Machine Learning Engineer Nanodegree

Capstone Project for completion of the Udacity Machine Learning Nanodegree

Aditya Pradhan
Nov 11th 2018

I. Definition

Project Overview

This project is a taxi fare prediction (regression) problem that uses a dataset from the NYC Taxi and Limousine Commission, which includes pickup time, geo-coordinates and number of passengers as well as the fare of that taxi ride

Taxi fare prediction is a desirable capability for taxi and transportation companies. It involves predicting a fare for a particular trip given the start, end and data/time when it is occuring. Forecasting price and showing it to consumers allows consumers to determine whether they should take a ride or not. This can influence both aggregate demand (number of consumers) as well as aggregate supply (number of taxi cabs)

In the platform economy, entrants such as Uber and Lyft have created liquid markets using pricing transparency and mobile connectivity to enable markets to form for transportation. Uber and Lyft are the first phase of Mobility-as-a-Service, a concept that is predicated upon having consumers being able to rent transportation for a price as a utility.

Predicting fares is the first step of many towards the MaaS revolution. This project is a current Kaggle playground competition called 'Taxi Fare Prediction'. The task is predicting the fare amount (inclusive of tolls) for a taxi ride in New York City given the pickup and dropoff locations.

More details can be seen at https://www.kaggle.com/c/new-york-city-taxi-fare-prediction

Some approaches taken to this problem that were published online are as below:

[1] Machine learning for the prediction of railway fares
https://www.ke.tu-darmstadt.de/lehre/arbeiten/bachelor/2013/Hirschmann_Fabian.pdf

[2] Fare and duration prediction: a study of NY taxi rides
http://cs229.stanford.edu/proj2016/report/AntoniadesFadaviFobaAmonJuniorNewYorkCityCabPricing-report.pdf

[3] Taxi Fare Rate Classification Using Deep Networks
https://www.researchgate.net/publication/324706525/download

[4] Travel Time Prediction using Machine Learning
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.701.3918&rep=rep1&type=pdf

[5] Network scale travel time prediction using Deep Learning
https://trid.trb.org/view/1494651

Since this is a Kaggle problem there are some papers on solving it:
[6] NYC Taxi Fare Prediction
https://cseweb.ucsd.edu/classes/wi17/cse258-a/reports/a077.pdf

As well as several approaches that are shared and documented as kaggle kernels at
https://www.kaggle.com/c/new-york-city-taxi-fare-prediction/kernels

Problem Statement

The problem is to predict the fare of a taxi ride in NYC given the pickup time and date, pickup coordinates and destination coordinates. The fare is a dependent variable while the pickup time and time, pickup and destination coordinates are the independent variables.

This is clearly a **regression problem**, in order to solve this a regression model will be setup that outputs the predicted fare. The model will then be trained on the dataset in order to be able to predict fares

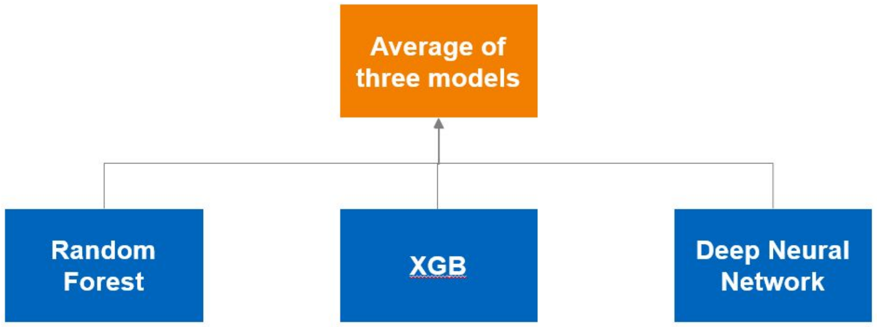
The following approach will be taken:

- Outliers will be removed as they will skew the model - an outlier will be defined as any row with a pick up or dropoff outside of the NYC metropolitan area as defined by coordinates
    - Null or empty rows will be dropped since there are over 50M rows of data
- Currently the data only has positional information (latitude and longitude of pickup and dropoff), pickup time information and # of passengers
    - The pickup time will be used to extract the day of the week as well as the hours of the day in order to capture intraday variation in traffic

- Manhattan distance (absolute difference of latitude and longitude) will be used as one distance metric . All distances will be standardized using a MinMaxScaler. This will be continuous variables

Solution approach

1. Clean data (see section on Data Preprocessing for details)
2. Perform feature engineering in order to create meaningful features that are likely correlated with the fare (see section on Data Preprocessing)
3. Create an ensemble of three different models: Random Forest, XGB and Deep Neural Network as detailed in steps [4],[5] and [6]



4. Train a random forest model
    a. Depth of tree will be used to tune the model
    b. Regularization to prevent overfitting
5. Train an xgboost model
    a. Max depth will be used as a hyperparameter to prevent overfitting
    b. Regularization to prevent overfitting
6. Train a separate fully connected Deep Neural Network (DNN) model. This will be a fully connected neural network with 2 - 4 layers in order to have enough model capacity to solve this problem
    a. Regularization to prevent overfitting
    b. Dropout to prevent overfitting
    c. Learning rate tuning to achieve a DNN that learns efficiently
7. Create an ensemble by averaging the prediction from steps [4], [5] and [6] . It is expected that the ensemble results in better performance (lower RMSE) than any individual method

Metrics

Evaluation Metric

The evaluation metric is the root mean-squared error or RMSE. RMSE measures the difference between the predictions of a model, and the corresponding ground truth. A large RMSE is equivalent to a large average error, so smaller values of RMSE are better.

