Becca Dura
CS167 Project A

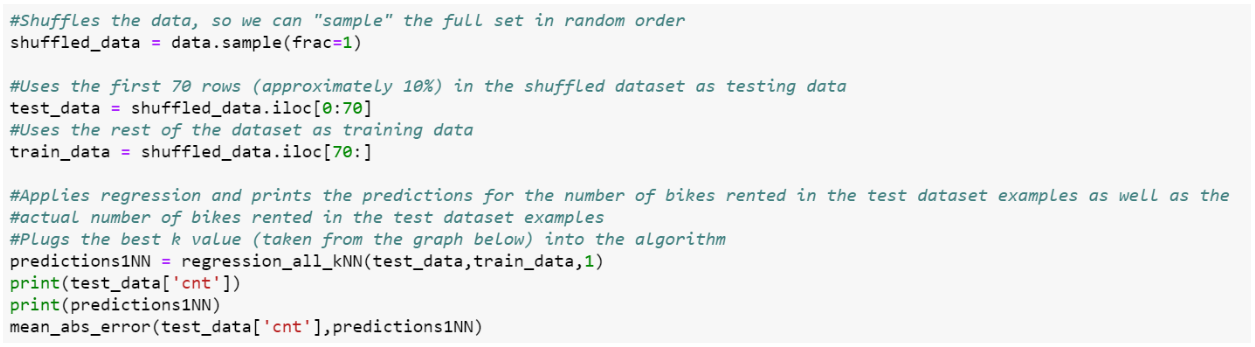
Project A Write-Up

Problem & Data

The goal of this program is to predict how much business a bike sharing company will get, or how many bikes will be rented (cnt), by implementing the k-Nearest Neighbor algorithm. To do this, I set cnt as my target variable and picked out 5 of the most relevant features to use as predictor variables- weathersit, atempc, mnth, holiday, and weekday. I chose weathersit and atempc because they tell us what the weather was like outside for each example. The weather is important because people won't necessarily be wanting to rent a bike if the weather is poor or the temperature is outside their comfort level. I chose atempc instead of tempc because I felt as if the feel of the temperature included more variables than tempc. It not only takes into consideration the actual temperature, but also accounts for the humidity level and the winds. On top of the weather features, I also decided to use some features surrounding the date. The mnth feature seemed important because the weather tends to vary based off the time of year, which I believe would affect bike usage. I chose month over season because weather can vary a lot within one season in some places. I also chose to use holiday and weekday as predictor features because I believe bike usage would be lower on holidays and on the days when people are at work (weekdays). In other words, I chose to use the description of the weather, what the temperature feels like, the month, whether or not it is a holiday, and what day of the week it is to predict how many bikes were rented each day.

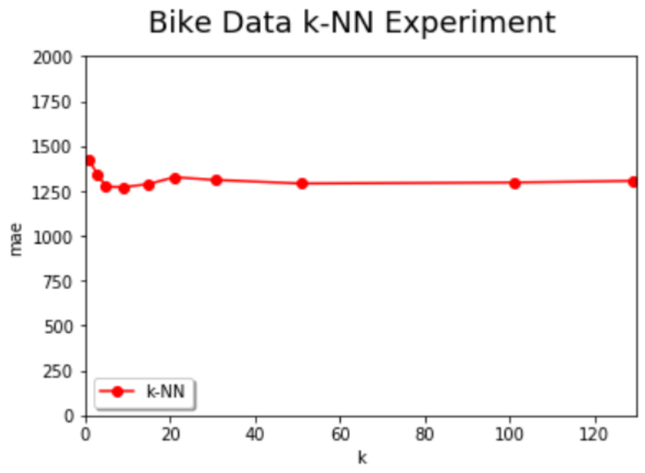
Experiments, Results, & Interpretations

I was successfully able to implement the k-Nearest-Neighbor algorithm on the bikeshare_projectA.csv data set, and the output is after the following block of code in each of my code files:



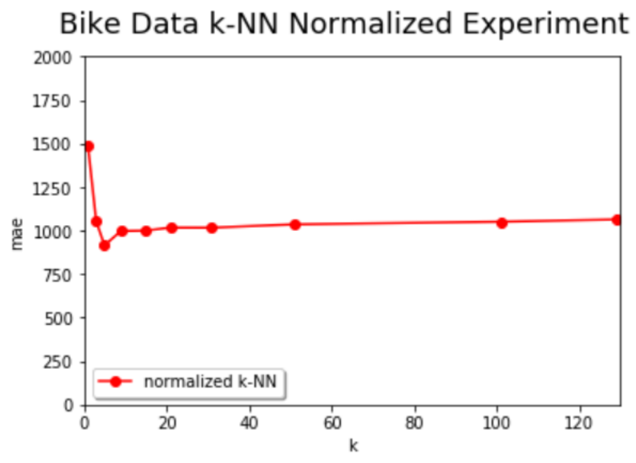
```
#Shuffles the data, so we can "sample" the full set in random order
shuffled_data = data.sample(frac=1)

#Uses the first 70 rows (approximately 10%) in the shuffled dataset as testing data
test_data = shuffled_data.iloc[0:70]
#Uses the rest of the dataset as training data
train_data = shuffled_data.iloc[70:]

#Applies regression and prints the predictions for the number of bikes rented in the test dataset examples as well as the
#actual number of bikes rented in the test dataset examples
#Plugs the best k value (taken from the graph below) into the algorithm
predictions1NN = regression_all_kNN(test_data,train_data,1)
print(test_data['cnt'])
print(predictions1NN)
mean_abs_error(test_data['cnt'],predictions1NN)
```

