Analyzing Data Generated by City Bicycle Rental Systems

Charles Schumacher
CSC 478-801
3/15/18

Introduction

In an era where ride sharing and environmental awareness are both increasing trends, bicycle rental systems in metropolitan areas have also begun to appear in growing numbers. The benefits of these systems are numerous: health benefits via exercise for users, cost effective form of alternate transportation that contributes to a reduction in traffic congestion, no chemicals or gases emitted which have a negative impact on the environment, etc. These systems have a net positive impact for both those who chose to partake, the cities who host them, and the environment that we all share

Not only are the bike rental systems positive installations, but they provide a unique and interesting pulse on the host cities and inhabitants. Unlike traditional shared transportation methods such as trains and buses, these systems provide timestamped location data for both ends of the journey. Because these systems operate on a credit basis (either through a pre-paid account or through a credit card swipe) they also know who is renting the bicycles, when they're renting, and how often they're renting.

The numerous touch points that are produced by these systems invites extensive analysis. Due to the nature of bicycle transportation and increased exposure to the literal environment, there are trends that may easily be captured by coupling the rental data points with weather data and seasonal status. Individuals who often choose a bicycle as their preferred means of transportation to work may find alternate means of transportation if weather is inclimate. On the flip side of that, if the weather is particularly nice and it is a weekend (when many individuals have a higher proportion of free time) there may be spikes in bicycle rentals because infrequent users may be more inclined to partake when conditions are optimal. In addition to the literal environment (ex. weather conditions) having a measurable impact on bike rental activity, there are socio-environmental factors which may potentially influence usage. For example, in the city of Chicago there are numerous city-sponsored events and festivals throughout the summer which draw massive crowds the range from hundreds of thousands of additional people downtown (ex Lollapalooza) to upwards of one million people gathered (ex. Taste of Chicago). In such scenarios, it is often favorable to use a bike rental which can allow for flexible navigation and potentially minimize the travel-time increase to usual routes. The potentially heightened propensity for alternate transportation such as a bicycle may be reflected in the rental counts for days during such periods.

This paper sets out to be an initial exploration into the bike rental system data that is publicly available and to identify foundational elements of the data publicly available.

Overview of Data Analyzed

The data analyzed for this paper is provided through the University of California-Irvine Center for Machine Learning and Intelligent Systems. The dataset contains 17,389 instances of bike rental data

from 2011-2012 in the Washington D.C. metro area. The full dataset contains features which describe the following:

- instant (the record index)
- dteday (date)
- season (1 = spring, 2 = summer, 3 = fall, 4 = winter)
- yr (0 = 2011, 1 = 2012)
- mnth (1 through 12)
- hr (1 through 24)
- holiday (if the day is a holiday or not)
- weekday (day of the week)
- workingday (0 = not a working day, 1 = working day)
- weathersit (1 = clear, few clouds, partly cloudy; 2 = mist+cloudy, mist+broken clouds, mist; 3 = light snow, light rain + thunderstorm + scattered clouds, light rain + scattered clouds; 4 = heavy rain + ice pallets + thunderstorm + mist, snow + fog
- temp (min-max normalized temp in Celsius)
- atemp (normalized feeling temperature in Celsius)
- hum (humidity – values are normalized with 100 being max)
- windspeed (normalized wind speed, values divided by 67)
- casual (count of casual users)
- registered (count of registered users)
- cnt (count of total rental bikes including both casual and registered

There is a great amount of potential exploration to be done with all of the above data points. For this paper I am only concerned with the season, year, month, hour, if it is a working day, the weather status, the normalized feeling temperature, humidity, windspeed, and count of total bike rentals.

Discussion of Methodology

Outlier Removal

Utilizing the subset of features listed, this paper describes an exploration into rental behaviors based on certain conditions (if it is a workday or not, and if there is inclimate weather, time of day, etc) to understand the predictive power of these conditions when combined with the bike rental system touchpoints provided. The problem of predicting bike rental counts based on limited environmental factors can be improved by eliminating outlier samples. The difficult with identifying the true outliers arises when comparing the different rental volume trends within a single feature. For example, the outlaying data points arise at different hours for working days when compared to non-working days, and the two distributions already have different behaviors (Figure 1 and Figure 2). The IQR (inter-quartile range) of a box plot is the area within the box and represents the “middle 50%”. The lines coming from the box in each direction represent 1.5*the IQR which is often used as a measure of describing normal data points that exist outside of the IQR. Beyond where those lines end there are sometimes single data points, which are sometimes (depending on the dataset) considered outliers.

