

Kaggler のための BigQuery 活用手法

佐藤 貴海

カリフォルニア大学アーバイン校

計算機科学専攻博士課程在籍



About Me: 佐藤 貴海 (SATO, Takami)

- 東京工業大学工学部経営工学科卒業 (2011 年)・修士修了 (2013 年)
 - 専門は数理最適化
- データサイエンティスト・機械学習エンジニアとして日本で 5 年程勤務
- 現在はカリフォルニア大学アーバイン校計算機科学専攻博士課程に在籍
 - 専門分野は機械学習のセキュリティ、特に自動運転システムを対象に研究



[Edit profile](#)

Takami Sato
@tkm2261

ML Security Researcher / Kaggle Master / CS Ph.D.
student at UCIrvine / MEng in Operations Research at
Titech

tkm2261.github.io Born November 13, 1988

Joined May 2012

684 Following 7,001 Followers

Dirty Road Can Attack: Security of Deep Learning based Automated Lane Centering under Physical-World Attack

Takami Sato*, Junjie Shen*, Ningfei Wang,
Yunhan Jack Jia, Xue Lin, and Qi Alfred Chen

AS²Guard Autonomous & Smart Systems
Guard Research Group

UCI

ByteDance

**Northeastern
University**

Published in Usenix Security '21

私と Kaggle

- KDD Cup の盛り上がりを見て興味をもつ (2014 年)
- 2016 年の Bosch Production Line Performance より本格参戦
- 以降、年に数コンペ参加、金メダル 1 個ぐらいのペースで参戦中
- Kagglers-ja slack (現在 11,850 人)参加者募集中！

Competitions Master 		
Current Rank 145 of 168,891	Highest Rank 68	
 4	 6	 4
Shopee - Price ...  5 months ago Top 1%	2nd of 2426	
Tweet Sentimen...  a year ago Top 1%	5th of 2225	
TalkingData AdT...  3 years ago Top 1%	11th of 3943	



Shopee - Price Match Guarantee

Determine if two products are the same by their images
Featured · Code Competition · 5 months ago

2/2426 



Tweet Sentiment Extraction

Extract support phrases for sentiment labels
Featured · Code Competition · a year ago

5/2225 



TalkingData AdTracking Fraud Detection Challenge

Can you detect fraudulent click traffic for mobile app ads?
Featured · 3 years ago

11/3943 



Instacart Market Basket Analysis

Which products will an Instacart consumer purchase again?
Featured · 4 years ago

15/2621 



Kaggleとは？

- **機械学習モデルのスコアを競うコンペを開催しているサイト**
 - 2017 年に Google が買収
- **数年前まで:**
 - 予測結果の CSV を提出する形式が主流
 - テストデータも配布されリーク発生もしばしば
 - ローカルでやりたい放題なので 100 モデルアンサンブルとかもあった
- **近年:**
 - Kaggle 上で GPU 付き Notebook が使える様に
 - Notebook を提出するコンペが主に
 - 計算時間制約のため無茶はできない
 - 時系列 API などにより現実に則した評価が可能に

Kaggle は業務に役に立つか？

直接は、

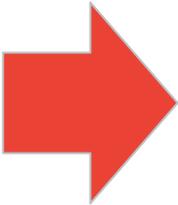
ほぼ役に立たない

入賞したからと言って高年収 DS 職が君を待っているとかは無い

Kaggle は業務に役に立つか？

- 大体の業務では。。。
 - 普通は機械学習が全く無いところに導入する。
 - 何もないから別に線形回帰でも成果が結構でる。
 - それよりデータ整備や実現場(工場)での運用整備が大変

