Bike Renting

Nirmal Sharma

13 March,2019

Contents

Chapter 1

1.1 Introduction

Biking is an excellent way to stay in shape while exploring local areas and communing with nature. With many biking enthusiasts eager to find new paths to explore in and around their local area . Our case study is regarding an organization who lends rental bike. According to business sense the demand of bikes for a particular day depends upon several factors like weather situation, season, holiday etc. It is important to know the demand of a particular day beforehand, so that they can meet the demand smoothly.

1.2 Problem Statement

The objective of this project is to count the number of bike hired on rent on daily basis. We are provided with the seasonal and environmental condition by using which we have to predict the count of bike on the particular day. We are provided with the data of year 2011 and 2012 with the count of bike rented on each day of the year

1.3 Data

Our task is to build the regression model which will predict the rental bike count on daily basis based on multiple predictor provided in the problem statement. Given below is the data set which we will be using for predicting the values of target variable ('cnt') of test dataset.

instant	dteday	season	yr	mnth	holiday	weekday	workingday	Weathersit	temp
1	1/1/2011	1	0	1	0	6	0	2	0.344167
2	1/2/2011	1	0	1	0	0	0	2	0.363478
3	1/3/2011	1	0	1	0	1	1	1	0.196364
4	1/4/2011	1	0	1	0	2	1	1	0.2
5	1/5/2011	1	0	1	0	3	1	1	0.226957

Table 1.1 : Bike Rental Sample Data (Columns: 1-10)

atemp	hum	windspeed	casual	Registered	Cnt
0.363625	0.805833	0.160446	331	654	985
0.353739	0.696087	0.248539	131	670	801
0.189405	0.437273	0.248309	120	1229	1349
0.212122	0.590435	0.160296	108	1454	1562
0.22927	0.436957	0.1869	82	1518	1600

Table 1.2 : Bike Rental Sample Data(Columns: 11-16)

As we can see above that the table have 15 predictor variables, using which we have to predict the count of bike rented on daily basis. The ‘casual’ and ‘registered’ variables simply tell us that how many casual user and how many registered user have rented the bike and cnt variable tell us about count of total rental bikes including both casual and registered.

S.No.	**Predictor**
1.	instant
2.	dteday
3.	season
4.	Year(yr)
5.	mnth
6.	holiday
7.	weekday
8.	workingday
9.	weathersit
10.	temp
11.	atemp
12.	hum
13.	windspeed
14.	casual
15.	registered

Table 1.3 : Predictor Variables

CHAPTER 2

METHODOLOGY

2.1 Pre Processing

We should always look at the data before preparing the model. However, in data mining terms looking at data refers to much more than just looking. Looking at the data means exploring the data, cleaning the data and visualizing the data through graphs and plot. This is often called Exploratory Data Analysis . Looking at the data we can see that the ‘instant’ and ‘dteday’ feature is not contributing much while calculating the count of rental bike, since both the feature contain unique values, that means we can remove this feature. Also, we can remove the ‘casual’ and ‘registered’ feature from the data set because ‘cnt’ feature is the sum of both the feature, hence this two feature are correlated to the ‘cnt’ and can be removed.

2.1.1 Missing Value Analysis

Before proceding, we must check our data set for the missing values. If there are missing values available in our data set then we must impute those values for getting better result. According to industry standards, if the data set have less than 30 % missing values than we must impute values. There are different techniques which can be used to impute missing values in data set.

2.1.2 Outlier Analysis

As we can see in the Figure 2.1 the ‘hum’ and ‘windspeed’ predictors are skewed. The skew in the distribution is mostly the presence of outliers and extreme values in data.