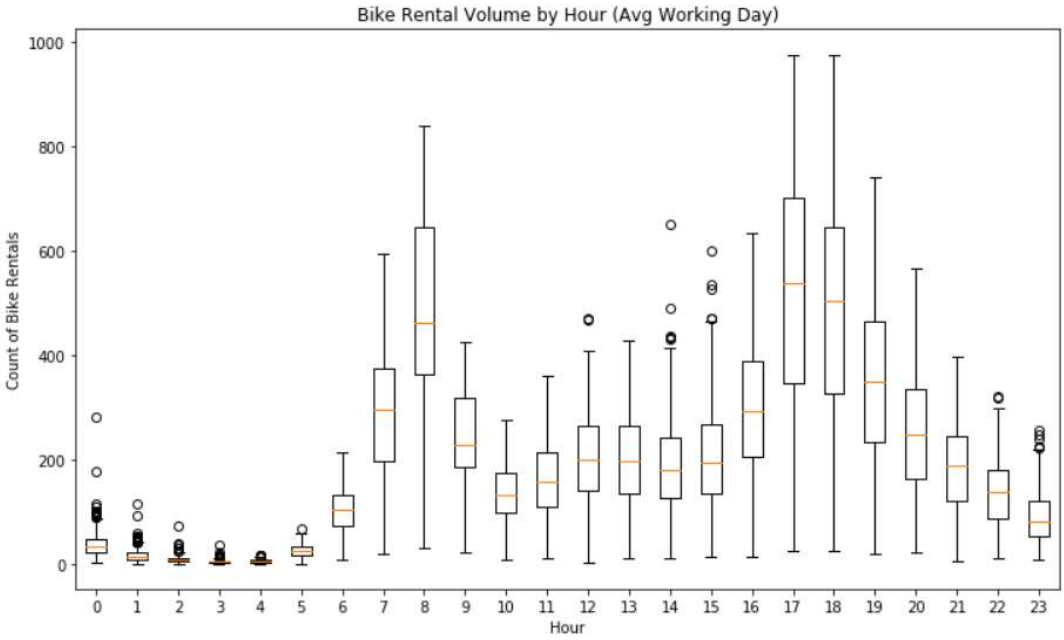
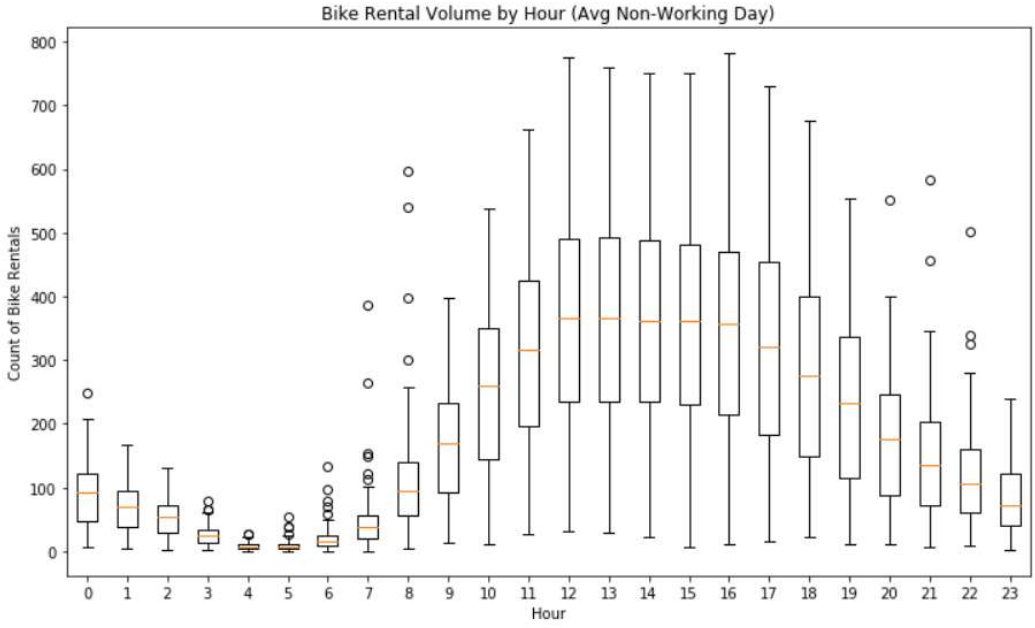


Figure 1: Box plot of rental volume by hour for average working day



E Figure 2: Box plot of rental volume by hour for average non-working day

From visual inspection it one can see that the outlaying values on a working day tend to appear between 10pm-5am and midday between noon and 3pm. On a non-working day we see an different trend where the rental volume increases and maintains a high value throughout the afternoon (where the working days will typically see a drop off in rental volume). For non-working days the bulk of outliers appear from 3am-8am and a few from 8pm-10pm.

When exploring other dimensions of the data such as weather status, there is somewhat of an expected drop-off in overall volume for rainy days (Figure 3). Comparing this to Figure 4 which depicts the hourly shape of our data for non-rainy days there is a much higher number of outliers (note that for all explorations in this paper a rainy day is defined as weathersit >= 3, and a non-rainy day is defined as weathersit <= 2).

