー、フッター、ページ番号の繰り返し、目次などの雑音は、チャンクに入る前に取り除きます。表や図を含む PDF は、既定のパーサーでは表の構造が崩れることがあります。この場合は、ナレッジベースの解析の設定で **Amazon Bedrock Data Automation** や FM によるパーサーを選びます（1.1.8）。

## 1.2.8 データをマスキング、編集、匿名化する [1.2]

> 試験ガイド 1.2.8：データをマスキング、編集、匿名化する。

個人情報や機密情報を含むデータを ML や生成 AI に使う時は、使う前に守る処理をかけます。似た言葉が並ぶので、意味を区別してください（表の 4 つ）。

| 手法 | やること | 元に戻せるか |
|---|---|---|
| **マスキング** | 値の一部を記号に置き換える（`090-****-1234`） | 戻せない。形は残る |
| **編集**（redaction） | 値そのものを消す、または `[NAME]` のような印に置き換える | 戻せない |

| 手法 | やること | 元に戻せるか |
| --- | --- | --- |
| **仮名化**（トークン化） | 値を別の識別子に置き換え、対応表を別に安全に持つ | 対応表があれば戻せる |
| **匿名化** | 個人を特定できないように、値を消したり粗くしたりする（年齢を年代にする） | 戻せない |

仮名化した値は、対応表と組み合わせれば個人を特定できます。そのため、規制上は個人データのまま扱われることが多い点に注意してください。

AWS で個人情報を見つけて伏せるサービスは次のとおりです。

| サービス | 対象 | できること |
| --- | --- | --- |
| **Amazon Macie** | S3 のオブジェクト | 機密データの自動検出とレポート（伏せる処理はしない） |
| **Amazon Comprehend** | テキスト | `DetectPiiEntities` で個人情報の種類と位置を返す。PII の編集ジョブで伏せた文書を作る |
| **Amazon Comprehend Medical** | 医療のテキスト | 保護対象保健情報（PHI）を検出する |
| **Amazon Transcribe** | 音声 | 文字起こしの時に PII を伏せる（コンテンツの編集） |
| **AWS Glue** | ETL のデータ | 機密データの検出の変換で、列の中の個人情報を見つけて伏せる |

| サービス | 対象 | できること |
| --- | --- | --- |
| **AWS Glue DataBrew** | 表形式のデータ | マスキング、ハッシュ、暗号化などの変換 |
| **Amazon Bedrock Guardrails** | FM の入力と出力 | 機密情報フィルタで PII をブロックまたはマスクする（4.3.9） |

使い分けの基本は「見つけるだけか、伏せるところまでか」です。Macie は S3 のどこに個人情報があるかを見つける道具で、中身を書き換えません。伏せる処理は Comprehend、Glue、DataBrew で行います。

ファインチューニングや RAG に使う文書は、取り込む前に伏せておきます。一度モデルに学習させた個人情報は、後から取り除けません。RAG の索引に入った個人情報は、検索で誰にでも返されるおそれがあります。

伏せ方は、後の用途に合わせて選びます。分析で「同じ顧客かどうか」だけ分かればよいなら、ハッシュや仮名化で識別子を置き換えます。値の傾向だけが要るなら、年齢を年代に、住所を都道府県にするなど粗くします（一般化）。

## 1.2.9 FM のファインチューニング、継続的な事前トレーニング、蒸留のデータを準備する [1.2]

> 試験ガイド 1.2.9：FM のファインチューニング、継続的な事前トレーニング、モデル蒸留を行うためのデータを準備する。

Amazon Bedrock のモデルのカスタマイズには、主に 3 つの方法があります。用意するデータの形は、3 つとも同じではありません。

| 方法 | 目的 | 用意するデータ |
| --- | --- | --- |
| **ファインチューニング** | 特定のタスクの出し方を教える（書式、口調、分類の仕方） | ラベル付きのデータ。プロンプトとその正しい応答の組 |
| **継続的な事前トレーニング** | 特定の分野の知識と言葉を覚えさせる | ラベルのない分野のテキスト（社内文書、論文、規程） |
| **モデル蒸留** | 大きいモデル（教師）の答え方を小さいモデル（生徒）に移し、安く速くする | プロンプトの集まり。教師モデルの応答は Bedrock が生成する（または呼び出しのログを使う） |

