Taxi Fare Amount in New York

Google Cloud Competition - Big Data Project

Andrea Bacciu
&
Riccardo Taiello

Master of Science in
Computer Science
Sapienza, University of Rome

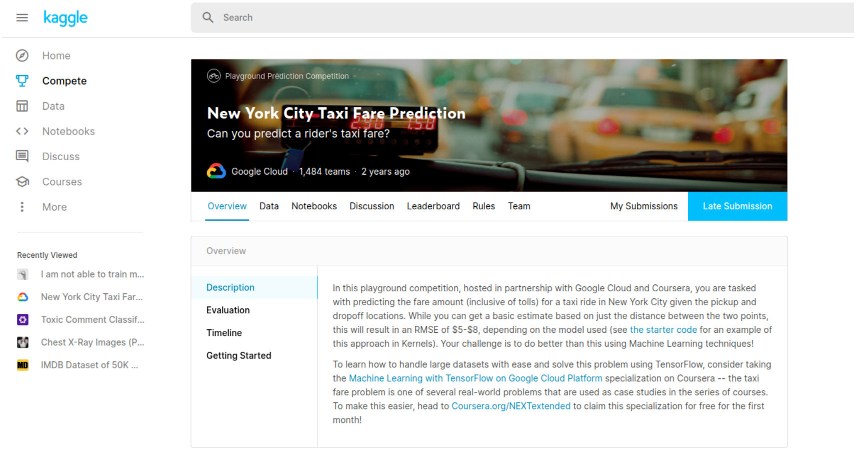
A. Y. 2019 - 2020

Table of contents

Outline

Google and Coursera's Kaggle Competition

- `https://www.kaggle.com/c/new-york-city-taxi-fare-prediction/overview`
- The goal of the task is to create a model capable of predicting the taxi fare.
- Geographical coordinates must be used as inputs.
- In order to do it we plug the knowledge of spatial coordinates



In this playground competition, hosted in partnership with Google Cloud and Coursera, you are tasked with predicting the fare amount (inclusive of tolls) for a taxi ride in New York City given the pickup and dropoff locations. While you can get a basic estimate based on just the distance between the two points, this will result in an RMSE of \$5-\$8, depending on the model used (see the starter code for an example of this approach in Kernels). Your challenge is to do better than this using Machine Learning techniques!

To learn how to handle large datasets with ease and solve this problem using TensorFlow, consider taking the Machine Learning with TensorFlow on Google Cloud Platform specialization on Coursera -- the taxi fare problem is one of several real-world problems that are used as case studies in the series of courses. To make this easier, head to Coursera.org/NEXTextended to claim this specialization for free for the first month!

Outline

Dataset

- The dataset is composed by 5.31Gb of CSV
- We use 4,000,000 sample over 55,000,000 sample
- The dataset has 6 columns
- We split the dataset into Train(80%), Dev(10%) and Test(10%)
- The dataset contains old and recent data and the roads can be changed.
- They can be subjected to different closings daily, for events or other types of extraordinary situations.

Information

1. **pickup_datetime** - timestamp - taxi ride started.
2. **pickup_longitude** - float - longitude coordinate of where the taxi ride started.
3. **pickup_latitude** - float - latitude coordinate taxi ride started.
4. **dropoff_longitude** - float - longitude coordinate of where the taxi ride ended.
5. **dropoff_latitude** - float - latitude coordinate of where the taxi ride ended.
6. **passenger_count** - integer - number of passengers in the taxi ride.

Metrics - Root Mean Square Error

- Root-Mean-Square-Error

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- RMSE is always non-negative
  - Value of 0 (almost never achieved in practice) would indicate a perfect fit to the data.
- A lower RMSE is better than a higher one.
- Comparisons across different types of data would be invalid because the measure is dependent on the scale of the numbers used.