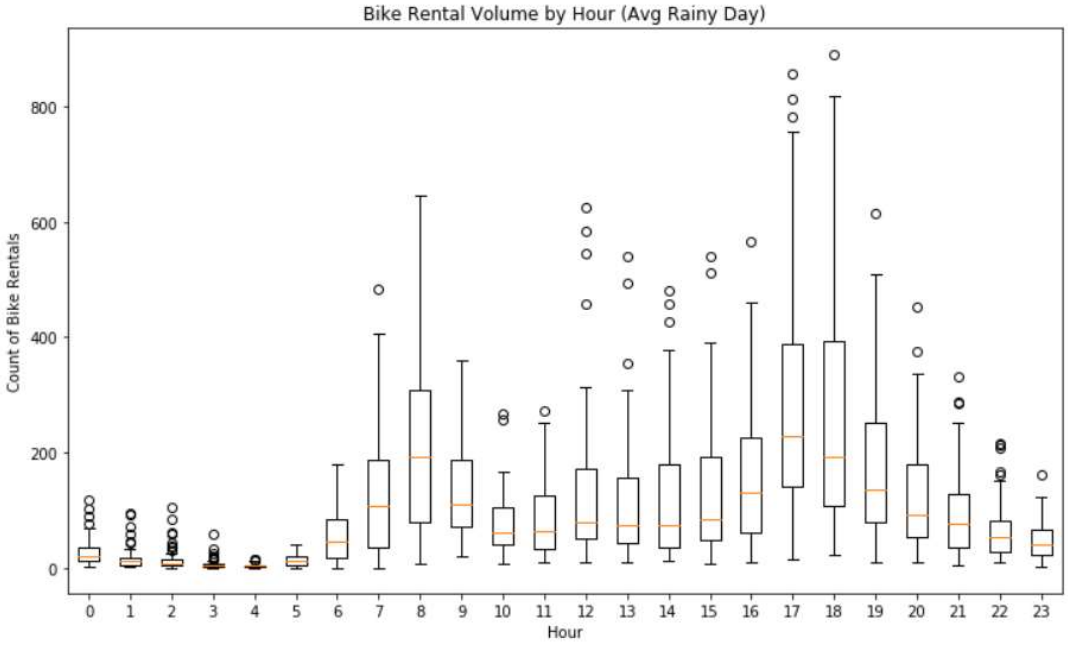


Figure 3: Box plot of bike rental volume by hour for average rainy day

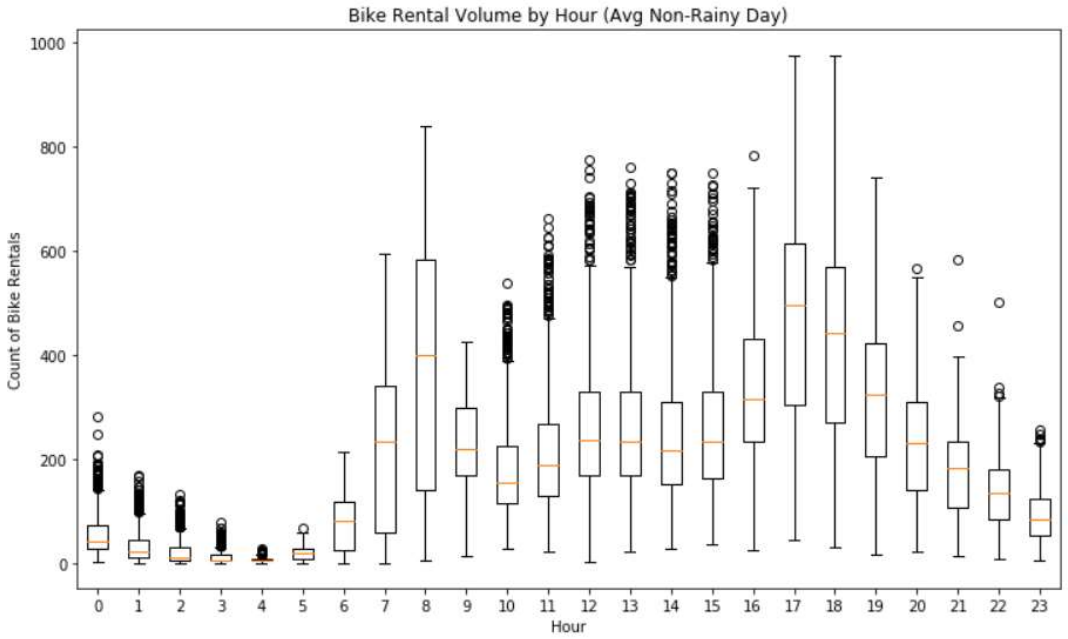


Figure 4: Box plot of bike rental volume by hour for average non-rainy day

To properly capture the true underlaying behavior, an effort was made to remove outliers in the most effective way possible. Two approaches were tested. The first was utilization of the LocalOutlierFactor() from the Sklearn library. The underlaying algorithms in this approach is clustering and assigning outlier vs non-outlier labels based on the number of nearest neighbors. The second approach for this was to iterate over each combination of two features and remove outliers that were outside of a certain multiple of the IQR (range of 0.0*IQR through 2.5*IQR were tested). The Sklearn method consistently returned 90% of the original data points as non-outliers and the remaining 10% as outliers, regardless of the number of neighbors used and model parameters tested. The outlier lists generated by the feature-combination filtering were dropped which deviated from the 90/10 non-outlayer/outlayer split in order to compare their predictive abilities more closely.

Predicting bike rental volume with environmental data

After generating two separate lists of outlaying points from the methods described above, the outlaying data points were filtered from separate copies of the original data in order to gauge the impact on predictive ability of learning models tested. Linear Regression and Random Forest Regression were both utilized and tested with a variety of model parameters, as well as with a scaled version of the data set and not scaled.

Linear Regression uses the list of features for a given data point to attribute a continuous value to that particular data point, and then given the optimal parameters for producing that continuous-valued label on the training set it will try to use the same to predict the continuous-label for the testing set. For Random Forest Regression, the algorithm will try to divide the data points based on certain features and how they reduce entropy (the measure of randomness in a dataset, it will try to divide the datapoints in such a way that there are more similarly datapoints in the resulting splits until there are only datapoints of a certain kind in the end nodes (or “leaves”) of the trees. The primary difference for this is

Predicting environmental attributes with bike rental data

For another exploration about the ability to predict environmental factors based on bike rental volume trends, I modified the original dataset to drop all of the weather columns since those columns represent one of the most easily measurable environmental behaviors. Scaled and not scaled versions of the original dataset were utilized with four different classification algorithms: Logistic Regression, Support Vector Machine, K-Nearest Neighbors, and Random Forest. Each of these models’ was tuned in order to try and find the best fit for the data in order to predict if 1. It is a rainy day or not, and 2. If it it a working day or not. Note that for the rainy day predictions, a day was given the label of “rainy” if its weathersit >=3, and given the label of “not-rainy” if its weathersit <= 2.

Logistic Regression is an algorithm tries to find a logistic-curve shaped hyperplane that will divide the points in the multi-dimensional space described by its features, in such a way that it can correctly apply labels to classes in the dataset. Support Vector Machines are similar in the sense that they try to identify the optimal hyperplane to divide the points in a such a way that they can apply correct class labels, but the primary difference is that it is not a hyplerplane of a logistic curve and that it iterates throughout all of the points in the dataset to try to identify the most optimal curve for splitting the data (which makes it more computationally expensive). K-Nearest Neighbors is an algorithm that plots all of the datapoints in your dataset in a multi-dimensional space which is describing by the features, and then applies class lables based on the “votes” of the n-nearest neighbors (the “n” being a parameter you can specify to

control how many neighbors are influencing the labeling) using various distance measures. For this paper we solely explored Eucliean distance which is essentially a measure of the distance in a straight line from one point to another. Random Forest for classification is similar to the regression version described above in the sense that it continually splits the data based on certain values within the data, with the goal of reducing entropy (aka getting increasingly more "pure" nodes that result from the splits) and ultimately finding the best way to apply class labels to the data points. The most obvious difference between using Random Forest for classification and regression is that with classification you are applying discrete class labels instead of continuous labels to the data.

Results

Predicting bike rental volume with environmental data

Because this is a regression problem at its core, Linear Regression was used to quickly evaluate numerous data frames with different outliers removed based on the two techniques described above. The overall RMSE for the original unfiltered dataset was 142.3, for the dataset with filtered values based on Sklearn's LocalOutlierFactor() was 141.7, and for the dataset with filtered values based on the best feature-combination was 98.9. RMSE describes the "Root Mean Square Error", which is a measure of the average distance from your predicted labels relative to the actual labels in the data set, meaning that a smaller RMSE is more optimal for our problem because it would mean that we are more accurately predicting the labels. Figure 5 shows a plot with a sample of the first data points predicted.