そして9割以上の Kagglers がよく目にする



**「Kaggle は実業務に使えない」
「問題解くよりも Kaggle コンペに
落とし込むことが数百倍大事」**

につながる。

なぜ Kaggle をやるのか

- **ペアプロと似たような効果**

- 実業務で同じ問題を複数人が別々に解くことは無い
- 自分が挑戦した同じ問題の他人のコードは読みやすい & 超勉強になる。

- **実務上は Kaggle 上位モデルを抑えておけば大体 OK**

- 最新論文を追うのは企業人にはつらい

- **設計するにも問題がある程度解けるべき**

- Kaggle 入賞は良い客観的証明になる。
- 知らない領域のデータの性質に詳しくなれる。

なので、

Kaggle は

Not for キャリアアップ

Good for スキルアップ & 知的探求

そうだ、

Kaggleで、
BigQueryを学ぼう！

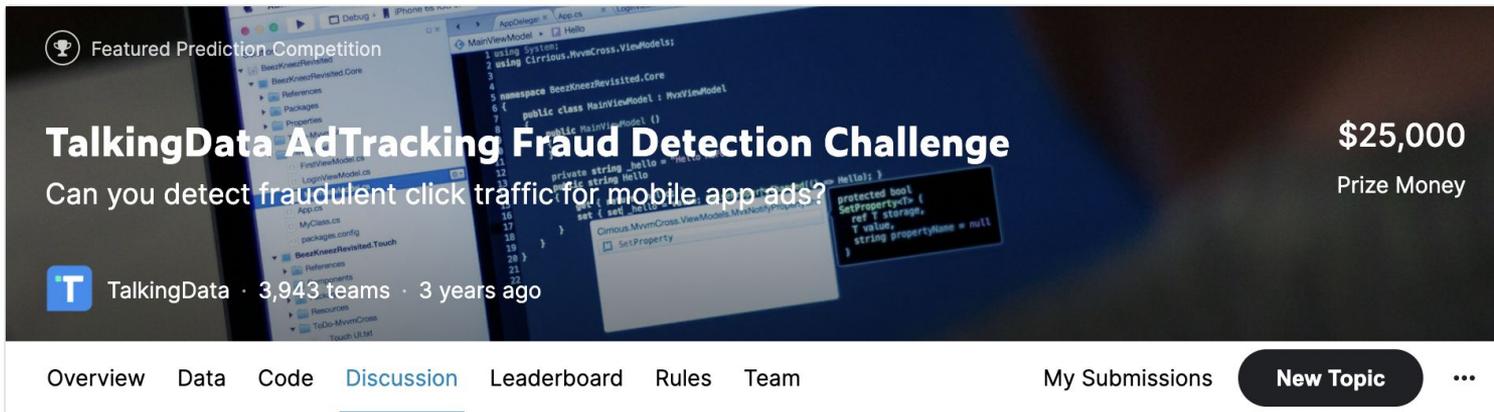


BigQuery

BigQuery の有用性

- **魔法かと思うほど速い**
 - 数億件の処理が一瞬で帰ってくる。
- **企業のデータはほぼすべてデータベースに入っている**
 - DS 職に SQL は必須知識
 - というか最近各社大体 BigQuery にデータが入ってる。
- **無料枠 + 従量課金制**
 - インスタンス立てるタイプだと月 3 万円以上は必要
 - Kaggle で使うレベルは従量課金 & 普通初期クーポン(\$300)に収まる。
- **DS としては分散処理のオプションは持っておくべき**
 - モデルが処理出来る形にするのは結構計算が重い。
 - SQL なら誰かに引き継ぎやすい。(Python・R よりは)

Kaggle での BigQuery の活用事例 (TalkingData)



The image shows a screenshot of the Kaggle competition page for 'TalkingData AdTracking Fraud Detection Challenge'. The page features a dark background with a code editor showing C# code for a mobile application. The main title is 'TalkingData AdTracking Fraud Detection Challenge' with a subtitle 'Can you detect fraudulent click traffic for mobile app ads?'. The prize money is listed as '\$25,000'. The competition is organized by TalkingData, with 3,943 teams and it started 3 years ago. The navigation bar includes 'Overview', 'Data', 'Code', 'Discussion', 'Leaderboard', 'Rules', 'Team', 'My Submissions', and a 'New Topic' button.

Featured Prediction Competition

TalkingData AdTracking Fraud Detection Challenge

Can you detect fraudulent click traffic for mobile app ads?

\$25,000
Prize Money

TalkingData · 3,943 teams · 3 years ago

Overview Data Code Discussion Leaderboard Rules Team My Submissions **New Topic** ...

- web のログから広告の不正なクリックを見つけるコンペ
- なんと訓練データのレコード数 184,903,890 件 (1.8 億件)

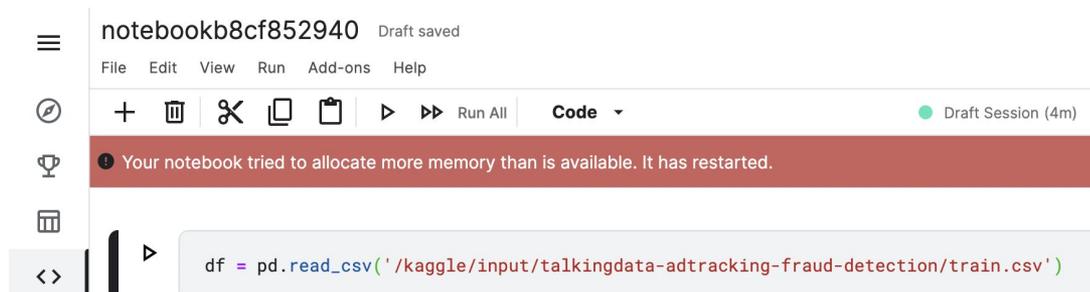
Data fields

Each row of the training data contains a click record, with the following features.

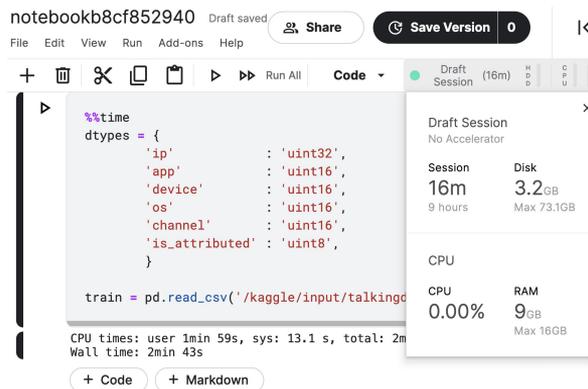
- ip : ip address of click.
- app : app id for marketing.
- device : device type id of user mobile phone (e.g., iphone 6 plus, iphone 7, huawei mate 7, etc.)
- os : os version id of user mobile phone
- channel : channel id of mobile ad publisher
- click_time : timestamp of click (UTC)
- attributed_time : if user download the app for after clicking an ad, this is the time of the app download
- is_attributed : the target that is to be predicted, indicating the app was downloaded

Kaggle での BigQuery の活用事例 (TalkingData)

メモリ 16 GB の Notebook では簡単に読むことすらできない



型指定すると読めるが、さらなる分析には厳しい

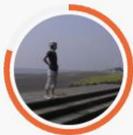


そうだ、

BigQuery を使おう！



私の 11 位の解法 with BigQuery



tkm2261

Topic Author

11th place

11th place features

Posted in [talkingdata-adtracking-fraud-detection](#) 3 years ago

Add Tags

<https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/discussion/56250>

97

- BigQuery 上での処理は約 20 分！
 - Python でやろうと思うと半日以上かかったはず。
 - 実際にこのスレッドで 1 人が 1 日で再現できた。
 - 褒められた



Izmaylov Konstantin • (213th in this Competition) •

great solution! most suitable for production

さぞかし BigQuery は Kaggle で大人気なのだろう

Kaggle で BigQuery と検索すると。。。。

← BigQuery

🔍 View all results for "BigQuery"

