データサイエンス 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いるおり、自分が取り組んでいることに関連する実際の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から、今すぐに実践できる実用的なユースケースをご紹介します。コードサンプルを含む必要情報を提供していますので、データブリックスのプラットフォームを実際に使ってみることができます。

第 2 章　データブリックスによる金融時系列分析の民主化

Databricks Connect と Koalas による開発の高速化

投稿者：
Ricardo Portilla

2019 年 10 月 9 日

金融機関におけるデータサイエンティスト、データエンジニア、アナリストの役割には、数千億ドル規模の資産を守り、フラッシュクラッシュなどの数兆ドル規模の影響から投資家を守ることが含まれます（ただし、これに限定されません）。これらの問題の根底にある最大の技術的課題の1つは、時系列操作のスケーリングです。ティックデータ、地理空間データやトランザクションデータなどの代替データセット、ファンダメンタルズデータなどは、金融機関が利用できる豊富なデータソースの一例です。これらは全てタイムスタンプでインデックス化されています。リスク、不正行為、コンプライアンスなどの金融におけるビジネス上の問題を解決するには、最終的には何千もの時系列を並行して集計し、分析できるかどうかにかかっています。RDBMS ベースの古いテクノロジーは、取引戦略を分析したり、何年も前のデータを使って規制分析を行ったりする場合には、簡単にはスケールできません。さらに、多くの既存の時系列技術は、標準 SQL や Python ベースの API ではなく、特殊な言語を使用しています。

幸いなことに、Apache Spark™ には、時系列処理を自然に並列化するウィンドウ機能など、多くの組み込み機能が搭載されています。さらに、おなじみの pandas 構文を使って Apache Spark 経由で分散機械学習クエリを実行できるオープンソースのプロジェクト Koalas は、データサイエンティストやアナリストにこの機能を拡張するのに役立っています。

このブログでは、何十万ものティッカーに対して時系列関数を並列に構築する方法を紹介します。次に、ローカル IDE で関数をモジュール化し、Databricks Connect でリッチな時系列機能セットを作成する方法を実演します。最後に、金融異常検知やその他の統計分析にフィードするデータ準備をスケーリングしようとしている pandas のユーザーの方に、市場操作の例を使って、Koalas がどのように典型的なデータサイエンスのワークフローにスケーリングを透明化するかをお見せします。

時系列データソースの設定

まず、伝統的な金融時系列のデータセットであるトレードと気配値を摂取することから始めましょう。このブログのためにデータセットをシミュレートしました。このデータセットは、取引報告施設（トレード）とナショナルベストビッドオファー（NBBO）フィード（ニューヨーク証券取引所などの取引所）から受け取ったデータをモデルにしています。データの例はこちらからご覧いただけます。

www.tickdata.com/product/nbbo/

この記事では、一般的に基本的な財務用語を想定しています。インベストペディアのドキュメントを参照してください。以下のデータセットから注目すべきことは各タイムスタンプに `TimestampType` を割り当てたので、トレードの約定時間はと引用符の変更時間は、正規化のために `event_ts` にリネームされました。さらに、この記事に添付されている完全な Notebook に示されているように、最終的には　これらのデータセットをデルタ形式に変換することで、データの品質を確保し、柱状体を維持します。形式で、以下のような対話型のクエリに対して最も効率的です。

```
trade_schema = StructType([  StructField("symbol",
    StringType()),  StructField("event_ts",
    TimestampType()),  StructField("trade_dt",
    StringType()),  StructField("trade_pr",
    DoubleType())
])

quote_schema = StructType([  StructField("symbol",
    StringType()),  StructField("event_ts",
    TimestampType()),  StructField("trade_dt",
    StringType()),  StructField("bid_pr",
    DoubleType()),  StructField("ask_pr",
    DoubleType())
])
```

```
display(spark.read.format("delta").load("/tmp/finserv/delta/trades"))
```

▸ (1) Spark Jobs

symbol	event_ts	trade_dt	trade_pr
AMH	2017-08-31T11:58:35.000+0000	2017-08-31	347.3411812850558
EMIS	2017-08-31T22:52:54.000+0000	2017-08-31	348.2907055152273
AMH	2017-08-31T04:33:52.000+0000	2017-08-31	346.3701388789535
AMH	2017-08-31T02:32:37.000+0000	2017-08-31	346.3012590012465
KIO	2017-08-31T06:03:36.000+0000	2017-08-31	349.5138613212247
EMIS	2017-08-31T18:00:38.000+0000	2017-08-31	348.0215275764011
EMIS	2017-08-31T03:39:54.000+0000	2017-08-31	348.51713303367943
EMIS	2017-08-31T02:59:52.000+0000	2017-08-31	348.54131225455575
KWR	2017-08-31T10:02:30.000+0000	2017-08-31	348.96337472824437

Showing the first 1000 rows.

```
display(spark.read.format("delta").load("/tmp/finserv/delta/quotes"))
```

▸ (1) Spark Jobs

symbol	event_ts	trade_dt	bid_pr	ask_pr
COST	2017-08-31T00:10:19.000+0000	2017-08-31	343.69295468812896	350.909849275807
AMD	2017-08-31T00:10:19.000+0000	2017-08-31	347.04709899077204	348.5183995843159
KYN	2017-08-31T00:10:19.000+0000	2017-08-31	348.53269061203054	351.41896433711137
KYN	2017-08-31T00:10:19.000+0000	2017-08-31	344.7049081216955	349.80283794725966
KYN	2017-08-31T00:10:19.000+0000	2017-08-31	346.216800782748	348.6772930682145
EMIS	2017-08-31T00:10:19.000+0000	2017-08-31	349.4801250232342	351.0930879023341
EMIS	2017-08-31T00:10:19.000+0000	2017-08-31	346.94067005458823	348.7309464067882
EMIS	2017-08-31T00:10:19.000+0000	2017-08-31	346.54222291125706	348.25466426470564
CAF	2017-08-31T00:10:19.000+0000	2017-08-31	348.11208695271176	352.34177898768933

Showing the first 1000 rows.

Apache Spark を使った時系列のマージと集計

今日の金融市場では、世界全体で 60 万以上の株式が公開されています。取引や気配値のデータセットがこれだけの量の証券にまたがることを考えると、簡単に拡張できるツールが必要になります。Apache Spark は ETL のためのシンプルな API を提供しており、並列化のための標準エンジンでもあるので、標準的なメトリクスをマージして集約するためのツールとして、流動性、リスク、不正行為の理解に役立ちます。まず、取引と気配値のマージから始め、取引データセットを集約して、データをスライスする簡単な方法を紹介します。最後に、このコードをクラスにパッケージ化して、Databricks Connect を使った反復開発を高速化する方法を紹介します。次のページのメトリクスに使用したフルコードは、添付の Notebook にあります。

As-of join

As-of join は、左のタイムスタンプの時点で有効な最新の右の値を返す、一般的に使用される"マージ"手法です。ほとんどの時系列分析では、複数のタイプの時系列がシンボル上で結合され、別の時系列（トレードなど）に存在する特定の時間における1つの時系列（例：NBBO）の状態を理解します。下の例では、全てのシンボルについて、全ての取引についてNBBO の状態を記録しています。下の図に示すように、最初のベースとなる時系列（取引）から始め、NBBO データセットをマージして、各タイムスタンプに「取引の時点での最新のビッドとオファー」が記録されるようにしました。最新のビッドとオファーがわかれば、次のように計算できます。

このような指標は、どの時点で流動性が低下しているか（大きなスプレッドで示されている）を理解するために、その差（スプレッドとして知られています）を測定しています。この種の指標は、アルファ値を高めるために取引戦略をどのように編成するかに影響を与えます。

まず、組み込みのウィンドウ関数 last を使って、時間順に並べた後の最後の非 NULL 引用符の値を探してみましょう。

```
# sample code inside join method

#define partitioning keys for window
partition_spec = Window.partitionBy('symbol')

# define sort - the ind_cd is a sort key (quotes before trades)
join_spec = partition_spec.orderBy('event_ts'). \
                    rowsBetween(Window.unboundedPreceding, Window.currentRow)

# use the last_value functionality to get the latest effective record
select(last("bid", True).over(join_spec).alias("latest_bid"))
```

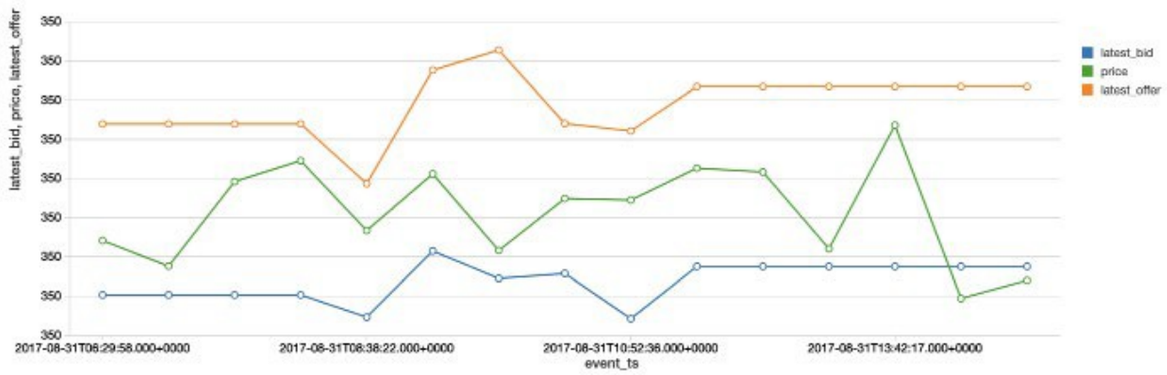
データをマージして引用符を添付するためにカスタムジョインを呼び出します。フルコードは添付の Notebook を参照してください。

```
# apply our custom join
mkt_hrs_trades = trades.filter(col("symbol") == "K")
mkt_hrs_trades_ts = base_ts(mkt_hrs_trades)
quotes_ts = quotes.filter(col("symbol") == "K")

display(mkt_hrs_trades_ts.join(quotes_ts))
```

```
1 display(mkt_hrs_trades_ts.join(quotes_ts))
```

▸ (5) Spark Jobs

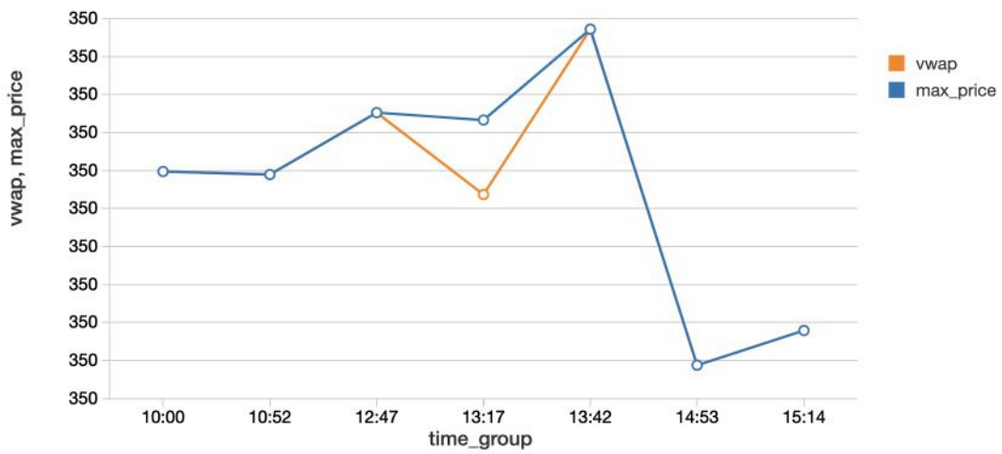
event_ts	price	symbol	ind_cd	latest_bid	latest_offer
2017-08-31T06:29:58.000+0000	347.4121586706382	K	1	346.0297772384752	350.39315623662594
2017-08-31T06:32:30.000+0000	346.7582132240916	K	1	346.0297772384752	350.39315623662594
2017-08-31T06:37:31.000+0000	348.919146315238	K	1	346.0297772384752	350.39315623662594
2017-08-31T06:56:24.000+0000	349.45235333868743	K	1	346.0297772384752	350.39315623662594
2017-08-31T08:52:59.000+0000	349.11648025163987	K	1	347.1462324487709	351.7600135730971
2017-08-31T09:22:55.000+0000	347.16036576622395	K	1	346.448634562581 96	352.27114879065283
2017-08-31T10:00:54.000+0000	348.4869310969907	K	1	346.5728444681869	350.40101772579453
2017-08-31T10:52:36.000+0000	348.44707325529976	K	1	345.42173015910055	350.21440300934785
2017-08-31T08:38:22.000+0000	347.6687817715506	K	1	345.46234681384203	348.86794603671 36



トレードパターンに対してVWAP をマークする

上記ではマージ手法を示しましたが、ここでは標準的な集計、すなわち出来高加重平均価格（VWAP）に焦点を当ててみましょう。この指標は、一日を通しての証券のトレンドと価値の指標となります。ラッパークラス内の VWAP 関数（添付の Notebook）は、VWAP が証券の取引価格を上回ったり下回ったりする場所を示します。特に、VWAP（オレンジ色）が取引価格を下回るウィンドウを特定することができ、その銘柄が買われすぎていることを示しています。

```
trade_ts = base_ts(trades.select('event_ts', symbol, 'price', lit(100).
alias("volume")))
vwap_df = trade_ts.vwap(frequency = 'm')

display(vwap_df.filter(col(symbol) == "K") \
    .filter(col('time_group').between('09:30','16:00')) \
    .orderBy('time_group'))
```



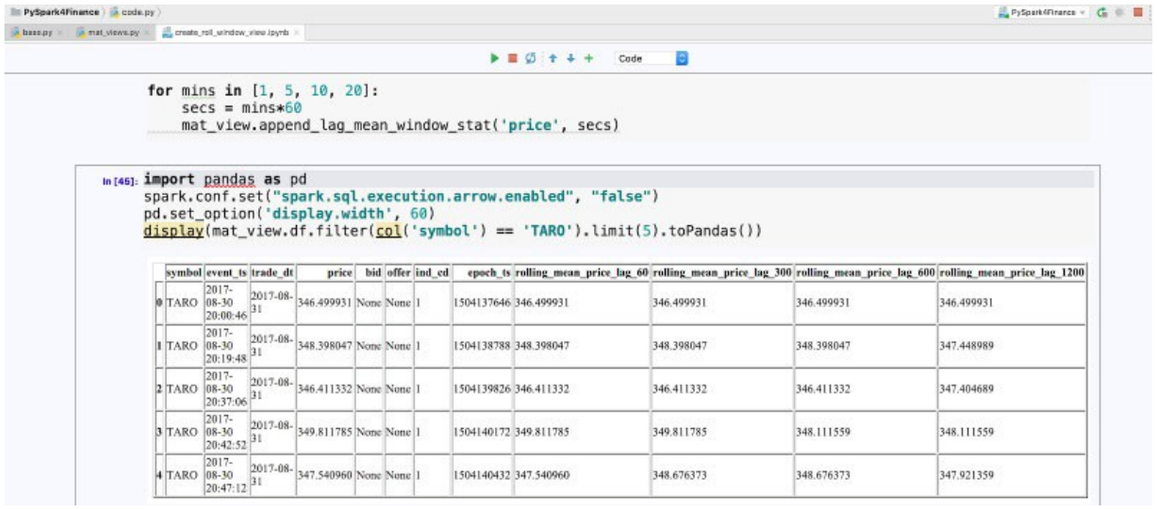
Databricks Connect を使用した反復開発の高速化

ここまでのところ、単発の時系列メトリクスのための基本的なラッパーをいくつか作成してきました。しかし、コードの本番化にはモジュール化とテストが必要で、これは IDE で行うのがベストです。今年、私たちは Databricks Connect を導入しました。これにより、ローカル IDE での開発が可能になり、ライブのデータブリックスクラスタに対するテストの経験が向上しました。Databricks Connect の財務分析におけるメリットとしては、小規模なテストデータに時系列の機能を追加することができ、機能を検証するために何年もの過去のティックデータに対してインタラクティブな Spark クエリを実行できる柔軟性が追加されたことが挙げられます。

PyCharm を使って、リッチな時系列フィーチャーセットを生成するための PySpark の機能をラップするために必要なクラスを整理しています。この IDE は、コードの補完、フォーマットの標準化、コードを実行する前にクラスやメソッドを素早くテストするための環境を提供してくれます。

```
from pyspark.sql.window import Window

import pandas as pd

from base import base_ts

class enrich_ts:

    def __init__(self, df):
        self.df = df.withColumn("epoch_ts", df.event_ts.cast("long"))

    # define custom data frame join which can scale to billions of quotes - this does not need to perform a full
  inner join but rather a UNION/SORT
    def append_lag_mean_window_stat(self, input_col, lag_nb):
        """
        :param other: Right side of the dataset being evaluted
        """
        windowSpec = Window.partitionBy('symbol') \
            .orderBy('epoch_ts') \
            .rangeBetween(-1*lag_nb, 0)

        self.df = self.df.withColumn("rolling_mean_" + input_col + "_lag_" + str(lag_nb), fn.mean(input_col).over(
  windowSpec))

    def append_prvlg_fields(self, other):
        left = base_ts(self.df)
```

ローカルクラスをロードし、スケーラブルなインフラストラクチャでインタラクティブなクエリを実行する Jupyter Notebook を使って、ラップトップから直接Spark コードを実行しています。コンソールペインには、ライブクラスタに対してジョブが実行されている様子が表示されています。



```
for mins in [1, 5, 10, 20]:
    secs = mins*60
    mat_view.append_lag_mean_window_stat('price', secs)
```

```
In [46]: import pandas as pd
spark.conf.set("spark.sql.execution.arrow.enabled", "false")
pd.set_option('display.width', 60)
display(mat_view.df.filter(col('symbol') == 'TARO').limit(5).toPandas())
```

	symbol	event_ts	trade_dt	price	bid	offer	ind_cd	epoch_ts	rolling_mean_price_lag_60	rolling_mean_price_lag_300	rolling_mean_price_lag_600	rolling_mean_price_lag_1200
0	TARO	2017-08-30 20:00:46	2017-08-31	346.499931	None	None	1	1504137646	346.499931	346.499931	346.499931	346.499931
1	TARO	2017-08-30 20:19:48	2017-08-31	348.398047	None	None	1	1504138788	348.398047	348.398047	348.398047	347.448989
2	TARO	2017-08-30 20:37:06	2017-08-31	346.411332	None	None	1	1504139826	346.411332	346.411332	346.411332	347.404689
3	TARO	2017-08-30 20:42:52	2017-08-31	349.811785	None	None	1	1504140172	349.811785	349.811785	348.111559	348.111559
4	TARO	2017-08-30 20:47:12	2017-08-31	347.540960	None	None	1	1504140432	347.540960	348.676373	348.676373	347.921359

最後に、ローカル IDE を使用して、同時に最大の時系列データセットのマテリアライズされた時系列ビューに追加することで、両方の利点を得ることができます。

Koalas を市場操作に活用

pandas API は Python でのデータ操作や分析のための標準ツールであり、NumPy、SciPy、Matplotlib などの Python データサイエンスのエコシステムに深く統合されています。Pandas の欠点としては、大量のデータに簡単にスケールできないことが挙げられます。財務データには常に何年分ものヒストリカルデータが含まれており、これはリスク集計やコンプライアンス分析には非常に重要です。

これを簡単にするために、バックエンドで Spark を実行しながら pandas の API を活用する方法として Koalas を導入しました。Koalas の API は pandas にマッチしているので、使いやすさを犠牲にすることはなく、スケーラブルなコードへの移行は1行のコード変更で済みます（次項の Koalas のインポートを参照）。Koalas の金融時系列問題への適合性を紹介する前に、金融詐欺における特定の問題、つまりフロントランニングについてのコンテキストから始めてみましょう。

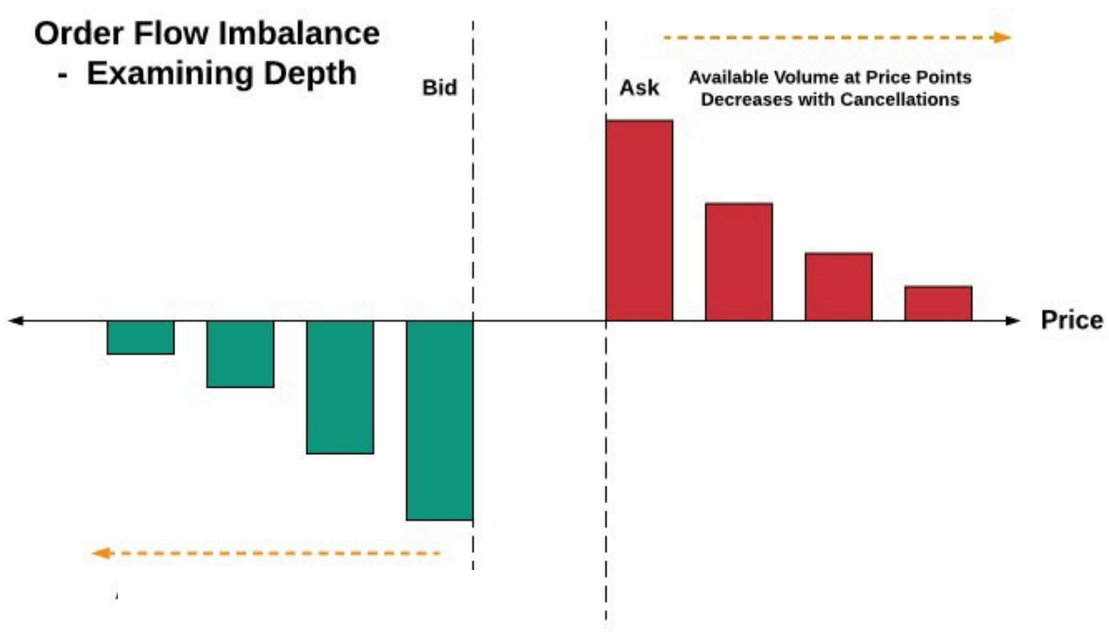
以下のシーケンスが発生するとフロントランニングが発生します。

1. 取引会社は、証券の価格に影響を与える可能性のある非公開情報を認識しています。
2. 会社は、大規模な大量注文（または大規模な総量を合計した注文の大規模なセット）を購入します。
3. 流動性がなくなると、証券価格が上昇します。
4. 会社は投資家に証券を販売し（これは以前の購入から上向きに駆動されている）、投資家は証券が取引された情報が非公開であったにもかかわらず、より大きな価格を支払うことを余儀なくされ、大きな利益を上げています。

出典： CC0 パブリックドメイン画像 https://pxhere.com/en/photo/1531985, https://pxhere.com/en/photo/847099

説明のために、ファーマーズマーケットとアップルパイのビジネスを使った簡単な例がここにあります。この例では、全国のアップルパイ事業に必要なリンゴの需要が迫っていることを知っているランナーのフレディが、その後、全てのファーマーズマーケットでリンゴを購入していることを示しています。これは、実質的には、フレディが前に購入することによって大きな影響を与えたので、フレディは買い手にプレミアムで彼のリンゴを販売することができます。

フロントランニングの検出には、オーダーフローの不均衡を理解する必要があります（下図参照）。特に、オーダーフローの不均衡の異常は、フロントランニングが発生している可能性のあるウィンドウを特定するのに役立ちます。



それでは、市場操作問題を解決しつつ、生産性を向上させるためにKoalasパッケージを使ってみましょう。具体的には、次のようなことに注目して、オーダーフローの不均衡の異常を見つけていきます。

- イベントの重複排除を同時に行う
- 需給の増加を評価するためのラグウィンドウ
- データフレームをマージしてオーダーフローの不均衡を集約する

時系列の重複排除

一般的な時系列データのクレンジングには、インプットと重複排除があります。高頻度のデータ（気配値データなど）では、重複した値が見つかることがあります。シーケンス番号のない時間ごとに複数の値がある場合、後続の統計分析が意味をなすように重複排除する必要があります。以下のケースでは、1回ごとに複数のビッド/アスク株の数量が報告されているので、オーダーの不均衡を計算するためには、1回ごとの最大深度の値を1つの値に頼りたいと思います。

```
import databricks.koalas as ks

kdf_src = ks.read_delta("...")
grouped_kdf = kdf_src.groupby(['event_ts'], as_index=False).max()
grouped_kdf.sort_values(by=['event_ts'])
grouped_kdf.head()
```

	Symbol	Date	Time	bid_pr	ask_pr	bid_shrs_qt	ask_shrs_qt	event_ts
39757	ITUB	03/05/2014	09:30:00.011	13.14	13.23	700.0	200.0	2014-03-05 09:30:00.011
39758	ITUB	03/05/2014	09:30:00.052	13.15	13.23	700.0	200.0	2014-03-05 09:30:00.052
39759	ITUB	03/05/2014	09:30:00.235	13.15	13.22	700.0	100.0	2014-03-05 09:30:00.235
39760	ITUB	03/05/2014	09:30:00.236	13.16	13.22	100.0	100.0	2014-03-05 09:30:00.236
39761	ITUB	03/05/2014	09:30:00.237	13.16	13.21	100.0	700.0	2014-03-05 09:30:00.237

Koalas を使った時系列ウィンドウイング

時系列の重複を排除したので、需要と供給を見つけるために窓を見てみましょう。時系列のウィンドウは、一般的に時間のスライスや間隔を見ることを意味します。ほとんどのトレンド計算（例えば、単純移動平均など）は、計算を行うために時間窓の概念を使用しています。 Koalas は、以下のように shift を使ってウィンドウ内のラグやリードの値を取得するためのシンプルな pandas のインターフェイスを継承しています（Spark のラグ関数に似ています）。

```
grouped_kdf.set_index('event_ts', inplace=True, drop=True)
lag_grouped_kdf = grouped_kdf.shift(periods=1, fill_value=0)

lag_grouped_kdf.head()
```

event_ts	Symbol	Date	Time	bid_pr	ask_pr	bid_shrs_qt	ask_shrs_qt
2014-03-05 09:30:00.011	0	0	0	0	0	0.0	0.0
2014-03-05 09:30:00.052	ITUB	03/05/2014	09:30:00.011	13.14	13.23	700.0	200.0
2014-03-05 09:30:00.235	ITUB	03/05/2014	09:30:00.052	13.15	13.23	700.0	200.0
2014-03-05 09:30:00.236	ITUB	03/05/2014	09:30:00.235	13.15	13.22	700.0	100.0
2014-03-05 09:30:00.237	ITUB	03/05/2014	09:30:00.236	13.16	13.22	100.0	100.0

タイムスタンプでマージし、Koalas の列の演算で不均衡を計算する

遅延値が計算できたので、このデータセットを元の相場の時系列とマージしたいと思います。以下では、Koalas のマージを使って、時間インデックスとのマージを行っています。これにより、需給計算に必要な統合されたビューが得られ、注文の不均衡指標につながります。

```
lagged = grouped_kdf.merge(lag_grouped_kdf, left_index=True, right_
index=True, suffixes=['', '_lag'])
lagged['imblnc_contrib'] = lagged['bid_shrs_qt']*lagged['incr_demand'] \
    - lagged['bid_shrs_qt_lag']*lagged['decr_demand'] \
    - lagged['ask_shrs_qt']*lagged['incr_supply'] \
    + lagged['ask_shrs_qt_lag']*lagged['decr_supply']
```