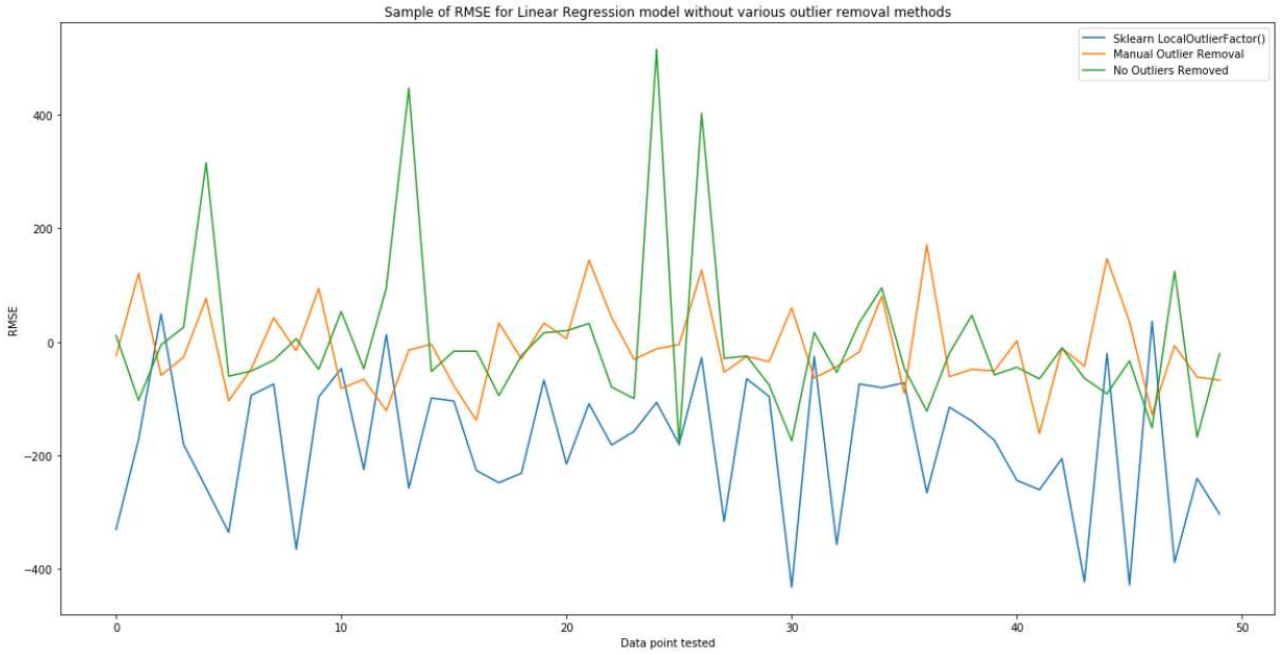


Figure 5: Linear Regression RMSE based on outlier removal methods

From this graph we can see that visually how the Sklearn method does not produce much better results than not filtering the data at all. Using the unfiltered dataset often causes predictions which are much higher than the actual value. The Local Outlier Factor model shown above often produces predictions

which are much lower than the actual value. It is visually apparent from the graph that the manual outlier removal from feature-combinations tested produces a better model while maintaining a comparable amount. Interestingly for this model, the feature combination that produced the most optimal results was workingday feature alone (combinations tested included each individual feature as well).

Using the absolute value of the coefficients of the Linear Regession model I determined that the atemp was the most important feature in predicting the count of bike rentals in a given instance, with a coefficient of 55.15. The mnth feature had the least predictive power of all of the features in the model with a coefficient of only 1.46.

As an additional exploration on this model which was tuned with outliers removed based on workingday alone, Linear Regression and Random Forest Regression were compared. The original Linear Regression model produced an RMSE of 98.9, while the Random Forest Regression produced an RMSE of 40.4, which is a significant improvement. A comparison of their RMSE for the first 50 samples is shown in Figure 6. The Random Forest took much longer to build than Linear Regression took to run, but the results produced were meaningfully better.

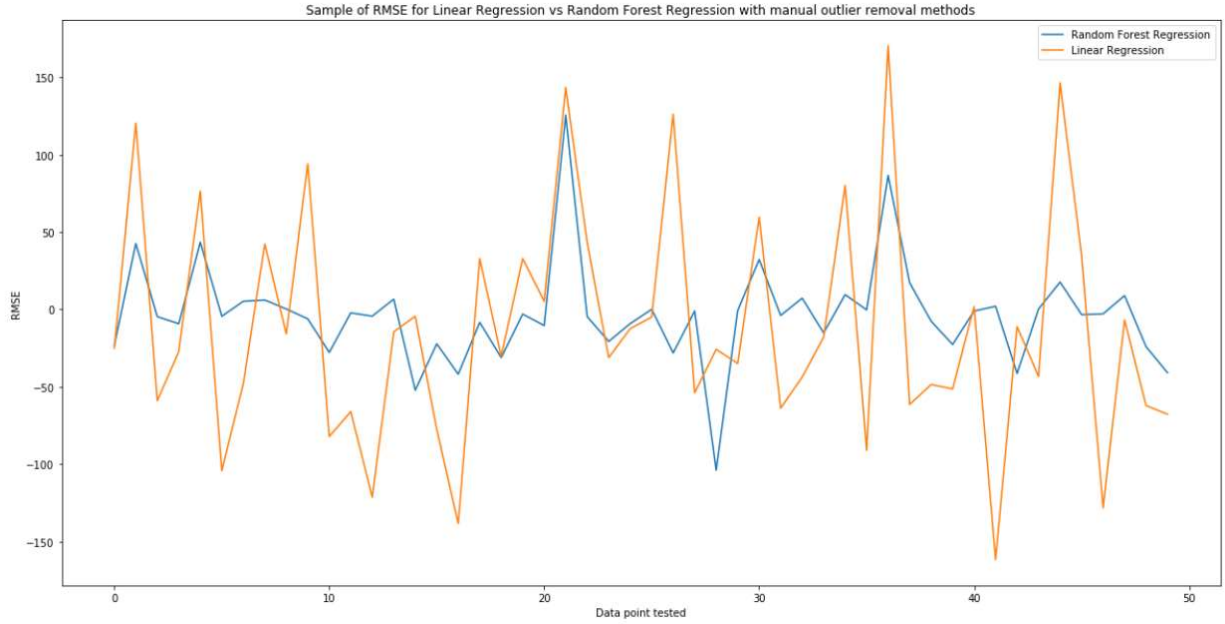


Figure 6: Linear Regression vs Random Forest Regression

Predicting environmental attributes with bike rental data

After dropping the outliers in the dataset based on the feature-combination filter, the same approach was utilized to compare different models in their predictive ability. In both tests, ability to predict if it is rainy or not, and ability to predict if it is a working day or not, four classification algorithms were tested: Logistic Regression, Support Vector Machines, K-Nearest Neighbors, and Random Forests.

For the first exploration into ability to predict if it's a rainy day or not, Random Forest ended up producing one of the strongest and more balanced models with a 35.4% F1 Score and a 90.3% accuracy. The model built 275 trees, required a minimum of 7 samples to split a node, and used balanced class weights respective to each tree built. KNN had a higher accuracy at 92.5% but was more biased towards the larger amount of non-rainy days and had a lower F1 score of 25.7%. The models were tested both with and without weights but I was not able to find an optimal balance to label the rainy and non-rainy days consistently. Logisitic Regression and SVM both produced subpar results. Logistic Regression was only slightly better than a coin flip in labeling correctly at 55% accuracy, and SVM had a high accuracy of 85% but very low F1 score at 9.3% (essentially this was labeled the majority of cases as non-rainy days and since there were more non-rainy days it produces a higher accuracy).