Outline

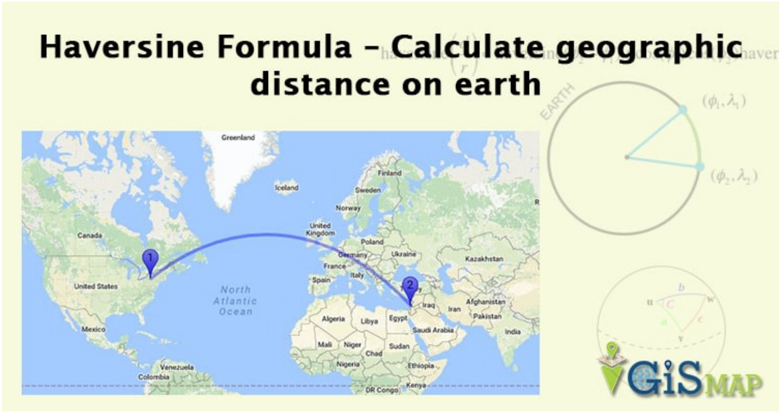
Distance Feature

- We work a lot on calcutation of meaningful distances ...
- We want calculate a real distance between two points.
- The first goal is to calculate a distance based on the road and the signs.
- The second goal is to calculate the elapsed time between two points.

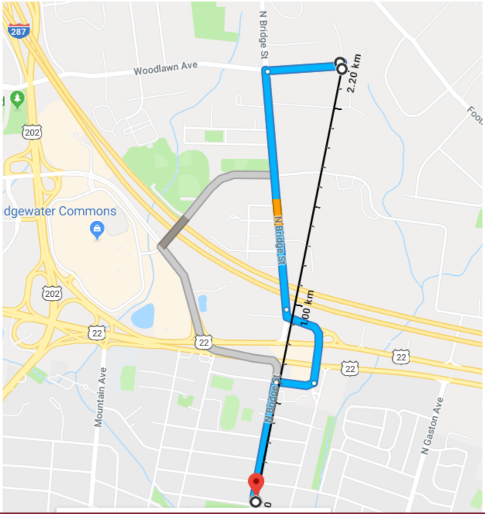
Flat earth

Haversine Distance

- It's the best mathematics distance for our task, since deal with geospatial coordinate, but get us a notion of **air-distance**.



Why real distance



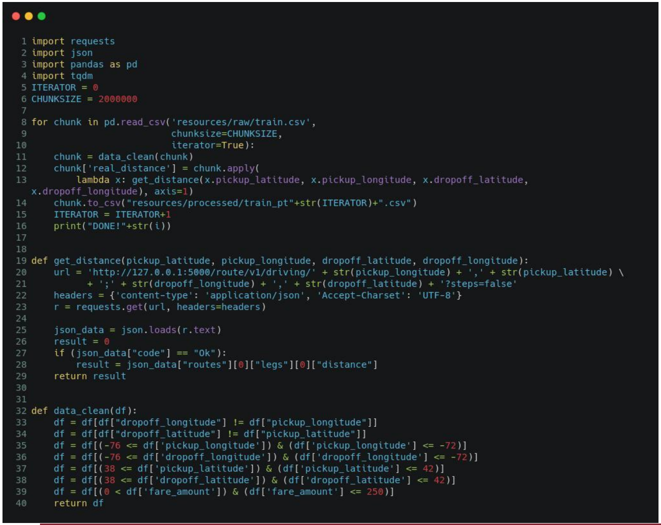
Google Maps & Waze API

- Unfortunately the Google Maps APIs are paid
- Waze API:
  - They are cool and free. Waze offer the taxi heuristics.
  - But they are limited and impossible to use on a dataset of this size.

OpenStreetMap Routing

- We use the offline OpenStreetMap Routing API
- To get high performance, we downloaded a NY map in local and we ran the server using Docker
- For further information see
  `https://github.com/Project-OSRM/osrm-backend`

