Predicting New York City Taxi Fares

COMP9061 – Practical Machine Learning – Assignment 2
Mike Leske – R00183658

Abstract In this research project several algorithms suitable for regression problems have been used to predict the NYC taxi fares from GPS latitude and longitude and date/time information. Linear regression models quickly completed training on 10 million observations. XGBoost produced the best performing model at the cost of approximately 30 minutes training time. Essential steps in preparing the dataset included the addition of a distance-to-airport feature, adding time-related features, scaling the latitude and longitude information and removing erroneous observations from the data set.

1 Introduction

New York City is world-famous for its amount of yellow cabs. With the rise of market disrupters like Uber and Lyft it became essential to upfront be aware of the expected ride cost to select the 'best' option in terms of monetary benefit for the user. In August/September 2018 Kaggle hosted a competition together with its partners Google and Coursera aiming to predict regular taxi fares in New York City. [1]

As part of the competition a training dataset of 55 million historic taxi rides (2009 - 2015) has been provided.

The problem statement of this Kaggle competition is defined as "predicting the fare amount (inclusive of tolls) for a taxi ride in New York City given the pickup and dropoff locations". [1] The predicted fare amount is to be provided in US Dollar and can serve as an estimate for a future taxi rides. A potential use case for such a prediction algorithm is the integration into online navigation services which – in addition to the routing information – intend to provide an estimate of taxi fare costs to a given route inside New York City.

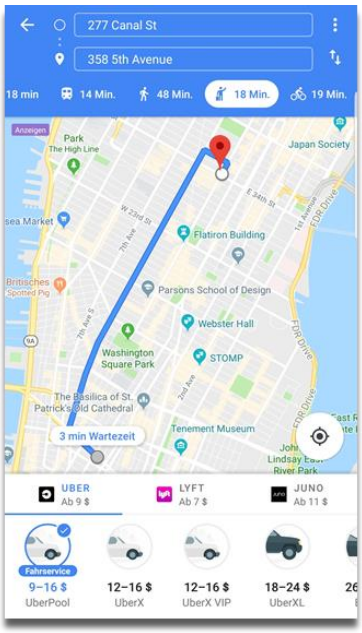

Figure 1 Google Maps providing Uber fare estimation

The outlined problem fits well into the domain of supervised regression problems for which a multitude of potential solutions exist including classical (multivariate) linear regression, decision trees, artificial neural networks and many more.

As part of this research project a set of algorithms will be implemented and compared against each other in respect of accuracy (best result) and efficiency (fastest result).

2 Research

The nature of the chosen dataset required research to be taken into multiple directions to improve the overall predition performance:

- Gradient Boosting (XGBoost) [2] has significantly improved the prediction quality compared to the algorithms provided as part of the scikit-learn library
- The interpretability of the taxi ride pickup and dropoff latitude/longitude was essential
- RandomizedSearchCV [3] was used as the sheer size of the dataset would have resulted in a prohibitively long GridSearchCV run.

2.1 Beyond the Random Forest: Gradient Boosting

Gradient Boosting builds on the foundations of boosting decision trees for classification problems, in which each tree is considered a weak learner, which does not aim modelling all dependencies of the provided dataset. Rather, subsequent weak learners focus on developing new decision trees to handle observations that were incorrectly handled by the previous weak learners. The ensemble of the weak learners can then represent a very strong learner.

The Gradient Boosting algorithm represents a generalization of classical boosting algorithms, e.g. AdaBoost, where the objective becomes the minimization of a loss function by creating weak learners sequentially using a gradient descent like procedure. This generalization allows gradient boosting algorithms to use arbitrary differentiable loss functions and therefore making the algorithm suitable also to regression and multiclass classification problems. [4]

Basic Gradient Boosting operations are defined by the loss function to be optimized, weak learner and the additive nature of the model, where sequentially new weak learners are added while previous weak learners remain unchanged.