The second exploration into ability to predict if it's a working day or not followed with an identical process to the rainy test prediction test, except that it included the weather columns and count of bike rentals. Once again, the Random Forest model produced the best model, but this time in terms of both F1 Score (90.4%) and accuracy (86.2%). KNN produced a decent model with an F1 score of 83.0% and accuracy of 76.0%, but was still better than SVM and Logistic Regression which both performed poorly relative to the other two.

In a final analysis of ability to distinguish the rainy and non-rainy days based on unsupervised clustering, I used KMeans with two clusters on the dataset from rainy/non-rainy predictions and assigned a label to each instance in order to compare it to the actual values. KMeans is an algorithm which randomly picks points to begin the clusters and sets them to the cluster-centers, and then it continues to pick points from the data set and adds them to clusters, and after each additional point is added it recalculates the center points. So at the beginning of a KMeans algorithm the cluster centers are shifting around more often and greater distances, but towards the end, the cluster centers are moving around much less (in theory) because you are adding points to the clusters which don't shift the overall centers as much relative to all the previous points in the set). The applications of this algorithm are numerous, a few of which being that you can use it to split up your dataset easily into separate groups based on how it "naturally" clusters, and another being that you can compare the actual labels of a class to the "naturally" occurring clusters when you set up a KMeans with the same number of clusters as there are discrete class labels.

Because this is a binary classification problem it was easy to generate confusion matrix metrics which are shown below in Figure 7.

True Positives Rate	True Negatives Rate	Positive Predicted Rate	Negative Predicted Rate	F1 Score	Accuracy
0.504923	0.53663	0.088511	0.924032	0.150619	0.534035

Figure 7: Confusion matrix metrics for KMeans clustering of rainy/non-rainy day labels

The overlap between the natural clusters generated and the actual rainy-day labels was minimal, as seen in the poor results above. Whether a day was correctly classified was slightly better than a coin-flip.

Conclusions

In the explorations described in this paper, Random Forest produced the optimal results for both prediction of number of bike rentals in a day, and in correctly assigning labels of rainy/non-rainy day or workingday/non-working day. While this model takes longer to run, the meaningfully better performance is worth the added computation time for a dataset like this which wouldn't need to be running numerous times throughout the day. In the latter tests, KNN produced better results than both Logistic Regression and SVM but not as good as Random Forest. When tuning the KNN model, a lower number of neighbors consistently produced a better result. Considering this fact about effective KNN models, it's not surprising that KMeans didn't separate the data well, when the labels were effectively assigned to the closest neighbors only and produced poor results when trying to compare larger pools of neighbors.

There is certainly potential to transform the data in ways beyond those explored in this paper and to test additional approaches for identifying the important points. While some features such as weather status may not be predicted with very high accuracy, others such as whether it is a working day or not can be predicted well. The features used in this paper were a subset of the full list which very likely contains additional information which could improve these models. Further exploration into warranted into what else can be predicted with bicycle rental counts over time. In future explorations I would try to add more features beyond weather and time dimensions in order to use the information contained in the bike rental volume for strengthening other models built around a certain city, and to identify what other interesting correlations exist.

Appendix

March 15, 2018

```
In [ ]: import numpy as np
        import pandas as pd
        import sklearn
        from sklearn.linear_model import LinearRegression, LogisticRegression
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.svm import SVC
        from sklearn.pipeline import Pipeline
        from sklearn.preprocessing import StandardScaler, Imputer
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import confusion_matrix, classification_report
        from sklearn.metrics import precision_recall_fscore_support as prfs
        import matplotlib
        import matplotlib.pyplot as plt
        from matplotlib.pyplot import figure
        from itertools import compress, combinations
        %matplotlib inline

In [ ]: df = pd.read_csv("hour.csv")
        df = df[['season', 'yr', 'mnth', 'hr', 'workingday', 'weathersit', 'atemp', 'hum',
        'windspeed', 'cnt']]
```

0.1 Plot of cnt by hour for the average working day

```
In [ ]: def houravgs(df):
            return df.groupby('hr',as_index=False)['cnt'].mean()

In [ ]: def barplothouravg(df, legendnames = [], titlenotes = ''):
            figure(figsize=(12,7))
            if df.shape[1] == 3:
                plt.bar(df.iloc[:,0]-0.125,df.iloc[:,1],width=0.25)
                plt.bar(df.iloc[:,0]+0.125,df.iloc[:,2],width=0.25)
                plt.legend(legendnames,loc='upper left')
            else:
                plt.bar(df.iloc[:,0], df.iloc[:,1])
            plt.xticks(np.arange(0,24))
            plt.xlabel('Hour')
            plt.ylabel('Count of Bike Rentals')
            plt.title('Bike Rental Volume by Hour '+ titlenotes)
            return

In [ ]: df_workday = df[df.workingday==1]
        workdayhravg = houravgs(df_workday)

In [ ]: barplothouravg(workdayhravg, titlenotes='(Avg Working Day)')

In [ ]: def hourlyvalues(df):
            hvlist = []
            for i in range(24):
                hrvals = df[df.hr==i].cnt
                hvlist.append(hrvals)
            return hvlist
```

```
In [ ]: workdayhrvals = hourlyvalues(df_workday)

In [ ]: def boxplothour(hrlist, titlenotes = ''):
            figure(figsize=(12,7))
            plt.boxplot(hrlist)
            plt.xticks(np.arange(1,25),np.arange(0,24))
            plt.xlabel('Hour')
            plt.ylabel('Count of Bike Rentals')
            plt.title('Bike Rental Volume by Hour '+ titlenotes)
            return

In [ ]: boxplothour(workdayhrvals,'(Avg Working Day)')

In [ ]: def meansRemoveOutliers(hrvals):
            fhrlist = []
            for i in range(24):
                q1 = hrvals[i].quantile(.25)
                q3 = hrvals[i].quantile(.75)
                iqr = q3 - q1
                fhrlist.append(hrvals[i][hrvals[i].between(q1-1.5*iqr,q3+1.5*iqr)])
            indices = np.array([])
            hmeans = np.array([])
            for i in range(len(fhrlist)):
                indices = np.append(indices,i)
                hmeans = np.append(hmeans,fhrlist[i].mean())
            df_hmeans = pd.DataFrame(data=hmeans, index = indices, columns = ['cnt'])
            return df_hmeans

In [ ]: fworkdayhravg = meansRemoveOutliers(workdayhrvals)

In [ ]: df_load = pd.concat([workdayhravg,fworkdayhravg],axis=1)
        barplothouravg(df_load,['Raw Hourly Data','Outliers Removed'],'(Avg Working Day)')
```

