エンタープライズ規模でのサプライチェーン最適化

Red Hat* OpenShift* Data Science とインテル® アーキテクチャーを使用してオープンソースの AI テクノロジーを活用

Ted Jones Red Hat シニア・ソリューション・アーキテクト
Karl Eklund Red Hat 主席アーキテクト
Karol Brejna インテル コーポレーション クラウド・ソリューション・アーキテクト
Piotr Grabuszynski インテル コーポレーション クラウド・ソリューション・エンジニア

サプライチェーンの崩壊、新型コロナウイルス感染症の影響、停滞する経済状況は、小売業者が直面している課題のほんの一部です。ビジネスプロセスの各段階で、起業家は会社の収益、競争力、または将来の方向性を決定する決断を下す必要があります。最適なソリューションの開発には、時間がかかり、要求が厳しく、誤りが発生しやすくなりがちです。

意思決定には、過去のデータによる裏付けが必要ですが、数千もの注文と配送から得られる膨大な量のデータを人間が分析することは困難なため、意思決定プロセスは、マシンラーニング（ML）を使用した自動メカニズムによってサポートするか、完全に置き換えるべきです。ML は、マシンがより多くのデータを分析して学習し、精度を向上するプロセスです。数百万ものエントリーでも問題なく処理できます。データ間で見つかった関係に基づいて数学的モデルが作成され、後で動作、結果、価格、配送時間などを予測するために使用されます。

ML を使用して問題を解決するには、別の角度から見ることも重要です。企業のプロセスを人工知能（AI）アルゴリズムでサポートするには、まず適切なモデルを準備しなければなりません。しかし、これには適切なツールを使用する必要があります。そのため、インテルと Red Hat は、ML 関連のプロセスを促進する新しいソリューションに常に取り組んでいます。Red Hat* OpenShift* Data Science などの製品は、データ転送、データマイニング、トレーニング、プロセスの自動化、モデルのデプロイを改善し、インテルは、計算効率を向上させるため、scikit-learn* 向けインテル® エクステンション（英語）（インテル® AI アナリティクス・ツールキットの一部）などの最適化されたハードウェアとソフトウェアを提供しています。

問題ステートメント

増え続ける問題の 1 つは、配送日の見積もりです。物資の不足は、利益を減少させ、コストを増加させる可能性があります。そのため、プロセスを制御し、関連するリスクを管理することが重要です。以下の例では、過去のデータを使用して遅配を予測する方法を示しています。遅配の可能性が高い場合、例えば、倉庫から追加の在庫を出したり、代替部品を使用したり、あるいはサプライヤーを変更するという決定が下されます。

アプローチの説明

配送遅延予測は分類問題の一種です。この場合、目標は、過去の納入履歴や交通データなどの一連の入力に基づき、配送が遅れるか、時間どおりかを予測します。これらの特徴は、モデルのトレーニングに使用されます。トレーニング済みのモデルは、新しい未知のデータに対して予測を行うために使用できます。このアプローチでは、1 つのシンプルで高速なモデルを使用して、すべての注文を「スキャン」し、遅延の可能性を調べます。リスクのある注文は、さらに詳しく分析できます（例えば、遅延の潜在的な理由を示すほかのアルゴリズムを使用するなど）。

テクノロジー

Red Hat* OpenShift* Data Science

Red Hat* OpenShift* Data Science は、インテリジェント・アプリケーションのデータ・サイエンティストやプログラマー向けのサービスであり、セルフマネージドまたはマネージド・クラウド・プラットフォームとして利用できます。ML モデルを実際の環境にデプロイする前に、迅速に開発、トレーニング、テストできる、完全にサポートされた環境を提供します。オンプレミス、パブリッククラウド、データセンター、エッジのいずれであっても、コンテナ上の本番環境で ML モデルをデプロイし、Red Hat* OpenShift* Data Science からほかのプラットフォームに簡単にエクスポートできます。

ML に Red Hat* OpenShift* Data Science を使用すると、多くの利点があります。このプラットフォームには、Jupyter* Notebook、TensorFlow*、PyTorch*、scikit-learn* など、データ・サイエンティストがワークフローで使用するさまざまな商用パートナーとオープンソースのツールおよびフレームワークが含まれています。Red Hat* OpenShift* Data Science は、安全でスケーラブルな環境を提供します。

