Machine Learning Engineer Nanodegree

# Capstone Project (REVISED)

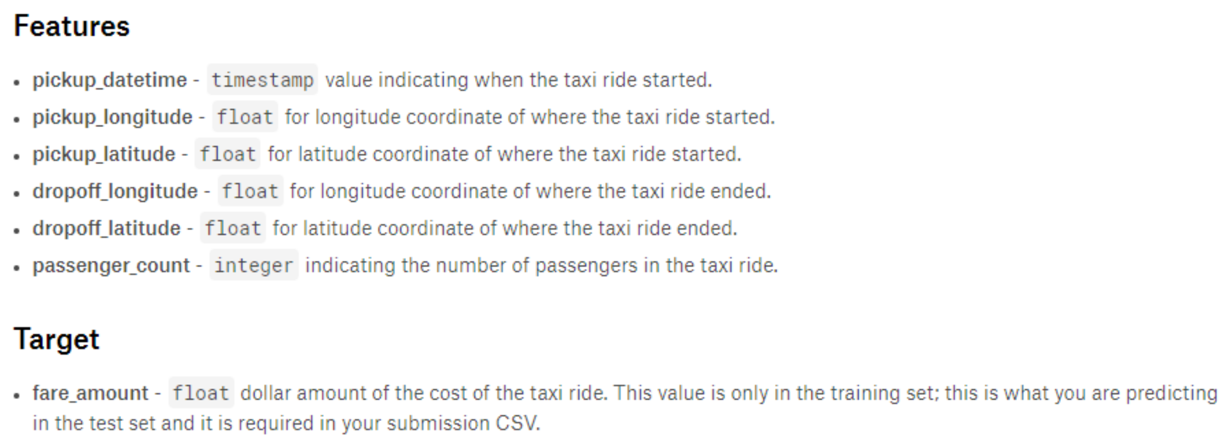
Vik Singh
November 23, 2018

## I. Definition

### Project Overview

The project undertaken was the Kaggle competition (https://www.kaggle.com/c/new-york-city-taxi-fare-prediction). The project deals with prediction of taxi fares in the New York city. The project involves around utilizing large dataset (55 million rows of data for training) that consists of data points on pickup/dropoff latitude and longitude, number of passengers, pickup date, time and corresponding fare of the taxi. The goal of the project is to build a model utilizing this available dataset that can predict the fare of the taxi in the future given above mentioned variables as input values. This model then can possibly be made available as phone app, with which a person can get good estimate of the fare that will charged by the taxi.  The problem resides in the domain of machine learning, where a model "learns" from the available data and then can predict target values, fare values in current context, on the new input values/feature values.

The dataset for solving this competition problem was provided by the above mentioned Kaggle competition itself (https://www.kaggle.com/c/new-york-city-taxi-fare-prediction/data). Two types of dataset were provided, namely, 'train.csv' and 'test.csv'. Both datasets were numeric text. The 'train.csv' consist of ~55 million rows of data that is to be utilized for training the Deep Neural Network (DNN) regression model that is described later in this report. The 'train.csv' is comprised of 6 features/variables along with target feature which are described in the picture below,



### 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

- **fare_amount** - `float` dollar amount of the cost of the taxi ride. This value is only in the training set; this is what you are predicting in the test set and it is required in your submission CSV.

The ‘test.csv’ consist of ~10,000 rows of input features/variables that is to be utilized for testing the performance of the final DNN regression model and contains the same ‘Features’ as depicted in the above picture. The ‘Target’ feature value is then supposed to be predicted by the DNN regression model for these 10K rows of testing data. Apart from this my personal motivation, as is also mentioned in the proposal report, is to become familiar with working and applying machine learning techniques to big training datasets.

## Problem Statement

The problem that needs solving is accurate estimation of the taxi fare given pickup/dropoff latitude and longitude, number of passengers, pickup date and time. This will involve learning from available data on these features and corresponding fare value to build a model that can give predicted fare amount, given above feature values as input. This is thus a regression problem, the attempt is to the solve the problem via nonlinear regression techniques to capture the nonlinear relationship between features and fit the model to the training data.

As described in the project overview, the dataset consists of training and testing data, where training data will be utilized to fit a regression model and then that model will be used to give prediction of fare values/Target feature on the testing dataset. The strategy that will be utilized to solve this problem will be to,

First analyze, preprocess and clean the training dataset to remove completely illogical (NULL) and unnecessary datapoints (outliers). The preprocessing part will also involve engineering the training and testing dataset to remove the ‘pickup_datetime’ feature values (timestamp) and instead add in features such as ‘hour_of_day’ and ‘year’ extracted from this timestamp. Additionally, I’ll be scaling both the datasets to remove skewness from the feature weights (also discussed later in the report).

Then, Deep Neural Network will be constructed, and this network will be optimized for parameters based DNN performance on the small subset of training data.  The final DNN network will then be trained utilizing the training dataset and the best model (DNN weights) will be saved for prediction on testing set.

This best DNN model will then be applied to testing dataset. The expected output is the 'fare_amount' target feature. This prediction of the target feature on the testing dataset will then be submitted to the Kaggle competition which will score & rank the submission based on the RMSE (root mean squared error) value, which is calculated the competition algorithm itself.