0.2 Plot of cnt by hour for the average non-working day

```
In [ ]: df_nonworkday = df[df.workingday==0]
        nworkdayhravg = houravgs(df_nonworkday)
        barplothouravg(nworkdayhravg,titlenotes='(Avg Non-Working Day - Raw)')

In [ ]: nworkdayhrvals = hourlyvalues(df_nonworkday)
        boxplothour(nworkdayhrvals,'(Avg Non-Working Day)')

In [ ]: fnworkdayhravg = meansRemoveOutliers(nworkdayhrvals)
        df_load = pd.concat([nworkdayhravg,fnworkdayhravg],axis=1)
        barplothouravg(df_load,['Raw Hourly Data','Outliers Removed'],'(Avg Non-Working Day)')
```

0.3 Plot of cnt by hour for the average rainy day

```
In [ ]: df_rainyday = df[df.weathersit>=3]
        rainydayhravg = houravgs(df_rainyday)
        barplothouravg(rainydayhravg,titlenotes='(Avg Rainy Day - Raw)')

In [ ]: rainydayhrvals = hourlyvalues(df_rainyday)
        boxplothour(rainydayhrvals,'(Avg Rainy Day)')

In [ ]: frainydayhravg = meansRemoveOutliers(rainydayhrvals)
        df_load = pd.concat([rainydayhravg,frainydayhravg],axis=1)
        barplothouravg(df_load,['Raw Hourly Data','Outliers Removed'],'(Avg Rainy Day)')
```

0.4 Plot of cnt by hour for the average non-rainy day

```
In [ ]: df_nonrainyday = df[df.weathersit<=2]
        nrainydayhravg = houravgs(df_nonrainyday)
        barplothouravg(nrainydayhravg,titlenotes='(Avg Non-Rainy Day - Raw)')

In [ ]: nrainydayhrvals = hourlyvalues(df_nonrainyday)
        boxplothour(nrainydayhrvals,'(Avg Non-Rainy Day)')

In [ ]: fnrainydayhravg = meansRemoveOutliers(nrainydayhrvals)
        df_load = pd.concat([nrainydayhravg,fnrainydayhravg],axis=1)
        barplothouravg(df_load,['Raw Hourly Data','Outliers Removed'],'(Avg Non-Rainy Day)')
```

0.5 Outlier removal methods tested

```
In [ ]: def removeOutliers(df,featurelist,thresh=1.5):
            df2 = df.copy()
            feat1 = featurelist[0]
            feat2 = featurelist[1]

            for i in df[feat1].unique():
                for j in df[feat2].unique():
                    vals = df[(df[feat1] == i) & (df[feat2] == j)].cnt
                    q1 = vals.quantile(.25)
                    q3 = vals.quantile(.75)
                    iqr = q3-q1
                    dropindices = df.index[((df[feat1] == i) & (df[feat2]==j) & (df.cnt >
        q3+thresh*iqr))
                                           | ((df[feat1] == i) & (df[feat2] ==j) & (df.cnt <
        q1-thresh*iqr))]
                    df2 = df2.drop(dropindices)

            return df2

In [ ]: def predictCnt(df, algo, impute=True, scale=True):
            df_train, df_test = df.iloc[:len(df)/2], df.iloc[len(df)/2:]
            X_train, y_train = df_train[df_train.columns.drop('cnt')], df_train.cnt
            X_test, y_test = df_test[df_test.columns.drop('cnt')], df_test.cnt

            pipechoices =
        [('myImputer',Imputer()),('myScaler',StandardScaler()),('masterAlg',algo)]
            filt = [impute,scale,True]
            pipecontent = list(compress(pipechoices,filt))

            mypipeline = Pipeline(pipecontent)
            mypipeline.fit(X_train,y_train)
            y_pred = mypipeline.predict(X_test)
            results = [y_test,y_pred]
            return results

In [ ]: def RMSE(y_test,y_pred):
            return (((y_test - y_pred)**2).sum()/len(y_test))**0.5

In [ ]: df2 = df.copy()
        df2 = df2.sample(frac=1)
```

0.5.1 Testing and Tuning Sklearn Outlier Detection Method

```
In [ ]: from sklearn.neighbors import LocalOutlierFactor

In [ ]: testdf = pd.DataFrame(columns = ['# neighbors','RMSE'])

In [ ]: for i in range(1,45,2):
            toutlierclf = LocalOutlierFactor(n_neighbors=i)
            toutlier_pred = toutlierclf.fit_predict(df2)
```

```
            df2_filtt = df2.copy()
            toutliers = np.argwhere(toutlier_pred == -1)
            df2_filtt = df2_filtt.drop([int(x) for x in toutliers.astype(list)])
            tfiltresults = predictCnt(df2_filtt,LinearRegression(),impute=True,scale=True)
            trmse = RMSE(tfiltresults[0],tfiltresults[1])
            temp = pd.DataFrame(data =[[i,trmse]], columns=['# neighbors','RMSE'])
            testdf = testdf.append(temp,ignore_index=True)

In [ ]: testdf.sort_values(by='RMSE').head(3)

In [ ]: outlierclf = LocalOutlierFactor(n_neighbors=31)
        outlier_pred = outlierclf.fit_predict(df2)
        print 'Predicted',np.bincount(outlier_pred+1)[0],'outliers,',np.bincount(outlier_pred+1)
        [2],'points remain'

In [ ]: print np.round(float(15641)*100/len(df2),decimals=2),"percent of original data with
        outliers removed"

In [ ]: df2_filt = df2.copy()
        outliers = np.argwhere(outlier_pred == -1)
        df2_filt = df2_filt.drop([int(x) for x in outliers.astype(list)])

In [ ]: sklrnresults = predictCnt(df2_filt,LinearRegression(),impute=True,scale=True)
        print 'The RMSE for the model predicted by Linear Regression is:
        ',RMSE(sklrnresults[0],sklrnresults[1])

In [ ]: output1 = sklrnresults[0]-sklrnresults[1]
        output1 = output1.iloc[:50]

        figure(figsize=(20,5))
        plt.plot(np.arange(len(output1)),output1)
        plt.xlabel('Data point tested')
        plt.ylabel('RMSE')
        plt.title('RMSE for Linear Regression model with sklearn-tuned outlier removal')
        plt.show()
```