インテル® AI アナリティクス・ツールキット

インテル® AI アナリティクス・ツールキットは、XGBoost、scikit-learn*、TensorFlow*、PyTorch* など、インテルにより最適化された AI ソフトウェア、ライブラリー、フレームワークのセットを提供します。インテル® アーキテクチャー上のエンドツーエンドのデータサイエンスとアナリティクスのパイプラインを高速化するのに役立ちます。このツールキットは、AI アプリケーションの作成をスピードアップして簡素化することを目的としています。また、複数のプラットフォームへのモデルのデプロイもサポートします。

シナジー

以下に示す例は、インテル® AI アナリティクス・ツールキットと、Red Hat* OpenShift* Data Science に組み込まれている Jupyter* Notebook カーネルを使用して作成しました。これは、データサイエンスとアナリティクスに安定した開発環境を提供するシンプルでエレガントな方法です。

リファレンス実装

ML ソリューションの作成プロセスには、通常いくつかのステップが含まれます。

- 問題の定義
- データの収集
- 環境の準備とツールの選択
- データの前処理と特徴量エンジニアリング
- トレーニング・データの準備
- 最適な ML アルゴリズムの選択
- トレーニング
- モデルの検証
- モデルの配布とデプロイ

問題の定義

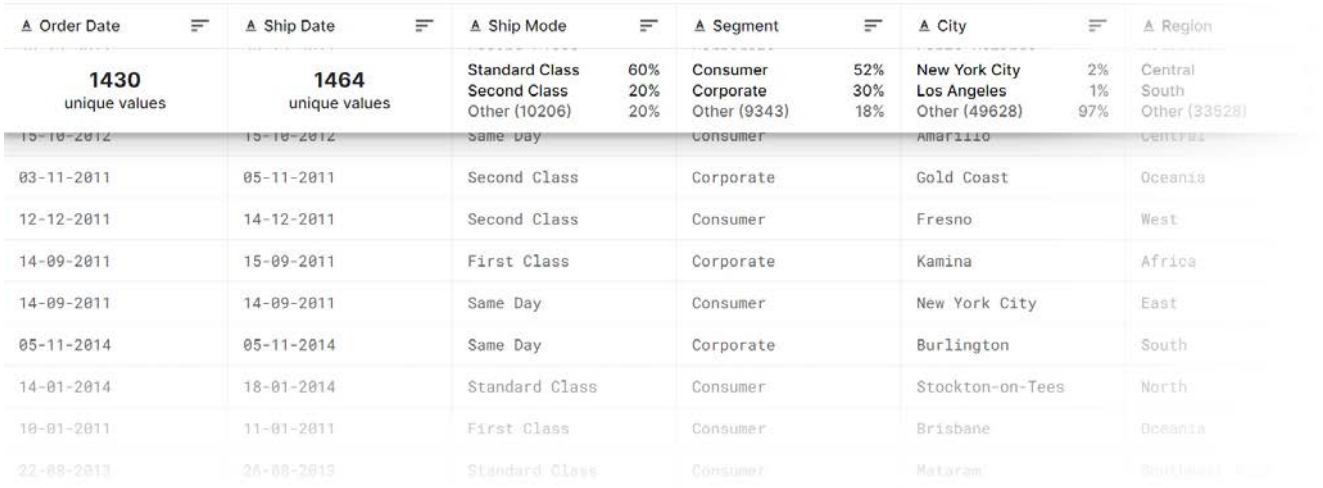
まず、解決する問題の種類を決定します。問題が複雑になるほど、ソリューションが見つかる可能性は低くなります。これは一般的なルールではありませんが、問題を単純化することから始める必要があります。この例では、配送が開始される前に潜在的な配送の問題を示すシンプルなソリューションが必要です。

データの収集

必要な答えが分かったら、その答えが含まれている可能性のあるデータを探します。場合によっては、複数のデータベースやファイルを調べ、それらを 1 つの統一されたセットに結合する必要があります（この記事のすべての例は、Kaggle*（英語）プラットフォームで公開されているオープンソースのデータに基づいています）。以下の例では、次の特徴を持つ過去のレコードを含む CSV ファイルを保有していると仮定します。