🔍 bigquery

Comments View all 2,119



Reply to BigQuery is our friend! 1y • Riid Answer Correctness Prediction

←私



Reply to Importing data to BigQuery 3y • Home Credit Default Risk

←私

Topics View all 537



BigQuery is our friend! Riid Answer Correctness Prediction

←私



Importing data to BigQuery Home Credit Default Risk

←私

Notebooks View all 7,147



My 15th solution features (mainly using BigQuery) 4y • 3m to run • Python • ^ 28

←私



Fork of BigQuery SQL reproducing Paulo Pinto's LB: Private • 4y • 1h to run • Python • ^ 0

←私

正直、

私以外 BigQuery を
Kaggle で使ってるの
を見たことはない。

Kaggle で BigQuery いる？

- **実際必須ではない**

- 近年は Notebook 提出が増えて、
実際学習用のコードも Python で書いておいたほうが何かと便利

- **お金の無い学生でもコンペで勝てる可能性**

- BigQuery は高額請求がバズって危険視されているが、
実際 3 ヶ月コンペ参加しても数千円ぐらいなもの
- Google Cloud でインスタンス借りたら数万は簡単に行く

- **SQL でデータ処理する経験を積める**

- 私は業務経験から受領したデータは DB に入れるよう叩き込まれている
- 学生(未経験)のうちに SQL でのデータ処理の経験を積める

- **さらに Window 関数が便利で速い(後述)**

- 正直これが私が BigQuery を Kaggle で使う理由の 9 割

実際便利なので、

**これから実コンペでの
BigQuery の活用事
例を紹介します。**

その前に、分析 SQL 超入門

“分析とは *Group By* である”

Takami Sato

例えばマーケティング系のデータだと
ユーザと商品をどう Group By(集約)して見るかとなる

- **ユーザー特徴**

- 年代で Group By した RFM (Recency Frequency Monetary)
- 性別で Group By した RFM
- 店舗で Group By した RFM
- ...

- **商品特徴**

- カテゴリで Group By した平均価格
- 価格帯で Group By した購買割合
- ...

- **ユーザーx商品特徴**

- 各年代の商品カテゴリの購買割合 etc.

更にどう集約するかも(平均 / 最大 / 最小 / 標準偏差等)

そのため基本となる SQL は

```
1  SELECT          ※結果をdmt_userとして保存
2  |   user_id,
3  |   COUNT(1) as cnt_user,
4  |   AVG(amount) as avg_user_amount,
5  |   MIN(amount) as min_user_amount,
6  |   MAX(amount) as max_user_amount,
7  |   STDDEV(amount) as std_user_amount,
8  FROM
9  |   train_transaction
10 GROUP BY
11 |   user_id
```

Group by で集約して。

```
1  SELECT
2  |   t.*,
3  |   u.avg_user_amount,
4  |   u.min_user_amount,
5  |   u.max_user_amount,
6  |   u.std_user_amount
7  FROM
8  |   test_transaction as t
9  LEFT OUTER JOIN
10 |   dmt_user as u
11 GROUP BY
12 |   t.user_id = u.user_id
```

LEFT OUTER JOIN でくっつける

でも時系列的なのも考慮したいよね

- ユーザーの直近 10 回の平均勾配金額
- 商品の直近 3 ヶ月の販売個数
- そもそも全体で Group by したら未来の情報はいっちゃう

でも時系列的なのも考慮したいよね

- ユーザーの直近 10 回の平均勾配金額
- 商品の直近 3 ヶ月の販売個数
- そもそも全体で Group by したら未来の情報はいっちゃん

**そう、簡単にできる
Window 関数なら。**

BigQuery では分析関数 (Analytic function) となっている

分析関数の構文

```
analytic_function_name ( [ argument_list ] ) OVER over_clause
```

over_clause:

```
{ named_window | ( [ window_specification ] ) }
```

window_specification:

```
[ named_window ]  
[ PARTITION BY partition_expression [, ...] ]  
[ ORDER BY expression [ { ASC | DESC } ] [, ...] ]  
[ window_frame_clause ]
```

window_frame_clause:

```
{ rows_range } { frame_start | frame_between }
```

rows_range:

```
{ ROWS | RANGE }
```

そんなに複雑ではなく「直近 10 回のユーザの平均購買額」の場合

```
SELECT  
| AVG(amount) OVER(partition by user_id order by timestamp ROWS BETWEEN 10 PRECEDING AND 1 PRECEDING)
```

平均購買額を ユーザごとに timestamp で過去 10 個について求める。

もし前回から過去全てを対象にしたい場合は

```
AVG(amount) OVER(partition by user_id order by timestamp ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING)
```

もし今回も対象にしたい場合は

```
AVG(amount) OVER(partition by user_id order by timestamp ROWS BETWEEN 10 PRECEDING AND CURRENT ROW)
```

さらに、もし過去 30 日の平均を計算したい場合

```
AVG(amount) OVER(partition by user_id order by timestamp RANGE BETWEEN 3600 * 24 * 30 PRECEDING AND 1 PRECEDING)
```