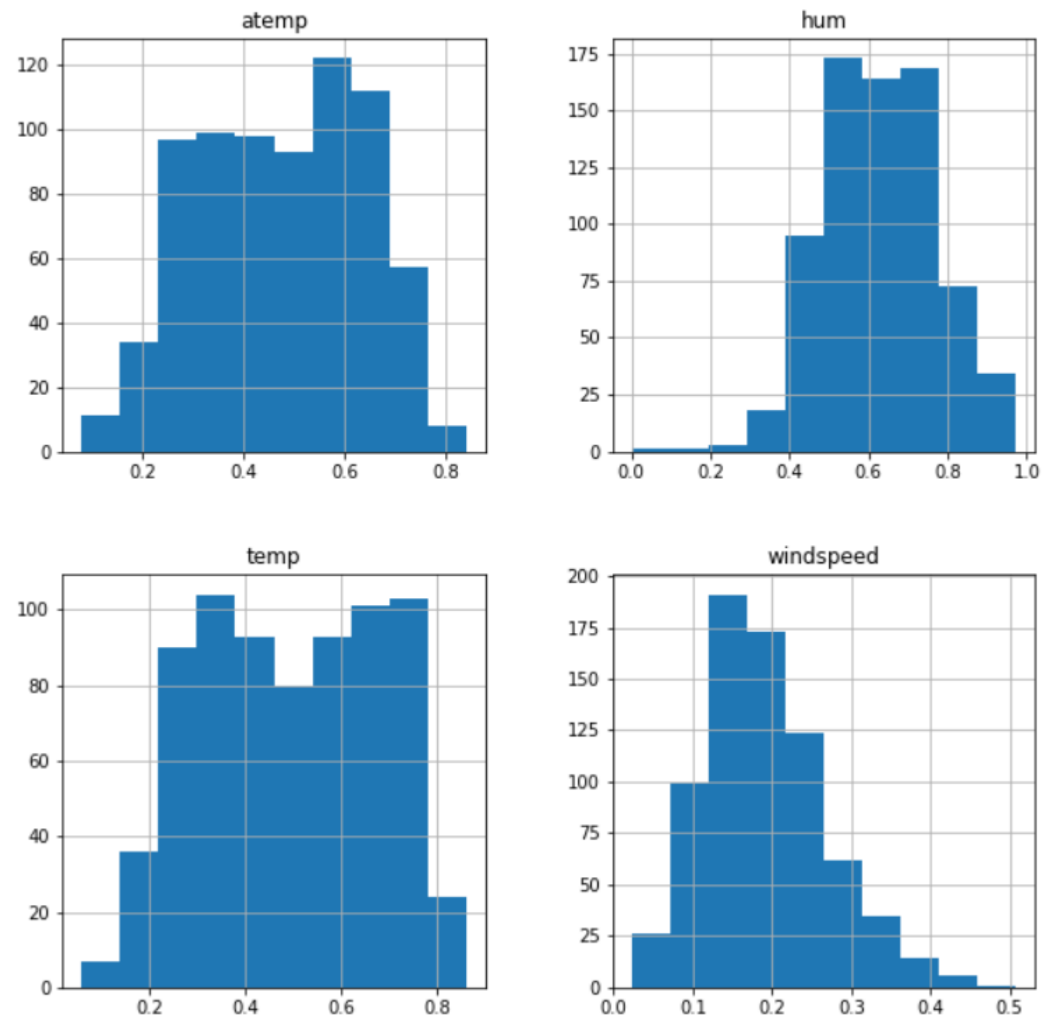


Table 2.1: Histogram of Numeric Predictor

One of the other step of preprocessing apart from the missing values analysis is presence of outliers. An outlier is an observation that lies an abnormal distance from other values in random sample from the population. In this case we can use one of the classic approach of removing outliers, Tukey's method. We can visualize the outliers using the boxplot. We can only plot the boxplot of numeric variables with the target variable. In our data set we have only four numeric features, hence we can only plot boxplot of these feature.

In Figure 2.2 we have plotted the box plot of the four features with respect to the count variable. A lot of useful inferences can be made from these plots.

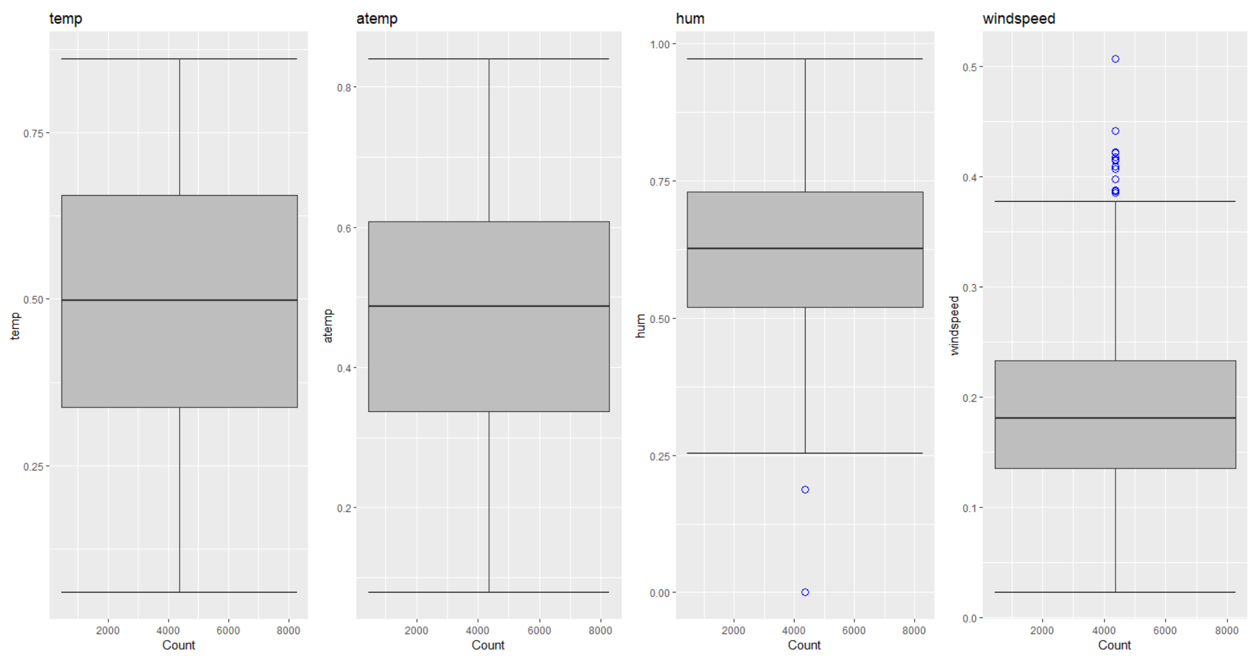


Figure 2.2: Boxplot for Outlier Analysis

As we can see in the above figure that there are some outliers present in the numeric data. We have to remove these outliers to bring our data in the proper shape. We can remove the outliers and impute the nearest value in place of outliers, in our case we have impute the values with Central Tendency (mean method).

After performing the outlier analysis and removing outliers using Tukey's Boxplot method. We can plot the histogram and boxplot of the numeric variables again and check how the distribution look. Figure 2.3 the histogram of numeric data after removing the outliers.