データは JSON Lines（1 行に 1 つの JSON）の形で S3 に置きます。ファインチューニングの基本の形は、プロンプトと応答の組です。

```
{"prompt": "次の問い合わせを分類してください: 画面が開かない",
"completion": "技術サポート"}
```

新しいモデルでは、会話の形（`system` と `messages` の配列）で書く形式を使います（Anthropic Claude 3 Haiku や Amazon Nova など）。形式はモデルごとに決まっているので、使うモデルのドキュメントに合わせてください。継続的な事前トレーニングでは、ラベルのないテキストだけを渡します。

```
{"input": "社内規程 第 12 条 ..."}
```

データを準備する時に確かめることを並べます。

- **形式**: 1 行 1 レコードの JSON Lines で、指定されたキーを持っている
- **長さ**: 1 レコードのトークン数がモデルの上限に収まっている

- **件数**: モデルが定める最小と最大の件数の範囲にある
- **分割**: 学習用と検証用に分ける（検証用のデータを指定すると検証の損失が出る）
- **品質**: 応答の正しさ、重複、個人情報、有害な内容を確かめる（1.3.6）

蒸留では、教師モデルが応答を作るので、用意するのはプロンプトが中心です。Bedrock はプロンプトを増やしたり、教師の応答を生成したりしてデータを合成します。本番で既に教師モデルを呼んでいるなら、**モデル呼び出しのログ** を蒸留のデータに使えます（呼び出しのログを有効にしておく）。

どの方法を選ぶかの判断は 2.1.2 で扱います。データの側で覚えておくのは「ラベル付きの組ならファインチューニング、ラベルなしの文書なら継続的な事前トレーニング、プロンプトだけなら蒸留」という対応です。

## 場面から選ぶ：変換、特徴量、前処理 [1.2]

タスク 1.2 のよくある言い回しと、選ぶものを表にしました。

| 要件の言い回し | 選ぶもの | 外す選択肢と理由 |
|---|---|---|
| コードを書かない分析者が、変換の手順を保存して毎日使いたい | AWS Glue DataBrew のレシピ | EMR はクラスターの管理が要る |
| 前処理を画面で試し、そのまま SageMaker Pipelines に組み込みたい | SageMaker Data Wrangler | DataBrew は ML のパイプラインへの書き出しが中心ではない |
| 毎日新しく届いたファイルだけを処理したい | Glue のジョブブックマーク | 毎回全件を処理すると時間と費用が増える |
| Kinesis のレコードを 1 件ずつ軽く整形したい | AWS Lambda | Flink は集計や状態を持つ処理向け |

| 要件の言い回し | 選ぶもの | 外す選択肢と理由 |
| --- | --- | --- |
| 右に長く裾を引く金額の列を扱いやすくしたい | ログ変換 | 最小最大スケーリングは外れ値に弱い |
| 距離を使うアルゴリズムの前に単位をそろえたい | スケーリング、標準化 | 木のモデルには効果が小さい |
| 順序のないカテゴリを線形モデルに渡したい | ワンホットエンコーディング | ラベルエンコーディングは大小の意味が入る |
| カテゴリの数が数万ある列を扱いたい | 特徴量ハッシュ、ターゲットエンコーディング | ワンホットは列が増えすぎる |
| 意味の近い文書を探したい | 埋め込みモデルとベクトル検索 | キーワードの一致だけでは言い換えを拾えない |
| テキストと画像を同じ空間で探したい | Titan Multimodal Embeddings | テキスト専用の埋め込みモデル |
| 長いマニュアルで、答えに節の文脈が要る | 階層チャンキング | 固定サイズの小さいチャンクは文脈が切れる |
| 話題が途中で変わる文章を意味の切れ目で分けたい | セマンティックチャンキング | 固定サイズは話題の途中で切れる |
| 部署ごとに見てよい文書を分けたい | メタデータを付けて検索でフィルタ | FM への指示だけでは守れない |
| 問い合わせのテキストから個人情報を伏せたい | Amazon Comprehend の PII の検出と編集 | Macie は S3 の中を見つけるだけで伏せない |
| 通話の録音の文字起こしで個人情報を伏せたい | Amazon Transcribe の PII の編集 | 文字起こしの後に別の仕組みを足すより手間が少ない |
| 社内用語を FM に覚えさせたい（ラベルなしの文書が大量） | 継続的な事前トレーニング | ファインチューニングはラベル付きの組が要る |

| 要件の言い回し | 選ぶもの | 外す選択肢と理由 |
| --- | --- | --- |
| 決まった書式で答えさせたい（例の組がある） | ファインチューニング | 継続的な事前トレーニングは書式を教えない |
| 大きいモデルの精度に近いまま推論を安くしたい | モデル蒸留 | 小さいモデルに切り替えるだけでは精度が落ちる |

