Pro Tip
A Project on NYC Taxi Tip Predictions

Hitesh Bahar & Ryan Lattanzi

March 9, 2019

Abstract

In this paper, we investigate predictions of tip amounts for yellow taxi cabs in New York City. We do not attempt to predict a continuous value; rather we group tip amounts into classes, transforming our problem from regression to classification. After feature engineering and feature selection, we implement KNN, Decision Trees, SVM, Naive Bayes, and XGBoost classifiers and compute their testing accuracies. Finally, we create an ensemble with the five classifiers to see if it improves our accuracies. Despite some uncertainties initially, we were happy to see the results are very good.

1 Introduction

The sea of yellow taxi cabs swarming the streets is one of the iconic images of the Big Apple. So, what better place to study taxi cab data than New York City? This is precisely what we set out to do. The NYC Taxi and Limousine Commission (TLC)[1] has collected an enormous amount of data for every taxi ride in New York City from 2009 up until now. We set out to get our hands dirty and put to use a multitude of the powerful classification algorithms we learned throughout the semester.

In section 2 we walk you through the problem setup and motivation. Section 3 describes the important aspects of preprocessing our data. Section 4 presents the reader with interesting analytics of the training data and includes some pretty pictures. Section 5 outlines ups, downs, confusions, and (finally) successes of the final experiment plan. We wrap everything up in section 6 with our results and conclusions.

2 Problem Setup

Of the many problems that could be formulated from observing this data, we attempt to predict the tip amount of customers given the other features that were collected. As predicting precise amounts would also be an interesting problem, we decided to transform this problem into a classification one because it is relevant to class material. We did this by sorting tip amounts into classes:

$$\begin{aligned} \$0 - 2 &\rightarrow \text{ Class 1} \\ \$2 - 4 &\rightarrow \text{ Class 2} \\ \$4 - 10 &\rightarrow \text{ Class 3} \\ > \$10 &\rightarrow \text{ Class 4} \end{aligned}$$

When contemplating this question, we did recognize its limitations. Of the many wonderful features that were collected, the data was still missing perhaps the most important feature(s) that could influence tip amount: the customer himself. That is, the customer's mood at the time of the ride, personality, selfish or altruistic tendencies, influence of or interaction with the driver, etc. So before executing this project, we kept in the back of our mind the possibility that our experiment could yield very low accuracies. Another limitation was scalability. Our computers as mere graduate students are not powerful enough to run such a vast number of data provided, so we had to be selective about which ones to use. We will elaborate more on this in section 3.

The main motivation for this problem was to work with big data to successfully manipulate and use it towards a solution. We also wanted to choose a dataset that would allow us to come up with creative ways of feature engineering and data cleaning in order to provide us with a full experience of the data science

pipeline. However, this information could be put to real-life use by persuading (or dissuading) potential drivers to work for the companies, or allowing companies to alter wages based on tips. Although it is true that taxi cabs are becoming obsolete to be replaced by companies such as Uber and Lyft, this model can easily be extended to cater to the newer taxi services.

3 Data Preprocessing

3.1 Selecting a Subset of the Data

The NYC TLC collected data for every yellow, green, or for-hire taxi rides for every month from 2009 until now. Because of the widespread use of taxis, each month had about nine million entries. In order to allow a scalable problem, we decided to only consider recent yellow taxi data from 2017 and the first six months of 2018 to account for relevant trends. However, this was still too large. To further scale down, we extracted only the rides that had payments made by credit card. This would help generalize our problem to modern taxi services since the method of payment is typically by card. Furthermore, to get a spread of different types of customers, we also extracted those rides that picked up customers in Times Square. With this subset, each month had approximately 200,000 entries, so our training set example was still around 8 million. Due to computational complexity issues that will be explained in section 5.2, this was still too large; it was necessary to downsize again. However, we realized downsizing more would also make our problem more realistic. In real life problems, coming about such a large labeled training set is seldom feasible; gathering labels is typically an expensive process. Hence we decided to reduce our training set to approximately 100,000. To achieve this, we randomly sampled 7,000 examples from each month and stacked them together via rowbind.

