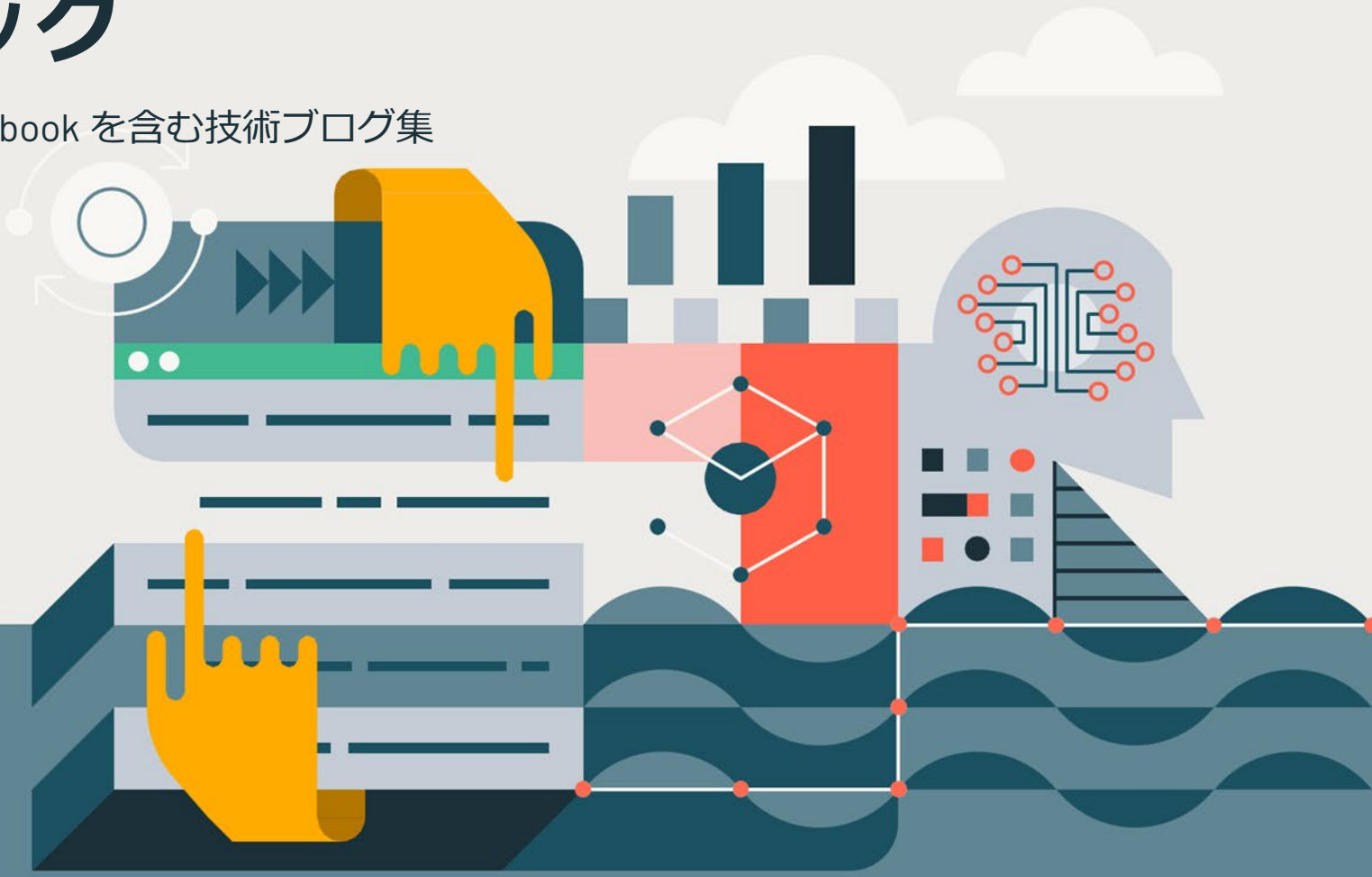
eBook

機械学習ユースケースのビッグブック

コードサンプルと Notebook を含む技術ブログ集



databricks

目次

第1章

序章

データサイエンスの世界は急速に進化しており、自組織の取り組みと関連性の深い、実用的なユースケースを見つけるのは容易ではありません。そこで、業界のオピニオンリーダーによるブログを集約し、実用的なユースケースのリファレンスガイドとして発行することにしました。Databricksプラットフォームをすぐに試していただけるよう、コードサンプルをはじめとする必要な情報を全て記載しています。

第 2 章

Part 1：動的タイムワープの概要

動的タイムワープと MLflow を活用した販売傾向の把握 - Part 1

投稿者：
Ricardo Portilla
Brenner Heintz
Denny Lee

2019 年 4 月 30 日

この Notebook を試してみる

概要

「ダイナミック・タイム・ワープ」（DTW：動的タイムワープ）という言葉を最初に読むと、「バック・トゥ・ザ・フューチャー」シリーズの中で、マーティ・マクフライが時速88マイルでデロリアンを運転しているイメージを思い浮かべるかもしれません。しかし、動的タイムワープはタイムトラベルではなく、比較データポイント間の時間指標が完全に同期していない場合に、時系列データを動的に比較するために使用される技術です。

後述のように、動的タイムワープの最も顕著な用途の 1 つは音声認識です。これは、 Google Home や Amazon Alexa デバイスを起動するための「目覚めの言葉」を識別するのに便利であることが想像できます。

動的タイムワープは、多くの異なる領域に適用できる便利で強力なテクニックです。動的タイムワープの概念を理解すると、日常生活での応用例や、将来的な応用例を簡単に見ることができます。以下の用途を考えてみてください。

- **金融市場：**完全に一致していなくても、似たような期間の株式売買データを比較すること。例えば、2月（28日）と3月（31日）の月次取引データを比較する。
- **ウェアラブルフィットネス器具のトラッカー：**歩行者の速度が時間の経過とともに変化した場合でも、歩行者の速度と歩数をより正確に計算できるようになりました。
- **経路の計算：**ドライバーの運転習慣について何か知っていれば、ドライバーの ETA に関するより正確な情報を計算することができます（例えば、彼らは直線道路を素早く運転しているが、左折するのに平均よりも時間がかかるなど）。

データサイエンティスト、データアナリスト、時系列データを扱う人は、データを整理することには慣れているため、このテクニックを容易に理解できるはずです。

このブログシリーズでは、以下のことについて探っていきます。

- 動的タイムワープの基本原理
- サンプルオーディオデータで動的タイムワープを実行する
- MLflow を用いたサンプル販売データの動的タイムワープの実行

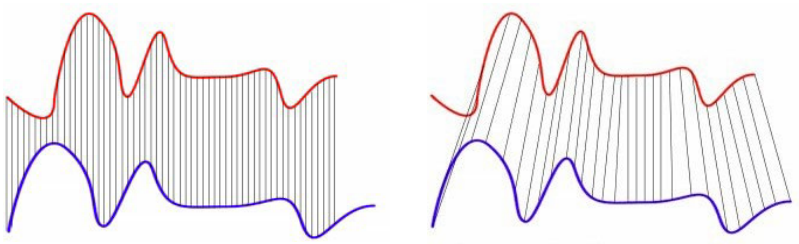
動的なタイムワープ

時系列比較法の目的は、2 つの入力時系列間の距離メトリックを生成することです。2 つの時系列の類似性または非類似性は、通常、データをベクトルに変換し、ベクトル空間内のそれらの点間のユークリッド距離を計算することによって計算される。

Dynamic time warping は、1970 年代から音波を音源として音声認識や単語認識に用いられてきた時系列比較技術であり、『Dynamic time warping for isolated word recognition based on ordered graph searching techniques』という論文がしばしば引用されます。

背景

この技術はパターンマッチングだけでなく、異常検知にも利用可能です（例：2 つの不連続な期間の時系列を重ね合わせて、形状が大きく変化した場合や、外れ値を調べる場合）。例えば、下のグラフの赤と青の線では、伝統的な時系列マッチング（ユークリッドマッチング）が非常に制限的です。一方、動的な時系列ワープ を使用すると、X 軸（すなわち時間）がずれていても、2 つの曲線を均等に一致させることができます。は必ずしも同期しているとは限りません。もう一つの方法は、これをロバストな非類似性スコアとして考えることです。この場合は、数字が小さいほどシリーズの類似性が高いことを意味します。



ユークリッドマッチング　　**動的タイムワープマッチング**

出典： Wiki Commons（Euclidean_vs_DTW.jpg）

二時系列（基準時系列と新時系列）は、以下のルールに従って関数 f(x)を用いて、最適（ワープ）パスを用いて大きさを一致させるように写像することができれば、似たようなものとみなされます。

$$f(x_i) \text{ maps to } f(x_j) \text{ when } i <= j$$

$$f(x_i) \text{ maps to } f(x_j) \text{ only when } (j - i) \text{ is within fixed range}$$

サウンドパターンマッチング

伝統的に、動的タイムワープは、オーディオクリップに適用され、それらのクリップの類似性を判断します。この例では、“The Expanse” というテレビ番組からの 2 つの異なる引用に基づいて、4 つの異なるオーディオクリップを使用します。4 つのオーディオクリップ（以下で聞くことができますが、これは必須ではありません）があり、そのうちの 3 つ（クリップ 1、2、4）は引用文に基づいています。

“Doors and corners, kid. That’s where they get you.”

そして、1 つのクリップ（クリップ 3）が引用です。

“You walk into a room too fast, the room eats you.”

Clip 1 Doors and corners, kid. That’s where they get you. [v1]

0:00 / 0:06

Clip 2 Doors and corners, kid. That’s where they get you. [v2]

0:00 / 0:08

Clip 3 You walk into a room too fast, the room eats you.

0:00 / 0:07

Clip 4 Doors and corners, kid. That’s where they get you. [v3]

0:00 / 0:07

出典：“The Expanse”

以下は、4 つのオーディオクリップの matplotlib を使用した可視化です。

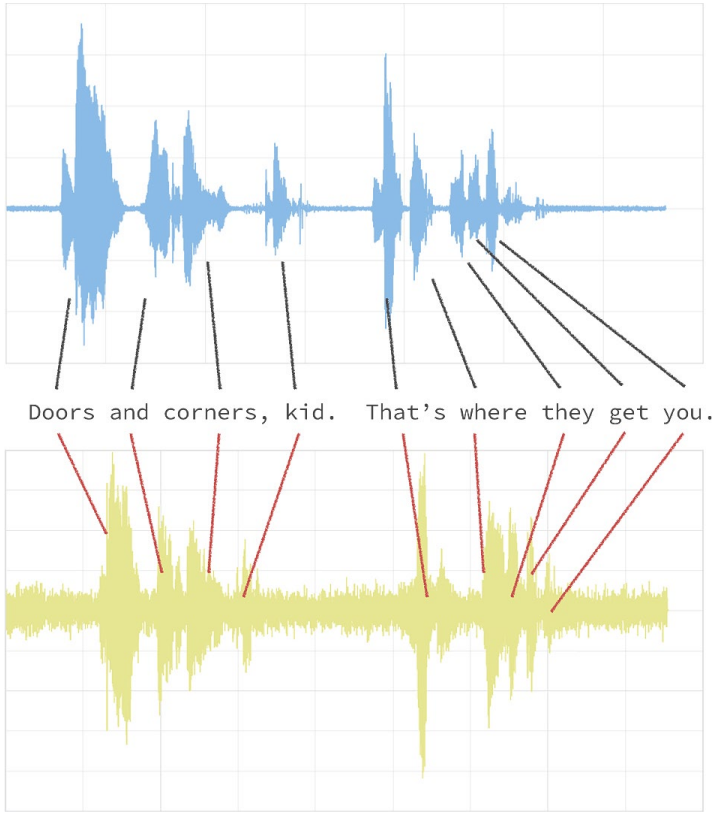
- クリップ 1：これは名セリフに基づく時系列です "Doors and corners, kid. That’s where they get you."
- クリップ 2：イントネーションや発話パターンが極端に誇張されているクリップ 1 をベースにした新しい時系列 [v2] です。
- クリップ 3：これもクリップ 1 と同じイントネーションとスピードで "You walk into a room too fast, the room eats you."
- クリップ 4：イントネーションや発話パターンがクリップ1と類似しているクリップ 1 を基にした新しい時系列 [v3] です。

これらのオーディオクリップを読み込み、matplotlib を使って可視化するコードのコードスニペットを以下に示します。

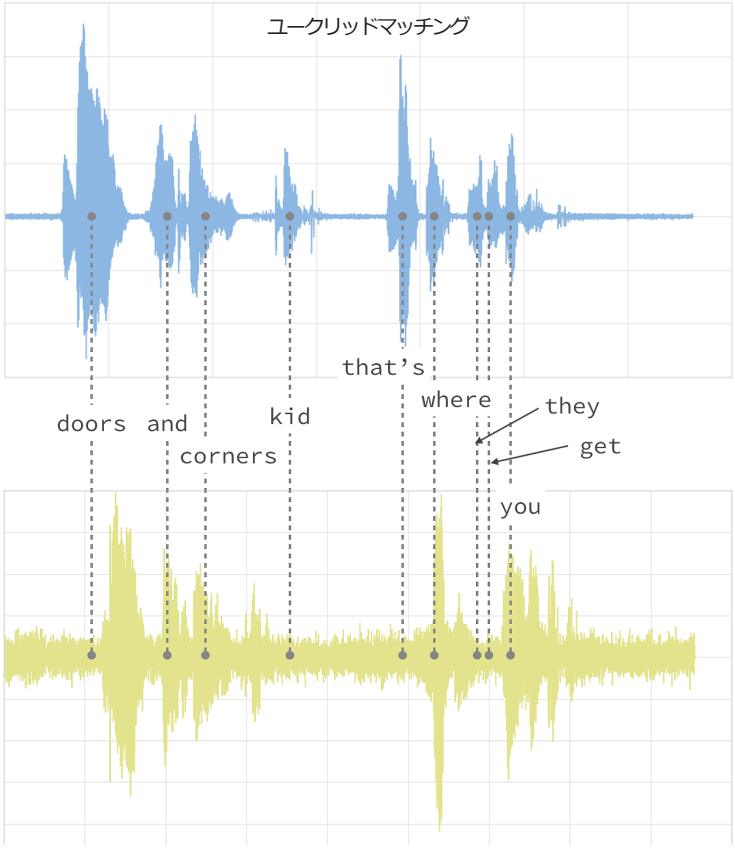
```
from scipy.io import wavfile
from matplotlib import pyplot as plt
from matplotlib.pyplot import figure

# Read stored audio files for comparison
fs, data = wavfile.read("/dbfs/folder/clip1.wav")

# Set plot style
plt.style.use('seaborn-whitegrid')

# Create subplots
ax = plt.subplot(2, 2, 1)
ax.plot(data1, color='#67A0DA')
...

# Display created figure
fig=plt.show()  display(fig)
```

完全なコードベースは、Notebook「Dynamic Time Warping Background」を参照してください。

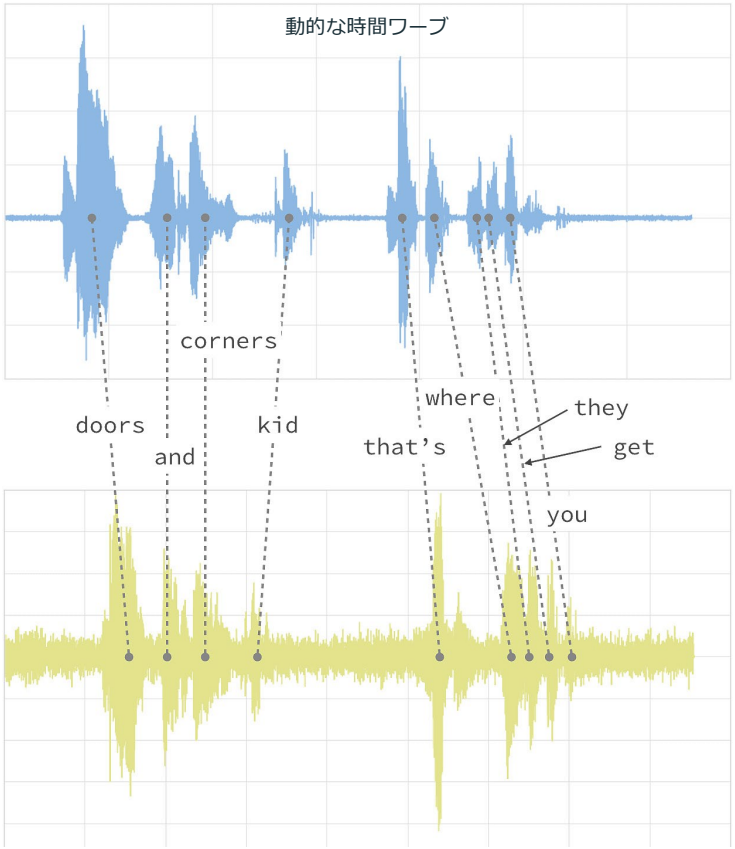
下図は、２つのクリップ（この場合はクリップ１とクリップ４）が同じ引用文に対してイントネーション（振幅）とレイテンシーが異なる様子を示しています。



従来のユークリッドマッチング（以下のグラフ）に従うと、振幅を割り引いても、元のクリップ（青）と新しいクリップ（黄）の間のタイミングは一致しません。



動的な時間ワープを使用して、これら 2 つのクリップ間の時系列比較を可能にするために時間をシフトさせることができます。

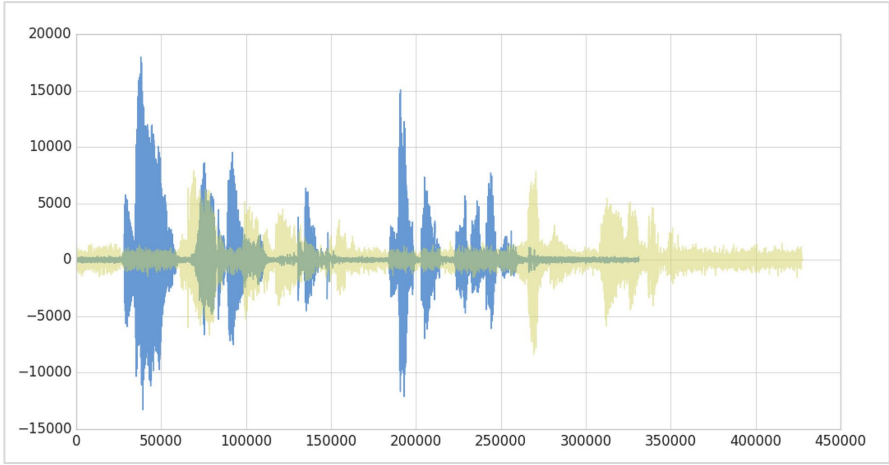


時系列比較には fastdtw PyPi ライブラリを使用します。Databricks のワークスペースに PyPi ライブラリをインストールする方法は、Azure | AWS を参照してください。fastdtw を使うことで、異なる時系列間の距離を素早く計算できます。

```
from fastdtw import fastdtw

# Distance between clip 1 and clip 2
distance = fastdtw(data_clip1, data_clip2)[0]
print("The distance between the two clips is %s" % distance)
```

完全なコードベースは、Notebook「Dynamic Time Warping Background」を参照してください。

ベース	クエリ	距離
Clip 1	Clip 2	480148446.0
	Clip 3	310038909.0
	Clip 4	293547478.0



概観：

- 前述のグラフにあるように、音声クリップが同じ単語とイントネーションであるため、クリップ 1 と 4 の距離が最も短くなっています。
- クリップ 1とクリップ3の間の距離もかなり短く（クリップ 4 と比較すると長いですが）、言葉は違ってもイントネーションやスピードは同じです。
- クリップ 1 と 2 は、同じ引用文を使っているにもかかわらず、イントネーションとスピードが極端に誇張されているため、最も距離が長くなっています。

上述のように、動的な時間ワープでは 2 つの異なる時系列の類似性を確認できます。

次のステップ

ここまで動的なタイムワープについて説明してきましたが、このユースケースを販売傾向の把握に適用してみましょう。

第 2 章

Part 2：動的タイムワープと MLflow を利用した販売傾向の把握

動的タイムワープと MLflow を活用した
販売傾向の把握 - Part 2

投稿者：
Ricardo Portilla
Brenner Heintz
Denny Lee

2019 年 4 月 30 日

Databricks の Notebook シリーズ（DBC 形式）を試す

背景

あなたが 3D プリント製品を作る会社を経営していると想像してみてください。昨年は、ドローンのプロペラが非常に安定した需要があることを知っていたので、それを製造して販売し、一昨年は携帯電話のケースを販売していました。**新しい年がすぐそこまで来ているので、製造チームと一緒に来年の生産物を考えようとしています。**あなたの倉庫のために 3D プリンタを購入することは、借金に深くあなたを入れたので、あなたのプリンタは、それらの支払いを行うために、全ての回で 100% の容量で実行されていることを確認する必要があります。

あなたは賢明な CEO として、来年の生産能力が変動することを予測しています。例えば、次のような週には生産能力が高くなるかもしれません。夏場（季節労働者を雇用する場合）、毎月第 3 週目に低くなります（3D プリンタのフィラメントのサプライチェーンに問題があるため）。下のチャートを見て、あなたの会社の生産能力の見積もりを見てみましょう。

毎週の需要と生産能力ができるだけ一致する製品を選択するという仕事を任されたと仮定します。あなたは、各製品の昨年の販売数を含む製品カタログに目を通しており、今年の販売数は似たようなものになると考えています。

生産能力を超えた週単位の需要がある商品を選ぶと、顧客からの注文をキャンセルしなければならなくなり、良くないビジネスのために。一方で、毎週の需要が十分にない商品を選んでしまうと、プリンタをフル稼働させることができず、借金の支払いに失敗する可能性があります。

ここでは、動的なタイム・ワープが有効に機能します。選択した製品の需要と供給が同期しない場合があるためです。需要を全て満たすのに十分な生産能力がない週もあるかもしれませんが、その前の週や後の週にもっと多くの製品を生産することで埋め合わせができるのであれば、顧客は気にしません。もし、ユークリッドマッチングを使って販売データと生産能力を比較することに限定すると、このことを考慮していない製品を選択してしまい、お金を置いていくことになるかもしれません。その代わりに、動的タイムワープを使用して、今年の貴社に最適な製品を選択します。

製品の販売データセットを読み込む

UCI データセットリポジトリにある週次売上高取引データセットを使用して、売上高ベースの時系列分析を行います。（出典：James Tan、jamestansc@suss.edu.sg、シンガポール社会科学大学）

```
import pandas as pd

# Use Pandas to read this data
sales_pdf = pd.read_csv(sales_dbfspath, header='infer')

# Review data
display(spark.createDataFrame(sales_pdf))
```

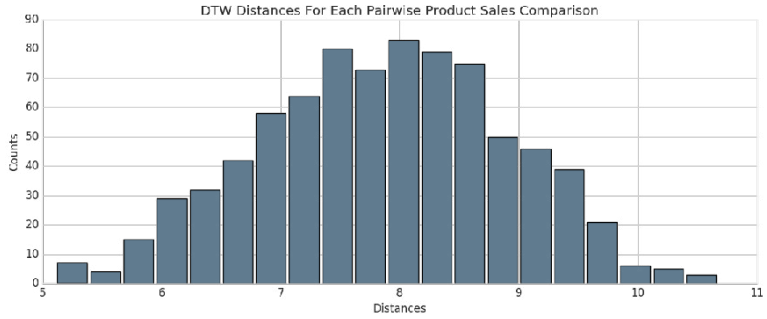
Product_Code	W0	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13
P1	11	12	10	8	13	12	14	21	6	14	11	14	16	9
P2	7	6	3	2	7	1	6	3	3	3	2	2	6	2
P3	7	11	8	9	10	8	7	13	12	6	14	9	4	7
P4	12	8	13	5	9	6	9	13	13	11	8	4	5	4
P5	8	5	13	11	6	7	9	14	9	9	11	18	8	4
P6	3	3	2	7	6	3	8	6	6	3	1	1	5	4
P7	4	8	3	7	8	7	2	3	10	3	5	2	3	4
P8	8	6	10	9	6	8	7	5	10	10	8	8	15	9

各製品は行で表され、年間の各週は列で表されます。値は週ごとに販売された各製品の単位数を表しています。データセットには 811 個の製品があります。

プロダクトコードで最適時系列までの距離を計算

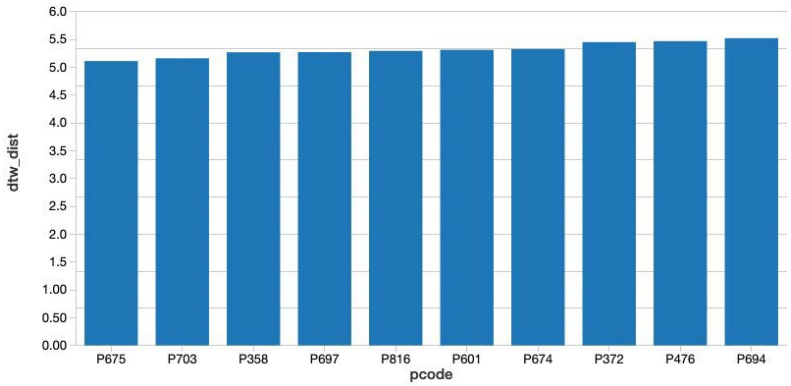
```
# Calculate distance via dynamic time warping between product code and
optimal time series
import numpy as np
import _ucrdtw

def get_keyed_values(s):
    return(s[0], s[1:])

def compute_distance(row):
    return(row[0], _ucrdtw.ucrdtw(list(row[1][0:52]),
list(optimal_pattern), 0.05, True)[1])

ts_values = pd.DataFrame(np.apply_along_axis(get_keyed_values, 1,
sales_pdf.values))
distances = pd.DataFrame(np.apply_along_axis(compute_distance, 1,
ts_values.values))
distances.columns = ['pcode', 'dtw_dist']
```