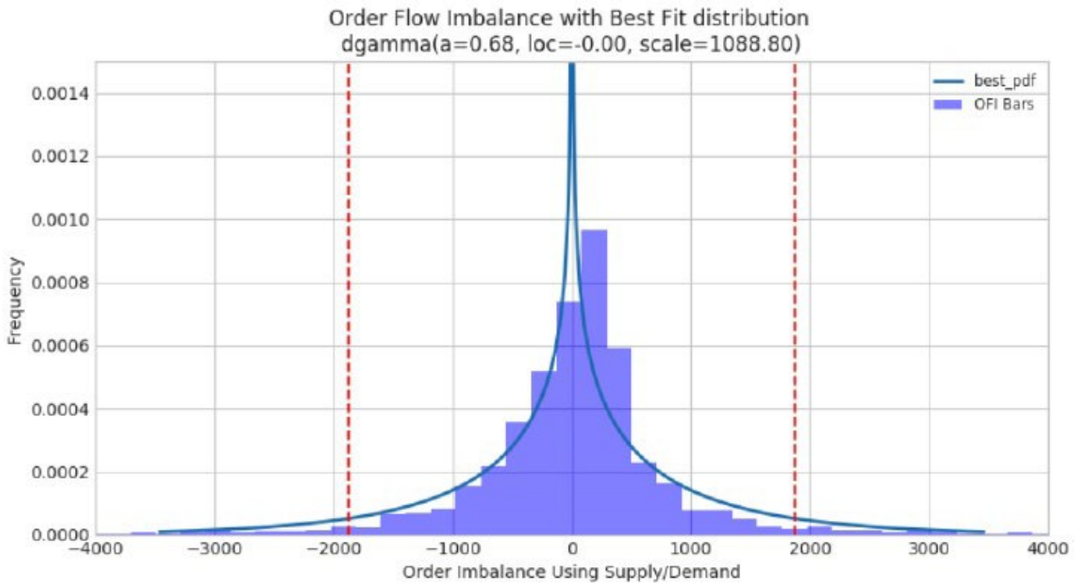
event_ts	Symbol	Time	bid_pr	ask_pr	bid_pr_lag	ask_pr_lag	imblnc_contrib
2014-03-05 09:30:00.011	ITUB	09:30:00.011	13.14	13.23	0	0	500.0
2014-03-05 09:30:00.052	ITUB	09:30:00.052	13.15	13.23	13.14	13.23	0.0
2014-03-05 09:30:00.235	ITUB	09:30:00.235	13.15	13.22	13.15	13.23	100.0
2014-03-05 09:30:00.236	ITUB	09:30:00.236	13.16	13.22	13.15	13.22	-600.0
2014-03-05 09:30:00.237	ITUB	09:30:00.237	13.16	13.21	13.16	13.22	-600.0

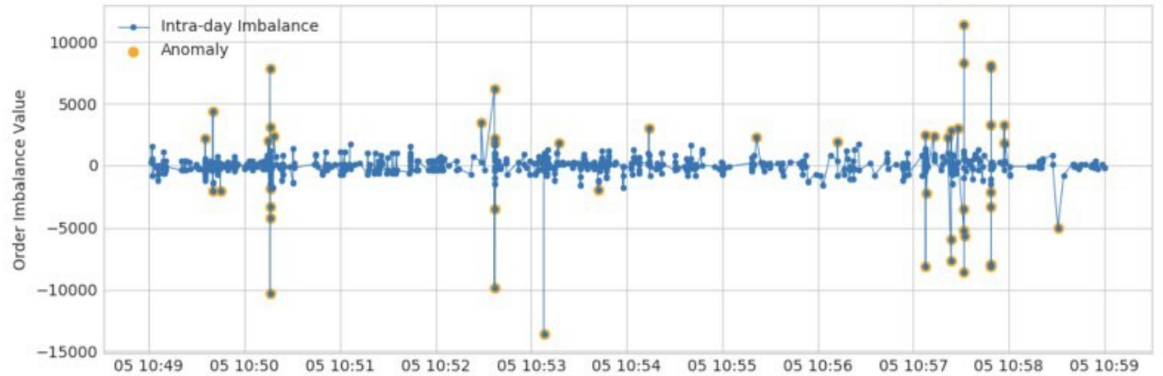
Koalas から NumPy で分布をフィットさせる

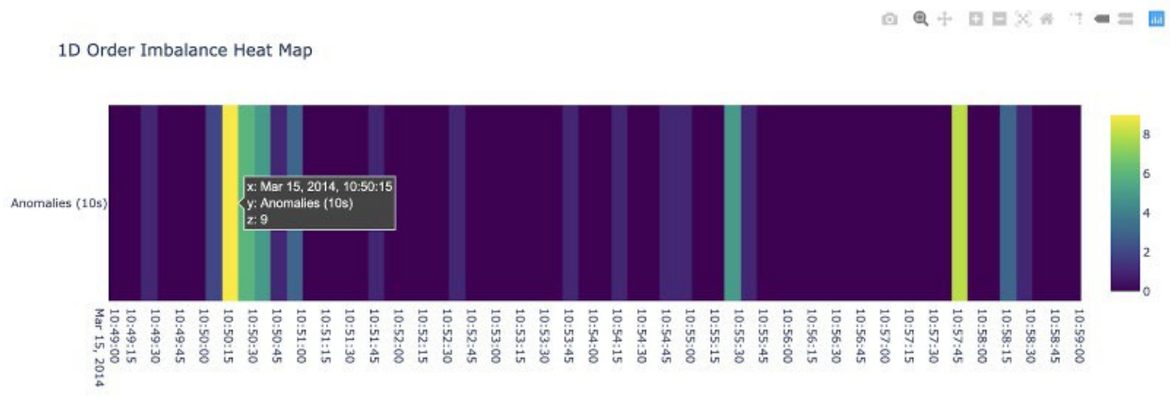
最初の準備が終わったら、Koalas のデータフレームを統計分析に役立つフォーマットに変換します。この問題では、先に進む前に、不均衡を分単位や他の時間単位で集計することができますが、説明のために、我々のティッカー "ITUB" の完全なデータセットに対して実行してみましょう。以下、我々は Koalas 構造体を NumPy データセットに変換し、SciPy ライブラリを使用してオーダーフローの不均衡の異常を検知できるようにしています。to_numpy() 構文を使用するだけで、この分析をブリッジすることができます。

```
from scipy.stats import t
import scipy.stats as st
import numpy as np

q_ofi_values = lagged['imblnc_contrib'].to_numpy()
```

下の図は、オーダー・フローの不均衡の分布を、不均衡の異常が発生したイベントを特定するために、5% と 95% のマーカーとともにプロットしたものです。分布をフィットさせ、このプロットを作成するコードについては、Notebook 全体を参照してください。Koalas/SciPy ワークフローで計算した不均衡の時間は、私たちが探していた市場操作スキームであるフロントランニングの潜在的なインスタンスと相関しています。







下の時系列の可視化では、上記の異常値として抽出された異常値がオレンジ色で強調されています。最後の可視化では、plotly ライブラリを使用して、時間窓と異常値の頻度をヒートマップの形でまとめています。具体的には、10:50:10-10:50:20 の時間枠を、フロントランニングの観点から潜在的な問題領域として特定します。

結論

この記事では、Apache Spark とデータブリックスを時系列分析に活用する方法を、ウィンドウやラッパーを使った直接的な方法と、Koalas を使った間接的な方法の両方で紹介してきました。ほとんどのデータサイエンティストは pandas API に依存しているので、Koalas は Apache Spark のスケールを可能にしながら pandas の機能を利用するのに役立ちます。時系列分析に Spark と Koalas を使うメリットは以下のとおりです。

- リスク、不正、コンプライアンスのユースケースでの時系列分析を、as-of join とシンプルな集計で並列化します。
- Databricks Connect を使用した高速な反復処理とリッチな時系列機能の作成
- データサイエンスやクオンツのチームを Koalas で武装させ、pandas の使いやすさや API を犠牲にすることなく、データ準備をスケールアップさせます。

今すぐデータブリックスのこの Notebook を試してください。金融時系列のユースケースでお客様をどのようにサポートするかについては、こちらからお問い合わせください。

> データブリックスの無料の Notebook を使って実験を始める

第 3 章　Part 1 : 動的タイムワープの概要

動的タイムワープと MLflow を利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

「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。

以下で説明するように、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。

- **FINANCIAL MARKETS：**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。
- **WEARABLE FITNESS TRCKERS：**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。
- **ROUTE CALCULATION：**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なら誰でもこのテクニックに精通しているはずです。完全に整列された時系列比較データは、完全に「整頓された」データと同じように野生で見ることができないことがあるからです。

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

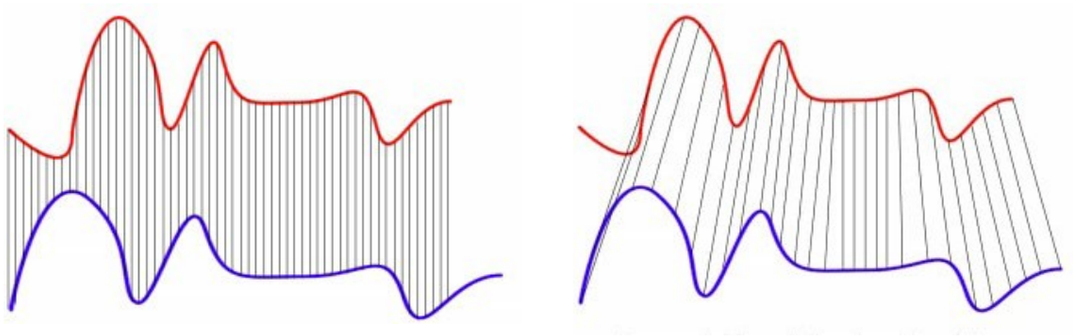
動的タイムワープ

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、よく引用されている論文に Dynamic time warping for isolated word recognition based on ordered graph searching techniques があります。

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]

<audio controls></audio> 0:00 / 0:06

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

<audio controls></audio> 0:00 / 0:08

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

<audio controls></audio> 0:00 / 0:07

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

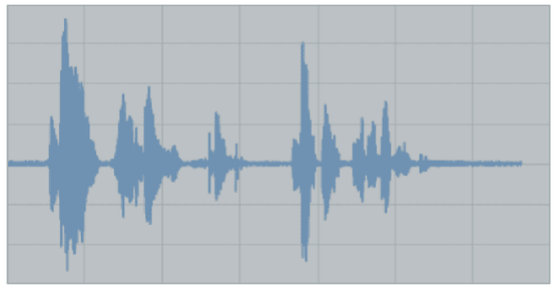
<audio controls></audio> 0:00 / 0:07

出典 : “The Expanse”

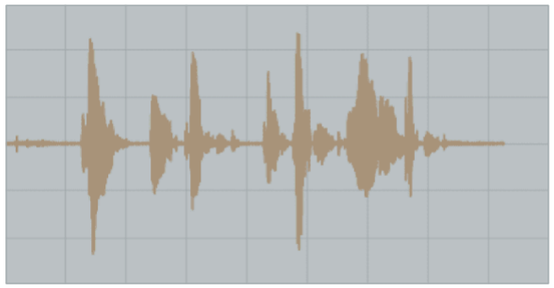
以下は、4 つのオーディオクリップの matplotlib を使用した可視化です。

- 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。

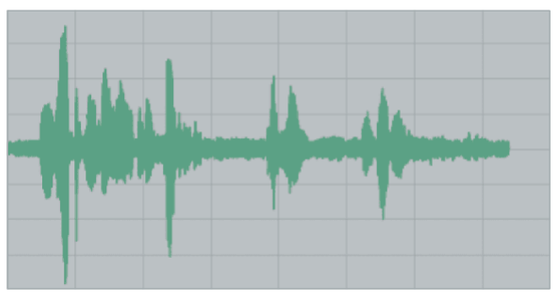
Clip 1 Doors and corners, kid. That’s where they get you. [v1]



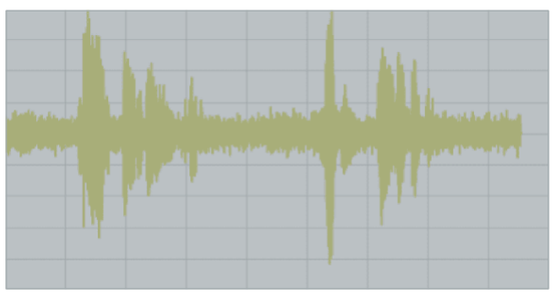
Clip 2 Doors and corners, kid. That’s where they get you. [v2]



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



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

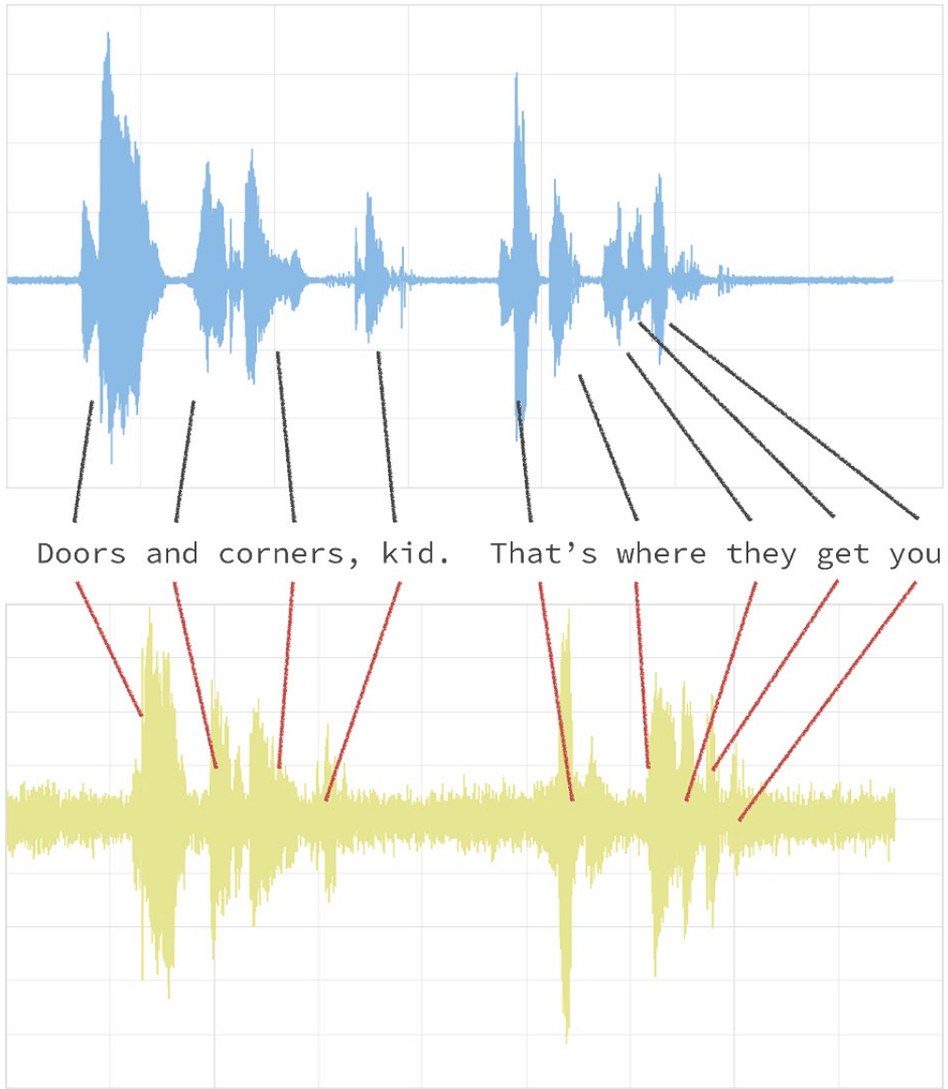


これらのオーディオクリップを読み込み、matplotlib を使って可視化するコードは、以下のコードスニペットにまとめられています。

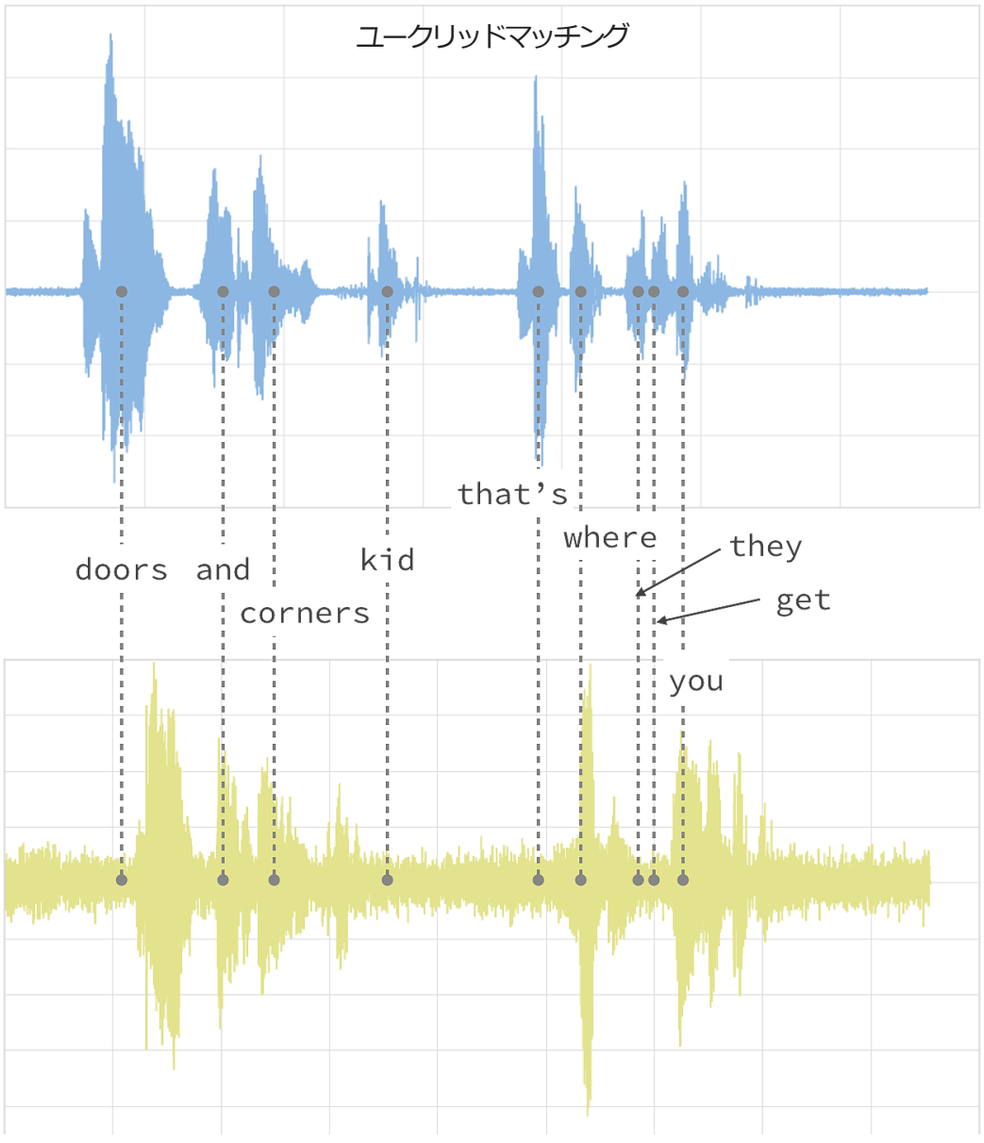
```
from scipy.io import wavfile
from matplotlib import pyplot as plt
from matplotlib.pyplot import figure

# 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」にあります。

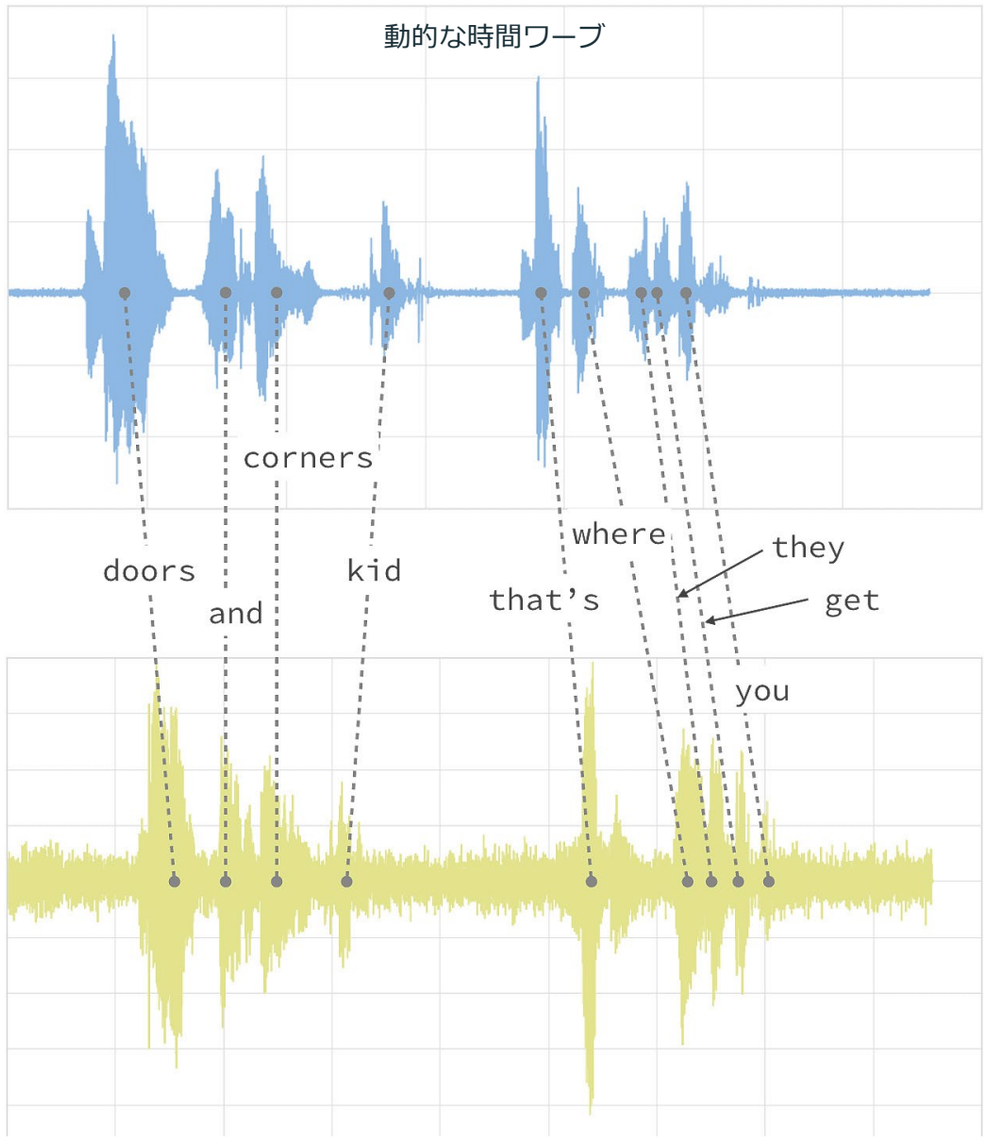
後述するように、2 つのクリップ（この場合はクリップ 1 とクリップ 4）が同じ引用文に対してイントネーション（振幅）とレイテンシーが異なる場合。



伝統的な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。

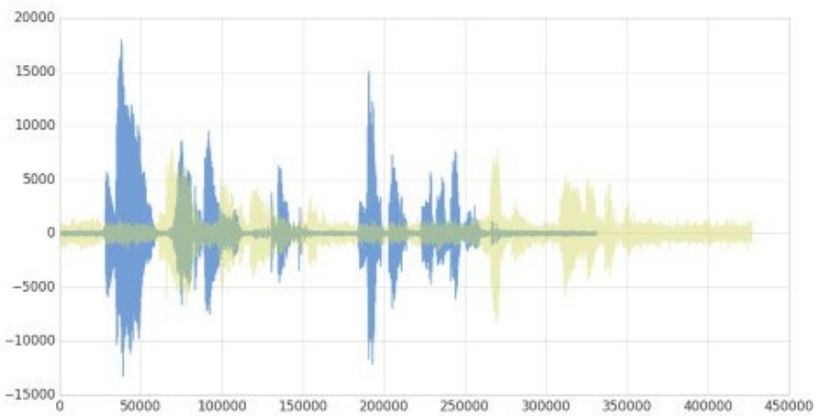


時系列比較には、fastdtw PyPi ライブラリを使用します。データブリックスのワークスペースに PyPi ライブラリをインストールする方法は、Azure | AWS にあります。 fastdtw を使うことで、異なる時系列間の距離を素早く計算することができます。

```
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」にあります。

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



簡単な観察：

- 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。

ご覧のように、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することができます。

次章では

ここまで動的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 3 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 を利用した販売傾向の把握 Part 2

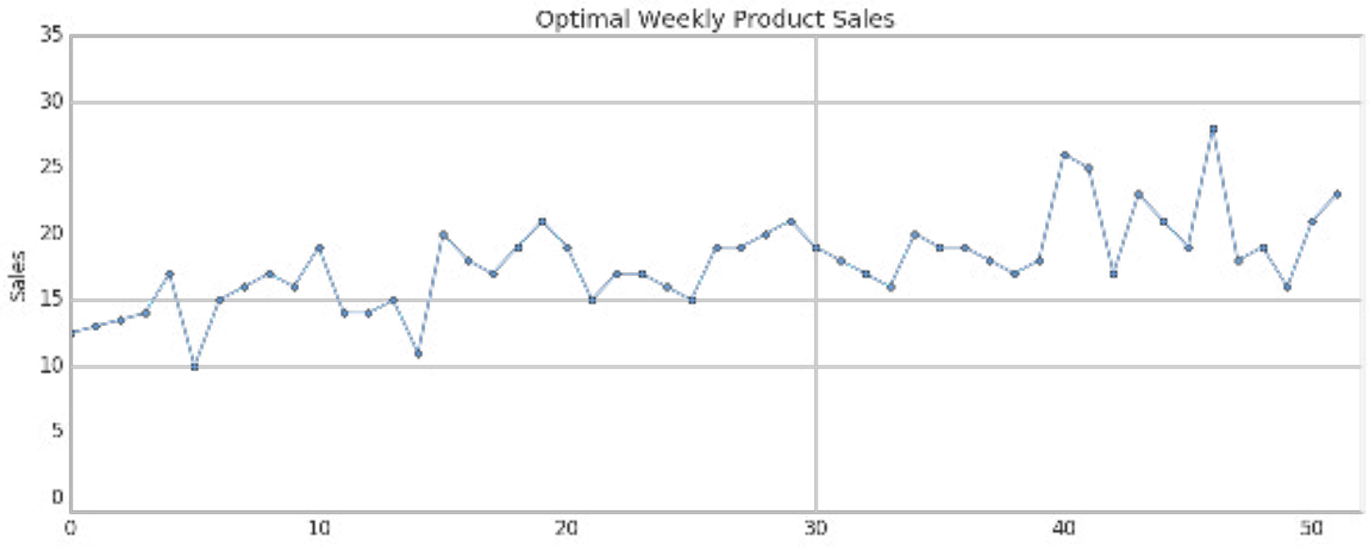
投稿者：
Ricardo Portilla
Brenner Heintz
Denny Lee