Improvements to the basic Gradient Boosting operation include:

- Tree Constraints
  - Number of decision trees
  - Depth of the decision trees
  - Number of nodes
  - Number of leaves
  - Number of observations per split
  - Minimum improvement to loss
- Shrinkage
  - Also known as “Learning Rate”
- Random Sampling
  - Also known as “Stochastic Gradient Boosting” where subsequent trees are created based on randomly sampled observations.
  - Stochastic Sampling can be based on observations and dataset columns to be used for a tree creation.
- Penalized Learning
  - L1 and L2 regularization of leaf values in decision trees and help avoiding overfitting.

XGBoost stands for eXtreme Gradient Boosting and is a well-known Gradient Boosting algorithm that recently dominated machine learning and Kaggle competitions especially for structured/tabular data where it has outperformed many classical machine learning algorithms. It implements all the improvements

to Basic Gradient Boosting listed above and is optimized to run in a performant manner, e.g. by leveraging parallel CPU cores and cache optimization. [4]

The default options for optimizing the XGBoost operations are specified in the official documentation for the XGBoost project [5].

2.2 Encoding Latitude and Longitude Information

In order to make good predictions on taxi ride fares it is important that the essential features (pickup/dropoff latitude/longitude) are represented in a format that allows machine learning algorithms an efficient computation.

Therefore, the latitude and longitude features have been transformed with different feature scalers (MinMaxScale, StandardScaler, RobustScaler, Normalizer, QuantileTransformer). It turned out that different machine learning algorithms preferred different scaler functions.

Also, a Principle Component Analysis (PCA) transformation of the original latitude and longitude parameters was evaluated.

In addition, a reverse address lookup could theoretically improve a model's predictive power by enriching the dataset with street, suburb and zip code information. However, achieving a reverse address lookup for 110 million latitude/longitude pairs was out of reach for this project.

2.3 RandomizedSearchCV

Even with a largely reduced dataset of only 1 million observations a classical GridSearchCV for hyper-parameter optimization would have taken a too long amount of time. RandomizedSearchCV from the scikit-learn library came to rescue, which significantly reduced the search runtime to a few minutes by performing a search across a sampled number of parameter settings.

This tradeoff for runtime over accuracy seems acceptable as an overly hand-optimized set of hyper-parameters risks to work well on the very dataset available, but the resulting model may not generalize well to unseen data.

3 Methodology

In preparation for running machine learning algorithms against the chosen dataset a number of preprocessing steps were implemented. The following sections provide details on those steps.

3.1 Dataset

The dataset [6] to be used for this competition is provided by Kaggle and its partners Google and Coursera. It contains two essential files:

- A train.csv file with a training set of 55 million taxi rides and
- A test.csv file with a test set of 9914 taxis rides.

For each observation (individual taxi rides) the following parameters are provided:

- key
- pickup_datetime
- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude
- passenger_count

While the training set (train.csv) contains the final taxi ride fare, the test set used for participation in the Kaggle competition does not provide this insight.

To frame memory and computational requirements a 1 million observations training dataset and a 10 million observations training dataset have been extracted from the original competition training dataset. The original dataset is well randomized and there is no temporal or geospatial dependency between the different observations. While the 10 million observations set has been used for the algorithm training, the 1 million observation set was used to speed up the RandomizedSearchCV hyper-parameter optimization.

Because the official test set comes without target values (“fare_amount”) it cannot be used to measure progress during the training. Therefore, a second test set, from now on called ‘evaluation set’, has been carved out of the 10 million observations training set. This evaluation test set was used to measure training progress. While first the training progress was measured against a 0.001% sized evaluation set – to match the size of the official competition test set (9914 observations) – tests revealed a training RMSE closer to the competition test result when working with a typically used 10% size evaluation set.

3.2 Data Preprocessing

The chosen dataset provides an enormous number of 55 million observations. Therefore, less focus was put on correcting outliers or imputing missing information (such observations were removed).

