Taxi trip time calculation using Supervised learning

Sourav Jha

September 21, 2017

1 Definition

1.1 Project Overview

1.1.1 Domain Background

The taxi industry is evolving rapidly. New competitors and technologies are changing the way traditional taxi services do business. While this evolution has created new efficiencies, it has also created new problems.

One major shift is the widespread adoption of electronic dispatch systems that have replaced the VHF-radio dispatch systems of times past. These mobile data terminals are installed in each vehicle and typically provide information on GPS localization and taximeter state. Electronic dispatch systems make it easy to see where a taxi has been, but not necessarily where it is going. In most cases, taxi drivers operating with an electronic dispatch system do not indicate the final destination of their current ride.

Another recent change is the switch from broadcast-based (one to many) radio messages for service dispatching to unicast-based (one to one) messages. With unicast-messages, the dispatcher needs to correctly identify which taxi they should dispatch to a pick up location. Since taxis using electronic dispatch systems do not usually enter their drop off location, it is extremely difficult for dispatchers to know which taxi to contact.

To improve the efficiency of electronic taxi dispatching systems it is important to be able to predict how long a driver will have his taxi occupied. If a dispatcher knew approximately when a taxi driver would be ending their current ride, they would be better able to identify which driver to assign to each pickup request.This problem is part of two kaggle competitions(ECML/PKDD 15*[1]* and NYC taxi *[5]*).

1.1.2 Dataset and Input

The dataset contains a complete year (from 01/07/2013 to 30/06/2014) of the trajectories for the 442 taxis running in the city of Porto, in Portugal . These taxis operate through a taxi dispatch central, using mobile data terminals installed in the vehicles.Each ride is categorized into three categories: A) taxi central based, B) stand-based or C) non-taxi central based. For the first, there is an anonymized id, when such information is available from the telephone call. The last two categories refer to services that were demanded directly to the taxi drivers on a B) taxi stand or on a C) random street.

Each data sample corresponds to one completed trip. It contains a total of 9 (nine) features, described as follows:

- **TRIP_ID**: (String) It contains an unique identifier for each trip.
- **CALL_TYPE**: (char) It identifies the way used to demand this service. It may contain one of three possible values:
  - 'A' if this trip was dispatched from the central.
  - 'B' if this trip was demanded directly to a taxi driver on a specific stand.
  - 'C' otherwise (i.e. a trip demanded on a random street).

- **ORIGIN_CALL**: (integer) It contains an unique identifier for each phone number which was used to demand, at least, one service. It identifies the trip's customer if CALL_TYPE='A'. Otherwise, it assumes a NULL value.
- **ORIGIN_STAND**: (integer): It contains an unique identifier for the taxi stand. It identifies the starting point of the trip if CALL_TYPE='B'. Otherwise, it assumes a NULL value.
- **TAXI_ID**: (integer): It contains an unique identifier for the taxi driver that performed each trip.
- **TIMESTAMP**: (integer) Unix Timestamp (in seconds). It identifies the trip's start.
- **DAYTYPE**: (char) It identifies the daytype of the trip's start. It assumes one of three possible values:
  - 'B' if this trip started on a holiday or any other special day (i.e. extending holidays, floating holidays, etc.).
  - 'C' if the trip started on a day before a type-B day.
  - 'A' otherwise (i.e. a normal day, workday or weekend).
- **MISSING_DATA**: (Boolean) It is FALSE when the GPS data stream is complete and TRUE whenever one (or more) locations are missing.
- **POLYLINE**: (String): It contains a list of GPS coordinates (i.e. WGS84 format) mapped as a string. The beginning and the end of the string are identified with brackets (i.e. [and], respectively). Each pair of coordinates is also identified by the same brackets as [LONGITUDE, LATITUDE]. This list contains one pair of coordinates for each 15 seconds of trip. The last list item corresponds to the trip's destination while the first one represents its start.

1.2 Problem Statement

In this project, I have built a predictive framework using supervised learning that is able to infer the trip time of taxi rides in Porto, Portugal based on their (initial) partial trajectories.It is a regression model to predict the taxi trip.