計算された動的時間ワープの「距離」列を使用すると、ヒストグラムで DTW 距離の分布を見ることができます。

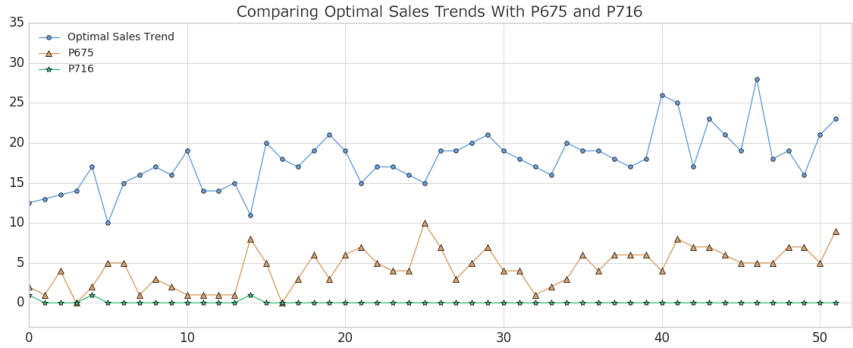


そこから、最適な販売動向に最も近い商品コード（計算された DTW 距離が最も小さいもの）を特定することができます。Databricks を使用しているため、SQL クエリで容易に選択できます。最も近いものを表示してみましょう。

```
%sql
-- Top 10 product codes closest to the optimal sales trend
select pcode, cast(dtw_dist as float) as dtw_dist from
distances order by  cast(dtw_dist as float) limit 10
```



このクエリを、最適な販売トレンドから最も遠い製品コードの対応するクエリとともに実行した結果、トレンドに最も近く、トレンドから最も遠い 2 つの製品を特定することができました。これら 2 つの製品をプロットして、それぞれの違いを見てみましょう。

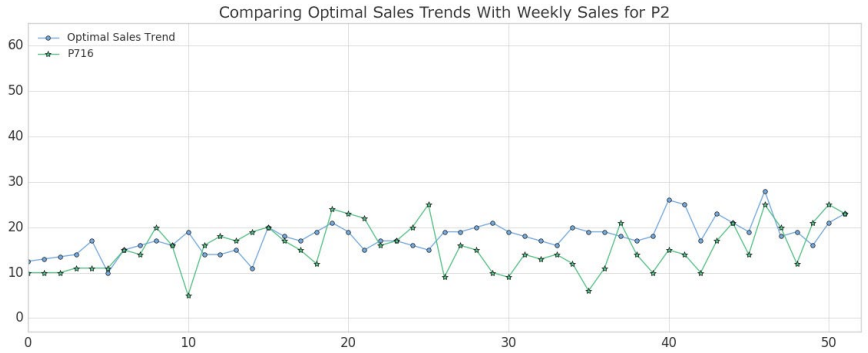


ご覧のように、製品 #675（オレンジ色の三角形）は、週次売上高の絶対値が思ったよりも低いものの、最適な販売トレンドに最もよくマッチしています（これについては後ほど修正します）。この結果は、DTW 距離が最も近い製品には、比較対象としているメトリクスを多少反映したピークとバレーがあると予想されるため、理にかなっています。（もちろん、製品の正確な時間指標は、動的な時間ワープのため、週ごとに異なります。）逆に、製品 #716（緑の星）は、最悪の一致を示す製品であり、ほとんど変動はありません。

最適な商品を探す — DTW の距離が小さく、絶対販売数が似ている場合

これで、工場の予想生産量（当社の「最適販売動向」）に最も近い製品のリストができたので、DTW 距離が小さい製品や絶対数が似ている製品に絞り込むことができるようになりました。候補としては、次のようなものが考えられます。製品 #202 は、DTW の距離が6.86 であるのに対し、人口の中央値の距離は 7.89であり、当社の最適なトレンドに非常に密接に追従しています。

```
# Review P202 weekly sales
 y_p202 = sales_pdf[sales_pdf['Product_Code'] == 'P202'].values[0][1:53]
```



MLflow を使用してアーティファクトとともにベスト・ワーストの製品を追跡する

MLflow は、実験、再現性、展開を含む機械学習のライフサイクルを管理するためのオープンソースのプラットフォームです。**Databricks の Notebook は、完全に統合された MLflow 環境を提供しており、実験の作成、パラメタやメトリクスのログ、結果の保存などを行うことができます。**MLflow を使い始めるための詳細については、ドキュメントをご覧ください。

MLflow の設計の中心は、各実験のインプットとアウトプットの全てを、体系的で再現性のある方法で記録できることにあります。データを通過するたびに、“Run“ として知られる、実験のログを記録することができます。

- **パラメタ（parameter）：**モデルへの入力
- **メトリクス（metrics）：**モデルの出力、またはモデルの成功の尺度
- **アーティファクト（artifact）：**モデルによって作成された全てのファイル - PNG プロットや CSV データ出力など
- **モデル（model）：**モデルそのものであり、後にリロードして予測値を提供するために使用することができます。

私たちの場合は、これを使用して、時系列データに適用できる最大のワープ量である「ストレッチファクター」を変化させながら、動的な時間ワープアルゴリズムをデータ上で数回実行することができます。MLflow の実験を開始し、`mlflow.log_param()`、`mlflow.log_metric()`、`mlflow.log_artifact()`、`mlflow.log_model()` を使って簡単にロギングできるようにするために、メイン関数を以下のようにラップします。

```
with mlflow.start_run() as run:
    ...
```

右記に簡略コードを示します。

```
import mlflow

def run_DTW(ts_stretch_factor):
   # calculate DTW distance and Z-score for each product
   with mlflow.start_run() as run:

      # Log Model using Custom Flavor
      dtw_model = {'stretch_factor' : float(ts_stretch_factor),
'pattern' : optimal_pattern}
      mlflow_custom_flavor.log_model(dtw_model, artifact_path="model")

      # Log our stretch factor parameter to MLflow
         mlflow.log_param("stretch_factor", ts_stretch_factor)

      # Log the median DTW distance for this run
         mlflow.log_metric("Median Distance", distance_median)

      # Log artifacts - CSV file and PNG plot - to MLflow
         mlflow.log_artifact('zscore_outliers_' + str(ts_stretch_factor) +
'.csv')
      mlflow.log_artifact('DTW_dist_histogram.png')

   return run.info

stretch_factors_to_test = [0.0, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5]
for n in stretch_factors_to_test:
   run_DTW(n)
```

データを実行するたびに、使用する「ストレッチファクター」パラメタのログと、DTW 距離メトリックのZスコアに基づいて外れ値として分類した製品のログを作成しました。さらに、DTW 距離のヒストグラムの成果物（ファイル）を保存もできました。**これらの実験の実行結果は Databricks 上にローカルに保存されており、後日でもアクセスしやすくなっています。**

MLflow が各実験のログを保存したので、結果を調べてみましょう。Databricks の Notebook から、右上の "Runs" アイコンを選択し、それぞれの実験の結果を表示して比較してみましょう。

驚くことではありませんが、「ストレッチ・ファクター」を増やすと、距離測定値は減少します。直感的には、これは理にかなっています。アルゴリズムに時間指標を前方または後方にワープさせる柔軟性を与えると、データに近いフィットを見つけることができます。本質的には、我々は分散のためにいくつかのバイアスを交換したのです。

MLflow でのロギングモデル

MLflow は、実験パラメタやメトリクス、成果物（プロットや CSV ファイルのようなもの）をログに記録するだけでなく、機械学習モデルをログに記録する機能を持っています。MLflow のモデルは、一貫した API に適合するように構造化されたフォルダであり、他の MLflow ツールや機能との互換性を確保しています。この相互運用性は非常に強力で、どのようなPythonモデルであっても、多くの異なるタイプの本番環境に迅速に展開することができます。

MLflow には、scikit-learn、Spark MLlib、PyTorch、TensorFlow などを含む、最もポピュラーな機械学習ライブラリの多くに共通のモデルフレーバーがプリロードされています。これらのモデルフレーバーは、このブログ記事で実証されているように、モデルが最初に構築された後にログを記録したり、再ロードしたりすることを容易にしてくれます。例えば、MLflow を scikit-learn で使用する場合、モデルのロギングは、実験の中から以下のコードを実行するのと同じくらい簡単です。

```
mlflow.sklearn.log_model(model=sk_model, artifact_path="sk_model_path")
```

これにより、サードパーティのライブラリ（XGBoost や spaCy など）やシンプルな Python 関数そのものからのモデルを、MLflow モデルとして保存することができます。Python 関数フレーバを使用して作成されたモデルは、同じエコシステム内に存在し、Inference API を通じて他の MLflow ツールと相互作用することができます。全てのユースケースに対応することは不可能ですが、Python 関数モデルフレーバーは可能な限り普遍的で柔軟性の高いものになるように設計されています。カスタム処理やロジック評価を可能にし、ETLアプリケーションに便利です。偶数の より多くの「公式」モデルのフレーバーがオンラインになっても、ジェネリックな Python 関数フレーバーは重要な「キャッチオール」としての役割を果たし、あらゆる種類の Python コードと MLflow の堅牢なトラッキングツールキットとの間の橋渡しをしてくれます。

Python の関数フレーバーを使ってモデルをログに記録するのは簡単なプロセスです。**どんなモデルや関数でもモデルとして保存することができますが、1つの条件があります。それは pandas のDataFrame を入力として受け取り、DataFrame または NumPy 配列を返さなければなりません。**この要件が満たされたら、関数を MLflow モデルとして保存するには、PythonModel を継承した Python クラスを定義し、ここで説明するようにカスタム関数で `.predict()` メソッドをオーバーライドする必要があります。

ある実行からログに記録されたモデルをロードする

いくつかの異なるストレッチファクターを用いてデータを実行したので、次のステップは当然のことながら、結果を検証し、ログに記録したメトリクスに応じて特に優れたモデルを探すことになります。**MLflow を使うとログモデルを作成し、それを使用して新しいデータでの予測を行うには、次の手順を使用します。**

1. モデルをロードしたい run のリンクをクリックしてください。
2. 「実行ID」をコピーします。
3. モデルが保存されているフォルダの名前をメモしておきます。私たちの場合は、単に"model“ という名前です。
4. 以下のようにモデルフォルダ名と run ID を入力してください。

```
import custom_flavor as mlflow_custom_flavor

loaded_model =
mlflow_custom_flavor.load_model(artifact_path='model',
run_id='e26961b25c4d4402a9a5a7a679fc8052')
```

モデルが意図したとおりに動作していることを示すために、モデルをロードし、変数 new_sales_units で作成した 2 つの新製品の DTW 距離を測定するために使用することができます。

```
# use the model to evaluate new products found in
'new_sales_units'  output = loaded_model.predict(new_sales_units)
print(output)
```

次のステップ

上述のように、私たちの MLflow モデルは新しい値や見たことのない値を容易に予測しています。また、Inference API に準拠しているため、任意のサービングプラットフォーム（Microsoft Azure MLやAmazon Sagemaker など）にモデルをデプロイしたり、ローカルのREST API エンドポイントとしてデプロイしたり、Spark-SQLで簡単に使用できるユーザー定義関数（UDF）を作成したりすることができます。

Databricks の統合データ分析プラットフォーム（Databricks Unified Data Analytics Platform）を活用して動的タイムワープによる販売動向を予測する方法をご紹介しました。Databricks の機械学習用ランタイムを使用した Notebook「Using Dynamic Time Warping and MLflow to Predict Sales Trends」（動的タイムワープと MLflow による売上動向の予測）をさっそくお試しください。

第 3 章

Facebook Prophet と Apache Spark™ による高精度で大規模な時系列予測

投稿者：
Bilal Obeidat
Bryan Smith
Brenner Heintz

2020 年 1 月 27 日

時系列予測 Notebooks を Databricks で試す

時系列予測・分析技術の進展により、小売業における需要予測の信頼性は向上しています。しかし、より正確なインベントリ管理を実現したい企業にとっては、予測の精度とタイミングが課題となっています。従来のソリューションにおいては拡張性や正確性の面で制約がありましたが、Apache Spark™ と Facebook Prophet の活用によってこれらの課題を克服する企業が増えてきています。

この記事では、 時系列予測の重要性、時系列データのサンプルの視覚化、Facebook Prophet を使ったシンプルな時系列予測モデルの構築について解説します。単一モデルの構築に慣れた後は、Prophet に Apache Spark™ のテクノロジーを組み合わせて、数百規模のモデルを一度にトレーニングする方法を紹介します。これにより、これまでほとんど達成されなかった細粒度で、SKU と店舗の組み合わせごとの正確な予測が作成できます。

ますます重要になる需要予測の正確性とリアルタイム性

商品やサービスの需要予測の正確度を改善するための、時系列分析の速度と精度の向上は小売業者の成功に不可欠です。店舗が管理する在庫が多すぎると、保管スペースの逼迫や商品の期限切れが生じ、インベントリの管理に多くの費用を投入せざるを得なくなります。さらに、そのような状況が放置されると、新商品の投入や消費パターンの変化にともなう好機を逃すことにもなりかねません。一方、店頭の商品が少なすぎれば、欠品が発生し、顧客の購買機会が失われることになります。予測エラーは小売業者にとって直接の損失となるばかりでなく、消費者が不満を抱くような状況が続けば、競合他社に顧客を奪われる恐れも生じます。

新たなニーズに対応する高精度な時系列予測手法とモデル

これまでの小売業界では、統合基幹業務（ERP）システムやサードパーティソリューションによる、シンプルな時系列モデルに基づいた需要予測システムが採用されてきました。しかし、技術的な進歩と業界における競争の激化を背景に、多くの企業がこれまでの線形モデルやアルゴリズムに変わる新しい手法を必要とするようになりました。

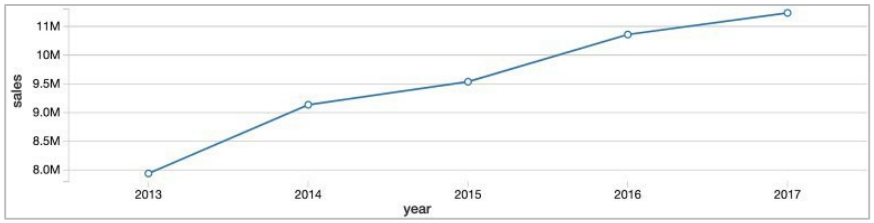
各企業のこれまでの時系列予測モデルに対して、データサイエンス分野で考案された Facebook Prophet などの新しい機械学習手法やモデルを、柔軟に適用できることが求められています。

企業内でこうした新しい需要予測ソリューションへの移行を進めるには、需要予測の複雑さに対処するだけでなく、効率的な分散処理環境を用意して、数十万から数百万にも達する機械学習モデルを遅滞なく生成できるようにする必要があります。Spark は、この分散モデルトレーニングを実現するソリューションであり、商品やサービス全体の需要だけでなく、各拠点における商品単位での需要予測も行えます。

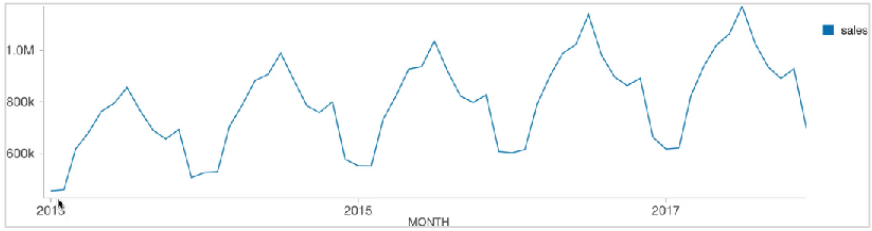
時系列データにおける季節変動需要の視覚化

ここでは、Prophet による個々の店舗と商品を対象とした高精度需要予測について見ていきます。データセットは Kaggle の一般公開データで、50 品目について10 店舗の日次売上を5年間記録したものを使用します。

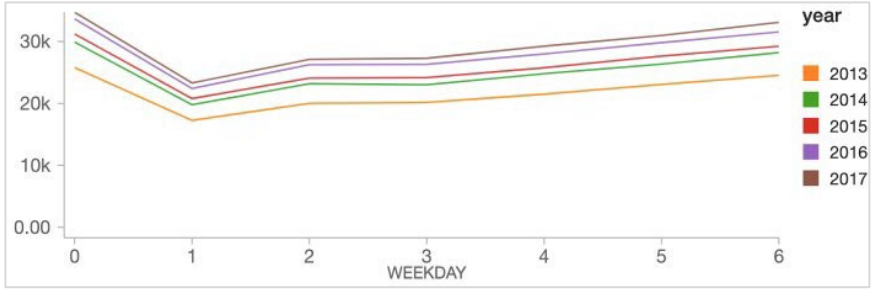
まず、全ての商品と店舗について、年間売上の全体像を確認しましょう。下のグラフからわかるように、商品の総売上高は大きな増減もなく毎年増加しています。



次に、同じデータを月単位で見てみましょう。そうすると、年単位の傾向とは違い、なだらかな増加ではなく、夏に上がり冬に下がるという季節的なパターン（季節変動）がはっきりと確認できます。Databricks のコラボレーション型 Notebook にあるデータ視覚化機能を使用すると、グラフにマウスポインタを合わせることで各月のデータ値が表示されます。



週単位では、売上のピークが日曜日（weekday 0）にあり、月曜日（weekday 1）に大きく落ち込んだ後、残りの平日に徐々に回復する傾向がみられます。



Prophet のシンプルな時系列予測のための解析モデル

上のグラフからわかるように、売上データからは年単位の増加傾向に加え、季節性のパターン（季節変動）と曜日によるパターンが見出せます。Facebook Prophet は、このような複数の傾向が重複したデータにも対応しています。

Prophet は scikit-learn API に準拠しており、誰でも簡単にsklearn を試すことができます。今回のサンプルであれば、2列の pandas DataFrame を API への入力として渡します。1列目に日付、2 列目に予測する値（ここでは売上）を指定します。データ形式に問題がなければ、容易にモデルが作成できます。

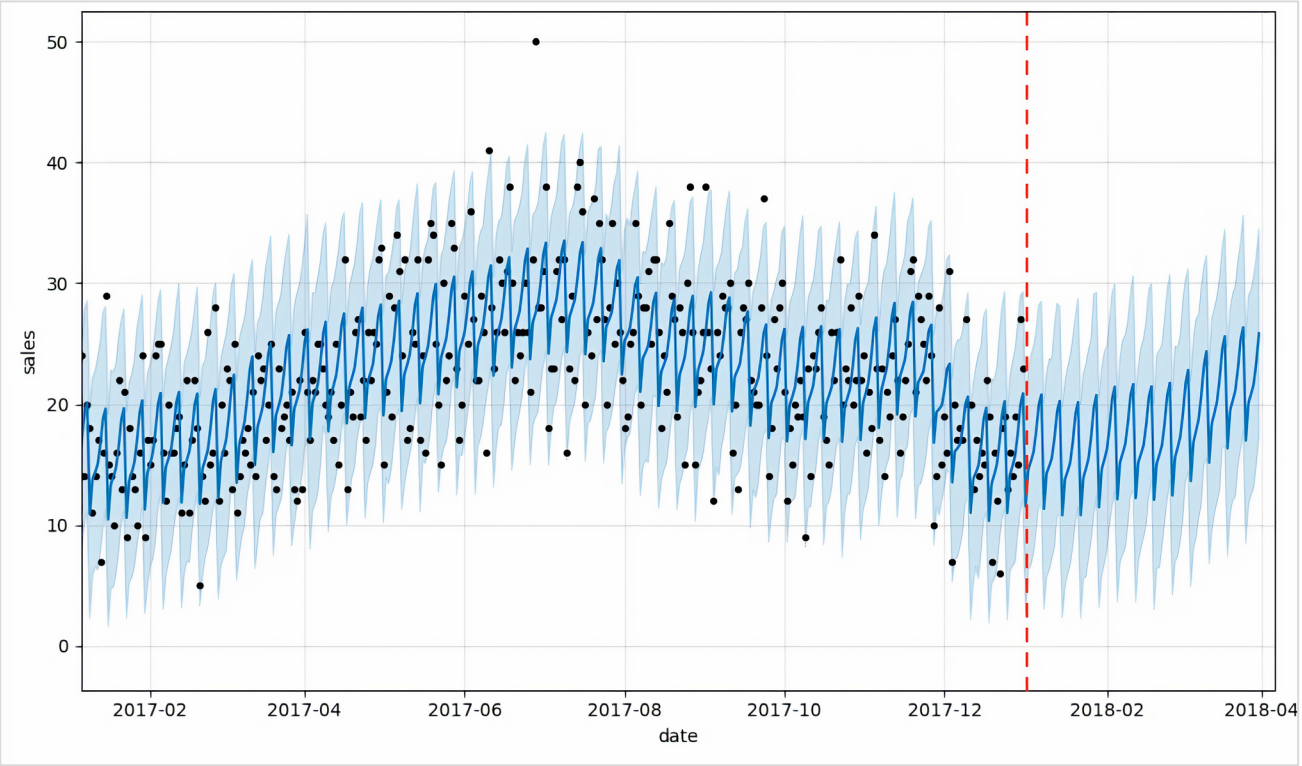
```
import pandas as pd
from fbprophet import Prophet

# instantiate the model and set parameters
model = Prophet(
    interval_width=0.95,
    growth='linear',
    daily_seasonality=False,
    weekly_seasonality=True,
    yearly_seasonality=True,
    seasonality_mode='multiplicative'
)

# fit the model to historical data
model.fit(history_pd)
```

これで、データに合致したモデルが作成できました。このモデルを使って90日分の予測を行ってみましょう。下のコードでは、Prophetの `make_future_dataframe` メソッドを使用して、データセットに日時の時系列データとその後の90日の予測を含めるよう指定します。

```
future_pd = model.make_future_dataframe(
    periods=90,
    freq='d',
    include_history=True
)

# predict over the dataset
forecast_pd = model.predict(future_pd)
```

これだけです。これで、 Prophet の .plot メソッドを使って、将来の予測値に加えて実データと推計値の両方を合わせて表示できるようになります。下のグラフのように、先に示した週単位と季節単位の需要パターンが予測結果にも反映されます。

```
predict_fig = model.plot(forecast_pd, xlabel='date',
ylabel='sales')  display(fig)
```



このグラフの見方については、少し説明が必要かもしれません。Bartosz Mikulski 氏がわかりやすく解説しています。端的にいえば、黒の点が実際の値を、濃い青色の部分が予測値を表しています。薄い青色の部分は 95% の不確実性を示す範囲です。

Prophet と Spark による数百の時系列予測モデルの並列トレーニング

ここまでで、単一の時系列予測解析モデルの作成方法を示しました。Apache Spark を活用することで、さらにその機能を高めることができます。具体的には、個々の商品と店舗を対象としたモデルを数百単位で生成することが可能です。単一モデルの場合、データセット全体を順次処理する形となり非常に時間がかかります。

数百単位のモデルを用意することで、たとえば、スーパーマーケットチェーンであれば、地域ごとの需要の違いに基づいて、各店舗で発注すべき生乳の量 などを正確に予測分析できるようになります。

Spark DataFrames を使用した時系列データの分散処理

膨大な数のモデルをトレーニングするには、通常、 Apache Spark などの分散データ処理エンジンが使用されます。 Spark クラスタを活用することで、モデルのサブセットのトレーニングがクラスタ内の複数のワーカーノードで並列処理され、時系列モデル全体のトレーニング時間を大きく削減できます。

クラスタのワーカーノードでトレーニングする場合も、相応のクラウドインフラストラクチャが必要で、その分のコストがかかります。しかし、クラウドリソースをオンデマンドで容易に利用できれば、必要なリソースを迅速にプロビジョニングできます。また、モデルのトレーニングやリソースの展開も短期間に行え、物理資産を長期間保持することなく、スケーラビリティを大きく向上させることができます。

なお、Spark の分散データ処理を実現するうえで重要な役割を果たしているのが Spark DataFrame です。 DataFrame にデータを読み込むことで、クラスタの各ワーカーにデータが分散されます。それぞれのワーカーでデータのサブセットを並列処理し、全体での実行時間を減らすことができるのは、この DataFrame の働きによるものです。

各ワーカーが処理を実行するには、必要なデータのサブセットにアクセスできる必要があります。DataFrame では、キーバリュー形式でデータをグループ化することで、個々のワーカーノードにデータを渡します。今回のケースでは、全ての時系列解析に使用するデータを店舗と商品の組み合わせで構成されるキーバリューとしてグループ化しています。

```
store_item_history
      .groupBy('store', 'item')
      # . . .
```

ここでは、groupBy コードを使用して、複数モデルトレーニングの効率的な並行処理について解説しています。ただし、実際に動作させるには、次の段落で説明するUDF を設定してデータに適用する必要があります。

pandas のユーザ定義関数（UDF）の活用

時系列データを店舗と商品で適切にグループ化できたら、グループごとに単一モデルのトレーニングを行います。トレーニングには、pandas のユーザ定義関数（UDF）を使用します。UDF は、DataFrame のデータグループごとにカスタム関数を適用できます。

各モデルのトレーニングだけでなく予測結果の生成にも使用できます。トレーニングや予測は DataFrame のグループごとに別々に実行されますが、各グループから出力される結果は新たに生成される単一の DataFrame にまとめられます。このような仕組みとなっているのは、予測が個々の店舗と商品・在庫について行われるのに対して、分析や管理の際には単一のデータセットとして結果を出力できるようにするためです。

下のPython コードは一部省略したものではありますが、UDF の作成はそれほど難しいものではありません。UDF では、返り値のデータスキーマと受け取るデータタイプを pandas_udf メソッドで規定します。その後に続けて、UDF で実行する処理内容について関数を定義します。