Instead, more times was spent on

- Feature Engineering to create more meaningful features from the raw data
- Data Cleaning to remove unrealistic and wrong/unusable observations
    - passenger_count is zero
    - wrong or missing latitude/longitude coordinates
    - negative fare_amount
    - extremely short rides of just a few meters (sometimes together with a fare of several hundreds of dollars)

3.2.1 Feature Engineering

The following features have been engineered for the training and test datasets:

Table 1 Feature Engineering

Name	Comment
"year", "weekday", "hour"	Learn time-based dependencies in the training taxi ride fares. These features were created from the "pickup_datetime" provided with the original dataset.
"distance_miles"	Calculate the taxi ride distance based on the Haversine formula [7] based on the provided GPS coordinates for pickup and dropoff.
"jfk_dist", "ewr_dist"	Calculate the shortest distance of either pickup or dropoff GPS coordinates to 2 NYC airports (JFK, EWR). These features are expected to help in modelling the special airport fares (flat rates).
"pickup_pca0", "pickup_pca1", "dropoff_pca0", "dropoff_pca1",	Transform locality information and thereby enrich the dataset.

In addition the latitude/longitude columns have been scaled. Section 3.3 lists the final experiments executed.

All feature engineering steps are implemented in `utils.py`.

3.2.2 Data Cleaning

The data cleaning targeted removing either outlier observations or observations with wrong information. Observations with missing variables also have been removed. The algorithms were trained using the 10 million observation training set, and the number of removed rows due to missing information was negligibly low.

The drastic action of dropping incorrect data rows is based on two observations:

1. The overall data set is so large with its 55 million rows that the loss of individual data observations does not affect the predictive power of the algorithms.
2. Wrong or missing longitude/latitude and passenger information cannot be imputed from the remaining data.

The following data cleaning was applied to the training dataset:

Table 2 Data Cleaning

Name	Comment
drop	Drop observation due to incomplete information.
"pickup_longitude", "dropoff_longitude"	Limit longitudes to the interval [-75, -73]. This effectively excludes extreme outliers and wrong information.
"pickup_latitude", "dropoff_latitude"	Limit latitudes to the interval [40, 42]. This effectively excludes extreme outliers and wrong information.
"passenger_count"	Limit passenger_count to the interval [0, 6] to match test dataset.
"fare_amount"	Limit fare_amount to the interval [2.5, 250]. This effectively excludes extreme outliers and wrong information.
"distance_miles"	Limit distance_miles to the interval [0.05, 100]. This effectively excludes extreme outliers and wrong information. It also removes observations with zero mobility between pickup and dropoff, which might indicate wrong data recording.

The data cleaning steps are implemented in `utils.py`. On average this action removed ~3.7% rows.

3.3 Modelling

The New York City Taxi Fare Prediction Kaggle competition presents itself as a classical regression type of problem, which is a well-studied domain with several potential solutions available ranging from classical Machine Learning (ML) algorithms to more sophisticated Gradient Boosting operations and Deep Learning (DL) architectures based on Artificial Neural Networks (ANN).

As described in section 1, the provided training data is labelled with typical taxi ride variables like pickup and dropoff coordinates and the number of passengers as well as the "label" in terms of cost of the ride ('fare_amount').

Therefore, the solution to this prediction problem can be targeted using Supervised Learning algorithms.

The following algorithms will be evaluated and compared:

- Multivariate **Linear Regression**
- **Support Vector Regression (LinearSVR)**
- **Decision Tree**
- **Other regressor (e.g. Ridge, LassoLars, ElasticNet, HuberRegressor)**
- **Random Forest**
- **Gradient Boosting (XGB)**

With the exception of XGBoost, which ships as a separate Python package, all other algorithms were based on the Python scikit-learn library.

Each of the listed regression algorithms was trained against the same 10 million observations training dataset first enhanced with basic feature engineering. Against Linear Regression, Decision Tree, Random Forest and XGBoost different Feature Scaling algorithms were evaluated. This decision was mainly based on runtime (Linear Regression and Decision Tree) and overall quality of the model (Random Forest and XGBoost).

For each training run, RMSE and R^2 metrics have been collected by evaluating the model prediction quality against the evaluation test set.

For the best unoptimized learner with default parameters (XGBoost) a hyper-parameter optimization using scikit-learn RandomizedSearchCV was conducted using the 1 million observations training set. Next, XGBoost was again trained against the 10 million observations training set with updated haper-parameters.

Additionally, for each experiment predictions have been created for the official competition test set and results were obtained by uploading the predictions to Kaggle.