I have used following supervised learning algorithms:

- Support Vector regression*[2]*
- XGBoost*[3]*
- Neural Networks*[4]*

I have majorly concentrated on XGboost, since it gave better result for default values.

1.3 Metrics

I have used the Root Mean Squared Logarithmic Error (RMSLE) to evaluate different models. The RMSLE is calculated as

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}((\log(p_i+1))-\log(a_i+1)))^2}$$

2 Analysis

2.1 Data Exploration and Visualization

Few general observations from the dataset :

- If MISSING_DATA is TRUE then the GPS data stream is incomplete, so we can remove the rows from dataset.There are only 10 rows with true value.
- Following features doesn't contain any relevant information: 'TRIP_ID', 'CALL_TYPE', 'ORIGIN_CALL', 'DAY_TYPE', 'ORIGIN_STAND', 'MISSING_DATA'.
  - TRIP_ID : It is just the identifier of trip.It doesn't relate with trip time.So,we can drop this column.
  - DAY_TYPE : For almost all the data points DAY_TYPE is 'A'.So,no relevant information.
  - 'TRIP_ID', 'CALL_TYPE', 'ORIGIN_CALL', 'ORIGIN_STAND' : All are related to place of origin of taxi.But,that information also availaible in POLYLINE feature as starting coordinates.Also,some of the values are NULL(e.g.ORIGIN_CALL will be null except for CALL_TYPE='A').

2.2 Algorithms and Techniques

2.2.1 XGBoost

XGBoost is an efficient and scalable implementation of gradient tree boosting. Its performance has been recognized by many winning solutions of Kaggle competitions. XGBoost is a model based on tree ensemble which is a set of classification and regression trees(CART).It's implementation is available in two forms

- xgb.XGBRegressor - this is similar to scikit implementation.
- xgb using DMatrix.

I have used xgb.XGBRegressor.For the XGBRegressor, i have given n_threads,min_child_weight and learning_rate params custom values,for others , i have used default values.To avoid overfitting, i have used k-fold cross validation where k = 10;

2.2.2 Support Vector Regression

The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. The main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.We have two variants osf SVR : one linear SVR and SVR with non linear kernel. I have used scikit' SVR with rbf(non-linear) kernel .

2.2.3 Neural Network

Neural networks systems are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve performance) to do tasks by considering examples, generally without task-specific programming. I have used scikit's MLPRegressor with 3 hidden layers and 200 iterations.

3 Methodology

3.1 Data Preprocessing

As mentioned in data exploration section following features doesn't contain any relevant information: 'TRIP_ID', 'CALL_TYPE', 'ORIGIN_CALL', 'DAY_TYPE', 'ORIGIN_STAND', 'MISSING_DATA'.Also, there are points which has missing data points.So, i cleaned the data set. Finally , i used the following feature :

