Profit Prediction: Bay Area Bikeshare

Analytical Project

Team DAD

Daniela Barrera, Daniel Man, Avi Rynderman

Introduction

The goal of this project was to use machine learning to predict profits on a given date for the Bay Area Bikeshare program. Our plan was to use data publicly available from the program to determine the ridership on a given day, month, or season based on the time of year, weather, and location. With this ridership data, we hoped to predict profits. Machine learning would be used in the data analysis, and prediction phases. By analyzing the weather data we hoped to determine what would classify as a "good weather day" and would would classify as a "bad weather day." This classification would be the result of clustering. We hoped to find unique patterns in the data and then use the clusters to form classes. After determining optimal weather classes, we would then use random forests and decision trees to find ridership based on the data features. The system design and implementation details will first be discussed with some information about each machine learning algorithm used, and the tools used in the project. The data preprocessing phase will then be detailed with an overview of the dataset, a discussion on the data preprocessing phase, and a segue into the actual machine learning done in this project. The implementation details for the machine learning algorithms used will then be presented. This will be followed by an analysis of the results. The report will conclude with an analysis of the project, difficulties faced, and a reflection on what worked well and what did not.

Dataset

The dataset used for this project was sourced from the Bay Area Bikeshare program (http://www.bayareabikeshare.com). The data is freely offered on the program's website. It contains just under a million individual rides sorted by date. Included with each ride entry is the number of seconds each bike was used for, where it was picked up, dropped off, which day it was used, and whether or not the rider was a program subscriber. Also included in the dataset was the weather for each day of the program. The weather data included temperature, humidity, cloud cover, precipitation index, zip code, and a couple other features. Station details were also included in the dataset such as the station location, station name, and station ID.

System Design & Implementation Details

Before working on our project we had to decide on a development environment and the algorithms we thought would be suitable for the project. After reviewing the available IDEs and sharing platforms available, we decided on ipython's Jupyter Notebooks for programming, github for collaboration, and **scikit** for our machine learning algorithms.

Jupyter Notebooks have been the IDE used in lecture, and offer a variety of services including a file system, integrated run control, graphics visualization, and integrated comments. Sharing these files and our data amongst members was accomplished with Github (https://github.com/avirynd/Bikeshare). This allowed quick uploading of files, edit tracking, and version control.

Our project includes the use of 3 machine learning algorithms; Random Forests, Clustering, and Decision Trees. These came from the **scikit libraries**. The implementation

details will be discussed below, with each team member detailing how they used the algorithm.

Clustering ***Daniel***

Clustering was an algorithm we justify in using to help visualize our data. By seeing the clusters, we hope to use them in conjunction with our other algorithms. For clustering, we wanted to gather the information surrounding the temperature and humidity to generate clusters based on the weather. This information is then plotted against the number of riders and the total amount of minute ridden for that given day. By having the two correlated, we have generated clusters to show how weather affects the number of ridership and the total amount of minute ridden. For our clustering algorithm, we decided to use K-Means. By having our weather data separated into different groups, we thought that K-Means was a good algorithm for the clustering. Unfortunately, after the implementation of K-Mean, we realized that K-Mean doesn't cluster the data in such a way that would be meaningful. Due to time constraints, we did not have enough time to try the other clustering algorithms, but we think that DBSCAN could have cluster the data the way we intended it to.

Decision Trees ***Daniela***

In order to classify a particular day as a day of good or bad ridership--and in turn profit--decision trees were used. Since this is solely based on the total length of time riding per day, the tree depth will be quite small. Despite this, decision trees are a great visual representation of large datasets such as this one. Also, as a base value to compare to, the mean of the total minutes per day would help differentiate good and bad days by creating a clear division. This could help keep track of certain days that are consistently good days and allow reliable predictions to be inferred based on the results.

Random Forests ***Avi***

As one of the 3 algorithms we intended to use, random forests was to be used to predict the total ridership on a given day, thus helping predict the total amount of money taken in by the system. If a user input data for a whole season, a prediction could be generated for that time span. Random forests use a collection of decision trees to generate a prediction. Each tree generated will be different due to the fact that only a selection of the features are chosen for each tree rather than every one. This prevents a dominating feature from being at the root of every tree. The algorithm produces the trees, and when data in input to the tree, the average prediction is chosen. The random forests classifier used in this project came from the scikit learn ensemble library as the "RandomForestClassifier." This specific implementation of the random forests algorithm was chosen due to our familiarity with it; we had seen an example in lecture and had source code that detailed its usage.