I tested my algorithm by performing cross-validation, which means I had a separate training and test set. I did that to ensure that my algorithm did not just predict the examples used to create the model well but did a good job of predicting how many bikes were rented on any given day. Since there were 730 examples, I randomly placed 70 (approximately 10%) of those in the test set. The rest went in the training set to help me create a better algorithm to predict the numbers of bikes rented. The metric I used to test my algorithm was the mean absolute error, which is the average distance between the actual and the predicted number of bikes rented each day. I chose this metric over mean squared error because it does not weight variables with greater ranges of results more heavily than others, like mean squared error does.



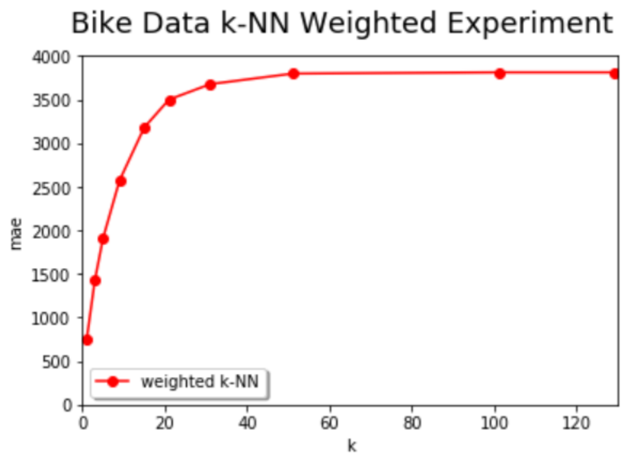
I tested my algorithm with a wide variety of k's, the lowest k being 1 and the highest k being 129. The graph above shows how the mean absolute error changes as k changes. As the graph shows, the lowest mean absolute error occurred with a k value of 9. Since a lower mean absolute error means the model is better at making predictions, I used 9 as my k value for the k-Nearest-Neighbor algorithm. The output from running the algorithm with 9 as my k value is in my code (location is explained at the beginning of this section). The graph above also shows that a very small k value (1 or 3) creates a model that is poor at predicting the number of bikes rented, and that k values after 9 predict the model almost the same, because they all have a very similar mean absolute error. That is because lower k values do not include enough examples to make a strong prediction. However, greater k values include more examples in the calculation of the mean absolute error, so the model's predictions are better and do not tend to vary as much. This is because examples that do not follow the model well or examples with outlier variables do not have as heavy of a weight in the predictions.

The normalized version of the model performs much better. To create this version of the model, I normalized all my predictor variables using the Z-score method. That ensured that the predictor variables were all on a smaller scale, so predictor variables with greater ranges of results were not weighted as heavily. It did this by calculating how many standard deviations a predictor variable was from its mean. Aside from normalizing the predictor variables, I kept everything else the same in my algorithm, so I was able to more easily compare it to the non-normalized version. I also used mean absolute error as my metric to test the algorithm again.



As shown above, the mean absolute error follows the same pattern it did for the non-normalized version. The k value producing the lowest mean absolute error is 5, which is close to the lowest k value from the non-normalized version (9). The normalized version also shows the mean absolute error being high for the lowest values of k and remaining steady for values of k after 9. The only major difference between the graphs for the normalized and non-normalized versions of the model is that the normalized version has much lower mean absolute errors for each value of k than the non-normalized version. For example, the lowest mean absolute error for the non-normalized version is approximately 1,250, but the lowest mean absolute error for the normalized version was approximately 900. That means that the normalized version of the model is better at making predictions for the number of bikes rented in a day, which makes sense considering that the predictor variables are all on a similar scale now.

The weighted version of the model also performs much better than the original, non-weighted version. In this version, I weighted the average of the k nearest neighbors based on how close they were instead of just taking the average of the k nearest neighbors. Aside from applying this weighting, I kept the rest of my algorithm the same to better compare its results with the non-weighted version. I used the mean absolute error to examine how well my model made predictions again.



The graph above that shows the relationship between k and the mean absolute error for the weighted version looks vastly different than the graphs for the original and the normalized versions. That is because I wanted to predict the target variable with the most weight. In other words, I summed the weights for each of the possible values the target column could take on, or the number of bikes that could have been rented, and then I picked the largest one. Since I was only supposed to pick one, the best k value for this model is 1. That is shown in the graph above. A k value of 1 results in a mean absolute error of approximately 750. After that, the mean absolute error increases at a high rate until a k value of approximately 20, at which point the mean absolute error is about 3,500. After that k value, the mean absolute error is steady for the most part. In other words, a k value of 1 always produces the best model for weighted k-Nearest-Neighbor algorithms. Overall, since the lowest mean absolute error for the weighted version is approximately 750, which is significantly lower than it was for the original and the normalized versions, the weighted k-Nearest Neighbor algorithm produces the model that is the best at predicting the number of bikes rented each day from the bike sharing company.