**Metrics**

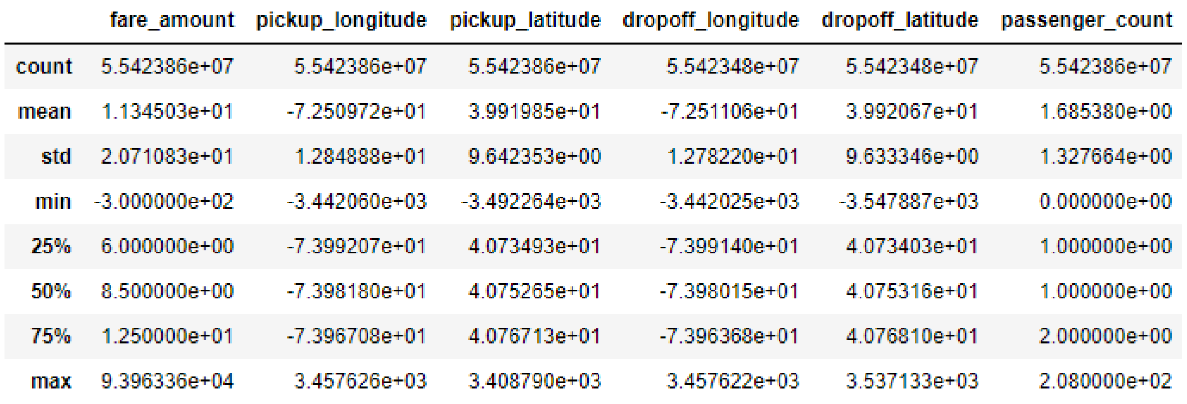
Since this is a regression problem and not a classification problem, the metric that will be used is RMSE (root mean squared error), between the predicted fare value and the actual fare value. A large RMSE is equivalent to large average error and vice-versa for small RMSE values, between the predicted value and the actual value. Furthermore, it gives error value in the units of the data that is being utilized for fitting the model. Additionally, the Kaggle competition also requires this metric to be utilized for model evaluation purposes.

## II. Analysis

### Data Exploration and Exploratory Visualization

The notebook that deals with this aspect of data exploration is named as "NY_taxi_feature_Cleaning_Engineering_Scaling" and is provided in both .html and .ipynb format. The dataset given for training in this Kaggle competition contained 55 million rows. One of the first things that was done was to optimize memory storage (refer above notebook) so that all the 55 million rows of data can be loaded to pandas dataframe quickly.  This was done by changing the data type of the features and then chunks of the dataframes loaded with data from .csv file were appended. Lastly, the loaded dataframe with raw data containing all the 55 million rows was saved in .feather format so that dataframe can loaded later on quickly when needed. Same procedure was utilized to save the testing data provided by Kaggle competition.

The statistics of these 55 million rows of training data was then checked (See the table below).



| | fare_amount | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passenger_count |
|---|---|---|---|---|---|---|
| count | 5.542386e+07 | 5.542386e+07 | 5.542386e+07 | 5.542348e+07 | 5.542348e+07 | 5.542386e+07 |
| mean | 1.134503e+01 | -7.250972e+01 | 3.991985e+01 | -7.251106e+01 | 3.992067e+01 | 1.685380e+00 |
| std | 2.071083e+01 | 1.284888e+01 | 9.642353e+00 | 1.278220e+01 | 9.633346e+00 | 1.327664e+00 |
| min | -3.000000e+02 | -3.442060e+03 | -3.492264e+03 | -3.442025e+03 | -3.547887e+03 | 0.000000e+00 |
| 25% | 6.000000e+00 | -7.399207e+01 | 4.073493e+01 | -7.399140e+01 | 4.073403e+01 | 1.000000e+00 |
| 50% | 8.500000e+00 | -7.398180e+01 | 4.075265e+01 | -7.398015e+01 | 4.075316e+01 | 1.000000e+00 |
| 75% | 1.250000e+01 | -7.396708e+01 | 4.076713e+01 | -7.396368e+01 | 4.076810e+01 | 2.000000e+00 |
| max | 9.396336e+04 | 3.457626e+03 | 3.408790e+03 | 3.457622e+03 | 3.537133e+03 | 2.080000e+02 |

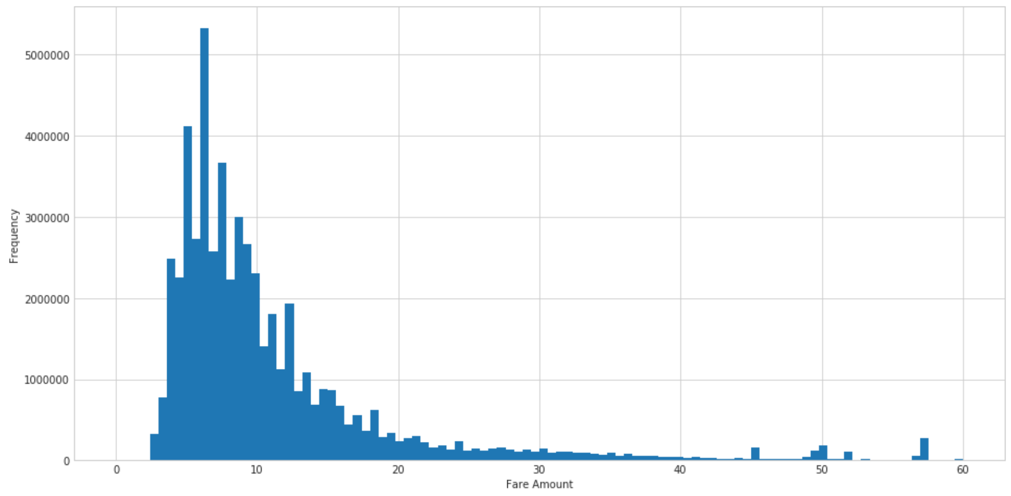
Some significant observations and problems can be deduced by visualizing this table. They are noted below.