RMSE is given by:

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

where y_i is the *ith* observation and

$$\hat{y}_i$$

is the prediction for that observation.

RMSE is the appropriate metric for this prediction task as

- It will measure the distance of the prediction from the target value
- Larger errors will be penalized more than smaller errors. This is key when the domain is considered whereby a prediction error in the order of cents will be acceptable and forgiven by both businesses and passengers whereas errors on the order to dollars are unlikely to the acceptable.
- RMSE is a standard metric that is well understood by most practitioners

RMSE was implemented as below in the code:

```
from sklearn.metrics import mean_squared_error

def rmse_error(y_true,y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))
```

II. Analysis

Data Exploration

Datasets and Inputs

File descriptions

The full dataset is available at https://www.kaggle.com/c/new-york-city-taxi-fare-prediction/data

- train.csv - Input features and target fare_amount values for the training set (about 55M rows).

Data fields

ID

- key - Unique string identifying each row in both the training and test sets.

Features

- pickup_datetime - timestamp value indicating when the taxi ride started.
- pickup_longitude - float for longitude coordinate of where the taxi ride started.
- pickup_latitude - float for latitude coordinate of where the taxi ride started.
- dropoff_longitude - float for longitude coordinate of where the taxi ride ended.
- dropoff_latitude - float for latitude coordinate of where the taxi ride ended.
- passenger_count - integer indicating the number of passengers in the taxi ride.

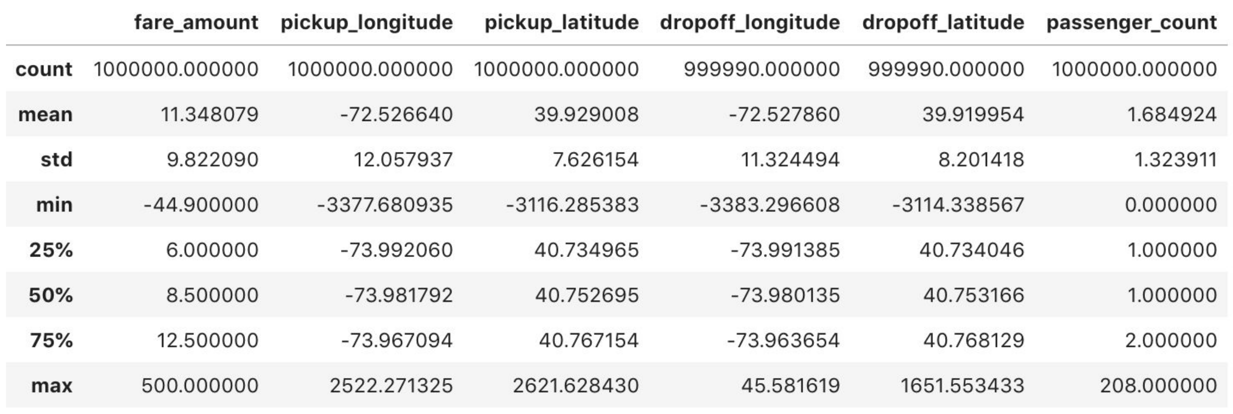
Target / Variable to predict

- fare_amount - float dollar amount of the cost of the taxi ride. This value is what is being predicted

The data has been provided in csv format as below:
Data sample:

	0	1
key	2009-06-15 17:26:21.0000001	2010-01-05 16:52:16.0000002
fare_amount	4.5	16.9
pickup_datetime	2009-06-15 17:26:21	2010-01-05 16:52:16
pickup_longitude	-73.8443	-74.016
pickup_latitude	40.7213	40.7113
dropoff_longitude	-73.8416	-73.9793
dropoff_latitude	40.7123	40.782
passenger_count	1	1

Statistics summarizing the data read in are:

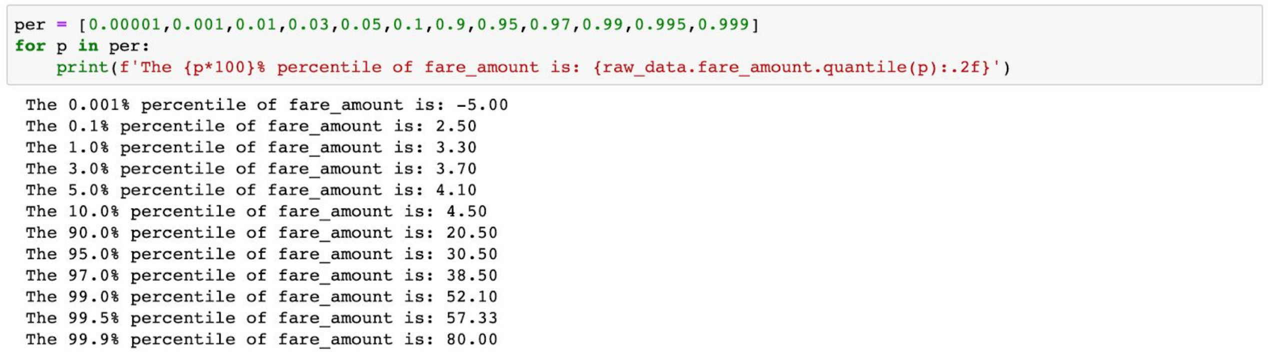


	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	1000000.000000	1000000.000000	1000000.000000	999990.000000	999990.000000	1000000.000000
mean	11.348079	-72.526640	39.929008	-72.527860	39.919954	1.684924
std	9.822090	12.057937	7.626154	11.324494	8.201418	1.323911
min	-44.900000	-3377.680935	-3116.285383	-3383.296608	-3114.338567	0.000000
25%	6.000000	-73.992060	40.734965	-73.991385	40.734046	1.000000
50%	8.500000	-73.981792	40.752695	-73.980135	40.753166	1.000000
75%	12.500000	-73.967094	40.767154	-73.963654	40.768129	2.000000
max	500.000000	2522.271325	2621.628430	45.581619	1651.553433	208.000000

The magnitude of the max values shows that there are likely outliers present in the data (e.g. the fare_amount varies from -$45 to $500)

- The negative values should clearly be removed
- The max fare of $500 seems absurdly high
- The range of latitude and longitude are absurd and likely data issues (e.g. the max values do not make sense)

To see how many outliers there are a percentile spread of the data was created as below:



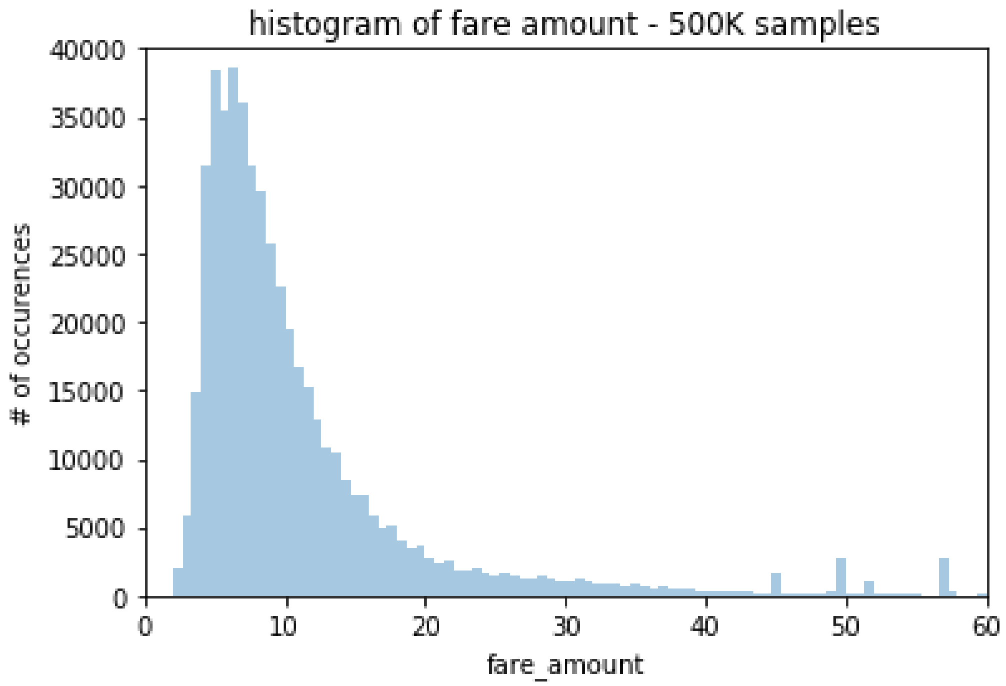
```
per = [0.00001,0.001,0.01,0.03,0.05,0.1,0.9,0.95,0.97,0.99,0.995,0.999]
for p in per:
    print(f'The {p*100}% percentile of fare_amount is: {raw_data.fare_amount.quantile(p):.2f}')
```

```
The 0.001% percentile of fare_amount is: -5.00
The 0.1% percentile of fare_amount is: 2.50
The 1.0% percentile of fare_amount is: 3.30
The 3.0% percentile of fare_amount is: 3.70
The 5.0% percentile of fare_amount is: 4.10
The 10.0% percentile of fare_amount is: 4.50
The 90.0% percentile of fare_amount is: 20.50
The 95.0% percentile of fare_amount is: 30.50
The 97.0% percentile of fare_amount is: 38.50
The 99.0% percentile of fare_amount is: 52.10
The 99.5% percentile of fare_amount is: 57.33
The 99.9% percentile of fare_amount is: 80.00
```

The 0.2% percentile to the 99% percentile range was chosen as it appeared to be a reasonable range for the fare to be in. Any data outside of that was removed as it was deemed to be a outlier

Exploratory Visualization

A histogram of the fare price was made in order to observe the distribution:

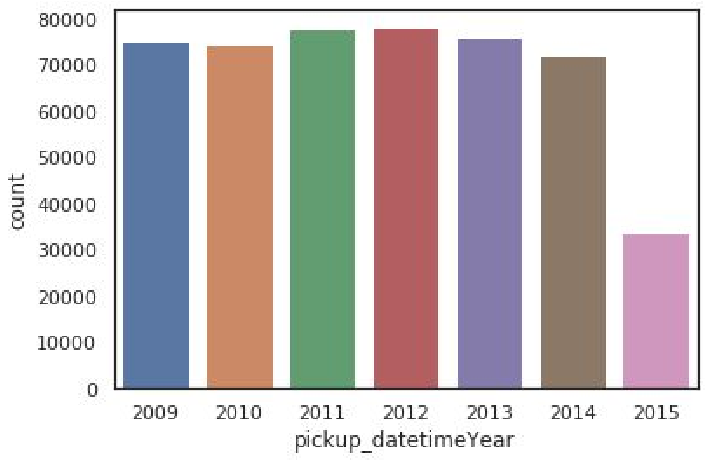


We see above that this is a skewed normal distribution with a mean of $11.35 and an s.d. of $9.83.

There are interesting bumps at $45, $50 and $58. These values are more prevalent than the distribution would expect. My hypothesis is that these bumps are present due to standard fares from popular point to point locations (e.g from airports to downtown). These are likely popular routes where the taxi business has high demand and also has a relatively monopoly (regulated monopoly position by the government)

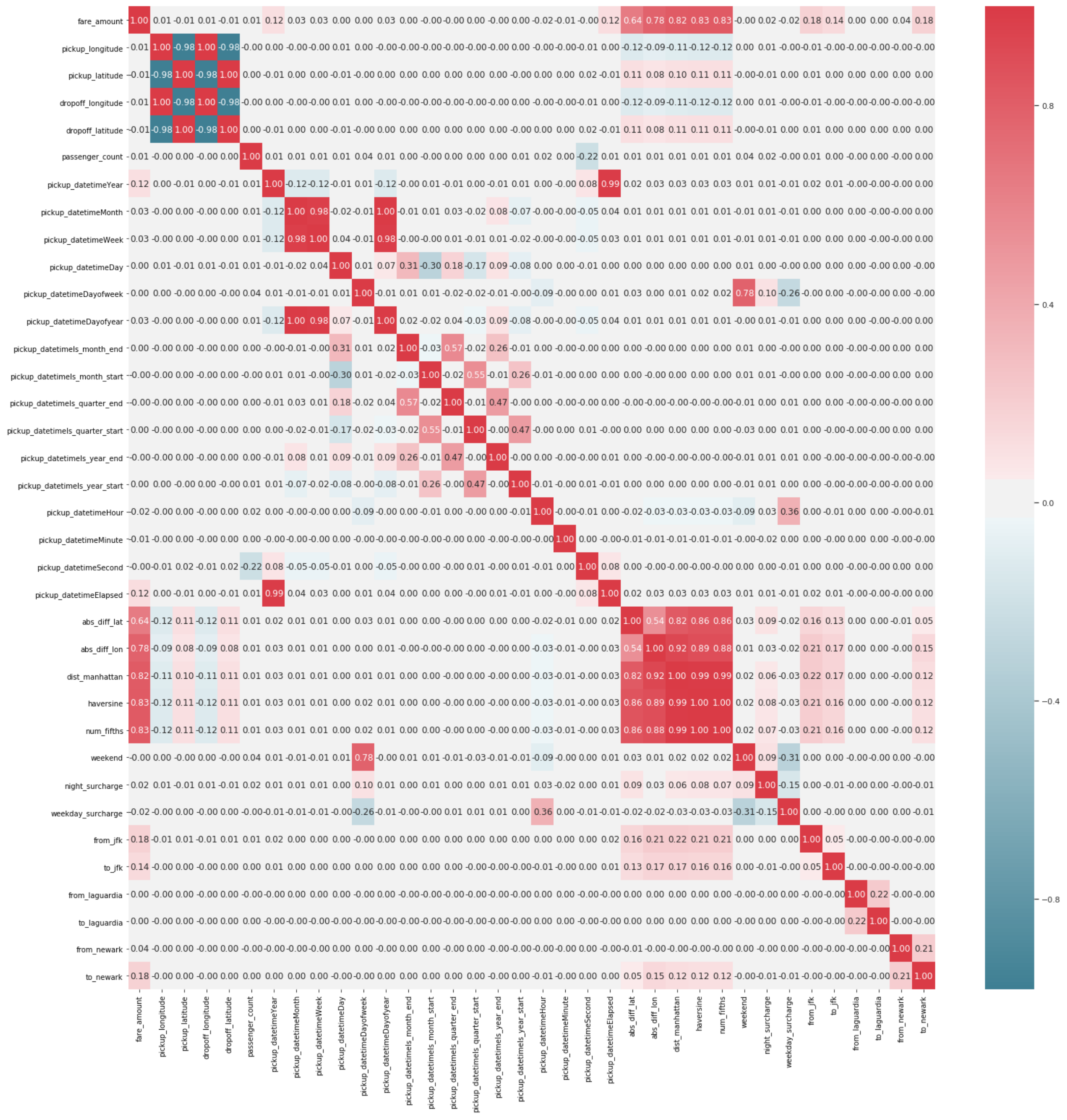
Some exploratory visualization was done to understand the dataset better. For example:

- The dataset contains data from many years:



- Data was balanced in terms of days of the week

The correlation heatmap (below) shows that distance is highly correlated as are the to / from airport indicator variables



Algorithms and Techniques

Random Forest

Random Forest Regression was chosen as it is a highly versatile machine learning algorithm and generally performs well on small datasets with tabular data

XGBoost
Boosted trees tend to work well on smaller datasets and tabular data. Boosted trees are a totally different technique to baging therefore it was expected that any bias in the model / errors in the model are uncorrelated with those of random forest

Deep Neural Network
A Deep Neural Network was chosen to incorporate a third technique. This generally performs better on a greater amount of data. As there are 55M rows this analysis is approaching the territory / scale where a DNN would be advantageous

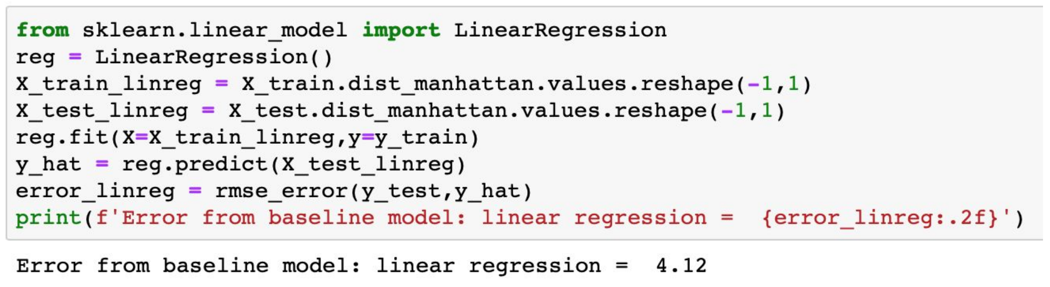
Ensemble
An ensemble of the three models (using the mean) was created in order to create a model aggregation that is better than the individual models

Benchmark

A linear regression model was created in order to have a reasonable baseline. The correlation between manhattan distance and fare amount is 0.82 therefore this appeared to be the simplest model to build as a baseline.

Intuitively modeling price as a linear function of distance would be consistent with most customers expectation

A model was constructed as below using the scikit-learn library:



```python
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
X_train_linreg = X_train.dist_manhattan.values.reshape(-1,1)
X_test_linreg = X_test.dist_manhattan.values.reshape(-1,1)
reg.fit(X=X_train_linreg,y=y_train)
y_hat = reg.predict(X_test_linreg)
error_linreg = rmse_error(y_test,y_hat)
print(f'Error from baseline model: linear regression =  {error_linreg:.2f}')
```

```
Error from baseline model: linear regression =  4.12
```

The RMSE of the linear regression was 4.12. This is the value with which model performance of more advanced techniques should be compared

III. Methodology

Data Preprocessing

The size of the dataset (55M rows) meant that simply using pandas and reading all data into memory using standard methods was not reasonable

In order to store the file in a manner that facilitated rapid iteration, the feather format (https://github.com/wesm/feather) was used. This writes the data to a format that is similar to how data is organized in memory and therefore resulted in much faster read/write times than the typical pandas read_csv call takes.

500K lines were sampled and used as the dataset for this analysis. 80% were used as the training set and 20% as the test set.