これを Pandas でやろうとするとかなり大変

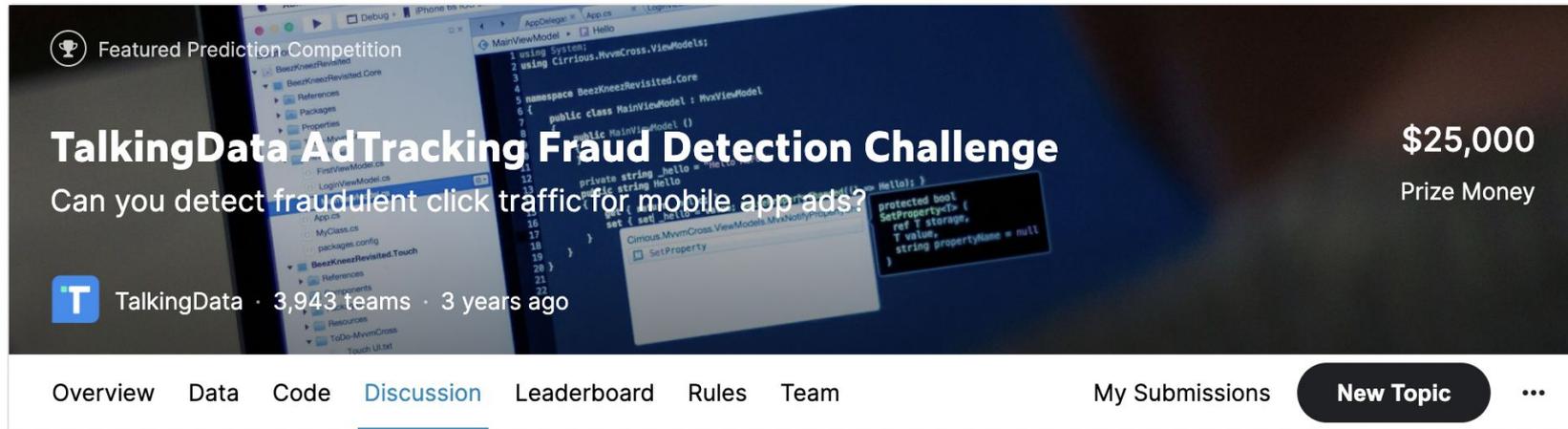
```
def agg_func(df):  
    df['avg_user_amount_visit_10'] = df.sort_values('timestamp')['amount'].rolling(window=10).mean()  
    return df  
df.groupby('user_id').apply(agg_func)
```

- User ごとの DF の中で処理を色々やるので激遅
- 速くする Tips は色々あるが可読性を損ないがち
- ↑みたいなのを書きたくないので BigQuery をよく使う

改めて、

これから実コンペでの
BigQuery の活用事
例を紹介します。

Kaggle での BigQuery の活用事例 (TalkingData)



The image shows a screenshot of a Kaggle competition page. The background is a dark-themed IDE with code snippets. The main text on the page reads: 'Featured Prediction Competition', 'TalkingData AdTracking Fraud Detection Challenge', 'Can you detect fraudulent click traffic for mobile app ads?', '\$25,000 Prize Money', 'TalkingData · 3,943 teams · 3 years ago'. At the bottom, there are navigation tabs: 'Overview', 'Data', 'Code', 'Discussion', 'Leaderboard', 'Rules', 'Team', 'My Submissions', and a 'New Topic' button.

Featured Prediction Competition

TalkingData AdTracking Fraud Detection Challenge

Can you detect fraudulent click traffic for mobile app ads?

\$25,000
Prize Money

TalkingData · 3,943 teams · 3 years ago

Overview Data Code Discussion Leaderboard Rules Team My Submissions [New Topic](#) ...

- このコンペでは機械的に特徴作成して試すのが鍵だった
- 遅めに参戦したのに勝てたのは BigQuery のおかげ
- 結局ひたすら Window 関数を変数を変えて試しまくった

Kaggle での BigQuery の活用事例 (Riild)

Featured Code Competition

Riild Answer Correctness Prediction

Track knowledge states of 1M+ students in the wild

\$100,000
Prize Money

Riild AEd Challenge · 3,395 teams · 9 months ago

Overview Data Code Discussion Leaderboard Rules Team My Submissions **New Topic** ...



tkm2261

BigQuery is our friend!

Posted in [riiid-test-answer-prediction](#) a year ago 58

Add Tags

- こちらもレコードが1億件あるので、当初試したGBDT用の特徴はBigQueryで作成
- その後Transformerが強いコンペになったのでその特徴もBigQueryで準備(次ページ)

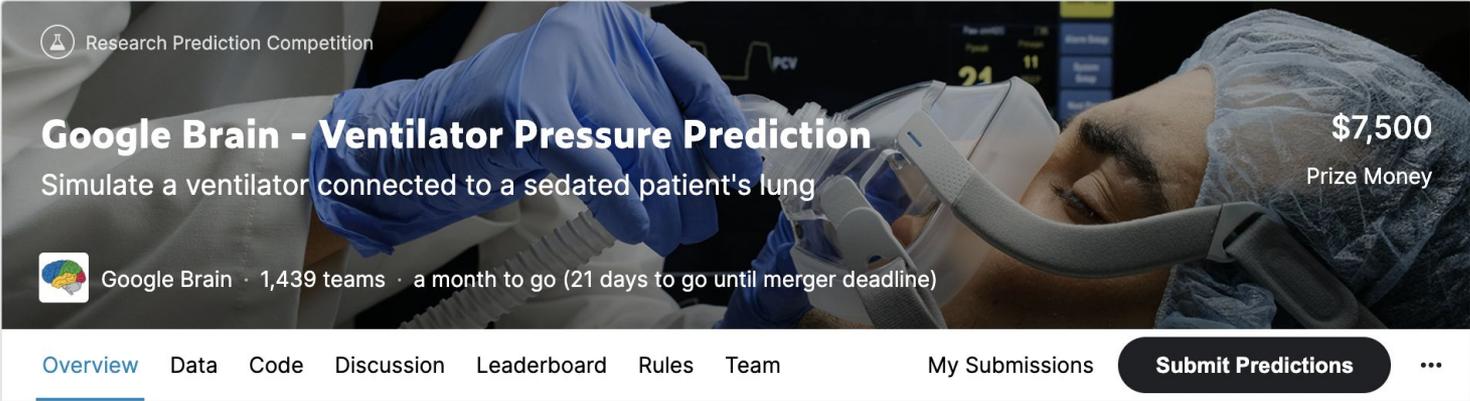
力技だが、リストの文字列を作ることが出来る

```
SELECT
  user_id,
  '[' || STRING_AGG(CAST(timestamp as STRING), ',' ORDER BY t.timestamp) || ']' timestamp,
  '[' || STRING_AGG(CAST(content_id as STRING), ',' ORDER BY t.timestamp) || ']' content_id,
  '[' || STRING_AGG(answered_correctly, ',' ORDER BY t.timestamp) || ']' answered_correctly,
  '[' || STRING_AGG(user_answer, ',' ORDER BY t.timestamp) || ']' user_answer,
FROM
  `train` as t
GROUP BY
  user_id
```