- Negative fare amount, these are not typical scenarios where passenger is supposed to be charged for riding the taxi.
- Very high value for the maximum of the fare_amount feature (93963).
- NY city latitude is ~ 40 and Longitude is ~ 73. Some of the values of the latitude and longitude were extreme, in fact, the latitude values can only go from -90 to 90, hence values like -3492.26 and 3408.70 are erroneous. These "bad" values were noticed in both pickup and dropoff latitude values.
    - The STD for pickup_latitude was ~9.64, again outlier or erroneous values as 1 degree change in latitude is equivalent to 69 miles.
- These erroneous values were also noted in pickup_longitude and dropoff_longitude feature values of the training dataset. For example, -3442.060 and 3457.62
    - Similar large STD value as latitude values were also noticed.
- Unreasonable passenger count: for example, entry values of 0 and 208 for the passenger_count feature.

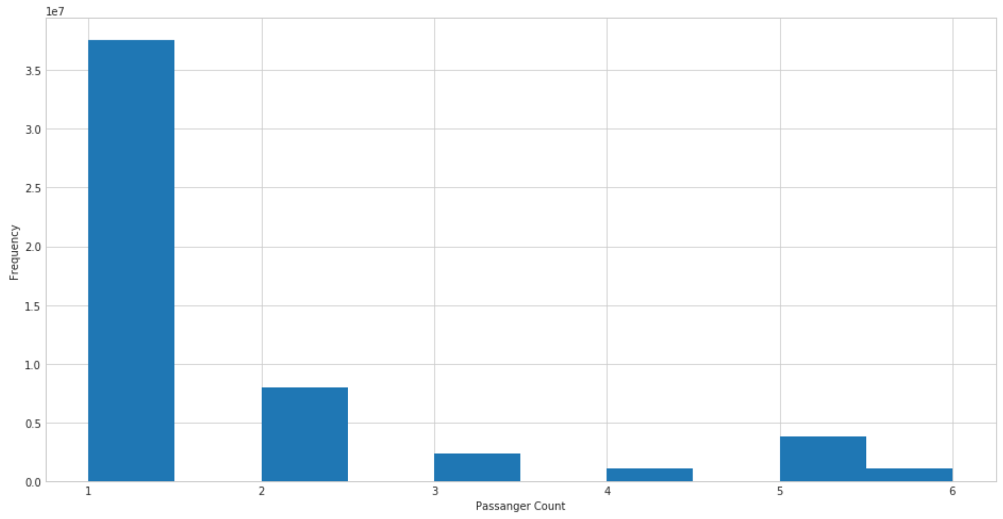
Apart from these points, pickup_datetime could also be an important feature, however, it is given in the timestamp format, hence some preprocessing/engineering to extract out and time and date from the timestamp would also be necessary. Additionally, some null datapoints are also present in the training set and hence they need to be removed too.

Moreover, some visualization is presented below that further provides insight regarding the dataset (See also Methodology section).

First, histogram for the frequency of the fare values in the 'fare_amount' feature within the range [$0 - $60] is presented below. This distribution suggests that most of the fare values of the training data points reside within the range [$0 - $60] (in fact, the distribution is most dense in the [$0 - $20] range). Hence, for the DNN model to fit to the training data well, the data points in the training set that are outside the [$0 - $60] range must be removed. This would help build/train appropriate model that is able to predict fare values for short rides that the dataset seem to suggest are the most frequent.



Second, histogram for the frequency of the values in the 'Passenger_count' feature is also provided within the range of [1-6] total passengers (See below). This plot suggests that most of the data in 'Passenger_count' feature reside within this range, with highest density of datapoints within [1-3] 'Passenger_count' range. Hence, it would be appropriate to remove all the data points from the training set that are outside the range of [1-6] number of passengers as those datapoints aren't general representation of the number of passengers that most taxi rides would occupy. Again, this will help with obtaining better predictions from the trained DNN model.



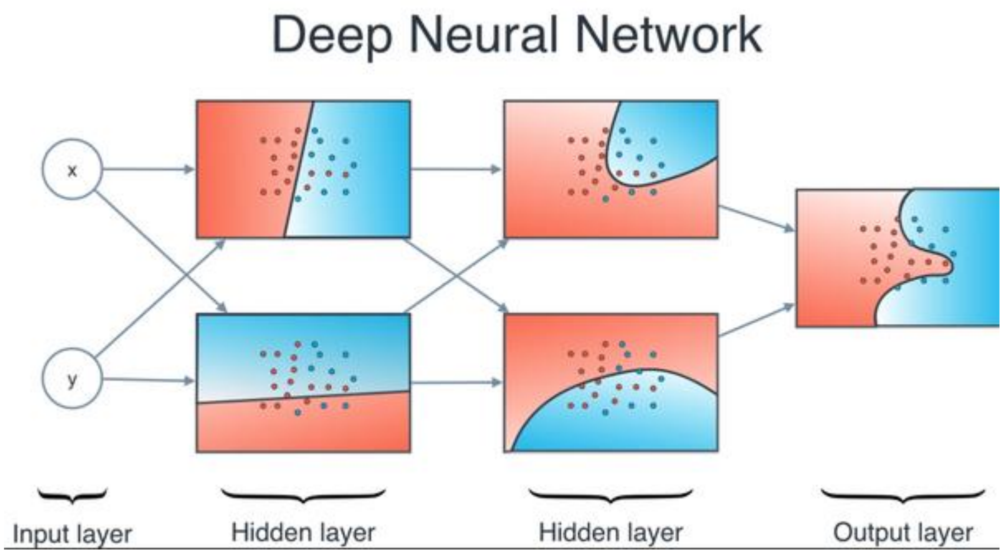
These issues are addressed in the next section where the preprocessing procedure and final statistics on the cleaned/optimized and engineered data is listed.