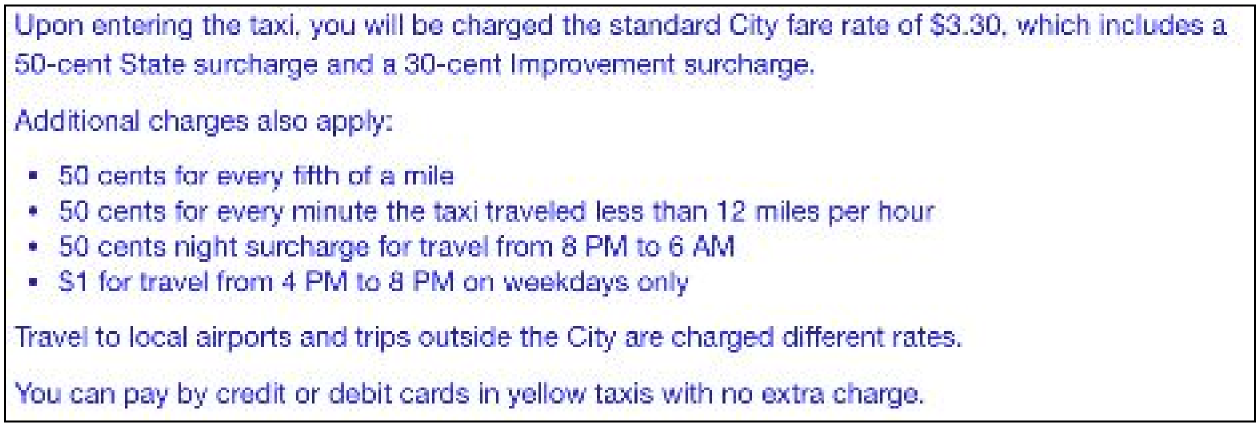
Feature Engineering

The pickup data time is a single variable and was split into several variables in order to capture some of the intricacies of the daily cycle (morning vs. afternoon vs. evening) as well as weekly variation (weekday vs weekend) and broader traffic patterns (lower traffic during holiday periods etc.)

```python
def add_datepart(df, fldname, drop=True, time=False):
    "Helper function that adds columns relevant to a date."
    fld = df[fldname]
    fld_dtype = fld.dtype
    if isinstance(fld_dtype, pd.core.dtypes.dtypes.DatetimeTZDtype):
        fld_dtype = np.datetime64

    if not np.issubdtype(fld_dtype, np.datetime64):
        df[fldname] = fld = pd.to_datetime(fld, infer_datetime_format=True)
    targ_pre = re.sub('[Dd]ate$', '', fldname)
    attr = ['Year', 'Month', 'Week', 'Day', 'Dayofweek', 'Dayofyear',
            'Is_month_end', 'Is_month_start', 'Is_quarter_end', 'Is_quarter_start', 'Is_year_end', 'Is_year_start']
    if time: attr = attr + ['Hour', 'Minute', 'Second']
    for n in attr: df[targ_pre + n] = getattr(fld.dt, n.lower())
    df[targ_pre + 'Elapsed'] = fld.astype(np.int64) // 10 ** 9
    if drop: df.drop(fldname, axis=1, inplace=True)
```

To understand the domain better, the NYC yellow taxi guide was consulted.


Upon entering the taxi, you will be charged the standard City fare rate of $3.30, which includes a 50-cent State surcharge and a 30-cent Improvement surcharge.

Additional charges also apply:

- 50 cents for every fifth of a mile
- 50 cents for every minute the taxi traveled less than 12 miles per hour
- 50 cents night surcharge for travel from 8 PM to 6 AM
- $1 for travel from 4 PM to 8 PM on weekdays only

Travel to local airports and trips outside the City are charged different rates.

You can pay by credit or debit cards in yellow taxis with no extra charge.

Source: https://www1.nyc.gov/nyc-resources/service/1271/yellow-taxi-fares

Based on these findings, a number of new features were created:

1. **Haversine** = Distance in miles converted from degrees using the haversine distance
2. **Dist_Miles_Fifths** = Number of miles in increments of 0.2 miles ('fifths')
3. **Night_Surcharge** = Night surcharge is the pickup time is between 8PM and 6AM
4. **Weekday_Surcharge** = Weekday surcharge if the pickup time starts between 4PM and 8PM on a weekday

NYC Airport	JFK	LaGuardia	Newark
Coordinates	40.6413° N, 73.7781° W	40.7769° N, 73.8740° W	40.6895° N, 74.1745° W

Source: Google

Using these coordinates, the variables from_jfk, to_jfk, from_laguardia, to_laguardia, from_newark, to_newark were constructed in order to capture if trips were to or from a New York City area airport and which one.

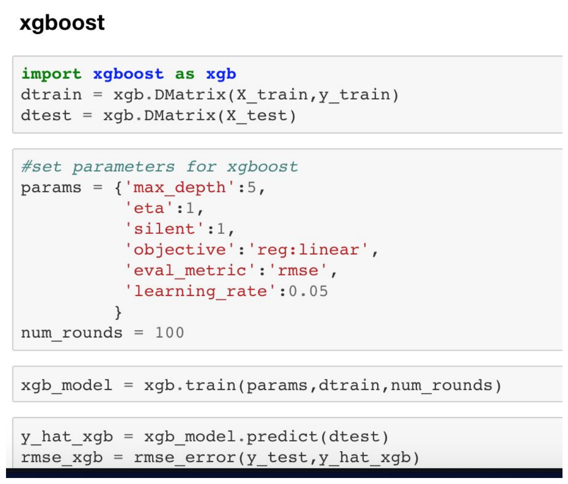
Implementation

Random Forest Model

```
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import make_pipeline

reg = make_pipeline(StandardScaler(),
                    RandomForestRegressor(n_estimators=100,max_depth=30,n_jobs=-1,min_samples_leaf=1,verbose=1))
reg.fit(X_train,y_train)
y_hat_rf = reg.predict(X_test)
error_rf = rmse_error(y_test,y_hat_rf)
print(error_rf)
```

A random forest model was implemented using scikit learn (as above). 100 estimators was used in order to have enough trees while still training within a reasonable amount of time. A max depth of 30 was found to have the best results