3.3.1 Metrics

As the problem statement points into the direction of a regression analysis, the competition evaluation is based on the root mean-squared error (RMSE) [8][9]. RMSE is well-suited for this problem statement as it measures the difference between the predicted taxi fares and the real taxi ride fares provided as part of the training dataset. In this sense the taxi ride fare ('`fare_amount`') is the regression's dependent variable.

RMSE is calculated as:

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

A smaller RMSE means the predicted values are closer to the real fares of the training set than the predictions of a model with a larger RMSE.

Another frequently used metric for regression problems is the coefficient of determination, also known as R-Squared – or short R^2. R^2 represents the fraction in which "the variance of errors is less than the variance of the dependent variable". [10][11]

R^2 is calculated as:

$$R^2 = 1 - \frac{SSres}{SStot}$$

In general language R^2 provides a metric on how much better the regression predictions are over always predicting the mean. R^2 ranges from -∞ to 1, where:

- $R^2 < 0$, predictions worse than always predicting the mean
- $R^2 = 0$, predictions not better than always predicting the mean
- $R^2 > 0$, predictions better than always predicting the mean

With R^2 as evaluation metric we aim for values close to 1.

3.3.1 Hyper-Parameter Optimization for XGBoost

Refining a Gradient Boosting algorithm essentially means tuning its multiple parameters on how additional trees are added to the overall model. Section 2.1 already introduced the different types of Gradient Boosting optimization variables. Details for any of these algorithm parameters are explained in the XGB documentation [5].

The python scikit-learn library RandomizedSearchCV was used to iterate through a sampled set of potentially optimizing parameters.

RandomizedSearchCV provided tremendous benefits in search runtime reduction compared to the classical GridSearchCV, at the cost of uncertainty whether the absolute optimum has been squeezed out of the model.

"Negative Mean Squared Error" was used as RandomizedSearchCV scoring function, which provides a compatible metric to the actual problem statements' metric (Root Mean Squared Error).

Table 3 shows the RandomizedSearch search space and the chosen optimized values :

Table 3 XGBoost Hyper-Parameter Optimization Search Space

Parameter	Search Space	Best
'subsample'	[0.7, 0.8, 0.9]	0.9
'max_depth'	[3, 5, 7, 9]	9
'learning_rate'	[0.05, 0.06, 0.075, 0.08, 0.1, 0.2, 0.3]	0.1
'colsample_bytree'	[0.4, 0.6, 0.8]	0.8
'colsample_bylevel'	[0.4, 0.6, 0.8]	0.6
'min_child_weight'	[1, 2, 4, 6, 8]	1

3.3.2 Experiments Executed

Table 4 lists the experiments executed and documented as part of this research project :

Table 4 Experiments Executed

#	Model	Scaling	HyperOpt	Metrics
1.1	Linear Regression	No	No	RMSE, R^2, Runtime
1.2	Linear Regression	QuantileTransformer + PCA (for lat/lon)	No	RMSE, R^2, Runtime
1.3	LinearSVR	No	No	RMSE, R^2, Runtime
1.4	Decision Trees	No	No	RMSE, R^2, Runtime
1.5	Decision Trees	MinMaxScaler	No	RMSE, R^2, Runtime
1.6	Other Lin Regressors	No/Yes	No	RMSE, R^2, Runtime
2.1	Random Forest	No	No	RMSE, R^2, Runtime
2.2	Random Forest	Yes	No	RMSE, R^2, Runtime
3.1	XGBoost	No	No	RMSE, R^2, Runtime
3.2	XGBoost	MinMaxScaler	No	RMSE, R^2, Runtime
3.3	XGBoost	MinMaxScaler + PCA (for lat/lon)	Yes	RMSE, R^2, Runtime
3.4	XGBoost (without passenger_count)	MinMaxScaler + PCA (for lat/lon)	Yes	6MSE, R^2, Runtime

Each experiment is documented in a separate Jupyter notebook.

All experiments were executed on the same computer with the following core specifications :

- Intel(R) Core™ i7-9750H CPU @ 2.60Ghz (Coffee Lake)
- 16 GB 2400 Mhz DDR4
- 500 GB SSD

4 Evaluation

The different models have been implemented, trained against the training dataset and finally predictions for the competitions dataset were inferred from the implementations' model.