- 注文 ID
- 注文日
- 配送日
- 配送モード
- 注文者 (ID、名前)
- 顧客区分 (コンシューマー、法人 ...)
- 配送先 (市町村、都道府県、国)
- 市場コード (US、APAC...)
- 地域
- 製品 (ID、カテゴリー)
- 注文内容 (価格、数量、割引、配送料)
- 注文優先順位

図 1 はデータの一部を示しています。



Order Date	Ship Date	Ship Mode	Segment	City	Region
1430 unique values	1464 unique values	Standard Class 60% Second Class 20% Other (10206) 20%	Consumer 52% Corporate 30% Other (9343) 18%	New York City 2% Los Angeles 1% Other (49628) 97%	Central South Other (33528)
15-10-2012	15-10-2012	Same Day	Consumer	Amarillo	Central
03-11-2011	05-11-2011	Second Class	Corporate	Gold Coast	Oceania
12-12-2011	14-12-2011	Second Class	Consumer	Fresno	West
14-09-2011	15-09-2011	First Class	Corporate	Kamina	Africa
14-09-2011	14-09-2011	Same Day	Consumer	New York City	East
05-11-2014	05-11-2014	Same Day	Corporate	Burlington	South
14-01-2014	18-01-2014	Standard Class	Consumer	Stockton-on-Tees	North
10-01-2011	11-01-2011	First Class	Consumer	Brisbane	Oceania
22-08-2013	26-08-2013	Standard Class	Consumer	Mataram	Southeast Asia

図 1. データセットを視覚化

また、運送会社と交わしたサービス・レベル・アグリーメント（SLA）があります。配送時間は SLA の規定に沿っている必要があります。

- 配送モードが即日（Same Day）の場合は即日配送
- 第 1 種郵便（First Class）の場合は翌日配送
- 第 2 種郵便（Second Class）の場合は 3 日以内に配送
- 普通郵便（Standard Class）の場合は 5 日以内に配送

環境の準備とツールの選択

Jupyter* Notebook などの開発プロセスをサポートするツールを使用して、プロジェクトの作業を円滑に進めることができます。Red Hat* OpenShift* Data Science UI は、このような環境を素早く立ち上げることができます（**図 2**）。インテルにより最適化されたソフトウェアがすでに統合されているため、環境の作成と管理が容易です。