- TAXI_ID,TIMESTAMP : are the same feature explained in Dataset and input section.
- startPLong,startPLat,endPLat,endPLong : They are starting point longitude,starting point latitude,ending point latitude,and ending point point longitude respectively.
- dist : dist is the distance between the starting and ending point coordinates.

```
#haversine code online code from kaggle
def haversine(lonlat1, lonlat2):
  lon_diff = np.abs(lonlat1[0]-lonlat2[0])*np.pi/360.0
  lat_diff = np.abs(lonlat1[1]-lonlat2[1])*np.pi/360.0
  a = np.sin(lat_diff)**2 + np.cos(lonlat1[1]) * np.cos(lonlat2[1]) * np.sin(lon_diff)**2
  d = 2*6371*np.arctan2(np.sqrt(a), np.sqrt(1-a))
  return(d)

def dista(row):
    return haversine([row['startPLong'],row['endPLong']],[row['endPLat'],row['startPLat']])

def preprocess(path,savePath) :

      df = pd.read_csv(path)
      # removing the data doesn,t
      df.drop(df[(df.MISSING_DATA == True)].index, inplace=True)
      # #create the regression value
      print("loading done ...........")
      df.drop(['TRIP_ID', 'CALL_TYPE', 'ORIGIN_CALL', 'DAY_TYPE', 'ORIGIN_STAND', 'MISSING_DATA'], axis=1,
              inplace=True)
      df['POLYLINE'] = [np.array(ast.literal_eval(x)) for x in df['POLYLINE']]
      df.drop(df[(df['POLYLINE'].map(len) == 0)].index, inplace=True)
      triptime = (df['POLYLINE'].map(len) - 1) * 15.0
      df['startPLong'] = df['POLYLINE'].apply(lambda x: x[0][0])
      df['startPLat'] = df['POLYLINE'].apply(lambda x: x[0][1])
      df['endPLat'] = df['POLYLINE'].apply(lambda x: x[len(x) - 1][1])
      df['endPLong'] = df['POLYLINE'].apply(lambda x: x[len(x) - 1][0])

      # #removing the features that are not relevant
      df['triptime'] = triptime
      df['dist'] = df.apply(dista, axis=1)
      df.drop(['POLYLINE'], axis=1, inplace=True)

      df.to_csv(savePath, index=False)
      # print( df['startPLong'])
      return df, triptime
```

Similarly, the training dataset also preprocessed. After preprocessing, the dataset will look like:

```
TAXI_ID,TIMESTAMP,startPLong,startPLat,endPLat,endPLong,triptime,dist
20000589,1372636858,-8.618643,41.141412,41.154489,-8.630838,330.0,1776.766029100033
20000596,1372637303,-8.639847,41.159826,41.170671,-8.66574,270.0,2480.3223994014115
20000320,1372636951,-8.612964,41.140359,41.14053,-8.61597,960.0,252.43917758355727
```

3.2 Algorithm Implementation

XGboost

```
def xgboost(X_train, y_train):
    model = xgb.XGBRegressor(nthread=4)
    params = {'min_child_weight': [2,3],
                'learning_rate':[0.4,0.5,0.6],
              'max_depth': [3,4]}
    grid = GridSearchCV(model, params,cv=10)
    grid.fit(X_train, y_train)
    joblib.dump(grid, 'xgboost.pkl')
    return grid
```

Support Vector Regression

```
def svr(X_train,y_train,K) :

    regressor = SVR(kernel='rbf')
    regressor.fit(X_train.values,y_train.values)

    # now you can save it to a file
    joblib.dump(regressor, 'svr.pkl')
    return regressor
```

Neural Network

```
def mlp(X_train,y_train) :

    mlp = MLPRegressor( hidden_layer_sizes=3, max_iter=200,
                        random_state=1)
    mlp.fit(X_train, y_train)
    return mlp
```

3.3 Benchmark

The problem is one of the kaggle competition ECML/PKDD 15*[1]*.The winner's solution had Root Mean Squared Logarithmic Error of 0.5092(Private LB[1]) and 0.5253 Public LB.For the project, I have compared my results with winner's result .

4 Experiments and Results

4.1 Model Evaluation and Validation

I chose two different set of features :

- feature set 1 : TAXI_ID,TIMESTAMP,startPLong,startPLat,endPLat,endPLong
- feature set 2 : TAXI_ID,TIMESTAMP,dist

First set , I have used it for all the algorithms having default parameter values.Second feature set, i have used only for XGBoost .

- I have trained all the three algorithms with default parameters. Since Support Vector Regression was taking lot of time , so, I took two different set of data to check the results :
  - 75000 data points for training
  - 100000 data points for training

  But, both had constant triptime for test set. Out of all the three algorithms using default values, XGBoost was better in terms of time and RMSLE score on kaggle.Following are initial scores :

Algorithm	Private LB[1]	public LB [1]
SVR	0.88500	0.93268
Neural Network	0.80469	0.86510
XGBoost	0.81446	0.83113

4.2 Refinement

From result of above step I decided to use the XGBoost for tuning the model .For tuning , i chose three parameters(with respective values) : 'min_child_weight': [2,3], 'learning_rate':[0.4,0.5,0.6],'max_depth': [3,4].the values i chose are random values. I tried same thing with feature set 2 , but RMLSE score was higher than the feature set 1.