For each algorithm several parameters were captured to support comparing them :

- Run-Time (Time needed to complete training)
- Result of default algorithm execution for validation and test datasets
- Result of hyper-parameter optimized XGB algorithm execution for validation and test datasets

Table 5 Experiment Results: RMSE, R2, Runtime, Overfit

Experiment	Train Time hh :mm :ss	Evaluation Test Set RMSE / R^2	Kaggle Test Set RMSE	Overfit
Linear Regression				
1.1	00:00:05	4.54 / 0.76	5.14	0.60
1.2	00:00:22	4.48 / 0.77	4.82	0.34
LinearSVR				
1.3*	00:04:02	4.86 / 0.74	5.43	0.57
Decision Tree				
1.4	00:03:40	4.49 / 0.77	4.86	0.37
1.5	00:03:38	4.54 / 0.78	4.53	-0.01
Other Lin. Reg**				
1.6	n/a	n/a	n/a	n/a
Random Forest***				
2.1	00:27:04	3.15 / 0.89	3.44	0.29
2.2	00:27:02	3.17 / 0.89	3.45	0.28
XGB****				
3.1	00:27:15	3.21 / 0.88	3.24	0.03
3.2	00:36:15	3.25 / 0.88	3.21	-0.04
3.3	00:24:27	3.12 / 0.89	3.11	-0.01
3.4	00:30:12	3.09 / 0.89	4.30	1.21

* LinearSVR resulted in prohibitively long runtime, so it was trained against the 1 million observation training set only.

** Other Linear Regression models have been trained (Ridge, LassoLars, ElasticNet, HuberRegressor). [12] But predictions for the Kaggle test set remained around RMSE 5.14 and documentation was omitted.

*** Random Forest was configured to build 50 n_estimators.

**** XGB was configured to run max 150 num_boost_rounds and stop when no improvement was found for 10 rounds (early stopping).

The results provided in table 5 represent the RMSE and R^2 of the evaluation test set and the RMSE of the competition test dataset predictions which can only be retreived by uploading a submission file to Kaggle. In addition the difference between evaluation set and Kaggle test set was calculated and is shown in column Overfit.

All experiments were executed on the same PC to allow runtime comparison. Figure 2 visualizes for each experiment the RMSE and R2 metrics for the evaluation test set and the RMSE for the competition test set.

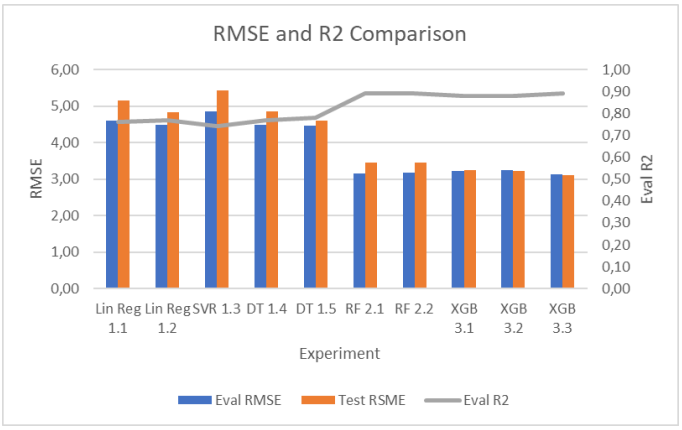

Figure 2 Comparison of RMSE and R2 metrics

While training the Linear Regression took the least time, the resulting test RMSE was far worse than those for Random Forest and XGBoost. Comparing Random Forest and XGBoost, the latter produced slightly better models based on RMSE in less time. The differences in R^2 metric against the evaluation test set are almost negligible and converge around 0.9.

The used metrics of RMSE and R^2 are highly correlated, i.e. better model predictions result in lower RMSE and higher R^2. Noticeably, for the given problem the RMSE metric provides a more granular view on model performance. For example, experiments 3.2 and 3.3 result in a very close R^2 metric of 0.88 and 0.89, whereas the actual RMSE improvement from experiment 3.2 to 3.3 is 4%.