2019 年 4 月 30 日

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。



あなたの仕事は、毎週の需要があなたの生産能力をできる限り満たす製品を選択することです。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いますが、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、プリンター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若干ずれてしまうことがあるからです。需要を全て満たすのに十分な生産能力がない週もあるでしょうが、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))
```

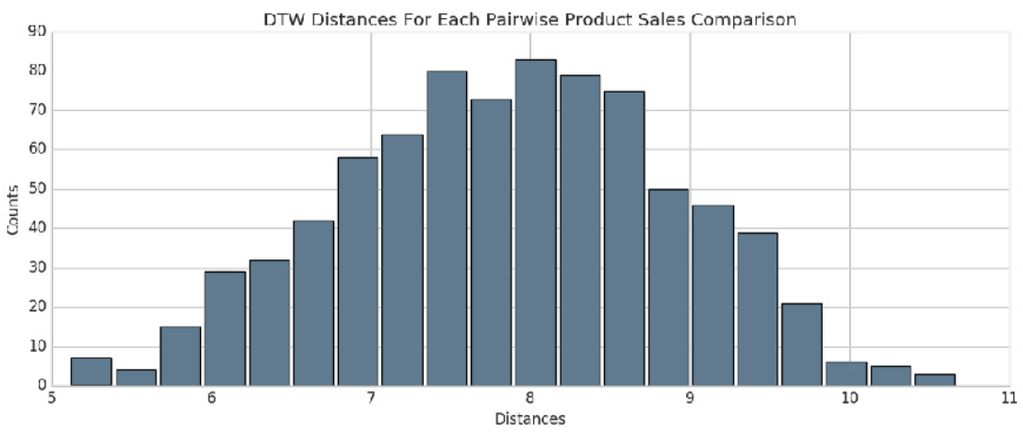
Product_Code	W0	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13
P1	11	12	10	8	13	12	14	21	6	14	11	14	16	9
P2	7	6	3	2	7	1	6	3	3	3	2	2	6	2
P3	7	11	8	9	10	8	7	13	12	6	14	9	4	7
P4	12	8	13	5	9	6	9	13	13	11	8	4	5	4
P5	8	5	13	11	6	7	9	14	9	9	11	18	8	4
P6	3	3	2	7	6	3	8	6	6	3	1	1	5	4
P7	4	8	3	7	8	7	2	3	10	3	5	2	3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

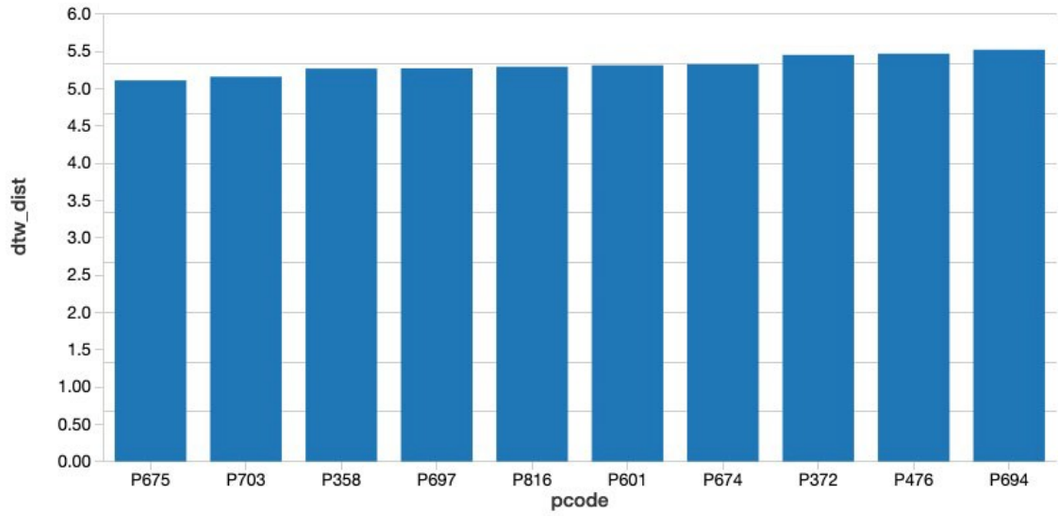
```
# Calculate distance via dynamic time warping between product code and
optimal time series
import numpy as np
import _ucrdtw

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。

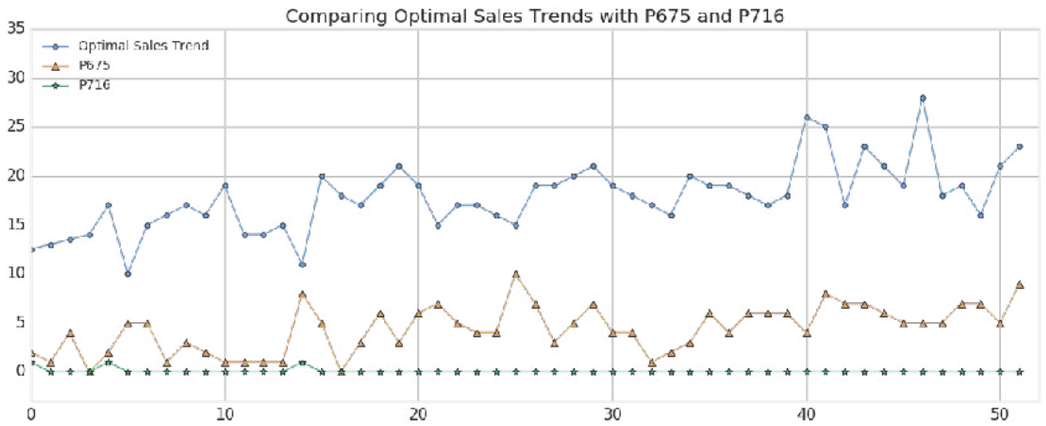


そこから、最適な販売動向に最も近い商品コード（計算された DTW 距離が最も小さいもの）を特定することができます。データブリックスを使っているので、SQL クエリを使って簡単に選択することができます。最も近いものを表示してみましょう。

```
%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。

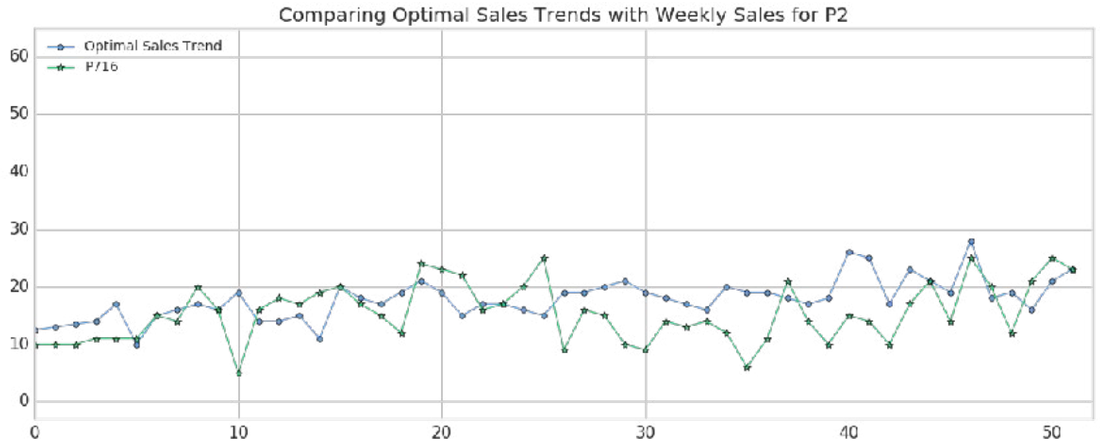


ご覧のように、製品 #675（オレンジ色の三角形）は、週次売上高の絶対値が思ったよりも低いものの、最適な販売トレンドに最もよくマッチしています（これについては後ほど修正します）。この結果は、DTW 距離が最も近い製品には、比較対象としているメトリクスを多少反映したピークとバレーがあると予想されるため、理にかなっています。（もちろん、製品の正確な時間指標は、動的な時間ワープのため、週ごとに異なります。）逆に、製品 #716（緑の星）は、最悪の一致を示す製品であり、ほとんど変動はありません。

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。**データブリックスの 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、パラメー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。

- **PARAMETERS：** モデルへの入力
- **METRICS：** モデルの出力、またはモデルの成功の尺度
- **ARTIFACTS：** 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
- **MODELS：** 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:
    ...
```