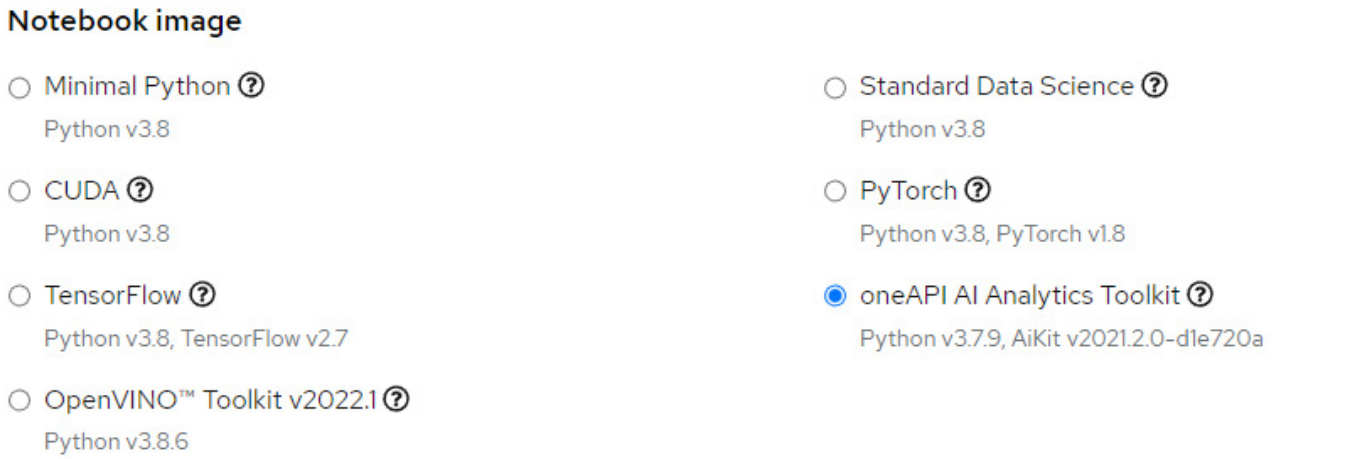


図 2. Red Hat* OpenShift* Data Science インターフェイスで Jupyter* Notebook を作成

ソリューションに適した AI フレームワークを選択します。オープンソースの ML ライブラリーとして人気の高い scikit-learn* は、教師あり / 教師なしトレーニングの幅広いアルゴリズムを提供しています。scikit-learn* 向けインテル® エクステンションは、インテル® ハードウェア上でパフォーマンスを向上します（「データ・サイエンティストが必要とする scikit-learn* のパフォーマンスをインテルが大幅に向上」(英語)を参照）。インテル® AI アナリティクス・ツールキットのコンテナ内で有効にするには、2 行のコードを追加するだけです。

```
from sklearnex import patch_sklearn
patch_sklearn()
```

この例では、scikit-learn* 向けインテル ® エクステンション・ライブラリーがインストールされた Red Hat* OpenShift* Data Science をベースにしています。提案するソリューションのアーキテクチャーを**図 3** に示します。

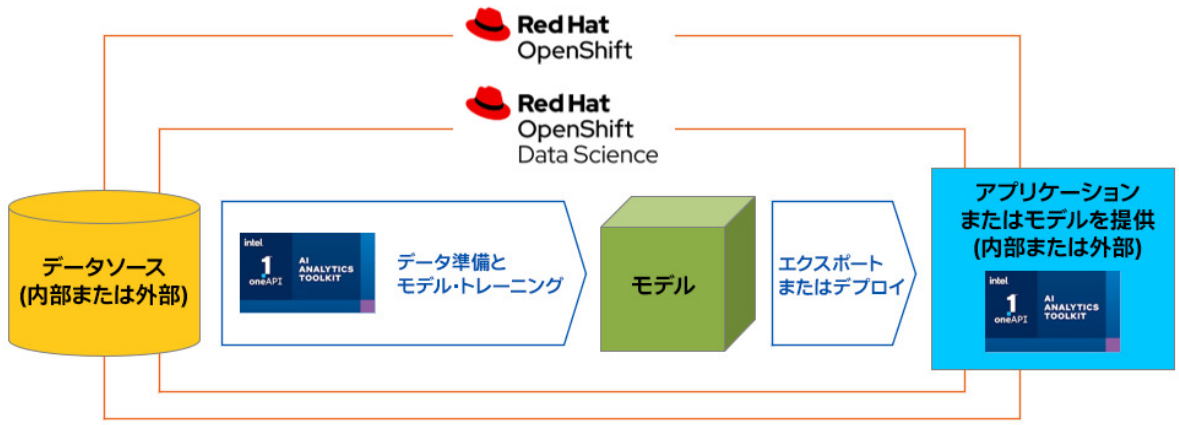


図 3. この例のソリューション・アーキテクチャー

データの前処理と特徴量エンジニアリング

すべてのデータを収集したら、それらをどのように処理するかを決定します。前処理では、データセットのクリーニングとフォーマット調整を行います。これには、欠損値や矛盾の除去、外れ値の処理、データセットのスケーリングなどが含まれます。次に、トレーニングに使用する特徴を決定します。特徴量エンジニアリングでは、データを選択して新しい特徴に変換し、データ内のパターンや関係をより適切に表現します。この例では、問題に対する明快な答えはありませんが、注文日、配送日、配送モード、SLA 条件が分かれば、遅延を計算することができます。

まず、データを読み込んで、pandas の DataFrame 形式で保存します。

```
df = pd.read_csv(DATA_PATH, encoding='latin-1', parse_dates=True)
```

日付をより便利な形式に変換し、配送時間を日数で計算する必要があります。

```
df['Order Date'] = pd.to_datetime(df['Order Date'], format="%d-%m-%Y")
df['Ship Date'] = pd.to_datetime(df['Ship Date'], format="%d-%m-%Y")

df['Delivery Time'] = (df['Ship Date'] - df['Order Date']).dt.days
```