- Transformer に入れるのにユーザ毎のレコードにしたかった
- STRING_AGG で並べたのをコンマ区切りで結合
- 前後にカッコをつければ完成
- Python で読むときは eval() は遅いので json.loads() を使うと良い
- こんなのも 1 億件あると Python だと超遅い

Kaggle での BigQuery の活用事例 (Ventilator Pressure Prediction)

- 現行コンペだけど公開 Notebook にちょうど良い例があったので紹介



Research Prediction Competition

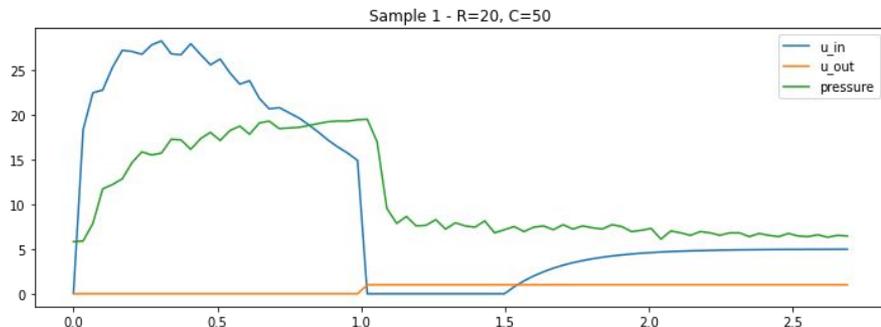
Google Brain - Ventilator Pressure Prediction

Simulate a ventilator connected to a sedated patient's lung

\$7,500
Prize Money

Google Brain · 1,439 teams · a month to go (21 days to go until merger deadline)

[Overview](#) [Data](#) [Code](#) [Discussion](#) [Leaderboard](#) [Rules](#) [Team](#) [My Submissions](#) [Submit Predictions](#) ...



- 3秒間の人工呼吸器の圧力を諸々の変数から予測



Improvement base on Tensor Bidirect LSTM (0.173)

Python · Google Brain - Ventilator Pressure Prediction

<https://www.kaggle.com/kensit/improvement-base-on-tensor-bidirect-lstm-0-173>

```
%%time
def add_features(df):
    df['area'] = df['time_step'] * df['u_in']
    df['area'] = df.groupby('breath_id')['area'].cumsum()
    df['cross'] = df['u_in'] * df['u_out']
    df['cross2'] = df['time_step'] * df['u_out']

    df['u_in_cumsum'] = (df['u_in']).groupby(df['breath_id']).cumsum()
    df['one'] = 1
    df['count'] = (df['one']).groupby(df['breath_id']).cumsum()
    df['u_in_cummean'] = df['u_in_cumsum'] / df['count']

    df['breath_id_lag'] = df['breath_id'].shift(1).fillna(0)
    df['breath_id_lag2'] = df['breath_id'].shift(2).fillna(0)
    df['breath_id_lagsame'] = np.select([df['breath_id_lag']==df['breath_id']], [1], 0)
    df['breath_id_lag2same'] = np.select([df['breath_id_lag2']==df['breath_id']], [1], 0)
    df['u_in_lag1'] = df['u_in'].shift(1).fillna(0)
    df['u_in_lag1'] = df['u_in_lag1'] * df['breath_id_lagsame']
    df['u_in_lag2'] = df['u_in'].shift(2).fillna(0)
    df['u_in_lag2'] = df['u_in_lag2'] * df['breath_id_lag2same']
    df['u_out_lag2'] = df['u_out'].shift(2).fillna(0)
    df['u_out_lag2'] = df['u_out_lag2'] * df['breath_id_lag2same']

    df['R'] = df['R'].astype(str)
    df['C'] = df['C'].astype(str)
    df['RC'] = df['R']+df['C']
    #df = pd.get_dummies(df)
    return df

train = add_features(train_ori)
```

CPU times: user 16.8 s, sys: 1.46 s, total: 18.2 s
Wall time: 17.9 s

- ここでは既存の特徴を組み合わせて新しい特徴を作っている。
- 赤枠の処理では1個前のと2個前の‘u_in’と‘u_out’を求めている。
- ただ breath_id の境目の処理で複雑になっており可読性が低い
- ただ Pandas 君はこうやってベクトル演算風には書かないと著しく遅い



Improvement base on Tensor Bidirect LSTM (0.173)

Python · Google Brain - Ventilator Pressure Prediction <https://www.kaggle.com/kensit/improvement-base-on-tensor-bidirect-lstm-0-173>

- もしわかりやすくuser_id 毎の group_by で実装すると物凄く遅い
 - 17.9 秒が 11 分に悪化、36 倍遅くなる
 - Pandas はこういった例が度々あるので注意が必要

```
%%time
def map_breath(_df):
    df = _df.copy()
    df['area'] = df['time_step'] * df['u_in']
    df['area'] = df['area'].cumsum()
    df['cross'] = df['u_in'] * df['u_out']
    df['cross2'] = df['time_step'] * df['u_out']

    df['u_in_cumsum'] = df['u_in'].cumsum()
    df['count'] = np.arange(df.shape[0]) + 1
    df['u_in_cummean'] = df['u_in_cumsum'] / df['count']

    for i in range(1, 3):
        df[f'u_in_lag{i}'] = df['u_in'].shift(i).fillna(0)
        df[f'u_out_lag{i}'] = df['u_out'].shift(i).fillna(0)

    df['R'] = df['R'].astype(str)
    df['C'] = df['C'].astype(str)
    df['RC'] = df['R'] + df['C']

    return df

train = train_ori.groupby('breath_id').apply(map_breath).reset_index(drop=True)
```

