# オープンソースによるMLOps環境の構築

本多　啓介　　URA

## 1. 機械学習（ML）のための継続的デリバリー（CD）と自動化パイプライン

　データサイエンティストにとってトレーニングデータを用いて一定の予測性能のMLモデルをオフラインで実装することは比較的容易である。しかし実際の課題は統合されたMLシステムを構築し、本番環境で継続的に運用することである。機械学習の成果を継続的にアプリケーション（システム）で利用する際の課題解決のため、ソフトウェア分野での開発手法・文化であるDevOpsを応用したMLOpsが提案されている[1]。MLOpsを実践するとシステム開発（Dev）とシステムオペレーション（Ops）が統合され、トレーニング、テスト、リリース、インフラ管理などすべてのステップで自動化とモニタリングが推進される[2]。Google社の定義によれば自動化のレベルは0～2の3段階に区分される。MLOps レベル2はすべてのステップが自動パイプライン化され、新しいソースコード、モデル、トレーニングデータが投入されると、ビルド、テスト、実行形式のデプロイが行われる（継続的インテグレーション、CI）。さらに新しい予測サービスの本番環境への投入が半自動で行われ、その経過がモニタリングされる（継続的デリバリー、CD）。

## 2. オープンソースによるMLOps環境の構築:Level 1

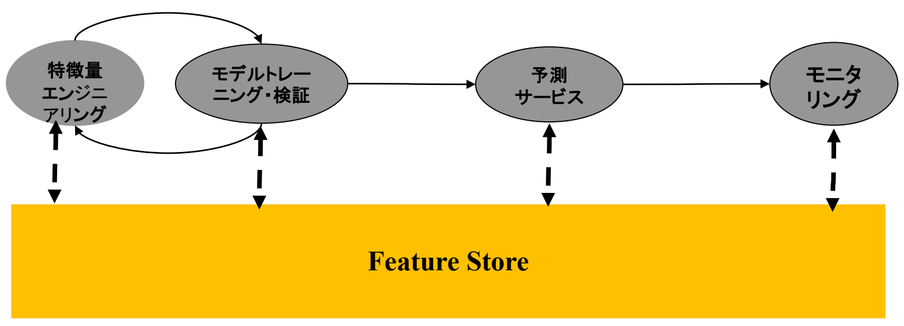
　MLOps環境を段階的に構築するため、まずモデルの継続的トレーニング処理を自動パイプライン化する。必要となるソフトウェアは特徴量ストアとデータフローを記述する統合プログラミングモデルである。

## 3. 特徴量ストア：Feast

モデル開発者とエンジニアリングのチームがMLシステムで生のデータから特徴量データに変換し、本番環境にデプロイするまでの工程を一貫して品質管理するには以下のような課題がある。

・生データから特徴量データへの適切な変換と再利用
・トレーニング時と本番環境での特徴量のズレの検知
・本番環境でのモニタリング

Feast [3]はこれらの課題に対処し、検索、共有、再利用を簡単に行うために専用にデザインされたオープンソースの特徴量ストアである。




たとえば、書誌データの分析システムで、論文の直近5年の引用数によって分野の注目度を測るサービスを機能として実装するとき、書誌データのIDと引用数の生データから特徴量が生成される。そして予測サービスとして用いられる学習結果を格納するカラムを持つ以下のようなテーブルが考えられる。

| 書誌ID | cite_count | size_5y/cite_count | hot_area? |
|---|---|---|---|
| 1001 | 10 | 0.8 | False |
| 1002 | 5 | 3.0 | True |
| 1003 | 11 | 7.0 | True |

書誌データIDと引用数の生データから特徴量を生成し、注目度予測のバッチ処理に提供する一連の処理のコードは以下のようになる。

```
from datetime import datetime
import pandas as pd

from feast import FeatureStore

entity_df = pd.DataFrame.from_dict(
    {

        "paper_id": [1001, 1002, 1003],
        "cited_count": [5, 7, 11],
    }
)

store = FeatureStore(repo_path=".")
training_df = store.get_historical_features(
    entity_df=entity_df,
    features=[
        "size_5y_cite_count:rate"
    ],
).to_df()

print("----- Feature schema -----¥n")
print(training_df.info())
```

次に、生成された特徴量をから注目度予測のオンライン処理のためのビューを取得するコードと実行結果は以下のようになる。

```
from pprint import pprint
from feast import FeatureStore
feature_store = FeatureStore('.')  # Initialize the feature store

feature_service =
feature_store.get_feature_service("document_area_v1")
feature_vector = feature_store.get_online_features(
    features=feature_service,
    entity_rows=[
        # {join_key: entity_value}
        {"document_id": 1001},
        {"document_id": 1003},
    ],
).to_dict()
pprint(feature_vector)
```