**Algorithms and Techniques**

Exploring the data provides insight that some preprocessing and engineering needs to be done. We will go over in detail in next section. Additionally, we would also need to utilize standard scaling to make sure that some features are not weighted more than other features just because of their range of values. This will transform the features into distribution with 0 mean and 1 standard deviation.

General regression techniques are widely employed to solve tasks where the goal is to predict continuous values. The technique I'll be using is Deep Neural Network (DNN) regression. A Deep Neural Network (DNN) is a weighted linear combination of the input features passed through an activation function (which adds nonlinearity) to obtain a highly non-linear model. DNN is widely know as Universal function approximator, suggesting that it can approximate the output of any function whether linear or non-linear.

An example picture is provided below which shows how a weighted linear combination of inputs (first hidden layer) is passed through activation function (such as **ReLu**) and then combined further with weights of the second hidden layer (also passed through activation function) to properly model a nonlinear boundary.



The advantages of non-linear models are that they are able to capture non-linear relationships among the features hence are able to fit to the training data well, resulting in a trained model that has high predictive capability corresponding to unseen data for the same set of features the model is trained on.

In context of the current Kaggle competition problem, The benchmark model also discussed later in the report is a simple linear model, however, it has a poor performance on the testing set with an RMSE of ~$6, suggesting that the features of the dataset have non-linear relationship with each other, thus validating that the use of non-linear machine learning models such as Deep Neural Network would prove useful for increasing the predictive performance and lower the RMSE values. Hence, I'll be using the DNN model with preprocessed features in the training set (input to the model) and the output of each hidden layer passed through an **ReLu** activation function. The final output of the model will be the target feature, which in our case is 'fare_amount' and RMSE value will be the metric to judge the DNN performance during training. For this regression algorithm I utilized parameters different from the default parameters in order to optimize model fitting and the implementation and is discussed in the next section.

One of the general weakness of the DNN models is that, for more complex datasets (larger number of features and datapoints/rows) the DNN architecture needs to be complex also (many hidden layers), and thus adds many more additional weights/parameters that need to be optimized/trained for fitting purposes. Hence, the training time increases significantly and can be on the order of multiple weeks and even months depending upon the hardware resources (high end GPU's) one has access too. In order to somewhat tackle this issue, I would like to note that I rented out cloud GPU machines from paperspace.com in order to run DNN regression training faster as running these calculations locally on my laptop was very time consuming due to unsuitable hardware specs.

## Benchmark

The benchmark I utilized for comparison is the simple linear model provided in the description of this Kaggle competition(https://www.kaggle.com/dster/nyc-taxi-fare-starter-kernel-simple-linear-model). This benchmark model is a simple weighted linear combination of the features, with weights calculated using ordinary least square method. It does not capture the non-linearities that might be present in the training dataset. The RMSE value utilizing this model was $5.74, hence my objective was to lower the RMSE value via utilizing DNN regression. Furthermore, I also submitted my predictions to the Kaggle competition for DNN regression models and the RMSE scores obtained are presented via snapshots in the later sections.

## III. Methodology

### Data Preprocessing

Based on the above section where some anomalies and problems were detected in the training dataset. Following preprocessing steps were implemented in the 'NY_taxi_feature_Cleaning_Engineering_Scaling' notebook.

Feature Cleaning:

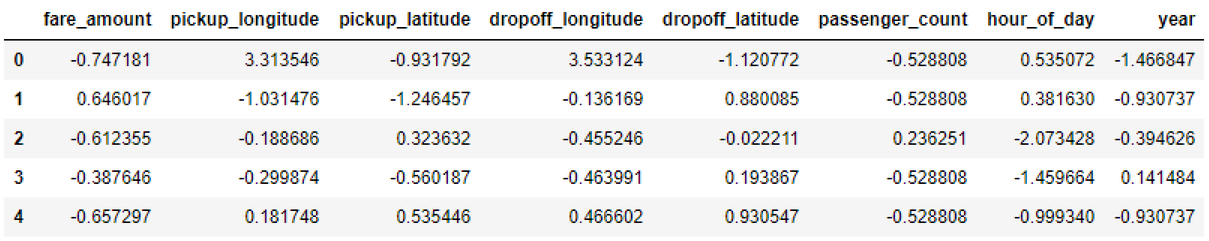
- First all the null data points were removed from the training set.
- Then all the data points with negative value for fare_amount feature were also excluded from the training set.
- In order to truncate the latitude and longitude feature values, I decided to exclude all the data points with latitude and longitude values that are +/- 1.4 degree of the latitude/longitude value of the center of New York city which was chosen to be 40.7128/-74.0060. The reasoning behind this is that ~ 1 degree change in lat/long values can results in 69 mile difference and I assume all the coordinates outside +/- 1.4 degree of the lat/long values of the NY city to be rare events or outliers and shouldn't be contributing to the training set data as most taxi rides would be within or near the NY city.
    - The removal of data from the training set based on this reasoning was applied to both pickup and dropoff latitude/longitude features.
- A regular taxi could fit at best 6 passengers (for e.g. some SUV's) hence all the data points with more than 6 value in the passenger_count feature was removed.
- Similarly, there needs to be at least 1 person in the taxi, for the fare to be charged, hence removed all the data points corresponding to less than 1 passenger from the training set.
- There were values in the fare_amount feature that were too high (for e.g. 93963), hence I decided to remove all the data points from the training set that were more than 60. I also showed in the "NY_taxi_feature_Cleaning_Engineering_Scaling" notebook that, removal of these points still leads to retention of ~99.7% of the training set data points, thus indicating that these high fare_amount feature values were indeed outliers and should not be contributing to training the regression model.