Manually iterating through combinations of features to test outlier removal

```
In [ ]: featlist = ['season','mnth','hr','workingday','weathersit','null']
        combolist = list(combinations(featlist,2))

In [ ]: df2['null'] = 1
        cntpredictions = pd.DataFrame(columns = ['feat1','feat2','thresh','RMSE','len(df)'])
        for j in [x*.1 for x in range(0,25)]:
            for i in range(len(combolist)):
                df2_filt = removeOutliers(df2,combolist[i],j)
                results = predictCnt(df2_filt,LinearRegression())
                rmse = RMSE(results[0],results[1])
                temp = pd.DataFrame(data =
        [[combolist[i][0],combolist[i][1],j,rmse,len(df2_filt)]],
                                    columns=['feat1','feat2','thresh','RMSE','len(df)'])
                cntpredictions = cntpredictions.append(temp,ignore_index=True)
        df2 = df2[df2.columns.drop('null')]

In [ ]: cntpredictions[cntpredictions['thresh'] >= 0.8].sort_values(by='RMSE', ascending =
        True).head(3)

In [ ]: df2_filt = df2.copy()
        for i in df.workingday.unique():
            vals = df2_filt[df2_filt.workingday == i].cnt
            q1 = vals.quantile(.25)
            q3 = vals.quantile(.75)
            iqr = q3-q1
            dropindices = df2_filt.index[((df2_filt.workingday == i) & (df2_filt.cnt >
        q3+.8*iqr))
                                         | ((df2_filt.workingday == i) & (df2_filt.cnt <
        q1-.8*iqr))]
            df2_filt = df2_filt.drop(dropindices)
```

```
In [ ]: print np.round(float(len(df2_filt))*100/len(df2),decimals=2),"percent of original data
        with outliers removed"
```

```
In [ ]: manualresults = predictCnt(df2_filt,LinearRegression())
        rmse = RMSE(manualresults[0],manualresults[1])
        print 'The RMSE for this model is',rmse
```

```
In [ ]: output2 = manualresults[0] - manualresults[1]
        output2 = output2.iloc[:50]

        figure(figsize=(20,5))
        plt.plot(np.arange(len(output2)),output2)
        plt.xlabel('Data point tested')
        plt.ylabel('RMSE')
        plt.title('RMSE for Linear Regression model with manually-tuned outlier removal')
        plt.show()
```

0.5.2 Building a pipeline that includes an imputer and standard scaler

```
In [ ]: df2_train, df2_test = df2.iloc[:len(df2)/2], df2.iloc[len(df2)/2:]
        X_train, y_train = df2_train[df2_train.columns.drop('cnt')], df2_train.cnt
        X_test, y_test = df2_test[df2_test.columns.drop('cnt')], df2_test.cnt

        mypipeline = Pipeline([
            ('myImputer',Imputer()),
            ('myScaler',StandardScaler()),
            ('masterAlg',LinearRegression())
        ])
```

0.5.3 Predicting the CNT using linear regression

```
In [ ]: mypipeline.fit(X_train,y_train)
        y_pred = mypipeline.predict(X_test)
```

```
In [ ]: print 'The RMSE for the model predicted by Linear Regression is: ',RMSE(y_test,y_pred)
```

```
In [ ]: output3 = y_test-y_pred
        output3 = output3.iloc[:50]

        figure(figsize=(20,5))
        plt.plot(np.arange(len(output3)),output3)
        plt.xlabel('Data point tested')
        plt.ylabel('RMSE')
        plt.title('RMSE for Linear Regression model without outlier removal')
        plt.show()
```

0.5.4 Generating a visual comparison of the manual method with the original df

```
In [ ]: figure(figsize=(20,10))
        plt.plot(np.arange(len(output2)),output3)
        plt.plot(np.arange(len(output2)),output2)
        plt.plot(np.arange(len(output2)),output1)
        plt.xlabel('Data point tested')
        plt.ylabel('RMSE')
        plt.legend(['Sklearn LocalOutlierFactor()','Manual Outlier Removal','No Outliers
        Removed'])
        plt.title('Sample of RMSE for Linear Regression model without various outlier removal
        methods')
        plt.show()
```

0.5.5 Using manual outlier removal, comparing Linear Regression to Random Forest Regressor

Random Forest:

```
In [ ]: forestmanualresults = predictCnt(df2_filt,RandomForestRegressor(n_estimators=50,min_samp
        les_split=2,random_state=0))
        rmse = RMSE(forestmanualresults[0],forestmanualresults[1])
        print 'The RMSE for this model is',rmse
```

```
In [ ]: output4 = forestmanualresults[0] - forestmanualresults[1]
        output4 = output4.iloc[:50]

        figure(figsize=(20,5))
        plt.plot(np.arange(len(output4)),output4)
        plt.xlabel('Data point tested')
        plt.ylabel('RMSE')
        plt.title('RMSE for Random Forest Regression model with manually-tuned outlier removal')
        plt.show()
```

Linear Regression:

```
In [ ]: figure(figsize=(20,10))
        plt.plot(np.arange(len(output2)),output4)
        plt.plot(np.arange(len(output2)),output2)
        plt.xlabel('Data point tested')
        plt.ylabel('RMSE')
        plt.legend(['Random Forest Regression','Linear Regression'])
        plt.title('Sample of RMSE for Linear Regression vs Random Forest Regression with manual
        outlier removal methods')
        plt.show()
```

0.6 Identifying the parameters which influence the prediction the most

```
In [ ]: modelcoefs = abs(mypipeline.get_params('masterAlg')['masterAlg'].coef_)
```

```
In [ ]: loader = np.array([(X_train.columns),(modelcoefs)]).T
        coef_rank = pd.DataFrame(data = loader, columns = ['Feature','abs(Model Coef)'])
```

```
In [ ]: coef_rank.sort_values(by = 'abs(Model Coef)', ascending=False)
```