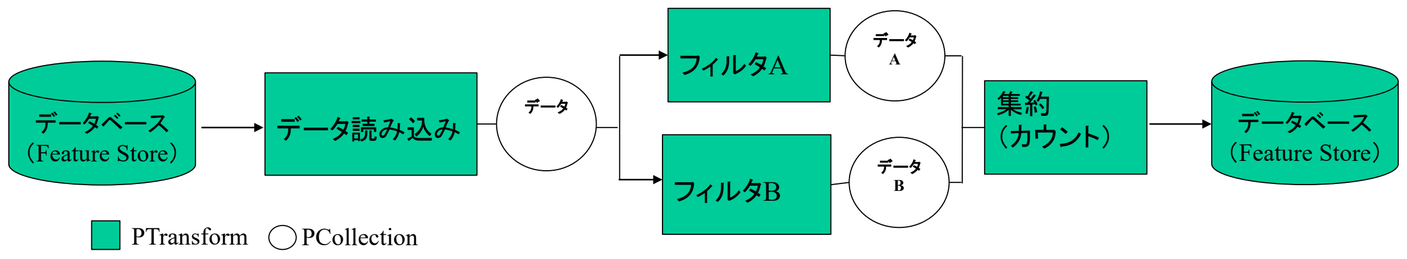
```
{
'size_5Y_cite_count': [0.8732735991477966, 8..0028438878059387],
'hot_are': [false, true],
'driver_id': [1001, 1003]
}
```

## 4. データフロー記述（並列パイプラインモデル）：Apache Beam

　特徴量ストアの導入により、特徴量生成からトレーニング、検証、本番環境での取り出しまでMLシステム内で共有、再利用する仕組みが導入できる。次にこれらのステップを自動化されたデータフローとして記述するプログラミングモデルを導入する。Apache Beam [4]は以下のようにバッチ、ストリーム処理を単純化、パイプライン処理として抽象化する機能を提供する。

・Pipeline : データ処理を有向非巡回グラフとしてパイプライン化する機能
・PCollection : データをParallel処理できるオブジェクトとして抽象化する
・PTransform : 受け取ったPCollectionに対する変換（マップ、フィルタ、集約） 機能
・IO Transform :外部データの読み書き機能を提供する

　以下は外部データの読み込みから変換、外部データへの書き込みまでの一連のデータフローをモデル化したイメージである。任意のタイミングで柔軟に並列処理したり集約の処理が記述でき、下位の分散環境の設定から離れ論理構造に集中できるのが特徴である。




以下はテキストファイルを読み込み、ワードカウント結果を出力するサンプルコードである。

```
# We use the save_main_session option because one or more DoFn's in this
  # workflow rely on global context (e.g., a module imported at module level).
  pipeline_options = PipelineOptions(pipeline_args)
  pipeline_options.view_as(SetupOptions).save_main_session = save_main_session

  # The pipeline will be run on exiting the with block.
  with beam.Pipeline(options=pipeline_options) as p:

    # Read the text file[pattern] into a PCollection.
    lines = p | 'Read' >> ReadFromText(known_args.input)

    counts = (
        lines
        | 'Split' >> (beam.ParDo(WordExtractingDoFn()).with_output_types(str))
        | 'PairWithOne' >> beam.Map(lambda x: (x, 1))
        | 'GroupAndSum' >> beam.CombinePerKey(sum))

    # Format the counts into a PCollection of strings.
    def format_result(word, count):
      return '%s: %d' % (word, count)

    output = counts | 'Format' >> beam.MapTuple(format_result)

    # Write the output using a "Write" transform that has side effects.
    # pylint: disable=expression-not-assigned
    output | 'Write' >> WriteToText(known_args.output)
```

## 5. まとめ

　既存の特徴量ストアと並列パイプラインモデルの組み合わせで、MLシステムのコア機能を容易に実装できる環境が整いつつあり、クラウド利用を前提として比較的小規模な（機械学習モデルの）開発者とエンジニアリングチームでも十分なサービスが提供できる。

### <参考情報>

[1] Kevin Stumpf (2020). Why We Need DevOps for ML Data, https://www.tecton.ai/blog/devops-ml-data/

[2] Google Cloud. MLOps: 機械学習における継続的デリバリーと自動化のパイプライン, https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning?hl=ja

[3] Willem Pienaar & Mike Del Balso (2021) . What is a Feature Store? , https://feast.dev/blog/what-is-a-feature-store/

[4] Google Cloud. Apache Beam のプログラミング モデル, https://cloud.google.com/dataflow/docs/concepts/beam-programming-model?hl=ja