Student ID Code: u1637462 21/12/2018

NYC Taxi Fare Prediction

The data set for the NYC Taxi Fare competition on Kaggle was explored and modelled in order to build an algorithm which best predicts the values of taxi fares in NYC. Features were extracted from the initial attributes of pickup, drop-off locations and timestamp for the journey. As expected the most useful feature for predicting was found to be the haversine distance between locations. The most successful model built was an extreme gradient boosting classifier which scored an RMSE of 3.051 on the test data. The code developed for the challenge can be found here.

1. Introduction

Predicting the taxi fare in advanced has become a vital task since the arrival of cab-sharing companies like Uber and Viavan. These companies use complex algorithms to precompute the cost of taxi rides acting as an additional service to the user, before requesting such a service [1]. More accurate results for these predictions are in the best interest of both the consumer and the companies providing these services. This is perhaps the most important reason for Kaggle's 'New York City Taxi Fare Prediction' playground challenge. This competition aims to predict the taxi fare of rides provided with the raw features of the passenger count, timestamp, pickup and drop-off locations for the ride. The dataset provided contains both a test and training file suggesting a supervised learning approach to train a regressor. The challenge is scored using a root mean squared error (RMSE) metric on the test data. As explained in the overview of the challenge, a naïve estimator considering only the distance would score an error of around $5-8 depending on the specifics of the model [2]. This paper will use this value as a baseline and perform several data analysis techniques such as feature engineering and machine learning models to try and beat this value and minimise the error. It will do so utilising Python and in particular several modules, such as Pandas for data analysis and Scikit-learn for the majority of the models built. Several models which improved the base line were developed through the uses of ensemble models such as random forest regressors and gradient boosting algorithms. Coupled with these models several new features were extracted starting from the distances and time features as well as more distinct features regarding the specifics of the problem. The final score for the RMSE was of 3.051 and achieved with a gradient boosting algorithm known as extreme gradient boosting.

2. Pre-processing

2.1 The Dataset

In order to build a successful prediction model the dataset was firstly examined. The data was provided as part of the challenge by Google Cloud and consists of two main csv files:

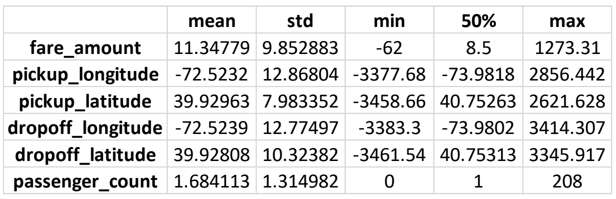
training and testing files, containing over 2GB of data. The bulk of this size deriving from the training data, consisting of over 55M rows, 6 features a key and a target variable (the fare amount).

This size provides the first challenge to analysing the data as importing the data as a simple pandas data-frame provides too expensive from a memory perspective for laptops without a significant amount of RAM. More advanced techniques can be used to import the whole data: when the file is imported to a pandas data-frame, the columns are treated as float64 types to store the results. However this precision is not needed for any of the features and as a result a more memory efficient set of types can be selected to decrease the memory usage. By then reading the data in chunks and concatenating it into a single final data-frame the total memory consumption decreases from over 3GB to just over 1.5GB [3]. The file can then be exported to a pickle file and read back into a data-frame at a much quicker rate.

A more straightforward approach is to just import a part of the data, say the first 2M rows, and use this sample as a representation of the whole data and preform the analysis on this data set. However by doing so you are discarding a big part of your training data which could help train models. Furthermore by importing the first 2M rows you could be introducing biases in the data given this is not a random selection within the whole dataset.