Feature Engineering:

- As noted earlier in the report, the time and date could influence the fare of the taxi, hence I implemented new features named as "hour_of_day" and "year" by extracting the *hour of the day* and *year* from the pickup_datetime timestamp in both training and testing dataframes. This was done with the assumption that month of the year would not affect the fare in any significant way, whereas time or hour of the pickup (higher fare at night) and year (inflation in fare) would be the only feature affecting the fare in meaningful way.

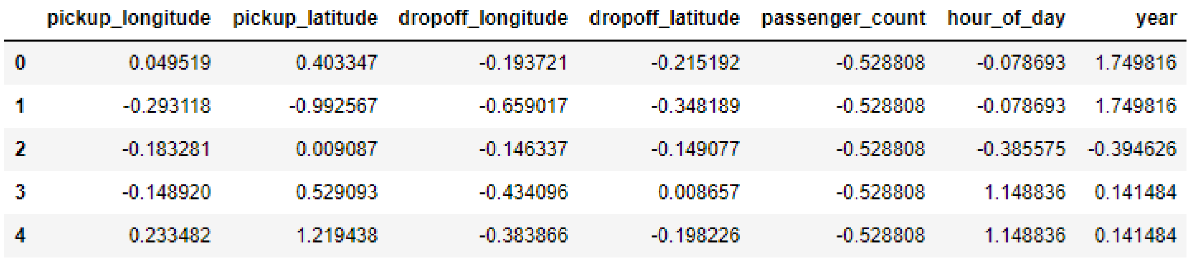
    - Concomitantly, the pickup_datetime column was no longer needed and hence I removed this feature from the training and testing dataframes after adding the above-mentioned features for hour and year.

- Lastly, I implemented Standard Scaler via sklearn library to scale the features of the training set so that the features weights are not skewed due to the range of the values of the features (feature with large values being weighted more in the regression model). This transforms the data into distribution with 0 mean and standard deviation of 1.
    - Scaling applied to both training set and testing set.
    - Scaler object was saved to inverse-transform later the predicted scaled fare_amount values that we'll get from the regression model, to finally obtain the actual fare_amount values that will be submitted to the Kaggle competition.
    - Scaled training and testing dataframes were saved in the .feather format to be used later for regression model.
- Head of the **scaled training dataframe**



| | fare_amount | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passenger_count | hour_of_day | year |
|---|---|---|---|---|---|---|---|---|
| **0** | -0.747181 | 3.313546 | -0.931792 | 3.533124 | -1.120772 | -0.528808 | 0.535072 | -1.466847 |
| **1** | 0.646017 | -1.031476 | -1.246457 | -0.136169 | 0.880085 | -0.528808 | 0.381630 | -0.930737 |
| **2** | -0.612355 | -0.188686 | 0.323632 | -0.455246 | -0.022211 | 0.236251 | -2.073428 | -0.394626 |
| **3** | -0.387646 | -0.299874 | -0.560187 | -0.463991 | 0.193867 | -0.528808 | -1.459664 | 0.141484 |
| **4** | -0.657297 | 0.181748 | 0.535446 | 0.466602 | 0.930547 | -0.528808 | -0.999340 | -0.930737 |

- Head of the **scaled testing dataframe**

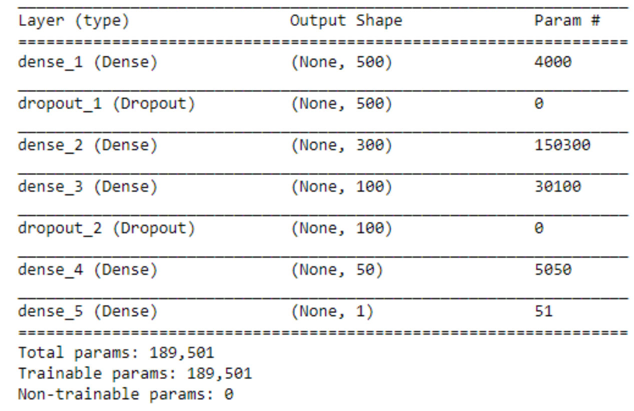


| | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passenger_count | hour_of_day | year |
|---|---|---|---|---|---|---|---|
| **0** | 0.049519 | 0.403347 | -0.193721 | -0.215192 | -0.528808 | -0.078693 | 1.749816 |
| **1** | -0.293118 | -0.992567 | -0.659017 | -0.348189 | -0.528808 | -0.078693 | 1.749816 |
| **2** | -0.183281 | 0.009087 | -0.146337 | -0.149077 | -0.528808 | -0.385575 | -0.394626 |
| **3** | -0.148920 | 0.529093 | -0.434096 | 0.008657 | -0.528808 | 1.148836 | 0.141484 |
| **4** | 0.233482 | 1.219438 | -0.383866 | -0.198226 | -0.528808 | 1.148836 | 0.141484 |