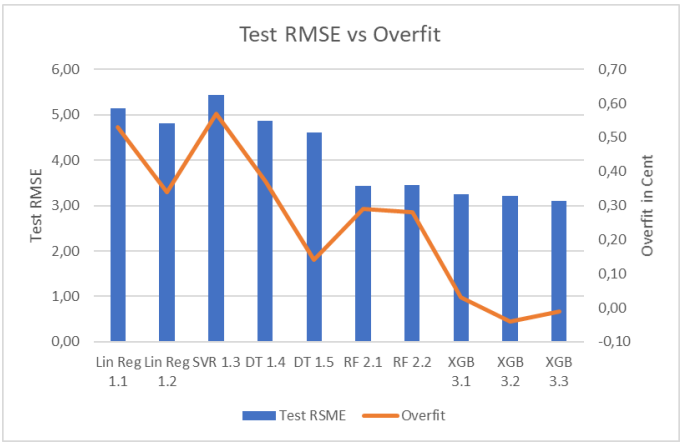

Figure 3 Comparison of Test RMSE vs Model Overfitting

Both, Figure 2 and Figure 3 highlight that experiments 2.1 – 3.3 resulted in the lowest RMSE, but only the XGBoost experiments minimized the overfitting between the training data and the test set.

Figure 4 presents the RMSE on the test set for a selected number of algorithms both before and after scaling latitude and longitude information. With the exception of the Random Forest all algorithms improved predictive power.

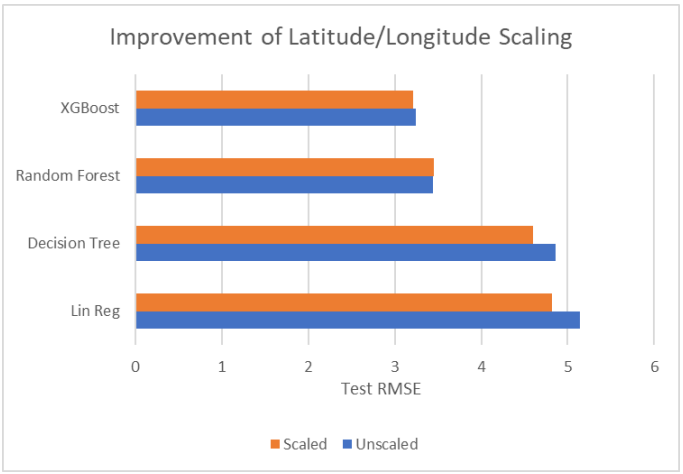

Figure 4 Improvement of Latitude/Longitude feature scaling

Finally, visualizing the taxi fare prediction of the evaluation test set against the target prices gives hints on areas to improve. Figure 5 scatters the predictions of the best experiment (XGBoost, 3.3) against the appropriate ground truth taxi fares.

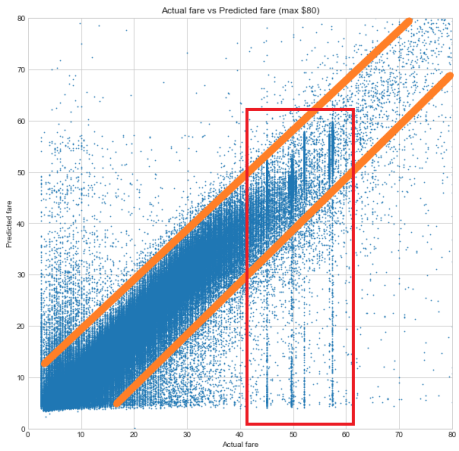

Figure 5 Visualizing Actual Fare vs Predicted Fare

A model with perfect predictions would result in a 45° line from 0,0 to 80,80. The orange corridor shows that the majority of all predictions are centered around such an optimal line. The narrower such corridor is, the better the model predicts the taxi fares. The red rectangle highlights vertical structures. These observations represent rides with special fares, e.g. airport rides.

A 2-to-2 dimensionality "reduction" – or better transformation – using PCA could not be successfully evaluated to improve interpretability of latitude and longitude information. While it consistently helped improving the predictive quality of the scikit-learn Linear Regression model (experiment 1.2), it mostly led tree-based learners (Decision Tree, Random Forest, XGBoost) to produce worse predictions.