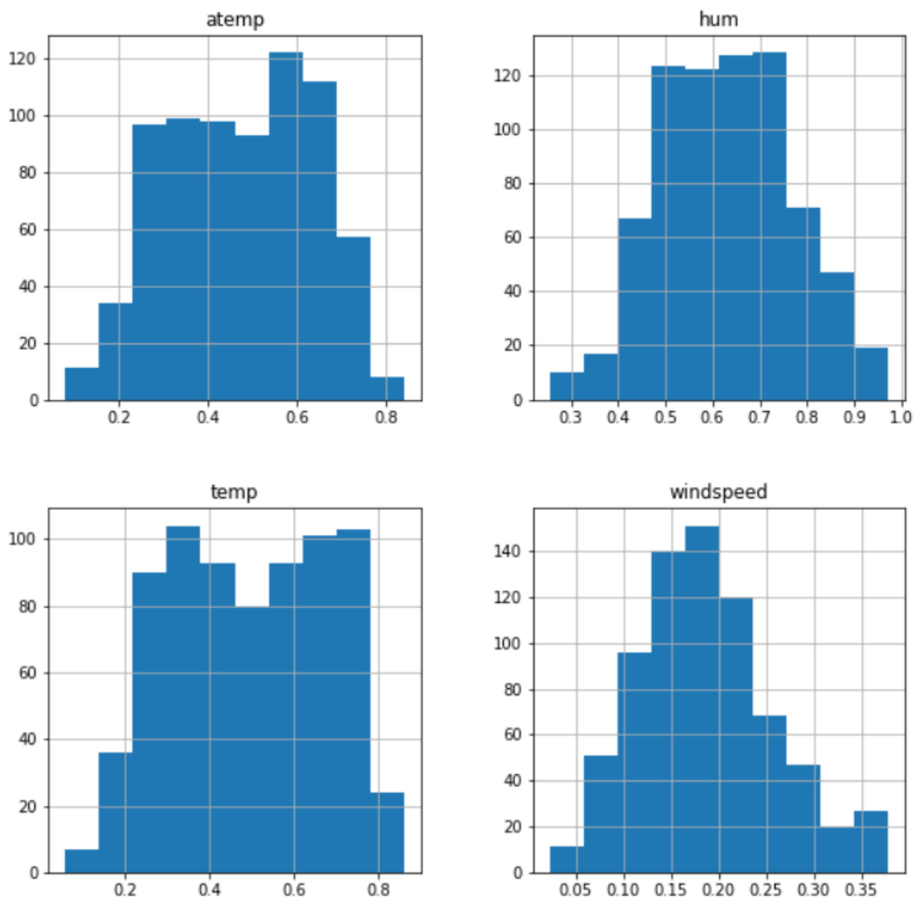


Figure 2.3 : Histogram of Numeric data after removing outliers

2.1.3 Feature Selection (Variable Reduction)

Before performing any type of modeling we need to check the importance of each predictor variable present in our analysis. There is a possibility that many variables are not at all important to the problem of prediction. The process of selecting only important features from our data set is called Feature Selection . There are several method for selecting feature from the data set. Below we have use Correlation method for Numeric Feature Selection and Variance Inflation Factor.

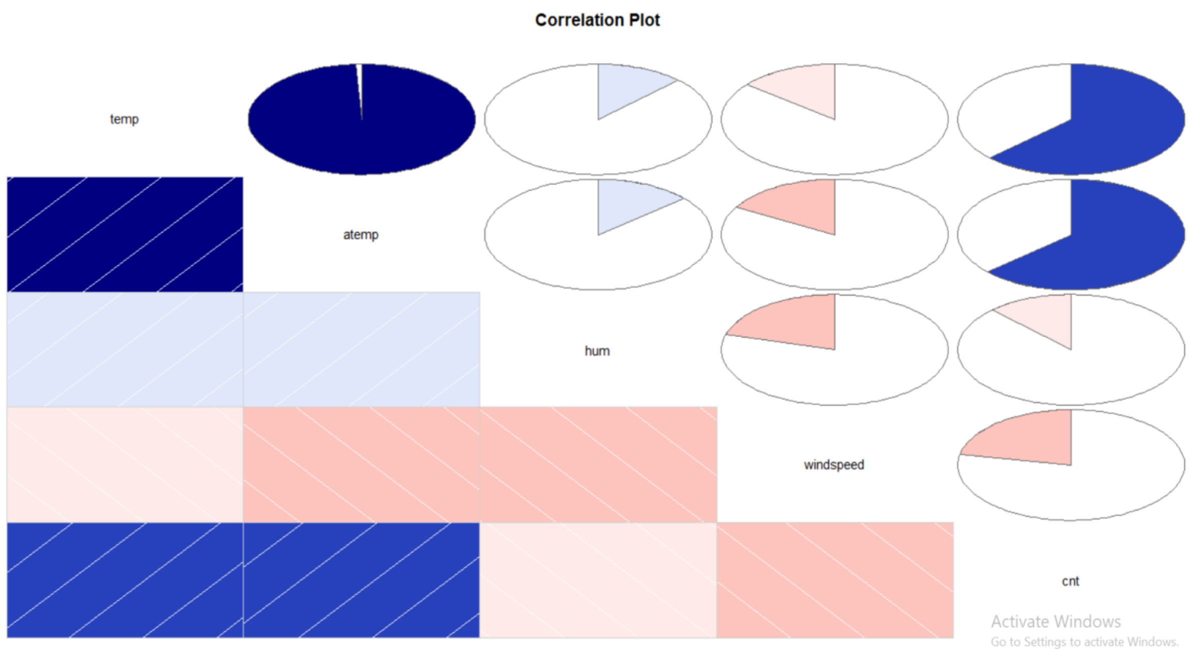
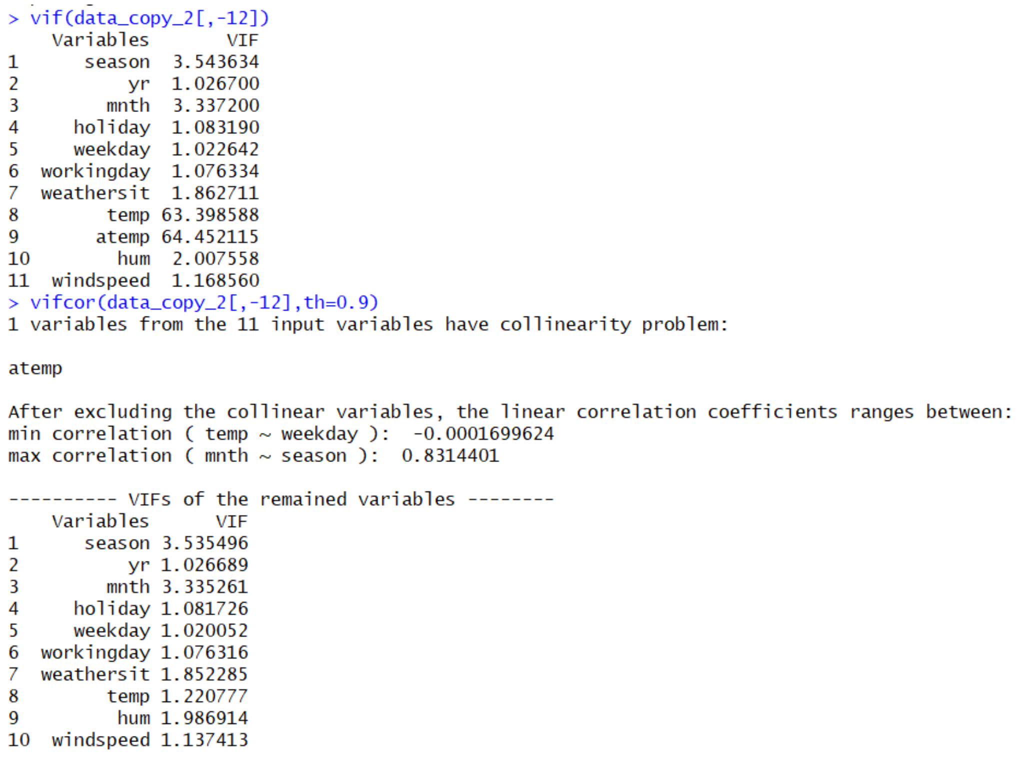