CPU times: user 10min 52s, sys: 8.74 s, total: 11min 1s
Wall time: 11min 1s



FAST PANDAS LEFT JOIN (357x faster than pd.merge)

ちょうど良い例なので、この
データで BigQuery のデモ
をします。

どうやってデータを出し入れするか知らないと使えないですしね。

Kaggle Notebook からでも連携は出来るが



The screenshot shows the header of a Kaggle notebook. On the left, there is a circular profile picture of Jessica Li, followed by the text "JESSICA LI · 1Y AGO · 9,925 VIEWS". On the right, there is a navigation bar with a back arrow, the number "38", a "Copy & Edit" button, and the number "96". Below the navigation bar is the title "Tutorial: How to use BigQuery in Kaggle Kernels" in a large, bold, black font. Underneath the title, it says "Python · No data sources" and provides a blue hyperlink: <https://www.kaggle.com/jessicali9530/tutorial-how-to-use-bigquery-in-kaggle-kernels>.

**おそらくKaggler はモデルの学習は手元だと思うので
直接 Google Cloud から触る方法を解説します。**

先日発表された Vertex AI Workbench とかでナウでヤングにやる方法もありそう。

まずインポートしたいデータを適当な GCS に置く

- UI で Drag & Drop するか gsutil コマンドなどで置きましょう
- Kaggle Notebook から直接 gsutil などでも上げることができます
- Gzip 圧縮でも BigQuery は読めるのでファイルが大きいときは活用

The screenshot shows the Google Cloud Platform console interface for a bucket named 'takami-kaggle-12'. The breadcrumb path is 'Buckets > takami-kaggle-12 > vent'. The bucket's location is 'us-central1 (Iowa)', storage class is 'Standard', and it is subject to object ACLs. A table below shows one object, 'train.csv', with a size of 419.7 MB, type 'application/octet-stream', and created on Oct 6, 202...

Google Cloud Platform My First Project

← Bucket details REFRESH LEARN

takami-kaggle-12

Location	Storage class	Public access	Protection
us-central1 (Iowa)	Standard	⚠ Subject to object ACLs	None

OBJECTS CONFIGURATION PERMISSIONS PROTECTION LIFECYCLE

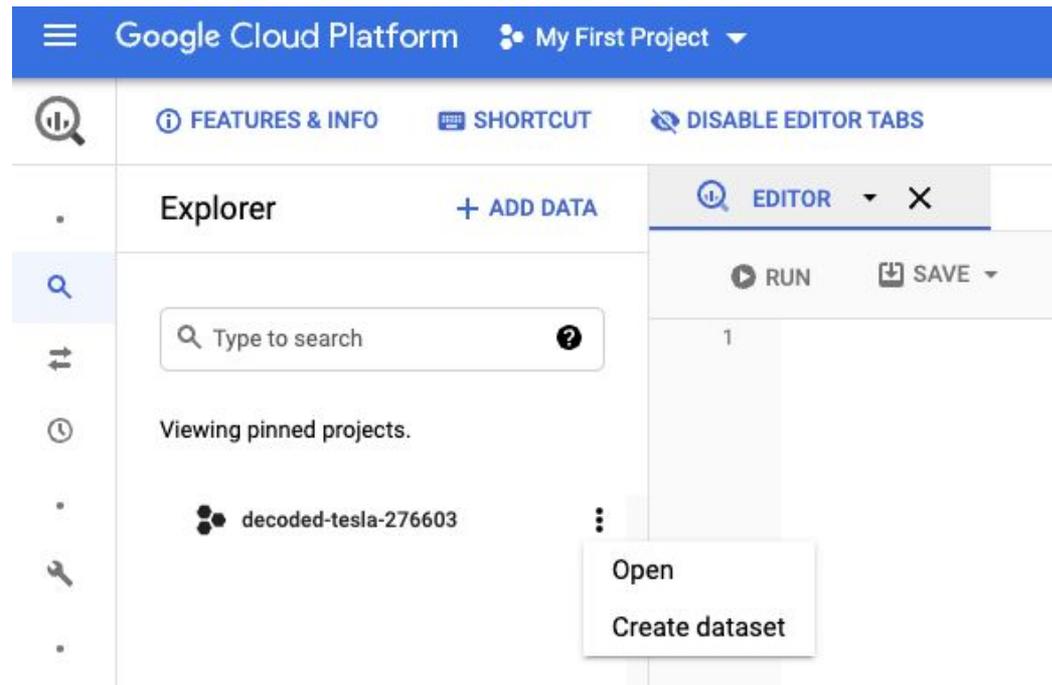
Buckets > takami-kaggle-12 > vent

UPLOAD FILES UPLOAD FOLDER CREATE FOLDER MANAGE HOLDS DOWNLOAD DELETE

Filter by name prefix only Filter Filter objects and folders Show deleted data

<input type="checkbox"/>	Name	Size	Type	Created	Storage class	Last modified
<input type="checkbox"/>	train.csv	419.7 MB	application/octet-stream	Oct 6, 202...	Standard	Oct 6, 202...

BigQuery の画面でデータセットを作成



Create dataset

Dataset ID *

vent

Letters, numbers, and underscores allowed

Data location

Default

Default table expiration

Enable table expiration ?

Default maximum table age

Encryption

Google-managed encryption key

No configuration required

Customer-managed encryption key (CMEK)

Manage via Google Cloud Key Management Service

CREATE DATASET

CANCEL

オプションはデフォルトで OK

作ったデータセットにテーブルを作成

The screenshot shows the Google Cloud Platform interface. At the top, there is a blue header with the Google Cloud Platform logo, the project name 'My First Project', and a search bar. Below the header, there are navigation options: 'FEATURES & INFO', 'SHORTCUT', and 'DISABLE EDITOR TABS'. The main area is divided into two panels. The left panel is the 'Explorer' view, showing a search bar and a list of projects. The right panel is the 'VENT' view, showing the details of a dataset named 'decoded-tesla-276603:vent'. A red box highlights the '+ CREATE TABLE' button in the top right corner of the dataset view. Below the dataset name, there are sections for 'Description' (None) and 'Dataset info'. A context menu is open over the dataset name, showing 'Open' and 'Delete' options.