以下の関数では、モデルの作成と設定、受け取るデータとの適合について定義されています。また、モデルでの予測が実行され、関数からの出力としてデータが返されます。

```
@pandas_udf(result_schema, PandasUDFType.GROUPED_MAP)
def forecast_store_item(history_pd):

    # instantiate the model, configure the parameters
    model = Prophet(
        interval_width=0.95,
        growth='linear',
        daily_seasonality=False,
        weekly_seasonality=True,
        yearly_seasonality=True,
        seasonality_mode='multiplicative'
    )

    # fit the model
    model.fit(history_pd)

    # configure predictions
    future_pd = model.make_future_dataframe(
        periods=90,
        freq='d',
        include_history=True
    )

    # make predictions
    results_pd = model.predict(future_pd)

    # . . .

    # return predictions
    return results_pd
```

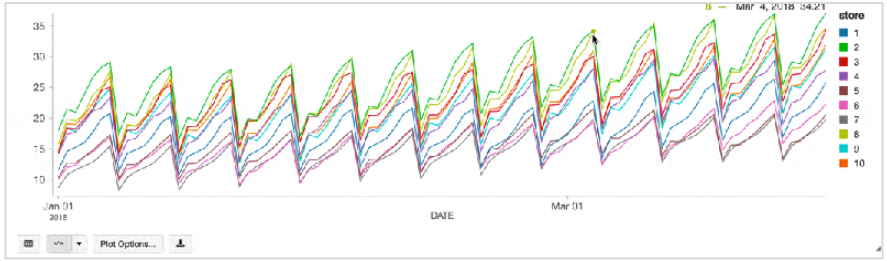
UDF の用意ができたら、データセットを店舗と商品で適切にグループ化するために、先に紹介した`groupBy` コマンドを使用します。あとは、UDF を DataFrame に `apply` すれば、モデルに合わせて UDF がデータグループごとの予測を行います。

各グループに対して関数が実行され、データセットが返されると、生成される予測データに結果が反映されます。このような仕組みにより、複数の時系列解析モデルから生成されるデータのトラッキングや、実運用への展開が可能となります。

```
from pyspark.sql.functions import current_date

results = (
    store_item_history
    .groupBy('store', 'item')
    .apply(forecast_store_item)
    .withColumn('training_date', current_date())
    )
```

次のステップ

このブログでは、店舗と商品の組み合わせを対象として時系列での需要予測のための解析モデルを構築しました。SQL クエリを使用すれば、個別の製品に絞った予測結果を表示させることもできます。下のグラフでは、商品 #1 について 10 店舗での需要予測結果が示されています。店舗ごとに需要の推移は異なっていますが、全ての店舗に一貫したパターンが見てとれ、想定どおりの結果となっています。



新しい売上データに基づいて新たに予測を行う場合も、容易に予測を生成して既存のテーブル構造に追加できます。事業環境の変化にあわせて想定を変えた分析も可能です。

さらに詳しい情報については、オンデマンドの Web セミナー「How Starbucks Forecasts Demand at Scale With Facebook Prophet and Azure Databricks」（スターバックスにおけるソリューション事例：Facebook Prophet と Azure Databricks を利用した大規模な需要予測）ご利用ください。

第 4 章

反復ニューラルネットワークを用いた多変量時系列予測の実行

Keras の実装による長短記憶（LSTM）を時系列予測に利用する

投稿者：Vedant Jain

2019年9月10日

この Notebook を試してみる

時系列予測は機械学習の重要な分野です。時系列データの性質上、正確なモデルの構築が困難な場合があります。機械学習のニューラルネットワークの進化により、従来の時系列予測アプローチでは範囲外あるいは困難であったさまざまな問題を解決できるようになりました。この記事では、Keras の実装である Long Short-Term Memory (LSTM) を時系列予測に、MLfLow をトラッキングモデルの実行に使用する方法を紹介します。

LSTMとは？

LSTM は反復ニューラル・ネットワーク（RNN）の一種であり、ネットワークが以前の多くのタイムステップから所定の時間に長期的な依存性を保持することを可能にします。RNN は、データの出力シーケンスが入力の一つとして機能するニューロンのための単純なフィードバックアプローチを使用して、その効果を得るように設計された。しかし、長期的な依存性は、消失勾配問題のためにネットワークを学習不能にする可能性があります。LSTM はこの問題を正確に解決するように設計されています。

正確な時系列予測は、古いデータと最近のデータの両方のビットの組み合わせに依存していることがあります。私たちは、何に注目すべきかを効率的に学習しなければならないが、学習すべきデータの歴史が長いかもしれないことを受け入れなければならない。LSTM は単純な DNN アーキテクチャと巧妙なメカニズムを組み合わせて、歴史のどの部分を「記憶」し、どの部分を「忘れる」べきかを長期間にわたって学習する。長いシーケンスにわたってデータのパターンを学習する LSTM の能力は、時系列予測に適しています。

LSTM のアーキテクチャの理論的な基礎については、こちら（第4章）を参照してください。

正しい問題と正しいデータセットを選択する

将来のファンド価格を基にしたポートフォリオの作成から電力供給網の需要予測まで、時系列の応用は無数にある。LSTM の価値を示すためには、まず正しい問題、さらに重要なのは正しいデータセットが必要です。例えば、家の中の湿度や温度を事前に予測して、スマートセンサーがエアコンを積極的に作動させるようにしたい場合や、将来の電力消費量を知りたい場合には、コスト削減を積極的に行うことができます。過去のセンサーと温度データの関係を知るには、過去のセンサーと温度データだけで十分ですが、LSTM が役立ちます。この目的のために、低エネルギービルでの家電製品のエネルギー使用に関する実験データを使用します。

実験

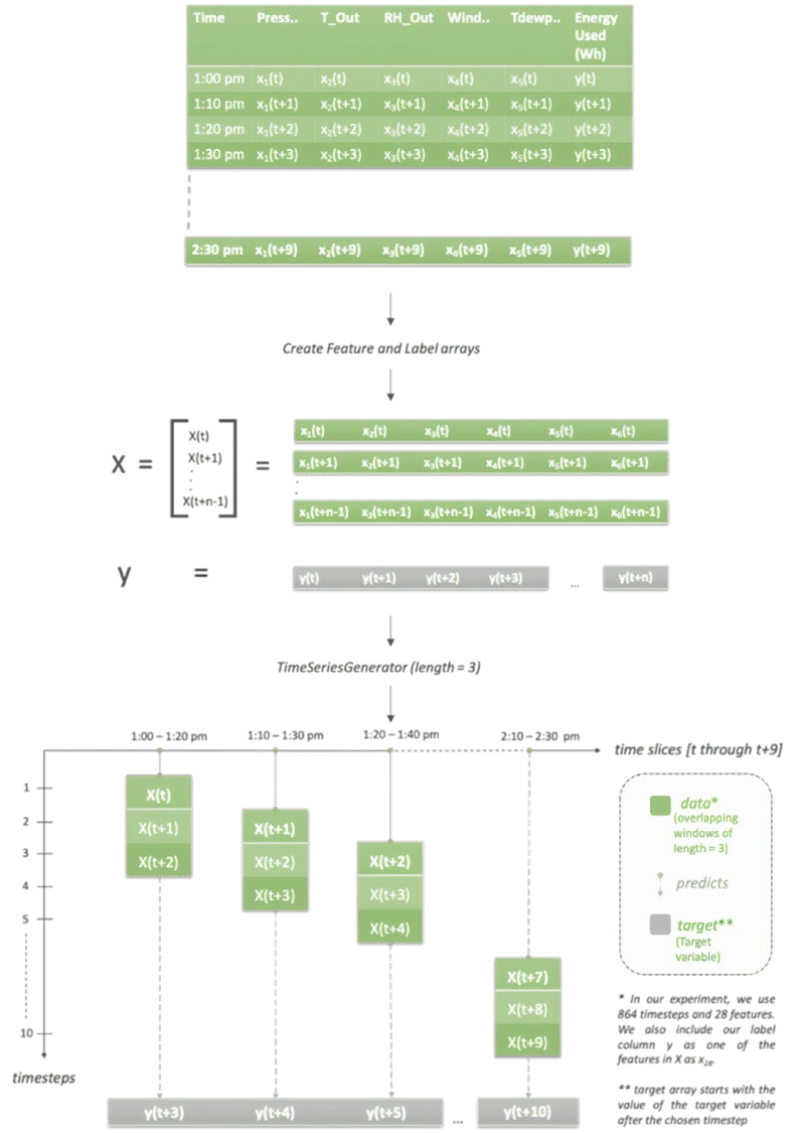
この実験で使用したデータセットは、家電のエネルギー使用量の回帰モデルの構築に最適です。家の温度と湿度を ZigBee 無線センサーネットワークでモニターしました。約 4.5 か月間、10 分間隔です。エネルギーデータは M-Bus のメーターで記録しました。最寄りの空港（ベルギーのキエフレス空港）の気象台からの気象情報は、公開データセットを利用しました。信頼性の高いプログノーシス（RP5.RU）からのデータを、日付と時間の欄を使用して実験データセットとマージしました。データセットは UCI 機械学習のリポジトリからダウンロードできます。

これを使用して次の日の家電の消費エネルギー予測モデルをトレーニングします。

データモデリング

ニューラルネットワークを訓練する前に、ネットワークが過去の一連の値から学習できるようにデータをモデル化する必要があります。特に、LSTM では、入力フィーチャの数によって、タイムステップごとのテストサンプルサイズの特定の 3D テンソル形式の入力データが期待されます。教師付き学習アプローチとして、LSTM は学習するために特徴量とラベルの両方を必要とする。時系列予測の文脈では、過去の値を特徴量として、未来の値をラベルとして提供することが重要であり、それによって LSTM は未来を予測する方法を学習できます。したがって、時系列データを 2D 配列 X に分解します。ここで、入力データは、バッチ内のタイムステップの所望の数で重複する時間差値で構成されています。入力フィーチャの全てのバッチについて予測しようとしているラベルまたは将来の値のみからなる 1D 配列 y を生成します。また、入力データには時間差値 y を含める必要があります。これにより、ネットワークはラベルの過去の値からも学習できます。

詳しい説明は右記の画像を参照してください。



我々のデータセットは 10 分のサンプルです。上の画像では、長さ = 3を選択していますが、これは、全てのシーケンス（10 分間隔）で 30 分のデータがあることを意味します。 この論理では、特徴量 'X' は値のテンソル [X(t), X(t+1), X(t+2)]、[X(t+2), X(t+3), X(t+4)]、[X(t+3), X(t+4), X(t+5)]... などでなければなりません。また、タイムステップの数や長さは 3 に等しいので、目標変数 y は [y(t+3), y(t+4), y(t+5)...y(t+10)] とします。また、グラフを見ると、各入力行について、将来の値を予測しているのは 1 つだけであることがわかります。しかし、より現実的なシナリオでは、以下の例のように、将来のさらに先の予測、すなわち、y(t+n+L) を選択することができます。

Keras API には TimeSeriesGenerator と呼ばれる組み込みのクラスがあり、重複する時間データのバッチを生成します。このクラスは、以下の場所で収集された一連のデータポイントを受け取ります。等間隔で、ストライド、履歴の長さなどの時系列パラメタとともに、トレーニング・検証用のバッチを作成します。

そこで、今回のユースケースでは、6 日間分の過去のデータから予測を学習し、将来のある時間、例えば 1 日後の値を予測したいとします。この場合、長さは 864 に等しく、これは 6 日間の 10 分のタイムステップの数です （24x6x6x6）。同様に、湿度、温度、気圧などの過去の値からも学習したいと考えています。このデータセットには合計 28 個の特徴があります。一時的なシーケンスを生成する際、ジェネレータは毎回 6 日分のデータからなるバッチを返すように設定されています。より現実的なシナリオにするために、1 日先の使用量を予測することを選択します（次の 10 分の時間間隔とは対照的に）。詳しくは、ノートの第 2 節「正規化してデータセットを準備する」を参照してください。

入力セットの形状は、(samples, timesteps, input_dim) [https://keras.io/layers/recurrent/] とします。各バッチについて、6 日分のデータを全て持っていることになり、864 行になります。バッチサイズは、グラデーション更新が行われる前のサンプル数を決定します。

```
# Create overlapping windows of lagged values for training and
testing datasets
timesteps = 864
train_generator = TimeseriesGenerator(trainX, trainY,
length=timesteps, sampling_rate=1, batch_size=timesteps)
test_generator = TimeseriesGenerator(testX, testY,
length=timesteps, sampling_rate=1, batch_size=timesteps)
```

チューニングパラメタの全リストはこちらを参照してください。

モデルトレーニング

LSTM は、 Backpropogation through time （BPTT） として知られている概念を用いて、ニューラルネットワークの長期依存性の問題に取り組むことができます。BPTT についての詳細はこちらをご覧ください。

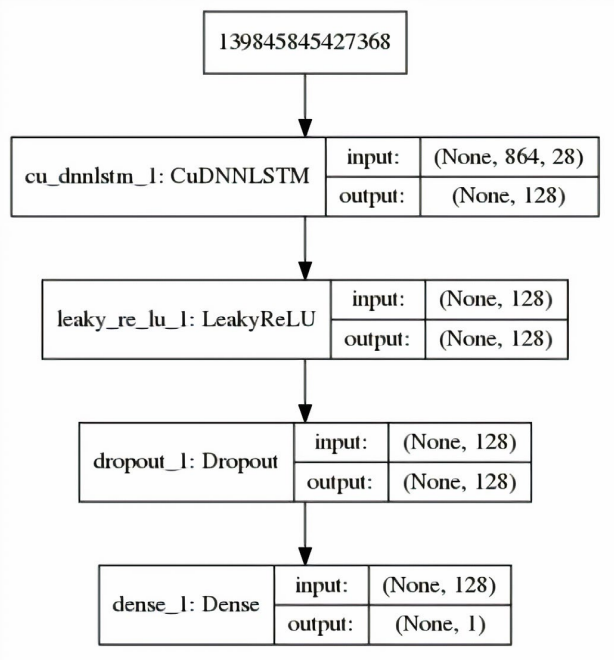
LSTM ネットワークを訓練する前に、ネットワークの品質を決定する Keras で提供されているいくつかの重要なパラメタを理解する必要があります。

1. epoch：データがニューラルネットワークに渡される回数
2. steps per epoch：訓練エポックが終了したとみなされる前のバッチ反復回数
3. activation：どの activation 関数を使用するかを記述したレイヤー
4. optimizer：Keras は組み込みのオプティマイザを提供します。

```
units = 128
num_epoch = 5000
learning_rate = 0.00144

with mlflow.start_run(experiment_id=3133492, nested=True):

    model = Sequential()
    model.add(CuDNNLSTM(units, input_shape=(train_X.shape[1],
    train_X.shape[2])))
    model.add(LeakyReLU(alpha=0.5))
    model.add(Dropout(0.1))
    model.add(Dense(1))

adam = Adam(lr=learning_rate)
# Stop training when a monitored quantity has stopped improving.
callback = [EarlyStopping(monitor="loss", min_delta = 0.00001,
patience = 50, mode = 'auto', restore_best_weights=True),
tensorboard]

# Using regression loss function 'Mean Standard Error' and
validation metric 'Mean Absolute Error'
model.compile(loss="mse", optimizer=adam, metrics=['mae'])
```

GPU の速度と性能を活かすために、LSTM の cuDNN 実装を使用しています。また、エポック数は任意に高いものを選びました。これは、データが最良のモデル適合を見つけるために、できるだけ多くの繰り返しを行うようにしたいからです。単位数については、28 個の特徴量がありますので、32 個から始めます。何度か繰り返してみると、128 を使うことでまともな結果が得られることがわかりました。

エポック数の選択については、アンダーフィッティングを避けるために高い数値を選択するのが良いアプローチです。オーバーフィットの問題を回避するために、Keras API に組み込まれたコールバック、特に EarlyStopping を使用できます。EarlyStopping は、モニターされている量の改善が止まったときにモデルの学習を停止します。この例では、監視量として損失を使用し、50 エポックで 1e-5 の減少がない場合は、モデルは学習を停止します。Keras には、よりスムーズなパラメタの分布を確保するためにネットワークにペナルティを与える正則化器（重み付き、ドロップアウト）が組み込まれているので、ネットワークは単位間で渡される文脈にあまり依存しません。ネットワークのレイヤーについては、以下の画像を参照してください。



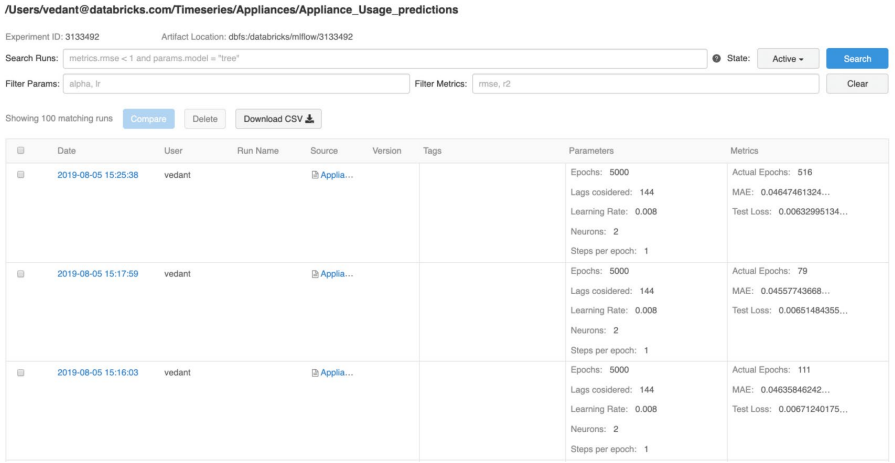
一方のレイヤーの出力を他方に送るためには、活性化関数が必要です。この場合、LeakyReLU を使用します。LeakyRelu は、その前身である Rectifier Linear Unit（整流器リニアユニット）または Relu（略してRelu）のより良いバリエーションです。

Kerasは、損失を減らし、エポックを繰り返し更新するためのさまざまなオプティマイザを提供しています。オプティマイザの完全なリストはこちらを参照してください。ここでは、確率的勾配降下のアダム版を選択します。

オプティマイザの重要なパラメタは `learning_rate` で、これはモデルの品質を大きく決定することができます。学習率についての詳細はこちらをご覧ください。0.001（デフォルト）、0.01、0.1などさまざまな値で実験した結果、0.00144 が学習速度と損失の最小化という点で最高のモデル性能を提供してくれることがわかりました。`LearningRateSchedular` コールバックを使用して、最適な値に学習率を調整することもできます。MLflow を使用して、複数のモデルの実行結果を追跡し、比較しました。

MLflow を用いたモデル評価とロギング

ご覧のように、ケラスのLSTMの実装では、かなりの数のハイパーパラメタを使用しています。最良のモデル適合を見つけるためには、さまざまなハイパーパラメタ、すなわち単位、エポックなどを実験する必要があります。また、過去のモデル実行を比較し、時間経過やデータの変化に伴うモデルの挙動を測定したいと思うでしょう。MLflow は、上記のようなことができる使いやすい UI を備えた素晴らしいツールです。ここでは、Keras やTensorFlow を使った開発、MLflow の実行のログ、時間経過に伴う実験の追跡など、MLflow を使っていかに簡単に開発できるかをご覧いただけます。



データサイエンティストは、MLflow を使用して、さまざまなモデルのメトリクスや、追加の可視化や成果物を追跡し、どのモデルを本番環境に導入すべきかの判断に役立てることができます。2 つ以上のモデルの実行を比較して、さまざまなハイパーパラメタの影響を理解し、最も最適なモデルを決定します。

データエンジニアは、本番の新しいデータにデプロイするためのトレーニングに使用したライブラリのバージョンとともに、選択したモデルを簡単に取得できるようになります。最終的なモデルは `python_function` フレーバーで永続化することができます。これをApache Spark UDF として使用することができ、一度 Spark クラスタにアップロードすると、将来のデータをスコア化するために使用されます。MLflow でサポートされているモデルフレーバーの全リストはこちら を参照 してください。

まとめ

- LSTM は、将来の発生を予測するために、過去の値から学習するために使用することができます。時系列のための LSTM は、古典的なアプローチで行われるようなある種の仮定をしないので、時系列問題のモデル化が容易になり、複数の入力間の非線形依存性を学習することが可能になります。
- LSTM ネットワークに投入する前にイベントのシーケンスを作成する場合、LSTMネットワークが過去のデータから学習できるように、入力からラベルを遅延させることが重要です。Keras の `TimeSeriesGenerator` クラスを使用すると、タイムラグしたデータセットをニューラルネットワークに投入する前に、さまざまなパラメタで時系列データセットを準備して変換できます。
- LSTM には、エポックやバッチサイズなどの一連の調整可能なハイパーパラメタがあり、これらは予測の品質を決定するために不可欠です。学習率は、モデルの重みの更新方法とモデルの学習速度を制御する重要なハイパーパラメタです。最良のモデル性能を得るには、学習率の最適値を決定することが重要です。学習率を最適化するために、`LearingRateSchedular` コールバックパラメタを使用することを検討してください。
- Keras は、あなたが解いている問題のタイプに応じて、異なるオプティマイザーを選択して使用することができます。一般的には、Adam が良い傾向にあります。MLflow UI を使用して、ユーザーはモデルの実行を並べて比較し、最適なモデルを選択することができます。
- 時系列では、LSTM ネットワークが正しいイベントのシーケンスからパターンを学習できるように、データの時間性を維持することが重要です。そのため、テストセットやバリデーションセットを作成する際や、モデルのフィッティングを行う際には、データをシャッフルしないことが重要です。
- 他の機械学習アプローチと同様に、LSTMはフィッティングの悪さに免疫がないので、Keras には `EarlyStopping`コールバックがあります。ある程度の直感と適切なコールバックパラメタがあれば、ハイパーパラメタのチューニングに力を入れなくても、まともなモデル性能を得ることができます。
- RNN、特に LSTM は、大量のデータが与えられたときに最適に動作します。そのため、利用可能なデータが少ない場合は、いくつかの隠れレイヤーを持つより小さなネットワークから始めることが好ましい。また、より小さなデータは、ユーザがより大きなバッチのデータをエポックごとに提供することを可能にし、より良い結果を得ることができます。

第 5 章

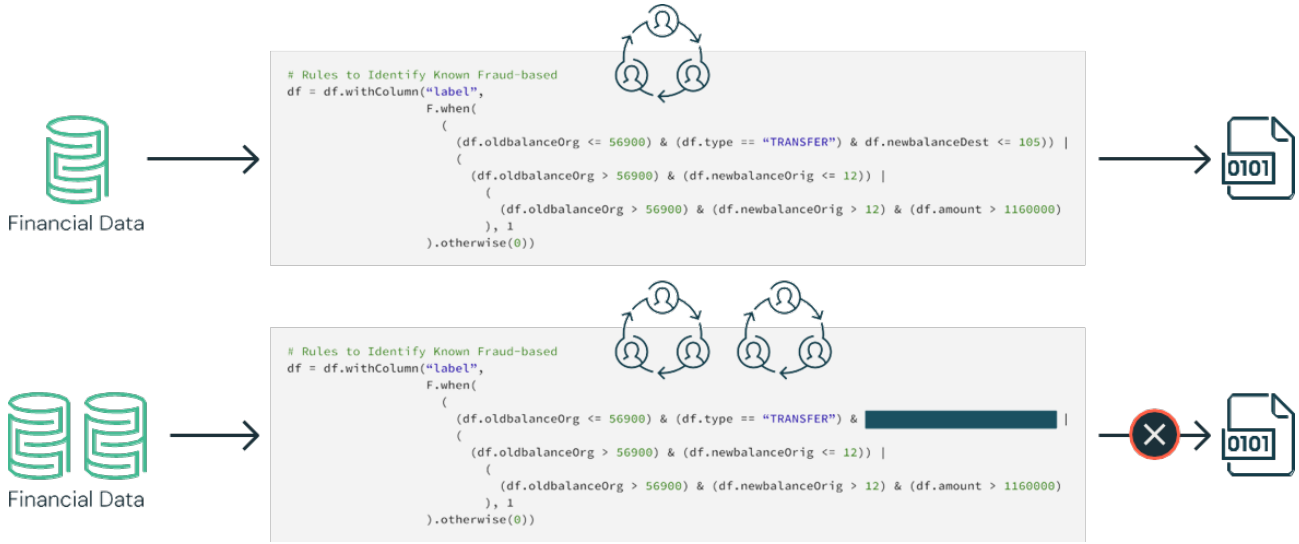
決定木と MLflow を用いた分析による金融詐欺検知の大規模展開

投稿者：
Elena Boiarskaia
Navin Albert
Denny Lee

2019 年 5 月 2 日

この Notebook を試してみる

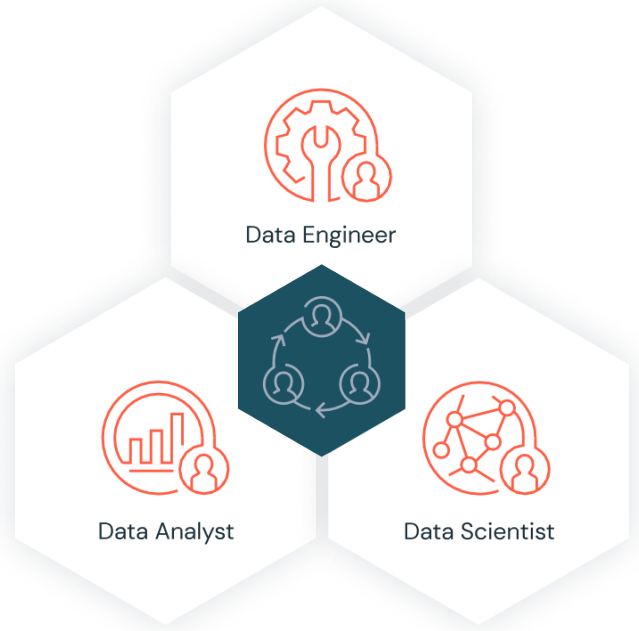
人工知能（AI）を活用した金融不正検知の大規模展開は、いかなるユースケースにおいても容易なことではありません。膨大の履歴データの取捨選択、絶えず進化する機械学習と深層学習技術の複雑さ、不正行為の実例の少なさなどが、不正パターンの検知を困難にしています。金融サービス業界においては、セキュリティに対する懸念の高まりや、不正がどのように特定されたかを説明することの重要性が加わり、複雑さがさらに増大しています。