Now that our problem was scaled appropriately, we chose our training set to be all of 2017 along with January, March, and May of 2018. This left our testing set to be February, April, and June of 2018. The reason for this particular split in the 2018 data is to take into account potential trends in customers as the seasons change. We allow one month in each season: winter, spring, and summer (for both training and testing data).

3.2 Feature Engineering

The reader can find a data dictionary of the original features in the appendix. Below is an example of how the raw data looked:

VendorID	tpep_pickup_	tpep_dropoff	passenger_c	trip_distance	RatecodeID	store_and_fv	PULocationID	DOLocationID	payment_typ	fare_amount	extra	mta_tax	tip_amount	tolls_amoun	improvemen	total_amount
1	4/1/18 0:47	4/1/18 1:08	1	6.7	1	N	152	90	2	22.5	0.5	0.5	0	0	0.3	23.8

Of these, we dropped the pick-up location, store and forward flag, and payment type features since each had only one value. We created a response column entitled *tip_class* by placing data into buckets corresponding to the bounds outlined in section 2. The rest of our feature engineering process consisted of manipulating the columns that collected pickup and drop-off dates and times.

First, we created four evenly sized time zones for pickups and drop-offs (midnight to 6 am, etc.). Then, based on the months, we created season variables (winter, spring, summer, fall). We also changed the dates to days of the week using the **weekdays** function in R. Finally, we one-hot encoded all of these categorical variables to get a 32-dimensional feature vector:

VendorID	passenger_c	trip_distance	RatecodeID	DOLocationID	fare_amount	extra	mta_tax	tip_amount	tolls_amoun	improvemen	total_amoun	szn1	szn2	szn3	szn4
1	2	3.3	1	4	23.5	0.5	0.5	4.95	0	0.3	29.75	0	0	0	0

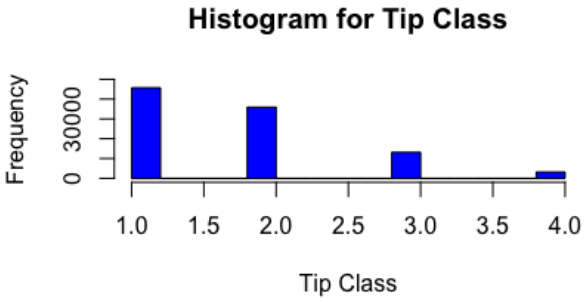
sun	mon	tues	wed	thurs	fri	sat	pt1	pt2	pt3	pt4	dt1	dt2	dt3	dt4	tip_class
0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	3

4 Data Analysis and Trends

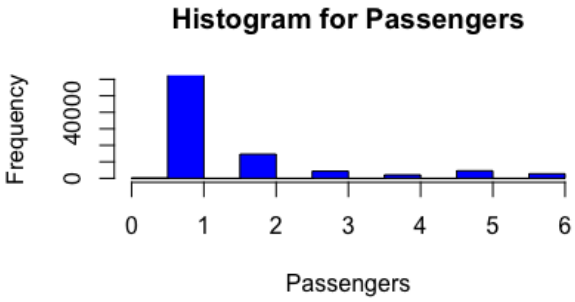
NOTE: All analysis and trends in this section were extracted from the training data.

It was interesting to explore the different relationships between tips and other features. First of all, we note that we had a nice spread of tip classes that followed an inverse relation which we would expect, as shown in Figure 1. Also shown in Figure 1 is the distribution of passenger counts. Table 1 exhibits some basic analytics from the data.

Figure 2 shows the relationships between the trip distance and trip fare features to the tip amount. We can see they are positively correlated to the tip amount, as we would expect. This is emphasized by including the linear line of best fit in each plot.