Figure 2.4 : Correlation Plot

As we can see above in correlation plot that **'atemp'** predictor is highly correlated with **'temp'** variable. So we can remove any, either remove 'atemp' or' temp' predictor from our data set.



```
> vif(data_copy_2[,-12])
    Variables       VIF
1      season  3.543634
2          yr  1.026700
3        mnth  3.337200
4     holiday  1.083190
5     weekday  1.022642
6  workingday  1.076334
7  weathersit  1.862711
8        temp 63.398588
9       atemp 64.452115
10        hum  2.007558
11  windspeed  1.168560
> vifcor(data_copy_2[,-12],th=0.9)
1 variables from the 11 input variables have collinearity problem:

atemp

After excluding the collinear variables, the linear correlation coefficients ranges between:
min correlation ( temp ~ weekday ):  -0.0001699624
max correlation ( mnth ~ season ):  0.8314401

---------- VIFs of the remained variables --------
    Variables      VIF
1      season 3.535496
2          yr 1.026689
3        mnth 3.335261
4     holiday 1.081726
5     weekday 1.020052
6  workingday 1.076316
7  weathersit 1.852285
8        temp 1.220777
9         hum 1.986914
10  windspeed 1.137413
```

Figure2.5 :Variance Inflation Factor for Feature Selection

After finding the variance inflation factor for all the predictors , we can see that ‘atemp’ and ‘temp’ have high vif , that means that these features are highly correlated with each other that means one of them can be removed and after performing Correlation Analysis and Variance Inflation Factor Analysis, we came to know that ‘atemp’ feature can be removed from the data set because it is dependent on the ‘temp’ variable.

2.1.4 Feature Scaling

Feature Scaling is the method which is used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization. As we can see in the problem statement that all the independent features are already normalized, that is why we don’t have to perform normalization in our dataset.

2.2 Modeling

2.2.1 Model Selection

The independent variables can fall in either of the categories :

1. Nominal

2. Ordinal

3. Interval

4. Ratio

If the dependent variable, in our case ‘cnt’, is Interval or Ratio then the normal method is to do a Regression Analysis, or Classification after binning. If the dependent variable is Ordinal, then both regression and classification can be done. If dependent variable is Nominal the only predictive analysis that we can perform is Classification. In our case the dependent variable is continuous that means we have to do Regression Analysis.

There are lot of machine learning algorithm available which can be used to solve the regressionproblem. Few of them are Decision Tree, Random Forest, Linear Regression, XGBoost etc. We can use any machine learning algorithm to predict the values of target variable. We will use multiple algorithm to find the values of target variable. And, then after calculating the values of target variable, we will select only that model which have the highest accuracy and less error.

2.2.2 Decision Tree

It is predictive model based on the branching series of boolean test. Decision Tree is basically used to create the predictive model which can be used to find value of target variable by using a decision rules which are collected from the past. Output of the decision tree is simple business rules. There are different algorithm present in the Decision Tree like rpart, C5.0, CART etc. The output of the decision tree is alway in form of rules, these rules are then applied on the test data to predict the values of target variables.