検知パターンを作成するために、まずドメインエキスパートが不正者が行うであろう行為を想定して一連のルールを作成します。ワークフローに金融詐欺検知の専門家を含めて、特定の動作に関する要件をまとめる場合もあります。その後、データサイエンティストは利用可能なデータのサブサンプルを取得し、これらの要件と、場合によっては既存の金融不正事例を参照して、深層・機械学習アルゴリズムのセットを選択します。次に、データエンジニアが、この検知パターンを本番環境で使用するために、結果として生じるモデルをしきい値（条件分岐の境目となる値）を持つルールセットに変換します。これには通常、SQL を使用します。

このアプローチにより、金融機関は一般データ保護規則（GDPR）に準拠した不正取引を特定する明確な特性を提示することができます。しかし、このアプローチにもいくつか課題があります。まず、ハードコードされたルールセットを使用した不正検知システムの実装は非常に脆弱です。不正パターンに変更を加えると、更新に非常に時間がかかってしまうため、現在の市場で起こっている不正行為の変化に追いついて、対応することが困難です。

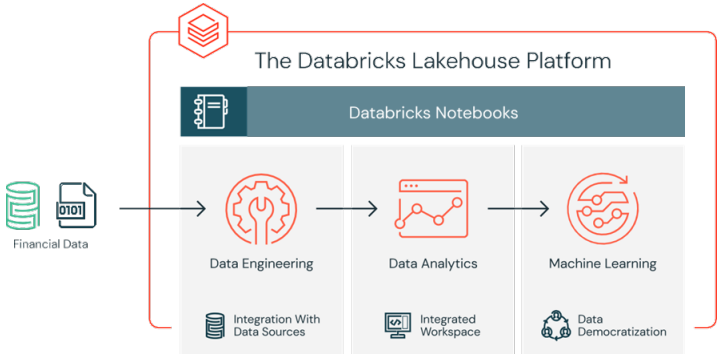
さらに、上記のワークフローのシステムは各々がサイロ化（孤立化）されることが多く、ドメインエキスパート、データサイエンティスト、データエンジニアが全て区分化されています。データエンジニアの重要な役割は、膨大な量のデータを管理し、ドメインエキスパートやデータサイエンティストの作業を本番レベルのコードに変換することです。共通のプラットフォームがない場合、ドメインエキスパートとデータサイエンティストは、分析のために単一のマシンに適合するサンプリングされたダウンデータに頼るしかありません。このようなサイロ化された環境では、お互いのコミュニケーションが難しくなり、コラボレーションの欠如につながります。



このブログでは、Databricks を活用して、不正検知のキープレイヤーであるドメインエキスパート、データサイエンティスト、データエンジニアを統合し、ルールベースの検知ユースケースを Databricks プラットフォーム上の機械学習ユースケースに変換する方法を紹介します。具体的には、大規模なデータセットからモジュラー機能を構築するフレームワークを活用して、機械学習で不正検知データパイプラインを作成し、リアルタイムでデータを視覚化・分析する方法、また、決定木（ディシジョン・ツリー）や Apache Spark MLlib を採用して不正を検知する方法・メリット・特徴を解説します。その後、MLflow を使用してモデルの反復処理と改良を行い、精度を向上させます。

機械学習によるソリューション

金融業界では、機械学習モデルの採用に比較的消極的な見方があります。それは、特定された不正なケースを正当化できない「ブラックボックス」ソリューションと考えられているためです。GDPR 要件と金融規制にデータサイエンスの能力を活用することは一見不可能です。しかし、いくつかの成功したユースケースでは、機械学習を適用して大規模に不正を検知することで、上述した多くの問題を解決できることが示されています。



実際に確認された不正行為の事例が少ないため、金融不正を検知する「教師あり機械学習モデル」をトレーニングすることは困難です。しかし、特定の不正行為を識別する既知のルールセットの存在は、合成ラベルセットと特徴量の初期セットを作成するのに役立ちます。また、この分野のドメインエキスパートが開発した検知パターンの出力は、適切な承認プロセスを経て本番環境で運用されているはずです。これは、予想される不正フラグを生成するので、機械学習モデルをトレーニングするための出発点として使用することができます。これにより、次の 3 つの懸念が同時に軽減されます。

1. トレーニングラベルの欠如
2. 使用する特徴量の決定
3. モデルに適したベンチマークの有無

ルールベースの不正フラグを認識する機械学習モデルをトレーニングすると、混同行列を介して予想される出力と直接比較できます。結果がルールベースの検知パターンと適合していれば、このアプローチによって、不正防止に機械学習を採用することへの信頼を得ることができます。また、このモデルの出力の解釈は非常に簡単なため、元の検知パターンと比較した場合の予想される検知漏れと誤検知の基本的な議論に役立つかもしれません。

機械学習モデルの解釈が難しいという懸念は、初期の機械学習モデルとして決定木モデルを使用することで解決できるかもしれません。決定木モデルは一連のルールに従ってトレーニングされているため、決定木は他の機械学習モデルよりも優れています。決定木モデルを使用するさらなるメリット・特徴としては、モデルの最大限の透過性です。このモデルは基本的に不正の意思決定プロセスを示しますが、人間の介入や、ルールやしきい値のハードコーディングを不要にします。もちろん、モデルの将来の反復過程では、最大の精度にするために異なるアルゴリズムを利用する可能性も理解する必要があります。アルゴリズムに組み込まれた特徴を理解しなければ、透過的なモデルは実現できないからです。解釈可能な特徴を持つことで、解釈可能で制御可能なモデルの結果が得られます。

機械学習アプローチを採用する最大のメリットは、初期のモデリング作業の後は、将来の反復がモジュール化され、ラベル、特徴量、あるいはモデルタイプのセットが容易にシームレスに更新され、運用までの時間が短縮される点です。この作業は、Databricks の統合分析プラットフォームの採用でさらに効率化します。このプラットフォームでは、ドメインエキスパート、データサイエンティスト、データエンジニアがデータセットを共有し、Notebook 環境で直接共同作業ができます。

データの取り込みと探索

この例では、合成データセットを使用します。データセットを自分でロードするには、Kaggleからローカルマシーンにダウンロードし、Azure や AWS 経由でデータをインポートしてください。

PaySimデータは、アフリカのある国で実施されたモバイルマネーサービスの1ヶ月の財務ログから抽出した、実際の取引のサンプルに基づくモバイルマネー取引をシミュレーションしたものです。次の表は、データセットが提供する情報を示しています。

Column Name	Description
step	maps a unit of time in the real world. In this case 1 step is 1 hour of time. Total steps 744 (30 days simulation).
type	CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.
amount	amount of the transaction in local currency.
nameOrig	customer who started the transaction
oldbalanceOrg	initial balance before the transaction
newbalanceOrig	new balance after the transaction
nameDest	customer who is the recipient of the transaction
oldbalanceDest	initial balance recipient before the transaction. Note that there is not information for customers that start with M (Merchants).
newbalanceDest	new balance recipient after the transaction. Note that there is not information for customers that start with M (Merchants).

データの探索

DataFrames を作成します。Databricks ファイルシステム（ DBFS ）にデータをアップロードしたので、Spark SQL を使って DataFrames を迅速かつ容易に作成できます。

```
# Create df DataFrame which contains our simulated financial fraud detection dataset
df = spark.sql("select step, type, amount, nameOrig, oldbalanceOrg,  newbalanceOrig, nameDest, oldbalanceDest, newbalanceDest from sim_fin_fraud_detection")
```

DataFrame を作成したので、スキーマと最初の 1000 行を見てデータを確認します。

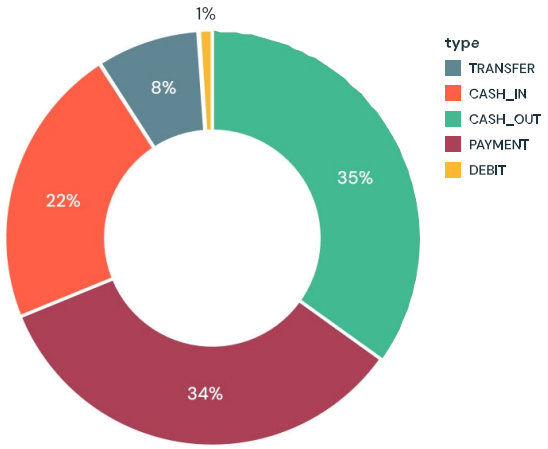
```
# Review the schema of your data
df.printSchema()
root
|-- step: integer (nullable = true)
|-- type: string (nullable = true)
|-- amount: double (nullable = true)
|-- nameOrig: string (nullable = true)
|-- oldbalanceOrg: double (nullable = true)
|-- newbalanceOrig: double (nullable = true)
|-- nameDest: string (nullable = true)
|-- oldbalanceDest: double (nullable = true)
|-- newbalanceDest: double (nullable = true)
```

step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest
1	PAYMENT	9839.64	C1231006815	170136	160296.36	M1979787155	0
1	PAYMENT	1864.28	C1666544295	21249	19384.72	M2044282225	0
1	TRANSFER	181	C1305486145	181	0	C553264065	0
1	CASH_OUT	181	C840083671	181	0	C38997010	21182
1	PAYMENT	11668.14	C2048537720	41554	29885.86	M1230701703	0
1	PAYMENT	7817.71	C90045638	53860	46042.29	M573487274	0
1	PAYMENT	7107.77	C154988899	183195	176087.23	M408069119	0
1	PAYMENT	7861.64	C1912850431	176087.23	168225.59	M633326333	0

取引の種類

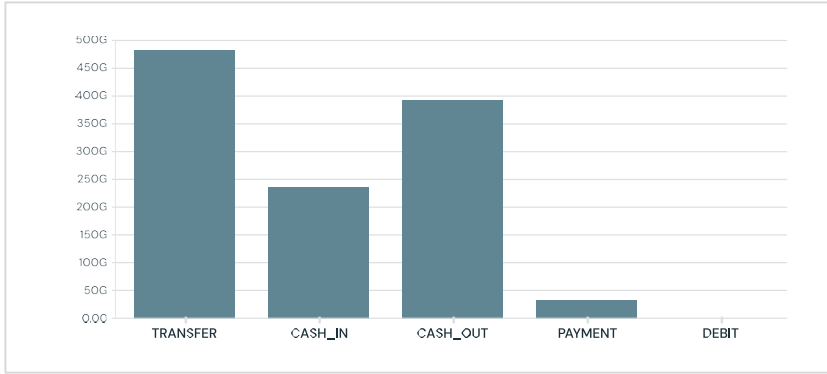
データをわかりやすいように視覚化して、データが捉えた取引の種類と、全体の取引件数に対する割合をみてみましょう。

```
%sql
-- Organize by Type
select type, count(1) from financials group by type
```



また、ここで取り上げているデータの金額がどのくらいかを理解するために、取引の種類と、現金の動きに基づいたデータ（総取引金額）を視覚化します。

```
%sql
select type, sum(amount) from financials group by type
```



ルールベースのモデル

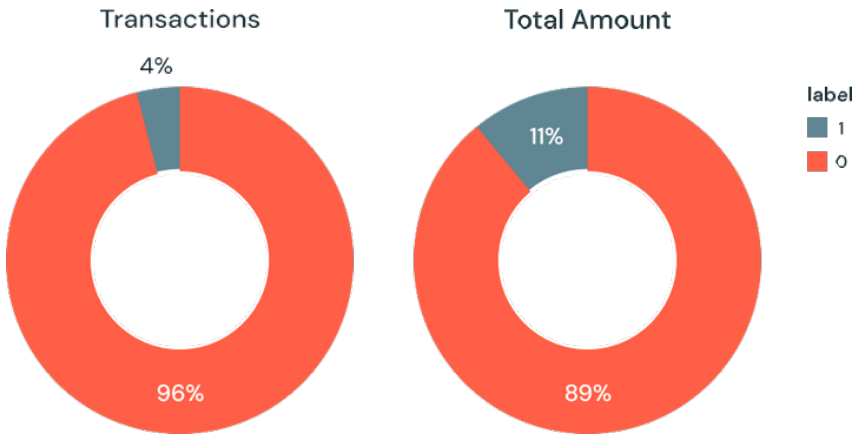
モデルのトレーニングをするために、既存の不正事例の大規模なデータセットを使用することはほとんどありません。多くの実用的なアプリケーションでは、ドメインエキスパートによって確立された一連のルールで不正な検知パターンを識別します。ここでは、こうしたルールに基づくラベルと呼ばれる列を作成します。

```
# Rules to Identify Known Fraud-based
df = df.withColumn("label",
                F.when(
                  (
                    (df.oldbalanceOrg <= 56900) & (df.type ==
"TRANSFER") & (df.newbalanceDest <= 105)) | ( (df.oldbalanceOrg > 56900)
& (df.newbalanceOrig <= 12)) | ( (df.oldbalanceOrg > 56900) &
(df.newbalanceOrig > 12) & (df.amount > 1160000)
                     ), 1
                ).otherwise(0))
```

ルールによってフラグを立てたデータの視覚化

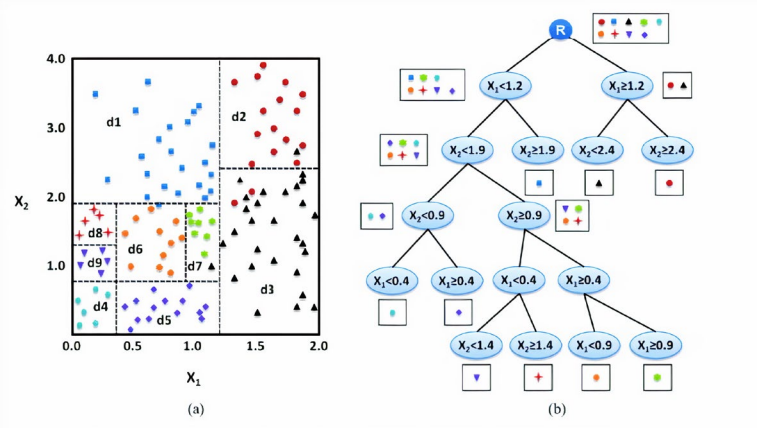
これらのルールは、多くの場合、かなりの数の不正なケースにフラグを立てます。フラグが立てられた取引の数を視覚化してみると、ルールは、取引の約 4% 、ドル総額の 11% を不正としてフラグを立てたことがわかります。

```
%sql
select label, count(1) as 'Transactions', sun(amount) as 'Total
Amount' from financials_labeled group by label
```



適切な機械学習モデルの選択

多くの場合、ブラックボックス的なアプローチでは不正検知ができません。まず、ドメインエキスパートは、取引が不正であると特定された理由を理解する必要があります。次に、措置が取られる場合は、証拠を法廷で提示する必要があります。このユースケースの対応には、解釈しやすいモデルの決定木（decision tree）が最適です。



トレーニングセットの作成

機械学習モデルを構築して検証するには、`.randomSplit` を使って 80/20 分割を行います。これにより、ランダムに選択されたデータの 80% がトレーニング用に、残りの 20% が結果の検証用に確保されます。

```
# Split our dataset between training and test datasets
(train, test) = df.randomSplit([0.8, 0.2], seed=12345)
```

機械学習モデルのパイプラインの作成

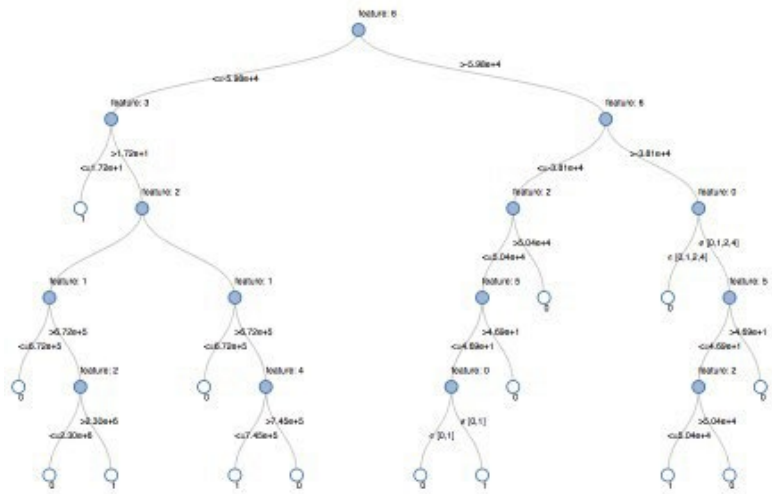
モデルのデータを準備するために、まず、.StringIndexer を使用してカテゴリ変数を数値に変換します。次に、モデルで使用する機能を全て組み立てる必要があります。決定木モデルに加えて、これらの特徴量の準備手順を含むパイプラインを作成し、さまざまなデータセットでこれらの手順を繰り返せるようにします。最初にパイプラインをトレーニングデータに適合させ、後の段階でそれを使ってテストデータを変換するので注意してください。

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import DecisionTreeClassifier

# Encodes a string column of labels to a column of label indices
indexer = StringIndexer(inputCol = “type”, outputCol = “typeIndexed”)

# VectorAssembler is a transformer that combines a given list of columns
into a single vector column
va = VectorAssembler(inputCols = [“typeIndexed”, “amount”, “oldbalanceOrg”,
“newbalanceOrig”, “oldbalanceDest”, “newbalanceDest”, “orgDiff”,
“destDiff”], outputCol = “features”)

# Using the DecisionTree classifier model
dt = DecisionTreeClassifier(labelCol = “label”, featuresCol = “features”,
seed = 54321, maxDepth = 5)

# Create our pipeline stages
pipeline = Pipeline(stages=[indexer, va, dt])

# View the Decision Tree model (prior to CrossValidator)
dt_model = pipeline.fit(train)
```

モデルの可視化

パイプラインの最終ステージである決定木モデル `display()` を呼び出すと、各ノードで選択した決定を含む初期の適合モデルを表示します。これにより、アルゴリズムが結果の予測にどのように到達したのかを理解しやすくなります。

```
display(dt_model.stages[-1])
```



決定木モデルの視覚的表現

モデルのチューニング

最適ツリーモデルを確保するために、複数のパラメタのバリエーションを用いてモデルをクロス検証します。データが 96% の負のケースと 4% の正のケースで構成されていることを考えると、不均衡な分布を説明するために、適合率・再現率（PR）の評価指標を使用します。

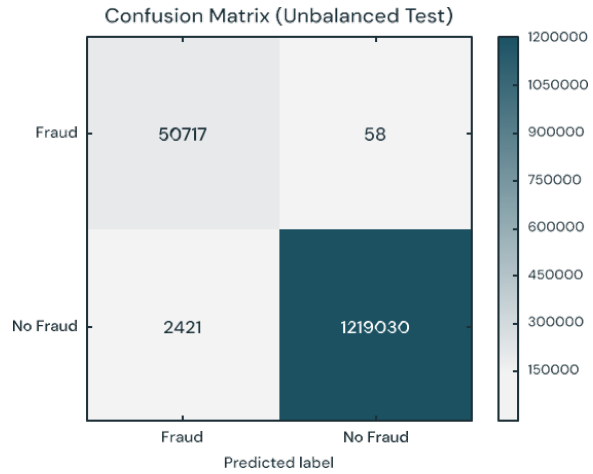
```
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
# Build the grid of different parameters
paramGrid = ParamGridBuilder() \
.addGrid(dt.maxDepth, [5, 10, 15]) \
.addGrid(dt.maxBins, [10, 20, 30]) \
.build()

# Build out the cross validation
crossval = CrossValidator(estimator = dt,
                          estimatorParamMaps = paramGrid,
                          evaluator = evaluatorPR,
                          numFolds = 3)

# Build the CV pipeline
pipelineCV = Pipeline(stages=[indexer, va, crossval])

# Train the model using the pipeline, parameter grid, and
preceding BinaryClassificationEvaluator
cvModel_u = pipelineCV.fit(train)
```

モデルの性能

モデルを評価するには、トレーニングセットとテストセットの適合率・再現率（PR）と ROC 曲線下の面積（AUC）メトリクスを比較します。PR と AUC は共にかなり高いようです。

```
# Build the best model (training and test datasets)
train_pred = cvModel_u.transform(train)
test_pred = cvModel_u.transform(test)

# Evaluate the model on training datasets
pr_train = evaluatorPR.evaluate(train_pred)
auc_train = evaluatorAUC.evaluate(train_pred)

# Evaluate the model on test datasets
pr_test = evaluatorPR.evaluate(test_pred)
auc_test = evaluatorAUC.evaluate(test_pred)

# Print out the PR and AUC values
print("PR train:", pr_train)
print("AUC train:", auc_train)
print("PR test:", pr_test)
print("AUC test:", auc_test)

---
# Output:
# PR train: 0.9537894984523128
# AUC train: 0.998647996459481
# PR test: 0.9539170535377599
# AUC test: 0.9984378183482442
```

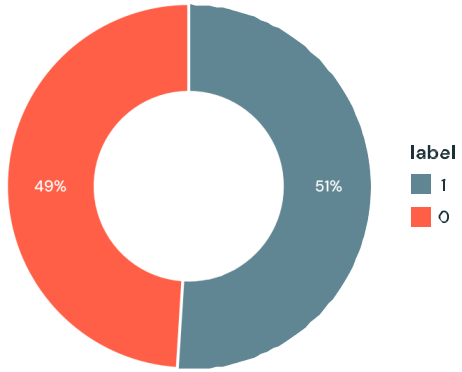
モデルが結果を誤って分類した過程を確認するために、matplotlib と pandas を使用して、混同行列を視覚化しましょう。



クラスのバランスをとる

このモデルは、識別された元のルールよりも 2,421 件多い事例を識別していることがわかります。より多くの潜在的な不正事例を検知することは良いことかもしれないので、この結果に関してそれほど心配する必要はありません。しかし、アルゴリズムによって検知されなかったが、最初に識別されていた事例が 58 件あります。ここで私たちが試みているのは、アンダーサンプリングを使用してクラスのバランスを取り、予測をさらに改善することです。 つまり、全ての不正事例は保持し、不正でない事例をダウンサンプルして数を一致させ、バランスの取れたデータセットの取得です。新しいデータセットを視覚化すると、Yes と No の事例は半々であることがわかりました。

```
# Reset the DataFrames for no fraud (`dfn`) and fraud (`dfy`)
dfn = train.filter(train.label == 0)
dfy = train.filter(train.label == 1)

# Calculate summary metrics
N = train.count()
y = dfy.count()
p = y/N

# Create a more balanced training dataset
train_b = dfn.sample(False, p, seed = 92285).union(dfy)

# Print out metrics
print("Total count: %s, Fraud cases count: %s, Proportion of fraud cases:
%s" % (N, y, p))
print("Balanced training dataset count: %s" % train_b.count())

---
# Output:
# Total count: 5090394, Fraud cases count: 204865, Proportion of fraud
cases: 0.040245411258932016
# Balanced training dataset count: 401898
---

# Display our more balanced training dataset
display(train_b.groupBy("label").count())
```



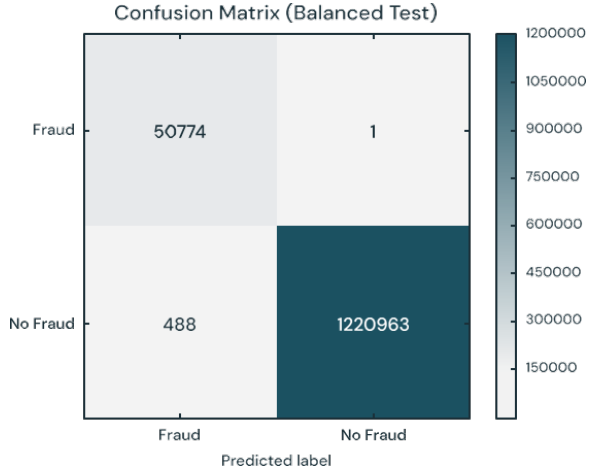
パイプラインの更新

機械学習パイプラインを更新し、新しいクロス検証を作成しましょう。機械学習パイプラインを使用しているため、新しいデータセットで更新するだけで、同じパイプラインステップをすぐに繰り返すことができます。

```
# Re-run the same ML pipeline (including parameters grid)
crossval_b = CrossValidator(estimator = dt,
estimatorParamMaps = paramGrid,
evaluator = evaluatorAUC,
numFolds = 3)
pipelineCV_b = Pipeline(stages=[indexer, va, crossval_b])

# Train the model using the pipeline, parameter grid, and
BinaryClassificationEvaluator using the `train_b` dataset
cvModel_b = pipelineCV_b.fit(train_b)

# Build the best model (balanced training and full test datasets)
train_pred_b = cvModel_b.transform(train_b)
test_pred_b = cvModel_b.transform(test)

# Evaluate the model on the balanced training datasets
pr_train_b = evaluatorPR.evaluate(train_pred_b)
auc_train_b = evaluatorAUC.evaluate(train_pred_b)

# Evaluate the model on full test datasets
pr_test_b = evaluatorPR.evaluate(test_pred_b)
auc_test_b = evaluatorAUC.evaluate(test_pred_b)

# Print out the PR and AUC values
print("PR train:", pr_train_b)
print("AUC train:", auc_train_b)
print("PR test:", pr_test_b)
print("AUC test:", auc_test_b)