0.7 Predicting whether it is a raining day (weathersit) or not from the non-weather columns

```
In [ ]: def binaryPerformance(results):
            tn, fp, fn, tp = sklearn.metrics.confusion_matrix(results[:,0],
        results[:,1]).ravel()
            tpr = float(tp)/(tp + fn)
            tnr = float(tn)/(tn + fp)
            ppr = float(tp)/(tp + fp)
            npr = float(tn)/(tn + fn)
            accuracy = float(tp+tn)/(tp+tn+fp+fn)
            f1 = 2.0/(1.0/tpr + 1.0/ppr)
            columns = ['True Positives Rate','True Negatives Rate','Positive Predicted Rate',
                       'Negative Predicted Rate','F1 Score','Accuracy']
            perfmetrics = pd.DataFrame(data=[[tpr,tnr,ppr,npr,f1,accuracy]],columns=columns)
            return perfmetrics
```

```
In [ ]: def predictWeather(df, algo, impute=True, scale=True):
            df_train, df_test = df.iloc[:len(df)/2], df.iloc[len(df)/2:]
            X_train, y_train =
        df_train[df_train.columns.drop(['weathersit','atemp','hum','windspeed'])],
        df_train.weathersit
```

```
        X_test, y_test =
    df_test[df_test.columns.drop(['weathersit','atemp','hum','windspeed'])],
    df_test.weathersit

        pipechoices =
    [('myImputer',Imputer()),('myScaler',StandardScaler()),('masterAlg',algo)]
        filt = [impute,scale,True]
        pipecontent = list(compress(pipechoices,filt))

        mypipeline = Pipeline(pipecontent)
        mypipeline.fit(X_train,y_train)
        y_pred = mypipeline.predict(X_test)
        results = np.array([y_test,y_pred])
        return np.transpose(results)
```

```
In [ ]: df3 = df2.copy()
        df3.weathersit = df3.weathersit >= 3
```

Logistic Regression

```
In [ ]: weathercf = predictWeather(df3,LogisticRegression(class_weight='balanced'))
        performance = binaryPerformance(weathercf)
        performance
```

SVM

```
In [ ]: weathercf2 = predictWeather(df3,SVC(kernel='sigmoid'))
        performance2 = binaryPerformance(weathercf2)
        performance2
```

KNN

```
In [ ]: knnweathertable = pd.DataFrame(columns = ['# Neighbors','Scaled','F1 Score','Accuracy'])
        for i in range(1,50,2):
            for j in range(0,2):
                resultlist = predictWeather(df3,KNeighborsClassifier(n_neighbors=i),scale=j)
                performance = binaryPerformance(resultlist)
                loader = pd.DataFrame(data=[[i,j,performance['F1
    Score'][0],performance['Accuracy'][0]]],columns=['# Neighbors','Scaled','F1
    Score','Accuracy'])
                knnweathertable = knnweathertable.append(loader)
```

```
In [ ]: knnweathertable.sort_values(by='Accuracy',ascending=False).head(3)
```

```
In [ ]: weathercf3 = predictWeather(df3,KNeighborsClassifier(n_neighbors=5),scale=1)
        performance3 = binaryPerformance(weathercf3)
        performance3
```

Random Forest

```
In [ ]: weathercf4 = predictWeather(df3,RandomForestClassifier(n_estimators=275,min_samples_spli
    t=7,random_state=0,class_weight='balanced_subsample'),scale=1)
        performance4 = binaryPerformance(weathercf4)
        performance4
```

0.8 Predicting whether it is a working day or not from the other columns

```
In [ ]: def predictWorkingDay(df, algo, impute=True, scale=True):
            df_train, df_test = df.iloc[:len(df)/2], df.iloc[len(df)/2:]
            X_train, y_train = df_train[df_train.columns.drop('workingday')],
        df_train.workingday
            X_test, y_test = df_test[df_test.columns.drop('workingday')], df_test.workingday
```

```
    pipechoices =
[('myImputer',Imputer()),('myScaler',StandardScaler()),('masterAlg',algo)]
    filt = [impute,scale,True]
    pipecontent = list(compress(pipechoices,filt))

    mypipeline = Pipeline(pipecontent)
    mypipeline.fit(X_train,y_train)
    y_pred = mypipeline.predict(X_test)
    results = np.array([y_test,y_pred])
    return np.transpose(results)
```

Logistic Regression

```
In [ ]: workcf = predictWorkingDay(df2,LogisticRegression(class_weight='balanced'),impute=False,
        scale=False)
        workperf = binaryPerformance(workcf)
        workperf
```

SVM

```
In [ ]: workcf2 = predictWorkingDay(df2,SVC(kernel='rbf',class_weight='balanced'))
        workperf2 = binaryPerformance(workcf2)
        workperf2
```

KNN

```
In [ ]: knnworktable = pd.DataFrame(columns = ['# Neighbors','Scaled','F1 Score','Accuracy'])
        for i in range(3,30,2):
            for j in range(0,2):
                resultlist =
        predictWorkingDay(df2,KNeighborsClassifier(n_neighbors=i,weights='distance'),scale=j)
                performance = binaryPerformance(resultlist)
                loader = pd.DataFrame(data=[[i,j,performance['F1
        Score'][0],performance['Accuracy'][0]]],columns=['# Neighbors','Scaled','F1
        Score','Accuracy'])
                knnworktable = knnworktable.append(loader)

In [ ]: knnworktable.sort_values(by='Accuracy',ascending=False).head(3)

In [ ]: workcf3 = predictWorkingDay(df2,KNeighborsClassifier())
        workperf3 = binaryPerformance(workcf3)
        workperf3
```

Random Forest

```
In [ ]: workcf4 = predictWorkingDay(df2,RandomForestClassifier(n_estimators=250,min_samples_spli
        t=5,random_state=0,class_weight='balanced_subsample'),scale=1)
        workperf4 = binaryPerformance(workcf4)
        workperf4

In [ ]: df_train, df_test = df.iloc[:len(df)/2], df.iloc[len(df)/2:]
        X_train, y_train = df_train[df_train.columns.drop('workingday')], df_train.workingday
        X_test, y_test = df_test[df_test.columns.drop('workingday')], df_test.workingday
        forest = RandomForestClassifier(n_estimators=250,min_samples_split=5,random_state=0,clas
        s_weight='balanced_subsample')
        forest.fit(X_train,y_train)

In [ ]: from sklearn.cluster import KMeans
```

In [ ]:
```
X_train = df3[df3.columns.drop(['weathersit','atemp','hum','windspeed'])]
X_test = df3.weathersit

mypipeline = Pipeline([
    ('myImputer',Imputer()),
    ('myScaler',StandardScaler()),
    ('masterAlg',KMeans(n_clusters=2))
])
mypipeline.fit(X_train)

X_pred = mypipeline.get_params('masterAlg')['masterAlg'].labels_
results = np.array([X_test, X_pred]).T
```

In [ ]:
```
binaryPerformance(results)
```