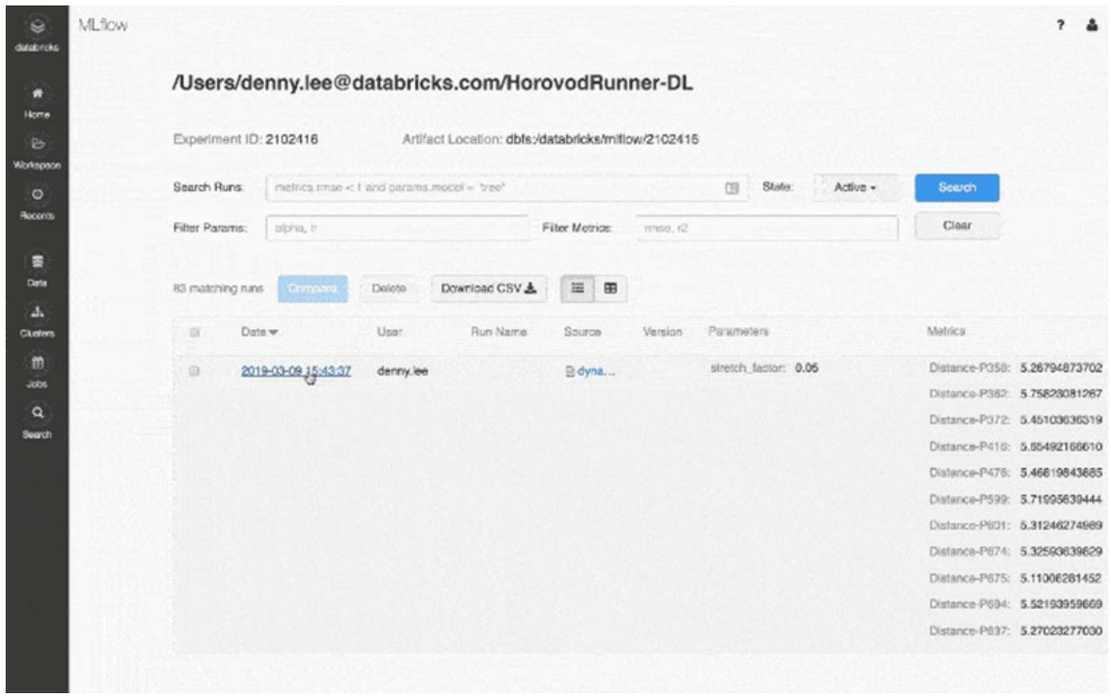
下の省略コードにそれを示します。

```
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、使用されている「ストレッチファクター」パラメータのログと、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）を保存することもできました。**これらの実験実行は、データブリックスにローカルに保存されており、後日実験結果を見ることになってもアクセスできるようになっています。**

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。データブリックス 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、実験パラメータやメトリクス、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。**

- 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。
- 「実行 ID」をコピーします。
- 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。
- 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

ご覧のように、私たちの MLflow モデルは、新しい値や見たことのない値を簡単に予測しています。また、Inference API に準拠しているので、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。

最後に、Databricks Unified Data Analytics Platform を使用して、動的タイムワープを使用して販売傾向を予測する方法を実演しました。今日は、Databricks Runtime for Machine Learning を使用した「売上動向を予測するための動的タイムワープと MLflow の使用」Notebook をお試しください。

> データブリックスの無料の Notebook を使って実験を始める

第 4 章　新しい安全在庫分析による インベントリの最適化

投稿者：
Bryan Smith
Rob Saker

2020 年 4 月 22 日

あるメーカーが顧客の注文を受けて作業をしているときに、重要な部品の納入がサプライヤーによって遅れていることに気がつきました。小売店では、予期せぬ理由でビールの需要が急増し、供給不足のために売上を失うことになります。需要を満たすことができないため、顧客はネガティブな経験をします。このような企業はすぐに収益を失い、あなたの評判は損なわれます。聞き覚えがありませんか？

理想的な世界では、商品の需要は簡単に予測できるでしょう。しかし実際には、最良の予測であっても、予想外の出来事によって影響を受けることがあります。混乱は、原材料の供給、貨物や物流、製造業の故障、予期せぬ需要などによって起こります。小売業者、流通業者、製造業者、サプライヤーは、顧客のニーズを確実に満たしつつ、過剰なインベントリを抱えないようにするために、これらの課題に取り組まなければなりません。そこで、安全在庫分析の改善方法が貴社のビジネスに役立ちます。

組織は、予想される需要を満たすために必要とされるところにリソースを配分することに常に取り組んでいます。当面の焦点は、予測の精度を向上させることにあることが多い。この目標を達成するために、企業はスケーラブルなプラットフォーム、社内の専門知識、洗練された新しいモデルに投資しています。

どんなに優れた予測でも将来を完全に予測できるわけではなく、突然の需要の変化によって棚が埋まったままになることもあります。これは、2020 年初頭に COVID-19 の原因となるウイルスへの懸念から、トイレットペーパーの品切れが広まったことに端を発しています。H-E-B の社長クレイグ・ボヤン氏のコメントによると、"通常 2 か月で販売しているものを 2 週間で販売した"とのことです。

生産量を拡大することは、この問題の単純な解決策ではありません。トイレットペーパーの大手メーカーであるジョージア・パシフィック社は、パンデミックの期間中、人々が家に留まることで、アメリカの平均的な家庭ではトイレットペーパーの消費量が 40% 増える見積もりをしていました。これに対応して、同社はトイレットペーパーの生産用に構成された14の施設で、生産量を 20% 増やすことができました。ほとんどの工場では、すでに 24 時間 365 日固定された生産能力で稼働しているため、これ以上の増産には、追加設備の購入や新工場の建設による生産能力の拡大が必要となります。

このような生産量の増加は、上流側に影響を及ぼす可能性があります。サプライヤーは、新たに規模を拡大して生産能力を拡大した場合に必要とされる資源を供給するのに苦労する可能性があります。トイレットペーパーは単純な製品ですが、その生産は、米国、カナダ、スカンジナビア、ロシアの森林地域から出荷されるパルプと、地元で調達される再生紙繊維に依存しています。初期埋蔵量が枯渇すると、メーカーが必要とする材料を収穫し、加工して出荷するまでに時間がかかります。

ブルウィップ効果と呼ばれるサプライチェーンの概念が、このような不確実性を支えています。サプライチェーン全体の歪んだ情報は、インベントリの大幅な非効率化、運賃や物流コストの増加、不正確な生産能力計画などを引き起こす可能性があります。在庫を正常に戻そうとするメーカーや小売業者は、サプライヤーが生産を再開するきっかけとなり、その結果、上流のサプライヤーが生産を再開するきっかけとなる可能性があります。慎重に管理されていない場合、小売業者やサプライヤーは、需要が通常に戻ったときに過剰インベントリや生産能力を抱えていることに気づくかもしれませんし、消費者が自分の個人的なインベントリのバックログを処理するために、通常よりもわずかに低下していることに気づくかもしれません。

このブルウィップ効果を軽減するためには、我々が予測する需要の不確実性を精査しながら、需要の動向を慎重に検討する必要があります。

安全ストック分析による不確実性の管理

COVID-19 パンデミックを取り巻く消費者需要の変化を予測することは難しいが、サプライチェーンを管理する全ての組織が対処しなければならない不確実性の概念の極端な例を浮き彫りにしている。消費者活動が比較的正常な時期であっても、製品やサービスに対する需要は変化し、それを考慮して積極的に管理しなければなりません。

最新の需要予測ツールは、週次や年次の季節性、長期トレンド、休日やイベント、天候、プロモーション、経済、その他の要因などの外部要因の影響を考慮して、需要の平均値を予測します。これらのツールは、予測された需要の単数値を作成しますが、半分はこの値を下回り、残りの半分はこの値を上回ると予測しているため、誤解を招く可能性があります。

平均予測値を理解することは重要であるが、それと同様に重要なのは、その両側の不確実性を理解することである。この不確実性は、潜在的な需要値の範囲を提供していると考えることができます。このように予測を考えることで、この範囲のどの部分に対処すべきかという会話を始めることができます。

統計的に言えば、潜在的な需要の全範囲は無限であり、したがって、100% 完全に対応できることはありません。しかし、理論的な対話をする必要があるずっと前に、潜在的な需要の範囲に対応する能力を少しずつ向上させると、インベントリ要件がかなりの（実際には指数関数的な）増加を伴うことを認識することができます。これにより、私たちは、潜在的な需要の全範囲の特定の割合に対処しようとする、組織の収益目標とインベントリのコストのバランスをとる、ターゲットを絞ったサービスレベルを追求することになります。

このサービスレベルの期待値を定義した結果、不確実性に対する緩衝材としての役割を果たすために、平均予測需要に対応するために必要な量以上の一定量の余剰在庫を保有しなければならないことになります。この安全在庫は、平均周期的な需要に対応するために必要なサイクル在庫に加えて、組織全体の目標のバランスを取りながら、実際の需要のほとんど（全てではないが）の変動に対応する能力を与えてくれます。

必要な安全在庫レベルの計算

サプライチェーンの古典的な文献では、安全在庫は需要の不確実性と納期の不確実性に対処する2つの式のうちの1つを使用して計算される。この記事では需要の不確実性に焦点を当てているので、不確実なリードタイムの考慮を排除することができ、考慮すべき単一の単純化された安全在庫の公式を残すことができます。

$$安全在庫 = Z * \sqrt{PC/T} * \sigma_D$$

一言で言えば、この式は、安全在庫が平均予測値（σD）の周りの需要の平均的な不確実性に、ストックしている（パフォーマンス）サイクルの期間（√PC/T）の平方根を乗じたものに、対応したい不確実性の範囲（Z）に関連する値を乗じたものとして計算されることを説明しています。この式の各構成要素は、完全に理解していただくために少し説明する必要があります。

前回は、需要が平均値付近の潜在的な値の範囲として存在していることを説明しましたが、今回の予測ではそれが発生しています。もしこの範囲がこの平均値の周りに均等に分布していると仮定すると、平均値の両側にあるこの範囲の平均値を計算することができます。これは標準偏差として知られています。需要の標準偏差としても知られる値 Σd は、平均値の周辺の値の範囲の尺度を提供してくれます。

この範囲は平均値を中心にバランスが取れていると仮定しているので、この範囲内の値のうち、その平均値から標準偏差がいくつか存在する割合を導き出すことができることがわかります。我々が対応したい潜在的な需要の割合を表すためにサービスレベルの期待値を使用する場合、我々は安全在庫の計画の一部として考慮する必要がある需要の標準偏差の数に戻ることができます。値の範囲のパーセンテージを捕捉するために必要な標準偏差の必要数（Z の式で表される z スコアとして知られています）の計算の背後にある実際の計算は少し複雑になりますが、幸いにも z スコア表は広く公開されており、オンライン計算機も利用できます。とはいえ、ここでは、一般的に採用されているサービスレベルの期待値に対応するいくつかの z スコア値を紹介します。

サービスレベル期待値	Z（z スコア）
80.00%	0.8416
85.00%	1.0364
90.00%	1.2816
95.00%	1.6449
97.00%	1.8808
98.00%	2.0537
99.00%	2.3263
99.90%	3.0902
99.99%	3.7190

最後に、安全在庫（√PC/T）を計算しているサイクルの期間を表す項にたどり着きます。平方根計算が必要な理由はさておき、これが式の最も単純な要素であることを理解してください。PC/T 値は、安全在庫を計算しているサイクルの期間を表しています。T による除算は、標準偏差値の計算に使用される単位と同じ単位でこの期間を表現する必要があることを思い出させてくれるものです。例えば、7日サイクルの安全在庫を計画している場合、日次需要値を利用して需要の標準偏差を計算している限り、この期間は7の平方根を取ることができます。

需要のばらつきを見積もるのは難しい

表面的には、安全在庫分析の必要条件の計算は非常に簡単です。サプライチェーンマネジメントの授業では、学生はしばしば需要の過去の値を提供され、そこから計算式の標準偏差成分を計算することができます。サービスレベルの期待値が与えられると、すぐにzスコアを導き出し、その目標レベルを満たすための安全在庫の要件をまとめることができます。しかし、これらの数字は間違っているか、少なくとも、ほとんど有効ではないという重大な仮定の外では間違っています。

安全在庫の計算で問題となるのは、需要の標準偏差である。標準式は、我々が計画している将来の期間の需要に関連した変動を知ることに依存しています。時系列の変動が安定していることは極めて稀です。その代わりに、それはしばしばデータのトレンドや季節的なパターンで変化します。イベントや外部回帰因子も同様に独自の影響を及ぼします。

この問題を克服するために、サプライチェーン・ソフトウェア・パッケージはしばしば、需要の標準偏差の代わりに二乗平均誤差（RMSE）や平均絶対誤差（MAE）のような予測誤差の尺度を代用しますが、これらの値は異なる（関連する概念ではありますが）概念を表しています。これはしばしば安全在庫の要件を過小評価することにつながりますが、このチャートでは 95% の期待値を設定したにもかかわらず、92.7% のサービスレベルが達成されていることが示されています。

また、ほとんどの予測モデルは、予測平均値を計算しながら誤差を最小化するように機能しているため、皮肉なことに、モデルのパフォーマンスの向上が過小評価の問題を悪化させてしまうことがよくあります。多くの小売業者が公表されているサービスレベルの期待値に向かって努力しているにもかかわらず、そのほとんどがその目標を達成できていないという認識の広まりは、このことが背景にあると考えられます。

今後の展望およびデータブリックスができる支援

問題に対処するための重要な第一歩は、安全性ストック分析計算の欠点を認識することです。認識だけでは満足できることはほとんどありません。

何人かの研究者が、安全性ストック推定を改善するという明確な目的のために、需要分散をより良く推定する技術を定義するために取り組んでいますが、これをどのように行うべきかについてはコンセンサスが得られていません。また、これらの技術をより簡単に実装できるようにするためのソフトウェアは、広く利用できるものではありません。

今のところは、サプライチェーン管理者の皆様には、過去のサービスレベルの実績を慎重に検証し、目標が達成されているかどうかを確認することを強くお勧めしたいと思います。これには、過去の実績と同様に過去の予測を慎重に組み合わせることが必要である。従来のデータベース・プラットフォームにデータを保存するにはコストがかかるため、多くの組織では過去の予測や原子レベルのソース・データを保存していませんが、データブリックスのようなプラットフォームを通じて提供されるオンデマンド計算技術を利用してアクセスできる高性能で圧縮された形式で保存されたデータを持つクラウドベースのストレージを使用することで、コスト効率が向上し、多くの組織でクエリ・パフォーマンスの向上を実現することができます。

自動化またはデジタル化されたフルフィルメントシステムが導入され、多くの購入オンラインピックアップインストア（BOPIS）モデルに必要とされるようになり、注文フルフィルメントに関するリアルタイムのデータを生成し始めると、企業はこのデータを使用して在庫切れの問題を検出し、期待されるサービスレベルや店舗内のインベントリ管理方法を再評価する必要性を示すことを望むようになります。これらの分析の実行に制限されていた製造業者は、以下のような分析を行っています。

毎日のルーチンでは、シフトごとに分析して調整したい場合があります。データブリックスのストリーミングインジェスト機能はソリューションを提供し、企業はほぼリアルタイムに近いデータで安全性の高いストック分析を行うことができます。

最後に、インベントリ計画プロセスへのより良いインプットを提供する予測を生成する新しい方法を検討してみてください。Facebook Prophet と並列化とデータブリックスのようなオートスケーリング・プラットフォームを組み合わせることで、多くの企業でタイムリーできめ細かな予測を実現しています。また、一般化自己回帰的条件付きヘテロスケマティック（GARCH）モデルのような他の予測技術では、安全在庫戦略を設計する上で非常に有益であることを証明する需要変動のシフトを調べることができるかもしれません。

セーフティストックの課題を解決することは、その旅に出ようとする組織にとって大きな潜在的利益をもたらしますが、最終状態への道筋が容易に定義されていないため、柔軟性が成功への鍵となるでしょう。私たちは次のことを信じています。

データブリックスは、この旅のための手段として独自の位置を占めており、お客様と一緒にナビゲートしていくことを楽しみにしています。

データブリックスは、この重要なトピックについての洞察を提供してくれた Southern Methodist University Cox School of Business の Sreekumar Bhaskaran 教授に感謝しています。

> データブリックスの無料の **Notebook** を使って実験を始める

第 5 章　サプライチェーン需要予測を改善する新しい手法

因果関係を考慮したきめ細かな需要予測

投稿者：
Bryan Smith
Rob Saker

2020 年 3 月 26 日

組織は急速に微細な需要予測を取り入れている

小売業者や消費財メーカーは、コストを削減し、運転資金を自由にし、オムニチャネル・イノベーションの基盤を作るために、サプライチェーン・マネジメントの改善を求める声が高まっています。消費者の購買行動の変化は、サプライチェーンに新たな負担をかけています。製品やサービスに対する需要は、労働、インベントリ管理、供給・生産計画、貨物・物流、その他多くの分野の意思決定に影響を与えるため、需要予測を通じた消費者需要の理解を深めることは、これらの取り組みのほとんどの出発点と考えられています。

マッキンゼー・アンド・カンパニーは、「Notes from the AI Frontier」の中で、小売業のサプライチェーン予測の精度を 10〜20% 向上させることで、インベントリコストを5% 削減し、収益を 2〜3% 増加させる可能性があると強調しています。従来のサプライチェーン予測ツールでは、望ましい結果は得られませんでした。小売業者のサプライチェーンの需要予測では、業界平均で 32% の不正確さがあると言われており、ほとんどの小売業者にとって、わずかな予測精度の改善でさえも潜在的な影響は計り知れません。その結果、多くの企業はパッケージ化された予測ソリューションから離れ、需要予測スキルを社内に持ち込む方法を模索し、計算効率のために予測精度を低下させていた過去の慣行を見直すようになっています。

これらの取り組みの主な焦点は、時間的および（場所や製品の）階層的な粒度のより細かいレベルでの予測を生成することである。細かい粒度の需要予測は、需要に影響を与えるパターンを、需要が満たされなければならないレベルに近いところで捉えることができる可能性を持っている。これまでは、小売業者は市場レベルや流通レベルである商品クラスの短期的な需要を1か月や1週間の期間で予測し、その予測値を使ってそのクラスの特定の商品を特定の店舗や日にどのように配置すべきかを配分していたかもしれませんが、細かい粒度の需要予測では、予測者は特定の場所での特定の商品のダイナミクスを反映したより局所的なモデルを構築することが可能になります。

細かい需要予測には課題がつきもの

細かい需要予測はワクワクするように聞こえるが、それには多くの課題がある。第一に、集約的な予測から離れることで、生成しなければならない予測モデルと予測の数が爆発的に増加します。必要とされる処理レベルは、既存の予測ツールでは達成できないか、あるいは情報を有用に利用するためのサービスウィンドウを大幅に超えてしまう。この制限により、企業は処理されるカテゴリーの数や分析の粒度をトレードオフにしてしまうことになります。

以前のブログ記事で検討したように、この課題を克服するためにApache Sparkを採用することで、モデラーが作業を並列化してタイムリーに効率的に実行できるようになります。データブリックスのようなクラウドネイティブのプラットフォームにデプロイすると、計算リソースを迅速に割り当ててからリリースすることができ、この作業のコストを予算内に抑えることができます。

第二に、克服するのがより困難な課題は、より細かい粒度でデータを調査した場合に、集合体として存在する需要パターンが存在しない可能性があることを理解することである。アリストテレスの言葉を借りれば、全体が総和よりも大きいことがよくあります。その部分の分析の詳細レベルが低くなるにつれて、より高いレベルの粒度でより簡単にモデル化されたパターンはもはや確実に存在しない可能性があり、より高いレベルで適用可能な技術を用いた予測の生成はより困難になります。予測の文脈におけるこの問題は、1950年代の Henri Theil にまで遡る多くの専門家によって指摘されています。

取引レベルの粒度に近づくにつれ、個々の顧客の需要や購入に影響を与える外部要因も考慮する必要があります。意思決定を行うためには、これらを直接組み入れる必要があるかもしれません。全体としては、これらは時系列を構成する平均値、トレンド、季節性に反映されるかもしれませんが、粒度の細かいレベルでは、これらを予測モデルに直接組み込む必要があるかもしれません。

最後に、粒度を細かくすると、データの構造が従来の予測手法を使用できなくなる可能性が高くなります。トランザクションの粒度に近づけば近づくほど、データの非活動期間に対処する必要がある可能性が高くなります。この粒度のレベルでは、従属変数は、特に販売台数などのカウントデータを扱う場合、単純な変換ができない歪んだ分布になる可能性があり、多くのデータサイエンティストが快適に過ごせる範囲外の予測技術の使用が必要になるかもしれません。

履歴データへのアクセス

詳しくはデータ作成 Notebook をご覧ください。

これらの課題を検証するために、ニューヨーク市の自転車シェアプログラム（別名 Citi Bike NYC）の一般の旅行履歴データを活用します。Citi Bike NYC は、人々を助けることを約束する会社です,「バイクのロックを解除します.ニューヨークをアンロックする。」。このサービスでは、ニューヨーク市内にある 850 以上のレンタル拠点で自転車をレンタルすることができます。同社には13,000 台以上のバイクのインベントリがあり、今後は 40,000 台まで増やす予定です。Citi Bike には 10 万人以上の加入者がおり、1日に約 1万 4000 台の自転車をレンタルしています。

Citi Bike NYC は、自転車を置いていた場所から将来の需要を予測した場所に再配置しています。Citi Bike NYC は、小売店や消費財企業が日常的に直面している課題と似たようなものを抱えており、需要を予測して適切な場所にリソースを配分するにはどうすればよいのでしょうか？需要を過小評価してしまうと、収益機会を逃し、顧客心理に悪影響を及ぼす可能性があります。需要を過大に見積もってしまうと、自転車の余剰インベントリが未使用のままになってしまいます。

この公開されているデータセットは、前月末から 2013 年半ばのプログラム開始までの各自転車レンタルの情報を提供しています。走行履歴データは、特定のレンタル・ステーションから自転車がレンタルされた正確な時間と、その自転車が別のレンタル・ステーションに返却された時間を特定します。Citi Bike NYC プログラムのステーションを店舗の場所として扱い、レンタルの開始を取引と考えると、長くて詳細な取引に近いものが得られます。予測を立てることができる歴史を持っています。

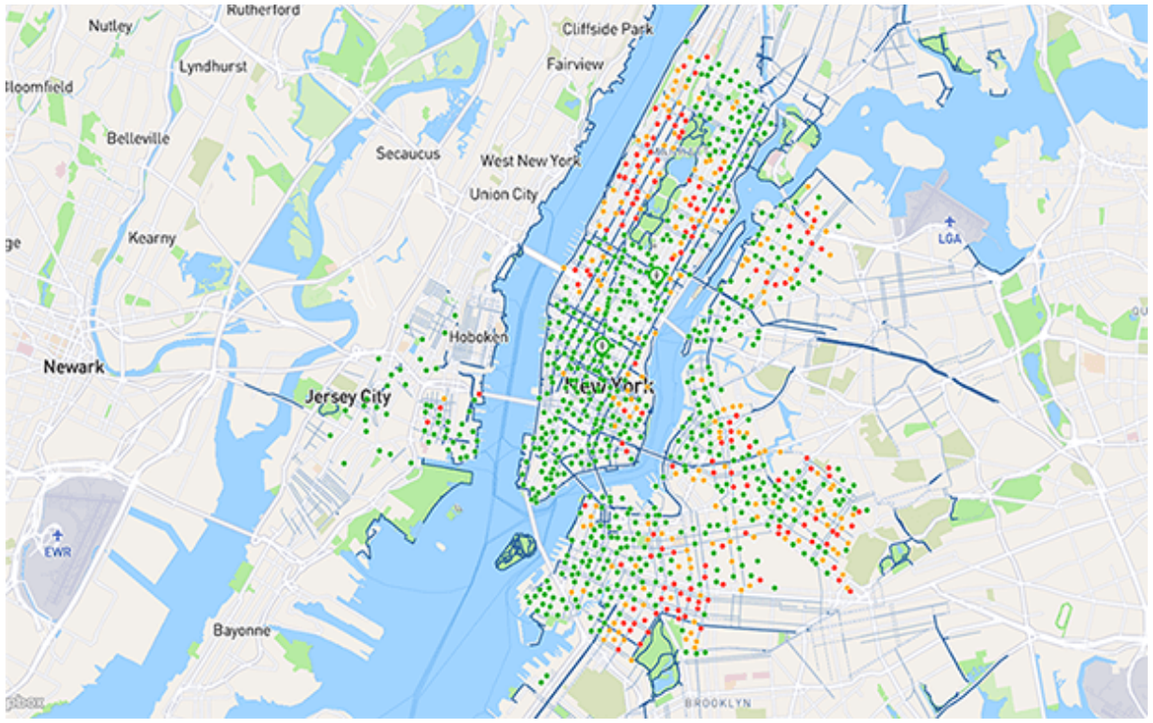
この演習の一環として、モデル化の取り組みに組み込む外部要因を特定する必要があります。外部要因として、休日のイベントと過去の（そして予測された）気象データの両方を活用します。休日データセットについては、Python の holidays ライブラリを使用して、2013 年から現在までの標準的な休日を単純に特定します。気象データには、人気の高い気象データアグリゲータ Visual Crossing の1時間ごとの抽出データを採用します。

Citi Bike NYC と Visual Crossing のデータセットには、当社が直接データを共有することを禁止する利用規約があります。弊社の結果を再現したい方は、データ提供者のウェブサイトにアクセスし、その利用規約を確認した上で、適切な方法でデータセットをそれぞれの環境にダウンロードしてください。当社は、これらの生データ資産を当社の分析に使用されるデータオブジェクトに変換するために必要なデータ準備ロジックを提供します。

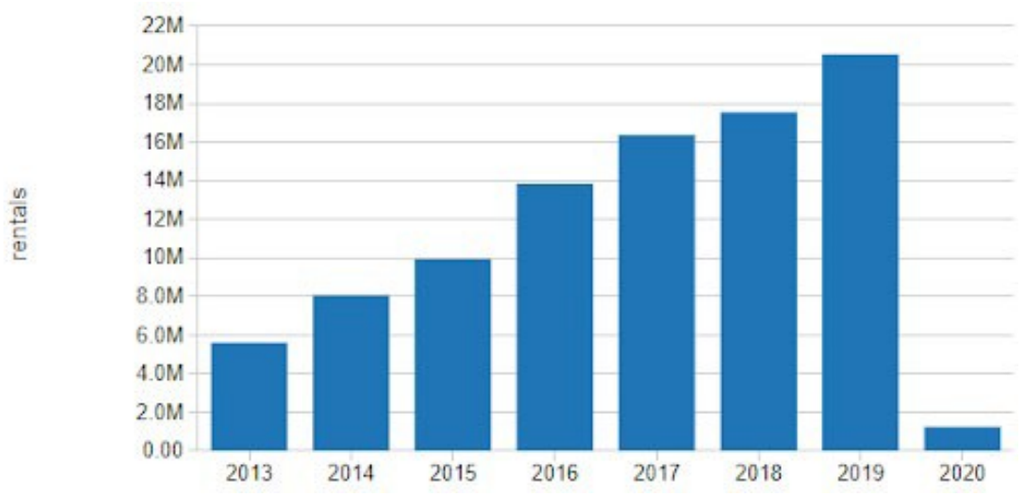
トランザクションデータの検討

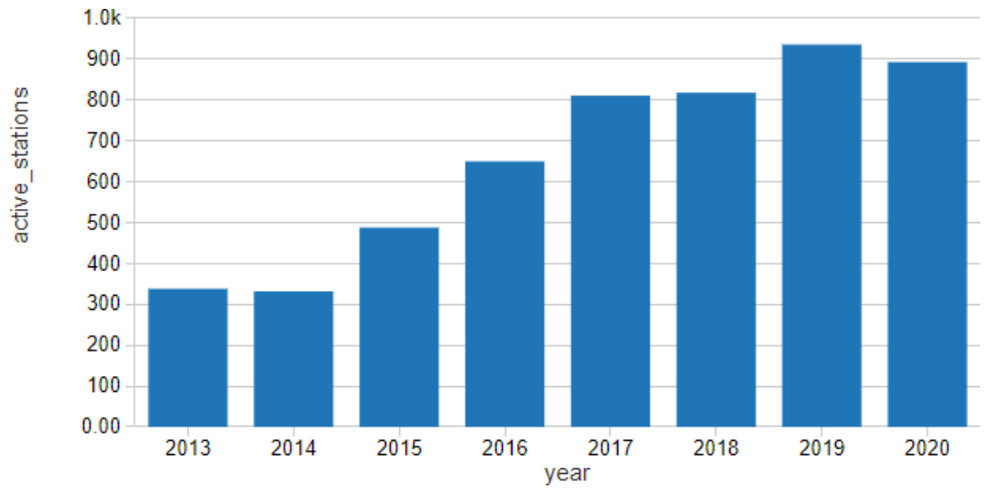
詳しくは探索分析 Notebook をご覧ください。

2020 年 1 月現在、Citi Bike NYC のバイクシェアプログラムは、マンハッタンを中心としたニューヨーク都市圏で稼働している 864 のアクティブステーションで構成されています。2019 年だけでも、400 万人強の顧客がユニークなレンタルを開始し、ピーク時には 14,000 件近くのレンタルが行われました。

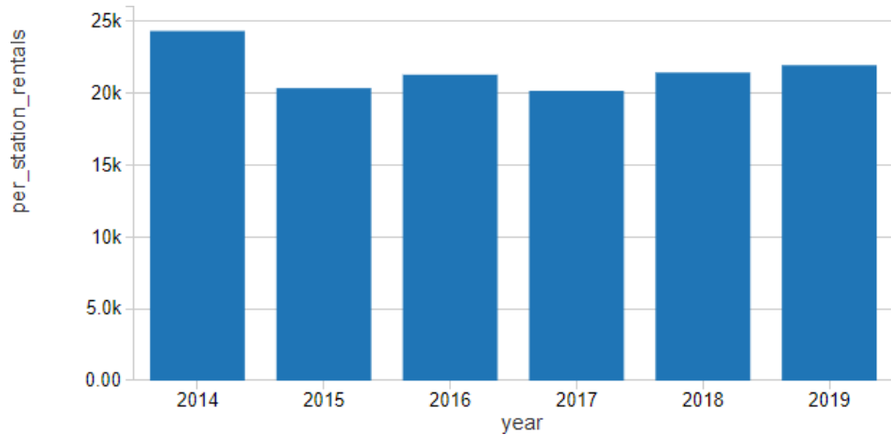


開始以来、レンタル台数は前年比で増加していることがわかります。この伸びの一部は自転車の利用率が上がったことによるものと思われますが、その多くは駅全体のネットワークの拡大に沿ったものと思われます。

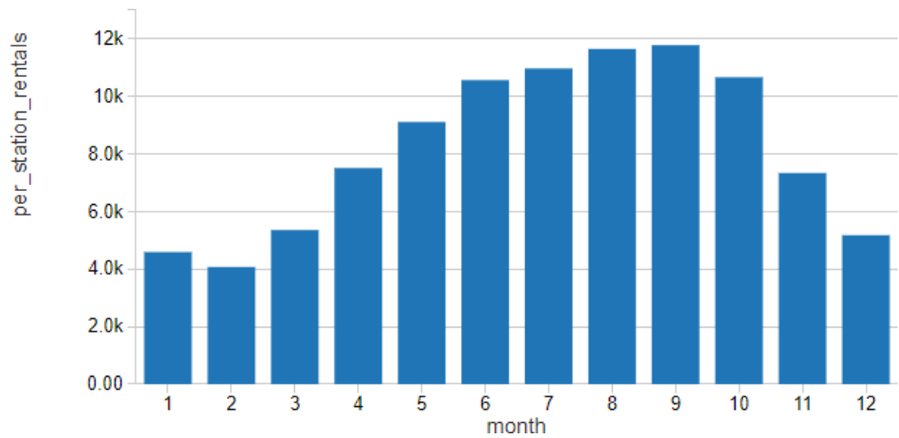




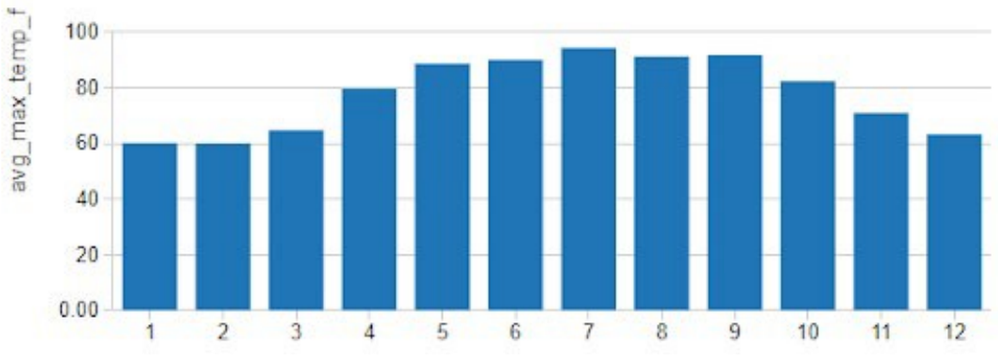
ネットワーク内のアクティブな駅の数で賃貸料を正規化すると、駅ごとの乗降客数の増加は、ここ数年、わずかに直線的な上昇傾向にあると考えられるように、緩やかに増加していることがわかります。



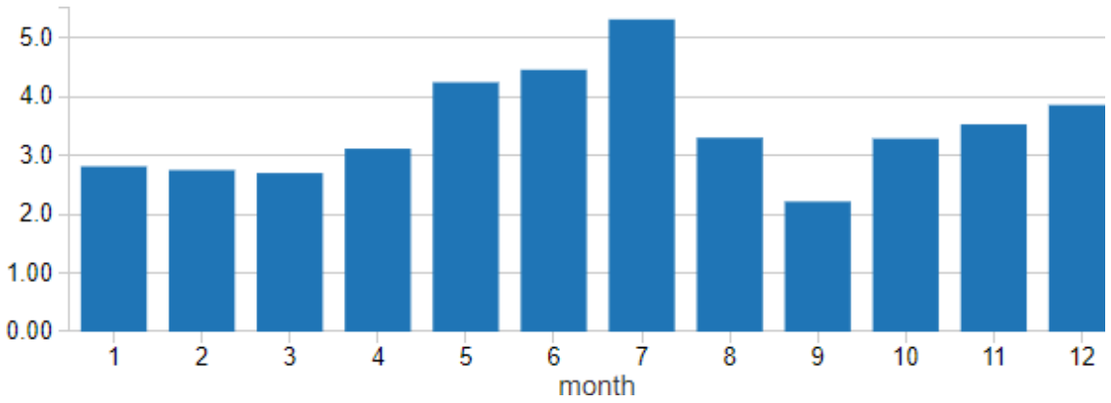
この正規化されたレンタル値を使ってみると、春、夏、秋に上昇し、外の天気が悪くなると冬に下がるという季節的なパターンがはっきりと見られます。



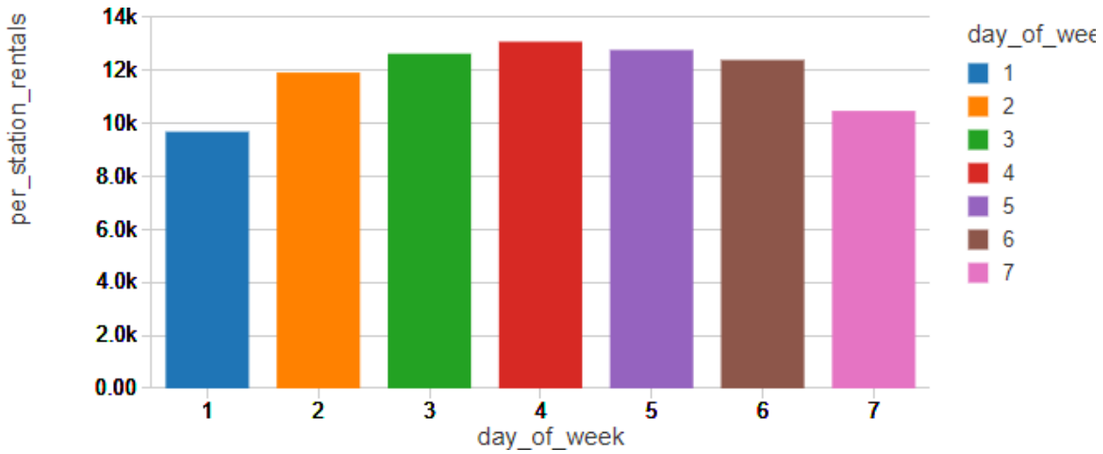
このパターンは、市の最高気温（華氏）のパターンに密接に追随しているように見えます。



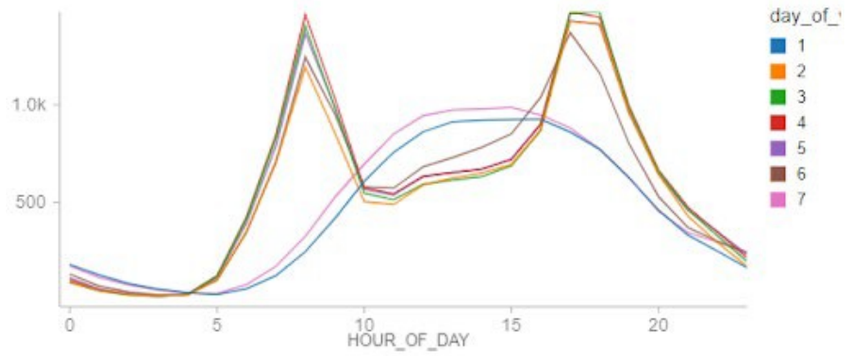
月別のライダー数と気温のパターンを切り離すのは難しいですが、降雨量（月平均インチ）はこのようなパターンを反映しているとは言い難いです。



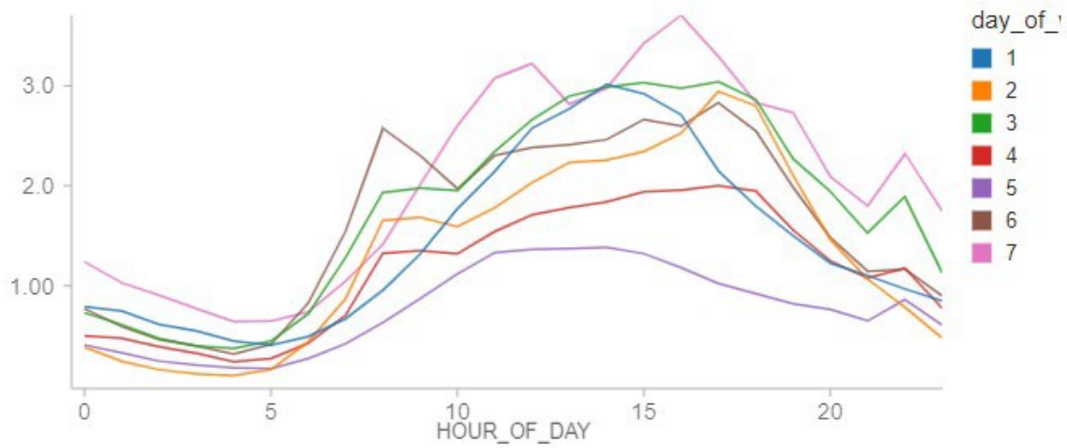
日曜日を「1」、土曜日を「7」とした週ごとの利用者数のパターンを調べてみると、ニューヨーカーは自転車を通勤手段として利用しているようで、他の多くの自転車シェアプログラムに見られるパターンとなっています。



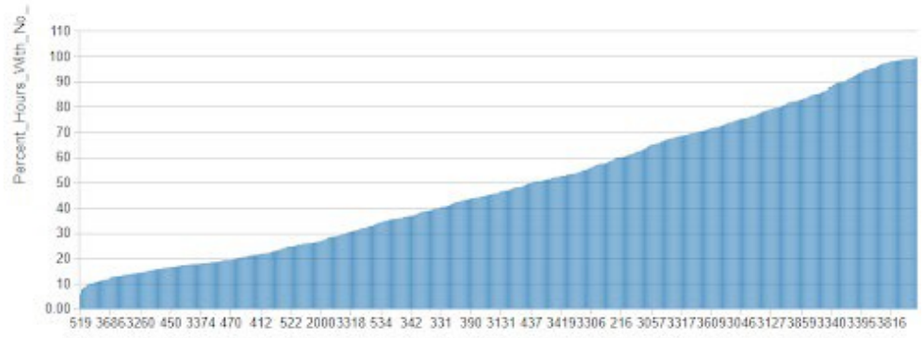
これらの利用パターンを時間帯別に見てみると、平日は通常の通勤時間帯に利用者が急増するパターンが見られます。また、週末になると、よりゆったりとした時間帯に利用されていることがわかり、先ほどの仮説を裏付ける結果となりました。



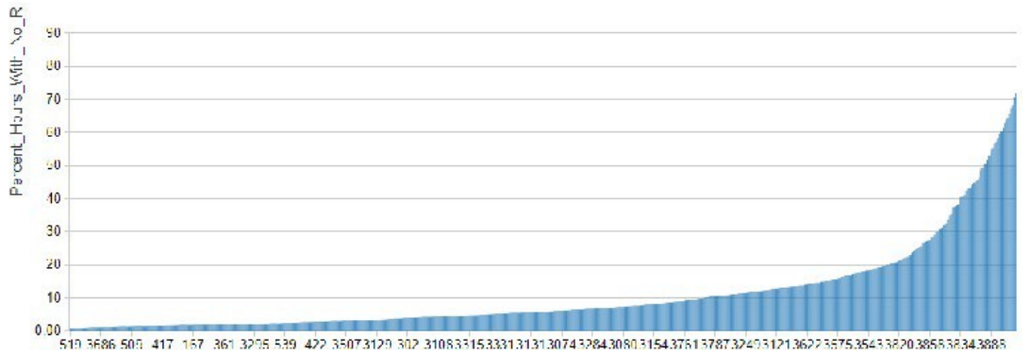
興味深いのは、休日は曜日に関係なく、週末の利用パターンに近い消費パターンを示していることである。祝日の発生頻度が低いことが、このような傾向の不規則性の原因となっているのかもしれません。しかし、信頼性の高い予測をするためには、休日を見極めることが重要であることが裏付けられているように思います。



時間ごとのデータを総合すると、ニューヨークはまさに眠らない街であることがわかる。実際には、レンタサイクルがない時間帯の割合が多い駅も多数あります。



これらの活動のギャップは、予測を生成しようとするときに問題となることがあります。1時間間隔から4時間間隔に移行することで、個々のステーションがレンタル活動を経験していない期間の数は大幅に減少しますが、この時間枠の中で活動していないステーションはまだ多く存在します。



より高いレベルの粒度に移行することで非活動期間の問題を回避する代わりに、我々は毎時レベルでの予測を試み、このデータセットを扱うのに役立つ代替的な予測手法がどのように役立つかを探ります。大部分が活動していない局の予測はあまり面白くないので、分析は最も活動的な上位200局に限定します。

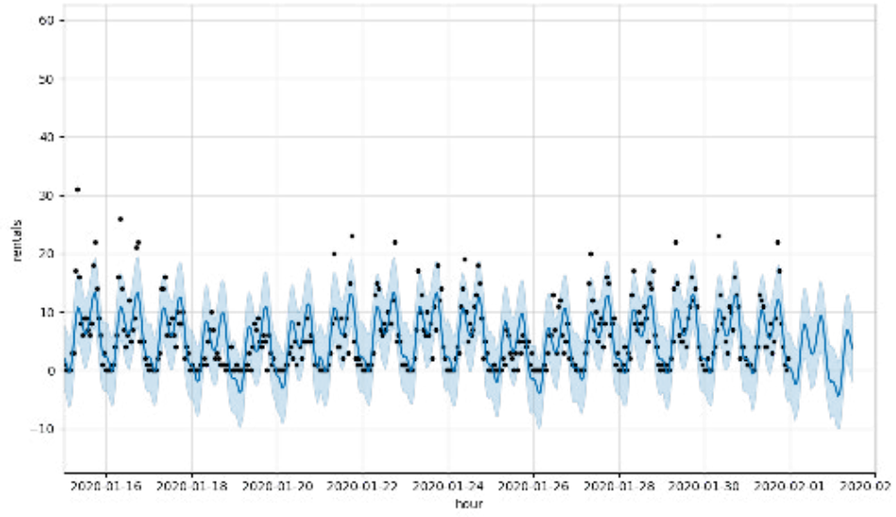
Facebook Prophet でバイクシェアのレンタルを予測する

最初の試みとして、駅ごとの自転車レンタルを予測するために、時系列予測のための人気の高い Python ライブラリである Facebook Prophet を利用しました。モデルは、日次、週次、年次の季節パターンを持つ線形成長パターンを探索するように構成されています。祝日に関連するデータセットの期間も特定し、これらの日付での異常行動がアルゴリズムによって検出された平均、トレンド、季節パターンに影響を与えないようにしました。

以前に参照したブログ記事で説明したスケールアウトパターンを用いて、最もアクティブな 200 の観測点についてモデルを訓練し、それぞれについて 36 時間の予測を生成した。モデルの平均二乗誤差（RMSE）は 5.44、平均平均比例誤差（MAPE）は 0.73でした。（ゼロ値の実績は MAPE の計算には 1 を使用します。）

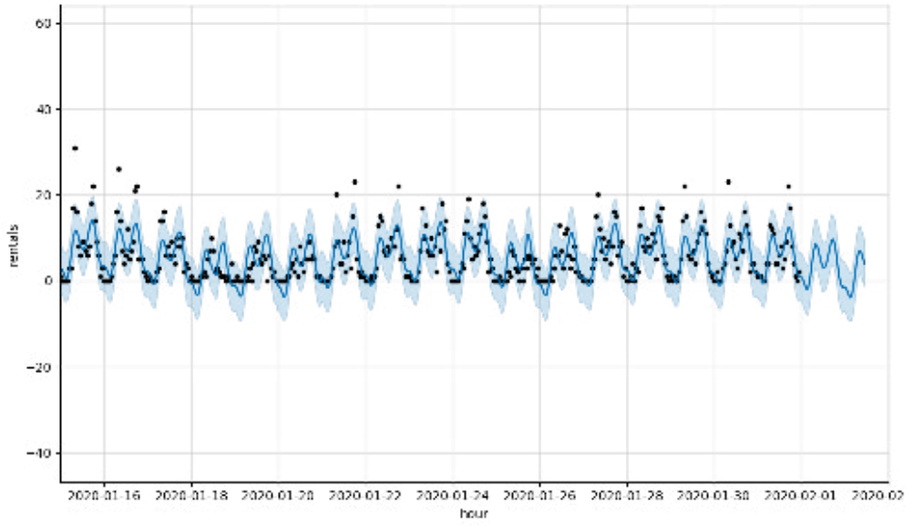
これらの指標は、モデルが賃貸料を予測するのにはそれなりに良い仕事をしているが、時間当たりの賃貸料が高くなると欠落していることを示しています。個々のステーションの売上データを可視化すると、ステーション518のチャートのようにグラフで見ることができます。E 39 St と 2 Ave の RMSE は 4.58、MAPE は 0.69 です。

詳細は時系列 Notebook を参照してください。



その後、モデルは気温と降水量を回帰因子として組み込むように調整された。結果として得られた予測の RMSE は 5.35、MAPE は 0.72でした。非常にわずかに改善されたとはいえ、モデルはまだ駅レベルで見られる乗降客数の大きな変動に対応するのは困難です。

詳細は、リプレッサーを用いた時系列 Notebook を参照してください。



両方の時系列モデルで高い値をモデル化するのが困難なこのパターンは、ポアソン分布を持つデータを扱う場合の典型的なパターンです。このような分布では、その上の値の長い尾を持つ平均の周りに多数の値があるでしょう。平均の反対側では、ゼロの床はデータを歪ませます。今日、Facebook Prophet はデータが正規分布（ガウス分布）を持つことを期待していますが、ポアソン回帰を組み込む計画が議論されています。

サプライチェーンの需要予測のための代替的なアプローチ

では、どのようにしてこれらのデータの予測を生成するのでしょうか？Facebook Prophet の管理人が検討しているように、一つの解決策は、伝統的な時系列モデルのコンテキストでポアソン回帰機能を活用することです。これは優れたアプローチかもしれませんが、広く文書化されていないので、他のテクニックを検討する前に自分たちで取り組むのは、私たちのニーズには最適なアプローチではないかもしれません。

別の潜在的な解決策は、非ゼロ値の規模とゼロ値期間の発生頻度をモデル化することである。それぞれのモデルの出力を組み合わせて、予測を組み立てることができます。Croston の方法として知られるこの方法は、最近リリースされた Croston Python ライブラリによってサポートされていますが、別のデータサイエンティストが独自の関数を実装しています。しかし、これはまだ広く採用されている方法ではありません（1970 年代までさかのぼっているにもかかわらず）、私たちの好みは、もう少し枠にとらわれない方法を探求することです。

この選好を考えると、ランダム・フォレスト回帰器はかなり理にかなっているように思えます。決定木は一般的に、多くの統計手法のようにデータ分布に同じ制約を課すことはありません。予測された変数の値の範囲は、モデルを訓練する前に平方根変換のようなものを使用して家賃を変換することに意味があるかもしれませんが、その場合でも、アルゴリズムがそれなしでどれだけうまく実行されるかを見ることができるかもしれません。

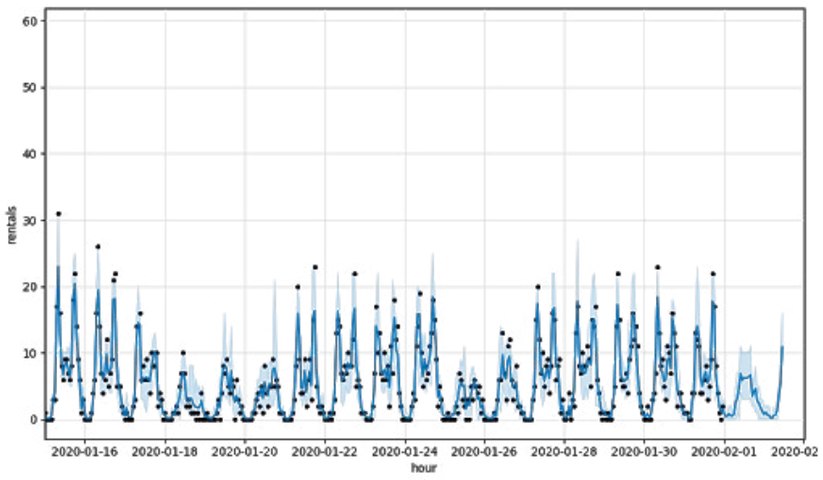
このモデルを活用するためには、いくつかの機能を開発する必要があります。探索的分析から、データには年間、週単位、日単位の両方のレベルで強い季節的パターンがあることが明らかになりました。これにより、年、月、曜日、時間帯を特徴として抽出することができます。また、休日のフラグを入れることもあります。

ランダムフォレスト回帰器と時間由来の特徴のみを使用して、全体的なRMSE は3.4、MAPE は 0.39となります。ステーション 518 については、RMSE とMAPE の値は以下のとおりです。3.09、0.38 となっています。

詳しくはTemporal Notebook をご覧ください。

降水量と気温のデータをこれらの同じ時間的特徴のいくつかと組み合わせて活用することで、より良い（完全ではないが）高い賃貸価格のいくつかに対応することができます。ステーション 518 の RMSE は 2.14 に低下し、MAPE は 0.26 に低下しました。全体として、RMSE は2.37に低下し、MAPE は 0.26 に低下しており、気象データは自転車の需要を予測する上で価値があることを示しています。

詳細については、「時間的および天候の特徴を持つランダムフォレスト」Notebook を参照してください。



結果の意味合い

粒度の細かいレベルでの需要予測では、モデル化へのアプローチを別の方法で考える必要があるかもしれない。高レベルの時系列パターンにまとめても問題ないと考えられるような外部からの影響力は、我々のモデルにもっと明示的に組み込む必要があるかもしれません。集計レベルでは隠れていたデータ分布のパターンがより容易に露出し、モデリングアプローチの変更が必要になるかもしれません。このデータセットでは、1時間ごとの気象データを含めることと、従来の時系列手法から、入力データについての仮定を少なくするアルゴリズムへとシフトすることで、これらの課題を解決することができました。

他にも探索する価値のある外部のインフルエンサーやアルゴリズムはたくさんあるかもしれませんが、この道を進んでいくうちに、これらの中には他のものよりもデータの一部のサブセットに適したものがあることに気づくかもしれません。また、新しいデータが入ってくると、以前はうまく機能していた技術を放棄し、新しい技術を検討する必要があるかもしれません。

細かい粒度の需要予測を探求している顧客によく見られるパターンは、トレーニングと予測サイクルごとに複数の手法を評価することで、自動化されたモデルのベークオフと表現することができます。ベークオフ・ラウンドでは、データの与えられたサブセットに対して最高の結果を出したモデルがラウンドを勝ち抜き、各サブセットが独自の勝ちモデルタイプを決定することができます。最終的には、採用したアルゴリズムとデータが適切に一致するような優れたデータサイエンスを確実に実行したいものですが、次から次へと記事にあるように、問題に対する解決策は常に1つだけとは限りませんし、ある時には他の時よりもうまくいくものもあります。Apache Spark やデータブリックスのようなプラットフォームを利用することで、これら全てのパスを探索し、ビジネスに最適なソリューションを提供するための計算能力にアクセスできるようになりました。

小売/消費財と需要予測リソースの追加

これらの開発者リソースを使って実験を始めましょう。

1. Notebooks
   - データ作成 Notebook
   - 探索分析 Notebook
   - 時系列 Notebook
   - リプレッサーを用いた時系列 Notebook
   - テンポラリー Notebook
   - テンポラル＆ウェザー機能を備えたランダムな森の Notebook
2. 小売業と消費財 のためのデータ分析とAI の大規模化ガイドをダウンロード
3. Dollar Shave Club と Zalando がどのようにデータブリックスを活用してイノベーションを起こしているかについては、小売・消費財のページをご覧ください。
4. 最近のブログ「Fine-Grained Time Series Forecasting at Scale with Facebook Prophet and Apache Spark」は、Databricks Unified Data Analytics Platform がどのようにタイムリーに、かつ粒度レベルで課題を解決し、製品のインベントリを正確に調整できるようにしているかを紹介しています。

第 6 章　Prophet と Apache Spark を活用した時系列予測の大規模展開

投稿者：
Bilal Obeidat
Bryan Smith
Brenner Heintz

2020 年 1 月 27 日

時系列予測の進歩により、小売業者はより信頼性の高い需要予測を作成することができるようになりました。現在の課題は、これらの予測をタイムリーに、かつ商品インベントリを正確に調整できるような粒度で作成することです。Apache Spark™ と Facebook Prophet を活用することで、これらの課題に直面している多くの企業は、従来のソリューションのスケーラビリティと精度の限界を克服できることを発見しています。

この記事では、時系列予測の重要性について説明し、サンプルの時系列データを可視化し、Facebook Prophet の使用方法を示すシンプルなモデルを構築します。一つのモデルを構築することに慣れたら、Prophet と Apache Spark™ の魔法を組み合わせて、一度に何百ものモデルをトレーニングする方法を紹介し、今までほとんど達成されなかった粒度レベルで個々の商品と店舗の組み合わせについて正確な予測を作成できるようにします。

正確でタイムリーな予測がこれまで以上に重要になっています。

製品やサービスの需要をよりよく予測するために、時系列分析のスピードと精度を向上させることは、小売業者の成功にとって非常に重要です。店頭に商品が多すぎると、棚や倉庫のスペースが圧迫され、商品の期限が切れる可能性があり、小売業者はインベントリに資金が縛られ、メーカーが生み出す新たな機会や消費者パターンの変化を利用することができなくなる可能性があります。店頭に置かれている商品が少なすぎると、顧客が必要とする商品を購入できない可能性がある。このような予測エラーは、小売業者の収益をすぐに失うだけでなく、時間の経過とともに消費者のフラストレーションが顧客を競合他社に向かわせることにもなりかねません。

新たな期待に必要な、正確な時系列予測手法とモデル

しばらくの間、企業資源計画（ERP）システムやサードパーティのソリューションは、単純な時系列モデルに基づいた需要予測機能を小売業者に提供してきました。しかし、技術の進歩と業界のプレッシャーの増大に伴い、多くの小売業者は、歴史的に利用可能な線形モデルやより伝統的なアルゴリズムを超えたものへの移行を模索しています。

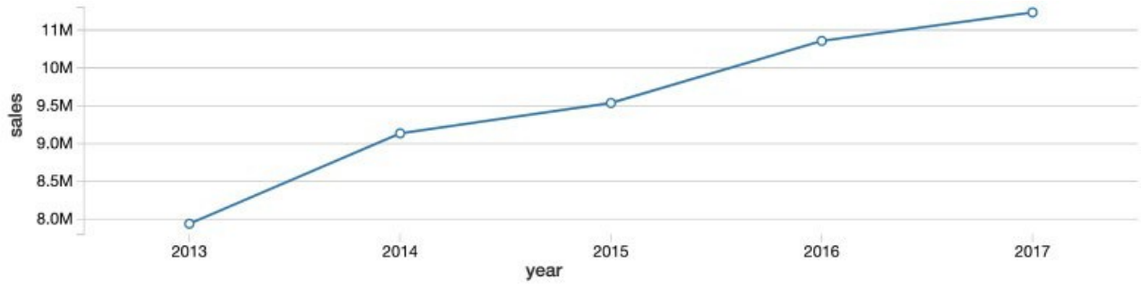
Facebook Prophet が提供するような新しい機能がデータサイエンスコミュニティから登場しており、企業はこれらの機械学習モデルを時系列予測のニーズに柔軟に適用することを求めています。

このような従来の予測ソリューションからの脱却の動きに伴い、小売店などでは需要予測の複雑さだけでなく、何十万、何百万もの機械学習モデルをタイムリーに生成するために必要な作業を効率的に分散させるための社内の専門知識を身につける必要があります。幸いなことに、Spark を使ってこれらのモデルのトレーニングを分散させることができるので、商品やサービスの全体的な需要だけでなく、各地域の商品ごとの固有の需要を予測することが可能になります。

時系列データにおける需要の季節性の可視化

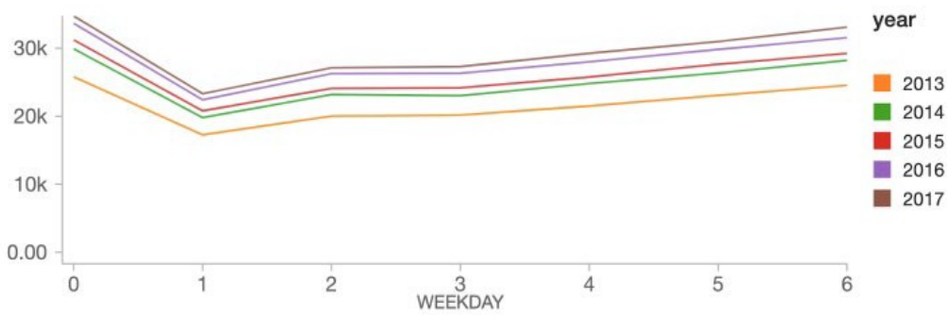
個々の店舗や商品の細かい需要予測を生成するための Prophet の使用を実証するために、Kaggle から公開されているデータセットを使用します。これは、10 の異なる店舗にまたがる 50 の個別商品の日販データの5年間のデータで構成されています。

まずは、全商品・全店舗の全体の年間売上高の推移を見てみましょう。ご覧のように、全商品の売上高は前年比で増加しており、プラトーを中心に明確な収束の兆しは見られません。



次に、同じデータを月別に見てみると、前年比の上昇トレンドが毎月着実に進んでいるわけではないことがわかります。その代わり、夏場にピークがあり、冬場に谷間があるという明確な季節的なパターンが見られます。Databricks Collaborative Notebooks に内蔵されているデータの可視化機能を使用して、チャート上でマウスを動かすことで、各月のデータの価値を確認することができます。

平日レベルでは、日曜日（平日 0）にピークを迎えた後、月曜日（平日 1）に大きく落ち込み、その後は順調に回復しています。



Facebook Prophet で シンプルな時系列予測モデルを作成する

上のグラフに示されているように、私たちのデータは、年間と週単位の季節的なパターンとともに、売上高が前年比で明らかに上昇していることを示しています。このようなデータの重複したパターンこそが、Prophet が対処するために設計されたものなのです。

Facebook Prophet はscikit-learn の API に従っているので、sklearn の経験があれば誰でも簡単に利用できます。2 カラムのpandas DataFrame を入力として渡す必要があります。1カラム目は日付、2 カラム目は予測する値（この場合は売上）です。データが適切な形式であれば、モデルの構築は簡単です。

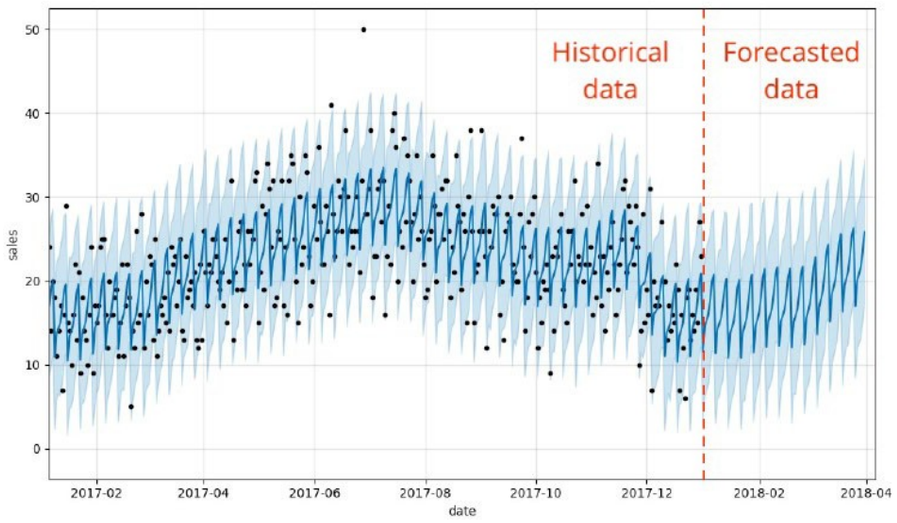
```
import pandas as pd
from fbprophet import Prophet