2.2 Data Cleaning

The next step in analysing is to clean the dataset from any obvious mistakes that are present within it. To do this simple statistical calculations of the dataset were made to notice any obvious outlier or mistakes as shown in Table 1.



	mean	std	min	50%	max
fare_amount	11.34779	9.852883	-62	8.5	1273.31
pickup_longitude	-72.5232	12.86804	-3377.68	-73.9818	2856.442
pickup_latitude	39.92963	7.983352	-3458.66	40.75263	2621.628
dropoff_longitude	-72.5239	12.77497	-3383.3	-73.9802	3414.307
dropoff_latitude	39.92808	10.32382	-3461.54	40.75313	3345.917
passenger_count	1.684113	1.314982	0	1	208

Table 1: The results for simple statistical information regarding the data for each feature. These results are calculated using the .describe() function for a pandas data-frame using 2M rows of the training data.

Firstly any missing entries were dropped as this only consisted of around 15 rows. Secondly the base fare of a journey in NYC is $2.50 [4], as a result any values lower than that are either errors or unrepresentative fares and were consequently dropped from the data. Similarly when looking up the coordinates for the town of NYC a set of boundaries for rides within NYC was made and values outside these boundaries was discarded as these points were considered outliers [5]. This lead to a significant drop in datapoints of nearly 40,000 points. Due to the significant amount of points dropped, better approaches than disregarding them could be examined, for example predicting the fare with a simple distance formula, as they could still represent useful data.

Thirdly the values for the passenger count also seem misleading as 204 as maximum number of people is evidently incorrect, as a result any values for passengers above 6 was disregarded. More robust cleaning could be made on the data-set, such as removing abnormal fare values as well as considering drop-off and pick-up locations in water as examined in [6]. Nonetheless the current cleaning procedure provides a sensible start to make some new features from the raw data.

2.3 Feature Engineering

The first and most obvious features were made in relation to the distance between pickup and drop-off location. The most straightforward approach is to take a simple linear distance in both latitude and longitude direction and use this as a way to determine a distance in both direction, even if not very precise as this doesn't take into account of the earth's spherical nature [7]. To do better than this we can use the haversine formula, which takes into account of this fact and calculates a single distance between the two locations [8].

Then another set of simple features were created from the date-time feature as you might expect some dependency caused from traffic or inflation changes which could be captured form temporal features. Hence a minutes, hours, weekday, month and year features were created from this.

Following this what can be understood from background reading and experience is that certain fares, for example to airports, are predetermined and as a result could be easier to predict [9]. Therefore a feature containing a 1, 2, 3 or 0 depending on the drop-off and pickup locations off different airports was introduced. Furthermore traffic flows in and out of the city might introduce some changes to the fare and therefore a feature giving a sense of direction could be beneficial. The way this was introduced was by giving each quadrant of a circle a number and assigning it to each cab journey. The cab journey direction was determined from the distance vectors engineered previously.

These features were then explored through several models explained subsequently and following this analysis some proved to be more predictive than others. In conclusion a total of 15 features were available for the models.

3. Results

3.1 Evaluation methods

Independent of the algorithm used a reliable and efficient way of evaluating our models is necessary in order to best select different regressors. As explained previously the metric for the competition is the RMSE. The RMS error is calculated using the following formula

$$RMSE = \sqrt{\frac{\sum_i (\hat{y_i} - y_i)^2}{n}}.$$

Where $\hat{y_i}$, y_i are the predicted and actual values respectively of the target variable.

Therefore when evaluating a regressor on our data we should take this into account and use this error metric in order to have the best performance on the test data. This metric also has the benefit of being very interpretable as it has the same dimensions as the predicted values. For example an error of 5 can be interpreted as an error of $5 compared to the predicted result [2].

Several methods can be used to apply this metric and evaluate a given model. A efficient and yet effective way is to split the pre-processed data into train and validate set. Use the first to train the model and then predict on both sets and get RMSE values for both the validate and training sets. The advantage is that it is not computationally expensive and it provides a good overview of whether you model is overfitting the training data as well as how accurate your result is.

More accurate techniques can be used such as using cross-validation (CV), however this is computationally expensive and as we are just interested in a quick and efficient way to calculate the error using CV entails a much slower process which outweighs its benefits, especially as we are dealing with such a big dataset.