(a) Histogram for Tip Class

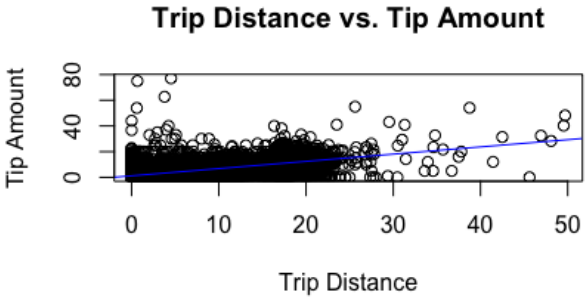


(b) Histogram for Passenger Count

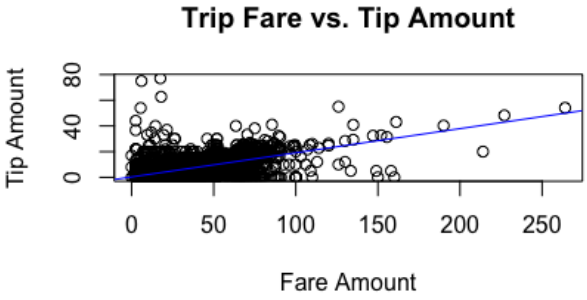
Figure 1: (a) shows the distribution of tip classes. Class 1 had 45,752, class 2 had 35,935, class 3 had 13,130, and class 4 had 3,183. (b) shows the distribution of passenger counts.

Table 1: Basic Analytics

Avg Tip	$2.89
Max Tip	$77.09
Avg Distance	3.05 mi
Max Distance	49.7 mi
Max Distance Tip	$48.22
Max Fare	$264
Max Fare Tip	$54.11



(a) Trip Distance vs. Tip Amount



(b) Trip Fare vs. Tip Amount

Figure 2: These plots indicate a positive correlation with the response variable.

5 Experiment

5.1 Approach

Real-life data science projects will not require writing our own code when implementing models, so we pulled our classifiers from sckit-learn[2]. The five models that were chosen for our multi-class problem were:

1. KNN
2. Decision Tree
3. SVM
4. Naive Bayes (Gaussian and Bernoulli)
5. XGBoost (Decision Trees as base classifiers)

After computing predictions for each, we constructed an ensemble with these five classifiers to see if there was any improvement. We did not want to run each model on the full 32- dimensional vector. Rather, we wanted to implement a sequential feature selection process from *mlxtend* [3] that returned the best feature combination for each classifier when fit to the training data. We thought this would reduce computational complexities of our problem. However, this was not the case, so we had to abandon the idea (see section 5.3).

5.2 Hyperparameters of Classifiers

Hyperparameters of interested classifiers were tuned via cross validation over a small grid of values. We pulled *GridSearchCV* from scikit-learn [4] to implement into our function. This takes a dictionary of hyperparameters as an input and performs a 3-fold cross validation (number of folds could be altered) to find the best value of the hyperparameters in question. We were limited by computational complexities, but with more computing power our function would easily generalize to search over more hyperparameters and more values if desired.

For the KNN classifier, the only parameter we tuned was $n\_neighbors$. Our grid consisted of $3, 5$, and 7 neighbors. Note the computational complexity of this task: we would have to run a KNN for each value and more 3 times for each value since we are cross validating. Hence, we our complexity is

$$3\Big(O(3nd) + O(5nd) + O(7nd)\Big).$$

Because the decision tree classifier was achieving approximately 99% testing accuracies, we decided not to tune the max depth of the tree and leave it as is.

SVMs have a multitude of hyperparameters that could be tuned. To name a few, we have the kernel function to use, kernel coefficients, and the penalty parameter, C, of the error term. For the sake of running time, we allowed the kernel coefficient to be fixed at a scaled constant. Thus, the cross validation considered a linear or RBF kernel, and C values of 1 or 10.