xgboost

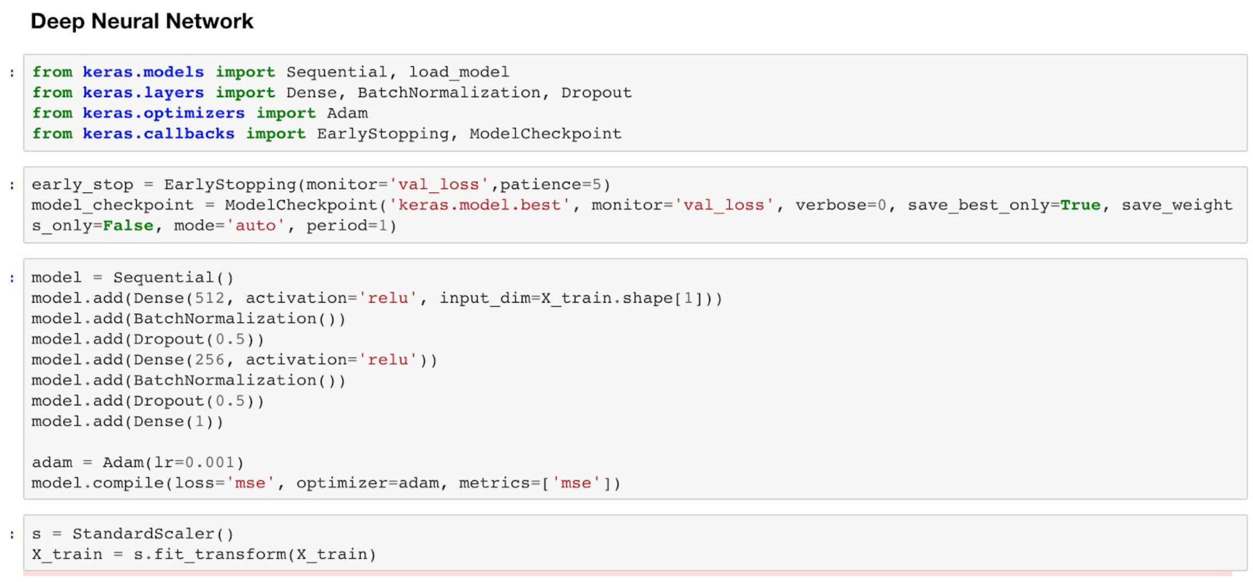
```
import xgboost as xgb
dtrain = xgb.DMatrix(X_train,y_train)
dtest = xgb.DMatrix(X_test)
```

```
#set parameters for xgboost
params = {'max_depth':5,
          'eta':1,
          'silent':1,
          'objective':'reg:linear',
          'eval_metric':'rmse',
          'learning_rate':0.05
         }
num_rounds = 100
```

```
xgb_model = xgb.train(params,dtrain,num_rounds)
```

```
y_hat_xgb = xgb_model.predict(dtest)
rmse_xgb = rmse_error(y_test,y_hat_xgb)
```

XGBoost was used leveraging the python xgb library. A max depth of 5, learning rate of 0.05 and 100 rounds showed the best performance in terms of RMSE of the validation set without overfitting.



Deep Neural Network

```
: from keras.models import Sequential, load_model
  from keras.layers import Dense, BatchNormalization, Dropout
  from keras.optimizers import Adam
  from keras.callbacks import EarlyStopping, ModelCheckpoint
```

```
: early_stop = EarlyStopping(monitor='val_loss',patience=5)
  model_checkpoint = ModelCheckpoint('keras.model.best', monitor='val_loss', verbose=0, save_best_only=True, save_weight
  s_only=False, mode='auto', period=1)
```

```
: model = Sequential()
  model.add(Dense(512, activation='relu', input_dim=X_train.shape[1]))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))
  model.add(Dense(256, activation='relu'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))
  model.add(Dense(1))

  adam = Adam(lr=0.001)
  model.compile(loss='mse', optimizer=adam, metrics=['mse'])
```

```
: s = StandardScaler()
  X_train = s.fit_transform(X_train)
```

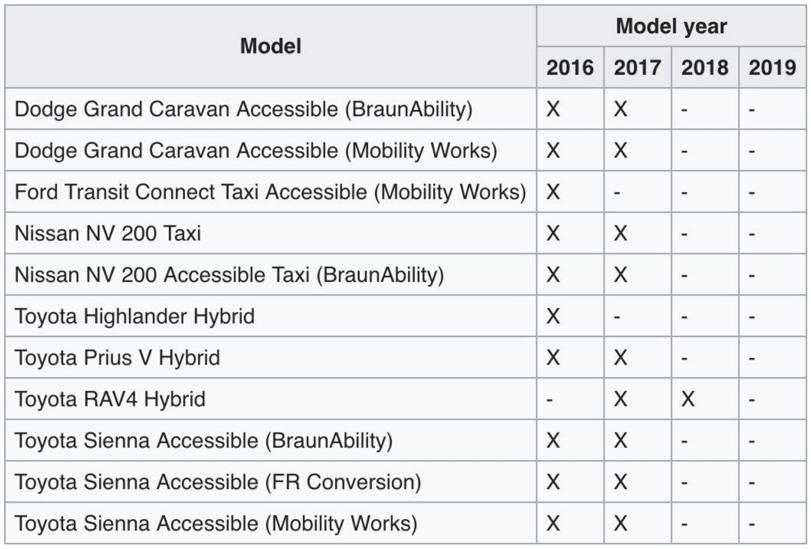
A Deep Neural Network (DNN) wasc created using Keras as above. BatchNormalization and Dropout were added in order to prevent overfitting.

Upon training several times the best learning rate was found to the 0.001 and training for longer epochs (100 epochs).

During training the early stopping setting was used with a patience of five epochs to keep training until performance did not improve for five epochs.