Google Cloud Platform My First Project Search products and resources

FEATURES & INFO SHORTCUT DISABLE EDITOR TABS

Explorer + ADD DATA

VENT

decoded-tesla-276603:vent + CREATE TABLE

Description None

Dataset info

Dataset ID	decoded-tesla-276603:vent
Created	Oct 6, 2021, 9:05:29 PM
Default table expiration	Never

Open Delete

GCS からテーブルをインポート

Create table

Source

Create table from:

Google Cloud Storage

Select file from GCS bucket: ?

✓ takami-kaggle-12/vent/train.csv

Browse

File format:

CSV

Source Data Partitioning

Destination

Search for a project

Enter a project name

Project name

My First Project

Dataset name

vent

Table type ?

Native table

Table name

train

Schema

Auto detect

Schema and input parameters

i Schema will be automatically generated.

- Schema Auto Detection で勝手に型推定して入れてくれます。
- Kaggle のきれいなデータで失敗することはほぼない
- 他のオプションは超大規模データようなので Kaggle では触らなくて OK
- 昔は業務で受領したデータのスキーマ書くだけで 1日使ってたので良い時代です。

下の Job history が緑になってれば完了

Google Cloud Platform My First Project Search products and resources

FEATURES & INFO SHORTCUT DISABLE EDITOR TABS

Explorer + ADD DATA VENT X

decoded-tesla-276603:vent

Description None

Dataset info

Dataset ID	decoded-tesla-276603:vent
Created	Oct 6, 2021, 9:05:29 PM
Default table expiration	Never
Last modified	Oct 6, 2021, 9:05:29 PM
Data location	US

JOB HISTORY QUERY HISTORY SAVED QUERIES

Job history REFRESH

Personal history Project history

Filter jobs

Today

9:16 PM Load gs://takami-kaggle-12/vent/train.csv to decoded-tesla-276603:vent.train

train

SCHEMA

DETAILS

PREVIEW

Table info

Table ID	decoded-tesla-276603:vent.train
Table size	368.41 MB
Long-term storage size	0 B
Number of rows	6,036,000
Created	Oct 6, 2021, 9:17:19 PM UTC-7
Last modified	Oct 6, 2021, 9:17:19 PM UTC-7
Table expiration	NEVER
Data location	US
Description	

先程の処理を BigQuery (SQL) で書くと (LAG で発生した null 以外)、

```
%%time
def add_features(df):
    df['area'] = df['time_step'] * df['u_in']
    df['area'] = df.groupby('breath_id')['area'].cumsum()
    df['cross'] = df['u_in'] * df['u_out']
    df['cross2'] = df['time_step'] * df['u_out']

    df['u_in_cumsum'] = (df['u_in']).groupby(df['breath_id']).cumsum()
    df['one'] = 1
    df['count'] = (df['one']).groupby(df['breath_id']).cumsum()
    df['u_in_cummean'] = df['u_in_cumsum'] / df['count']

    df['breath_id_lag'] = df['breath_id'].shift(1).fillna(0)
    df['breath_id_lag2'] = df['breath_id'].shift(2).fillna(0)
    df['breath_id_lagsame'] = np.select([df['breath_id_lag']==df['breath_id']], [1], 0)
    df['breath_id_lag2same'] = np.select([df['breath_id_lag2']==df['breath_id']], [1], 0)
    df['u_in_lag1'] = df['u_in'].shift(1).fillna(0)
    df['u_in_lag1'] = df['u_in_lag1'] * df['breath_id_lagsame']
    df['u_in_lag2'] = df['u_in'].shift(2).fillna(0)
    df['u_in_lag2'] = df['u_in_lag2'] * df['breath_id_lag2same']
    df['u_out_lag2'] = df['u_out'].shift(2).fillna(0)
    df['u_out_lag2'] = df['u_out_lag2'] * df['breath_id_lag2same']

    df['R'] = df['R'].astype(str)
    df['C'] = df['C'].astype(str)
    df['RC'] = df['R']+df['C']
    #df = pd.get_dummies(df)
    return df

train = add_features(train_ori)
```

CPU times: user 16.8 s, sys: 1.46 s, total: 18.2 s
Wall time: 17.9 s



```

RUN SAVE SCHEDULE MORE
This query will process

1 SELECT
2 *,
3 SUM(time_step * u_in) OVER(partition by breath_id order by time_step ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS area,
4 u_in * u_out AS cross1,
5 time_step * u_out AS cross2,
6 ROW_NUMBER() OVER(partition by breath_id order by time_step) AS count1,
7 SUM(u_in) OVER(partition by breath_id order by time_step ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS u_in_cumsum,
8 AVG(u_in) OVER(partition by breath_id order by time_step ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS u_in_cummean,
9 LAG(u_in, 1) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_in_lag1,
10 LAG(u_in, 2) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_in_lag2,
11 LAG(u_out, 1) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_out_lag1,
12 LAG(u_out, 2) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_out_lag2,
13 CAST(R as string) || CAST(C as string) AS RC
14 FROM
15 `decoded-tesla-276603.vent.train`
16 ORDER BY
17 breath_id, time_step
```

個人的にはとてもとても読みやすい

拡大版

 RUN  SAVE  SCHEDULE  MORE

 This query will process

```
1 SELECT
2     *,
3     SUM(time_step * u_in) OVER(partition by breath_id order by time_step ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS area,
4     u_in * u_out AS cross1,
5     time_step * u_out AS cross2,
6     ROW_NUMBER() OVER(partition by breath_id order by time_step) AS count1,
7     SUM(u_in) OVER(partition by breath_id order by time_step ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS u_in_cumsum,
8     AVG(u_in) OVER(partition by breath_id order by time_step ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS u_in_cummean,
9     LAG(u_in, 1) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_in_lag1,
10    LAG(u_in, 2) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_in_lag2,
11    LAG(u_out, 1) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_out_lag1,
12    LAG(u_out, 2) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_out_lag2,
13    CAST(R as string) || CAST(C as string) AS RC
14 FROM
15     `decoded-tesla-276603.vent.train`
16 ORDER BY
17     breath_id, time_step
```