Data Preprocessing and Change of Scope

Data preprocessing formed the bulk of our project. When we downloaded the data and looked through it, we thought that there was plenty of data for our machine learning and that

it was ready for use. Unfortunately, we realized that the data would need to be preprocessed for our use. This section will detail the initial dataset, our data preprocessing, the difficulties faced in this phase, and our change of scope in the project.

The dataset was found on the Bay Area Bikeshare program website. It downloads as 3 separate folders, one for each year the program has been around. As mentioned above, the dataset looks ready to go on first inspection. The data in each folder was broken into 3 separate files; weather data, trip data, and station data.

The weather data consisted of the weather conditions for a specific day for 5 separate cities based on ZIP code. Each object had 22 features which included the date, ZIP code, and weather conditions. Between these 22 features we decided to keep 7 features: date, mean temperature, mean humidity, precipitation index, cloud cover, events, and zipcode. We knew our goal was to use classification algorithms, so we needed distinct categories for each feature. Our preprocessing was done through python and, as will be mentioned throughout this paper, our group did not have experience with pandas/dataframe handling in the language. Our attempts to stratify the data were frustrating, to say the least. Ultimately we prevailed. The temperature, humidity, precipitation index, and cloud cover datasets were stratified and converted into categories ranging from 0-10. The dates were converted to datetime objects.

For the weather data, precipitation index and cloud cover were used to extract the information into 5 distinct groups. For cloud cover, the data range from 0-8, this value is then categorized into 5 groups: Clear: '0-1', Little Cloudy: '2', Cloudy: '3-4', Very Cloudy: '5-6', and Overcast: '4'. For the precipitation index, it was more complicated than the cloud cover. The dataset for precipitation index included values of “T”, which signify a value of less than 0.01. However, because the dataset uses T in the column, the numbers were treated like string instead of integers. To overcome this problem, we gave T an arbitrary small value and converted the column to floats. This allow us to categorize them into the following 5 groups: None: '0 < 0.20', Light: '0.20 < 0.50', Medium: '0.50 < 1', Pouring: '1 < 1.99', Heavy: '1.99 < 4’

After inspecting the station data, we determined that it would be easier to drop all station data from our datasets. Station names changed between years, and there were various dependencies between lists. Processing this data would have added to the complexity of our project without much gain. This was one factor in the decision to treat the bay area as a single unit rather than breaking it up into the zip codes included--Palo Alto, SF, Mountain View, Redwood City, San Jose. All station data was subsequently dropped.

Of the 3 datasets included, the trip data was by far the largest. There were almost 1 million instances in the dataset, 300,000 for each year. Included in the trip data was the start and end stations/terminals, total seconds the bike was used, date, customer/subscriber label, and zipcode of the bike member. It was only after starting the project and investing effort that we realized there was no identifier tagging rides to individuals. After discussing our game plan, we determined the best course of action would be to lump all the rides together based on date. This would allow use to create new entries in the table. As a consequence of this action, our dataset was reduced from 1 million instances to 1099. Again, we faced challenges with

lack of familiarity of python pandas. Adding up the number of minutes passengers had ridden and then lumping the data by date took considerable time to figure out. This reduction brought our dataset down from the recommended size for the project to one that was below recommendation. We wanted to keep our initial project idea as close to our proposal as possible so we decided to proceed with the smaller dataset.

After preprocessing our data we were left with a table that included dates, precipitation index, cloud cover, temperature, humidity, rain, and day of the week. Our project goal was changed accordingly. We decided to continue using clustering to determine optimal weather conditions, but instead of 1 million entries we would instead use the 1100 produced by our data preprocessing. Random forests would also be used, but instead of predicting ridership for a given day, the classifier would be used to predict for a given weekday. Note the subtle difference, we won't target based on calendar day, but on weekday exclusively. Decision trees remained, but also with a reduced dataset. Our plan was to compare the result of using decision trees with the results of random forests.