**場面 1**：保険の会社が、約款の PDF（表や注記が多い）を RAG で使いたい場面です。答えに要る条件が表の中にあり、既定の設定では表の構造が崩れて正しく答えられません。この時は、ナレッジベースの解析の設定で Bedrock Data Automation や FM によるパーサーを選び、表の構造を保ったまま取り出します。チャンクの大きさを変えるだけでは、崩れた表は直りません。

**場面 2**：ある会社がファインチューニングのデータとして、過去のチャットの記録 2 万件を使おうとしています。記録には顧客の氏名と電話番号が含まれています（伏せる前の生のデータ）。学習の前に Comprehend などで個人情報を伏せ、応答の正しさと重複を確かめてから JSON Lines にします。個人情報を含んだまま学習すると、モデルが答えの中にそれを出すおそれがあります。後から取り除くことはできません。

## 1.3 データ品質を検証し、バイアスを管理する [1.3]

タスク 1.3 は、変換したデータが学習に使える状態かを確かめる 7 つのスキルです。品質の検証、ラベル付け、バイアスの特定と軽減、クラス不均衡、クリーニングに加えて、C02 では生成 AI の学習データの整合性（1.3.6）と、テキストや画像を含むデータのバイアス（1.3.4）が加わりました。

### 1.3.1 データ品質を検証する [1.3]

> 試験ガイド 1.3.1：データ品質を検証する (DataBrew、AWS Glue Data Quality の使用などによる)。

データ品質は、いくつかの観点に分けて測ります。どの観点が壊れているかで、打ち手が変わるからです。

| 観点 | 意味 | 例 |
|---|---|---|
| 完全性 | 値が欠けていない | 必須の列に空の値がない |
| 一意性 | 重複がない | 注文 ID が重複していない |
| 妥当性 | 値が決まった範囲や形式に収まる | 年齢が 0 以上 120 以下、日付の形式が正しい |
| 一貫性 | 別の列や別の表と矛盾しない | 出荷日が注文日より後 |
| 鮮度 | 値が新しい | 最終更新が 24 時間以内 |
| 正確性 | 値が現実と合っている | 住所が実在する |

AWS で品質を測る道具は 3 つです（Glue Data Quality、DataBrew、Data Wrangler）。

**AWS Glue Data Quality** は、DQDL（Data Quality Definition Language）という言葉でルールを書き、Data Catalog のテーブルや Glue の ETL ジョブの中で評価します。ルールの例は `IsComplete "customer_id"`、`IsUnique "order_id"`、`ColumnValues "age" between 0 and 120`、`Completeness "email" > 0.95`、`RowCount > 1000` です。データを見てルールを提案する **ルールの推奨** の機能もあります。ETL ジョブの中で評価すれば、ルールに違反した時にジョ

ブを止めたり、結果を CloudWatch に送ったりできます（品質の悪いデータを下流に流さない）。

**AWS Glue DataBrew** は、画面でデータを見ながら品質を確かめる道具です。**プロファイルジョブ** を実行すると、列ごとの欠損の数、一意の値の数、分布、外れ値、列同士の相関が一覧になります。データ品質のルールセットを作り、プロファイルジョブで検証することもできます（ルールごとに合否が出る）。

**SageMaker Data Wrangler** の **Data Quality and Insights Report** は、ML の前処理の目線で品質を調べます。欠損、重複、外れ値に加えて、ターゲットとの関係が強すぎる列（リーケージの疑い）や、クイックモデルの精度まで出します。