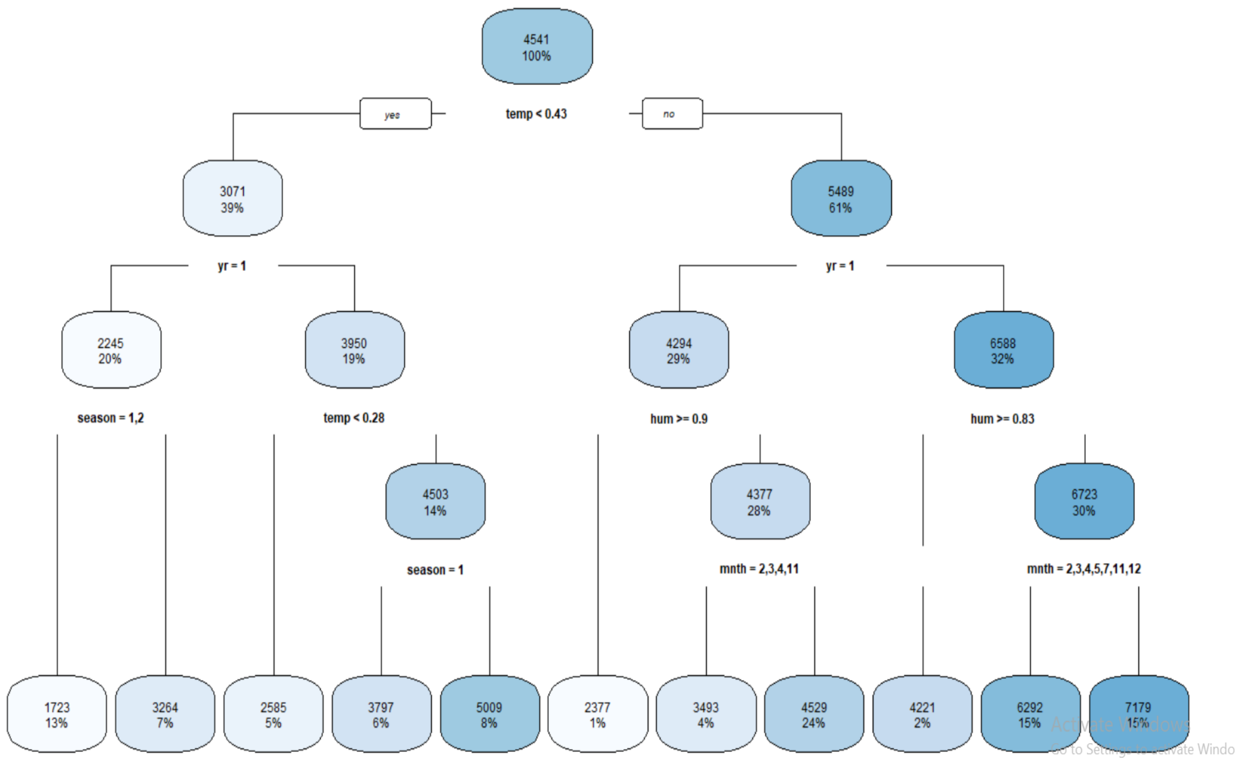
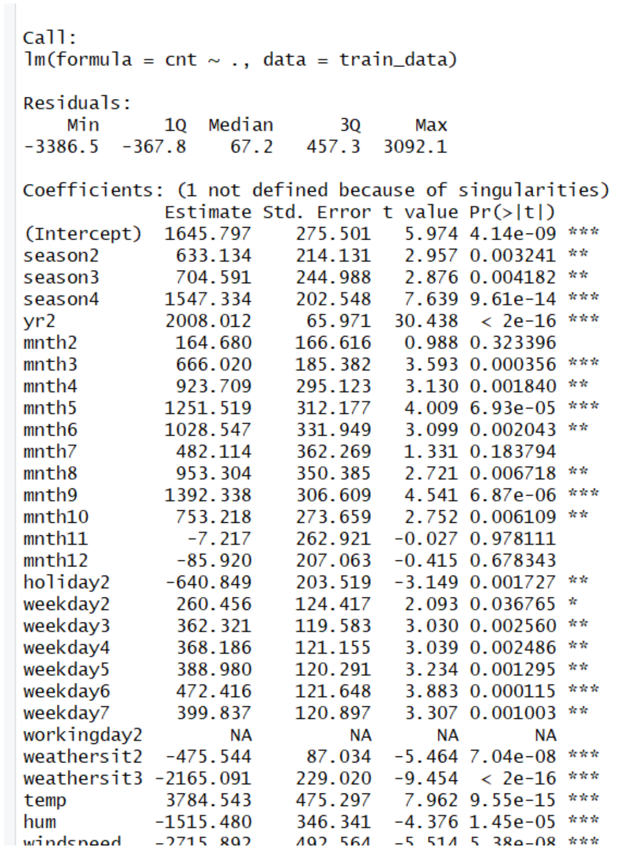


Figure 2.6 : Decision Tree Rules

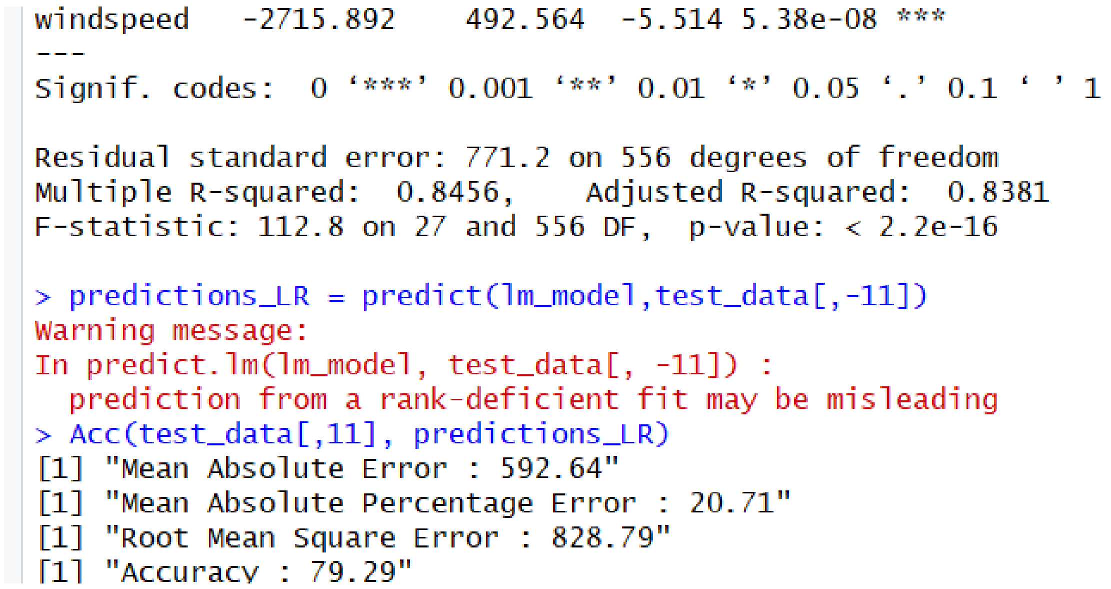
The rpart method of Decision Tree is giving us the accuracy 78.02% ,which is not very impressive. We should try some other models to get the better accuracy than the Decision Tree Model.