Machine Learning

K-Means Clustering

The K-Means algorithm we are using is from the scikit learn library. To use the algorithm, we imported into our notebook using this command:

```
from sklearn.cluster import KMeans
```

Before using the algorithm, we choose from our dataset the columns we want to represent in each cluster. For our x_set, we want to store all the relevant information regarding the temperature, humidity, precipitation, cloud cover, number of rides, and total minutes. In the y_set, we have the category for our weather data in which it ranks whether a day is cloudy or not and if it was raining or not. The entries in this set are scaled from 0-4 from low to high. After having our x and y set we can implement K-Means:

```
model = KMeans(n_clusters=5)
model.fit(x)
```

We chose 5 clusters because we have 5 different categorized weather conditions. Since K-Means is an unsupervised, we want to see how it handles the labeling so we use this function:

```
model.labels_
```

Decision Trees Implementation

Once the preprocessing stage was completed, it was time to implement the decision trees algorithm available through scikit learn. To use this it had to be imported in python using the following commands:

```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
```

Before anything could be done, the dataset "RidesMinutes.csv" had to be imported as well, and also import "test_train_split" so that it could be used for the tree.

```
from sklearn.model_selection import train_test_split
```

Next, now that a method to split the test and training dataset has been imported, the data has to be split and the tree is built using the training data.

```
X_train, X_test, y_train, y_test = train_test_split(
    ride_min.TotalMinutes, ride_min.Result, stratify=ride_min.Result,
random_state=0)
#build the tree using the training data
tree = DecisionTreeClassifier()
X_train=X_train.reshape(-1,1)
X_test=X_test.reshape(-1,1)
tree.fit(X_train, y_train)
```

The reason for the reshape is that it would not allow a one dimensional array as an input due to deprecation. X_train.reshape(-1,1) is used when only utilizing one feature, as opposed to one sample.

Random Forest Implementation

After preprocessing the data, the random forests algorithm was ready to be used. Again, the algorithm came from the scikit learn library, and was imported into our program using the following command in python:

```
from sklearn.ensemble import RandomForestClassifier
```

When instantiating the tree, 2 arguments were passed in: n_estimators and random state. N_estimators, or the number of trees used in the forest, was chosen as 7 after some trial and error. This number produced the highest combined value for accuracy on the test data and accuracy on the training data. The following code was used to instantiate the random forest:

```
forest = RandomForestClassifier(n_estimators=7, random_state=2)
```