---
# Output:
# PR train: 0.999629161563572
# AUC train: 0.9998071389056655
# PR test: 0.9904709171789063
# AUC test: 0.9997903902204509
```

結果の確認

それでは、新しい混同行列の結果をみてみましょう。モデルが不正事例を誤認したのは1件のみでした。クラスのバランスを取ることで、モデルが改善されたことがわかります。



モデルのフィードバックと MLflow の利用

実運用モデルを選択したら、モデルがまだ対象となる行動を識別していることを確実にするために、継続的にフィードバックを収集します。ルールベースのラベルから始めているため、人のフィードバックに基づいて検証された真のラベルを将来のモデルに提供したいと考えます。この段階は、機械学習プロセスの信頼性を維持するために非常に重要です。アナリストは全てのケースをレビューできないため、モデルの出力の検証用に慎重に選んだケースを提示したいと考えます。例えば、モデルの確実性が低い予測は、アナリストのレビュー候補となります。フィードバックが追加されることで、モデルは変化する状況にあわせて改善され、進化し続けます。

MLflow は、異なるモデルのバージョンを学習する際に、サイクル全体を通して私たちを助けてくれます。異なるモデル構成やパラメタの結果を比較し、実験を追跡できます。例えば、ここでは MLflow UI を使用して、均衡・不均衡のデータセット上でトレーニングされたモデルの PR と AUC を比較しました。データサイエンティストは、MLflow を使用して、さまざまなモデルメトリックスや追加の視覚化やアーティファクトの経過を追跡し、本番運用のモデルの決定に役立てることができます。そして、データエンジニアは、.jar ファイルとしてトレーニングに使われるライブラリーのバージョンと併せて、選択したモデルを容易に取り込み、本番環境にデプロイできます。このように、モデルの結果をレビューするドメインエキスパート、モデルを更新するデータサイエンティスト、本番でモデルを展開するデータエンジニアの間の共同作業は、この反復プロセスを通じて強化されます。

結論

ルールベースの不正検知ラベルを使用し、Databricks with MLflow によって ML モデルに変換する方法の例をレビューしました。この方法により、スケーラブルでモジュール化されたソリューションを構築でき、変化する不正のパターンに対応できます。不正を特定するための機械学習モデルを構築することで、モデルを進化させ、新たな不正の可能性のあるパターンを特定するためのフィードバックループを作成できます。特に決定木モデルは、解釈のしやすさと確度のため、機械学習を不正検知プログラムに導入する際の出発点として最適であることがわかりました。

この取り組みに Databricks のプラットフォームを使用する大きなメリットは、データサイエンティスト、エンジニア、ビジネスユーザーがシームレスに連携して作業できることです。データの準備、モデルの構築、結果の共有、モデルの本番への投入を同じプラットフォーム上で行えるようになり、かつてないコラボレーションが可能になりました。チームのサイロ化が解消して信頼関係が構築され、効果的でダイナミックな不正検知プログラムにつながります。

無料トライアルで Notebook を試して、自分のモデルを作ってみませんか？

第 6 章

機械学習を活用した デジタル病理画像解析の自動化

イメージングの技術的進歩と新しい効率的な計算ツールの利用可能性により、デジタル病理学は研究と診断の両方の環境で中心的な役割を果たしています。Whole Slide Imaging（WSI）はこの変革の中心にあり、病理学のスライドを高解像度の画像に迅速にデジタル化することを可能にしています。スライドを即座に共有して分析できるようにすることで、WSIはすでに再現性を向上させ、教育や遠隔病理学サービスの強化を可能にしています。

今日では、高解像度でのスライド全体のデジタル化が、1分以内で安価にできるようになりました。その結果、デジタル化されたスライドの大規模なカタログを取得する医療機関やライフサイエンス機関が増えています。これらの大規模なデータセットは、機械学習を用いて自動診断を構築するために使用することができ、スライド（またはそのセグメント）を特定の表現型を発現しているものとして分類したり、スライドから定量的なバイオマーカーを直接抽出したりすることができます。機械学習と深層学習の力を使えば、数千枚のデジタルスライドを数分で解釈することができる。これは、病理部門、臨床医、研究者ががんや感染症の診断や治療を行う際の効率性と有効性を向上させるための大きな機会を提供します。

投稿者：
Amir Kermany
Frank Austin Nothaft

2020年1月31日

この Notebook を試してみる

デジタルパソロジーワークフローの普及を妨げる 3 つの共通課題

多くのヘルスケアおよびライフサイエンス企業は、人工知能をスライド画像全体に適用することの潜在的な影響を認識していますが、自動スライド解析パイプラインの実装は依然として複雑です。運用中の WSI パイプラインは、低コストでデジタイザースライドの高スループットを日常的に処理できなければなりません。組織がデータサイエンスをサポートした自動デジタル病理学ワークフローの実装を妨げる 3 つの共通課題が見えてきました。

1. **データの取り込みとエンジニアリングパイプラインが低速で高コスト：**
   WSI 画像は通常、非常に大きく（通常、1 スライドあたり 0.5〜2 GB）、大規模な画像前処理が必要になる場合があります。
2. **深層学習をテラバイト規模の画像にスケールさせる場合の問題：**
   数百の WSI を使用した中程度のサイズのデータセットで深層学習モデルを学習するには、1 つのノードで数日から数週間かかる場合があります。このような待ち時間は、大規模なデータセットでの迅速な実験を妨げます。複数のノードにまたがる深層学習のワークロードを並列化することでレイテンシを減らすことができますが、これは高度な技術であり、典型的な生物学的データサイエンティストの手の届かないところにあります。
3. **WSI のワークフローの再現性の確保：**
   患者データに基づいた新たな洞察を得るためには、結果を再現できることが非常に重要である。現在のソリューションはほとんどがアドホックであり、機械学習モデルのトレーニング中に使用された実験やデータのバージョンを効率的に追跡する方法を提供していません。

このブログでは、Databricks 統合データ分析プラットフォームを使用して、これらの課題に対処し、WSI 画像データ上にエンドツーエンドでスケーラブルな深層学習ワークフローを展開する方法について説明します。スライド上の転移の領域を特定する画像セグメンテーションモデルをトレーニングするワークフローに焦点を当てます。この例では、Apache Spark を使用して画像のコレクション全体でデータ準備を並列化し、pandas UDF を使用して多くのノード間で事前学習されたモデルに基づいて特徴を抽出し（転移学習）、MLflow を使用してモデルトレーニングを再現性よく追跡します。

WSI 上でのエンドツーエンドの機械学習

Databricks のプラットフォームを使用して WSI データ処理パイプラインを高速化する方法を実演するために、Camelyon16 グランドチャレンジデータセットを使用します。これは、我々のワークフローを実証するために、乳がん組織からの TIFF 形式の 400 枚のホールスライド画像からなるオープンアクセスのデータセットです。Camelyon16 データセットのサブセットは、/databricks-datasets/med-images/camelyon16/ (AWS | Azure) の下の Databricks から直接アクセスできます。

スライド内のがん転移を含む領域を識別する画像分類器を訓練するために、図 1 に示すように、以下の 3 つのステップを実行します。

1. **パッチの生成：**病理医によって注釈された座標を使用して、スライド画像を同じサイズのパッチにクロップします。各画像は何千ものパッチを生成することができ、腫瘍または正常とラベル付けされています。
2. **深層学習：**転移学習によって、事前学習したモデルを使用して画像パッチから特徴を抽出し、Apache Spark を使用してバイナリ分類器をトレーニングして腫瘍と正常パッチを予測します。
3. **スコアリング：**次に、MLflow を使用して記録された学習モデルを使用して、与えられたスライド上に確率ヒートマップを投影します。

Human Longevity が放射線画像の前処理に使用したワークフローと同様に、Apache Spark を使用してスライドとアノテーションの両方を操作します。モデルのトレーニングでは、Keras の事前トレーニング済みの InceptionV3 モデルを使って特徴を抽出することから始めます。

この目的のために、我々は Pandas の UDF を利用して特徴抽出を並列化しています。この技術の詳細については、転移学習のための特徴抽出（AWS | Azure）を参照してください。なお、この手法は InceptionV3 に特化したものではなく、他の事前学習済みモデルにも適用可能です。

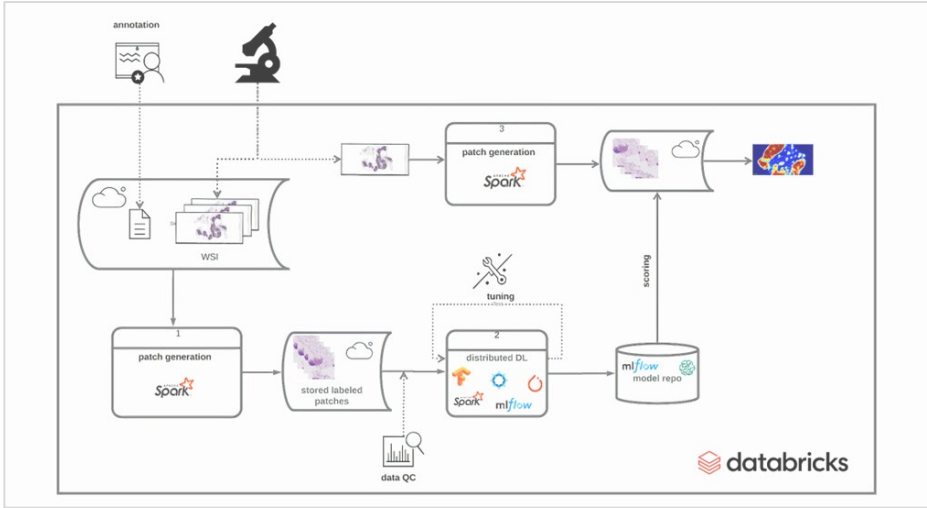


図1： WSI データに基づいた DL モデルのトレーニングと展開のための
エンドツーエンドソリューションの実装

画像の前処理と ETL

病理医は、Automated Slide Analysis Platform などのオープンソースのツールを使用して、WSI 画像を非常に高い解像度でナビゲートし、スライドに注釈を付けて臨床的に関連性のある部位をマークできます。アノテーションは XML ファイルとして保存でき、サイトを含むポリゴンのエッジの座標や、ズームレベルなどの他の情報と一緒に保存できます。一連の基底真実スライド上でアノテーションを使用するモデルを訓練するには、画像ごとにアノテーションされた領域のリストをロードし、これらの領域を画像と結合し、アノテーションされた領域を削除する必要があります。このプロセスが完了したら、画像パッチを機械学習に使用できます。

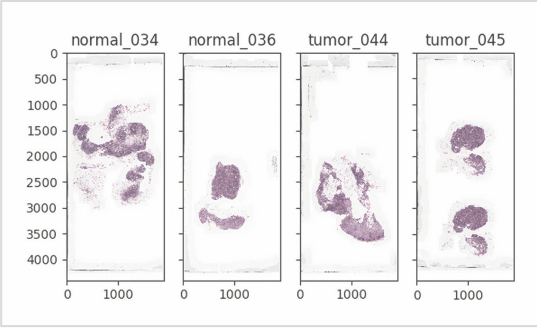


図2：Databricks Notebook の WSI イメージの可視化

このワークフローでは XML ファイルに保存されたアノテーションを使用しますが、ここでは簡単のため、Camelyon16 のデータセットに NCRF 分類器を構築した Baidu Research チームが作成した前処理済みのアノテーションを使用しています。これらのアノテーションは CSV エンコードされたテキストファイルとして保存され、Apache Spark が DataFrame にロードします。次の Notebook セルでは、腫瘍と正常パッチの両方のアノテーションをロードし、ラベル 0 を正常スライスに、1 を腫瘍スライスに割り当てます。次に、座標とラベルを一つの DataFrame に結合します。

多くの SQL ベースのシステムでは組み込みの操作が制限されていますが、Apache Spark ではユーザー定義関数（UDF）を豊富にサポートしています。UDF を使うと、Apache Spark DataFrame 内のデータに対して、Scala、Java、Python、R のカスタム関数を呼び出すことができます。ワークフローでは、OpenSlide ライブラリを使用して画像からパッチを抽出する Python UDF を定義します。処理する WSI の名前、パッチの中心の X と Y の座標、パッチのラベルを受け取り、後にトレーニングに使用されるタイルを作成する Python 関数を定義します。

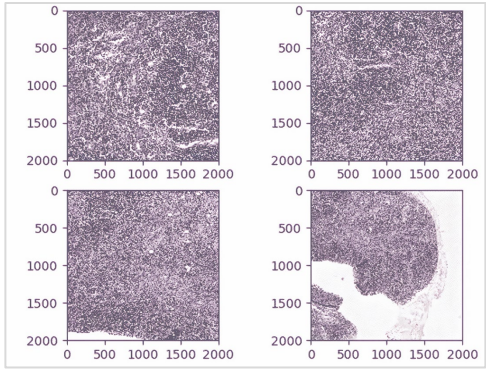


図3：異なるズームレベルでのパッチの可視化

次に OpenSlide ライブラリを使ってクラウドストレージから画像を読み込み、与えられた座標範囲をスライスします。OpenSlide は、Amazon S3 や Azure Data Lake Storage からのデータ読み込む方法をネイティブに理解していませんが、Databricks File System （DBFS） FUSE レイヤーを使用することで、OpenSlide は複雑なコード変更なく、これらのブロブストアに格納されているデータに直接アクセスできます。そして、私たちの関数が DBFS FUSE レイヤーを使用してパッチを書き戻します。

このコマンドでは、Camelyon16 データセットから約 174,000 個のパッチを databricks-datasets 上に生成するのに約 10 分かかります。コマンドが完了したら、パッチをロードして Notebook に直接インラインで表示できます。

転移学習と MLflow を用いた、腫瘍・正常病理分類器のトレーニング

前のステップでは、パッチと関連するメタデータを生成し、生成した画像タイルをクラウドストレージを使用して保存しました。さて、スライドのセグメントに腫瘍転移が含まれているかどうかを予測するためのバイナリ分類器を訓練する準備ができました。これを行うには、事前に学習した深層ニューラルネットワークを使用して各パッチから特徴を抽出し、分類タスクにsparkmlを使用するために転移学習を使用します。この手法は、多くの画像処理アプリケーションにおいて、スクラッチからのトレーニングよりも優れていることが多いです。まず、Keras の事前学習された重みを使用して、InceptionV3 のアーキテクチャから始めます。

Apache Spark の DataFrames には Image スキーマが組み込まれており、全てのパッチを直接DataFrameにロードすることができます。次に、Pandas UDF を使用して、Keras を使用して InceptionV3 に基づいてイメージを特徴量に変換します。各画像を特徴化したら、spark.ml を使って、各パッチの特徴とラベルの間のロジスティック回帰を行います。ロジスティック回帰モデルを MLflow でログに記録し、後でサービスのためにモデルにアクセスできるようにしています。

Databricks 上で ML ワークフローを実行する場合、ユーザーは管理された MLflow を利用することができます。Notebook を実行するたびに、またトレーニングを行うたびに、MLflow は自動的にパラメタ、メトリクス、および指定された成果物をログに記録します。さらに、学習したモデルは保存され、後でデータ上のラベルを予測するために使用することができます。MLflow がどのようにしてDatabricks 上での ML ワークフローのフルサイクルを管理するために活用できるかについては、興味のある読者の方は、これらのドキュメントを参照してください。

ワークフロー	時間
パッチ生成	10 分
特徴エンジニアリング、トレーニング	25 分
スコアリング（各スライドごと）	15 秒

表 1： Databricks ML ランタイム 6.2 を使用した 2-10 r4.4xlarge ワーカーを使用したワークフローのさまざまなステップのランタイム（databricks-datasets に含まれるスライドから抽出された 17 万のパッチについて）

表1 は、ワークフローのさまざまな部分に費やされた時間を示しています。17 万個のサンプルでのモデルトレーニングは 25 分以内で、87% の精度で行われていることがわかります。

実際には、より多くのパッチが存在する可能性があるため、分類に深層ニューラルネットワークを使用することで、精度を大幅に向上させることができます。このような場合には、分散訓練技術を利用して訓練プロセスをスケールアップさせることができます。Databricks のプラットフォーム上では、HorovodRunner ツールキットがパッケージ化されており、ML コードにわずかな変更を加えるだけで、大規模なクラスタに分散してトレーニングタスクを実行することができます。このブログ記事では、Databricks 上で ML ワークフローをスケーリングする方法について、その背景を説明しています。

推論

分類器を訓練したので、スライド上に転移の確率のヒートマップを投影するために分類器を使用します。そのためには、まずスライド上の関心のあるセグメントにグリッドを適用してから、学習プロセスと同様にパッチを生成し、予測に使用できる Spark DataFrame にデータを取得します。その後、MLflow を使用して学習したモデルをロードし、予測値を計算する DataFrame への変換として適用できます。

画像を再構成するために、Python の PIL ライブラリを使用して、転移部位を含む確率に応じて各タイルの色を変更し、全てのタイルをパッチで貼り合わせます。下の図 4 は、腫瘍セグメントの 1 つに確率を投影した結果を示しています。赤色の濃度がスライド上で転移の確率が高いことを示していることに注意してください。

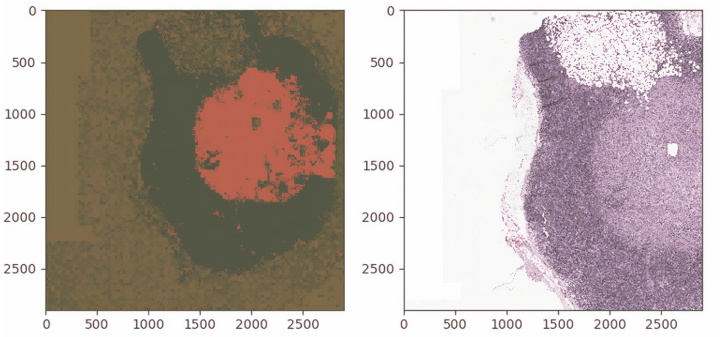


図 4： WSI の特定のセグメントへの予測値のマッピング

病理画像の機械学習を始める

今回のブログでは、Databricks と Spark SQL、SparkML、MLflow を使って病理画像上の機械学習のためのスケーラブルで再現性の高いフレームワークを構築する方法を紹介しました。具体的には、十分な規模の転移学習を使って識別子を使ってスライドのセグメントにがん細胞が含まれる確率を予測し、トレーニングされたモデルを使用してスライド上のがん増殖を識別してマッピングしました。

まずは、Databricks の無料トライアルを利用して WSI セグメンテーションの Notebook を試してみてください。

その他のソリューションについては、医療・ライフサイエンスのページをご覧ください。

第 7 章

車の分類のための畳み込みニューラルネットワークの実装

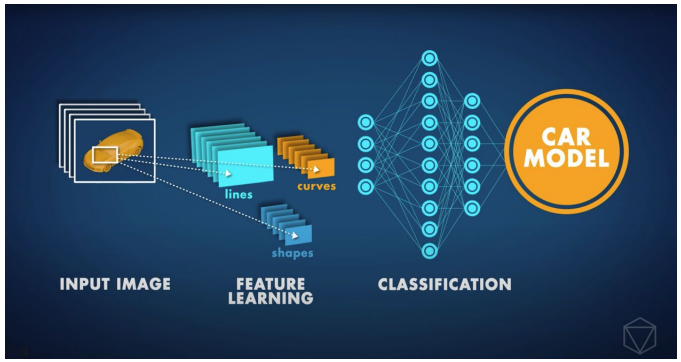
投稿者：
Dr. Evan Eames
Henning Kropp

2020年5月14日

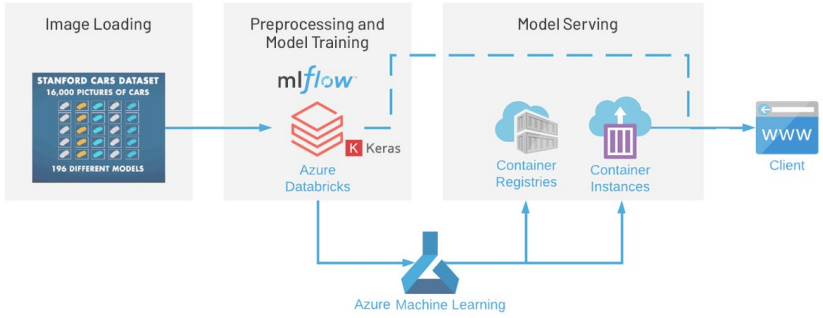
畳み込みニューラルネットワーク（CNN）は、主にコンピュータビジョンのタスクに使用される最先端のニューラルネットワークアーキテクチャです。CNN は、画像認識、物体定位、変化検知など、多くの異なるタスクに適用することができます。最近、当社のパートナーである Data Insights 社は、大手自動車会社から難しい依頼を受けました。それは、与えられた画像の中の車種を識別できるコンピュータ・ビジョン・アプリケーションを開発することです。異なる車種が非常に似ているように見えたり、どんな車でも周囲の環境や撮影角度によって大きく違って見えたりすることを考えると、このようなタスクは最近まで不可能とされていました。

しかし、2012 年頃から「The Deep Learning Revolution」（深層学習革命）よって、このような問題に対応できるようになりました。車の概念を説明される代わりに、コンピュータが画像を繰り返し学習し、そのような概念を自ら学習することができるようになったのだ。ここ数年、人工ニューラルネットワークの技術革新により、人間レベルの精度で画像分類を行うことができる AI が誕生しました。このような開発をもとに、私たちは深層 CNN を用いて車をモデル別に分類しました．ニューラルネットワークは、196 種類のモデルからなる 16,000 枚以上の車の写真を含むスタンフォード・カーズデータセット上で訓練されました。時間の経過とともに、ニューラルネットワークが車の概念と異なるモデルの区別方法を学習し、予測の精度が向上し始めたことがわかりました。

入力レイヤーと出力レイヤーとの間に複数のレイヤーを有する人工ニューラルネットワークの例を下図に示します。入力が画像、出力がカーモデルの分類です。



パートナーと協力し、データの前処理に Apache Spark™ と Koalas、モデルのトレーニングには Tensorflow を使った Keras、モデルと結果の追跡には MLflow、REST サービスの展開には Azure ML を使用し、エンドツーエンドの ML パイプラインを構築しました。Azure Databricks 内のこの構成は、ネットワークを高速かつ効率的にトレーニングするために最適化されており、多くの異なる CNN 構成をより迅速に試行できます。わずか数回の試行の後でも、CNN の確度は約 85% に達しました。



画像を分類するための人工ニューラルネットワークの設定

この記事では、ニューラルネットワークを稼働させる際に使用される主なテクニックのいくつかを概説します。ニューラルネットワークを自分で実行したい場合は、詳細なステップバイステップのガイドが含まれている完全なノートを以下で見つけることができます。

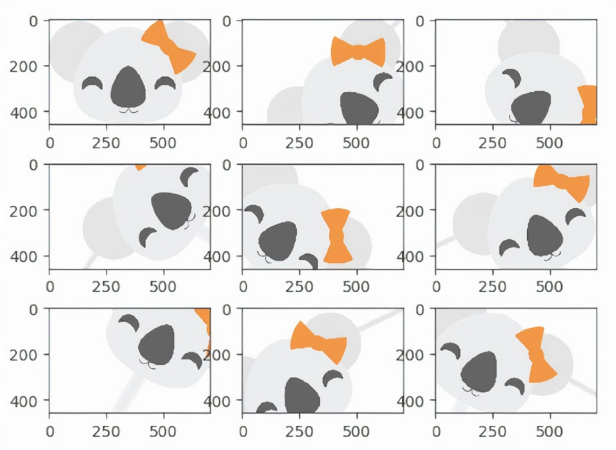
このデモでは、公開されている Stanford Cars データセットを使用していますが、これはより包括的な公開データセットの一つです。データは、ワークスペースにマウントできる ADLS Gen2 ストレージアカウントを通じて提供されます。

データの前処理の最初のステップでは、画像を hdf5 ファイル（トレーニング用とテスト用に 1 つずつ）に圧縮します。これをニューラルネットワークが読み込むことができます。hdf5 ファイルは、提供されている Notebook の一部として提供されている ADLS Gen2 ストレージの一部なので、お望みであれば、このステップを完全に省略することもできます。

- Stanford Cars のデータセットを HDF5 ファイルに読み込む
- イメージアップに Koalas を使う
- Keras で CNN をトレーニングする
- モデルを REST サービスとして Azure ML にデプロイする

Koalas を使った画像整形

収集されるデータの量と多様性は、深層学習モデルで達成できる結果に大きな影響を与えます。データ増強は、実際に新しいデータを収集しなくても学習結果を大幅に向上させることができる戦略です。大規模なニューラルネットワークを訓練するために一般的に使用されているクロップ、パディング、水平反転などのさまざまなテクニックを使用して、訓練やテストのために画像の数を増やすことで、データセットを人為的に膨らませることができます。



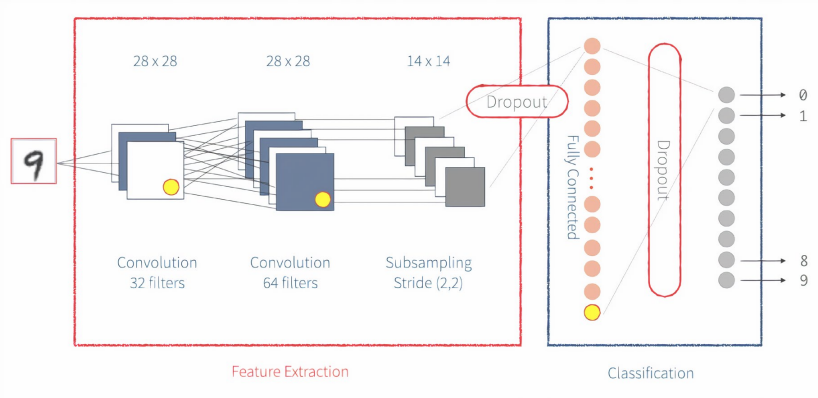
大規模なトレーニングデータのコーパスに画像処理を適用するのは、特に異なるアプローチの結果を比較する場合、非常にコストがかかることがあります。Koalas を使えば、Pythonで既存の画像増強のフレームワークを試したり、pandas API でお馴染みのデータサイエンスを使って、複数ノードを持つクラスタ上で処理をスケーリングしたりすることが簡単にできるようになります。

Keras での ResNet のコーディング

CNN を分解するとき、CNN は異なる「ブロック」で構成され、各ブロックは入力データに適用される操作のグループを単純に表しています。これらのブロックは大きく分けて以下のように分類されます。

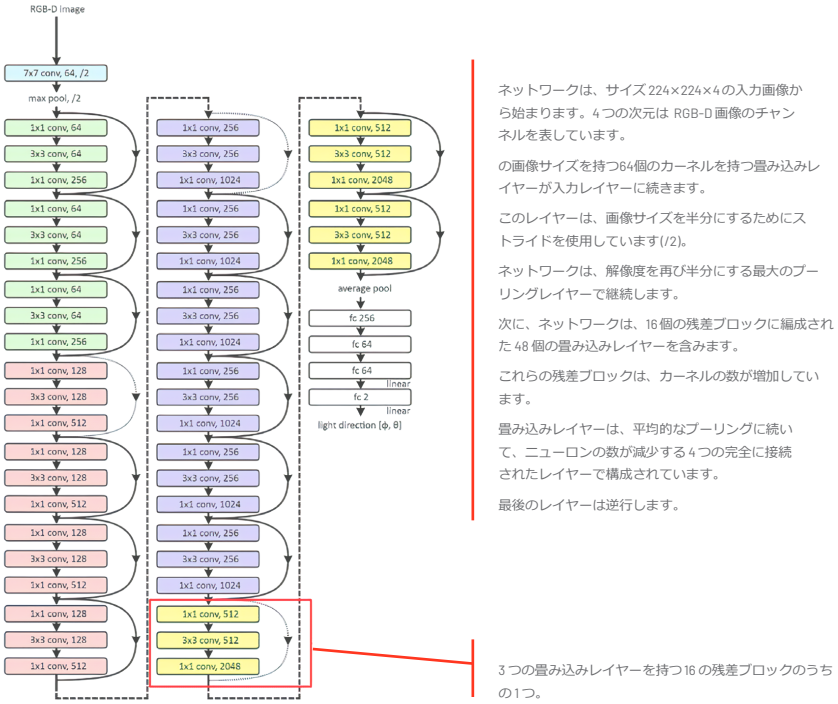
- **識別ブロック：**データの形状を同じに保つ一連の操作
- **畳み込みブロック：** 入力データの形状を小さくする一連の操作

CNN は、入力イメージをコンパクトな数値グループに減らすアイデンティティブロックと畳み込みブロック（または convBlocks）の両方のシリーズです。これらの結果として得られるそれぞれの数値は（正しくトレーニングされていれば）、最終的には画像の分類に役立つ何かを教えてくれるはずです。残留CNNはブロックごとにステップを追加します。ブロックを構成する演算が適用される前に一時的な変数としてデータを保存し、この一時的なデータを出力データに追加します。一般的に、この追加ステップは各ブロックに適用されます。例として、以下の図は、手書き数字を認識するための簡略化された CNN を示しています。



ニューラルネットワークを実装するにはさまざまな方法があります。直感的な方法の1つが、Keras を利用する方法です。Keras は、ニューラルネットワークを構成する個々のステップを実行するためのシンプルなフロントエンドライブラリを提供します。Keras は、Tensorflow バックエンドまたは Theano バックエンドで動作するように設定することができます。ここでは、Tensorflow バックエンドを使用します。

で終了します。Keras のネットワークは、以下のように複数のレイヤーに分割されています。私たちのネットワークでは、レイヤーの顧客実装も定義しています。



ネットワークは、サイズ 224×224×4 の入力画像から始まります。4 つの次元は RGB-D 画像のチャンネルを表しています。

の画像サイズを持つ64個のカーネルを持つ畳み込みレイヤーが入力レイヤーに続きます。

このレイヤーは、画像サイズを半分にするためにストライドを使用しています(/2)。

ネットワークは、解像度を再び半分にする最大のプーリングレイヤーで継続します。

次に、ネットワークは、16 個の残差ブロックに編成された 48 個の畳み込みレイヤーを含みます。

これらの残差ブロックは、カーネルの数が増加しています。

畳み込みレイヤーは、平均的なプーリングに続いて、ニューロンの数が減少する 4 つの完全に接続されたレイヤーで構成されています。

最後のレイヤーは逆行します。

3 つの畳み込みレイヤーを持つ 16 の残差ブロックのうちの 1 つ。

スケールレイヤー

訓練可能な重みを持つ任意のカスタム操作に対して、Keras では独自のレイヤーを実装することができます。膨大な量の画像データを扱う場合、メモリの問題に直面することがあります。初期状態では、RGB 画像には整数データ（0-255）が含まれています。バックプロパゲーション中に最適化の一部として勾配降下を実行すると、整数勾配ではネットワーク重みを適切に調整するのに十分な精度が得られないことがわかります。そのため、フロート精度に変更する必要がありますが、ここで問題が発生することがあります。画像を 224×224×3 に縮小した場合でも、1万枚のトレーニング画像を使用すると、10 億以上の浮動小数点エントリが発生することになります。データセット全体を浮動小数点精度にするのではなく、一度に 1 画像ずつ、必要なときだけスケーリングする「スケールレイヤー」を使用するのがより良い方法です。これは、モデルのバッチ正規化の後に適用します。この Scale レイヤーのパラメタは、トレーニングで学習可能なパラメタでもあります。

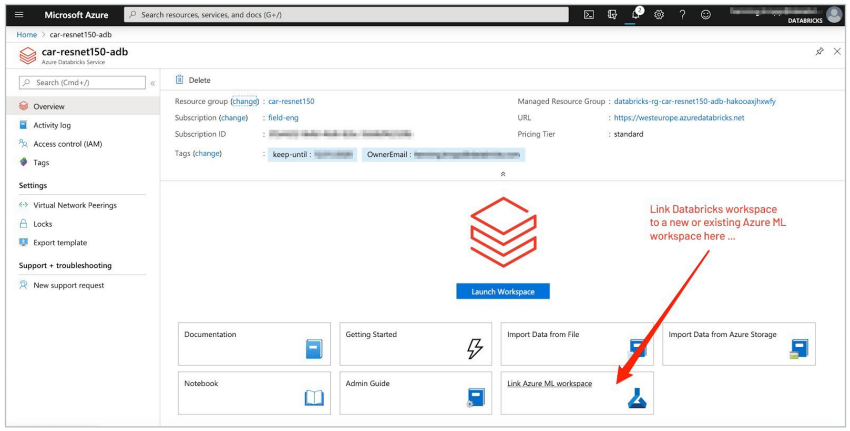
スコアリングの際にもこのカスタムレイヤーを使用するには、モデルとともにクラスをパッケージ化する必要があります。MLflow では、Keras custom_objects 辞書に名前（文字列）と Keras モデルに関連付けられたカスタムクラスや関数をマッピングすることで、これを実現します。MLflow はこれらのカスタムレイヤーを CloudPickle を使って保存し、`mlflow.keras.load_model()` と `mlflow.pyfunc.load_model()` でモデルが読み込まれたときに自動的に復元します。