3 つの使い分けは、目的で決まります（下の 3 通り）。パイプラインの中で毎回自動で検証して止めたいなら Glue Data Quality です。人が画面でデータを理解したいなら DataBrew のプロファイルが向いています（結果を画面で見る）。学習の前に ML の目線で問題を洗い出すなら Data Wrangler を選びます（リーケージの確認はここだけ）。

品質の検証は、1 回で終わりではありません。学習用のデータを作るたびに同じルールで確かめ、結果を残します。本番の推論データの品質は、SageMaker Model Monitor のデータ品質モニタリングで継続して見ます（4.1.3）。

### 1.3.2 データにラベルと注釈を付ける [1.3]

> 試験ガイド 1.3.2：データにラベルと注釈を付ける。

教師あり学習には、正解のラベルが付いたデータが要ります。ラベル付けを AWS で行う中心のサービスは **Amazon SageMaker Ground Truth** です。

Ground Truth のラベル付けジョブは、次の順で作ります。

1. S3 に置いたデータの一覧（入力マニフェスト）を用意する
2. タスクの種類を選ぶ（画像分類、境界ボックス、セマンティックセグメンテーション、テキスト分類、固有表現の抽出、動画、3D 点群など）
3. 作業する人（ワークフォース）を選ぶ
4. 作業の説明と画面（ワーカーのテンプレート）を用意する
5. 結果を出力マニフェストとして S3 で受け取る

ワークフォースは 3 種類から選べます。

| ワークフォース | 誰が作業するか | 選ぶ場面 |
|---|---|---|
| パブリック（Amazon Mechanical Turk） | 世界中の不特定の作業者 | 機密でない簡単な作業を大量に安く |
| プライベート | 自社の社員や契約者 | 機密データ、専門知識が要る作業 |
| ベンダー | AWS Marketplace のラベル付け業者 | 専門の業者に任せたい |

機密データや個人情報を含むデータは、パブリックのワークフォースに出してはいけません。この場面の答えはプライベートのワークフォースです。

品質を上げる仕組みも問われます。Ground Truth は、1 つのデータを複数の作業者に見せ、答えを統合します（注釈の統合）。多数決や、作業者の信頼度を重みにした統合を使います。ラベルを後から確かめ直す **ラベルの検証ジョブ** と **ラベルの調整ジョブ** もあります。

費用を抑える機能が **自動データラベル付け** です。人がラベルを付けたデータでモデルを学習し、モデルが自信を持って判定できるデータには機械がラベルを付けます。自信のないデータだけを人に回すので、人の作業量が減ります（能動学習）。データの件数が多い時に効きます。

専門家のチームにラベル付けの運用ごと任せたい時は **SageMaker Ground Truth Plus** を選びます。ワークフォースやワークフローの設計を自分でする必要がありません。

生成 AI では、ラベル付けの中身も変わります。FM の応答に対して「どちらの応答が良いか」を人が選ぶ比較のデータや、応答の良し悪しを採点するデータを作ります。これらはファインチューニングや評価（2.3.7）で使われます。

### 1.3.3 データのバイアスのソースを特定して軽減する [1.3]

> 試験ガイド 1.3.3：AWS のツールや手法 (データセットの分割、シャッフル、拡張など) を使用して、データのバイアスのソースを特定して軽減する。

ここで言うバイアスは、データが現実や目的の母集団を偏って写していることです。偏ったデータで学習したモデルは、偏った予測をします。バイアスが入る経路を知っていれば、どこで直すかが決まります。

| バイアスの種類 | 入り方 | 例 |
|---|---|---|
| 選択バイアス | データの集め方が偏る | アプリの利用者だけのデータで全国民を予測する |
| サンプリングのバイアス | 一部の集団が少なすぎる | 高齢者のデータが全体の 2% しかない |
| 測定のバイアス | 測り方や記録の仕方が偏る | 地域によってセンサーの精度が違う |
| ラベルのバイアス | 付ける人の判断が偏る | 作業者によって「不適切」の基準が違う |
| 時間のバイアス | 集めた時期が偏る | 繁忙期のデータだけで年間を予測する |

偏りは集め方だけで生まれるわけではありません。データの扱い方でも起きます。データセットの分割、シャッフル、拡張は、その代表的な手当てです。

**データセットの分割** は、学習、検証、テストにデータを分ける作業です。分け方を誤ると評価が偏ります（実力より良く、または悪く出る）。分類では、ラベルの比率を各セットで同じにする **層化分割** を使います。時系列のデータは、時刻の順に分けます（過去で学習し、未来で評価する）。同じ顧客のデータが学習とテストの両方に入らないよう、顧客の単位で分けることもあります（グループ分割）。