# 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)
```

モデルをデータに当てはめることができたので、それを使って 90 日間の予測を立ててみましょう。以下のコードでは、prophet の make_future_dataframe メソッドを使用して、過去の日付と 90 日以降の日付の両方を含むデータセットを定義しています。

```
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、実際のデータと予測データがどのように並んでいるかを可視化することができます。Prophet のビルトインの .plot メソッドを使って、実際のデータと予測されたデータがどのように並んでいるか、そして将来の予測を視覚化することができるようになりました。ご覧のように、先ほど説明した週ごとの需要パターンと季節ごとの需要パターンは実際に予測結果に反映されています。

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



この可視化は雑然としています。Bartosz Mikulski 氏は、チェックアウトする価値のある優れた内訳を提供しています。一言で言えば、黒い点が当社の実績を表し、濃い青色の線が当社の予測を表し、薄い青色のバンドが当社の（95%）不確実性区間を表しています。

Prophet とSpark で何百もの時系列予測モデルを並列にトレーニング

1つの時系列予測モデルを構築する方法をデモしたので、Apache Spark のパワーを使って、さらに多くの努力をしてみましょう。目標は、データセット全体の予測を1つ生成するのではなく、商品と店舗の組み合わせごとに何百ものモデルと予測を生成することです。

このようにモデルを構築することで、例えば食料品店のチェーン店では、サンダスキーの店舗に注文すべき牛乳の量を正確に予測することができます。クリーブランドの店舗で必要とされる量とは異なります。

Spark DataFrames を使って時系列データの処理を分散する方法

データサイエンティストは、Apache Spark のような分散データ処理エンジンを使用して大量のモデルを訓練するという課題に頻繁に取り組んでいます。Spark クラスタ を利用することで、クラスタ内の個々のワーカーノードは、他のワーカーノードと並行してモデルのサブセットを訓練することができ、時系列モデルのコレクション全体を訓練するのに必要な全体の時間を大幅に短縮することができます。

もちろん、ワーカー・ノード（コンピュータ）のクラスタ上でモデルをトレーニングするには、より多くのクラウド・インフラストラクチャが必要であり、それにはコストがかかります。しかし、オンデマンドのクラウドリソースを簡単に利用できるため、企業は必要なリソースを迅速にプロビジョニングすることができます。

これにより、物理的な資産に長期的なコミットメントをすることなく、大規模なスケーラビリティを実現することができます。

Spark で分散データ処理を実現するための重要な仕組みが DataFrame です。Spark の DataFrame にデータをロードすることで、データはクラスタ内のワーカーに分散されます。これにより、これらのワーカーはデータのサブセットを並行して処理することができ、作業に必要な全体の時間を短縮することができます。

もちろん、各ワーカーは、作業を行うために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。キー値に関するデータをグループ化することで、この場合はストアとアイテムの組み合わせについて、これらのキー値に関する全ての時系列データを特定のワーカー・ノードにまとめます。

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

> ここでは、groupBy のコードを共有して、多くのモデルを効率的に並列に学習できることを強調していますが、次のセクションでUDFを設定してデータに適用するまでは、実際には使用されません。

Pandas のユーザー定義機能を活用する

時系列データを店舗と項目ごとに適切にグループ化したので、各グループごとに1つのモデルを訓練する必要があります。これを達成するために、pandas のユーザー定義関数（UDF）を使用すると、DataFrame の各データグループにカスタム関数を適用することができます。

このUDFは、各グループのモデルを訓練するだけでなく、そのモデルからの予測値を表す結果セットを生成します。しかし、この関数は DataFrame 内の各グループを他のグループとは独立して訓練し、予測しますが、各グループから返された結果は、便利なように 1 つのDataFrame に集められます。これにより、商品レベルの予測を生成しつつ、結果を1つの出力データセットとしてアナリストや管理者に提示することが可能になります。

以下のPython のコードを見ればわかるように、UDF を構築するのは比較的簡単です。UDF はpandas_udf メソッドでインスタンス化され、それが返すデータのスキーマと、それが受け取ることを期待するデータのタイプを識別します。これに続いて，UDF の作業を実行する関数を定義します.

関数の定義の中で、モデルをインスタンス化し、設定し、受け取ったデータに適合させます。モデルは予測を行い、そのデータは関数の出力として返されます。

```
@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
```

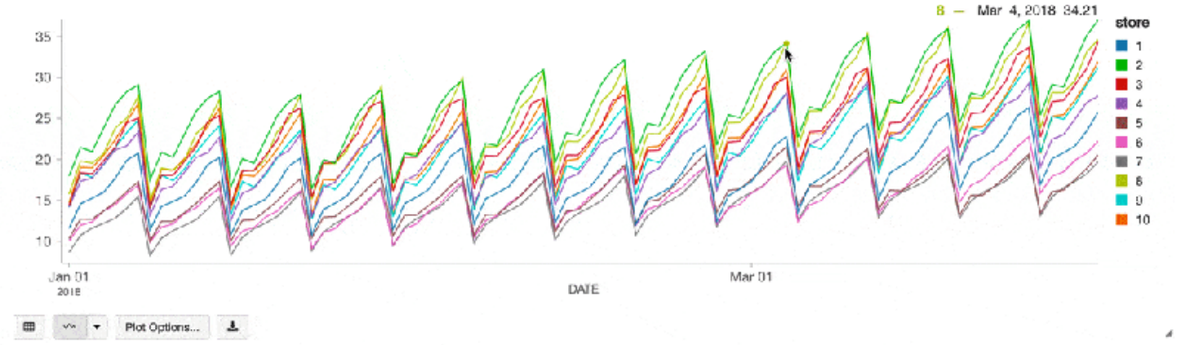
ここで、全てをまとめるために、先ほど説明した groupBy コマンドを使用して、データセットが特定の店舗とアイテムの組み合わせを表すグループに適切に分割されていることを確認します。次に、UDF を DataFrame に適用するだけで、UDF がモデルを適合させ、データの各グループ化について予測を行うことができます。