Finally as the test data is small we can actually submit several models quite efficiently and quickly and get the actual RMSE score without too much difficulty. Having said this it would still be useful, especially for feature selection and tuning hyper-parameters to have a reliable method of predicting accuracy for each algorithm.

3.2 Base Model

The problem in question is an example of a supervised regression problem. This is due to the continuous distribution of value that a fare can have and consequently appropriate techniques should be used to build a classifier for the data. By plotting the results of the haversine distance against the fare amount, as shown in Figure 1, a clear relationship between distance and fare can be seen.

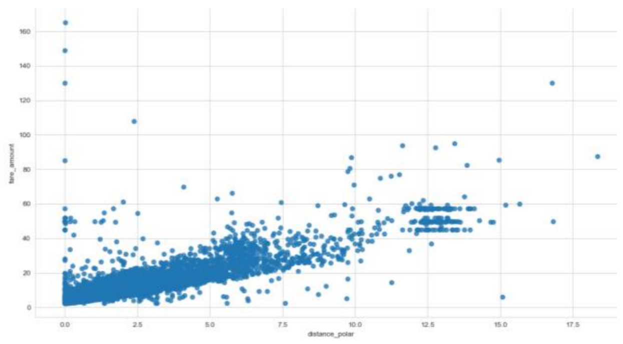


Figure 1: A plot of the haversine (or polar) distance against the fare amount for a sample of 1000 rows in the training data. A clear linear relationship can be seen. Furthermore horizontal lines independent of distance for the range between 12 and 15 highlight the previous intuition of flat fares for certain rides.

As a result the first and most simple model built acting as a base for the whole problem was a simple linear regression on the data using the haversine distance as the only feature. When scoring this result training and validate errors were respectively: 5.61 and 5.62.

What can be seen immediately is very similar results for both errors highlighting that the regressor isn't overfitting. This is to be expected as we are using the lowest dimensionality possible and as a result, due to the low complexity we would not expect any overfitting [10]. When predicting on the test data and submitting the predictions the error achieved is 5.61. This is further confirmation that our method for validating the data gives accurate results.

The first thing we can do to improve this result is add some of the features that we have created and that were present in the data. To better select features which will affect the regression model we can look for correlations between each feature and the target variable as seen in Figure 2.

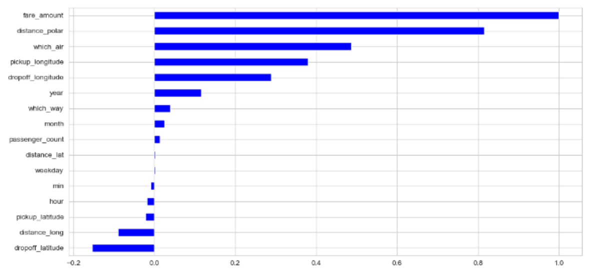


Figure 2: A bar plot of the correlations between attributes and 'fare_amount', the target variable. This was computed with the pandas functions .corrs() and the plotted using the matplotlib module in python.

What can be noticed is that most of the features created have a degree of correlation with the target variable and as a result could add accuracy to the regressor. As seen in both Figure 1and 2, apart from the distance, the airport feature especially seems to be an important attribute to gain information from. On the other hand, features such as the passenger count seems to carry little information. This seems reasonable as the fare amount in a journey is independent of the number of passengers. What can be inferred from this is that given the slight correlation that each feature has the best model will have all of the features present, even though arguably some could be discarded in order to decrease the complexity of the model. Feature selection confirms this and with all the features present in the model the RMSE scores is lowest, with values of 5.38 and 5.37 on the train and validate sets. To further improve the score more complex models should be built to get better results.

3.3 Random Forest

To increase accuracy further we must abandon a simple linear model as more complex relationships might exist between the different features in the data-set. A random forest regressor could help us capture these relationships. A random forest is an ensemble of decision trees and as a result it provides a more robust regressor to predict the results. It does come with drawbacks as it is obviously much more computationally expensive compared to more simple models. Furthermore unlike linear regression models these classifier become much less interpretable.