2.2.3 Linear Regression

The regression is basically used to predict the relationship among two or more independent variables and dependent variable. Let's use linear regression model to predict the values of target variables.



```
Call:
lm(formula = cnt ~ ., data = train_data)

Residuals:
    Min      1Q  Median      3Q     Max 
-3386.5  -367.8    67.2   457.3  3092.1 

Coefficients: (1 not defined because of singularities)
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  1645.797    275.501   5.974 4.14e-09 ***
season2       633.134    214.131   2.957 0.003241 ** 
season3       704.591    244.988   2.876 0.004182 ** 
season4      1547.334    202.548   7.639 9.61e-14 ***
yr2          2008.012     65.971  30.438  < 2e-16 ***
mnth2         164.680    166.616   0.988 0.323396    
mnth3         666.020    185.382   3.593 0.000356 ***
mnth4         923.709    295.123   3.130 0.001840 ** 
mnth5        1251.519    312.177   4.009 6.93e-05 ***
mnth6        1028.547    331.949   3.099 0.002043 ** 
mnth7         482.114    362.269   1.331 0.183794    
mnth8         953.304    350.385   2.721 0.006718 ** 
mnth9        1392.338    306.609   4.541 6.87e-06 ***
mnth10        753.218    273.659   2.752 0.006109 ** 
mnth11         -7.217    262.921  -0.027 0.978111    
mnth12        -85.920    207.063  -0.415 0.678343    
holiday2     -640.849    203.519  -3.149 0.001727 ** 
weekday2      260.456    124.417   2.093 0.036765 *  
weekday3      362.321    119.583   3.030 0.002560 ** 
weekday4      368.186    121.155   3.039 0.002486 ** 
weekday5      388.980    120.291   3.234 0.001295 ** 
weekday6      472.416    121.648   3.883 0.000115 ***
weekday7      399.837    120.897   3.307 0.001003 ** 
workingday2        NA         NA      NA       NA    
weathersit2  -475.544     87.034  -5.464 7.04e-08 ***
weathersit3 -2165.091    229.020  -9.454  < 2e-16 ***
temp         3784.543    475.297   7.962 9.55e-15 ***
hum         -1515.480    346.341  -4.376 1.45e-05 ***
```

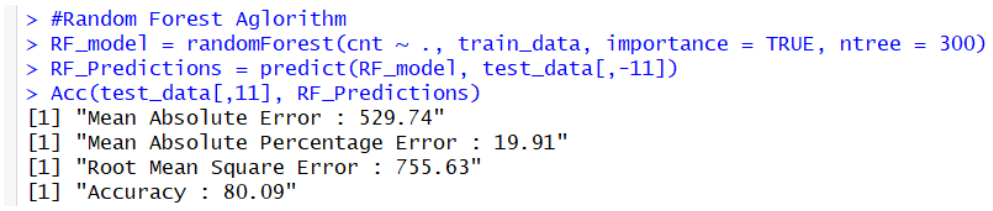


```
windspeed   -2715.892    492.564  -5.514 5.38e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 771.2 on 556 degrees of freedom
Multiple R-squared:  0.8456,	Adjusted R-squared:  0.8381 
F-statistic: 112.8 on 27 and 556 DF,  p-value: < 2.2e-16

> predictions_LR = predict(lm_model,test_data[,-11])
Warning message:
In predict.lm(lm_model, test_data[, -11]) :
  prediction from a rank-deficient fit may be misleading
> Acc(test_data[,11], predictions_LR)
[1] "Mean Absolute Error : 592.64"
[1] "Mean Absolute Percentage Error : 20.71"
[1] "Root Mean Square Error : 828.79"
[1] "Accuracy : 79.29"
```

As we can see linear regression model is giving us the accuracy of 79.29% , which is better than the decision tree but not very impressive. Let's implement some other models to find the better accuracy than linear regression.

2.2.4 Random Forest

Random forest is an ensemble that consists of many decision trees. Random forest is basically combination of weak learner to produce the strong learning model. Let us apply random forest algorithm and try to find the values of target variable.



```
> #Random Forest Aglorithm
> RF_model = randomForest(cnt ~ ., train_data, importance = TRUE, ntree = 300)
> RF_Predictions = predict(RF_model, test_data[,-11])
> Acc(test_data[,11], RF_Predictions)
[1] "Mean Absolute Error : 529.74"
[1] "Mean Absolute Percentage Error : 19.91"
[1] "Root Mean Square Error : 755.63"
[1] "Accuracy : 80.09"
```

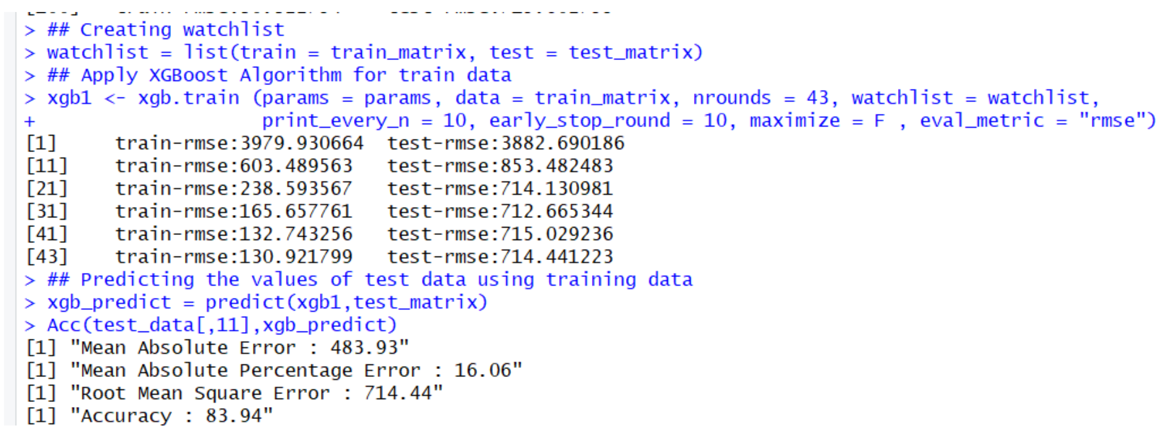
As we can see above that Random Forest is giving us the accuracy of 80.09%, which is better than decision tree and linear regression model. But as per Machine learning, the accuracy is still not good enough.