この関数を各グループに適用することで返されるデータセットは、予測を生成した日付を反映して更新されます。これは、最終的に本番に向けて機能を導入する際に、異なるモデルの実行中に生成されたデータを追跡するのに役立ちます。

```
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

これで、各商品の組み合わせごとに時系列予測モデルを構築しました。SQL クエリを使用して、アナリストは各製品に合わせた予測を表示することができます。下のグラフでは、10店舗における製品#1の予測需要をプロットしています。ご覧のように、需要予測は店舗によって異なりますが、一般的なパターンは全ての店舗で一貫しています。



新たな販売データが到着すると、効率的に新たな予測を作成し、既存のテーブル構造に追加することができるため、アナリストは状況の変化に応じてビジネスの予測を更新することができます。

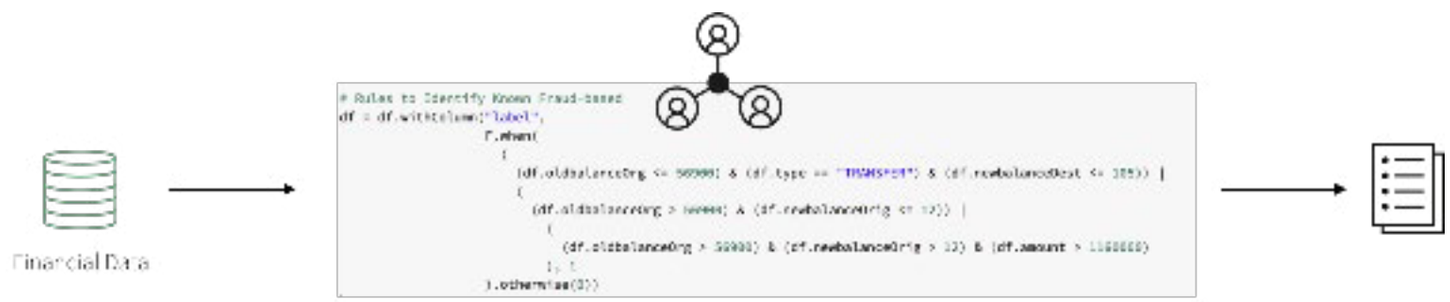
> データブリックスの無料の **Notebook** を使って実験を始める

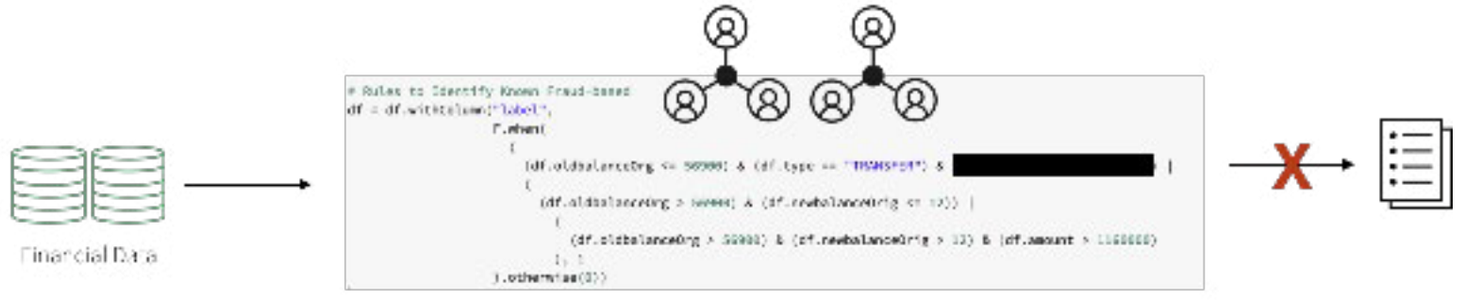
第 7 章　データブリックス上で決定木と MLflow を用いて金融詐欺検知を大規模展開

投稿者：
Elena Boiarskaia
Navin Albert
Denny Lee

2019 年 5 月 2 日

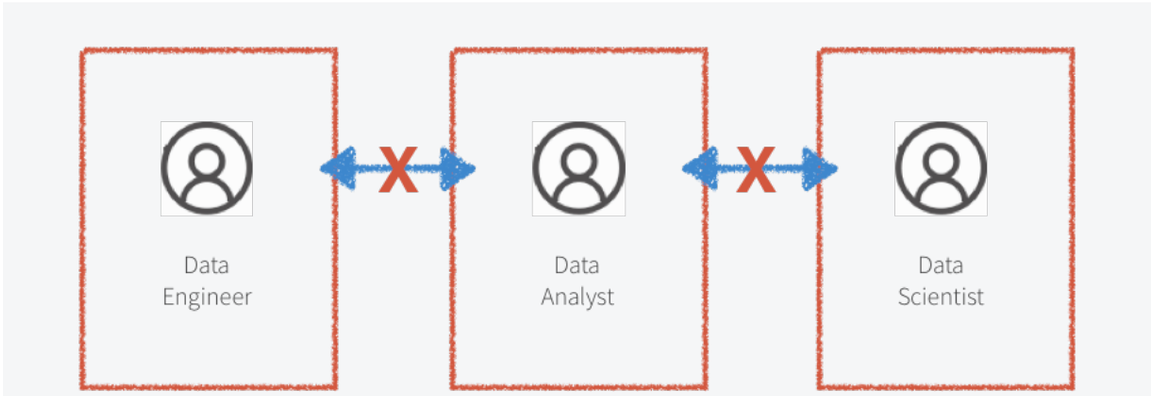
人工知能を使用して大規模な不正行為のパターンを検出することは、どのようなユースケースであっても課題となります。膨大な量の過去データをふるいにかける必要があること、絶えず進化する機械学習やディープラーニング技術の複雑さ、そして不正行為の実際の事例の数が非常に少ないことは、干し草の山の中から針がどのように見えるかわからない中から針を見つけることに匹敵します。金融サービス業界では、セキュリティへの懸念や、不正行為がどのように特定されたかを説明することの重要性が加わり、作業の複雑さがさらに増しています。



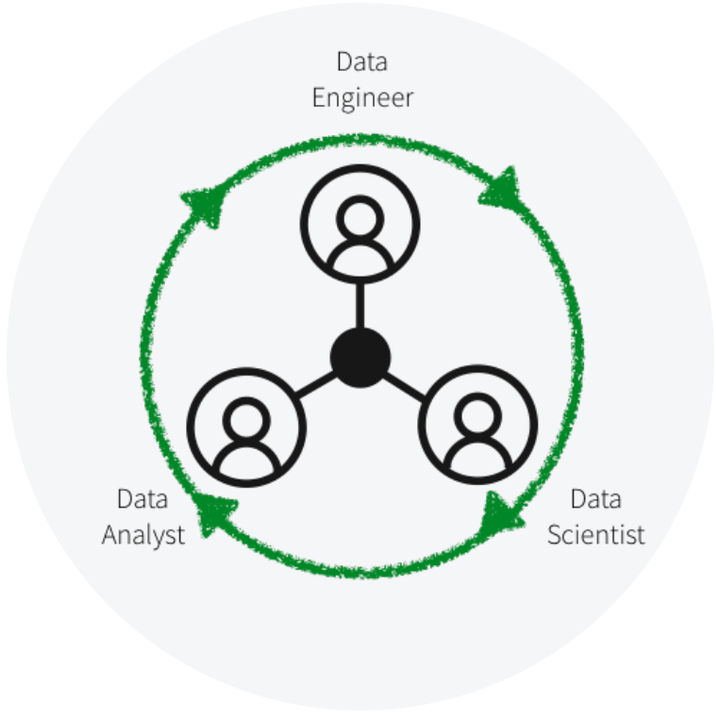


これらの検出パターンを構築するために、ドメインエキスパートのチームが、詐欺師の一般的な行動に基づいた一連のルールを作成します。ワークフローには、金融詐欺検知の分野の専門家が、特定の行動に関する一連の要件をまとめることが含まれます。データサイエンティストは、利用可能なデータからサブサンプルを抽出し、これらの要件と場合によっては既知の詐欺事例を用いて、ディープラーニングまたは機械学習アルゴリズムのセットを選択します。このパターンを本番で使用するために、データエンジニアは、結果として得られたモデルをしきい値を持つ一連のルールに変換することができます。

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に至った一連の特徴を明確に提示することができます。しかし、このアプローチには多くの困難も伴います。ハードコード化された一連のルールを使用した不正検知システムの実装は、非常にもろいものです。不正行為のパターンに変更があった場合、更新には非常に長い時間がかかります。そのため、現在の市場で起こっている不正行為の変化に追いつき、適応することが難しくなってしまいます。



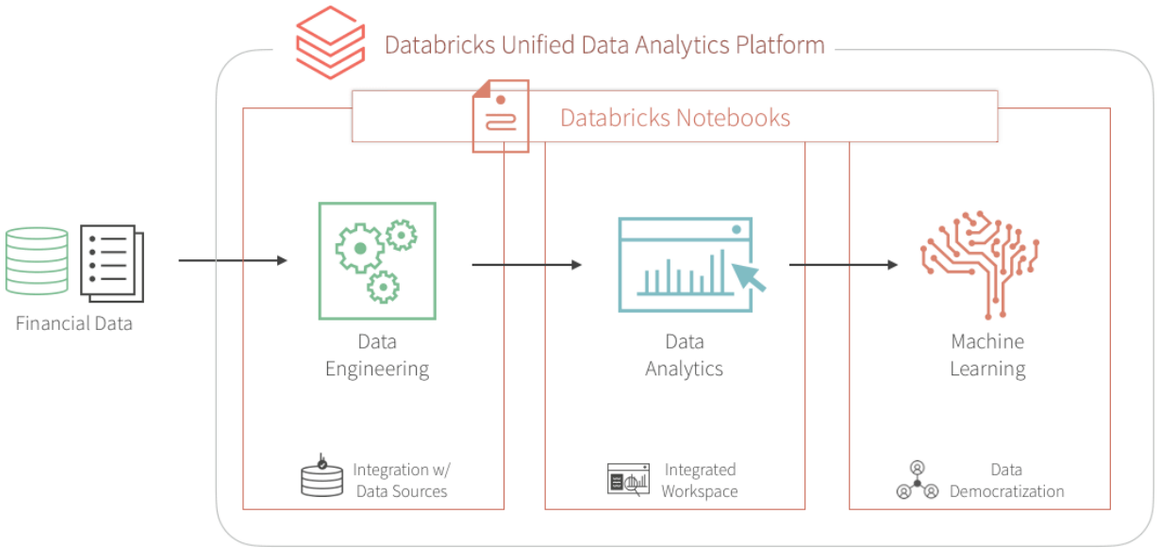
さらに、上述したワークフローのシステムはサイロ化されていることが多く、ドメインエキスパート、データサイエンティスト、データエンジニアが全てコンパートメント化されています。データエンジニアは、大量のデータを管理し、ドメインエキスパートとデータサイエンティストの作業を本番レベルのコードに変換する役割を担っています。共通のプラットフォームがないため、ドメインエキスパートとデータサイエンティストは、分析のために1台のマシンに収まるサンプルダウンされたデータに頼らざるを得ません。これがコミュニケーションの難しさにつながり、最終的にはコラボレーションの欠如につながります。



このブログでは、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一して、このようなルールベースの検知ユースケースをいくつかデータブリックスのプラットフォーム上で機械学習ユースケースに変換する方法を紹介します。機械学習による不正検知のデータパイプラインを作成し、大規模なデータセットからモジュール化された特徴を構築するフレームワークを活用して、リアルタイムでデータを可視化する方法を学びます。また、決定木と Apache Spark MLlib を使って不正行為を検知する方法も学びます。その後、MLflow を使用してモデルを反復処理し、精度を向上させるための改良を行います。

機械学習で解決

機械学習モデルは、特定された不正事例を正当化する方法がない「ブラックボックス」ソリューションを提供すると考えられているため、金融の世界では機械学習モデルには一定の消極的な傾向があります。GDPR の要件や金融規制により、データサイエンスの力を活用することは一見不可能に見えます。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スケールでの不正行為を検知することで、上述した問題のホストを解決できることが示されています。



金融詐欺を検知するために教師付き機械学習モデルを訓練することは、実際に確認された詐欺行為の例が少ないため、非常に困難である。しかし、特定のタイプの不正行為を識別する既知のルールのセットが存在することで、以下のようなモデルを作成することができます。合成ラベルのセットと特徴の初期セット。この分野のドメインエキスパートによって開発された検出パターンの出力は、おそらく次のようなプロセスを経ています。本番さながらの適切な承認プロセスが必要となります。これは、予想される不正行為のフラグを生成し、したがって、機械学習モデルを訓練するための出発点として使用することができます。

これは同時に3つの懸念事項を緩和します。

1. トレーニングラベルの不足
2. どのような機能を使うかの判断
3. モデルに適切なベンチマークを持つこと

ルールベースの不正行為フラグを認識するために機械学習モデルを訓練すると、混乱行列を介して期待される出力と直接比較することができます。結果がルールベースの検出パターンと密接に一致する場合には、このアプローチは、懐疑論者との機械学習ベースの不正行為防止への信頼を得るのに役立ちます。このモデルの出力は非常に解釈しやすく、元の検出パターンと比較した場合の予想される偽陰性と偽陽性の基本的な議論として役立つかもしれません。

さらに、機械学習モデルが解釈しにくいという懸念は、初期の機械学習モデルとして決定木モデルが使用されれば、さらに緩和されるかもしれません。モデルは一連のルールに対して訓練されているので、決定木はおそらく他のどの機械学習モデルよりも優れています。さらなる利点は、もちろん、モデルの透明性を最大限に高めたことです。これは、詐欺の意思決定プロセスを本質的に示しますが、人間の介入やルールや閾値をハードコーディングする必要はありません。もちろん、モデルの将来の反復は、最大の精度を達成するために別のアルゴリズムを完全に利用する可能性があることを理解しておかなければなりません。モデルの透明性は、アルゴリズムに組み込まれた特徴を理解することによって最終的に達成されます。解釈可能な特徴を持つことは、解釈可能で防御可能なモデルの結果をもたらします。

機械学習アプローチの最大の利点は、最初のモデリング作業の後に、将来の反復がモジュール化され、ラベル、特徴、モデルタイプのセットの更新が非常に簡単かつシームレスに行えるため、生産までの時間が短縮されることです。これは、データブリックスのコラボレーティブ Notebook でさらに促進され、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同じデータセットを使って大規模に作業し、Notebook 環境で直接コラボレーションすることができます。さあ、さっそく開始しましょう。

データの取り込みと探索

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

PaySim のデータは、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ヶ月間の財務ログから抽出された実際の取引のサンプルに基づいて、モバイルマネーの取引をシミュレートしています。下の表は、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).

データを探る

DataFrame の作成： Databricks File System（DBFS）にデータをアップロードしたので、Spark SQL を使って素早く簡単に DataFrames を作成することができます。

```
# 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1,000 行を見てデータを確認してみましょう。

```
# 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)
```

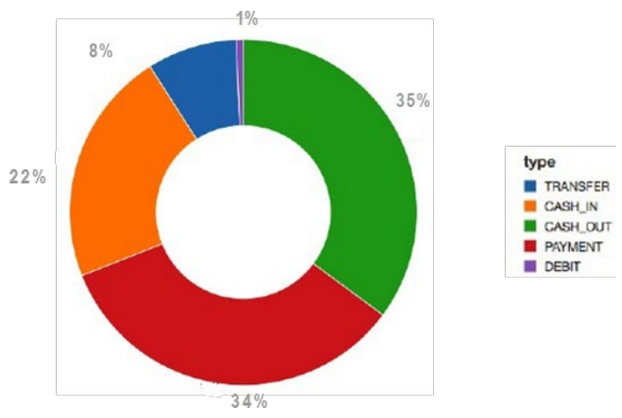
step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest
1	PAYMENT	9839.64	C1231006815	170136	160296.36	M1979787155	0
1	PAYMENT	1864.28	C1666544295	21249	19384.72	M2044282225	0
1	TRANSFER	181	C1305486145	181	0	C553264065	0
1	CASH_OUT	181	C840083671	181	0	C38997010	21182
1	PAYMENT	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
1	PAYMENT	4024.36	C1265012928	2671	0	M1176932104	0

Showing the first 1000 rows.

トランザクションの種類

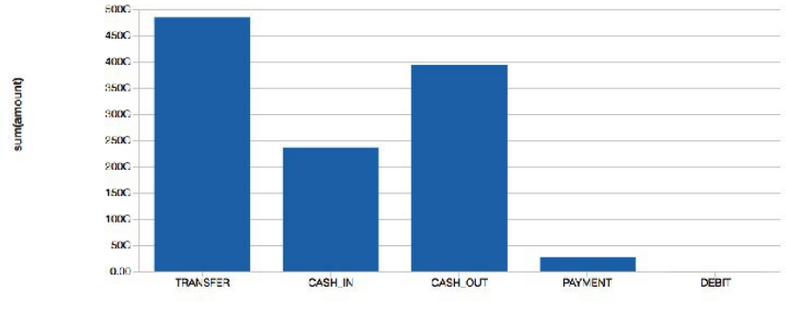
データを可視化して、データが捉えている取引の種類と全体の取引量への貢献度を把握してみましょう。

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



また、どのくらいの金額なのかを把握するために、取引の種類に応じたデータや、現金の移動量（金額の合計）に対する寄与度なども可視化してみましょう。

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



ルールベースのモデル

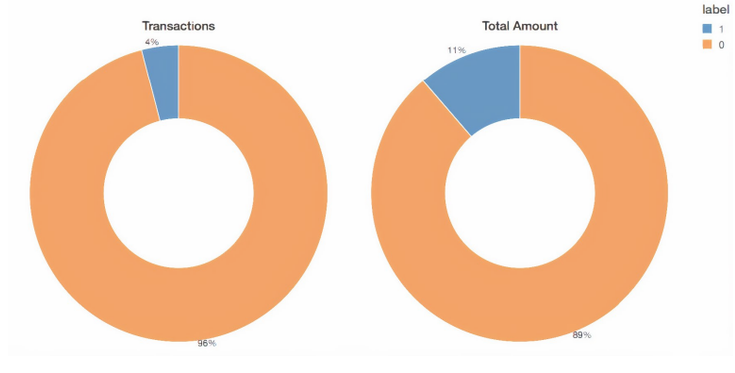
モデルを訓練するために、既知の不正事例の大規模なデータセットから始めることはないでしょう。ほとんどの実用的なアプリケーションでは、不正検知パターンは、ドメインの専門家によって確立された一連のルールによって識別されます。ここでは、これらのルールに基づいて「ラベル」と呼ばれる列を作成します。

```
# 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))
```

ルールでフラグが立てられたデータの可視化

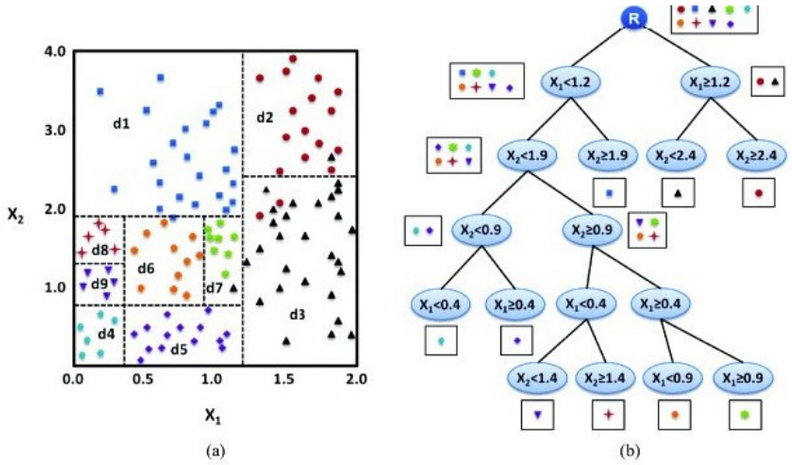
このようなルールでは、かなりの数の不正行為にフラグが立てられることが多いです。フラグされた取引の数を可視化してみましょう。このルールでは、ケースの約 4%、総ドル額の 11% が不正行為としてフラグが立てられていることがわかります。

```
%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、ドメインの専門家は、なぜ取引が不正行為であると特定されたのかを理解する必要があります。そして、アクションが取られる場合、証拠は法廷で提示されなければなりません。意思決定木は簡単に解釈できるモデルであり、このユースケースの出発点として最適です。ディシジョン・ツリーについては、このブログ「The wise old tree」を読んでみてください。



トレーニングセットの作成

ML モデルを構築して検証するために，.randomSplit を用いて 80/20 分割を行います。これにより、トレーニングのためにデータの 80% をランダムに選択し、残りの 20% を結果の検証のために確保します。

```
# 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

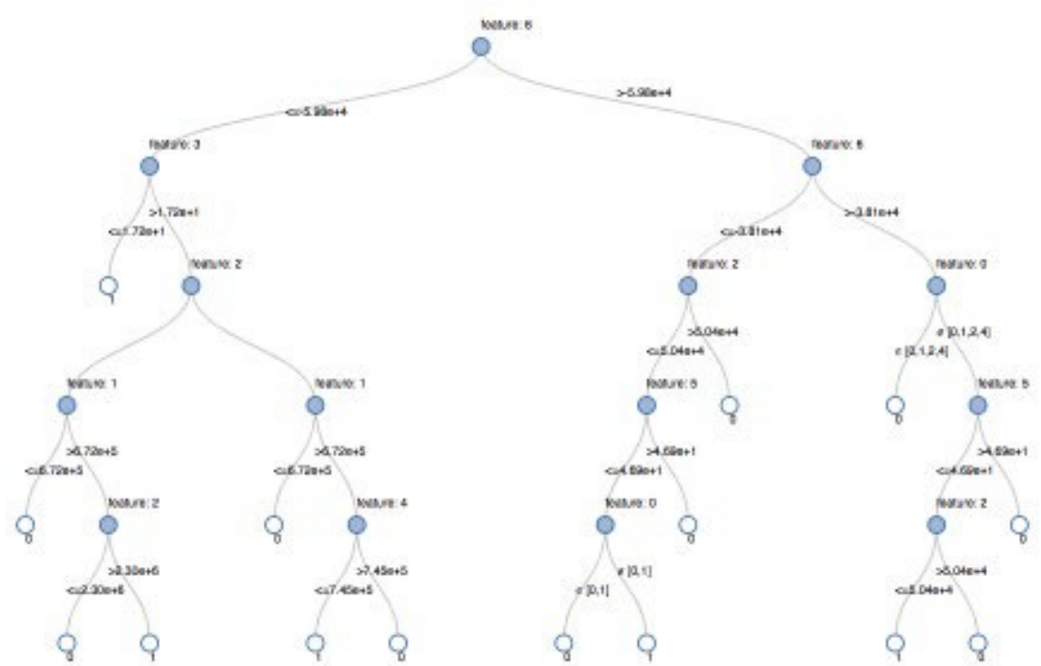
モデルのデータを準備するために、まず、.StringIndexer を使用してカテゴリ変数を数値に変換しなければなりません。次に、モデルに使用したい全ての特徴をアセンブルしなければなりません。決定木モデルに加えて、これらの特徴準備ステップを含むパイプラインを作成し、異なるデータセットでこれらのステップを繰り返すことができるようにします。最初に学習データにパイプラインを適合させ、後のステップでテストデータの変換に使用することに注意してください。

```
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

パイプラインの最後のステージである決定木モデルである display() を呼び出すことで、各ノードで選択された決定を持つ初期フィットモデルを表示することができます。これは、アルゴリズムがどのようにして結果の予測値に到達したかを理解するのに役立ちます。

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



決定木モデルの視覚的表現

モデルチューニング

最適なツリーモデルが得られることを確認するために、いくつかのパラメータのバリエーションを用いてモデルを交差検証します。我々のデータが 96% の負のケースと 4% の正のケースで構成されているので、不均衡な分布を説明するために Precision-Recall（PR）評価メトリックを使用します。

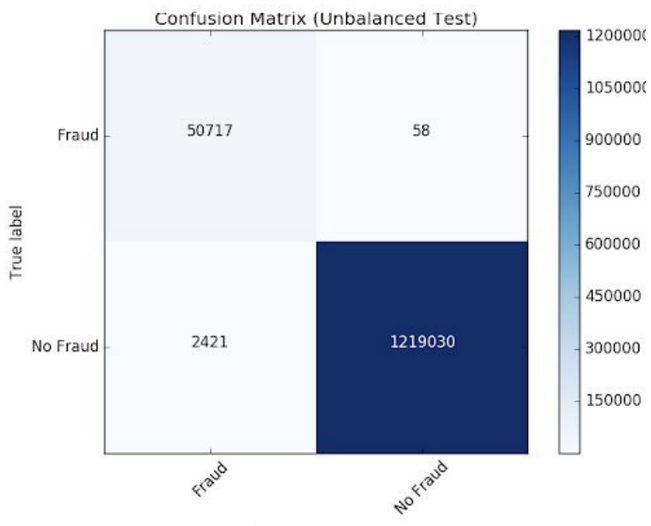
```
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

# 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)
```

モデル性能

学習セットとテストセットの精度-リコール（PR）と ROC 曲線下面積（AUC）を比較することで、モデルを評価する。PR と AUC はともに非常に高い値を示した。

```
# 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
```

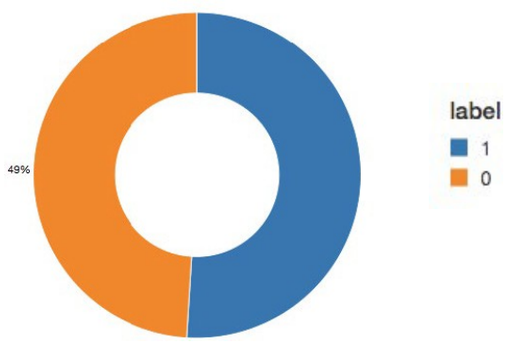
モデルがどのように結果を誤分類したかを見るために、Matplotlib と pandas を使用して混同行列を可視化してみましょう。



クラスのバランスをとる

このモデルは、元のルールが識別したケースよりも 2,421 件多くのケースを識別していることがわかります。これは、より多くの潜在的な不正事例を検出することは良いことかもしれないので、それほど心配するほどのことではありません。しかし、アルゴリズムによって検出されなかったが、元々特定されていたケースが 58 件あります。私たちは、アンダーサンプリングを使用してクラスのバランスをとることで、予測をさらに改善しようとしています。つまり、全ての不正事例を残しておき、その数に合わせて非不正事例をダウンサンプリングして、バランスのとれたデータセットを得るのです。新しいデータセットを可視化すると、イエスとノーのケースが半々になっていることがわかります。

```
# 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())
```



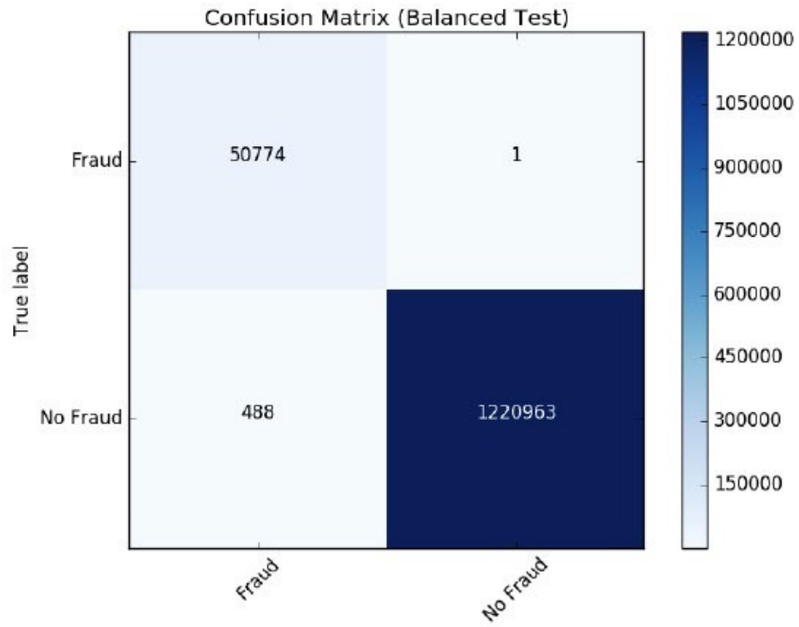
パイプラインの更新

それでは、MLパイプラインを更新して、新しいクロスバリデータを作成してみましょう。MLパイプラインを使用しているので、新しいデータセットで更新するだけで、すぐに同じパイプラインの手順を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
```

結果を見直す

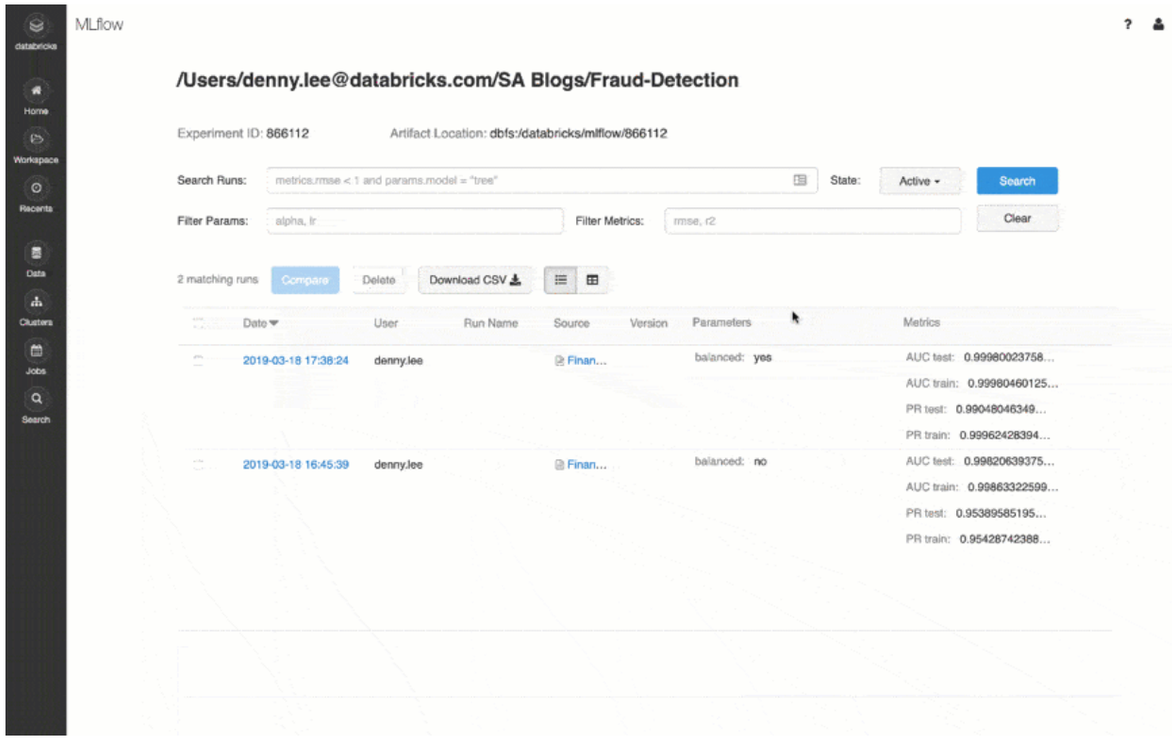
では、新しい混乱マトリックスの結果を見てみましょう。このモデルは、不正なケースを1つだけ誤認していました。クラスのバランスをとることで、モデルは改善されたようです。



モデルのフィードバックと MLflow の利用

生産のためにモデルが選択されたら、モデルがまだ関心のあ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たいと考えています。ルールベースのラベルから始めているので、人間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ています。この段階は、機械学習プロセスの信頼性と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することはできないので、モデルの出力を検証するために慎重に選ばれたケースを提示するようにしたいと考えています。例えば、モデルの確実性が低い予測は、アナリストがレビューするのに適した候補となります。このような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合わせて改善され、進化し続けることが保証されます。

MLflow は、異なるモデルのバージョンを学習する際に、この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ータの結果を比較しながら、実験を追跡することができます。例えば、ここでは、MLflow UI を使って、バランスのとれたデータセットとバランスのとれていないデータセットで学習したモデルの PR と AUC を比較することができます。データサイエンティストは、MLflow を使用して、様々なモデルのメトリクスや、追加の可視化や成果物を追跡することができ、どのモデルを本番環境に導入すべきかの判断を助けることができます。データエンジニアは、選択したモデルとトレーニングに使用したライブラリのバージョンを.jarファイルとして簡単に取得して、本番の新しいデータに展開することができます。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連携は、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 を使用して機械学習モデルに変換する方法の例をレビューしました。このアプローチにより、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することができ、常に変化し続ける不正行為のパターンに対応することができます。不正行為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、新たな不正行為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することができます。特に決定木モデルは、その解釈のしやすさと優れた精度のため、機械学習を不正行為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。

この取り組みにデータブリックス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。プロセスを実現します。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うことができるようになり、これまでにないコラボレーションが可能になりました。このアプローチにより、これまでサイロ化していたチーム間の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。

わずか数分で無料トライアルに申し込んで、この Notebook を試してみて、自分のモデルを作り始めてみてはいかがでしょうか。

> データブリックスの無料の **Notebook** を使って実験を始める

第 8 章　Virgin Hyperloop One 社が Koalas を活用して処理時間を数時間から数分に短縮した方法

Pandas のコードを Apache Spark™ にシームレスに切り替えるためのフィールドガイド

投稿者：
Patryk Oleniuk
Sandhya Raghavan

2019 年 8 月 22 日

Virgin Hyperloop One では、Hyperloop を現実のものとし、航空会社のスピードで乗客や貨物を移動させることができるようにしていますが、そのコストは航空旅行の何分の一かです。商業的に実行可能なシステムを構築するために、私たちは Devloop テストトラックの実行、多数のテストリグ、様々なシミュレーション、インフラ、社会経済データなど、膨大で多様な量のデータを収集し、分析しています。これらのデータを扱うスクリプトのほとんどは、pandas をメインのデータ処理ツールとして、Pythonライブラリを使って書かれており、全てを束ねています。このブログ記事では、Koalas を使ってデータ分析をスケーリングし、わずかなコード変更で大規模なスピードアップを実現した経験を共有したいと思います。