We did not tune any hyperparameters of the Naive Bayes Classifier, but we note the NB algorithms to choose. Because we had a mixture of continuous and categorical features, running either a Gaussian or Bernoulli NB over all of the features would (and did) yield disappointing accuracies. Thus, we independently ran Gaussian NB on the continuous features and Bernoulli NB on categorical features. To get our predicted class labels, we multiply the likelihoods found from each algorithm to produce our posteriors for prediction.

5.3 Initial Issues

We encountered a daunting issue when we attempted to run our experiment with 8 million training examples and feature selection: our code could not get through even the first classifier after running all night. Our first thought was to reduce the sample size - we had not even considered the feature selection to be part of the issue! The time complexities of each classifier are shown in Table 2, where n is the number of training examples, d is the dimension of the feature vector, r is the number of classes, k is the hyperparameter for KNN, K is the total number of trees for XGBoost, and p is the depth of the decision tree.

Table 2: Complexities

Classifier	Complexity
KNN	$O(knd)$
Dec Tree	$O(pnd)$
SVM	$O(n^2)$-$O(n^3)$
Naive Bayes	$O(rd)$
XGBoost	$O(Kpn(\log n))$
Ensemble	$O(n)$

We can see that the SVM has the largest computational cost and our experiment corroborated its long running time. After reducing the training set as mentioned previously, our issue remained particularly at the SVM classifier. We turned our attention to the feature selection function. This function took as an input a tuple of a range of features to search over and find the combination that optimized accuracy. In our case, this tuple was $(1, 30)$ because we had 30 features. Even with a greedy search, this would require the SVM (and all other classifiers) to run too many times to be feasible. We had to rid our function of this search in order for it to run.

6 Results and Conclusions

The performance of each model was determined by testing accuracies. We were surprised to see great results. They are as follows:

Table 3: Testing Accuracies

Classifier	Accuracy
KNN	83.7
Dec Tree	98.9
SVM	99.5
XGBoost	91.2
Ensemble	98
Naive Bayes	48.8

As we can see, all classifiers except NB yielded favorable results. Because of this, we decided to remove it from our ensemble so as to not skew the results. We feel our logic was sound for implementing this classifier, but we will continue to investigate its behavior by modifying its implementation method. Perhaps we did not extract the correct likelihood probabilities to create our posteriors. If this was the case, another method that would bypass this confusion is to discretize the continuous features into buckets, one-hot encode them, and run a Bernoulli NB on this new data.

Besides that classifier, these powerful results indicate it is possible to predict the range of a tip amount without considering the personality of the customer himself. This was very surprising to us, as we thought this was a limitation of our experiment. Nonetheless, we are excited about the results. Because of these high accuracies, a natural extension of this project would be to run a multitude of regression models in an attempt to predict the actual tip value. Using classification was a good starting point to see if predictions were possible given the features that were obtained.

Overall, we feel this project taught us a lot in terms of being resilient when things did not go to plan. There were many times in which we had to reevaluate our methods (either by removing or adding some components) in order to improve our results. Our favorite part was the data preprocessing and analytics.

7 Contributions

It is difficult to list each individual contributions since both members worked on the project together; we seldom worked on it separately. However, Hitesh was the man in charge of running the code, most of the data preprocessing, and part of data analysis. Ryan wrote the ensemble code and implementation of classifiers, some data analysis, and played a big role in writing the report. All of the ideas to include in the paper and for the project itself were contributed by both authors (this includes which classifiers to use, how to formulate the problem, etc.)

8 Sources and Acknowledgements

[1] http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

[2] https://scikit-learn.org/stable/

[3] https://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/#overview

[4] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV.predict

We would like to thank Professor Jana Doppa (WSU) for the past semester. He challenged us everyday in order for us to learn the topics thoroughly to carry them out to the real world. Although it was tough, we enjoyed the class.