Data passed into the classifier was set up with a 33/66% split of the data, 33% going towards testing and 66% going towards training. The split was done randomly using the train_test_split form the sklearn.model_selection library with the following line of code:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=25)
```

Analysis

Before we analyze the result of K-Means clustering, we have to see how our data would look if we were to cluster them into the results we expected. To do this, we created a scatter plot to plot the number of rides and the total minutes ridden with our weather data of cloudy description and precipitation description respectfully.

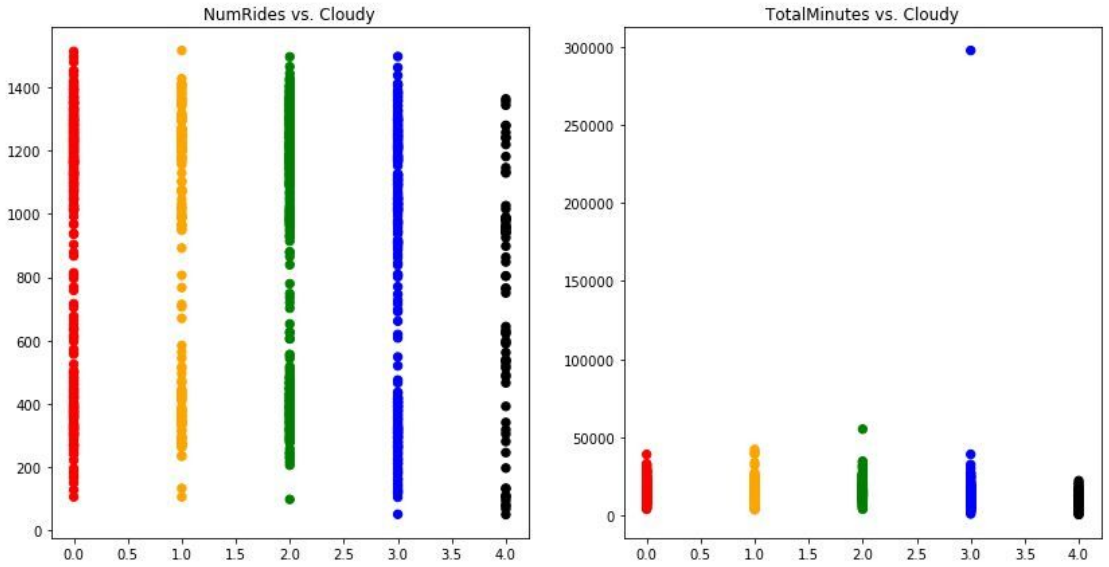

Figure 1. Results expected from cloudy description from numbers of rides and total minute ridden.

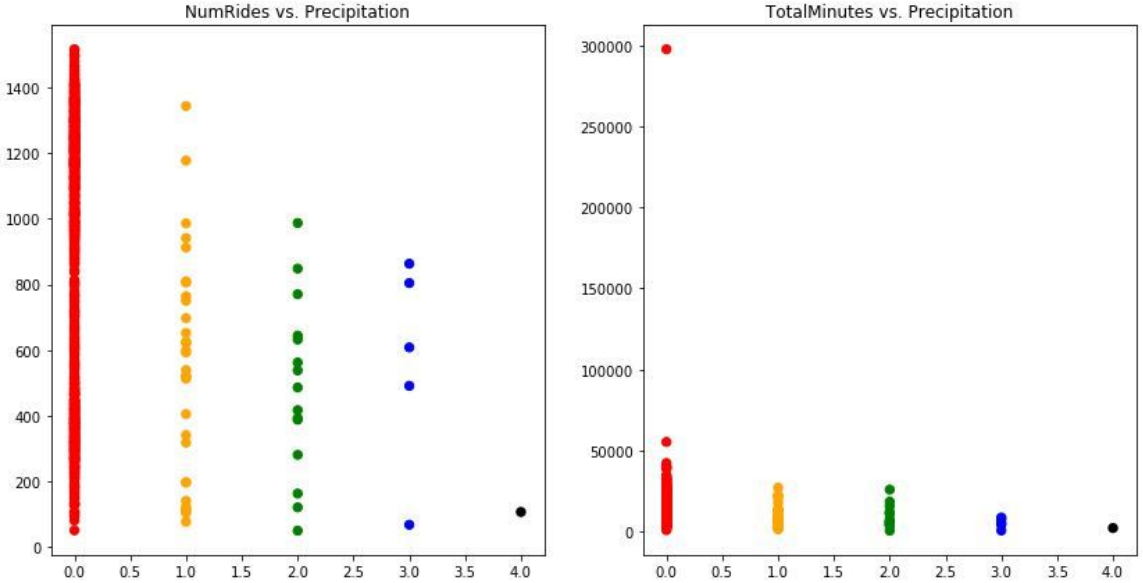

Figure 2. Results expected from cloudy description from numbers of rides and total minute ridden.