OpenStreetMap Routing - Script



```python
import requests
import json
import pandas as pd
import tqdm
ITERATOR = 0
CHUNKSIZE = 2000000

for chunk in pd.read_csv('resources/raw/train.csv',
                         chunksize=CHUNKSIZE,
                         iterator=True):
    chunk = data_clean(chunk)
    chunk['real_distance'] = chunk.apply(
        lambda x: get_distance(x.pickup_latitude, x.pickup_longitude, x.dropoff_latitude,
x.dropoff_longitude), axis=1)
    chunk.to_csv("resources/processed/train_pt"+str(ITERATOR)+".csv")
    ITERATOR = ITERATOR+1
    print("DONE!"+str(i))


def get_distance(pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude):
    url = 'http://127.0.0.1:5000/route/v1/driving/' + str(pickup_longitude) + ',' + str(pickup_latitude) \
          + ';' + str(dropoff_longitude) + ',' + str(dropoff_latitude) + '?steps=false'
    headers = {'content-type': 'application/json', 'Accept-Charset': 'UTF-8'}
    r = requests.get(url, headers=headers)

    json_data = json.loads(r.text)
    result = 0
    if (json_data["code"] == "Ok"):
        result = json_data["routes"][0]["legs"][0]["distance"]
    return result


def data_clean(df):
    df = df[df["dropoff_longitude"] != df["pickup_longitude"]]
    df = df[df["dropoff_latitude"] != df["pickup_latitude"]]
    df = df[(-76 <= df['pickup_longitude']) & (df['pickup_longitude'] <= -72)]
    df = df[(-76 <= df['dropoff_longitude']) & (df['dropoff_longitude'] <= -72)]
    df = df[(38 <= df['pickup_latitude']) & (df['pickup_latitude'] <= 42)]
    df = df[(38 <= df['dropoff_latitude']) & (df['dropoff_latitude'] <= 42)]
    df = df[(0 < df['fare_amount']) & (df['fare_amount'] <= 250)]
    return df
```

Datatime Feature

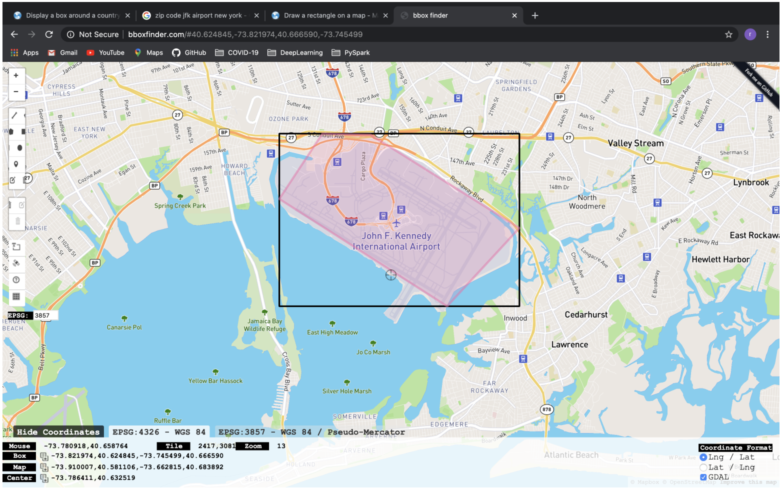
- Split the date into:
  - minute;
  - hour;
  - day's number during the week;
  - day's number during the month;
  - month;
  - year.
- Two features (*is_rush_hour*, *is_night*) which describes the chance to have an extra *fare_amount*

Airport Feature

- We drew planes over the map of New York by taking the coordinates of the airport areas
  - Departures Airport: Boolean
  - Arrivals Airport: Boolean
  - Departures or Arrivals: Boolean
  - {Name of airport} Arrivals: Boolean
  - {Name of airport} Departures: Boolean

Brooklyn
Crown Heights
Brownsville
East New York
Howard Beach
South Ozone Park
Springfield Gardens
Laurelton
Temporaneamente chiuso
20 min
8,5 miglia
East Flatbush
Flatbush
Canarsie
Belt Pkwy
Aeroporto John Fitzgerald Kennedy...

Airport Feature


bboxfinder.com/#40.624845,-73.821974,40.666590,-73.745499
John F. Kennedy International Airport
Valley Stream
Lynbrook
East Rockaway
Hewlett Harbor
Cedarhurst
Lawrence
Inwood
Jamaica Bay Wildlife Refuge
Hide Coordinates
EPSG:4326 - WGS 84
EPSG:3857 - WGS 84 / Pseudo-Mercator
Coordinate Format
Lng / Lat
Lat / Lng
GDAL

```
Mouse   -73.780918,40.658764        Tile  2417,3081  Zoom  13
Box     -73.821974,40.624845,-73.745499,40.666590
Map     -73.910007,40.581106,-73.662815,40.683892
Center  -73.786411,40.632519
```

Airport Intuition

Measurement	**JFK**	**EWR**	**LGA**
count	11672	8	96585
mean	**9.85**	**9.85**	**30.18**
std	**10.55**	**4.68**	**7.98**
min	3.70	4.90	3.70
25%	18.1	7.10	26.25
50%	25.50	8.50	30.50
75%	33.50	10.80	30.30
max	49.70	19.30	49.70