Firstly utilizing the default parameters of Scikit -learn's RandomForestRegressor on the validate and train sets gives a much lower RMSE of 3.97 and 1.68.

As we can see this already constitutes a significant improvement compared to the linear model. The difference between the results obviously suggests overfitting. Even though random forests are usually more robust classifiers against overfitting we must tweak the parameters for this particular example as the results suggest this model is strongly overfitting. To do this we look at two important parameters which define a random forest: the number of trees built and the max depth of each tree. The reason for addressing these two parameters in particular is because they address the issue of overfitting [11]. In particular by building more estimators the model is less likely to overfit as the different trees are built from scratch and as a result such produce different results for the same problem. Similarly by placing a maximum depth to each tree this also leads to less complicated trees and as a result a more general model. By tuning these values manually on the random forest an improvement was seen with using 100 estimators and a max depth of 10. Re-running the algorithm the RMSE scores for train and validate changed to: 3.73 and 4.02

Hence we can see that we have achieved the desired result and combated overfitting. Moreover we can use the feature importance function built in the classifier to give us the most important features used for classification. Even though the model is too complicated to interpret, this can give us an indication of which features contribute to a successful model.

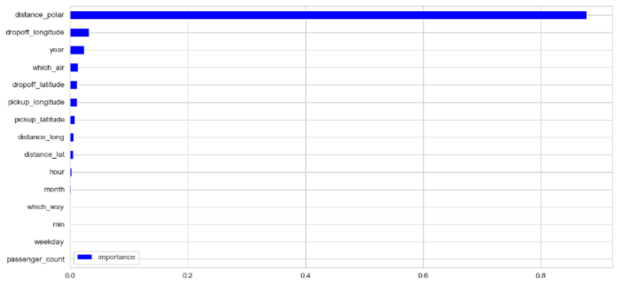


Figure 3: A representation of the relative feature importance for a random forest regressor when run with 2M rows of data. The importance's are found using the built in function .feature_importance() for a given classifier.

Figure 3 is in agreement with the results from the Figure 2 and what we had expected before the challenge. Distance is still the most important feature used by the algorithm. It also still suggests that the majority of the features created are useful to some degree.

To improve the model further, more rigours techniques can be used to choose the initial hyper-parameters for the model. Scikit-learn provides functions which systematically cross-validate the RMSE scores for a predefined set of parameters by iterating through each different selected hyper-parameter and as a result finding the best hyper-parameters for a given problem [12]. This approach is extremely computationally expensive, especially for random forests, and as a result only few parameters were tried using this approach. In particular, these were: max_depth = [10, 20] and n_estimators = [10, 100, 150, 200]. When running the algorithm the values of 10 and 150 were chosen respectively. Therefore the model was run on the test data to see what result this yielded on the actual data. The model produced a RMSE error of 3.35, a significant improvement from previously placing this result in the top 38 percentile for the competition.

3.4 Gradient Boosting

To improve the result further a series of options are available. Firstly we could try and run the regressors with more data as only 2M rows out of the 55M are currently in use. This however would time-consuming and as a result different approaches should be taken. Yet more complex algorithms commonly referred to as gradient boosting algorithms (GB) can be explored as these tend to yield better performance in many cases [13].

Two common algorithms of this type are known as Extreme Gradient Boosting (XGB) and Light Gradient Boosting (LGB). LGB is named like this due to its efficiency in scaling to large amounts of data and as a result could potentially scale to larger chunks of the data-set [14]. Just as for the random forest these two algorithms were tested independently whilst conducting similar hyper-parameter tuning as explained above. XGB gave the best train and validate score: 2.40 and 3.65. Whilst LGB performed slightly worse giving a validate and train score of: 3.62 and 3.80.

Just as previously, given the decrease in the RMSE score on the train and validate set the two classifiers were tested on the test data and submitted. The score respectively were 3.051 and 3.186. However as the LGB algorithm scales so well this value was computed using 10M rows and not 2M as for the other models.