After visualization of cluster, we applied the K-Means algorithm to see the clusters generated by K-Means.

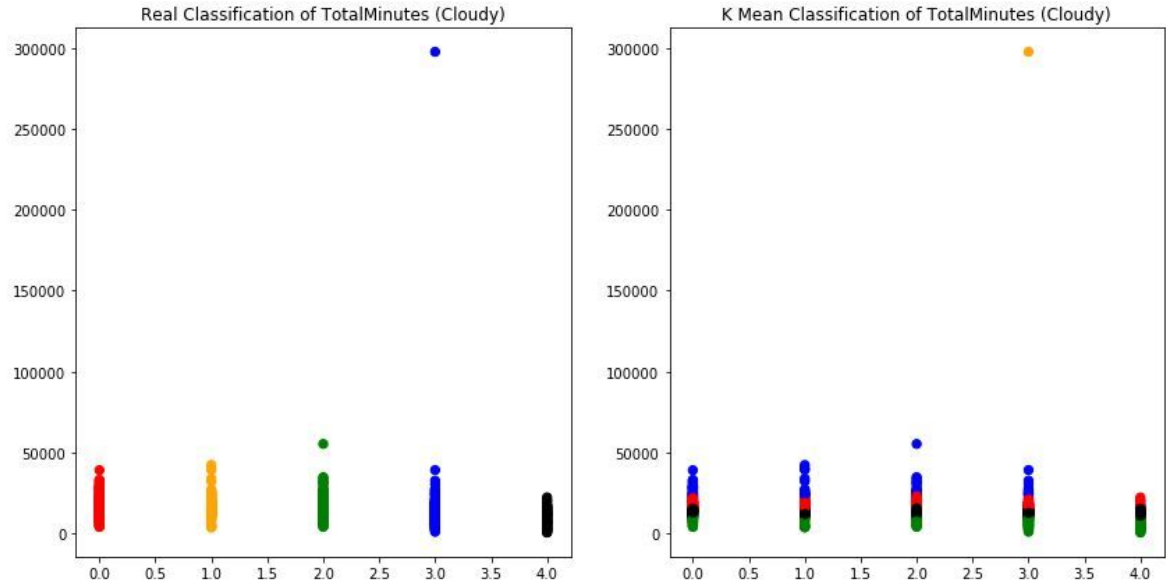

Figure 3. Results compared from our real classification to K-Mean's classification for Cloudy description.

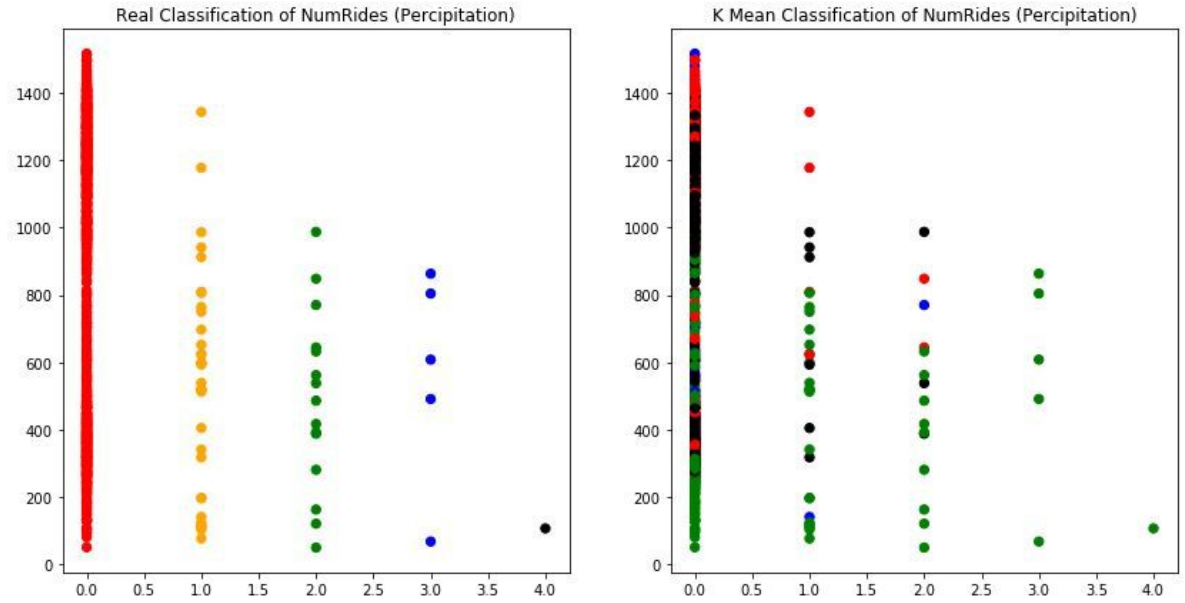

Figure 4. Results compared from our real classification to K-Mean's classification for Precipitation description.

After comparing the results of the two, we notice the flaw of using K-Means. The centroid generated by the K-Means are only clustering the data in a row fashion, therefore all the data near the bottom is cluster as one group and the cluster are rank hieratically. While this may be useful in other cases our clusters need to be column based and K-Means does not perform well in our case.

The decision trees algorithm seemed to result in high accuracies in both training and testing data. The accuracies were 100% and 99.6%, respectively.