**シャッフル** は、データの並び順を混ぜる作業です。ファイルが日付順やクラス順に並んだままだと、ミニバッチごとに偏ったデータで学習が進みます。学習の前やエポックごとにシャッフルします。SageMaker AI の学習ジョブでは、S3 のデータを読む時に `ShuffleConfig` でシャッフルを指定できます。時系列の予測では、分割の前に全体をシャッフルしてはいけません（未来の情報が学習に入るため）。

**拡張**（augmentation）は、少ない集団のデータを人工的に増やす作業です。画像なら回転や反転、明るさの変更を加えます。テキストなら言い換えや逆翻訳を使います（1.2.6）。表形式なら SMOTE などで合成します（1.3.5）。

バイアスを見つける AWS の道具は **SageMaker Clarify** です。Clarify は学習の前にデータのバイアスを測り（1.3.4）、学習の後にモデルの予測のバイアスを測ります（2.3.9）。Data Wrangler の画面からもバイアスのレポートを作れます。

軽減の手は、バイアスの入り方に合わせて選びます。集め方が偏っているなら、足りない集団のデータを追加で集めます。数が少ないだけなら、オーバーサンプリングや拡張、重み付けを使います。ラベルが偏っているなら、ラベル付けの指示を見直し、複数の作業者の答えを統合します（1.3.2）。性別や人種な

どの属性の列を消すだけでは、偏りは消えません。郵便番号のような別の列が、その属性の代わりになる（代理変数）ことがあるためです。

### 1.3.4 数値、テキスト、画像のデータにバイアスメトリクスを適用する [1.3]

> 試験ガイド 1.3.4：数値アセット、テキストアセット、画像アセットにバイアスメトリクスを適用して、マルチモーダルなデータ分布を最適化する。

バイアスメトリクスは、データの偏りを数値にする物差しです。SageMaker Clarify の学習前のバイアスメトリクスは、データを **ファセット**（性別や年齢層など、偏りを調べたい属性）で分け、集団の間の差を測ります。

| メトリクス | 何を測るか | 読み方 |
|---|---|---|
| **クラス不均衡（CI）** | ファセットの集団の件数の差 | -1 から 1。0 に近いほど釣り合っている |
| **ラベルの比率の差（DPL）** | 集団ごとの正のラベルの割合の差 | 0 に近いほど差が小さい |
| **KL ダイバージェンス（KL）** | 集団ごとのラベルの分布の違い | 0 以上。大きいほど分布が違う |
| **JS ダイバージェンス（JS）** | KL を対称にしたもの | 0 以上。大きいほど分布が違う |
| **Lp ノルム（LP）** | ラベルの分布の距離 | 0 以上 |
| **全変動距離（TVD）** | ラベルの分布の差の半分の L1 距離 | 0 以上 |
| **コルモゴロフ・スミルノフ（KS）** | ラベルの分布の最大の差 | 0 から 1 |

| メトリクス | 何を測るか | 読み方 |
| --- | --- | --- |
| **条件付き人口統計的格差（CDDL）** | 部分集団ごとに見た、不利なラベルの偏り | 集計で隠れる偏り（シンプソンのパラドックス）を見つける |

CI はラベルを見ずに件数だけを比べます。DPL から先はラベルの分布を比べます。つまり「ある集団のデータが少ない」のは CI、「ある集団だけ承認の割合が低い」のは DPL で見つかります（2 つの問題は別物）。

計算の例です。ファセットの一方（a）が 800 件、他方（d）が 200 件なら、CI は (800 − 200) ÷ (800 + 200) で 0.6 になります。承認の割合が a で 50%、d で 30% なら、DPL は 0.5 − 0.3 で 0.2 です。CI が 0 に近くても DPL が大きい、ということも起こります（件数は同じでも、承認の割合が違う）。

