PREDICTING NYC TAXI TRIP DURATION

MIKEL BOBER-IRIZAR – MLND CAPSTONE PROJECT

1 DEFINITION

1.1 PROJECT OVERVIEW & INTRODUCTION

Taxis are a vital part of the ecosystem of many cities. In New York City alone, over 300,000 yellow cab trips are taken every day, in addition to almost 300,000 ridesharing trips. Due to the advent of ridesharing apps such as Uber, the total number of

To be able to operate a taxi service at such a large scale, companies use electronic dispatching systems to efficiently assign cabs to customers and spread out cabs across cities where they might be needed in order to maximize the number of rides each cab can take in a day.

One vital aspect of such systems is to predict how long a taxi trip will take once a cab is taken, so the system can understand how long a specific cab will be removed from the pool of free cabs and when it will be free to pick up more passengers. This allows dispatchers (and automated dispatch systems) to reduce passenger waiting times and increase revenue from each cab, creating a win-win situation.

The machine learning website Kaggle has hosted several competitions to predict the duration of taxi trips, such as the ECML/PKDD 2015 competition which used data collected from taxi trips in Porto, Portugal. In order to build an algorithm that can accurately predict the duration of a taxi cab trip, I will be using the dataset provided by the recently launched and ongoing New York City Taxi Trip Duration competition, which has a very large dataset released by the NYC Taxi and Limousine Commission.

1.2 PROBLEM STATEMENT

The Kaggle dataset consists of data collected from New York City over a period of 2009 to 2016 – in total, it contains information from over two million rides.

The goal of the problem is to predict the duration in seconds that a given taxi trip will take – making it a supervised regression problem. My capstone project will focus on trying to solve this problem and getting a maximal score on the Kaggle leaderboard based on the metrics described in the following section.

As inputs, we are given information such as the company which runs the taxi, the starting and ending locations, and date/time information – this is the information that we need to predict ride duration from. As the data given is tabular in nature, my approach will be to use standard supervised regression algorithms such as decision trees, support vector machines or linear regression.

Overall, I will approach the problem in multiple steps. Firstly, I will explore and visualize the data to gain an understanding of the features and where the 'signal' in the dataset lies. This will allow me to go onto feature preprocessing, where I will convert the features into formats more suited towards the classifiers I will be using, as well as performing 'feature engineering', a term commonly used on Kaggle which refers to creating entirely new features which may be more predictive out of the existing features.

After I have created my final feature space that I am happy with, I will build a classifier upon the data to obtain a score. When I have obtained my classifier, I will perform parameter tuning to maximize the score obtained on a validation set, and then use the best model to create final predictions which I will then upload to the Kaggle leaderboard to obtain my final score.

I anticipate that my final solution will consist of a feature processing pipeline followed by a single trained supervised model which outputs predictions for the test set that can be uploaded to the Kaggle leaderboard.

1.3 METRICS

To measure the performance of the model, I will use the Root Mean Squared Logarithmic Error metric (referred to as RMSLE).

The RMSLE is defined on the Kaggle evaluation page as:

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

Where:
N is the number of samples,
p_i is the prediction of the trip duration,
a_i is the actual duration
$\log(x)$ is the natural logarithm of x.

This metric is **identical to Root Mean Squared Error (RMSE),** which is widely used for regression problems, **except that** the "log1p" of both the target and predicted values are taken

beforehand. `log1p(x)` is defined as `log(x + 1)` - the +1 is there to avoid taking the log of 0 (which is undefined).

This parallelism to RMSE is very useful, because it means that we can make a learner that optimizes RMSE (most out of the box regression algorithms) instead directly optimize RMSLE by taking the log1p of the target value beforehand and feeding that to the learner in place of the original target.

There are two reasons I have decided to use this metric. Firstly, this is the official metric that we need to optimize for the Kaggle competition and the metric upon which the submissions to the competition are ranked, so it makes sense to also try to optimize this locally.

In addition, I believe using the log-error also makes more sense than directly using the error for each trip. This is because we care more about the error in each sample **relative to the trip time** as opposed to simply the absolute error.