私たちが成長し続け、新しいものを作り続けると、データ処理のニーズも高まります。データ処理の規模と複雑さが増してきたため、pandas ベースの Python スクリプトでは、ビジネスのニーズを満たすには時間がかかりすぎていました。そこで、高速な処理時間と柔軟なデータストレージ、オンデマンドでのスケーラビリティを期待して Spark を採用しました。しかし、pandas スベースのコードベースを PySpark に移行するためには、多くのカスタム変更を行わなければなりませんでした。より高速なだけでなく、理想的にはコードの書き換えを必要としないソリューションが必要でした。このような課題に直面した私たちは、他の選択肢を調査することになりましたが、このような面倒なステップをスキップする簡単な方法があることを発見し、非常に嬉しく思いました。

Koalas の Readme に以下のように記載されています

> Koalas プロジェクトは、Apache Spark の上に pandas DataFrame API を実装することでビッグデータを扱う際のデータサイエンティストの生産性を高めています。
>
> （...）
>
> すでに pandas に精通している場合は、学習曲線なしで Spark を使ってすぐに生産性を上げることができます。
>
> Pandas（テスト、より小さなデータセット）と Spark（分散データセット）の両方で動作する単一のコードベースを持っています。

この記事では、これが（ほとんどの場合）真実であることと、なぜ Koalas が試してみる価値があるのかを示してみたいと思います。Pandas の 1% 未満の行に変更を加えることで、Koalas と Spark を使ってコードを実行することができました。実行時間は数時間から数分と 10 倍以上に短縮できましたし、水平方向へのスケールが可能な環境なので、さらに多くのデータに対応できるようになりました。

クイックスタート

Koalas をインストールする前に、Spark クラスタが設定されていて、PySpark で使用できることを確認してください。その後、実行するだけです。

```
pip install koalas
```

または、conda のユーザーのために

```
conda install koalas -c conda-forge
```

詳しくは Koalas のReadme を参照してください。

```
import databricks.koalas as ks
kdf = ks.DataFrame({'column1':[4.0, 8.0]}, {'column2':[1.0, 2.0]})
kdf
```

見てのとおり、Koalas は pandas のようなインタラクティブなテーブルをレンダリングすることができます。非常に便利です。

基本操作の例

この記事のために、4 列で構成、行数をパラメータ化したテストデータを生成しました。

```
import pandas as pd
## generate 1M rows of test data
pdf = generate_pd_test_data( 1e6 )
pdf.head(3)
>>>      timestamp pod_id trip_id speed_mph
0 7.522523 pod_13 trip_6 79.340006
1 22.029855 pod_5 trip_22 65.202122
2 21.473178 pod_20 trip_10 669.901507
```

免責事項

これはパフォーマンス評価に使用されるランダムに生成されたテストファイルで、Hyperloop のトピックに関連していますが、当社のデータを表すものではありません。この記事に使用されたテストスクリプトの全文はこちらからご覧いただけます。
https://gist.github.com/patryk-oleniuk/043f97ae9c405cbd13b6977e7e6d6fbc

例えば、全てのポッドトリップで、いくつかの重要な記述的分析を評価したいと思います。ポッドトリップ1回あたりの移動時間は？

次のオペレーションが必要です。

1. ['pod_id','trip_id']でグループ化します。
2. トリップごとに、最後のタイムスタンプ - 最初のタイムスタンプとしてtrip_time を計算します。
3. ポッドトリップ時間の分布（平均、stddev）の計算

短くて遅い（Pandas）- スニペット#1

```
import pandas as pd
# take the grouped.max (last timestamp) and join with grouped.min (first
timestamp)
gdf = pdf.groupby(['pod_id','trip_id']).agg({'timestamp': ['min','max']})
gdf.columns = ['timestamp_first','timestamp_last']
gdf['trip_time_sec'] = gdf['timestamp_last'] - gdf['timestamp_first']
gdf['trip_time_hours'] = gdf['trip_time_sec'] / 3600.0
# calculate the statistics on trip times
pd_result = gdf.describe()
```

長くて速い（PySpark）- スニペット#2

```
import pyspark as spark
# import pandas df to spark (this line is not used for profiling)
sdf = spark.createDataFrame(pdf)
# sort by timestamp and groupby
sdf = sdf.sort(desc('timestamp'))
sdf = sdf.groupBy("pod_id", "trip_id").agg(F.max('timestamp').
alias('timestamp_last'), F.min('timestamp').alias('timestamp_first'))
# add another column trip_time_sec as the difference between first and
last
sdf = sdf.withColumn('trip_time_sec', sdf2['timestamp_last'] -
sdf2['timestamp_first'])
sdf = sdf.withColumn('trip_time_hours', sdf3['trip_time_sec'] / 3600.0)
# calculate the statistics on trip times
sdf4.select(F.col('timestamp_last'),F.col('timestamp_first'),F.col('trip_
time_sec'),F.col('trip_time_hours')).summary().toPandas()
```

短くて早い（Koalas）- スニペット#3

```
import databricks.koalas as ks
# import pandas df to koalas (and so also spark) (this line is not used
for profiling)
kdf = ks.from_pandas(pdf)
# the code below is the same as the pandas version
gdf = kdf.groupby(['pod_id','trip_id']).agg({'timestamp':
['min','max']})
gdf.columns = ['timestamp_first','timestamp_last']
gdf['trip_time_sec'] = gdf['timestamp_last'] - gdf['timestamp_first']
gdf['trip_time_hours'] = gdf['trip_time_sec'] / 3600.0
ks_result = gdf.describe().to_pandas()
```

スニペット#1と#3については、コードが全く同じなので、「Sparkの切り替え」はシームレスであることに注意してください。ほとんどのpandasスクリプトについては、import pandas databricks.koalas を pd として変更してみることもできますし、いくつかのスクリプトは、以下に説明する制限事項がありますが、微調整を加えても問題なく動作するものもあります。

結果

全てのスニペットは、同じ pod-trip-times の結果を返すことが確認されています。Pandas と Spark の記述方法と要約方法は、ここで説明したように若干異なりますが、パフォーマンスにはあまり影響しないはずです。

サンプル結果

Cmd 7

```
1 ks_result[['summary',   'trip_time_hours']]
```

Out[105]:

	summary	trip_time_hours
0	count	625
1	mean	0.5761789650162432
2	stddev	0.004673946270277798
3	min	0.5539411756993352
4	25%	0.5739794243951338
5	50%	0.5772501165562476
6	75%	0.5795291941218781
7	max	0.5831203330956781

Command took 0.02 seconds -- by patryk.oleniuk@hyperloop-one.com at 8/8/2019, 1:06:14 PM on ML Analytics Cluster

さらに高度な例：UDF と複雑な操作

今度は同じ DataFrame を使ってさらに複雑な問題に挑戦し、pandas とKoalas の実装の違いを見てみましょう。

目標：ポッドトリップ1回あたりの平均速度を分析する

1. [pod_id','trip id'] でグループ化
2. ポッドトリップごとに，速度（時間）チャートの下の領域を求めて，総移動距離を計算する（ここで説明した方法）
3. グループ化されたdfを timestamp 列でソートする
4. タイムスタンプの差分を計算する
5. 速度との差分を乗算する ーその時間の差分で移動した距離になります。
6. distance_travelled 列を合計するーポッドトリップごとの移動距離の合計が表示されます。
7. 旅行時間を timestamp.last - timestamp.first として計算する。
   （前段落のように）
8. average_speed を distance_travelled /trip time として計算する
9. ポッドトリップ時間の分布（平均、stddev）を計算する

このタスクは、カスタム適用関数とユーザー定義関数（UDF）を使って実装することにしました。

Pandas - スニペット #4

```
import pandas as pd
def calc_distance_from_speed( gdf ):
 gdf = gdf.sort_values('timestamp')
 gdf['time_diff'] = gdf['timestamp'].diff()
 return pd.DataFrame({
  'distance_miles':[ (gdf['time_diff']*gdf['speed_mph']).sum()],
  'travel_time_sec': [ gdf['timestamp'].iloc[-1] - gdf['timestamp'].iloc[0] ]
})
results = df.groupby(['pod_id','trip_id']).apply( calculate_distance_from_speed)
results['distance_km'] = results['distance_miles'] * 1.609
results['avg_speed_mph'] = results['distance_miles'] / results['travel_time_sec'] / 60.0
results['avg_speed_kph'] = results['avg_speed_mph'] * 1.609
results.describe()
```

PySpark - スニペット #5

```
import databricks.koalas as ks
from pyspark.sql.functions import pandas_udf, PandasUDFType
from pyspark.sql.types import *
import pyspark.sql.functions as F
schema = StructType([
 StructField("pod_id", StringType()),
 StructField("trip_id", StringType()),
 StructField("distance_miles", DoubleType()),
 StructField("travel_time_sec", DoubleType())
])
@pandas_udf(schema, PandasUDFType.GROUPED_MAP)
def calculate_distance_from_speed( gdf ):
 gdf = gdf.sort_values('timestamp')
 print(gdf)
 gdf['time_diff'] = gdf['timestamp'].diff()
 return pd.DataFrame({
  'pod_id':[gdf['pod_id'].iloc[0]],
  'trip_id':[gdf['trip_id'].iloc[0]],
  'distance_miles':[ (gdf['time_diff']*gdf['speed_mph']).sum()],
  'travel_time_sec': [ gdf['timestamp'].iloc[-1]-gdf['timestamp'].iloc[0] ]
})
sdf = spark_df.groupby("pod_id","trip_id").apply(calculate_distance_from_speed)
sdf = sdf.withColumn('distance_km',F.col('distance_miles') * 1.609)
sdf = sdf.withColumn('avg_speed_mph',F.col('distance_miles')/ F.col('travel_time_sec') /
60.0)
sdf = sdf.withColumn('avg_speed_kph',F.col('avg_speed_mph') * 1.609)
sdf = sdf.orderBy(sdf.pod_id,sdf.trip_id)
sdf.summary().toPandas() # summary calculates almost the same results as describe
```

Koalas - スニペット #6

```
import databricks.koalas as ks
def calc_distance_from_speed_ks( gdf ) -> ks.DataFrame[ str, str, float , float]:
 gdf = gdf.sort_values('timestamp')
 gdf['meanspeed'] = (gdf['timestamp'].diff()*gdf['speed_mph']).sum()
 gdf['triptime'] = (gdf['timestamp'].iloc[-1] - gdf['timestamp'].iloc[0])
 return gdf[['pod_id','trip_id','meanspeed','triptime']].iloc[0:1]

kdf = ks.from_pandas(df)
results = kdf.groupby(['pod_id','trip_id']).apply( calculate_distance_from_speed_ks)
# due to current limitations of the package, groupby.apply() returns c0 .. c3 column names
results.columns = ['pod_id', 'trip_id', 'distance_miles', 'travel_time_sec']
# spark groupby does not set the groupby cols as index and does not sort them
results = results.set_index(['pod_id','trip_id']).sort_index()
results['distance_km'] = results['distance_miles'] * 1.609
results['avg_speed_mph'] = results['distance_miles'] / results['travel_time_sec'] / 60.0
results['avg_speed_kph'] = results['avg_speed_mph'] * 1.609
results.describe()
```

Koalas の apply の実装は PySpark の pandas_udf をベースにしており、スキーマ情報を必要とするため、関数の定義では型ヒントも定義しなければなりません。パッケージの作者は新しいカスタム型、 ks.DataFrame と ks.Series をヒントを導入しました。残念ながら、現在の apply メソッドの実装はかなり面倒で、同じ結果にたどり着くのに少し苦労しました（列名が変わったり、groupby キーが返ってこなかったり）。しかし、全ての動作はパッケージのドキュメントで適切に説明されています。

パフォーマンス

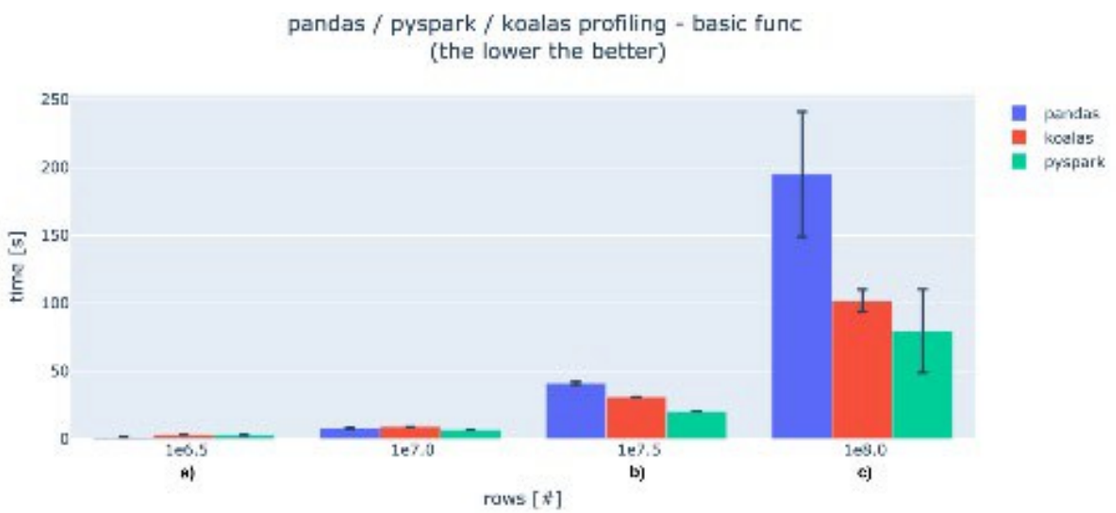
Koalas のパフォーマンスを評価するために、異なる行数のコードスニペットをプロファイリングしました。

プロファイリング実験は、以下のクラスタ構成を使用して、データブリックスのプラットフォーム上で行われました。

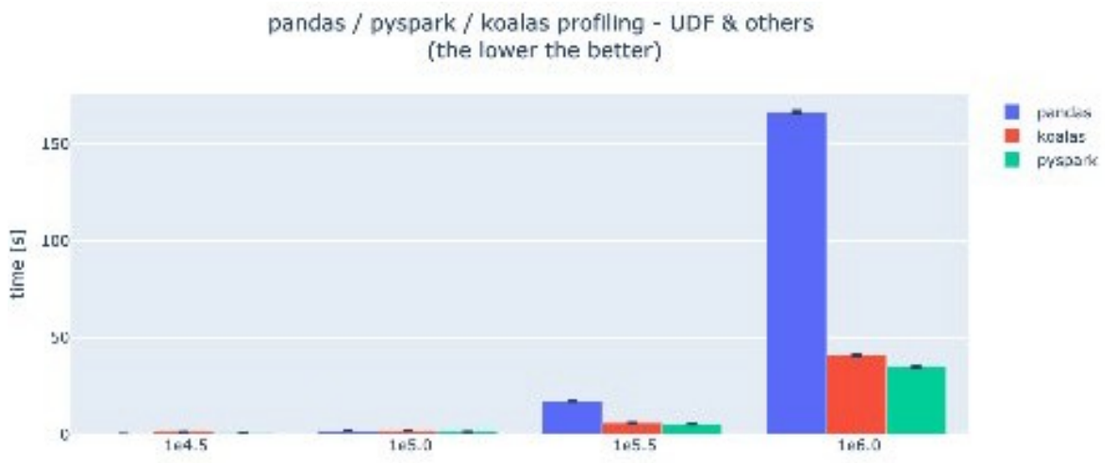
- Spark ドライバノード（pandas スクリプトの実行にも使用されます）：8 CPU コア、61GB RAM
- 15 台の Spark ワーカーノード。4CPUコア、各 30.5 GB RAM（合計：60 CPU/457.5 GB

基本操作

データが小さい場合は、初期化操作やデータ転送が計算に比べて膨大になるため、 pandas の方がはるかに高速です（マーカーa）。データ量が多くなると、pandas の処理時間は分散解を上回ります（マーカーb）。次に、Koalas のパフォーマンスの低下を見ることができますが、データ量が増えるにつれて PySpark に近づいていきます（マーカーc）。



UDF



議論

1つのノードでは処理できないような大きなデータセットでもすぐにスケーラブルに実行できるようにしたい場合には、Koalas が適しているようです。Koalas に素早くスワップした後、Spark クラスタをスケーリングするだけで、より大きなデータセットを許可し、処理時間を大幅に改善することができます。パフォーマンスは PySpark のものと同等（ただし、データセットのサイズやクラスタにもよりますが、5%から50%程度は低くなります）になるはずです。

一方で、Koalas のAPI レイヤーは、特にネイティブの Sparkと比較して、目に見えるパフォーマンスの低下を引き起こします。結局のところ、計算パフォーマンスが重要な優先事項であれば、Python から Scala への切り替えを検討すべきでしょう。

限界と差異

Koalas を使い始めて数時間の間、あなたは "なぜこれが実装されていないのか" と疑問に思うかもしれません。現在、このパッケージはまだ開発中で、いくつかの pandas API 機能が不足していますが、今後数ヶ月のうちに実装される予定です。

（例えば groupby.diff() や kdf.rename() など）

また、プロジェクトへの貢献者としての経験から、いくつかの機能は Spark API で実装するには複雑すぎるか、パフォーマンスが大幅に低下するためにスキップされています。例えば、DataFrame.values は、1つのノードのメモリ内で作業セット全体をマテリアライズする必要があるため、最適ではないし、不可能な場合もあります。 ドライバの最終的な結果を取得する必要がある場合は、 DataFrame.to_pandas() または DataFrame.to_numpy() を使用します。

もう一つ重要なことは、Koalas の実行チェーンが pandas とは違うということです。データフレーム上の操作を実行する際には、操作のキューに入れられますが、実行はされません。結果が必要な場合のみ、例えば kdf.head() や kdf.to_pandas() の操作が実行されます。これは Spark を使ったことがない人にとっては誤解を招くかもしれません。

結論

Koalas のおかげで pandas のコードを「Spark-ify」するための負担を軽減することができました。もしあなたも pandas コードのスケーリングに悩んでいるのであれば、ぜひこちらも試してみてください。もし、どうしても動作が見当たらない、あるいは pandas との矛盾を発見した場合は、コミュニティとしてパッケージが積極的かつ継続的に改善されていくことを保証できるように、ぜひ問題を公開してください。また、気軽に貢献してください。

リソース

1. Koalas GitHub： https://github.com/databricks/koalas
2. Koalasのドキュメント：https://koalas.readthedocs.io
3. この記事からのコードスニペット：https://gist.github.com/patryk-oleniuk/043f97ae9c405cbd13b6977e7e6d6fbc

> データブリックスのこれらの Notebook（pandas と Koalas）は無償でご利用いただけます

第 9 章　データブリックスで Apache Spark を使用したショッピング体験のパーソナライズ

投稿者：Brett Bevers

2017 年 3 月 31 日

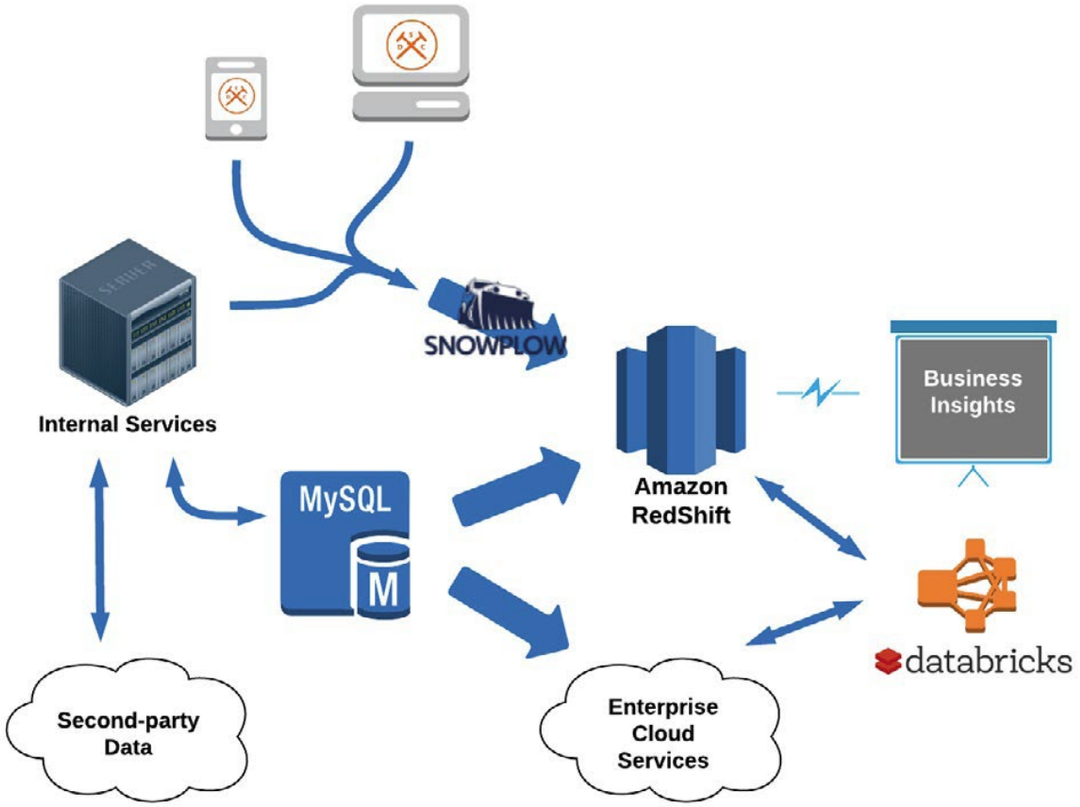
Dollar Shave Club（DSC）は、男性のシェービングやグルーミングのニーズに応える方法を変えることを使命とする、男性向けライフスタイルブランドおよび E コマース企業です。データは、最先端のユーザーエクスペリエンスを実現する上で最も重要な資産であると考えられます。データを通じたパーソナライズされた顧客体験の構築に向けた取り組みにおいて、データブリックスは重要なパートナーとなっています。この記事では、データブリックスのプラットフォームが、強力なカスタム機械学習パイプラインの開発と展開の全ての段階をどのようにサポートしたかを説明します。

DSC の主なサービスは、カミソリカートリッジの月額プランで、会員に直接配送されます。会員の皆様には、1ページの Web アプリまたはネイティブ・モバイル・アプリでご入会いただき、アカウントを管理していただいております。会員の皆様には、ご来店の際に、グルーミング用品やバスルーム用品のカタログをご覧いただくことができます。また、会員様とゲストの皆様には、当クラブの特徴的なスタイルをお楽しみいただくために作成されたオリジナルコンテンツや記事、動画などをお楽しみいただけます。中学時代の保健体育を彷彿とさせない記事で、健康や身だしなみに関する好奇心を満たしていただけます。スタイルや仕事、人間関係のヒントを得ることもできますし、「地球上の文明はいつまで続くのか」というような大きな疑問を DSC が楽しく取り上げた記事を読むこともできます。また、DSC はソーシャルメディア・チャンネルで人々を巻き込むことにも力を入れており、会員は熱心に参加することができます。個々の会員にとって最も関心の高いコンテンツやオファーを見極めることで、より個人的でより良い会員体験を提供することができます。

Dollar Shave Club（DSC）のデータ

DSC で は、会員やゲストとのやりとりにより、膨大なデータが生成されます。このデータが会員の体験を向上させるための資産となることを知っていたため、当社のエンジニアリングチームは、最新のデータインフラストラクチャに早期に投資しました。当社のウェブアプリケーション、内部サービス、データインフラストラクチャは、100% AWS でホストされています。Redshift クラスタが中央のデータウェアハウスとして機能し、さまざまなシステムからデータを受け取ります。レコードは、本番用データベースからウェアハウスに継続的にレプリケートされています。 また、データは、オープンソースのストリーミングプラットフォームである Apache Kafka を介して、アプリケーション間や Redshift に移動します。当社では、高度にカスタマイズ可能なオープンソースのイベントパイプラインである Snowplow を使用して、Web およびモバイルクライアントからイベントデータを収集しています。サーバサイドのアプリケーションを使用しています。クライアントは、ページビュー、リンククリック、ブラウジング活動、および全ての数多くのカスタムイベントとコンテキストを使用しています。データが Redshift に届くと、モニタリング、可視化、洞察のために様々な分析プラットフォームからアクセスされます。

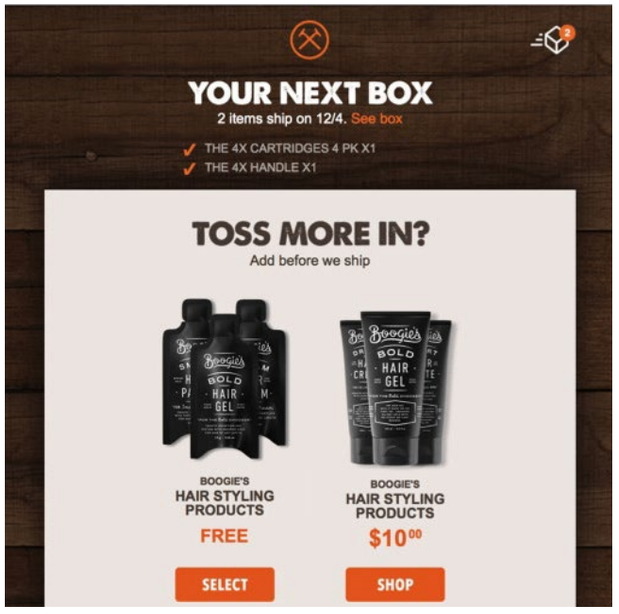
このレベルの可視性があれば、データから学び、それに基づいて行動する機会はいくらでもあります。しかし、これらの機会を特定し、スケールで実行するためには、適切なツールが必要です。ETL、ストリーム処理、機械学習のためのエンジンを備えた最先端のクラスタコンピューティングフレームワークである Apache Spark は、当然の選択です。さらに、データエンジニアリングのためのデータブリックスの最新の開発により、Spark は非常に簡単に使い始めることができ、IDEとデプロイメントパイプラインの両方に適したプラットフォームを提供しています。データブリックスを使った最初の日には、新しいクラスのデータ課題に対応できるようになっていました。



ユースケース：推薦エンジン

私たちがデータブリックスで開発した最初のプロジェクトの一つは、予測モデリングを使用して、私たちが作成した製品の推奨事項を最適化することを目的としています。製品を最適化するためのモデリング 提言 会員の皆様には、特定の メールチャンネル。会員の方には 前週メール が発送されます。 これらのメールは、会員の皆様に近日中の発送予定をお知らせするとともに、箱に入れることができる追加商品をご提案します。会員の方は、数回クリックするだけで、メールに記載されているお勧めの商品を追加することができます。私たちの目標は、特定のメンバーのために、毎月のメールでどの商品をどのような優先順位でプロモーションするかを規定した商品ランキングを作成することでした。

私たちは、各製品に対する会員の関心度を示す傾向のある行動を徹底的に探索することを計画しました。メンバーデータの約 12 のセグメントからさまざまなメトリクスを抽出し、そのデータを何百ものカテゴリ、アクション、タグでピボットし、離散化された時間でイベント関連のメトリクスをインデックス化します。全体では、大規模なメンバーのコホートについて、約 10,000 個の機能を調査範囲に含めることができました。大規模で高次元でスパースなデータセットを扱うために、Spark Core、Spark SQL、Mllib を使って ETL とデータマイニングを自動化することにしました。最終的な製品は、生産データ上で訓練・調整された線形モデルの集合体であり、それらを組み合わせて製品ランキングを作成することができます。



Spark 上で完全自動化されたパイプラインを以下の段階で開発することにしました。

1. 倉庫からデータを抽出する（Redshift）
2. メンバーごとのデータの集計とピボット
3. 最終モデルに含める機能を選択

ステップ 1：データを抽出

私たちはまず、リレーショナル・データベース内のさまざまなデータ・セグメントを見ることから始めます。各データセグメントを理解し、それがどのように解釈され、どのようにクリーンアップされる必要があるのかを理解する必要があります。この作業は、データとそのライフサイクルに関する専門知識と制度的な知識を結集して行う重要な作業であるため、その過程で学んだことを文書化して伝えることが重要です。データブリックスは Spark シェルへの "Notebook "インターフェースを提供しています。