テキストと画像のデータには、表の列のようなファセットがそのままは存在しません。そこで、まずデータから属性を取り出します。

- **テキスト**: 言語、話題、文書の出どころ、登場する属性語（性別や年代を表す語）をメタデータとして付ける。Amazon Comprehend で言語やエンティティを取り出す
- **画像**: 撮影の条件（明るさ、角度）、写っているものの属性、画像の出どころをラベルとして付ける。Amazon Rekognition や Ground Truth の注釈を使う
- **数値**: 属性の列をそのままファセットに使う。連続値は区間に分ける

属性が付けば、CI や DPL を同じように計算できます。「顔の画像のうち、暗い照明で撮られたものが 3% しかない」「問い合わせのテキストの 95% が日本語で、英語の問い合わせの精度が低い」といった偏りが数値で見えます（CI と DPL）。

**マルチモーダルなデータ分布の最適化** とは、これらのメトリクスを見ながら、テキスト、画像、数値のそれぞれで分布を目的に近づけることです。手段は、足りない集団のデータの追加収集、拡張、リサンプリング、重み付けです。直した後にもう一度メトリクスを計算し、差が縮んだかを確かめます。

偏りを直す目標は「全部を均等にする」ことだけではありません。本番で出会うデータの分布に近づけることが目的です。本番で英語の問い合わせが 30% あるなら、学習データでも英語が十分な量になるように整えます。

## 1.3.5 クラス不均衡を解決する [1.3]

> 試験ガイド 1.3.5：数値データセット、テキストデータセット、画像データセットのクラス不均衡を解決する。

クラス不均衡は、分類のラベルの件数が大きく偏っている状態です。不正取引が全体の 0.5% しかない、といった例です（不正検知で典型的な形）。このままでは、モデルは「すべて正常」と答えるだけで 99.5% の正解率を出してしまいます。

打ち手は、データの側、学習の側、評価の側の 3 つに分かれます。

| 側 | 打ち手 | 内容 |
|---|---|---|
| データ | ランダムオーバーサンプリング | 少ないクラスを複製して増やす |
| データ | **SMOTE** | 少ないクラスの近い点の間に、新しい点を合成する（数値の特徴量） |
| データ | アンダーサンプリング | 多いクラスを減らす。情報が失われる |

| 側 | 打ち手 | 内容 |
|---|---|---|
| データ | 拡張 | 画像の回転や反転、テキストの言い換えで少ないクラスを増やす |
| データ | 合成データ | FM で少ないクラスのテキストや画像を作る |
| 学習 | クラスの重み付け | 少ないクラスの誤りに大きな罰を与える（XGBoost の `scale_pos_weight` など） |
| 学習 | しきい値の調整 | 予測の確率のしきい値を 0.5 から動かす。下げると正と判定する件数が増えて見逃しが減り、上げると誤検知が減る |
| 評価 | 指標の選び方 | 正解率を使わず、適合率、再現率、F1、PR 曲線の AUC で見る |

データの種類ごとの手当ても区別してください。SMOTE は数値の特徴量の空間で点を合成する方法で、画像やテキストにはそのまま使えません。画像なら回転、反転、切り抜き、色の変更で増やします。テキストなら言い換え、逆翻訳、同義語の置き換え、FM での生成を使います。

オーバーサンプリングは、分割の後、学習データだけにかけます。分割の前に複製すると、同じデータが学習とテストの両方に入り、評価が実力より良く出ます（データの漏れ）。

XGBoost の `scale_pos_weight` は、負のクラスの件数を正のクラスの件数で割った値が目安です。SageMaker AI の Linear Learner にも、正のクラスの重みを決める設定があります（`positive_example_weight_mult`）。

評価の指標も忘れずに変えます（2.3.6）。不正の検知で見逃しを減らしたいなら再現率、誤検知を減らしたいなら適合率を重視します。

### 1.3.6 AI トレーニングデータの整合性を検証する [1.3]

> 試験ガイド 1.3.6：AI トレーニングデータの整合性を検証する (プロンプトと応答のペアの検証、コンテンツの安全性のスクリーニングなど)。

FM のファインチューニングや蒸留に使うデータは、モデルの振る舞いをそのまま決めます。間違った応答や有害な内容が混ざると、モデルはそれを覚えます。そのため、学習の前に整合性を確かめる作業が要ります。