```
mlflow.keras.log_model(model, "model", custom_objects={"Scale":
Scale})
```

MLflow と Azure 機械学習で結果を追跡する

機械学習の開発には、ソフトウェア開発以上の複雑さが伴います。無数のツールやフレームワークが存在するため、実験を追跡し、結果を再現し、機械学習モデルを展開することが困難になります。Azure Machine Learning と一緒に、Azure Databricks を使用して機械学習アプリケーションを確実に構築、共有、デプロイするために、MLflow を使用してエンドツーエンドの機械学習ライフサイクルを加速し、管理することができます。

自動的に結果を追跡するために、既存または新規の Azure ML ワークスペースを Azure Databricks ワークスペースにリンクすることができます。さらに、MLflow は Keras モデルの自動ロギング (`mlflow.keras.autolog()`) をサポートしているので、ほとんど手間がかかりません。

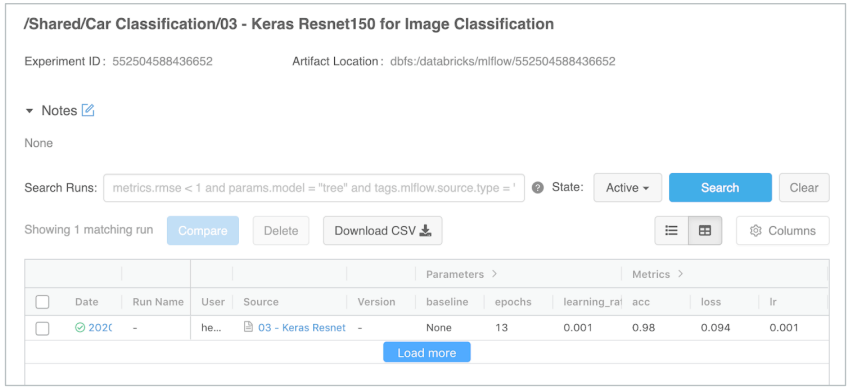


MLflow に組み込まれているモデル永続化ユーティリティは、Keras のようなさまざまな一般的な ML ライブラリからのモデルをパッケージ化するのに便利ですが、全てのユースケースをカバーしているわけではありません。例えば、MLflow の組み込み機能では明示的にサポートされていない ML ライブラリのモデルを使いたい場合があるかもしれません。あるいは、独自の推論コードやデータをパッケージ化して MLflow モデルを作成したい場合もあるでしょう。幸いなことに、MLflow はこれらのタスクを達成するため、カスタム Python モデルとカスタムフレーバーという 2 つのソリューションを提供しています。

このシナリオでは、REST API クライアントからのリクエストをサポートするモデル推論エンジンを使用できることを確認したいと思います。このために、以前に構築した Keras モデルをベースにしたカスタムモデルを使用して、内部に Base64 エンコードされたイメージを持つ JSON DataFrame オブジェクトを受け入れています。

```
import mlflow.pyfunc

class AutoResNet150(mlflow.pyfunc.PythonModel):

    def predict_from_picture(self, img_df):
    import cv2 as cv
    import numpy as np
    import base64

    # decoding of base64 encoded image used for transport over http
    img = np.frombuffer(base64.b64decode(img_df[0][0]), dtype=np. uint8)
    img_res = cv.resize(cv.imdecode(img, flags=1), (224, 224),
cv.IMREAD_UNCHANGED)
    rgb_img = np.expand_dims(img_res, 0)

    preds = self.keras_model.predict(rgb_img)
    prob = np.max(preds)

    class_id = np.argmax(preds)
    return {"label": self.class_names[class_id][0][0], "prob":
"{:.4}".format(prob)}

    def load_context(self, context):
    import scipy.io
    import numpy as np
    import h5py
    import keras
    import cloudpickle
    from keras.models import load_model

    self.results = []
    with open(context.artifacts["cars_meta"], "rb") as file:
        # load the car classes file
        cars_meta = scipy.io.loadmat(file)
        self.class_names = cars_meta['class_names']
        self.class_names = np.transpose(self.class_names)
    with open(context.artifacts["scale_layer"], "rb") as file:
        self.scale_layer = cloudpickle.load(file)

    with open(context.artifacts["keras_model"], "rb") as file:
        f = h5py.File(file.name,'r')
        self.keras_model = load_model(f, custom_objects={"Scale":
self.scale_layer})

    def predict(self, context, model_input):
    return self.predict_from_picture(model_input)
```

次のステップでは、この py_model を使用して Azure Container Instances サーバーにデプロイしますが、これは MLflow の Azure ML 統合によって実現できます。



Azure コンテナインスタンスでの画像分類モデルの展開

これまでに、訓練された機械学習モデルができ、クラウド上の MLflowでワークスペースにモデルを登録しました。最後のステップとして、このモデルを Web サービスとして Azur eコンテナインスタンス上にデプロイしたいと思います。

ウェブサービスはイメージであり、この場合は Docker イメージです。スコアリングロジックとモデル自体をカプセル化しています。このケースでは、カスタム MLflow モデル表現を使用しています。これにより、スコアリングロジックが REST クライアントからのケアイメージをどのように取り込むか、そしてレスポンスがどのように形作られるかを制御することができます。

```
# Build an Azure ML Container Image for an MLflow model
azure_image, azure_model = mlflow.azureml.build_image(
                      model_uri="{}/py_model"
                          .format(resnet150_latest_run.info.
artifact_uri),
                      image_name="car-resnet150",
                      model_name="car-resnet150",
                      workspace=ws,
                      synchronous=True)

webservice_deployment_config =AciWebservice.deploy_configuration()


# defining the container specs
aci_config = AciWebservice.deploy_configuration(cpu_cores=3.0,
memory_gb=12.0)

webservice = Webservice.deploy_from_image(
    image=azure_image,
    workspace=ws,
    name="car-resnet150",
    deployment_config=aci_config,
    overwrite=True)

webservice.wait_for_deployment()
```

コンテナインスタンスは、ワークフローをテストして理解するのに最適なソリューションです。スケーラブルな本番環境へのデプロイには、Azure Kubernetes Service の使用を検討してください。デプロイ方法について詳しくは、Microsoft の資料「Deploy machine learning models to Azure」を参照してください。

CNN 画像分類を始める

この記事とノートでは、エンドツーエンドのワークフロートレーニングを設定し、Azure上の本番でニューラルネットワークをデプロイする際に使用する主なテクニックを示しています。リンク先の Notebook の演習では、Keras、Databricks Koalas MLflow、Azure ML などのツールを使用して、独自の Azure Databricks 環境内でこれを作成するために必要な手順を解説しています。

開発者向けリソース

ノートブック：

- Load Stanford Cars data set into HDF5 files
  （Stanford Cars のデータセットを HDF5 ファイルに読み込み）
- Use Koalas for image augmentation（Koalas を使用した画像拡張）
- Train the CNN with Keras（Keras による CNN のトレーニング）
- Deploy model as REST service to Azure ML
  （モデルを REST サービスとして Azure ML にデプロイする）

動画：

- AI Car Classification With Deep Convolutional Neural Networks on Databricks
  （Databricks 上の深層畳み込みニューラルネットワークを用いた AI による車の分類）

GIT HUB：

- EvanEames | Cars

スライド資料：

- A Convolutional Neural Network Implementation for Car Classification
  （車の分類のための畳み込みニューラルネットワークの展開）

PDF ファイル：

- Convolutional Neural Network Implementation on Databricks
  （Databricks 上での畳み込みニューラルネットワークの展開）

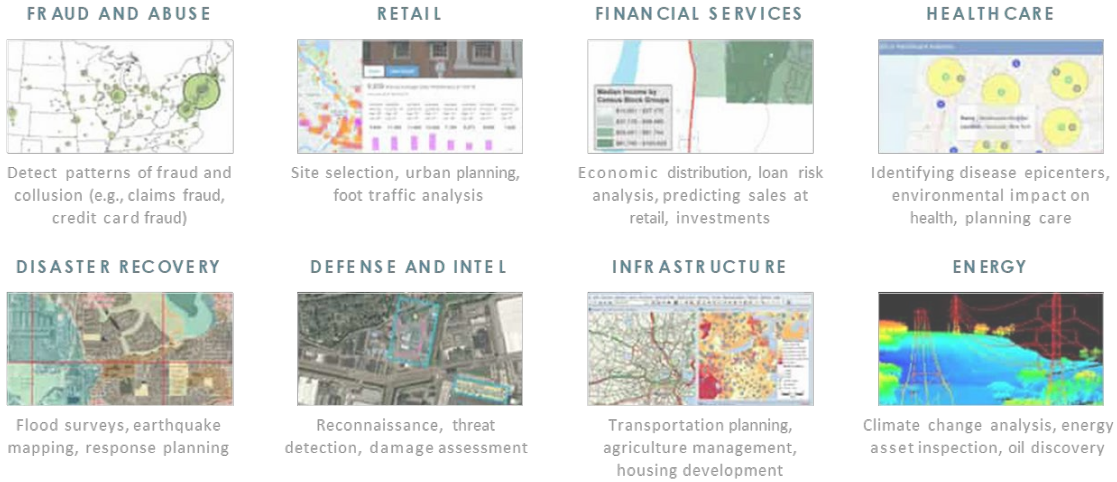
第 8 章

大規模な地理空間データの処理と分析

投稿者：
Nima Razavi
Michael Johns

2019年12月5日

近年のテクノロジーの進化と統合により、リアルタイムで正確な地理空間情報・ジオデータを活用した市場が活性化しています。数十億のハンドヘルドデバイスや IoT 機器、航空機、人工衛星に搭載された数千のリモートセンシングプラットフォームから、1日あたり数百エクサバイト規模の地理空間情報・ジオデータが生成されています。このような地理空間ビッグデータの拡大に、近年の機械学習の進展が加わり、業界ではこれを活用した新製品やサービスの開発が進められています。



図の説明：地理空間情報・ジオデータによるマップは、災害対策、防衛・インテリジェンス、インフラ事業、医療サービスなど、多くの分野で活用されています。

企業における地理空間情報・ジオデータの活用代表例として、ドローンを利用したマッピングや現地調査などのサービス提供があります。（参考：「インテリジェントクラウドとインテリジェントエッジの発展」）。地理空間データの活用で急速な成長を遂げているもう1つの産業は、自動運転車です。スタートアップ企業に加え、既存企業も車載センサーから豊富なコンテキスト情報を含んだ大量の地理空間データや空間インデックスを収集し、次世代の自動運転技術の開発に乗り出しています（参考：「データブリックス、Wejo のモビリティデータエコシステム構築を支援」）。小売業者や行政機関でも、地理空間情報・ジオデータの活用が模索されています。歩行者の交通量を分析することで、新規店舗の立地調査や行政による都市計画の改善に生かすことができます（参考：「歩行者交通情報データセットの構築」）。このように多くの業界で活用を期待されている地理空間情報・ジオデータですが、課題も存在します。

大規模な地理空間情報の分析における課題

第一の課題は、ストリーミングおよびバッチアプリケーションのスケーリングです。地理空間データの急増と、空間インデックスなどのアプリケーションが必要とするSLA に対して、従来のストレージや処理システムの能力では太刀打ちできない状況になっています。顧客データは、データ量、速度、ストレージコスト、スキーマオンライト（Schema on Write）の厳格な適用などの要件により、ここ数年、垂直方向にスケールした既存のジオデータベースから、データレイクへと流出しています。地理空間情報・ジオデータに投資する企業はあるものの、大規模で複雑なこれらのデータセットをダウンストリームの分析用に準備するための適切な技術アーキテクチャを持っているケースはほとんどありません。さらに、高度なユースケースでは大規模なデータを必要とする場合が多く、AI主導のプロジェクトの大半は、パイロット版から本運用への移行に失敗しています。

第二の課題は、地理空間データにおけるデータ形式の互換性です。この数十年の間に、下記のように多数の地理空間形式や、 位置情報を収集できる二次的データソースがそれぞれに特化した状態で開発されており、複雑さを増しています。

- GeoJSON、KML、Shapefile、WKT などのベクトル（ベクター）データ形式
- ESRI Grid、GeoTIFF、JPEG 2000、NITF などのラスターデータ形式
- AIS や GPS デバイスで使用される航法システム基準
- JDBC / ODBC 接続を介して、PostgreSQL／PostGIS として利用できる地理情報データベース
- Hyperspectral、Multispectral、Lidar、Radar プラットフォームでのリモートセンサー形式
- WCS、WFS、WMS、WMTS などの OGC Web 標準
- 位置情報タグが付いたログ、写真、動画、ソーシャルメディア
- 何らかの位置を示す非構造化データ

本記事では、大規模な地理空間情報・データの分析や処理に関するブログシリーズ第一弾として、Databricks の統合データ分析プラットフォームを使用して、上述の 2 つの主要課題を解決する一般的なアプローチの概要を解説します。

地理空間ワークロードのスケーリング

データブリックスは、世界中の数千社のお客様に、ビッグデータ分析と機械学習用の統合データ分析プラットフォーム（UDAP）を提供しています。UDAP は、Apache Spark™、Delta Lake、MLflow から構成されており、広範なサードパーティエコシステムやライブラリと統合されています。Databricks 統合データ分析プラットフォームは、実運用ワークロードに、エンタープライズクラスのセキュリティ、サポート、信頼性、性能を大規模に提供するソリューションです。地理空間情報・ジオデータベースのワークロードは複雑で、全てのユースケースを単体のライブラリではカバーできないため、Apache Spark でも地理空間のデータ型は直接には提供されていません。ただし、オープンソースコミュニティや各企業において、空間データを扱うライブラリの開発が進められており、その結果、多くの選択肢が利用できるようになっています。

空間結合や最近傍探索などの地理空間データ操作についてスケーリングを行う場合には、一般的に 3 つの方法があります。

1. Apache Spark の地理空間情報・ジオデータの分析を拡張する専用ライブラリを使用する：Databricks のユーザーが使用するライブラリとしては、GeoSpark、GeoMesa、GeoTrellis、Rasterframes などが挙げられます。通常、これらのフレームワークでは複数言語のバインディングが提供されており、形式化されていない場合に比べてスケーリングやパフォーマンスも向上しています。ただし、習熟するまでに多少の時間がかかります。

2. 下記のようなシングルノードライブラリをユーザー定義関数（UDF）を使って都度ラッピングし、Spark DataFrame を使用して分散環境で処理する：
   - GeoPandas
   - Geospatial Data Abstraction Library (GDAL)
   - Java Topology Service (JTS)

   このアプローチはコードの追加や修正がほとんど不要で、既存のワークロードをスケーリングする場合に最も簡単です。ただし、リフトアンドシフトになるため、パフォーマンスの低下も見込まなければなりません。

3. グリッドシステムを使用してデータをインデックス化し、生成されたインデックスを活用して地理空間情報・ジオデータベースを操作する：これは非常に大規模なワークロードやコンピューティングリソースに制約があるワークロードを扱う際に一般的なアプローチです。グリッドシステムの例には、S2、GeoHex、Uber の H3 などが挙げられます。グリッドシステムでは、地理上の特徴を、同一のポリゴン（多角形）やポイント（点）の組み合わせによるマス目（グリッド）に沿った近似値で表現します。そうすることで、地理空間データをそのまま処理することによる負荷の増大を避け、より効率的なスケーリングを図ります。具体的な実装としては、グリッドの精度を 1 つに固定することで、若干の精度低下を許容してパフォーマンスを優先する方法と、複数のグリッド精度を導入してパフォーマンスの低下を許容する代わりに精度を維持する方法があります。

以下の例は、ニューヨーク市のタクシーの乗り降りについてまとめられたデータセットです。また、ニューヨーク市のタクシー区域のデータを、ポリゴンで構成されるジオメトリとして使用しています。ニューヨーク市の 5 つの行政区とその近隣区域が対象です。元の CSV ファイルを Delta Lake テーブルに変換するための準備とクリーニングの手順については、こちらの Notebook を参照してください。Delta Lake テーブルに変換することで、より正確で効率よく扱えるデータソースとなります。

ベースとなる DataFrameは、Delta Lake テーブルから Databricks を使用して読み込んだタクシーの乗降データです。

```
%scala
val dfRaw =
spark.read.format("delta").load("/ml/blogs/geospatial/delta/nyc-green")
display(dfRaw) // showing first 10 columns
```

vendor_id	pickup_datetime	dropoff_datetime	store_and_forward	rate_code_id	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passener_count
2	2017-09-30 23:48:04	2017-09-30 23:57:43	N	1	82	7	2	1.89	9
2	2017-09-30 23:50:24	2017-09-30 23:55:30	N	1	25	181	6	1.26	6
2	2017-09-30 23:28:29	2017-09-30 23:37:29	N	1	41	159	1	2.28	9
2	2017-09-30 23:46:44	2017-09-30 23:54:59	N	1	42	41	1	1.09	7
2	2017-09-30	2017-09-30 23:31:49	N	1	33	189	1	2.35	10

Showing the first 1000 rows.

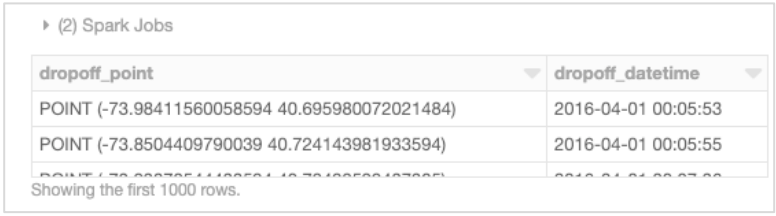
例：Databricks を利用して Delta Lake テーブルから読み込んだ地理空間データ

Apache Spark の地理空間ライブラリを使用したデータベース操作

ここ数年で、Apache Spark の地理空間情報の分析機能を拡張する複数のライブラリが開発されています。ユーザー定義型（UDT）やユーザー定義関数（UDF）を個別に登録して地理空間データに適用可能にするには、その都度、空間データを処理するためのロジックを用意する必要があります。これらのフレームワークを使用すれば個別の対処をある程度は省略し、負担を軽減できます。なお、本記事では複数の空間データフレームワークを使用していますが、それらはさまざまな機能について解説することを主眼として選択されたものです。Databricks で空間ワークロードを処理する際には、上述以外のフレームワークが適している場合もあります。

次に、ベースデータとして読み込んだ DataFrame を、緯度・経度属性に基づいてポイントジオメトリに変換します。分散環境での実行のために UDF を使用します。詳しくは本記事末尾にある Notebook を参照してください。フレームワークのクラスタへの追加、UDF／UDT を登録するための初期化呼び出しの方法を解説しています。手始めに、ベクトルデータの処理に特化したフレームワーク GeoMesa をクラスタに追加しました。データの取り込みは主に JTS と Spark SQL を統合して行います。そうすることで、登録済みの JTS ジオメトリクラスへの変換および利用が容易になります。ポイントジオメトリオブジェクトは、st_makePoint 関数に緯度・経度を渡して作成します。これは UDF 関数で、データ列に直接適用できます。