For example, one would say that a 5 minute trip incorrectly predicted as 2 minutes is much worse than a 50 minute trip incorrectly predicted as 47 minutes. If the RMSE metric was used, these two trips would have the same error. However, RMSLE would penalize the 5 minute trip more, even though both predictions were incorrect by three minutes. This seems more reasonable to me than penalizing them both equally, so for this reason I think RMSLE better represents what the model is actually trying to learn.

2. ANALYSIS

2.1. DATA EXPLORATION

All code for this section and section 2.2 is in `data_exploration.ipynb`

As part of the dataset from Kaggle, we are given two files, a `train.csv` and a `test.csv`. These two files represent the training and testing data given by Kaggle – the formats of the files are identical except for the fact that the testing data does not have target values included.

The training set contains 1,458,664 trip records, while the test set contains 625134 records, making this a rather large dataset. Each trip record is represented by a row in the csv file, and has several features given for it. I have described each feature given briefly below:

Feature	**Description**
`id`	The ID of the trip. Not to be trained on
`vendor_id`	A categorical variable indicating the taxi provider associated with the record

`pickup_datetime`	The date and time that the taxi meter was engaged (and the passenger was picked up)
`dropoff_datetime`	The date and time that the taxi meter was disengaged. **This feature is only present in the training set.**
`passenger_count`	The number of passengers in the vehicle
`pickup_longitude`	The longitude of the passenger pickup location as a float
`pickup_latitude`	The latitude of the passenger pickup location as a float
`dropoff_longitude`	The longitude of the passenger's destination as a float
`dropoff_latitude`	The latitude of the passenger's destination as a float
`store_and_fwd_flag`	Whether the trip was "store and forward", meaning that the vehicle did not have any connection to the server during the trip and trip details were uploaded later. Denoted by "Y" or "N" values
`trip_duration`	The total duration of the trip in seconds. This feature only appears in the training set and is the **target value**.

This means we have a total of 8 features that we can train on, which includes one categorical, one Boolean and one timestamp feature, the rest being float-valued.

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration
count	1458644.000	1458644.000	1458644.000	1458644.000	1458644.000	1458644.000
mean	1.665	-73.973	40.751	-73.973	40.752	959.492
std	1.314	0.071	0.033	0.071	0.036	5237.432
min	0.000	-121.933	34.360	-121.933	32.181	1.000
25%	1.000	-73.992	40.737	-73.991	40.736	397.000
50%	1.000	-73.982	40.754	-73.980	40.755	662.000
75%	2.000	-73.967	40.768	-73.963	40.770	1075.000
max	9.000	-61.336	51.881	-61.336	43.921	3526282.000

From the statistics above we can see that the (latitude, longitude) pairs are clustered in a very small region around (-74, 40) – this is expected since that is the location of New York. However, there are some very large outliers thousands of miles away – these are most likely GPS errors. A similar case can be seen in the `trip_duration` target variable: The max of this variable is equal to just over 40 days, which is obviously an erroneous measurement. Such outliers are to be expected in such a large dataset. However, they are not very common – for example, there are only 82 samples out of 1.5 million which have abnormally low longitude values, so these anomalies are likely to be ignored by most machine learning algorithms.

The categorical `vendor_id` variable contains only two possible IDs, with 47% of the taxi trips containing the vendor ID '1' and the rest containing '2' – this variable should instead be treated as a simple boolean variable. The `store_and_fwd_flag` variable is also a boolean, but is only positive in 0.5% of taxi trips, showing a rare event. It's unclear how this variable could affect ride times.