Table: Measurement with the respect to the target label

New Features After Extraction

1. **haversine_distance**: Float
2. **real_distance**: Float
3. **is_night**: Boolean $hour \leq 20 \vee hour \geq 6$
4. **is rush hour**: Boolean
   $day\_of\_week \leq 5 \wedge (hour \geq 16 \wedge hour \leq 20)$
5. **is_jfk**: Boolean
6. **is_lga**: Boolean
7. **is_ewr**: Boolean

Outlier deletion (Round 1)

- Departures coordinates equals to arrivals coordinates
- Coordinates outside the New York area
- Taxi fare prices negative
- Taxi fare over 250$

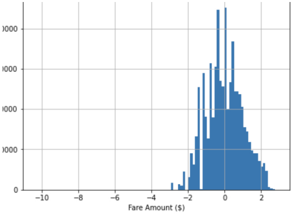
REMARK

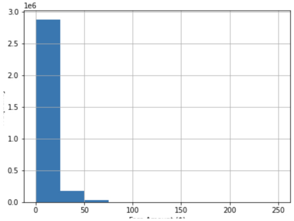
The second round concerning outlier deletion, it performed after the feature extraction step.

Outlier Deletion (Round 2)

The second round, was performed with statistical approach, such as:

- Real-distance, log-scale it, since it follows a power-low distribution. Then **outliers** detection keept the 97% perncentile of the Gaussian (result of the log-scale) distrubution.
- Fare-amount, the same trick of the real-distance, but here we removed outilier just for the train dataset, since is the target-variable. We keept the 98 % percentile of the scaled distibution.

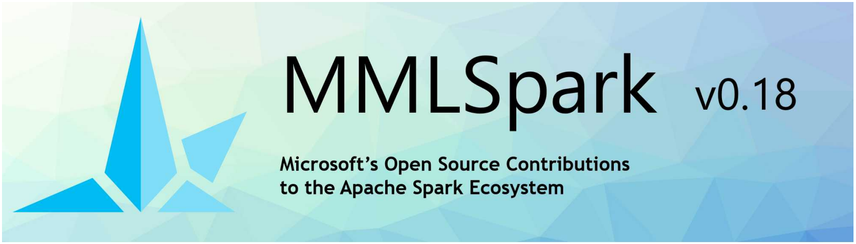




Outline

Used Models

- PySpark - Linear Regression model (baseline)
- PySpark - Random Forest Regressor
- PySpark - Microsoft - LGBM Light Gradient Boosting Machine Regressor


MMLSpark v0.18
Microsoft's Open Source Contributions to the Apache Spark Ecosystem

Training phase

- We applied this pipeline to train and test our models:
  1. In order to find the best set of hyperparameters we performed the grid search and cross validation
  2. We split the dataset into three portions (train, dev, test)
- To simulate a real case scenario (deployment) we merge the train and dev as a single dataset, we refit and then test on test.

Outline

Results

Model	**Dev**	**Test**
Linear Regr. (baseline)	3.42	5.23
Random Forest Regr.	3.27	5.07
LGBM Regr.	**2.83**	**4.4**

Table: Results of our model expressed in the RMSE Metrics (Lower is better)

Conclusions

- The best model was the LGBM Regressor from Microsoft implementation.
- However, we used a fragment of the dataset because we didn't have a real cluster available.
- We treated the project in a setting of a real scenario and we tested it in a deployment settings.
- We consider this problem very important, for example for all those mobility apps (such as UBER) who want to offer travel advice.

Outline

Recap

- We performe Preprocessing:
    - Real distance computation (Docker)
    - Outlier deletion
    - Statistical approach (percentile techniques)
- Feature enginner:
    - Distance Feature
    - Timestamp Feature
    - Airport Feature
- We used 3 different algorithms to address the problem
- In combination with Grid Search
- We simulate a sort of deployment setting using all avaiable data

Reference

- https://www.kaggle.com/c/new-york-city-taxi-fare-prediction/overview
- https://mmlspark.blob.core.windows.net/website/index.html
- https://github.com/Project-OSRM/osrm-backend
- http://bboxfinder.com