## Implementation and Refinement

<u>Deep Neural Network (DNN) regression:</u>

- The implementation is given in the ‘Deep_Neural_Network_Regression’ notebook.
- Before, I discuss the implementation, I would like to mention that if you go over the discussions and kernels for this Kaggle competition, very few people attempted the Neural Network implementation. Hence, I decided to tackle this problem via DNN regression which though resource intensive, does provide a unique way to look at this problem.
- I utilized Keras with Tensorflow backend to develop the Deep Neural Network model.
- The training set now consisted of 7 features and 1 target feature (fare_amount).
- Training set was split into testing and validation set via sklearn train_test_split() method.
- Loss function utilized was the ‘mean_squared_error’. Taking square root of this quantity gives the RMSE (See the results section).
- DNN Model checkpoints were also defined to save the best model weights.
- At first, I tried to utilize all the 55 million rows of training data points which I why I rented the P5000 GPU machine on paperspace.com. However, after implementing even the most basic Neural Network (NN) architecture (for e.g. 16,8,1), I realized it was taking very long time to even run 1 epoch of training and additionally, I was largely underfitting the data with predicted value on the validation set fluctuating around the constant value of ~-0.08. Even with small learning rate of ~0.0001, I was getting large underfitting, which made me realize that I needed more deep layers in the NN architecture. However, this introduced more weights that needs to be optimized, hence lead to longer training time. After trial and error procedure of changing number of hidden layers and number of neurons in each hidden layer, I settled with utilizing 1 million data points from the training set with 3 hidden layers in the DNN architecture, and 500 neurons contained in the first layer. Below is the final DNN architecture utilized,