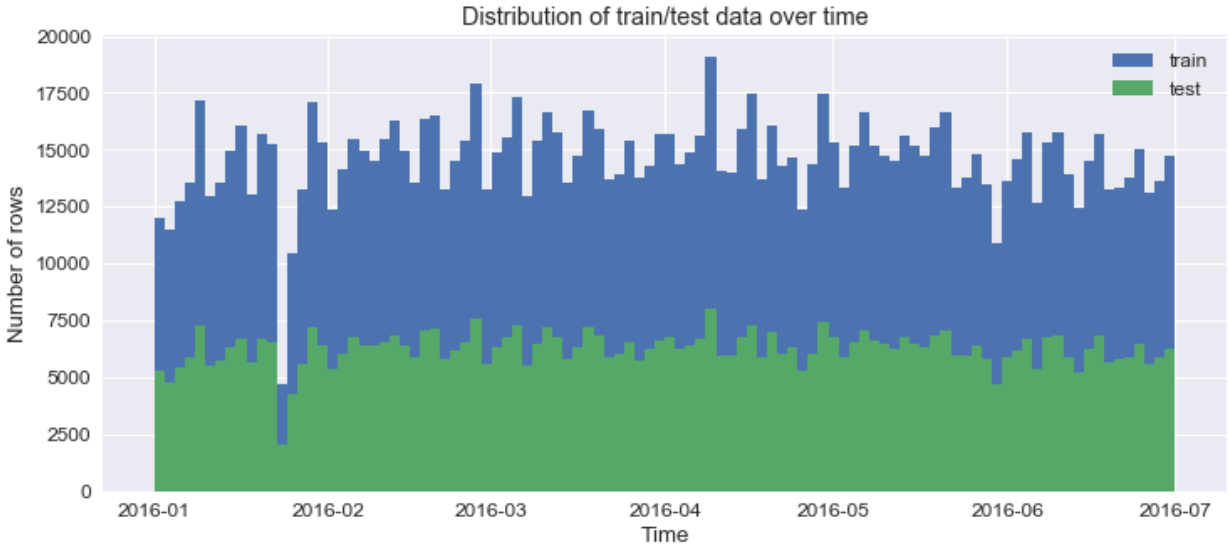
2.2 DATA VISUALIZATION

In this section, I will visualize and analyze some interesting aspects of the dataset given to gain a better understanding of different aspects of the data. The information gained from this is vital in order to figure out what feature engineering may help extract signal from the data, as well as what types of models may work well on the data.

I hypothesise that this problem has a substantial aspect of time-dependence, meaning that the distribution of the target changes a lot with respect to the time of day (traffic conditions may change, for example), as well what day of the week or year it is (special occasions may affect taxi trips, for example).

For this reason, I think it is important to investigate how the pickup datetime affects the data. One important aspect is to look at how the split between the training and testing sets are done. There are two different ways this could have been done:

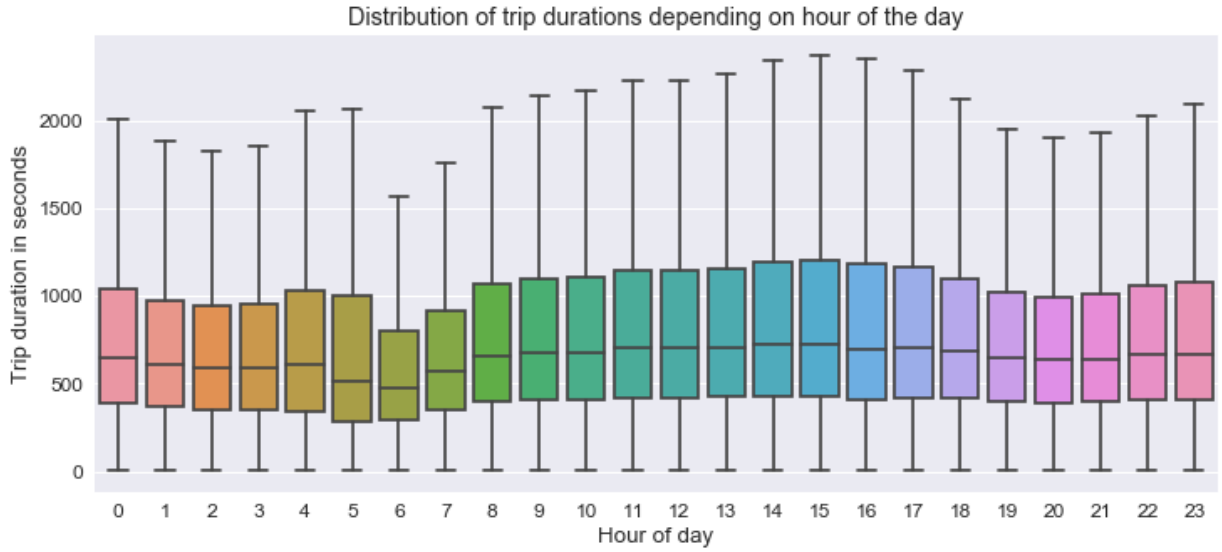
1) The testing data is sampled from the **same period of time** as the training data. This would allow us to get a lot more information out of the training set – for example, when predicting on a sample in the test set, we could look at trips made around the same time period in the training set in order to understand the current traffic situation.
2) The testing data is sampled from a period of time **in the future** (relative to the training data). This creates a very different problem: instead of trying to predict a subset of the