2.2.5 Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting is optimized distributed gradient boosting library. It uses the gradient boosting framework at the core. It is used for supervised ML problems. XGBoost is better than other model because it is enabled with parallel processing, enabled cross validation, handle missing values internally and have provision of regularization technique to avoid overfitting.

2.2.5.1 Extreme Gradient Boosting with Predefined parameters

In XGBoost, only numeric variables are used to predict the target variable. So first thing we do is we convert all the categorical variables to dummy variables.
After converting all the categorical variables to numeric then we define the parameters for XGBoost and by doing cross validation, we find the minimum number of rounds required. And then we apply our XGBoost model on our train data to predict the values of target variable of test data.



```
> ## Creating watchlist
> watchlist = list(train = train_matrix, test = test_matrix)
> ## Apply XGBoost Algorithm for train data
> xgb1 <- xgb.train (params = params, data = train_matrix, nrounds = 43, watchlist = watchlist,
+                    print_every_n = 10, early_stop_round = 10, maximize = F , eval_metric = "rmse")
[1]     train-rmse:3979.930664  test-rmse:3882.690186
[11]    train-rmse:603.489563   test-rmse:853.482483
[21]    train-rmse:238.593567   test-rmse:714.130981
[31]    train-rmse:165.657761   test-rmse:712.665344
[41]    train-rmse:132.743256   test-rmse:715.029236
[43]    train-rmse:130.921799   test-rmse:714.441223
> ## Predicting the values of test data using training data
> xgb_predict = predict(xgb1,test_matrix)
> Acc(test_data[,11],xgb_predict)
[1] "Mean Absolute Error : 483.93"
[1] "Mean Absolute Percentage Error : 16.06"
[1] "Root Mean Square Error : 714.44"
[1] "Accuracy : 83.94"
```

As we can see above that XGBoost with predefined parameters is giving us the accuracy of 83.94% which is better than Decision Tree, Linear Regression and Random Forest models.

CHAPTER 3

CONCLUSION

3.1 Model Evaluation

Now that we have few models for predicting the target variable, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models. We can compare the model using any of the following criteria:

1. Predictive Performance

2. Interpretability

3. Computational Efficiency

We will use Predictive Performance as our criteria to compare and evaluate the models. Predictive Performance can be measured by comparing Predictions of models with real values of target variable and calculating some average error measure.

Model	**Mean Absolute Error**	**Mean Absolute Percentage Error**	**RootMean Square Error**	**Accuracy**
Decision Tree	599.49	21.98	856.19	78.02%
Linear Regression	892.64	20.71	828.79	79.28%
Random Forest	529.74	19.91	755.63	80.09%
XGBoost	483.93	16.06	714.44	83.94%

Table 3.1: Model Evaluation

3.1.1 Mean Absolute Error (MAE)

Mean Absolute Error is the average mean of the absolute Errors and is calculated by the below mentioned formula

MAE = (1/n)*(Σ abs (y actual - y predict))

MAE is one of the error measures used to calculate the predictive performance of the model. We will apply this measure to our model and try to analyze our model.

As we can see in Table 3.1, we have calculate the Mean Absolute Error of all the models which we have coded for predicting the target variable of test data set. From the above table we can say that XGBoost is giving us the least value of MAE.

3.1.2 Mean Absolute Percentage Error (MAPE)

Mean Absolute Percentage Error measures accuracy as a percentage of the error and is calculated by the below formula.

MAPE = $(1/n)*(\Sigma \text{ abs}((y \text{ actual} - y \text{ predict})/y \text{ actual}))$

MAPE is one of the error measures used to calculate the Mean Absolute Percentage Error of all the models. In Table 3.1 we can say that XGBoost is giving us the least value of MAPE.

3.1.3 Root Mean Square Error (RMSE)

Root Mean Square Error is frequently used measure to calculate the difference between values predicted by model or an estimator and the values observed. The RMSE of the model is calculated by the below mentioned formula.

RMSE = $\sqrt{}\ ((1/n)*(\Sigma \text{ abs } (y \text{ actual} - y \text{ predict})^2))$

In Table 3.1 we can see that XGBoost model is giving least value of Root Mean Square Error.

3.1.4 Accuracy

Accuracy is one of the measure which we can use to evaluate the performance of the model.

As we can see in Table 3.1 the maximum accuracy is given by the XGBoost model.

3.2 Model Selection

We can use XGBoost model as we can see above in the table that this model is given us the maximum accuracy and considering other metrics like MAE, RMSE and MAPE, the XGBoost model is the one which is performing better than the other models.