Spark のプログラミングモデルを使いながら、インタラクティブにデータを操作することができます。Spark の Notebook は、アイデアを試してみたり、すぐに結果を共有したり、後で参照できるように作業の記録を残しておくのに最適なことがわかりました。

各データセグメントについて、レコードのクリーニングと非正規化の仕様を抽出モジュールにカプセル化します。多くの場合、Redshift からテーブルをエクスポートし、動的にSQL クエリを生成し、Spark SQL に任せればよいのです。

必要に応じて、Spark の DataFrames API を使った機能的なプログラミングをきれいに導入することができます。 そして、ドメイン固有のメタデータのアプリケーションは、抽出器の中に自然な形で存在しています。 重要なのは、特定のデータセグメントを処理するための最初のステップが、他のセグメントやパイプラインの他のステージからきれいに分離されていることです。抽出器は独立して開発し、テストすることができます。また、他の探索や生産パイプラインに再利用することもできます。

```
def performExtraction(
extractorClass, exportName, joinTable=None, joinKeyCol=None,
startCol=None, includeStartCol=True, eventStartDate=None
):
customerIdCol = extractorClass.customerIdCol
timestampCol = extractorClass.timestampCol
extrArgs = extractorArgs(
customerIdCol, timestampCol, joinTable, joinKeyCol,
startCol, includeStartCol, eventStartDate
)
Extractor = extractorClass(**extrArgs)
exportPath = redshiftExportPath(exportName)
return extractor.exportFromRedshift(exportPath)
```

データ抽出パイプラインのコード例：パイプラインは、いくつかの抽出クラスによって実装されたインターフェイスを使用し、動作をカスタマイズするために引数を渡します。パイプラインは、各抽出の詳細には依存しません。

```
def exportFromRedshift(self, path):
export = self.exportDataFrame()
writeParquetWithRetry(export, path)
return sqlContext.read.parquet(path)
.persist(StorageLevel.MEMORY_AND_DISK)

def exportDataFrame(self):
self.registerTempTables()
query = self.generateQuery()
return sqlContext.sql(query)
```

エクストラクターインターフェイスのコード例：多くの場合、エクストラクタは単に SQL クエリを生成して SparkSQL に渡すだけです。

ステップ 2：集計とピボット

ウェアハウスから抽出されたデータは、ほとんどが個々のイベントや関係性に関する詳細な情報です。しかし、私たちが本当に必要としているのは、ある製品または別の製品への関心を示す行動を効果的に検索できるように、時間の経過とともに集約されたアクティビティの説明です。特定のイベントタイプを秒単位で比較しても、実りのあるものではありません。このレベルの粒度では、データがあまりにもまばらで、機械学習の良い材料にはなりません。まず最初にすべきことは、イベント関連のデータを離散的な期間にわたって集約することです。イベントをカウント、合計、平均、頻度などに還元することで、以下のようなことが可能になります。メンバー間の比較がより意味のあるものになり、データマイニングがより容易になります。もちろん、同じイベントのセットを時間だけでなく、いくつかの異なる次元で集計することもできます。これらの数値のどれか、または全てが、興味深いストーリーを物語っているかもしれません。

データセット内の複数の属性のそれぞれを集約することは、しばしばデータをピボット（またはロールアップ）することと呼ばれています。メンバーごとにデータをグループ化し、時間やその他の興味深い属性でピボットすると、データは個々のイベントや関係性に関するデータから、メンバーを記述する特徴の（非常に長い）リストに変換されます。各データセグメントについて、データを意味のある方法でピボットするための具体的な方法を以下のようにカプセル化しています。

自身のモジュールを使用しています。これらのモジュールをトランスフォーマーと呼びます。ピボットされたデータセットは非常に広範囲になることがあるため、DataFrames よりも RDD を使用した方がパフォーマンスが高いことがよくあります。一般的には，ピボットされた特徴量の集合をスパースベクタ形式で表現し，キーバリュー RDD 変換を用いてデータを削減しています．メンバーの振る舞いを疎なベクトルで表現することで，メモリ上のデータセットのサイズを小さくすることができ，また，パイプラインの次の段階で MILib で使用するための訓練セットを簡単に生成することができます。

ステップ 3：データマイニング

この時点では、各会員の機能が非常に膨大な量になっているので、各製品について、それらの機能のどのサブセットが、会員がその製品の購入に興味を持っているかを示すのに最適なのかを判断したいと考えています。これはデータマイニングの問題です。考慮すべき特徴量が少なければ（例えば、数千ではなく数十であれば）、いくつかの合理的な方法で進めることができます。しかし、考慮される特徴の数が多いということは、特に難しい問題です。データブリックスのプラットフォームのおかげで、私たちはこの問題に膨大な計算時間を簡単にかけることができました。私たちは、比較的小さく、ランダムにサンプリングされた特徴量のセットに対してモデルを学習し、評価する方法を用いました。数百回の繰り返しの中で、それぞれが高性能モデルに大きく貢献する特徴のサブセットを徐々に蓄積していきます。モデルを訓練し、そのモデル内の各特徴の評価統計量を計算するには計算コストがかかります。しかし、大規模な Spark クラスタをプロビジョニングして各製品の作業を行い、作業が終了したら終了させることは問題ありませんでした。

データマイニングの進捗状況を把握できることが不可欠です。プロセスが最高性能のモデルに収束するのを妨げているバグやデータ品質の問題がある場合は、処理時間を何時間も無駄にしないように、できるだけ早く発見する必要があります。そのために、各イテレーションで収集された評価統計を可視化するシンプルなダッシュボードをデータブリックス上で開発しました。

最終モデル

MLlib の評価モジュールは、モデルのパラメータの調整を非常に簡単にします。 ETLとデータマイニングのハードワークが完了すれば、最終的なモデルの作成はほぼ簡単です。最終的なモデルの係数とパラメータを決定した後、本番での製品ランキングの生成を開始しました。データブリックスのスケジューリング機能を使用して、その日にメール通知を受け取るメンバーのそれぞれの製品ランキングを生成するために、毎日ジョブを実行しました。各メンバーの特徴ベクトルを生成するために、オリジナルのトレーニングデータを生成したのと同じ抽出器と変換器モジュールを最新のデータに適用するだけです。これにより、開発時間を前倒しで短縮できるだけでなく、探査パイプラインと生産パイプラインの二重保守の問題を回避することができます。また、トレーニングデータと正確に同じ意味と文脈を持つ特徴に対して、モデルが最も有利な条件で適用されていることを保証します。

データブリックスと Apache Spark を使った今後の予定

この製品推奨プロジェクトは大成功を収め、DSC でも同様の意欲的なデータプロジェクトに取り組むようになりました。データブリックスは、特にデータ製品の開発ワークフローをサポートする上で重要な役割を果たし続けています。大規模なデータマイニングは、戦略的に重要な問題に対処するための情報収集に欠かせないツールとなっており、その結果得られた予測モデルは、本番でのスマート機能を強化するために展開することができます。機械学習に加えて、Spark Streaming 上に構築されたストリーム処理アプリケーションを採用したプロジェクトもあります。例えば、様々なイベントストリームを消費してメトリクスの収集やレポートを簡単に作成したり、システム間のデータをほぼリアルタイムで複製したりしています。そしてもちろん、Spark 上で開発されているETLプロセスも増えてきています。

第 10 章　データブリックスで Apache SparkR を使用した大規模なシミュレーションの並列化

投稿者：
Wayne W. Jones
Dennis Vallinga
Hossein Falaki

2017 年 6 月 23 日

序章

Apache Spark 2.0 では、既存のR関数を並列化できるようにするための Apache Spark の R インターフェースである SparkR に新しいAPIファミリーが導入されました。新しい dapply, gapply, spark.lapply メソッドはRユーザーにエキサイティングな可能性をもたらします。この記事では、Shell Oil Company とデータブリックスが共同で行ったユースケースの詳細を紹介します。

ユースケース：在庫の推薦

シェルでは、現在の在庫管理は、ベンダーの推奨事項、過去の運用経験、および「直感」の組み合わせに よって行われていることが多く、そのため、これらの決定に過去のデータを取り入れることは限られており、その結果、シェルの拠点（石油リグなど）で在庫が過剰または不十分になることがあります。

プロトタイプツールである Inventory Optimization Analytics ソリューションは、シェルが SAP の在庫データ上で高度なデータ分析技術を使用して、次のことができることを証明しています。

- 倉庫のインベントリレベルを最適化
- 安全在庫水準の見通し
- 動きの遅い材料を合理化する
- 材料リストの非在庫品・在庫品の見直しと再割り当て
- 材料の重要性を特定する（例：部品表のリンク、過去の使用状況またはリードタイムを介して）

データサイエンスチームは、材料の推奨インベントリレベル要件を計算するために、マルコフ連鎖モンテカルロ（MCMC）ブートストラップ統計モデルをRに実装しました。個々の材料モデルでは、過去の発行物の分布を把握するために、10,000回の MCMC の反復シミュレーションを行います。

累積的には、計算タスクは大きいですが、幸いなことに、モデルはそれぞれの材料に独立して適用することができるので、恥ずかしいほど並列的な性質のものです。

既存の設定

現在、フルモデルは48コア、192GB RAMのスタンドアロン物理オフラインPCで実行されています。MCMCブートストラップモデルは、多数のサードパーティ製Rパッケージを使用したカスタムビルドされた機能セットです。

```
(“fExtremes”, “ismev”, “dplyr”, “tidyr”, “stringr”).
```

このスクリプトは、シェルの各ロケーションを反復処理し、歴史的資料を48個のコアにまたがって、ほぼ同じ大きさの資料グループに分散させます。その後、各コアは、個々のマテリアルにモデルを反復的に適用します。マテリアルをグループ化しているのは、各マテリアルの単純なループでは、各計算に2〜5秒かかるため、オーバーヘッド（Rプロセスの開始など）が大きくなりすぎるからです。コア間の材料グループジョブの分散は、R並列パッケージを介して実装されています。

個々の48のコアジョブの最後の処理が完了すると、スクリプトは次のロケーションに移動し、プロセスを繰り返します。このスクリプトは、シェルの全てのロケーションの推奨インベントリレベルを計算するのに合計約48時間かかります。

データブリックスでApache Sparkを使う

Shellは、多くのコアを持つ単一の大規模なマシンに頼るのではなく、クラスタコンピューティングを利用してスケールアウトすることにしました。Apache Sparkの新しいR APIは、このユースケースに適していました。SparkRのスケーラビリティとパフォーマンスを検証するために、2つのバージョンのワークロードをプロトタイプとして開発しました。

プロトタイプI：概念実証（PoC）

最初のプロトタイプでは、新しいSparkR API がワークロードを処理できるかどうかを素早く検証するために、コードの変更を最小限に抑えました。全ての変更をシミュレーションステップに限定したのは以下のとおりです。

各シェルのロケーションリスト要素に対して：

1. 入力された日付をSpark DataFrameとして並列化する
2. SparkR::gapply()を使用して、チャンクごとに並列シミュレーションを行う

既存のシミュレーションコードベースを限定的に変更することで、データブリックス上の50 ノードのSparkクラスタでの総シミュレーション時間を3.97時間に短縮することができました。

プロトタイプII：性能向上

最初のプロトタイプはすぐに実装できましたが、1つの明らかなパフォーマンスのボトルネックがありました。それは、シミュレーションのイテレーションごとにSparkジョブが起動されることです。データは非常に偏っており、その結果、各ジョブの間、ほとんどの実行者は、次のジョブから作業を引き継ぐ前に、はぐれ者が終了するまでアイドル状態で待機しています。さらに、各ジョブの開始時には、クラスタ上のほとんどのCPUコアがアイドル状態の間、Spark DataFrameとしてデータの並列化に時間を費やしています。

これらの問題を解決するために、前処理ステップを変更して、全ての場所と材料の値の入力と補助的な日付を前もって生成するようにしました。入力データは大きなSpark DataFrameとして並列化しました。次に、場所IDと材料IDの2つのキーを持つ単一のSparkR::gapply()コールを使用してシミュレーションを実行しました。

これらの簡単な改良により、データブリックス上の50ノードのSparkクラスタでシミュレーション時間を45分に短縮することができました。

SparkRの改良点

SparkRはApache Sparkの最新の追加機能の一つであり、本作業の時点ではapply APIファミリがSparkRの最新の追加機能となっていました。今回の実験を通して、SparkRのいくつかの制限やバグを特定し、Apache Sparkで修正しました。

- [SPARK-17790]2GB以上のR data.frameの並列化に対応
- [SPARK-17919]SparkRでRbackendへのタイムアウトを設定できるように
- [SPARK-17811]SparkRでは、Date列にNAまたはNULLがある場合にdata.frameを並列化できない

次のステップ

SparkR開発者の方で、SparkRに興味がある方は、データブリックスのアカウントを取得して、SparkRのドキュメントをご覧ください。

第 11 章　導入事例

数百万人の顧客をパーソナライズされた体験につなげるグローバルなテクノロジーとメディア企業である Comcast は、膨大なデータ、壊れやすいデータパイプライン、貧弱なデータサイエンスのコラボレーションに苦戦していました。Delta Lake や MLflow を含むデータブリックスを使用することで、ペタバイト級のデータに対応したパフォーマンスの高いデータパイプラインを構築し、何百ものモデルのライフサイクルを簡単に管理することができ、音声認識と機械学習を活用した非常に革新的でユニークな視聴体験を生み出し、受賞歴のある視聴者体験を実現しました。

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、一時停止ボタンを押している暇はありません。コムキャストは、データの取り込みから機械学習モデルの展開に至るまで、アナリティクスへのアプローチ全体を近代化する必要があることに気付きました。

ソリューション とメリット

コムキャストは、アナリティクスへの統一的なアプローチにより、AI を活用したエンターテインメントの未来に向けて前進することができ、競合他社に負けない顧客体験で視聴者を魅了し、満足させ続けることができます。

- **エミー賞受賞の視聴者エクスペリエンス：**コムキャストは、エンゲージメントを高めるインテリジェントな音声コマンドを使って、非常に革新的で受賞歴のある視聴者体験を実現することができます。
- **データ処理コストを 10 倍削減：**Delta Lake は、パフォーマンスを向上させながら 640 台のマシンを 64 台に置き換え、データインジェストを最適化することを可能にしました。チームは分析に多くの時間を割くことができ、インフラ管理にかける時間を減らすことができます。
- **高度なデータサイエンスの基盤：**Delta Lake のアップグレードと使用により、単一のインタラクティブなワークスペースで異なるプログラミング言語を使用できるようになり、データサイエンティスト間のグローバルなコラボレーションが促進されました。また、Delta Lake はデータチームがデータ パイプライン内の任意のポイントでデータを使用することを可能にし、新しいモデルの構築やトレーニングをより迅速に行うことができるようになりました。
- **モデルの展開の高速化 ：** Comcast は近代化により、運用チームが異種プラットフォーム上にモデルを展開する際の展開時間を数週間から数分に短縮しました。

> データブリックスを使うことで、より多くの情報を得ることができ、より迅速な意思決定が可能になりました。
>
> コムキャスト社
> プロダクト分析・行動科学シニアディレクター
> Jim Forsythe 氏

詳しく見る

第 11 章　導入事例

REGENERON

> データブリックスの統合データ解析プラットフォームは、医療サイエンティストから計算生物学者まで、統合医薬品開発に関わる全ての人が、私たちの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しています。
>
> リジェネロン社
> ゲノム情報学部長
> Jeffrey Reid 博士

レジェネロンの使命は、ゲノムデータの力を利用して、必要な患者さんに新しい薬を届けることです。しかし、このデータを人生を変える発見や標的となる治療法に変換することは、これまで以上に困難なことではありませんでした。処理性能が低く、スケーラビリティにも限界があるため、データチームはペタバイト級のゲノムデータや臨床データを解析するために必要なものが不足していました。データブリックスは、ゲノムデータセット全体を迅速に解析し、新しい治療法の発見を加速させることができるようになりました。

ユースケース

現在、医薬品開発のパイプラインにある全ての実験薬の 95% 以上が失敗すると予想されています。これらの取り組みを改善するために、Regeneron Genetics Center は、40 万人以上の人々の配列決定されたエクソームと電子カルテをペアにして、最も包括的な遺伝学データベースの1つを構築しました。しかし、この膨大なデータセットの解析には多くの課題がありました。

- ゲノムデータや臨床データは非常に分散化されているため、10 TB のデータセット全体に対して分析やモデルのトレーニングを行うことは非常に困難です。
- 800 億以上のデータポイントで分析をサポートするために、レガシーアーキテクチャをスケールアップするのは困難でコストがかかります。
- データチームは、分析に使用できるようにデータをETLしようとするだけの日々を過ごしていました。

ソリューション とメリット

データブリックスは、データサイエンスの生産性を向上させることで、業務を簡素化し、創薬を加速させる、Amazon Web Services 上で動作する統合データ分析プラットフォームを レジェネロンに提供します。これにより、これまで不可能だった新しい方法でデータを分析することが可能になります。

- **加速化された医薬品の標的特定：** 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のクエリを実行するのにかかる時間を 30 分から 3 秒に短縮し、600 倍の改善を実現しました。
- **生産性の向上 ：** コラボレーションの改善、自動化された DevOps、パイプラインの高速化（ETL を 3 週間から 2 日で完了）により、チームはより広範な研究をサポートできるようになりました。

詳しく見る

第 11 章　導入事例

データの利用可能性が爆発的に増加し、市場競争が激化する中、保険会社は顧客により良い価格設定を提供することに挑戦しています。Nationwide 社では、ダウンストリーム ML のために何億件もの保険記録を分析する必要があるため、従来のバッチ分析プロセスでは時間がかかり、精度が低く、保険金請求の頻度や重症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。データブリックスを導入したことで、ディープラーニングモデルを大規模に採用して、より正確な価格予測を行うことが可能になり、保険金請求からの収益を増やすことができました。

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、保険金請求の頻度が低く、予測不可能な変動性のある保険記録を分析しなければならず、結果的に不正確な価格設定になってしまうというデータの課題がありました。

ソリューショ ンとメリット

Nationwide 社では、Databricks Unified Data Analytics Platform 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。完全に管理された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のデータ駆動の新たな機会が生まれました。

- **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 時間未満に改善され、パフォーマンスが 9 倍向上しました。
- **より迅速な機能化：**データエンジニアリングは、5 時間から 20 分程度までの15倍の速さで特徴を識別することができます。
- **高速なモデルトレーニング：**トレーニング時間を 50% 短縮し、新しいモデルの市場投入までの時間を短縮しました。
- **改良されたモデルスコアリング：**モデルのスコアリングを 3 時間から 5 分未満に加速し、60 倍の改善を実現しました。

> データブリックスを使用することで、全てのデータに対してモデルをより迅速にトレーニングすることができるようになり、結果として、より正確な価格予測が可能になり、収益に大きな影響を与えました。
>
> ネーションワイド社
> データサイエンティスト
> Bryn Clark 氏

詳しく見る

第 11 章　導入事例

CONDÉ NAST

> データブリックスは、非常に強力なエンドツーエンドのソリューションです。データブリックスの導入によって、さまざまなバックグラウンドを持つチームメンバーが素早く大量のデータにアクセスして活用し、実行可能なビジネス上の意思決定を行え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

コンデナストは世界有数のメディア企業であり、そのポートフォリオにはニューヨーカー、ワイアード、ヴォーグなどの最も象徴的な雑誌タイトルが含まれています。同社はデータを利用して、紙媒体、オンライン、ビデオ、ソーシャルメディアで10億人以上にリーチしています。

ユースケース

大手メディア出版社として、コンデナストは 20 以上のブランドをポートフォリオとして管理しています。毎月 1 億人以上の訪問者と 8 億人以上のページビューを獲得し、膨大な量のデータを生み出しています。データチームは、機械学習を利用してパーソナライズされたコンテンツの推奨やターゲットを絞った広告を提供することで、ユーザーのエンゲージメントを向上させることに注力しています。

ソリューショ ンとメリット

データブリックスはコンデナストに完全に管理されたクラウドプラットフォームを提供し、業務を簡素化し、優れたパフォーマンスを提供し、データサイエンスの革新を可能にしています。

- **顧客満足度の向上：**データパイプラインの改善により、コンデナストはより良い、より迅速、より正確なコンテンツレコメンデーションを行い、ユーザーエクスペリエンスを向上させることができます。
- **スケーラビリティ：**データセットはもはやコンデナストの処理能力や洞察を得る能力を超えることはできません。
- **より多くのモデルを開発中：** MLflow を使用することで、コンデナストのデータサイエン スチームはより迅速に製品を革新することができます。これまでに 1,200 以上のモデルを導入しています。

詳しく見る

第 11 章　導入事例

SHOWTIME® は、「シェイムレス」、「ホームランド」、「ビオンズ」、「ザ・チ」、「レイ・ドノバン」、「SMILF」、「ザ・アフェア」、「パトリック・メルローズ」、「アワー・カートゥーン・プレジデント」、「ツイン・ピークス」などの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 のデータ戦略チームは、組織全体のデータとアナリティクスの民主化に注力しています。彼らは、膨大な量の加入者データ（例：視聴した番組、時間帯、使用したデバイス、加入履歴など）を収集し、機械学習を利用して加入者の行動を予測し、スケジューリングや番組編成を改善しています。

ソリューショ ンとメリット

データブリックスは、SHOWTIME が組織全体でデータと機械学習を民主化し、よりデータドリブンな文化を創造するのに役立っています。

- **パイプラインが 6 倍高速化：**24 時間以上かかっていたデータパイプラインが 4 時間以内に実行されるようになり、チームはより迅速な意思決定が可能になりました。
- **インフラの複雑性を取り除く：**クラウド上で完全に管理されたプラットフォームを自動クラスター管理することで、データサイエンスチームはハードウェアの設定、クラスターのプロビジョニング、デバッグなどではなく、機械学習に集中することができます。
- **サブスクライバーのエクスペリエンスの革新：**データサイエンスのコラボレーションと生産性の向上により、新しいモデルや機能の市場投入までの時間が短縮されました。チームはより迅速に実験を行うことができ、加入者にとってより良い、よりパーソナライズされた体験を提供できるようになりました。

> データブリックス のプラットフォームを導入したことで、かつて直面していたシステム構成上の問題が全て解消されました。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ました。
>
> 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

詳しく見る

第 11 章　導入事例

> データブリックスは、シェルにとって非常に大きな価値を生み出しました。インベントリ最適化ツール（データブリックス上に構築された）は、私の組織から生まれた最初のスケールアップしたデジタル製品であり、グローバルに展開されているという事実は、現在、毎年数百万ドルのコスト削減を実現していることを意味します。
>
> シェル社
> 高度分析 CoE ゼネラルマネージャー
> Dniel Jeavons 氏

シェルは、石油・ガス探査および生産技術のパイオニアであり、世界有数の石油・天然ガス生産者、ガソリンおよび天然ガスのマーケティング業者、石油化学メーカーです。

ユースケース

生産を維持するために、シェル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います。操業停止を回避するためには、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が、同様に重要なのは、コストがかさむ可能性のある過剰な在庫を持たないことです。

ソリューショ ンとメリット

データブリックスは、シェルにクラウドネイティブの統合分析プラットフォームを提供し、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ます。

- **予測モデル化 ：** 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 以上の場所で展開しています。
- **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しています。
- **パフォーマンスの向上 ：**データサイエンスチームはパフォーマンスの向上に重点を置き、インベントリの分析と予測にかかる時間をデータブリックス上の50ノードの Apache Spark™ クラスターで48時間から 45 分に短縮し、パフォーマンスを 32 倍に向上させました。
- **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 11 章　導入事例

Riot Games の目標は、世界で最もプレイヤーにフォーカスしたゲーム会社になることです。2006 年に設立され、LAに拠点を置く Riot Games は、League of Legends ゲームで最もよく知られています。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罵声に対抗することで、ゲーム体験を向上させます。

ソリューショ ンとメリット

データブリックスは、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。

- **ゲーム内での購入経験の向上：**500 B以上のデータポイントに基づいてユニークなオファーを提供するレコメンドエンジンを迅速に構築し、生産することが可能。ゲーマーは欲しいコンテンツをより簡単に見つけることができるようになりました。
- **ゲームラグの減少：**ネットワークの問題をリアルタイムで検出するMLモデルを構築し、Riot Games がプレイヤーに悪影響を及ぼす前に停止を回避できるようにしました。
- **分析の高速化：** データ準備と探査の処理性能をEMRと比較して 50%向上させ、解析を大幅に高速化しました。

> 私たちはデータサイエンティストをクラスター管理から解放したいと考えていました。簡単にデータブリックス で管理されたSparkソリューションを使用することで、このようなことが可能になりました。これで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

詳しく見る

第 11 章　導入事例

Quby

> データブリックスは、Delta Lake と構造化されたストリーミングの力を介して、私たちはアラートを提供することができます。これにより、お客様は快適性に影響が出る前に問題に対応したり、家庭内の調整を行ったりすることができます。
>
> クビー社
> データサイエンティスト責任者
> Stephen Galsworthy 氏

Quby は、人々にエネルギー使用量、快適性、家庭のセキュリティなどを制御するスマート・エネルギー管理デバイスである Toon の背後にあるテクノロジー企業です。Quby のスマートデバイスは数百種類に及びます。

ヨーロッパ全土の何千もの家庭で使用されています。そのため、同社は、家庭内のあらゆる電化製品のセンサーから収集したペタバイトの IoT データで構成されるヨーロッパ最大のエネルギーデータセットを維持しています。このデータを利用して、顧客がより快適な生活を送ることができるように支援するとともに、パーソナライズされたエネルギー使用方法を提案することでエネルギー消費量を削減することを使命としています。

ユースケース

パーソナライズされたエネルギー使用の推奨。機械学習と IoT データを活用して、家庭内のエネルギー消費を削減するためのパーソナライズされた推奨事項を提供する「Waste Checker」アプリを開発しました。

ソリューショ ンとメリット

データブリックスは Quby に統合データ分析プラットフォームを提供しており、データサイエンスとエンジニアリングにまたがるスケーラブルで協調的な環境を醸成し、データチームがより迅速にイノベーションを起こし、ML を活用したサービスを Quby の顧客に提供することを可能にしています。

- **コスト削減：** データブリックスが提供するコスト削減機能（自動スケーリング・クラスターや Spot インスタンスなど）により、Quby は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することができるようになった。
- **イノベーションの加速：** レガシーアーキテクチャでは、概念実証から本番までに12ヶ月以上を要していました。現在、データブリックスを使用すると、同じプロセスが8週間以内で完了する。これにより、Quby のデータチームは顧客のために ML を活用した新機能をより迅速に開発することが可能になった。
- **エネルギー消費量の削減：** Quby は、廃棄物チェッカーアプリを通じて、パーソナライズされたレコメンデーションを活用することで節約可能な 6,700 万キロワット時以上のエネルギーを特定した。

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データとAIの会社です。Comcast、Condé Nast、Nationwide、H&Mなど、世界中の数千もの企業が、データエンジニアリング、機械学習、分析のためのデータブリックスのオープンで統一されたプラットフォームを利用しています。データブリックスは、サンフランシスコに本社を置くベンチャー企業で、世界中にオフィスを構えています。Apache Spark™、Delta Lake、MLflowの創始者によって設立されたデータブリックスは、データチームが世界で最も困難な問題を解決できるように支援することを使命としています。

データブリックスのSNSでも詳しい情報を公開しています。

Twitter

LinkedIn

Facebook

データブリックスの無料お試し

ご相談・お問い合わせ