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 500)               4000      
_________________________________________________________________
dropout_1 (Dropout)          (None, 500)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 300)               150300    
_________________________________________________________________
dense_3 (Dense)              (None, 100)               30100     
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 50)                5050      
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 51        
=================================================================
Total params: 189,501
Trainable params: 189,501
Non-trainable params: 0
```

- I additionally implemented 3 drop out layers with only 0.1 probability of dropping out the output of the layer. This is because underfitting was the main issue I was running into, but at the same time I didn’t want to fit to the noise in the data so that the final model can be more generalized.
- **ReLu** activation function was utilized on the output of each layer in the DNN, to add non-linearity to the model.
- Adam optimization of the DNN parameters was utilized and the Learning rate was changed from default 0.001 to 0.0001 to “smoothly” reach the loss function minima, instead of fluctuating around it. This also helped in improving the underfitting issue.
- I initially ran the training for 50 epochs, which however took 5 hours to train but no significant improvement in the validation loss function was seen after 10-15 epochs and the best validation loss in terms ‘mean_squared_error’ was ~0.13 or in terms of RMSE ~0.36. Hence, I changed the value of epochs to 10 for the final run.
- Additionally, I choose smaller batch size = 10, because then weights/parameters were updated after each propagation and hence lead to faster convergence towards lower validation loss values as compared to updating parameters after propagating the entire training data set through the DNN model.

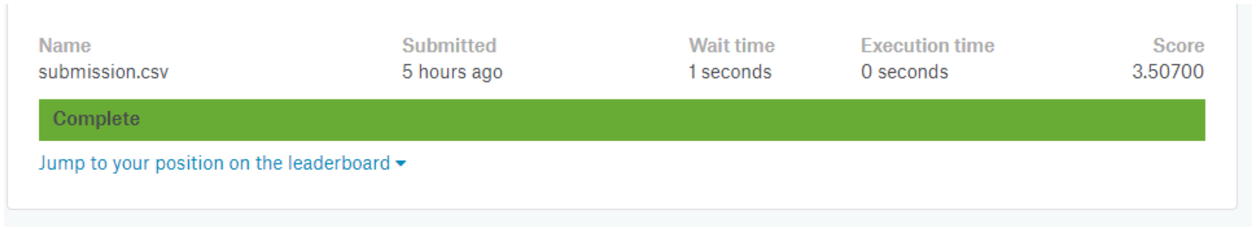
## IV. Results

### Model Evaluation and Validation

The derivation of the DNN model is described in detail in the previous section. For model evaluation, first RMSE was calculated on the validation set, i.e. average error between the predicted ‘fare_amount’ value and the actual ‘fare_amount’ values. This is also provided in the ‘Deep_Neural_Network_Regression’ notebook. The final RMSE value came out to be ~0.36 (note that the inputs were scaled data). The comparison between the predicted ‘fare_amount’ and the real ‘fare_amount’ is presented in the ‘Free-Form Visualization below’.

Apart from this, testing data provided by the Kaggle competition was then utilized to obtain predictions in scaled ‘fare_amount’ values which were then inverse-transformed into the real $ predictions. The result was saved in submission.csv and this submission file was then upload to the Kaggle competition which gave a public score in terms of RMSE value on the testing set. Below are some of the snapshots from submission to this competition.

The final score in terms of RMSE was 3.507. Implications are discussed in justifications.

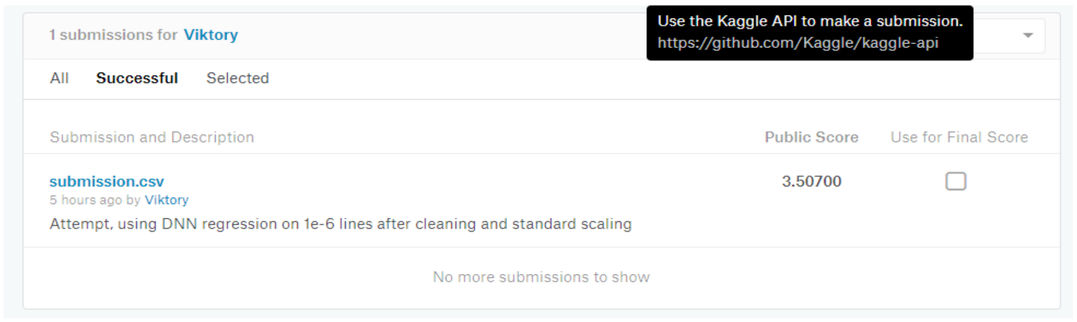


| Name | Submitted | Wait time | Execution time | Score |
|---|---|---|---|---|
| submission.csv | 5 hours ago | 1 seconds | 0 seconds | 3.50700 |

Complete

Jump to your position on the leaderboard

Typo in the image below, it should say 1e+6 lines (1 million rows of data)



1 submissions for Viktory

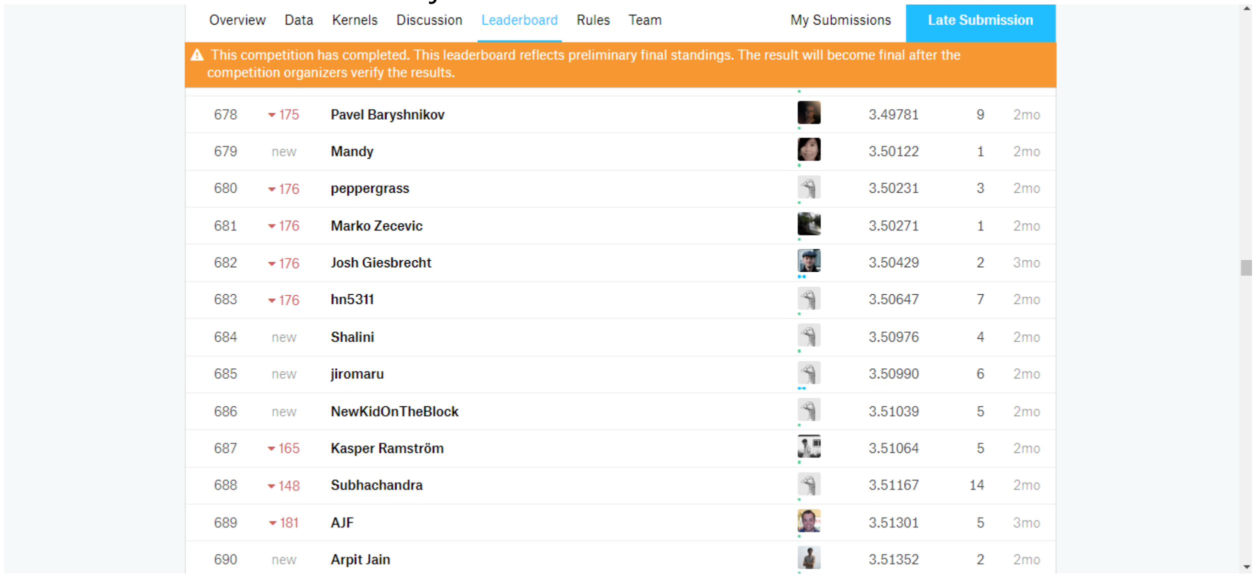
Use the Kaggle API to make a submission.
https://github.com/Kaggle/kaggle-api

All Successful Selected

| Submission and Description | Public Score | Use for Final Score |
|---|---|---|
| submission.csv<br>5 hours ago by Viktory<br>Attempt, using DNN regression on 1e-6 lines after cleaning and standard scaling | 3.50700 | ☐ |

No more submissions to show

Only late submission was allowed for this competition currently, hence the tentative rank on the leaderboards would be ~ 678 out of 1438 ranks. I also discuss improvement in the leaderboard rank in the justification below.



This competition has completed. This leaderboard reflects preliminary final standings. The result will become final after the competition organizers verify the results.

| | | | | | |
|---|---|---|---|---|---|
| 678 | ▾175 | Pavel Baryshnikov | 3.49781 | 9 | 2mo |
| 679 | new | Mandy | 3.50122 | 1 | 2mo |
| 680 | ▾176 | peppergrass | 3.50231 | 3 | 2mo |
| 681 | ▾176 | Marko Zecevic | 3.50271 | 1 | 2mo |
| 682 | ▾176 | Josh Giesbrecht | 3.50429 | 2 | 3mo |
| 683 | ▾176 | hn5311 | 3.50647 | 7 | 2mo |
| 684 | new | Shalini | 3.50976 | 4 | 2mo |
| 685 | new | jiromaru | 3.50990 | 6 | 2mo |
| 686 | new | NewKidOnTheBlock | 3.51039 | 5 | 2mo |
| 687 | ▾165 | Kasper Ramström | 3.51064 | 5 | 2mo |
| 688 | ▾148 | Subhachandra | 3.51167 | 14 | 2mo |
| 689 | ▾181 | AJF | 3.51301 | 5 | 3mo |
| 690 | new | Arpit Jain | 3.51352 | 2 | 2mo |

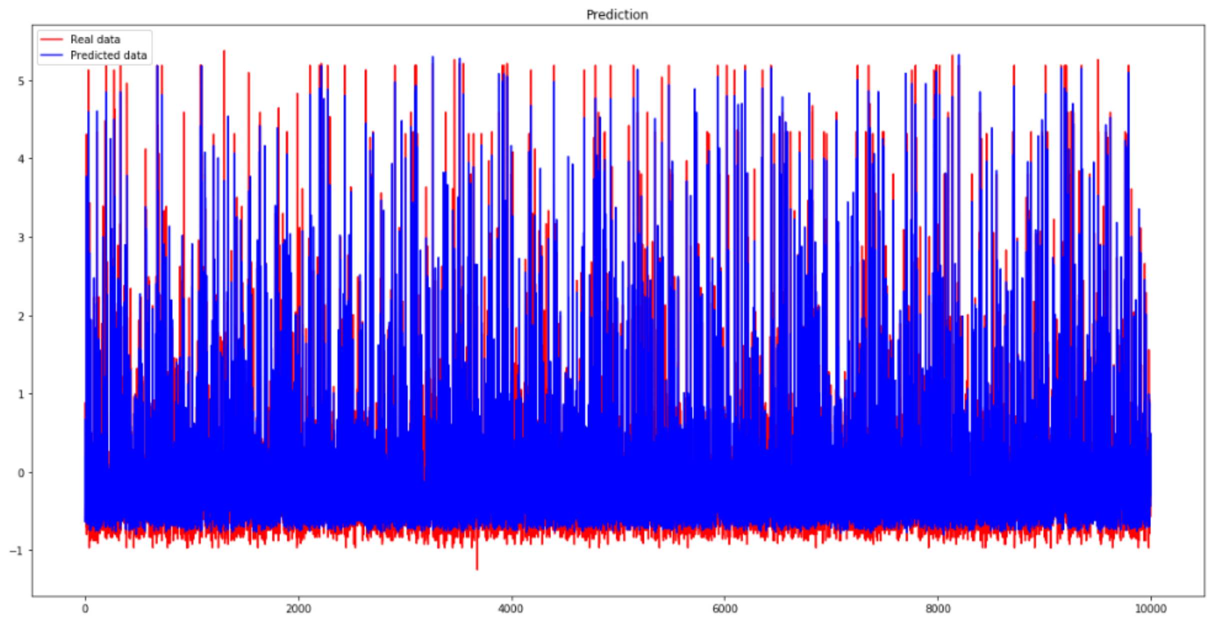