taxi rides around the same time, the challenge is instead about creating a model which could predict taxi trip time on a future day (where the outcome of recent taxi trips are not known). This option would make more sense for the data, as taxi companies are not interested in predicting on past data but instead obtaining a model that can tell them how long future rides will take.

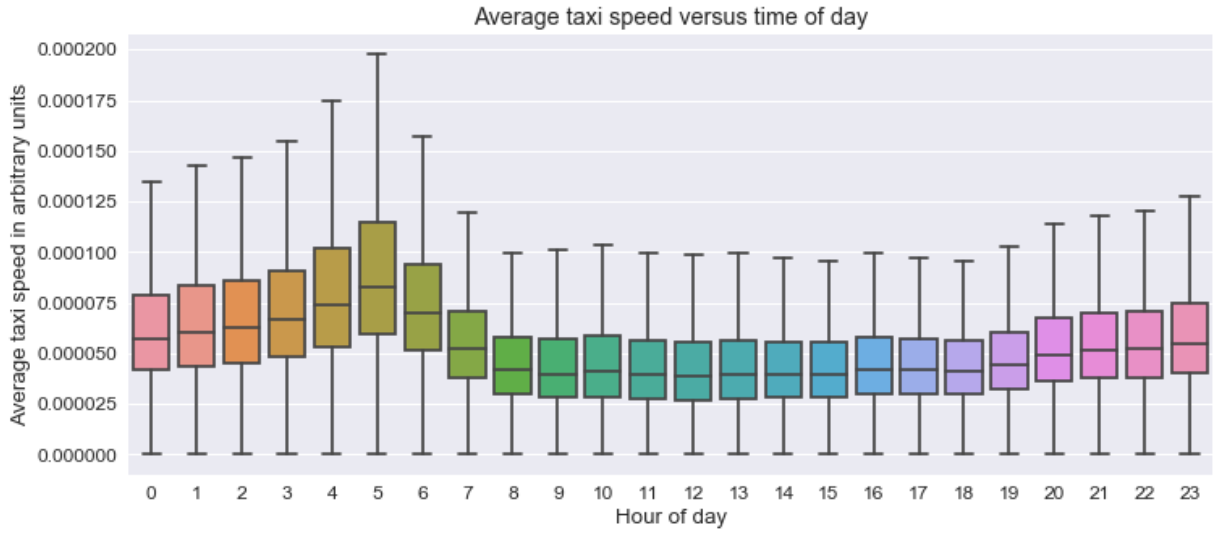
From the above plot, we can observe that the dataset is relatively uniformly (apart from an unusual dip in in January) distributed throughout the first half of 2016 (183 days in total). More interestingly however, both the training and test sets occupy the same distribution, which shows that the first approach described above was taken in preparing the data. This means that we are actually given target values for other taxi trips around the same time period of each test trip, and these can be used in order to better predict the trip time.

We can also look at how the trip duration changes based on the time of day using a boxplot:



This graph shows that the duration of taxi trips varies quite dramatically based on the time of day. In the early morning (5-8am) and to a lesser extent in the evening (19-21), customers tend to take quicker trips than during other parts of the day. At 8am and onwards, the average trip duration tends to get longer. I believe this is due to increased traffic conditions at peak times meaning that taxis cannot travel as fast. However, to rule out the possibility of people simply travelling further during the day, I need to analyze the speed of the taxis directly.

Since I am only given co-ordinates of the pickup and dropoff points, we are not given any information about the speed of the car. However, this can be approximated and inferred from other information: For this analysis, I will use the Manhattan (L1) distance between the two points and divide this by the trip duration in order to get a rough estimate of average speed. If we then plot the calculated speed using another boxplot, more interesting observations arise.