このデータと SLA から遅延を計算できます。新しいカラムを作成し、ラベルを付けます:遅延(1)または時間どおり(0)。

```
df['SLA'] = df['Ship Mode'].map({
    'Same Day': 0,
    'First Class': 1,
    'Second Class': 3,
    'Standard Class': 5,
})

df['Delay in Days'] = (df['Delivery Time'] - df['SLA']).map(lambda d: d if d > 0 else 0)

df['Delay'] = df['Delay in Days'].map(lambda d: 1 if d > 0 else 0)

df['Delay'].value_counts()
```

```
0 31741
1 19549
Name: Delay, dtype: int64
```

値「Delay」が答えです。この例では、31,741 件が時間どおりで、19,549 件が遅延です。「Delay」カラムは、トレーニング中にターゲット（予測する値）として使用します。

トレーニング・データの準備

これまでのプロセスは、必要な特徴を含むコレクションの構築に重点を置いていました。有効なモデルを作成するには、データのバランス、分類、シャッフルなど、さらに多くの操作が必要になるかもしれません。その後、データをトレーニング・セットとテストセットに分割します。この例では、遅延と時間どおりの比率は約 2：3 であり、これは許容範囲ですが、データを分類する必要があります。

```
# 分類するカラム
CATEGORIZE = [
    'Ship Mode',
    'Customer ID',
    'Segment',
    'City',
    'State',
    'Country',
    'Postal Code',
    'Market',
    'Region',
    'Product ID',
    'Category',
    'Sub-Category',
    'Order Priority',
]

pd_categories = {}

for c in CATEGORIZE:
    pd_categories[c] = pd.Categorical(df[c])
    df[c] = pd_categories[c]
    df[f'{c}-cat'] = df[c].cat.codes

df.head()
```

分類後、データをトレーニング・セットとテストセットにランダムに分割します。

```
m = np.random.rand(len(pds)) < 0.8

ds_train = pds[m]
ds_test = pds[~m]
```

最適なアルゴリズムの選択

問題に応じて、教師あり / 教師なしアルゴリズムを使用できます。この決定を行う際には、データの種類、目標、パフォーマンス、および精度を考慮することが重要です。アルゴリズムの評価と比較には、予測精度や曲線下面積（AUC）などの既知のメトリックを使用できます。AUC は、真陽性と偽陽性の予測プロットの下の面積として計算されます。したがって、常に 0 から 1 の間の数値になります（大きいほど良い）。

アプリケーションに最適なアルゴリズムを確認します。ガウス・ナイーブ・ベイズ、ランダムフォレスト、LightGBM、決定木、XGBoost、K 近傍法の 6 つの異なる分類器からモデルをトレーニングして評価します。アルゴリズムごとに、平均 AUC と 20 反復の実行時間を計算します（**図 4**）。結果から、このデータに最適なモデルはランダムフォレストであることが分かります。ここでは、ランダムフォレストを使用します。

トレーニング

```
from statistics import mean

NUMBER_OF_ITERATIONS = 20

time_results = {}
roc_auc_results = {}

for _ in range(NUMBER_OF_ITERATIONS):
    models = [
        ("NB",     GaussianNB()),
        ("RF",     RandomForestClassifier()),
        ("lgbm",   lgb.LGBMClassifier()),
        ("Dtree",  DecisionTreeClassifier()),
        ("XGB",    xgb.XGBClassifier()),
        ("KN",     KNeighborsClassifier()),
    ]

    for name, model in models:
        start_time = time()
        kf = KFold(shuffle=True, random_state=7919)
        cv_result = cross_val_score(model, ds_train_features, ds_train_target, cv=kf, scoring="roc_auc")
        stop_time = time()

        mean_cv_results = mean(cv_result)
        print(name, cv_result, mean_cv_results)

        total_t = stop_time - start_time
        try:
            time_results[name].append(total_t)
        except KeyError:
            time_results[name] = [total_t]

        try:
            roc_auc_results[name].append(mean_cv_results)
        except KeyError:
            roc_auc_results[name] = [mean_cv_results]
```

```
------------------------ RESULTS FOR OPTIMIZED SKLEARN ------------------------
| name   | iters | avg time      | min time      | max time      | avg auc       |
| NB     | 20    | 0.0972        | 0.0884        | 0.1522        | 0.6808        |
| RF     | 20    | 2.4618        | 2.3030        | 2.8867        | 0.9237        |
| lgbm   | 20    | 1.4740        | 1.0682        | 3.2824        | 0.8647        |
| Dtree  | 20    | 1.0654        | 1.0400        | 1.1016        | 0.8174        |
| XGB    | 20    | 9.0131        | 5.8058        | 26.3675       | 0.8811        |
| KNN    | 20    | 0.2834        | 0.2675        | 0.3175        | 0.5271        |
```