## Justification

The RMSE for the benchmark model was ~6 as compared to RMSE on the competition testing set using DNN model, which is ~3.5. Hence there is significant improvement in the RMSE than using the benchmark model, thus I believe the final solution is significant enough to have solved the problem with room for further improvement (discussed in next section)

It must be noted here that this significant improvement in the RMSE value was obtained from DNN model via utilizing just 1 million rows, i.e. 1/55 of the complete training dataset. DNN regression was able to introduce non-linearities into fitting of the model and thus were able to capture non-linear relationships among the feature set. Thus, utilizing larger training set could improve the model prediction and could shift the rank to higher position on the leaderboards.

## V. Conclusion

### Free-Form Visualization



This plot is also presented in the ‘Deep_Neural_Network_Regression’ notebook. What this plot shows is the final DNN model predictions (blue lines) for the scaled ‘fare_amount’ along with the real scaled ‘fare_amount’ values (red line) on the validation dataset. This plot is showing first 10000 values for the ‘fare_amount’ feature out of the 200,000 total data points in the validation set.

The plot depicts that the model can predict the target feature value with good accuracy. The model is however underestimating large ‘peaks’ or high ‘fare_amount’ values specially values that are 5+, however, it should also be noted that the density of these 5+ peaks as compared to density of other peaks (for e.g. peaks with values <2) is quite low, and thus can be attributed as ‘noise’ in current setting of training set size (1 million rows). Hence, driving the model towards accurate prediction of these 5+ peaks would amount to overfitting and a less generalized model. In order to drive the model towards prediction of these points and while also maintaining generalizability would require more training data to be utilized.

## Reflection

The project involved utilizing the large training set data to build a regression model that can predict the target feature values and all the relevant training data and corresponding target values along with testing dataset were provided by Kaggle competition.

The most interesting or rather annoying thing I found was that your need very good computer hardware specs to handle this size of data (all 55 million rows of training data) which unfortunately a lot of currently available laptops and PC’s do not have.

For example, I wanted to further engineer the feature data to reduce the number of inputs to the DNN architecture, such that lat/long values of the pickup and dropoff would be utilized to obtain actual distance or distance from a fixed point in NY city (such as Central Park). For this I tried to utilize python’s GeoPy library and it would still take me ~ 7-10 mins to even process 100,000 lines on my laptop, thus multiple hours to process the entire training dataset. Though I switched to rented cloud machines, but still it was not good enough. Due to this I had to resort to some simple feature engineering and cleaning. Similar problem again came for the running the DNN model, and thus finally had to resort to using less training data.

I believe utilizing all the 55 million data points along with multi-layer DNN model is possible with powerful computer hardware, along with training for a week to get a good model fit. This type of predictive power utilizing ‘Big Data’ is immensely useful however, seems to be out of reach for average person currently, and thus skews the distribution of this predictive power to just academia or large corporations.

Overall, I do believe the current methodology of DNN regression, even the current DNN architecture with slight tweaks via adding more hidden layers for larger training dataset, is suitable for modeling this competition’s large dataset, but requires powerful computational resources.

**Improvement**

- First thing that should improve DNN model predictions is the inclusion of the more training data, maybe all, if possible.
- Secondly, I believe autoencoders could also be utilized on the training dataset for dimensionality reduction, for example, reduction in the dimension of feature space from 7 to 3.
- Including more hidden layers in the DNN model if utilizing larger training dataset along with running simulation for longer epoch in hopes of finding 'better' local minima for the loss function while also incorporating higher probability of dropout in the dropout layer to prevent overfitting could also improve model predictions and hence lower RMSE values.

I believe looking at the kernels and discussion of this competition, XGboost which is a gradient boosting library was highly popular, resulting in good RMSE scores, however, I wanted to the explore the DNN modeling on this competition dataset and hence went the current DNN model route. Hence, DNN model could serve as a benchmark for XGBoost regression and comparison can be made between the prediction performance of these models with training on 1e+6 data points.