Feature Set	Private LB[1]	public LB[1]
set 1	0.74086	0.79720
set 2	0.76327	0.85503

[1]Leaderboard Scores of the kaggle competition

The best estimator :

```
XGBRegressor(base_score=0.5, colsample_bylevel=1, colsample_bytree=1, gamma=0,
       learning_rate=0.4, max_delta_step=0, max_depth=4,
       min_child_weight=3, missing=nan, n_estimators=100, nthread=4,
       objective='reg:linear', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1)
```

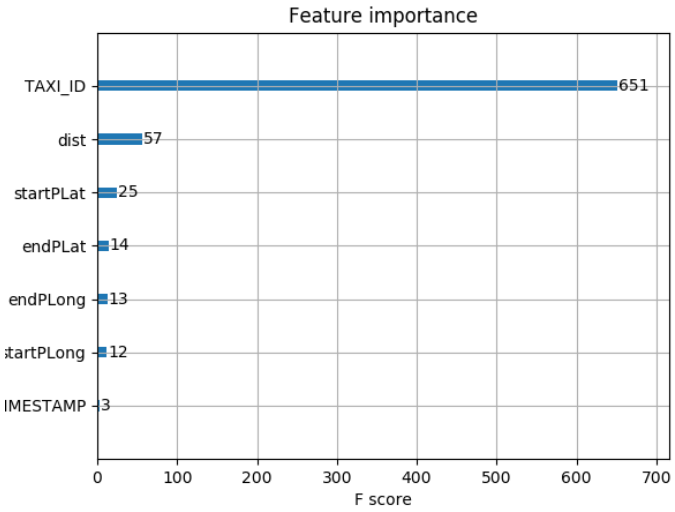
4.3 Justification

As from table above , the best score is (0.74086 (Private LB) and0.79720(public LB)),which beats the benchmark of kaggle competition.But,if we compare the results with winner (0.52528(private LB) and 0.50390(public LB)),its not that good. There is still lot of scope of improvement in the model, but for the significant improvement requires more feature engineering rather than change in model.Example -I tried two different value of n_estimators=[500,5000] in XGBoost ,but the final score didn't improve (0.75042 (Private LB) and 0.83920 (Public LB)).

5 Conclusion

5.1 Free-Form Visualization

The importance of each feature was visualized with the following plot.



The most important feature is TAXI_ID.The strange part is that the feature TIMESTAMP has the lowest importance.But, what i believe is the TIMESTAMP have information about the traffic density on road at that point of time on a particular day.example - On weekends/holiday, traffic will be low during morning time.

5.2 Reflection

The most difficult and one of the best part of the project was correct feature selection.Most of the features didn't have any relevant information.To extract out the best feature for training , required some hit and

trial and as well as knowledge about feature.One of the important learning from this project is XGBoost parameter tuning.

5.3 Improvement

The major points of improvement :

- Better Feature set : I went through some of the solutions of kaggle competition.I found the major difference between my solution and others was the feature set.
- Tuning of Support Vector Regression and Neural Network model :I would like to try the tuning of Support Vector Regression and Neural Network model.But,this may or may not give better result as i tried with Support Vector regression with linear kernel, results didn't improve.

References

[1] https://www.kaggle.com/c/pkdd-15-taxi-trip-time-prediction-ii

[2] http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

[3] http://xgboost.readthedocs.io/en/latest/model.html

[4] http://scikit-learn.org/stable/modules/neural_networks_supervised.html

[5] https://www.kaggle.com/c/nyc-taxi-trip-duration

[6] http://xgboost.readthedocs.io/en/latest/python/python_api.html

[7] https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

[8] https://github.com/hochthom/kaggle-taxi-ii

[9] http://xgboost.readthedocs.io/en/latest/how_to/param_tuning.html