図 4. 簡単な評価の結果

ML トレーニングは、入力特徴をターゲットにマップするデータ内のパターンを見つけます。トレーニング済みのモデルにはこれらの依存関係が含まれるため、将来の値を予測できます。ソリューションの開発中に、選択したアルゴリズムとライブラリーに応じて、トレーニング・プロセスの設定を変更できます。これらの設定を調整すると、モデルが改善または悪化する可能性があるため、さまざまな値を試してみる価値があります。この例では、`class_weight` パラメーターを設定して、データのわずかな不均衡を補正します。それ以外は、デフォルトのパラメーターを使用したトレーニングで最良の結果が得られました。

```
rfc = RandomForestClassifier(
    class_weight={0:2, 1:3}
)

rfc.fit(ds_train_features, ds_train_target)
```

最終モデルの検証

MLモデルの検証は、未知のデータに対してその精度を評価するプロセスです。モデルを実世界にデプロイする前に、データ・サイエンティストが問題を発見して修正できるため、このステップは非常に重要です。精度が低すぎる、または非常に高い場合は、データが少なすぎる、重要な特徴がない、データ漏洩、学習不足または過学習などの潜在的な問題を示している可能性があります。基本的な評価には、前述の精度と AUC メトリックを使用できます。検証プロセスを開始し、データを収集するコードを以下に示します。

```
ds_test_features = ds_test[FEATURES]
ds_test_target = ds_test[TARGET]

preds = rfc.predict(ds_test_features)
preds_prob = rfc.predict_proba(ds_test_features)[:, 1]

acc_score = accuracy_score(ds_test_target, preds)
auc_score = roc_auc_score(ds_test_target, preds_prob)

print(f'acc: {acc_score} | auc: {auc_score}')
```

```
acc: 0.8664118396550035 | auc: 0.9381309986948686
```

約 86% の精度を達成し、AUC は 0.93 を上回りました。データ準備段階でのランダムな要因（シャッフルなど）により、結果がわずかに異なる場合があります。

ソリューションのベースラインを計算して、健全性チェックを行います。毎回「0」と答えるモデルをシミュレートし、その精度を計算します。

```
accuracy_score(ds_test_target, np.zeros_like(ds_test_target)), roc_auc_score(ds_test_target, np.zeros_like(ds_test_target))
```

```
(0.6153169873986519, 0.5)
```

ベースラインの精度は約 62% なので、ベースラインと比較して大きな改善が見られます。

結果を視覚化するため、グラフを使用できます。その 1 つが、受信者動作特性 (ROC) 曲線です。**図 5** は、今回の実験における ROC 曲線を示したものです。この曲線の下の面積が AUC 値です。

```
%matplotlib inline

fpr, tpr, thresholds = roc_curve(ds_test_target, preds_prob)

plt.plot(fpr, tpr)
plt.show()
```