```
#Print accuracy of the model
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.996
```

Figure 5: Numerical results of Decision Trees on "RidesMinutes.csv".

The perfect accuracy on the training data is to be expected though, so the next thing to check is the test set. The max_depth was left at 4, other options were tested but yielded the same result.

```
print("Accuracy on training set: {:.3f}".format(tree2.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree2.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.996
```

Figure 6: Numerical results of Decision Trees using depth=4.

Possible reasons behind the two outputs for accuracy being the same could be attributed to the fact that only one feature was used to determine whether a day was good or bad. This is because the main focus was how many minutes a day people were riding in total, and if it was above the average of all the days, it was considered a good day.

The random forest classifier output scored relatively low on accuracy for training and test data, especially when compared to examples that we'd seen prior. The accuracy on the training set was 81%, while the accuracy on the test set was 36%. There are many reasons why these numbers are so low. First of all, the dataset used to train the algorithm is small. This is a result of the preprocessing and change of scope for our project. We went from a dataset of 1 million items to 1100. Another factor may have been a mismatch in data. Although we did our best to ensure there were no issues with this problem, something may have mismatched the data. Yet another factor may have been how we preprocessed the data. Most of the fields, be they temparature, humidity, or ridership were split into categories ranging from 0-10. These categories may have been chosen incorrectly, or may have hidden much of the data. This would have directly resulted in less accuracy in the random forests decision trees.

After the algorithm was run, a look at the feature importance chart indicated which features had the most effect on determining classes. As can be seen in the chart below, cloud cover and day of the week were most significant with precipitation being least significant. This is counter to what we expected as our 'ground truth.' One would expect rain to directly cause less ridership, but this doesn't seem to be the case. On further reflection, when it does rain in the bay area outside of winter, it usually happens at night. Rain is sometimes forecast during the day but rarely happens. Using time series data to analyze when the rain occurred would probably add to the significance of the feature.

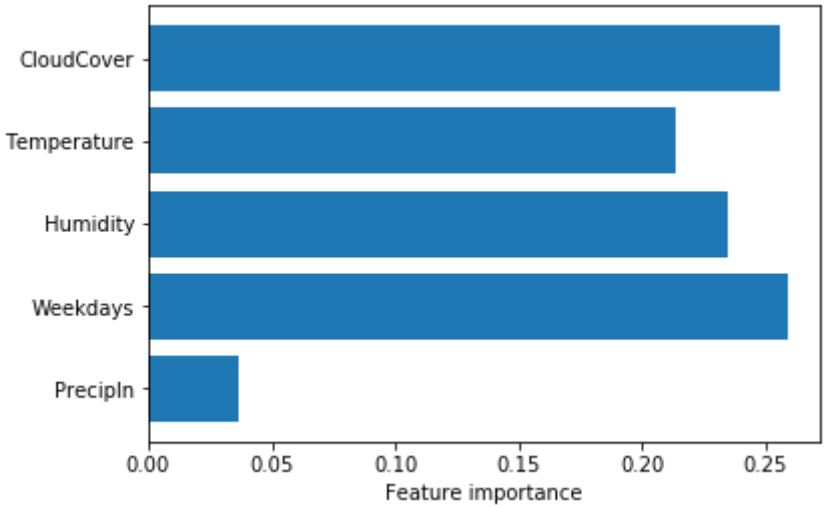

Figure 7. Feature importance for Random Forest Machine Learning Algorithm

Although we did graph the trees produced by the random forests, they were too large to include in this document. By altering the default parameters of the random forest algorithm to produce fewer leaves, less depth, or fewer nodes this would have changed, but doing so resulted in a reduction of accuracy on both training and test data. The default values were thus kept and the trees produced were allowed to 'grow' as needed.

Discussion & Conclusions

In our team's initial planning phases, we decided which data sets and topic we would like to work on. We chose to find the correlation between biking and good weather, while also seeing how it would affect and possibly predict bikeshare profit. During one of the meetings we began to decide on the algorithms to be used--decision trees, random forests and clustering. Also, we brought up that the initial data will not be overwritten, only reclassified so that it would better suit the algorithms to be used. Additionally the duration of rides in minutes and total minutes per day had to be added to the existing data, since the initial duration of a ride was given in seconds. On April 27th, each member was assigned a task to reclassify or create categories to be used by the algorithms. Later on May 4th we checked in and decided on final preparations, we gave each team member an algorithm to focus on. Clustering was to be used for specific days or types of weather; for example to cluster mondays. With bike share data for a specific day along with the weather data, one could infer the number of rides. Decision trees to put ridership into distinct ranges based on the amount of time in order to determine good days. Meanwhile, random forests was to be used to predict the number of rides on a given day of the week and weather through multiple subtopics.

One difficulty we came across near the end of our project was between preprocessing and implementing the algorithms. This was mainly due to the way we manipulated the data during preprocessing cut down the amount of instances used in the final stages. Also, For decision trees, the graph depiction kept resulting in an error when it tried to call graphviz.

This was even with MacPorts installed and graphviz was installed through “brew install graphviz”.

Overall, the process was lengthy and required additional research to complete which did amplify our knowledge on the subject. Our team worked well together and communicated efficiently, leading to a great project environment. Through these exercises, it has become evident that such algorithms are instrumental in handling large datasets such as these, and preprocessing may even take longer than implementing the algorithms themselves.