この結果を保存したいので Query Settings を開く

```
1 SELECT
2 *,
3 SUM(time_step * u_in) OVER(partition by breath_id order by time_step ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS count1,
4 u_in * u_out AS cross1,
5 time_step * u_out AS cross2,
6 ROW_NUMBER() OVER(partition by breath_id order by time_step ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS count1,
7 SUM(u_in) OVER(partition by breath_id order by time_step ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS count1,
8 AVG(u_in) OVER(partition by breath_id order by time_step ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS count1,
9 LAG(u_in, 1) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_in_lag1,
10 LAG(u_in, 2) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_in_lag2,
11 LAG(u_out, 1) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_out_lag1,
12 LAG(u_out, 2) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_out_lag2,
13 CAST(R as string) || CAST(C as string) AS RC
14 FROM
15 `decoded-tesla-276603.vent.train`
16 ORDER BY
17 breath_id, time_step
```

- 宛先を指定して保存
- 他のオプションはデフォルトで OK

Query settings

Destination

- Save query results in a temporary table
- Set a destination table for query results

Project name

My First Project

Dataset name

vent

Table name

train2

Destination table write preference

- Write if empty
- Append to table
- Overwrite table

Results size

Allow large results (no size limit)

Resource management

Job priority

- Interactive
- Batch

Cache preference

Use cached results

Sessions management

Use session mode

Additional settings

SQL dialect

- Standard
- Legacy

Processing location

United States (US)

SAVE

CLOSE

そして実行！

The screenshot shows a BigQuery interface with a SQL query editor and a results table. The query is a complex SELECT statement with multiple window functions. The execution status is highlighted with a red box, indicating the query is complete.

```
1 SELECT
2   *,
3   SUM(time_step * u_in) OVER(partition by breath_id order by time_step ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS area,
4   u_in * u_out AS cross1,
5   time_step * u_out AS cross2,
6   ROW_NUMBER() OVER(partition by breath_id order by time_step) AS count1,
7   SUM(u_in) OVER(partition by breath_id order by time_step ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS u_in_cumsum,
8   AVG(u_in) OVER(partition by breath_id order by time_step ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS u_in_cummean,
9   LAG(u_in, 1) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_in_lag1,
10  LAG(u_in, 2) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_in_lag2,
11  LAG(u_out, 1) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_out_lag1,
12  LAG(u_out, 2) OVER (PARTITION BY breath_id ORDER BY time_step) AS u_out_lag2,
13  CAST(R as string) || CAST(C as string) AS RC
14 FROM
15   `decoded-tesla-276603.vent.train`
16 ORDER BY
17   breath_id, time_step
```

Processing location: US Destination table: decoded-tesla-276603:vent.train2 Write if empty No cached results

Query results [SAVE RESULTS](#) [EXPLORE DATA](#)

Query complete (17.4 sec elapsed, 368.4 MB processed)

Row	id	breath_id	R	C	time_step	u_in	u_out	pressure	area	cross1	cross2
1	1	1	20	50		0.0	0.08333400563464438	0	5.837491705069121	0.0	0.0
2	2	1	20	50	0.033652305603027344	18.383041472634716	0	5.9077938505203464	0.6186317295502293		0.0

- 実行時間は 17.4 秒と元の 17.9 秒に比べてほぼ同程度
 - 11 分と比べれば雲泥の差、可読性も高い。
- 小さいデータなので差はほぼないが、BigQuery はメモリに乗らないデータも高速に処理できるので色々試してみてください。

出力結果を手元に DL

The screenshot shows the Google Cloud Platform BigQuery interface. The top navigation bar includes 'Google Cloud Platform', 'My First Project', and a search bar. The main interface is divided into several sections:

- Explorer:** On the left, a tree view shows a project named 'decoded-tesla-276603' with sub-projects 'vent', 'train', and 'train2'. 'train2' is selected and highlighted.
- Table Selection:** At the top, two tables are listed: 'VENT' and 'TRAIN2'. 'TRAIN2' is selected.
- Actions:** Above the table name, there are buttons for 'QUERY', 'SHARE', 'COPY', 'DELETE', and 'EXPORT'. The 'EXPORT' button is highlighted with a red box.
- Table Schema:** Below the table name, the 'SCHEMA' tab is active, showing the 'Table schema' for 'train2'. It includes a filter input and a table with columns: 'Field name', 'Type', 'Mode', 'Policy Tags', and 'Description'. The visible rows are:

Field name	Type	Mode	Policy Tags	Description
id	INTEGER	NULLABLE		
breath_id	INTEGER	NULLABLE		
P	INTEGER	NULLABLE		
- Export Options:** To the right of the schema, there are links for 'Explore with Data Studio', 'Export to GCS' (highlighted with a red box), and 'Scan with DLP'.

- テーブルを選んで GCS へ出力
- 10 MB 以下だと直接 DL できるが GCS に書いておいたほうが無難
 - Kaggle だと大体 CUI の Linux マシンで gsutil コマンドで DL
- 一連の流れは bq コマンドと gsutil コマンドで自動化もできる

まとめ

- Kaggle はデータ分析の勉強 & キャッチアップに最適
- 業務でよく使われる BigQuery に Kaggle で慣れよう！
- BigQuery は魔法の様に速い & 時系列処理が Window 関数で書きやすい
- Kaggle で BigQuery 使う仲間が欲しい。



Thank you!

質問などはTwitterの [@tkm2261](#)まで