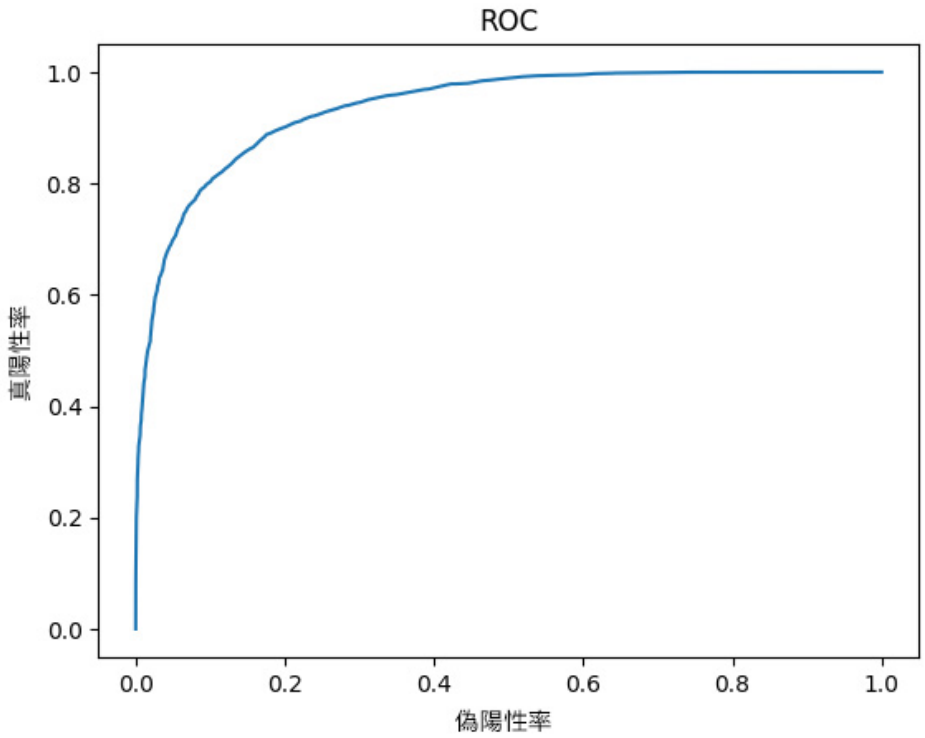


図 5. この実験における ROC 曲線

また、どの特徴が出力に最も大きな変化をもたらすか視覚化することができます（**図 6**）。

```
importances = rfc.feature_importances_
std = np.std([tree.feature_importances_ for tree in rfc.estimators_], axis=0)

forest_importances = pd.Series(importances, index=FEATURES)

fig, ax = plt.subplots()
forest_importances.plot.bar(yerr=std, ax=ax)
fig.tight_layout()
```