3.5 Further Ideas

The last result submitted scores in the top 20% of the total ranking of the competition and consequently reducing the error even further becomes even more challenging. Nonetheless several ideas can still be used to improve further. Firstly there are still lots of different algorithms which have not be implemented which could produce more successful results, for example any deep learning techniques, which are considered the state of the art predictors nowadays [11]. This is especially true as they require large amounts of data to train as we have in this problem [15]. Another approach could simply be to ensemble different models together by means of weighted averages, depending on their errors, and use the averaged predictions as the final ones. These techniques could all potentially lead to an improved result for the RMSE.

4. Conclusion

In conclusion, a significant improvement in the base model was achieved. This was achieved through several different techniques: firstly analysing and cleaning the data-set and then using several different models to try and predict on the test data. Alongside alternative feature engineering techniques were produced with regards to the specifics of the problem such as trying to encode if the journey derived or went to an airport. The best model proved to be the XGB regressor coupled with all the features available. Finally as discussed previously more computational power could allow for more performance for task such as choosing better hyper-parameters for the models and using more data for training. A novel approach to improve the result further could also to use alternative data sources to gain additional information. For example data such as the weather, average congestion rates or similar information could all help in the predictions of the results.

References

[1] UBER, “Riding with Uber- Upfront pricing,” [Online]. [Accessed 17 12 2018].

[2] Kaggle, “New York City Taxi Fare Prediction,” 8 2018. [Online]. Available: https://www.kaggle.com/c/new-york-city-taxi-fare-prediction#description. [Accessed 12 2018].

[3] szelee, “How to import a CSV file of 55 million rows,” 9 2018. [Online]. Available: https://www.kaggle.com/szelee/how-to-import-a-csv-file-of-55-million-rows. [Accessed 12 2018].

[4] NYC, “Yellow Taxi Fares,” [Online]. Available: https://www1.nyc.gov/nyc-resources/service/1271/yellow-taxi-fares. [Accessed 17 12 2018].

[5] W. Koehrsen, “A Walkthrough and a Challenge,” 10 2018. [Online]. Available: https://www.kaggle.com/willkoehrsen/a-walkthrough-and-a-challenge. [Accessed 12 2018].

[6] A. v. Breemen, “NYC Taxi Fare - Data Exploration,” 8 2018. [Online]. [Accessed 12 2018].

[7] D. Sterling, “NYC Taxi Fare Starter Kernel - Simple Linear Model,” 2018. [Online]. Available: https://www.kaggle.com/dster/nyc-taxi-fare-starter-kernel-simple-linear-model. [Accessed 12 2018].

[8] G. R. van Brummelen, Heavenly Mathematics: The Forgotten Art of Spherical Trigonometrry, Princeton University Press, 2013.

[9] The City of New York, “Taxicab Rate of Fare,” 2018. [Online]. Available: http://www.nyc.gov/html/tlc/html/passenger/taxicab_rate.shtml. [Accessed 12 2018].

[10] M. G. Simon Rogers, A first course in machine learning, Boca Raton: CRC Press, 1979.

[11] R. E. Neapolitan, Contemporary Artificial Intelligence, Second Edition : With an Introduction to Machine Learning, Boca Raton, FL : CRC Press, 2018.

[12] scikit-learn Documentation, “sklearn.model_selection.RandomizedSearchCV,” 2018. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html. [Accessed 12 2018].

[13] A. K. Alexey Natekin, “Gradient boosting machines, a tutorial,” 03 12 2013. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3885826/. [Accessed 15 12 2018].

[14] “Python-package Introduction, LightGBM Documnetation,” 2018. [Online]. Available: https://lightgbm.readthedocs.io/en/latest/Python-Intro.html.

[15] C. M. Bishop, “Neural networks and their applications,” *Review of Scientific Instruments,* vol. 65, no. 6, 1994.