**プロンプトと応答のペアの検証** では、次の点を確かめます。

- **形式**: JSON Lines として読めて、必要なキー（`prompt` と `completion`、または `messages`）を持つ
- **長さ**: プロンプトと応答の合計がモデルのトークンの上限に収まる。空の応答がない
- **正しさ**: 応答が質問に正しく答えている。正解が分かるタスクは自動で照合し、残りは抜き取りで人が見る
- **一貫性**: 同じ種類の質問に、同じ書式と口調で答えている
- **重複**: 同じペアや、ほぼ同じペアが何度も入っていない
- **汚染**: 評価用のデータと同じペアが学習データに入っていない

**コンテンツの安全性のスクリーニング** では、有害な内容と機密情報を取り除きます。

| 確かめる内容 | 使える AWS の道具 |
| --- | --- |
| ヘイト、暴力、性的な内容、侮辱などの有害なテキスト | Amazon Bedrock Guardrails の `ApplyGuardrail` API、Amazon Comprehend の有害なコンテンツの検出 |
| プロンプト攻撃（ジェイルブレイク）の例文 | Bedrock Guardrails のプロンプト攻撃のフィルタ |
| 個人情報 | Amazon Comprehend の PII 検出、Bedrock Guardrails の機密情報フィルタ（1.2.8） |
| 不適切な画像 | Amazon Rekognition のコンテンツモデレーション |

`ApplyGuardrail` API は、FM を呼ばずにガードレールの判定だけを行います。学習データの一件一件に当てて、ブロックされた行を除けます。

大量のデータを人が全部見ることはできません。機械で振り分け、疑わしいものと抜き取りの一部を人が見る、という組み合わせにします。FM に採点させる方法（LLM-as-a-judge、2.3.9）も、応答の質の粗い振り分けに使えます。

整合性の検証は、データを作り直すたびに同じ手順で行います。どの版のデータで学習したかを記録しておけば、問題が見つかった時にさかのぼれます（3.3.6）。

### 1.3.7 データをクリーニングする [1.3]

> 試験ガイド 1.3.7：データクリーニングを行う (外れ値の検出、欠損データの補完、重複排除などによる)。

データのクリーニングは、壊れた値や余分な行を直す作業です。中心になるのは、外れ値、欠損、重複の 3 つです。

**外れ値の検出** には、統計の方法とモデルの方法があります。

| 方法 | 考え方 |
| --- | --- |
| z スコア | 平均から標準偏差の何倍離れているかで判定する（3 倍を超えたら外れ値、など） |
| 四分位範囲（IQR） | 第 1 四分位数と第 3 四分位数の幅の 1.5 倍より外を外れ値とする。分布の偏りに強い |
| **Random Cut Forest（RCF）** | SageMaker AI の組み込みアルゴリズム。異常の度合いをスコアで返す |
| 可視化 | 箱ひげ図、散布図、ヒストグラムで目で見る（Data Wrangler、DataBrew） |

外れ値は、見つけたら消すものとは限りません。入力の誤り（年齢が 999）なら直すか消します。本物の珍しい値（高額の取引）なら残します（消すとモデルが珍しい値を学べない）。不正の検知のように、外れ値そのものが予測したい対象のこともあります。

**欠損データの補完** は、欠けている値の理由と量で方法を選びます。

| 方法 | 向く場面 |
| --- | --- |
| 行を消す | 欠損が少なく、でたらめに起きている |
| 列を消す | その列の大半が欠けている |
| 平均値、中央値で埋める | 数値の列。外れ値が多いなら中央値 |
| 最頻値で埋める | カテゴリの列 |
| 前後の値で埋める | 時系列 |

| 方法 | 向く場面 |
| --- | --- |
| モデルで予測して埋める | ほかの列から推測できる（k-NN での補完など） |
| 「欠損」を表す列を足す | 欠けていること自体に意味がある |

補完に使う平均や中央値は、学習データだけで計算します。テストデータを含めて計算すると、評価が実力より良く出ます（1.2.4 と同じ理由）。

**重複排除** は、同じ内容の行を 1 つにする作業です。完全に同じ行は簡単に消せますが、表記ゆれで少しだけ違う行（「株式会社 A」と「(株)A」）はそのままでは見つかりません。この場合は、表記を正規化してから比べるか、あいまい一致を使います。AWS Glue の **FindMatches** 変換は、ML を使って同じものを指すレコードを見つけます（例を人がいくつかラベル付けして学習させる）。