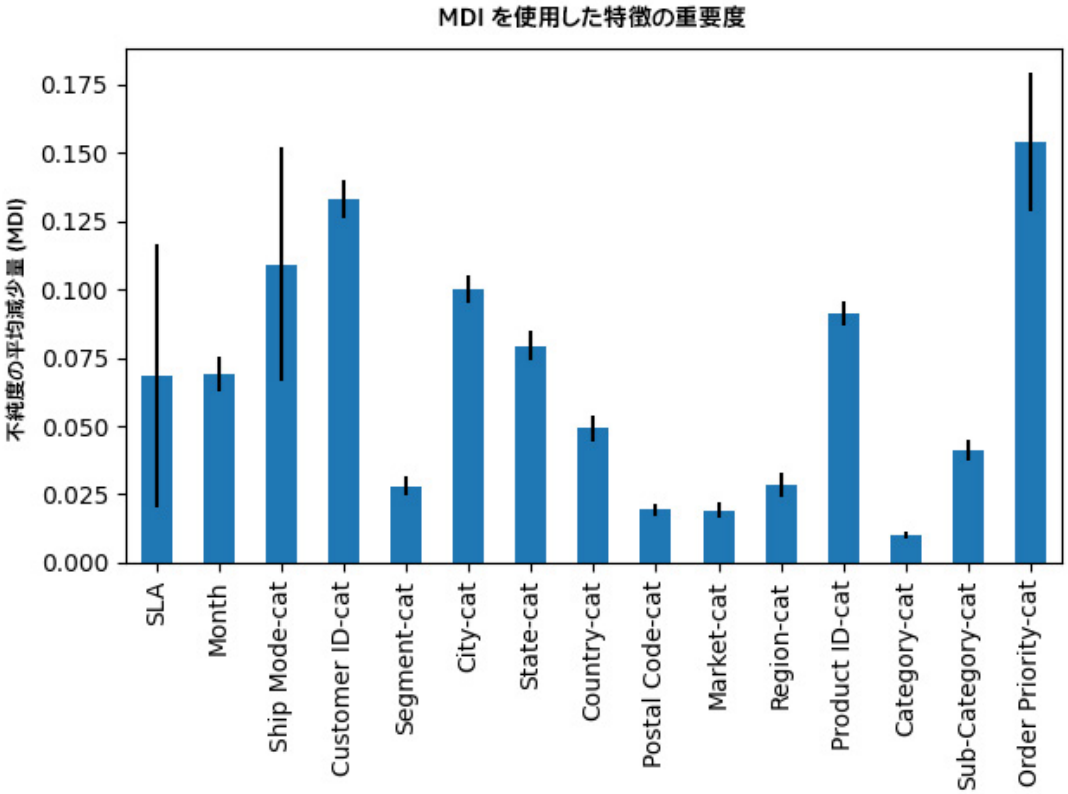


図 6. この例の特徴の重要度を示す図

推論の実行：簡単な方法

健全性チェックのため、同じ Jupyter* Notebook の中でモデルを使用してみます。検証データから 1 行を選択し、サンプルを用意します。

```
INPUT_DATA = {
    'Customer ID':     'AA-00000',
    'Order Date':      '12-07-2022',
    # …
    'Quantity':        '12',
    'Order Priority':  'Medium',
}
```

このモデルを使用するには、トレーニング時に行ったように、入力データを変換する必要があります。トレーニング時に使用したのと同じカテゴリーを使用することが重要です。例えば、「New York」が 129 にマッピングされていた場合、ここでも同じ数字を使用する必要があります。

```
INPUT_DATA['Postal Code'] = float(INPUT_DATA['Postal Code'])

input_d = pd.DataFrame(data={k: [v] for k, v in INPUT_DATA.items()})

udf = pd.DataFrame()

# SLA
udf['SLA'] = input_d['Ship Mode'].map({
    'Same Day': 0,
    'First Class': 1,
    'Second Class': 3,
    'Standard Class': 5,
})

# 月
udf['Month'] = pd.to_datetime(input_d['Order Date'], format="%d-%m-%Y").dt.month

# 配送モード - cat
for c in CATEGORIZE:
    try:
        udf[f'{c}-cat'] = pd_categories[c].categories.get_loc(INPUT_DATA[c])
    except KeyError:
        udf[f'{c}-cat'] = -1

udf.head()
```

	SLA	Month	Ship Mode-cat	Customer ID-cat	Segment-cat	City-cat	State-cat	Country-cat	Postal Code-cat	Market-cat	Region-cat	Product ID-cat	Category-cat	Sub-Category-cat	Order Priority-cat
0	0	7	1	-1	0	375	145	47	-1	4	3	8246	2	0	3

推論を実行します。

```
if_delay = not rfc.predict(udf)[0]
proba_of_delay = rfc.predict_proba(udf)[:, 1][0]
proba_of_delay = (1 - proba_of_delay if if_delay else proba_of_delay) * 100

print('We expect' + (' no' if if_delay else '') + f' delay (with probability ~{proba_of_delay:.0f}%)')
```

```
We expect no delay (with probability ~93%)
```

まとめ

今日の起業家は、サプライチェーンの崩壊、新型コロナウイルス感染症の影響、停滞する経済状況などの困難に直面しています。小売業者は、収益、競争力、または会社の将来の方向性に影響を与える選択を、ビジネスプロセスのあらゆるレベルで決断する必要があります。ML は、意思決定プロセスを大幅に簡素化し、その精度を向上できます。この記事では、例を使って小売配送の遅延を検出する方法を示しました。

企業は、過去のデータに基づいた手法とアルゴリズムを使用して、潜在的な配送遅延を警告するソリューションを作成できます。このアプローチにより、すべての配送を迅速かつ高精度にスキャンできます。問題のある注文は、さらに詳しく分析することが可能です。例えば、遅延時間、遅延に最も影響を与える要因、部品やサービスを時間どおりに提供できるサプライヤーなどの情報を得られます。詳しい分析では、サプライヤーが提供する製品、カテゴリー、またはデータに応じて異なるアルゴリズムを使用し、複数のモデルを生成して予測精度を高めることができます。

Red Hat* OpenShift* Data Science と scikit-learn* 向けインテル® エクステンションは、データサイエンスと分析プロセスを改善する強力な組み合わせです。インテルにより最適化されたライブラリーを含む Red Hat* OpenShift* Data Science に組み込まれた Jupyter* Notebook のおかげで、環境を迅速かつ容易に準備できます。