Interestingly, the best performing model (XGB, experiment 3.3) had the PCA transformation turned on, but the overall model improvement between experiment 3.2 and 3.3 is attributed to the hyper-parameter optimization. Figure 6 displays the feature importance analysis of experiment 3.3 and confirms the criticality of latitude/longitude features and its distance and PCA derivative. The passenger_count has the least relevance. However, when removing the passenger_count in experiment 3.4 the resulting model drastically loses predictive power on the Kaggle test set.

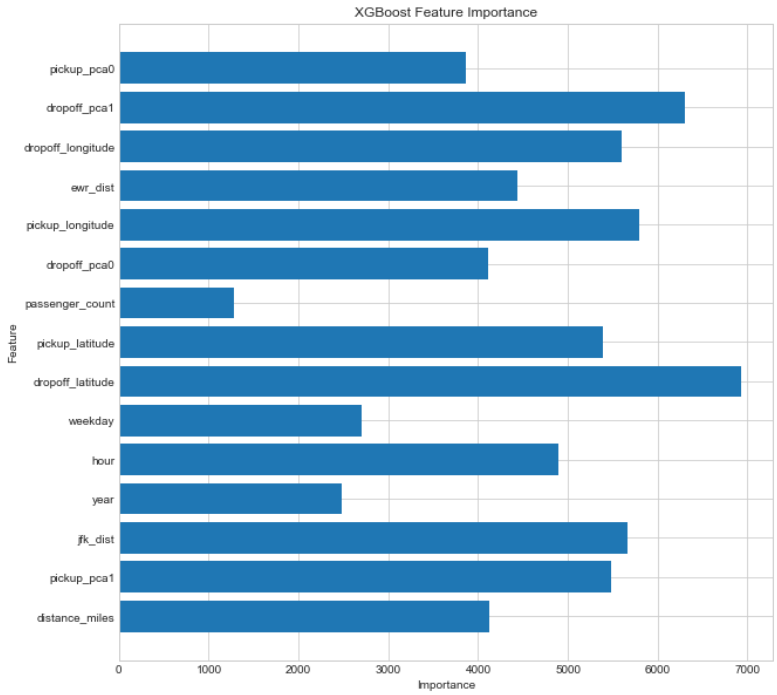

Figure 6 Feature Importance of experiment 3.3

5 Conclusion

In this research project several algorithms suitable for regression problems have been to used predict the NYC taxi fares. The algorithms ranged from simple Linear Regression models able to train on the dataset within seconds to Random Forests and Gradient Boosting trees, which provided the highest predictive power, but required far more computing resources (runtime) during training.

Different experiments have indicated that transforming and scaling the GPS latitude and longitude information helped the algorithms to generalize better and, hence, produce better predictions.

The search for optimized hyper-parameters only had a minor impact on the quality of the fare predictions. Considering the problem domain any customer-facing application would be expected to round up to the next full dollar and therefore prediction improvements in the range of a few cents have academic value only.

5.1 Future Work

Figure 5 highlighted that even the best model still had problems in correctly identifying "special fare" rides which is likely caused by a too strong emphasis on the taxi ride distance. Therefore, more time should be spent to improve locality encoding from latitude and longitude features.

Overall work should also be done to narrow down the orange prediction corridor from Figure 5.

References

[1] https://www.kaggle.com/c/new-york-city-taxi-fare-prediction

[2] https://xgboost.readthedocs.io

[3] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

[4] Book: Machine Learning Mastery - Discover XGBoost With Python!

[5] https://xgboost.readthedocs.io/en/latest/parameter.html

[6] https://www.kaggle.com/c/new-york-city-taxi-fare-prediction/data

[7] https://en.wikipedia.org/wiki/Haversine_formula

[8] https://www.kaggle.com/c/new-york-city-taxi-fare-prediction#evaluation

[9] https://en.wikipedia.org/wiki/Root-mean-square_deviation

[10] https://en.wikipedia.org/wiki/Coefficient_of_determination

[11] https://people.duke.edu/~rnau/rsquared.htm

[12] http://scikit-learn.org/stable/modules/linear_model.html