```
%scala val df = dfRaw
.withColumn("pickup_point", st_makePoint(col("pickup_longitude"),
col("pickup_latitude")))
.withColumn("dropoff_point",
st_makePoint(col("dropoff_longitude"),col("dropoff_latitude")))
display(df.select("dropoff_point","dropoff_datetime"))
```



(2) Spark Jobs

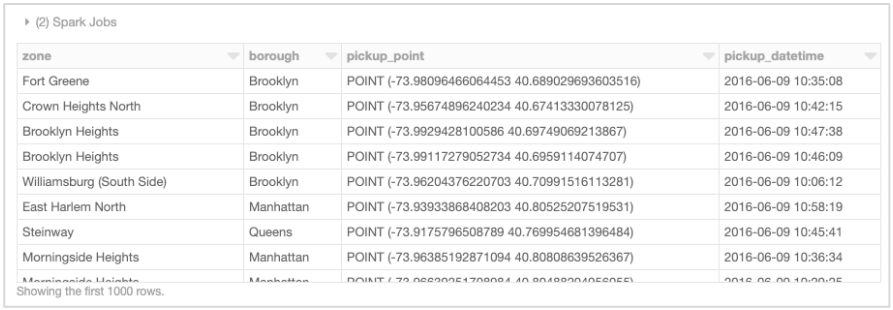
dropoff_point	dropoff_datetime
POINT (-73.98411560058594 40.695980072021484)	2016-04-01 00:05:53
POINT (-73.8504409790039 40.724143981933594)	2016-04-01 00:05:55

Showing the first 1000 rows.

例： UDF を使用して分散環境で DataFrame を操作し、地理空間データの緯度・経度属性をポイントジオメトリに変換する。

分散環境下での空間結合も可能です。サンプルでは、GeoMesa 提供の st_contains UDF を使用して、乗車地点に対する全ポリゴンの結合結果を生成しています。

```
%scala
val joinedDF = wktDF.join(df, st_contains($"the_geom", $"pickup_point"))
display(joinedDF.select("zone","borough","pickup_point","pickup_datetime"))
```



(2) Spark Jobs

zone	borough	pickup_point	pickup_datetime
Fort Greene	Brooklyn	POINT (-73.98096466064453 40.689029693603516)	2016-06-09 10:35:08
Crown Heights North	Brooklyn	POINT (-73.95674896240234 40.67413330078125)	2016-06-09 10:42:15
Brooklyn Heights	Brooklyn	POINT (-73.9929428100586 40.69749069213867)	2016-06-09 10:47:38
Brooklyn Heights	Brooklyn	POINT (-73.99117279052734 40.6959114074707)	2016-06-09 10:46:09
Williamsburg (South Side)	Brooklyn	POINT (-73.96204376220703 40.70991516113281)	2016-06-09 10:06:12
East Harlem North	Manhattan	POINT (-73.93933868408203 40.80525207519531)	2016-06-09 10:58:19
Steinway	Queens	POINT (-73.9175796508789 40.769954681396484)	2016-06-09 10:45:41
Morningside Heights	Manhattan	POINT (-73.96385192871094 40.80808639526367)	2016-06-09 10:36:34

Showing the first 1000 rows.

例： GeoMesa が提供する st_contains UDF を使用し、ピックアップポイントに対する全ポリゴンの結合結果を生成

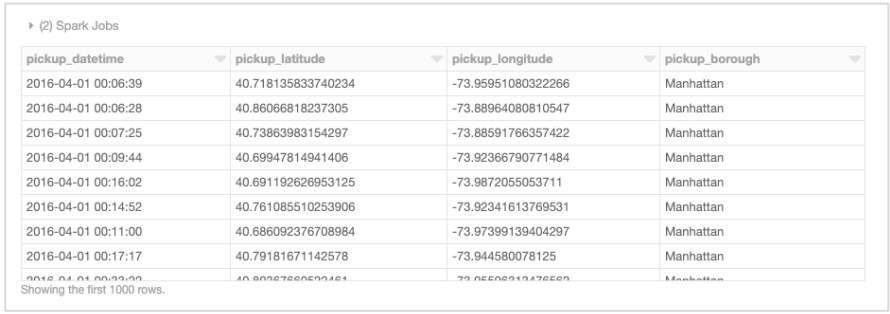
シングルノードライブラリを UDF でラップする

DataFrame の地理空間データ操作を分散環境で行うには、上記の空間フレームワークを使った方法に加え、既存のシングルノードライブラリをその都度 UDF でラッピングする方法もあります。この方法は、Scala、Java、Python、R、SQL など、Spark の全ての言語バインディングで利用できます。既存のワークロードを最小限のコード変更で使用できる最も簡単なアプローチです。このシングルノードでの方法を実行するには、まずニューヨーク市の行政区のデータを読み込み、ポイントインポリゴン操作を行う find_borough(...) UDFを定義します。次に geopandas を使用して GPS 位置情報を行政区に割り当てます。なお、この処理はベクトル化されたUDF でも実行可能です。パフォーマンスがより向上します。

```
%python
# read the boroughs polygons with geopandas
gdf =
gdp.read_file("/dbfs/ml/blogs/geospatial/nyc_boroughs.geojson")

b_gdf = sc.broadcast(gdf) # broadcast the geopandas dataframe to
all nodes of the cluster
def find_borough(latitude,longitude):
   mgdf = b_gdf.value.apply(lambda x: x["boro_name"] if
x["geometry"].intersects(Point(longitude, latitude))
   idx = mgdf.first_valid_index()
   return mgdf.loc[idx] if idx is not None else None

find_borough_udf = udf(find_borough, StringType())
```

これで、UDF を適用して Spark DataFrame にデータ列を追加できるようになりました。各乗車ポイントに行政区名が割り当てられます。

```
%python
# read the coordinates from delta
df = spark.read.format("delta").load("/ml/blogs/geospatial/delta/nyc-green")
df_with_boroughs = df.withColumn("pickup_borough", find_borough_udf(col("pickup_latitude"),col(pickup_longitude)))
display(df_with_boroughs.select(
   "pickup_datetime","pickup_latitude","pickup_longitude","pickup_borough"))
```



(2) Spark Jobs

pickup_datetime	pickup_latitude	pickup_longitude	pickup_borough
2016-04-01 00:06:39	40.718135833740234	-73.95951080322266	Manhattan
2016-04-01 00:06:28	40.86066818237305	-73.88964080810547	Manhattan
2016-04-01 00:07:25	40.73863983154297	-73.88591766357422	Manhattan
2016-04-01 00:09:44	40.69947814941406	-73.92366790771484	Manhattan
2016-04-01 00:16:02	40.691192626953125	-73.9872055053711	Manhattan
2016-04-01 00:14:52	40.761085510253906	-73.92341613769531	Manhattan
2016-04-01 00:11:00	40.686092376708984	-73.97399139404297	Manhattan
2016-04-01 00:17:17	40.79181671142578	-73.944580078125	Manhattan

Showing the first 1000 rows.

例： シングルノードを使用した場合の例。Geopandas によって GPS 位置情報を
ニューヨーク市行政区に割り当てている。

グリッドシステムによる空間インデックス生成

地理空間情報・ジオデータベースの操作は、その性質上、多くのコンピューティングリソースを必要とします。ポイントインポリゴン、空間結合、最近傍探索、ルートのスナッピングなど、いずれも複雑な操作です。グリッドシステムによる空間インデックス化は、このような地理空間操作をできるだけ避けることを目的としています。このようなアプローチを採用することで、実装時の効率的なスケーリングを実現しつつ、近似値を求める操作に際して警告を発することも可能になります。では、H3 の簡単なサンプルを見てみましょう。

H3 での空間操作のスケーリングは、基本的に 2 つの手順に分けられます。最初の手順で、ポイントやポリゴンで構成される地理的特徴について、H3 インデックスを計算します。地理的特徴は geoToH3(...)UDF で定義しています。2 つ目の手順で、H3 インデックスを使って地理空間操作（ポイントインポリゴン、空間結合、k 近傍探索など）を行います。今回のサンプルでは、multiPolygonToH3(...)UDF で定義しています。

```
%scala
import com.uber.h3core.H3Core
import com.uber.h3core.util.GeoCoord
import scala.collection.JavaConversions._
import scala.collection.JavaConverters._
object H3 extends Serializable {
   val instance = H3Core.newInstance()
}

val geoToH3 = udf{ (latitude: Double, longitude: Double,
resolution: Int) =>
   H3.instance.geoToH3(latitude, longitude, resolution)
}

val polygonToH3 = udf{ (geometry: Geometry, resolution: Int) =>
   var points: List[GeoCoord] = List()
   var holes: List[java.util.List[GeoCoord]] = List()
   if (geometry.getGeometryType == “Polygon”) {
      points = List(
         geometry
            .getCoordinates()
            .toList
            .map(coord => new GeoCoord(coord.y, coord.x)): _*)
   }
   H3.instance.polyfill(points, holes.asJava, resolution).toList
}
```

```
val multiPolygonToH3 = udf{ (geometry: Geometry, resolution: Int)
=>
   var points: List[GeoCoord] = List()
   var holes: List[java.util.List[GeoCoord]] = List()
   if (geometry.getGeometryType == “MultiPolygon”) {
      val numGeometries = geometry.getNumGeometries()
      if (numGeometries > 0) {
         points = List(
            geometry
               .getGeometryN(0)
               .getCoordinates()
               .toList
               .map(coord => new GeoCoord(coord.y, coord.x)): _*)
      }
      if (numGeometries > 1) {
         holes = (1 to (numGeometries - 1)).toList.map(n => {
            List(
               geometry
                  .getGeometryN(n)
                  .getCoordinates()
                  .toList
                  .map(coord => new GeoCoord(coord.y, coord.x)):
_*).
asJava
         })
      }
   }
   H3.instance.polyfill(points, holes.asJava, resolution).toList
```

ここで、上記の 2 つの UDF をニューヨーク市のタクシーのデータに適用します。また、行政区を示すポリゴンにも適用し、H3インデックスを生成します。

```
%scala
val res = 7 //the resolution of the H3 index, 1.2km
val dfH3 = df.withColumn(
   “h3index”,
   geoToH3(col(“pickup_latitude”), col(“pickup_longitude”),
lit(res))
)
val wktDFH3 = wktDF
   .withColumn(“h3index”, multiPolygonToH3(col(“the_geom”),
lit(res)))
   .withColumn(“h3index”, explode($”h3index”))
```

緯度と経度を示すポイントと行政区のポリゴンによるジオメトリを指定すると、h3index フィールドを結合条件とした空間結合が実行できるようになります。このように変数を指定することで、各ポリゴンの範囲内に含まれるポイントの数を集計することなどが行えます。通常は、数千から数百万のポリゴンと照合しなければならないポイントは数百万から数十億あり、これにはスケーラブルなアプローチが必要とされます。このブログでは取り上げませんが、近似値の処理が十分でない場合に地理空間操作を補完するために、空間インデックスを使用する手法などもあります。

```
%scala
val dfWithBoroughH3 = dfH3.join(wktDFH3,”h3index”)

display(df_with_borough_h3.select(“zone”,”borough”,”pickup_
point”,”pickup_datetime”,”h3index”))
```

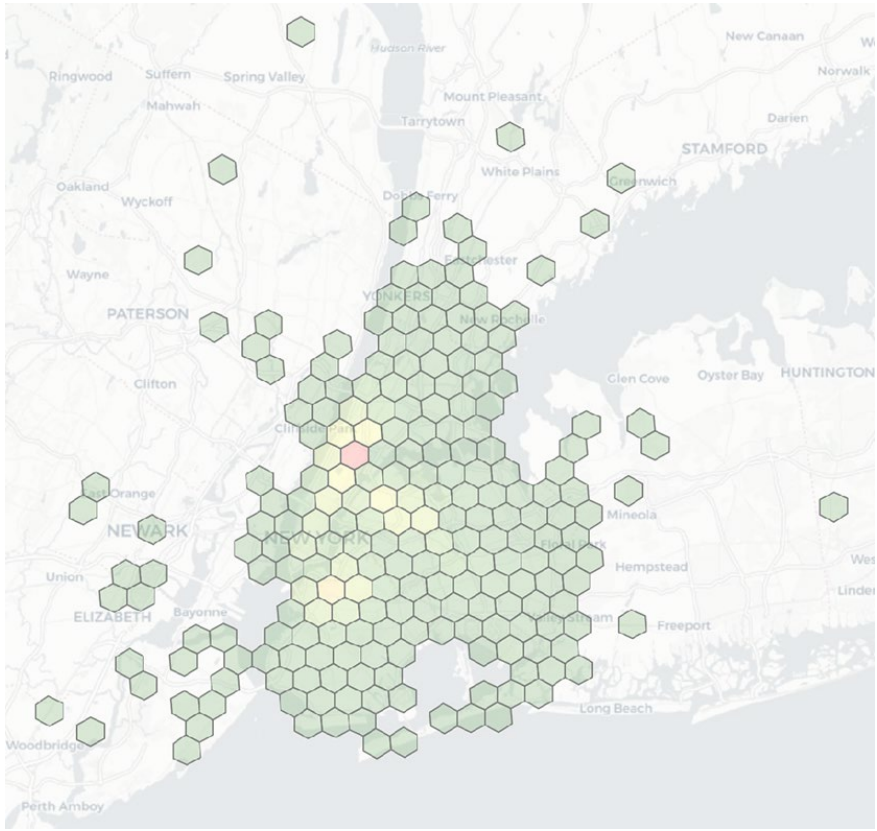
▸ (1) Spark Jobs

zone	borough	pickup_point	pickup_datetime	h3index
Morningside Heights	Manhattan	POINT (-73.95296478271484 40.80758285522461)	2016-06-09 10:14:34	613229523000885247
Central Harlem	Manhattan	POINT (-73.94908905029297 40.80293655395508)	2016-06-09 10:04:08	613229523028148223
Brooklyn Heights	Brooklyn	POINT (-73.99422454833984 40.69488525390625)	2016-06-09 10:52:24	613229551411003391
Van Nest/Morris Park	Bronx	POINT (-73.84475708007812 40.847774505615234)	2016-06-09 10:23:52	613229520937287679
Astoria	Queens	POINT (-73.9139633178711 40.76524353027344)	2016-06-09 10:25:38	613229524726841343
Morningside Heights	Manhattan	POINT (-73.95944213867188 40.80912399291992)	2016-06-09 10:42:56	613229523000885247
Park Slope	Brooklyn	POINT (-73.98164367675781 40.66694641113281)	2016-06-09 10:29:28	613229552660905983
Park Slope	Brooklyn	POINT (-73.97588348388672 40.67397689819336)	2016-06-09 10:53:01	613229552669294591
East Harlem North	Manhattan	POINT (-73.95088958740234 40.79750061035156)	2016-06-09 10:02:37	613229523015565311

Showing the first 1000 rows.

例： 緯度・経度ポイントとポリゴンジオメトリの空間結合を示した DataFrame テーブル。
特定のフィールドを結合条件として使用。

ここでは、タクシーの降車位置を視覚化した図を示します。緯度と経度による区分の精度を 7（辺長 1.22 km）に指定し、各区分で集計された数に基づいて色分けしています。



例： タクシーの降車地点の地理的可視化の例。タクシーの降車位置に関する地理空間を視覚化した例。緯度・経度による区分の大きさを 7（辺長1.22km）に指定し、各区分の集計数に基づいて色分けしている。

Databricks による空間データ形式の処理

地理空間情報・ジオデータは、緯度や経度など地球上の物理的な場所や範囲を示す参照点と、属性によって表現される特徴から構成されます。それらを規定するファイル形式には複数の種類がありますが、ここでは代表的なベクトル（ベクター）形式やラスター形式を取り上げて、Databricks での読み取り操作について示します。

ベクトルデータ

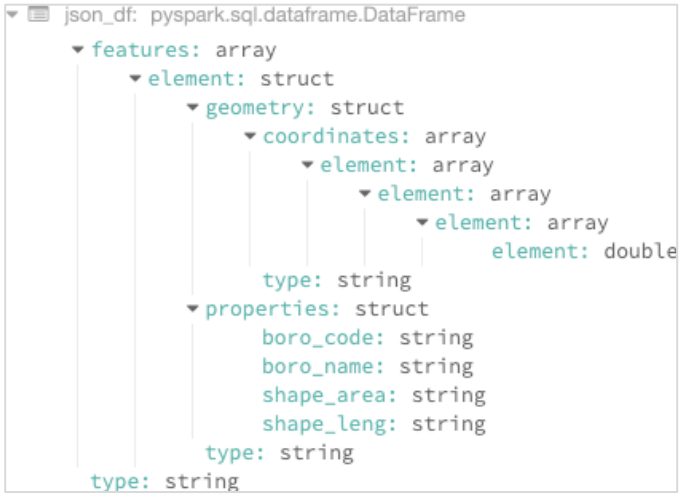
ベクトルデータとは、X 座標（経度）と Y 座標（緯度）で地理空間を表現したものです。高さを考慮する場合には、Z 座標（標高：メートル単位）も使用されます。基本的なオブジェクトの種類には、ポイント（点）、ライン（線分）、ポリゴン（多角形）があり、データ格納時の形式として Well-known-text (WKT)、GeoJSON、Shapefile がよく使用されます。以降でこの 3 つの形式について解説します。

WKT 形式で保存されたジオメトリを使用して、ニューヨーク市のタクシーのデータを読み込みます。このブログ内で使用されている他の API やデータソースと形式や仕様を揃えるため、データ構造には DataFrame を使用します。WKT のテキストコンテンツは the_geom フィールドに含まれており、st_geomFromWKT(...) UDF を呼び出すことで、対応する JTS のジオメトリクラスに容易に変換できます。

```
%scala
val wktDFText = sqlContext.read.format("csv")
 .option("header", "true")
 .option("inferSchema", "true")
 .load("/ml/blogs/geospatial/nyc_taxi_zones.wkt.csv")

val wktDF = wktDFText.withColumn("the_geom",
st_geomFromWKT(col("the_geom"))).cache
```

GeoJSON は、多くのオープンソース GIS パッケージで、特徴、プロパティ、空間範囲などのさまざまな地理空間データ構造のエンコードに使用されています。サンプルでは、ニューヨーク市の行政区分を読み込み、ワークフローに合わせてアプローチを指定します。データは JSON 形式に準拠しているため、Databricks に組み込まれている JSON リーダーを使用できます。.option(“multiline”,”true”) を使えば、ネストされたスキーマを持つデータを読み込めます。

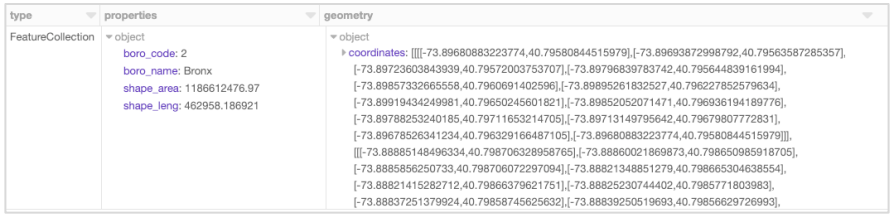
```
%python
json_df =
spark.read.option(“multiline”,”true”).json(“nyc_boroughs.
geojson”)
```



例： Databricks の JSON リーダーに .option(“multiline”,”true”) を使用し、ネストされたスキーマを持つデータを読み込んだ例。

Spark の explode 関数を使用すると、上記の状態から任意のフィールドを選択して、最上位の列に移すことができます。例えば、ジオメトリ、プロパティ、タイプを上位に移し、WKT の例と同様に、対応する JTS クラスにジオメトリをデータ変換することが可能です。

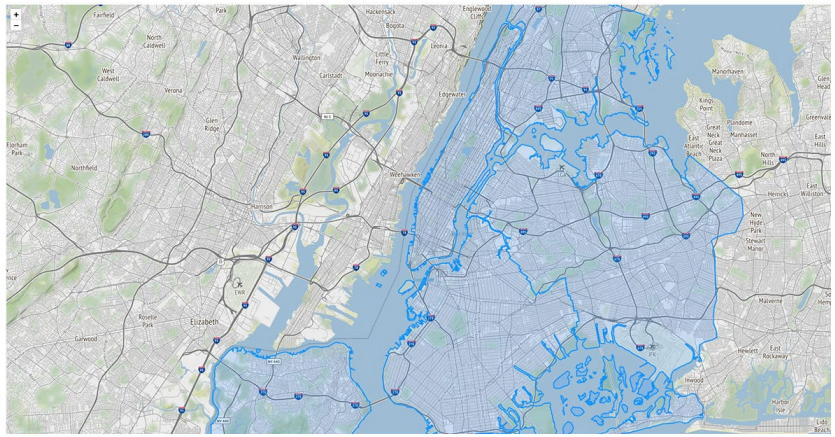
```
%python
from pyspark.sql import functions as F
json_explode_df = ( json_df.select(
 “features”,
 “type”,
F.explode(F.col(“features.properties”)).alias(“properties”)
).select(“*”,F.explode(F.col(“features.geometry”)).
alias(“geometry”)).drop(“features”))

display(json_explode_df)
```



例： Spark の explode 関数を使用して特定のフィールドを最上位レベルに移動しDataFrame テーブルを表示する。

既存の DataFrameを使用するか、Python の空間データレンダリングライブラリである Folium で直接データをレンダリングして、ニューヨーク市のタクシー区域データを Notebook 内で視覚化することもできます。そのためには、Databricks File System (DBFS) を使用します。DBFS は分散ストレージレイヤで実行され、既存のファイルシステム標準のデータ形式を維持したまま、コードを動作させることができます。この DBFS の FUSE マウントを使用することで、ローカルの API 呼び出しでファイルの読み込みと書き込みを行えるようになり、分散環境にない API でも容易にデータを読み込んで、インタラクティブにレンダリングできるようになります。下のコードでは、Python の open(...)コマンドで、引数の先頭に「/dbfs/」を付けることで、FUSE マウントを有効にしています。

```
%python
import folium
import json

with open (“/dbfs/ml/blogs/geospatial/nyc_boroughs.geojson”, “r”)
as myfile:
 boro_data=myfile.read() # read GeoJSON from DBFS using FuseMount

m = folium.Map(
 location=[40.7128, -74.0060],
 tiles=’Stamen Terrain’,
 zoom_start=12
)
folium.GeoJson(json.loads(boro_data)).add_to(m)
m # to display, also could use displayHTML(...) variants
```



例： 既存の DataFrameを使用するか、Python の空間データレンダリングライブラリ Foliumで直接データをレンダリングし、タクシー区域データを Notebook 内で視覚化することも可能。

Shapefile は、ESRI によって開発されたベクトルデータ形式です。空間的な位置と地理上の特徴に関する属性を格納します。拡張子の異なる同名のファイルを同一ディレクトリに置く形で構成され、*.shp、*.shx、*.dbf の必須ファイルに加えて複数のファイルを置くことができます。他によく使用されるものとして KML がありますが、今回は省略します。ニューヨーク市の建築物のデータに対して Shapefile を適用してみます。Shapefile の読み取りにはいくつか方法がありますが、ここでは GeoSpark を使用します。まず、デフォルトで利用できる ShapefileReaderを使用して rawSpatialDf DataFrame を生成します。

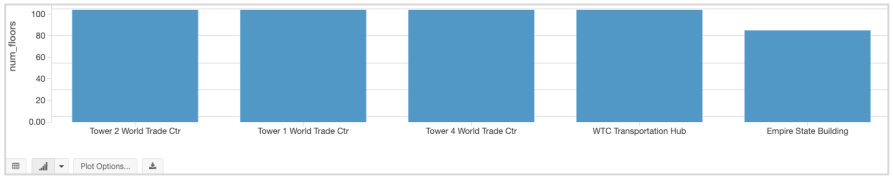
```
%scala
var spatialRDD = new SpatialRDD[Geometry]
spatialRDD = ShapefileReader.readToGeometryRDD(sc, “/ml/blogs/
geospatial/shapefiles/nyc”)

var rawSpatialDf = Adapter.toDf(spatialRDD,spark)
rawSpatialDf.createOrReplaceTempView(“rawSpatialDf”) //DataFrame
now available to SQL, Python, and R

display(json_explode_df)
```

rawSpatialDf を一時的なビューとして登録することで、Spark SQL の標準の構文で DataFrame を操作できるようになります。UDF を適用して Shapefile 形式の WKT をジオメトリにデータ変換するといった操作が可能です。

```
%sql
SELECT *,
 ST_GeomFromWKT(geometry) AS geometry -- GeoSpark UDF to convert
WKT to Geometry
FROM rawspatialdf
```

さらに、Databricks に組み込まれたインライン分析視覚化機能によって、ニューヨーク市の高層ビルを高い順にグラフ化するといったこともできます。