Appendix A - R Code

Complete R Code

```
rm(list=(ls()))

getwd()


#Load the data

data=read.csv("E:/Edwisor/Projects/Bike Rent/day.csv",header=T)


# installing packages


x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "C50","xgboost",
"e1071","gridExtra",

     "MASS",'DataCombine', "gbm","dplyr","Matrix","dummies")


lapply(x, require, character.only = TRUE)

rm(x)

str(data)


# Missing Value

sapply(data, function(x) sum(is.na(x)))


#Copy of a a data

data_copy=data
```

```
# Removing Features "instant" , "dteday" ,"registered" and "casual" from the dataset. These
features does have any relevance.

data_copy=subset(data_copy,select=-c(instant,dteday,registered,casual))

dim(data_copy)


# Changing the datatype of target variable from integer to numeric

data_copy$cnt=as.numeric(data_copy$cnt)



#Changing the datatype to factor and assigning the labels

for(i in 1:ncol(data_copy)){

 if(class(data_copy[,i]) == 'integer'){

   data_copy[,i]=as.factor(data_copy[,i])

   data_copy[,i]= factor(data_copy[,i],labels = 1:length(levels(factor(data_copy[,i]))))

 }

}


str(data_copy)



#Getting all the numeric columns from the data frame

numeric_index = sapply(data_copy,is.numeric)

numeric_data=data_copy[,numeric_index]

cnames = colnames(numeric_data)
cnames = cnames[-5]

cnames


#Creating box plot of the numeric data for outlier analysis

for ( i in 1:length(cnames))

{

  assign(paste0("gn",i), ggplot(aes_string(y = (cnames[i]), x = "cnt"), data =
subset(data_copy))+

         stat_boxplot(geom = "errorbar", width = 0.5) +

         geom_boxplot(outlier.colour="blue", fill = "grey" ,outlier.shape=1,

                      outlier.size=3, notch=FALSE) +

         theme(legend.position="bottom")+

         labs(x="Count")+

         ggtitle(paste(cnames[i])))+scale_x_discrete(breaks = NULL)

}




#Plotting Box Plot on the Plot window

gridExtra::grid.arrange(gn1,gn2,gn3,gn4,ncol=4)


#Creating the histogram of the features having outliers plot

hist(data_copy$windspeed)

hist(data_copy$hum)
```

```
#Finding all the outliers in the data set

for(i in cnames){

  val = data_copy[,i][data_copy[,i] %in% boxplot.stats(data_copy[,i])$out]

  data_copy[,i][data_copy[,i] %in% val] = NA

}


#Calculating Missing values in data frame(train_data)

missing_val_removed=data.frame(apply(data_copy,2,function(x){sum(is.na(x))}))


#Imputing NA values with mean Method

data_copy$hum[is.na(data_copy$hum)] = mean(data_copy$hum, na.rm = T)

data_copy$windspeed[is.na(data_copy$windspeed)] = mean(data_copy$windspeed, na.rm
= T)


#Histogram of features after removing the outliers

hist(data_copy$windspeed)

hist(data_copy$hum)


#Feature Selection#

#Plotting Correlation plot of the numeric data
```

```
corrgram(data_copy[,numeric_index], order = F,

        upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")



#Checking the correlation among all the features using variance inflation factor.


data_copy_2= data_copy

for(i in 1:ncol(data_copy_2)){

  if(class(data_copy_2[,i]) == 'factor'){

    data_copy_2[,i]=as.numeric(data_copy_2[,i])

  }

}

#Multicollinearity test

install.packages("usdm")

library(usdm)


vif(data_copy_2[,-12])

vifcor(data_copy_2[,-12],th=0.9)



#Feature Selection


data_copy3 = subset(data_copy,select =-c(atemp))

str(data_copy3)
```

```
#Modeling#


#Spliting the data into train and test data

set.seed(1234)

train.index = sample(1:nrow(data_copy3),0.8*nrow(data_copy3))

train_data = data_copy3[train.index,]

test_data = data_copy3[-train.index,]


str(train_data)



#Defining function which will we used to find the accuracy of the model

## Mean Absolute Percentage Error

MAPE = function(y, yhat){

  mean(abs((y - yhat)/y))*100

}


## Mean Absolute Error

MAE = function(y,f){

mean(abs(y-f))

}


## Root Mean Square Error

RMSE = function(y,f){

  sqrt(mean((y-f)^2))
}



## Accuracy
Acc = function(test_data_true, predicted_values){
  mean_abs_error = format(round(MAE(test_data_true,predicted_values),2),nsmall = 2)
  root_mean_sq_er = format(round(RMSE(test_data_true,predicted_values),2),nsmall = 2)
  Error = format(round(MAPE(test_data_true,predicted_values),2),nsmall = 2)
  Accuracy = 100 - as.numeric(Error)
  print(paste0("Mean Absolute Error : ", mean_abs_error))
  print(paste0("Mean Absolute Percentage Error : " , Error))
  print(paste0("Root Mean Square Error : ", root_mean_sq_er))
  print(paste0("Accuracy : ", Accuracy))
}



#Decision Tree#
install.packages("rpart.plot")
library(rpart.plot)
library(rpart)

##Rpart for regression
dt_model = rpart(cnt~.,data=train_data,method = 'anova')
rpart.plot(dt_model)
rpart.rules(dt_model)

predict_dt = predict(dt_model,test_data[,-11])

Acc(test_data[,11],predict_dt)



## Error Rate 21.98

## Accuracy 78.02



#Linear Regression

lm_model= lm(cnt~.,data= train_data)

summary(lm_model)

predictions_LR = predict(lm_model,test_data[,-11])

Acc(test_data[,11], predictions_LR)


## Error Rate 20.71

## Accuracy 79.29


#Random Forest Aglorithm

RF_model = randomForest(cnt ~ ., train_data, importance = TRUE, ntree = 300)

RF_Predictions = predict(RF_model, test_data[,-11])


Acc(test_data[,11], RF_Predictions)


## Error 19.91
## Accuracy 80.09


##Gradient Boosting Algorithm##

## creating Spare Matrix which converts Categorical Variables to dummy variables

trainm = sparse.model.matrix(cnt~.-1,data = train_data)

train_label <- train_data[,"cnt"]

train_matrix = xgb.DMatrix(data = as.matrix(trainm),label = train_label)


testm = sparse.model.matrix(cnt~.-1,data = test_data)

test_label = test_data[,"cnt"]

test_matrix = xgb.DMatrix(data = as.matrix(testm), label = test_label)


#Defining Parameters for Xgboost

params <- list(booster = "gbtree", objective = "reg:linear", eta=0.2, gamma=0, max_depth=6,

          min_child_weight=1,

          subsample=1, colsample_bytree=1)


## Cross Validation - for finding the minimum number of rounds required to find the best
accuracy


xgbcv <- xgb.cv( params = params, data = train_matrix, nrounds = 100, nfold = 5, showsd =
F, stratified = T,
              maximize = F)


## Creating watchlist

watchlist = list(train = train_matrix, test = test_matrix)


## Apply XGBoost Algorithm for train data

xgb1 <- xgb.train (params = params, data = train_matrix, nrounds = 43, watchlist =
watchlist,

               print_every_n = 10, early_stop_round = 10, maximize = F , eval_metric = "rmse")


## Predicting the values of test data using training data

xgb_predict = predict(xgb1,test_matrix)

Acc(test_data[,11],xgb_predict)


##Error 16.06

##Acuracy 83.94
```

References

- Edwisor.com - Online Learning Material
- https://www.analyticsvidhya.com/