テキストのデータでは、ほぼ同じ文書の重複が問題になります。同じ文書が何度も入ると、モデルはその内容を過大に覚えます。RAG の索引では、同じ内容のチャンクが検索結果の上位を占めてしまいます。埋め込みの近さやハッシュで、ほぼ同じ文書を見つけて除きます（ニアデュプリケートの除去）。

クリーニングの手順は、再現できる形で残します。DataBrew のレシピ、Glue のジョブ、Data Wrangler のデータフロー、SageMaker Processing のスクリプトのどれかに書いておけば、新しいデータにも同じ処理をかけられます。

## 場面から選ぶ：品質とバイアス [1.3]

タスク 1.3 のよくある言い回しと、選ぶものを表にしました。

| 要件の言い回し | 選ぶもの | 外す選択肢と理由 |
| --- | --- | --- |
| ETL の中で品質のルールを評価し、違反したら止めたい | AWS Glue Data Quality（DQDL） | 画面で見るだけの道具では止められない |

| 要件の言い回し | 選ぶもの | 外す選択肢と理由 |
|---|---|---|
| どんなルールを書けばよいか分からない | Glue Data Quality のルールの推奨 | 全部を手で書く |
| データの欠損と分布を画面で一覧したい | DataBrew のプロファイルジョブ | ETL のコードを書く |
| ターゲットリーケージの疑いを学習の前に見つけたい | Data Wrangler の Data Quality and Insights Report | DataBrew のプロファイルはリーケージを見ない |
| 医療の画像に専門家がラベルを付けたい（機密） | Ground Truth のプライベートワークフォース | パブリックのワークフォースに機密データは出せない |
| ラベル付けの費用を下げたい（件数が多い） | Ground Truth の自動データラベル付け | 全件を人が付ける |
| ラベル付けの運用ごと専門のチームに任せたい | Ground Truth Plus | 自分でワークフォースを管理する |
| ある集団のデータの件数が少ないかを数値で見たい | Clarify の CI | DPL はラベルの割合の差を見る |
| ある集団だけ承認の割合が低いかを数値で見たい | Clarify の DPL | CI は件数しか見ない |
| 不正が 0.5% しかないデータで学習したい | SMOTE、重み付け、しきい値の調整、PR 曲線の AUC | 正解率で評価する |
| 画像のクラスの偏りを直したい | 少ないクラスの画像の拡張 | SMOTE は数値の特徴量向け |
| ファインチューニングのデータの有害な内容を除きたい | `ApplyGuardrail` API、Comprehend の有害なコンテンツの検出 | 人が全件を読む |
| 表記ゆれで同じ顧客が別の行になっている | Glue の FindMatches | 完全一致の重複排除 |

| 要件の言い回し | 選ぶもの | 外す選択肢と理由 |
| --- | --- | --- |
| 外れ値の度合いをスコアで出したい | Random Cut Forest | 目で見て判断するだけ |

**場面 1**：住宅ローンの審査のモデルで、学習データの 85% が都市部の申込者でした。地方の申込者の承認の割合も低いことが分かっています。CI で地方の件数の少なさを、DPL で承認の割合の差を確かめ、地方のデータの追加や重み付けで分布を直します。性別や地域の列を消すだけでは、郵便番号などの代理変数から偏りが残ります（列の削除だけでは足りない）。直した後にもう一度メトリクスを計算して、差が縮んだかを確かめます。

**場面 2**：時系列の売上の予測で、データ全体をシャッフルしてから学習とテストに分けました。テストの精度は高いのに、本番では外れます。原因は、未来の日付のデータが学習に入ったことです（リーケージ）。時系列は時刻の順に分け、過去で学習して未来で評価します。



## サンプルはここまでです

続き(第2章〜巻末)は、AWS MLA-C02 問題集のご購入で、頻出度別ノート・暗記用ノートと合わせて3点すべてダウンロードできます。

# AWS MLA-C02 学習用テキスト(無料サンプル)

**版**

2026年9月版(AWS 公式 Exam Guide（MLA-C02・2026年9月29日取得）準拠)

**発行**

IT資格道場(https://www.shikaku-doujou.com)

**お問い合わせ**

info@shikaku-doujou.com