Here, the effect is even more pronounced. We can see that by my speed metric, the early morning is the time at which taxis travel fastest, and this slows down very substantially (average speed roughly halves) beginning at around 7am onwards. In fact, we can see that these observations almost perfectly match data released by the NYC Taxi Commission in 2013, which put the peak speed at 5:18AM and showed the same relationship for the rest of the day.

From the above analysis, I can see that time has a very large effect on taxi trip duration. For this reason, I will focus on building features that can capture this relationship (as well as the relationship with taxi speed) in the modelling phase, which should help predict trip duration at different times of day better.

2.3 ALGORITHMS AND TECHNIQUES

XGBoost

For training models, I have decided to use the XGBoost algorithm. XGBoost is an implementation of a machine learning algorithm known as Gradient Boosted Trees (GBT), also referred to as gradient boosting machines (GBMs). The concept of this algorithm is as follows:

1) Fit a decision tree to the data
2) Evaluate the model against each data point
3) For each sample in the dataset, increase its weight based on how incorrect the model was in the last step
4) Fit another decision tree to the data using the reweighted data – this new tree will fit more to the areas where the previous trees did not perform well
5) Add this new tree to the ensemble, go to 2

The result of this is an algorithm which has the benefits of other tree ensemble models (such as random forests) while outperforming other similar techniques in performance.

I have chosen to use this algorithm for multiple reasons. Since it is a tree-based algorithm, it is completely scale-invariant. This means it can very easily handle features with bad scales, such as latitude and longitude, reducing the amount of preprocessing that needs to be done.

In addition, due to its tree-based nature, it can learn non-linear relationships as well as relationships between features. This allows it to directly use the latitude and longitude features – it can learn different patterns for different areas of New York by splitting on these features where a linear model would not be able to. For example, using four splits in a branch the model could check if the car is located in a specific rectangle of the latitude and longitude.

XGBoost is widely used (and often wins) in Kaggle competitions as it is known to tend to outperform similar supervised models extremely well on tabular data with a range of complex non-linear features, as we are given here.

The downside of using XGBoost is that it can potentially be prone to overfitting. However, this is much less of an issue when large datasets are present (as is the case here), and this can also be mitigated by using a validation set. The model can be evaluated on the validation set after every tree is added and 'early stopping' can be used to stop training the model when it stops improving on the validation set, lest it overfit.

When training, XGBoost takes in a matrix of (samples, features) and a vector of target values, returning a model. This model can then be used on another matrix of the same format (the test set) to return a vector of predicted target values. XGBoost only supports numerical input features, so features such as datetime and categorical variables will need to be transformed beforehand.

There are a few basic XGBoost parameters that need to be tuned to obtain optimal results:

Parameter	**Default**	**Explanation**
`max_depth`	6	The maximum depth (number of splits in a branch) of each individual tree in the ensemble.
`colsample_bylevel`	1.0	The proportion of input features which are available to the model to split on at each level of the tree. Decreasing this means trees are more likely to be different, which can improve ensemble performance.
`subsample`	1.0	The proportion of the data which each tree is trained on. Decreasing this has a 'bagging' effect which works for the same reason as for `colsample_bylevel`.
`eta`	0.3	The learning rate (amount of reweighing between trees). Decreasing this always helps performance, but learning takes much longer for diminishing returns.

2.4 BENCHMARK

Arguably the simplest benchmark which can be used to compare between models is the performance obtained by always predicting the same value (this can be considered equivalent to a prior probability). In this case, I will find the single trip duration that minimizes RMSLE on the training set, and then use this to create a submission on the Kaggle leaderboard.

The optimal trip duration estimate can be found with:

$$optimal\ trip\ duration = expm1(mean(log1p(trip\ durations)))$$

This yields us an trip duration estimate of **642.54 seconds,** which gives us a **0.796 RMSLE score** on the training set and a **0.798** score on the Kaggle public leaderboard.

Thus, we can say that for a model to have learnt anything less trivial than the prior of the data, it must have a RMSLE error of less than our benchmark **0.798** on the leaderboard.