Refinement

- Modelling traffic and particularly if a route is likely to hit a traffic hotspot during rush hour
- Size of car / make of car (i.e. it is likely that an ‘XL’ taxi with greater capacity has a higher fare than a ‘normal’ taxi). A list of approved vehicles on wikipedia shows that there is significant variation including hybrids



Model	Model year			
	2016	2017	2018	2019
Dodge Grand Caravan Accessible (BraunAbility)	X	X	-	-
Dodge Grand Caravan Accessible (Mobility Works)	X	X	-	-
Ford Transit Connect Taxi Accessible (Mobility Works)	X	-	-	-
Nissan NV 200 Taxi	X	X	-	-
Nissan NV 200 Accessible Taxi (BraunAbility)	X	X	-	-
Toyota Highlander Hybrid	X	-	-	-
Toyota Prius V Hybrid	X	X	-	-
Toyota RAV4 Hybrid	-	X	X	-
Toyota Sienna Accessible (BraunAbility)	X	X	-	-
Toyota Sienna Accessible (FR Conversion)	X	X	-	-
Toyota Sienna Accessible (Mobility Works)	X	X	-	-

IV. Results

Model Evaluation and Validation

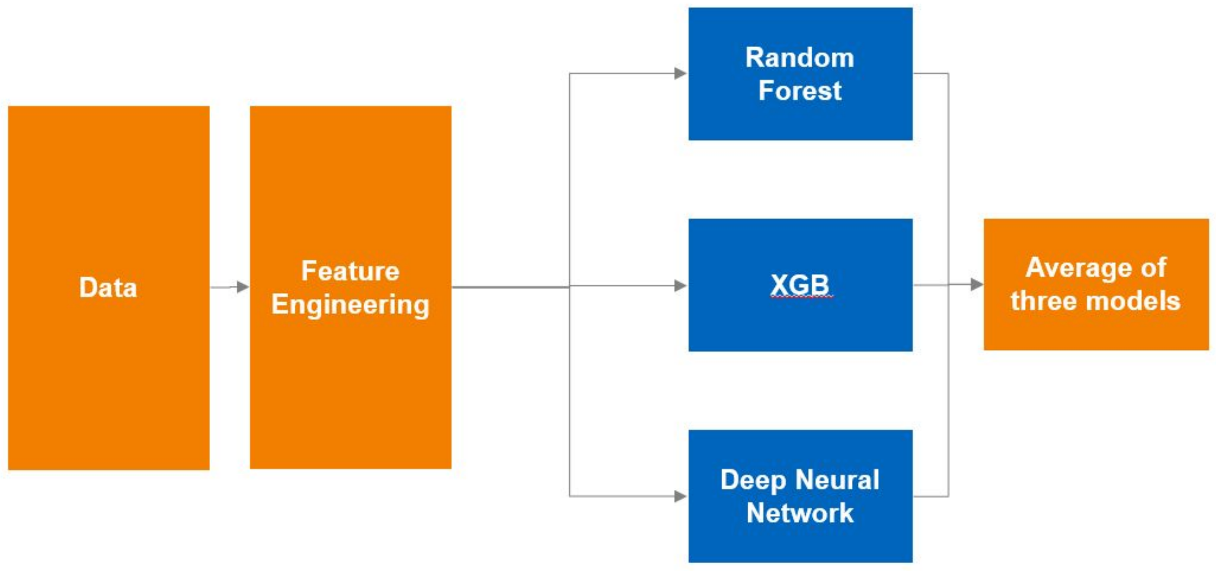
Justification

Model	Baseline Lin Reg	Random Forest	XGBoost	Neural Net	Ensemble
RMSE	5.52	4.37	4.32	4.73	4.308
Improvement	0.00%	20.83%	21.74%	14.31%	21.96%

The ensemble model was a 22% improvement over the baseline. This is a fairly substantial improvement. The ensemble performs better than all models though only slightly better than XGBoost.

V. Conclusion

Upon reflection, it seems that the approach taken (as depicted graphically below) was a reasonable approach and has led to improvement over the baseline.



Testing on Kaggle showed that this entry would be 925 out of 1488 entries (top 62%). This is certainly not good performance and there is a lot of room for improvement.

Improvement

There are several improvements that can be made:

- Adding supplementary datasets could augment the breadth and depth of information the model has to consider and could likely result in more correlations being present. In particular there are three data sets that could be useful:
    - Traffic
    - Weather
    - Accidents

- Collecting the dropoff time in order to have a dropoff time feature is likely to significantly improve the performance. The dropoff time would allow the trip duration to be known. Trip duration is likely correlated with the fare amount as the taxi fare explicitly has a dependency on speed ($0.50 for every minute the taxi travelled less than 12 miles an hour). In heavy traffic this is likely a significant predictor.
- Training with the full data set (55M) rather than ~0.5M would lead to greater generalizability. The data subset used appeared to be representative but there may be nuances. A big data framework (e.g. Spark) could have been useful to process the large amounts of data
- Segmenting data set and training models on short haul trips differently from long haul trips could be an approach in order to train models for particular types of trip rather where certain factors might be more important than others rather than training a model on the entire problem space