```
%sql
SELECT name,
 round(Cast(num_floors AS DOUBLE), 0) AS num_floors --String to
Number
FROM rawspatialdf
WHERE name <> ‘’
ORDER BY num_floors DESC LIMIT 5
```



例： Databricks によるインライン分析グラフ化の例。ニューヨーク市の高層ビルを高い順に表示。

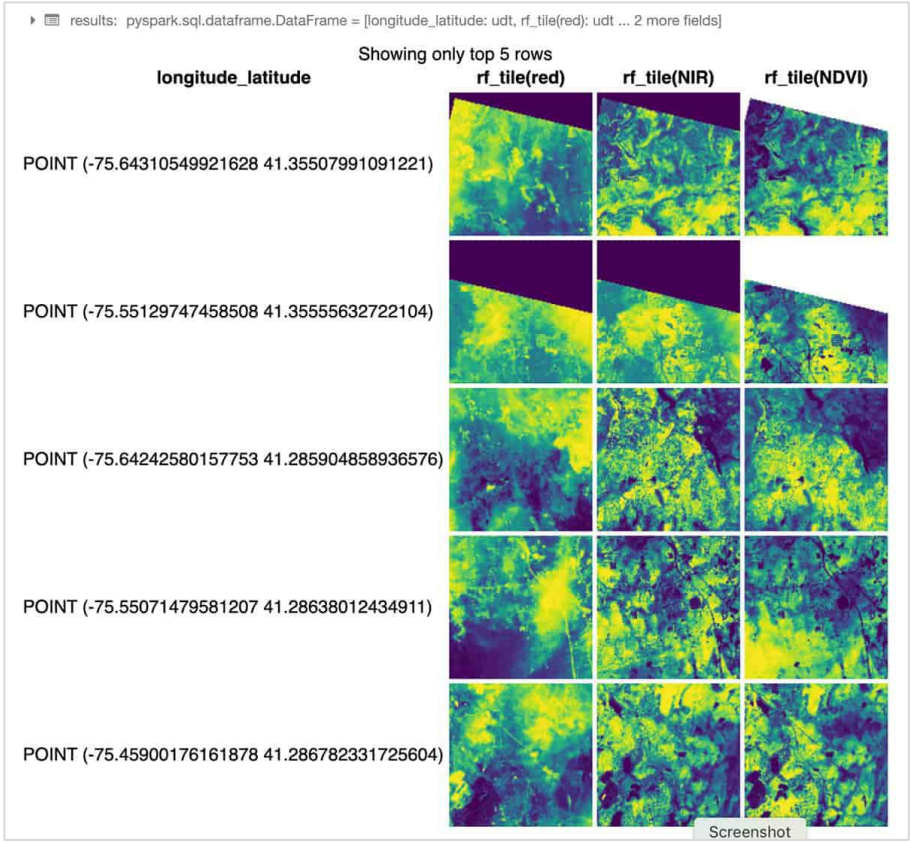
ラスターデータ

ラスターデータは、特徴情報が行と列で構成された格子状のセル（またはピクセル）を意味します。連続または離散したデータとしても扱われます。衛星画像、写真測量、スキャン地図などは全てラスターベースの地球観測（EO）データです。

次の Python コードのサンプルでは RasterFrames を使用しています。RasterFrames は DataFrame を扱う地理空間分析フレームワークで、GeoTIFF 形式の Landsat 8 画像（赤と近赤外線）の 2 つのバンドを読み込み、正規化差植生指数（NDVI）と組み合わせます。以下の例では、このデータを使用してニューヨーク市の植物衛生状況を評価しています。RasterFrame コンテンツの操作には rf_ipython モジュールを使用し、赤（red）、近赤外線（NIR）、正規化差植生指数（NDVI）のそれぞれの色調に応じた表示が行えるようになっています。Databricks の displayHTML(...)コマンドによって、Notebook 内に結果を表示させることもできます。

```
%python
# construct a CSV "catalog" for RasterFrames `raster` reader
# catalogs can also be Spark or Pandas DataFrames
bands = [f'B{b}' for b in [4, 5]]
uris = [f'https://landsat-pds.s3.us-west-
2.amazonaws.com/c1/L8/014/032/LC08_L1TP_014032_20190720_20190731_01_T1/LC08
_L1TP_014032_20190720_20190731_01_T1_{b}.TIF' for b in bands]
catalog = ','.join(bands) + '\n' + ','.join(uris)

# read red and NIR bands from Landsat 8 dataset over NYC rf =
spark.read.raster(catalog, bands) \
.withColumnRenamed('B4', 'red').withColumnRenamed('B5', 'NIR') \
.withColumn('longitude_latitude',
st_reproject(st_centroid(rf_geometry('red')), rf_crs('red'),
lit('EPSG:4326'))) \
.withColumn('NDVI', rf_normalized_difference('NIR', 'red')) \
.where(rf_tile_sum('NDVI') > 10000)

results = rf.select('longitude_latitude', rf_tile('red'), rf_tile('NIR'),
rf_tile('NDVI') displayHTML(rf_ipython.spark_df_to_html(results))
```

RasterFrame では、カスタム Spark データソースを通じて、GeoTIFF、 JP2000、MRF、HDF などのさまざまなラスターデータ形式を複数のサービスで読み込むことができます。また、ベクトル形式の GeoJSONと WKT/WKB の読み込みにも対応しています。RasterFrame コンテンツは、200 以上のラスター関数とベクトル関数を通じて、フィルタリング、変換、サマライズ、再サンプリング、ラスタライズを行えます。上のサンプルコードでは、st_reproject(...) や st_centroid(...) が使用されています。Python、SQL、Scala用の API も提供されており、Spark ML との相互運用も可能です。

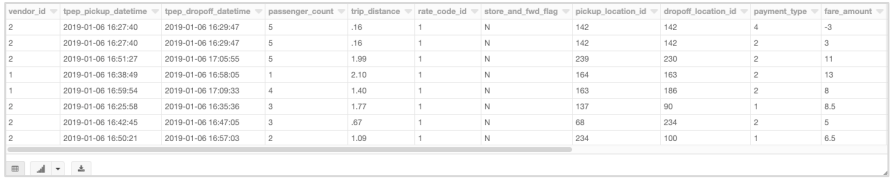


例： RasterFrame コンテンツは、200 以上のラスター関数とベクトル関数によるフィルタリング、変換、サマライズ、再サンプリング、ラスタライズが可能。

ジオデータベース

ジオデータベースは、小規模データについてはファイル単位、中規模データについては JDBC / ODBC 接続を介した利用が可能です。Databricks を使用すれば、あらかじめ用意されている JDBC／ODBC データソースで複数の SQL データベースにクエリできます。下に示す PostgreSQL への接続は、小規模のワークロードに対してよく使用されるもので、PostGIS の拡張機能を適用して実行されます。このような形で接続することで、ユーザーは既存のデータベースの状態を変更することなく、アクセスを維持できます。

```
%scala
display(
  sqlContext.read.format("jdbc")
    .option("url", jdbcUrl)
    .option("driver", "org.postgresql.Driver")
    .option("dbtable",
      """(SELECT * FROM yellow_tripdata_staging
      OFFSET 5 LIMIT 10) AS t""") //predicate pushdown
    .option("user", jdbcUsername)
    .option("jdbcPassword", jdbcPassword)
  .load)
```



vendor_id	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	rate_code_id	store_and_fwd_flag	pickup_location_id	dropoff_location_id	payment_type	fare_amount
2	2019-01-06 16:27:40	2019-01-06 16:29:47	5	.16	1	N	142	142	4	-3
2	2019-01-06 16:27:40	2019-01-06 16:29:47	5	.16	1	N	142	142	2	3
2	2019-01-06 16:51:27	2019-01-06 17:05:55	5	1.99	1	N	239	230	2	11
1	2019-01-06 16:38:49	2019-01-06 16:58:05	1	2.10	1	N	164	163	2	13
1	2019-01-06 16:59:54	2019-01-06 17:09:33	4	1.40	1	N	163	186	2	8
2	2019-01-06 16:25:58	2019-01-06 16:35:36	3	1.77	1	N	137	90	1	8.5
2	2019-01-06 16:42:45	2019-01-06 16:47:05	3	.67	1	N	68	234	2	5
2	2019-01-06 16:50:21	2019-01-06 16:57:03	2	1.09	1	N	234	100	1	6.5

Databricks で地理空間情報の分析を始める

多くの企業や行政機関が、空間参照データと企業のデータソースを組み合わせて実用的な情報を抽出し、さまざまなユースケースでイノベーションを実現しようとしています。このブログでは、Databricks の統合データ分析プラットフォームを使用して、地理空間情報ワークロードのスケーリングを容易にし、大規模データの収集、格納、分析を可能とするクラウドの活用方法について解説しました。

今後のブログでは、Databricks を活用した大規模地理空間データ処理のさらに高度な内容を取り上げる予定です。地理空間データ形式の詳細と、このブログで紹介したフレームワークについては、次の Notebook で確認できます。

- Data Prep Notebook
- GeoMesa + H3 Notebook
- GeoSpark Notebook
- GeoPandas Notebook
- Rasterframes Notebook

地理空間データについては、Databricks のドキュメントを今後も更新していきます。

第 9 章　導入事例

コムキャスト、エンターテインメントの未来を提供

「Databricks を導入したことで、より多くの情報に基づく意思決定がより迅速に行えるようになりました。」

コムキャスト社　プロダクト分析・行動科学部門
シニアディレクター Jim Forsythe 氏

米メディア大手のコムキャスト（Comcast）社は、テクノロジーを活用して数 100 万の視聴者に対し、パーソナライズされたエクスペリエンスを提供することを目指していました。しかし、データパイプラインの処理能力が不足していること、データサイエンスに関わる部門間のコラボレーションが困難であることが、目標達成の障壁となっていました。コムキャスト社は、問題の解決策として Delta Lake や MLflow が統合されている Databricks を導入。ペタバイト規模のデータのための高性能なデータパイプラインを構築し、機械学習モデル 100 種類以上のライフサイクルの管理を簡素化しました。その結果、エミー賞受賞にもつながる、革新的でパーソナライズされた視聴者エクスペリエンスを実現しました。

ユースケース

競争の激しいエンターテインメント業界では、立ち止まることは後退を意味します。コムキャスト社は、データの取り込みから、お客様に喜ばれる新機能を提供する機械学習モデルの展開まで、分析へのアプローチをモダナイズする必要があることに気づきました。

ソリューショ ンと効果

コムキャスト社は、分析のための統合プラットフォームの導入によって AI を活用した未来型エンターテインメントを先取りし、視聴者エクスペリエンスをより魅力的なものにすることでエンゲージメントを維持し、競争優位性を高めています。

- **エミー賞に輝く視聴者エクスペリエンス：**Databricks の導入により、インテリジェントな音声コマンドを使用した革新的な視聴者エクスペリエンスを実現。エンゲージメントを高めることに成功し、エミー賞を受賞。
- **コンピューティングのコストを 1/10 に削減：**Delta Lake の利用により、データの取り込みを最適化し、性能を向上させると同時にマシンの台数を 640 から 64 に削減。インフラ管理が容易になり、データの分析に注力できるようになった。
- **データサイエンスの生産性向上：**インタラクティブな単一のワークスペースで複数の言語をサポートすることで、グローバルに分散するデータサイエンティスト間のコラボレーションを促進。さらに、Delta Lake により、データ部門はデータパイプライン上のデータにいつでもアクセスできるようになり、迅速なモデルの構築とトレーニングが可能になった。
- **モデル展開の高速化：**異なるプラットフォームでのモデル展開が可能になり、展開時間が数週間から数分に短縮。

詳しく見る

第 9 章　導入事例

リジェネロン、ゲノム解読で創薬を加速

「Dababricks 統合データ解析プラットフォームの導入により、医療サイエンティストから計算生物学者まで、医薬品開発に携わる全ての人がデータに容易にアクセスし、分析し、インサイトを抽出できるようになりました。」

リジェネロン社
ゲノム情報学部長 Jeffrey Reid 博士

米バイオ医薬品製薬大手のリジェネロン（Regeneron）社は、ゲノムデータの活用によって新薬の開発を促進し、その薬を必要とする患者のもとに 1 日でも早く届けることを使命としています。しかし、ゲノム配列などの DNA 解析データから、人生に関わるような重大な発見や標的療法を生み出すことは、容易なことではありません。リジェネロン社のデータ部門は、ペタバイト規模のゲノム配列データと臨床データを分析する必要がありましたが、当時は処理性能とスケーラビリティが十分ではありませんでした。しかし現在は、Databricks の導入によって、ゲノムデータセット全体を素早く分析できるようになり、新しい治療方法の発見が加速されました。

ユースケース

新薬開発の成功率は低く、バイオ医薬品を含む現在医薬品開発パイプラインにある全実験薬の 95% 以上が失敗に終わると予測されています。リジェネロン遺伝学センターでは、40 万人以上のエクソーム解析データ（エキソン配列データ）と臨床データを含む電子健康記録（EHR）をペアにしたデータベースを構築し、改善に挑みました。このデータベースは、現在最も包括的な遺伝子データベースの 1 つとなっています。しかし、リジェネロン社は、この膨大なデータセットの分析において、次のような課題を抱えていました。

- ゲノムデータや臨床データが広く分散していたため、10 TB におよぶデータセット全体を分析してモデルをトレーニングすることは非常に困難だった。
- 従来のアーキテクチャを拡張しても、800 億以上のデータポイントの分析をサポートするのは困難で、コストも見合わなかった。
- データを分析に使用できるように ETL 処理するだけで、データ部門は何日もかかっていた。

ソリューショ ンと効果

リジェネロン社では、Amazon Web サービス（AWS）で実行する Databricks の統合データ分析プラットフォームの導入により、データサイエンスの生産効率を向上させ、運用をシンプルにし、バイオ医薬品創薬を加速させています。これにより、以前は不可能であった新しい方法でのデータ分析が可能に成りました。

- **創薬標的同定の高速化：**データサイエンティストや計算生物学者がデータセット全体に対して行うクエリの実行時間が 600 倍高速になった（30 分から 3 秒に短縮）。
- **生産性の向上：**コラボレーションの改善、DevOps の自動化、パイプラインの高速化（ETL は 3 週間から 2 日に短縮）を実現し、より広範な研究をサポートできるようになった。

詳しく見る

第 9 章　導入事例

ネーションワイド、保険数理モデルで保険を改革

「Databricks の導入により、データの種類にかかわらず、モデルをより迅速にトレーニングできるようになりました。その結果、より正確な価格予測が可能になり、収益に大きな影響を及ぼしています。」

ネーションワイド社
データサイエンティスト Bryn Clark 氏

利用可能なデータ量が爆発的に増大し、市場競争が激化するなか、保険会社はよい価格を顧客に提供することを目指しています。ネーションワイド（Nationwide）社では、ダウンストリーム ML のために数億件におよぶ保険記録の分析を必要としていましたが、従来のバッチ分析プロセスでは時間がかかり、確度が低く、保険金請求の頻度や重大度を予測するためのインサイトが限られていることに気づきました。Databricks を導入したことで、深層学習モデルの採用により、確度の高い価格予測が可能になり、保険業務の収益の増大に成功しています。

ユースケース

正確な保険料設定の鍵は、保険金請求から得られる情報を活用することにあります。しかし、保険金の請求頻度が低く、予測不可能で変動性の高い保険データを分析の対象としなければならず、結果的に価格設定が不正確になるという課題がありました。

ソリューショ ンと効果

ネーションワイド社では、Databricks 統合データ分析プラットフォームを活用して、データの取り込みから深層学習モデルの展開まで、分析プロセス全体を管理しています。フルマネージドプラットフォームにより、IT 運用が簡素化され、データサイエンスチームによるデータ活用の新たな機会を創出しています。

- **大規模なデータ処理：**データパイプライン全体のランタイムが 34 時間から 4 時間未満に短縮し、パフォーマンスが 9 倍向上した。
- **特徴量抽出の迅速化：**データエンジニアリングによる特徴量の識別時間が 5 時間から 20 分程度まで短縮した。
- **高速なモデルトレーニング：**トレーニング速度が 50% 向上し、新しいモデルの市場投入が迅速化した。
- **改良されたモデルスコアリング：**モデルのスコアリング時間が 3 時間から 5 分未満に短縮し、60 倍の高速化が実現した。

第 9 章　導入事例

コンデナスト、データと AI を活用した体験で読者エンゲージメントを強化

「Databricks は、強力なエンドツーエンドのソリューションです。専門分野や経験にかかわらず、チーム全員が大規模なデータに素早くアクセスし、実践的な気づきを得られるようになりました。」

コンデナスト社
AI インフラ担当主任エンジニア Paul Fryzel 氏

VOGUE、The New Yorker、WIRED など著名な雑誌を発行する米コンデナスト（Condé Nast）社は、データを駆使することで、印刷物、オンライン媒体、動画、SNS を通じて 10 億人を超える読者にコンテンツを提供しています。

ユースケース

出版大手コンデナスト社は、20 以上のブランドを運営しています。月当たりの Web コンテンツ閲覧者数は 1 億を超え、ページビューは 8 億回を超え、膨大なデータが蓄積されていました。データ部門は、機械学習の活用により、パーソナライズされたコンテンツの推薦とターゲティング広告を配信し、ユーザーエンゲージメントの向上を図りました。

ソリューショ ンと効果

コンデナスト社は、Databricks の提供する完全管理のクラウドプラットフォームを導入することで、オペレーションの簡素化、性能の向上、データサイエンスのイノベーションを実現しました。

- **顧客満足度の向上：**データパイプラインの改善により、コンデナスト社は、より適切なおすすめコンテンツを迅速に提供できるようになり、ユーザーエクスペリエンスが向上した。
- **集約型アプローチ：**データエンジニアリングとデータサイエンス部門が、共通のプラットフォームを共有し、新たなコンテンツ商品やエクスペリエンスの創出や問題解決に向けて協力体制を構築した。
- **スケーラビリティ：**データセットの増大に対応できるスケーラビリティと洞察力が得られた。
- **実稼働モデルの増産：** MLflow を利用することでデータサイエンスチームイノベーションが促進され、現在 1,200 のモデルを運用するに至っている。

詳しく見る

第 9 章　導入事例

ショータイム、ML の活用により データドリブンなコンテンツを配信

「Databricks プラットフォームの導入によって システム構成上の問題が全て解消しました。 データサイエンス部門の業務が飛躍的に進歩し、 生産性が大幅に向上しています。」

ショータイム社
データ戦略・消費者分析部門シニア VP
Josh McNutt 氏

SHOWTIME® は、「Shameless」、「Homeland」、「Billions」、「The Chi」、「Ray Donovan」、「SMILF」、「The Affair」、「Patrick Melrose」、「Our Cartoon President」、「Twin Peaks」など、受賞歴のあるオリジナルシリーズや限定シリーズを放送しているプレミアムテレビネットワークおよびストリーミングサービスです。

ユースケース

Showtime 局のデータ戦略部門は、組織全体におけるデータと分析の民主化に注力しています。膨大な視聴者データ（視聴した番組、時間帯、使用したデバイス、サブスクリプション履歴など）を収集し、機械学習を利用して視聴者の行動を予測し、番組構成や配信スケジュールの改善を図っています。

ソリューショ ンと効果

Databricks は、Showtime 局の組織全体におけるデータと機械学習の民主化を支援し、データをより積極的に活用する企業文化の醸成に寄与しています。

- **パイプラインを 6 倍高速化：**データパイプラインの高速化により、これまで 24 時間以上かかっていた作業が 4 時間未満で完了するようになり、より迅速な意思決定が可能になった。
- **インフラの複雑さを排除：**自動クラスタ管理機能を備えたフルマネージド型クラウドプラットフォームにより、データサイエンス部門は、ハードウェア構成、クラスタのプロビジョニング、デバッグなどに煩わされることなく、機械学習に集中できるようになった。
- **視聴者エクスペリエンスの革新：**データサイエンスにおけるコラボレーションと生産性の向上により、新しいモデルや特徴量の実運用化までの時間が短縮された。テストの迅速化により、より適切にパーソナライズされた視聴者エクスペリエンスが提供できるようになった。

詳しく見る

第 9 章　導入事例

シェル、よりクリーンな世界のためのエネルギーソリューションによる変革

「Databricks は我々に多大な価値をもたらしています。これまでで最大のスケールアップ製品として、Databricks プラットフォームを基盤とするインベントリ最適化ツールをグローバルに展開しています。結果として、年間数百万ドルのコスト削減を達成しています。」

シェル社
高度分析 CoE ゼネラルマネージャー
Daniel Jeavons 氏

シェル（Shell）社は、石油・ガス探査および生産技術のパイオニアであり、石油・天然ガスの生産、ガソリンおよび天然ガスの販売、石油化学製品の製造など、世界有数のエネルギー事業を展開しています。

ユースケース

シェルでは、世界各地の施設に 3,000 種類以上のスペアパーツをストックして生産の維持を図っています。操業を停止しなければならないような事態を防ぐには、適切な部品を適切なタイミングで入手することが重要です。しかし同時に、過剰な在庫を保持することによるコスト増大も回避しなければなりません。

ソリューショ ンと効果

Databricks のクラウドネイティブな統合データ分析プラットフォームが、シェルにおけるインベントリ管理とサプライチェーン管理の改善を支援しています。

- **予測型モデリング ：**スケーラブルな予測モデルを開発し、3,000 種類以上の材料を 50 以上の場所で展開。
- **履歴分析：**各材料モデルは、過去の課題分布を捉えるために、10,000 回のマルコフ連鎖モンテカルロ反復をシミュレート。
- **パフォーマンスの改善：**パフォーマンスの改善を重視するデータサイエンスチームは、Databricks 上の 50 ノードの Apache Spark™ クラスタでのインベントリの分析と予測にかかる時間を 48 時間から 45 分に短縮。
- **コスト削減 ：**年間数百万ドルに相当するコスト削減。

詳しく見る

第 9 章　導入事例

Riot Games、ゲーマーエンゲージメント向上と離脱回避に AI を活用

「データサイエンティストをクラスタの管理から解放したいと考えており、Databricks を導入しました。使いやすいマネージド型 Spark ソリューションにより、開発チームはゲーム体験の向上に集中できるようになりました。」

ライアットゲームズ社
データサイエンティスト Colin Borys 氏

Riot Games は、世界で最もプレイヤーに焦点を当てたゲーム会社になることを目標としています。同社は 2006 年に設立され、LA を拠点としています。League of Legend が有名で、毎月 1 億人以上のゲーマーがプレイしています。

ユースケース

ネットワークパフォーマンスの監視と、ゲーム内での暴言の制御により、ゲーム体験を向上させます。

ソリューショ ンと効果

Databricks は、Riot Games がスケーラブルで高速な分析を提供することで、プレイヤーのゲーム体験を向上させることを可能にします。

- **ゲーム内の購入体験の向上：**5千億以上のデータポイントに基づいて独自のオファーを提供する推薦エンジンを素早く構築し、本番で使用可能にすることで、ゲームプレイヤーは、必要なコンテンツを容易に発見できるようになった。
- **ゲームラグの解消：**ネットワークの問題をリアルタイムで検知する ML モデルを構築したことで、障害による機能停止を回避し、プレイヤーへの悪影響を解消した。
- **分析の高速化：**EMR と比較してデータの準備と調査の処理性能が 50% 向上し、分析が大幅に高速化した。

詳しく見る

第 9 章　導入事例

Eneco、ML の活用により エネルギー消費量と運用コストを削減

「Delta Lake と構造化ストリーミング機能を備えた Databricks の導入により、お客様へのアラートをほぼ遅延なく送信できるようになりました。その結果、お客様が住宅内の問題に対してプロアクティブに対応できるようになりました。」

エネコ社
データサイエンス部門長 Stephen Galsworthy 氏

エネコ（Eneco）社は、エネルギー使用量、快適性、家庭のセキュリティなどをコントロールできるスマートエネルギー管理デバイス「Toon」を開発したテクノロジー企業です。Eneco のスマートデバイスは、ヨーロッパ中の数十万の家庭に設置されており、住宅内の機器に取り付けられたセンサーから収集したペタバイト規模の IoT データからなるヨーロッパ最大のエネルギーデータセットを保持しています。このデータをもとに、エネルギー消費に関するパーソナライズされたレコメンデーションにより、顧客におけるエネルギー消費の抑制と快適な生活の支援をミッションとしています。

ユースケース

エネルギー消費に関するパーソナライズされたレコメンデーション — 機械学習と IoT データを活用し、各家庭のエネルギー消費量を低減させるためのパーソナライズされたレコメンデーションを提供するアプリ「Waste Checker」を開発しました。

ソリューショ ンと効果

エネコでは、Databricks 統合データ分析プラットフォームの導入により、データサイエンスとエンジニアリングの間に拡張性と協調性のある環境を構築し、データチームのイノベーションを加速させ、機械学習を活用したサービスの提供を迅速化しています。

- **コスト削減：**Databricks が提供するコスト削減機能（自動スケーリングクラスタ、Spot インスタンスなど）により、インフラストラクチャ管理の運用コストを大幅に削減しつつ、大量のデータを処理できるようになった。
- イノベーションの促進：従来型のアーキテクチャでは、PoC（概念実証）から本運用までに 12 か月以上を要していたが、Databricks を導入したことで、8 週間以内に完了できるようになった。これにより、エネコのデータチームによる ML を活用した新しい機能の開発と提供が迅速化した。
- エネルギー消費量の削減：アプリ「Waste Checker」によって、パーソナライズされたレコメンデーションを顧客に提供できるようになり、 6700万キロワット以上のエネルギーの削減が可能になった。

詳しく見る

データブリックスについて

データブリックスは、米国サンフランシスコに本社を置き、世界中に拠点を持つデータと AI の企業です。Apache Spark、Delta Lake、MLflow のオリジナル開発メンバーによる創業以来、データの活用によって難題解決に挑む組織の支援に取り組んできました。Databricks レイクハウスプラットフォームは、コムキャスト（Comcast）、コンデナスト（Condé Nast）、H&M をはじめ、フォーチュン 500 企業の 40% を含むさまざまな業界の 5,000 社以上におけるデータ、分析、AI の取り組みに活用されています。

Twitter、LinkedIn、Facebook での情報発信も行っております。

カスタムデモのご予約

Databricks 無料お試し

databricks