IST687 – HomeWork 9 Support Vector Machines Lab

Dan Burke

9/19/2021

Submitted: 9/21/2021

Step 1: Load the data

Let go back and analyze the air quality dataset (if you remember, we used that previously, in the visualization lab). Remember to think about how to deal with the NAs in the data.

```
#Load the Air Quality Data Set
data("airquality")
```

Step 2: Create train and test data sets Using techniques discussed in class, create two datasets – one for training and one for testing.

```
airQual <- airquality


airQual <- na.omit(airQual)

randIndex <- sample(1:dim(airQual)[1])
summary(airQual)
```

```
##      Ozone           Solar.R           Wind             Temp
##  Min.   :  1.0   Min.   :  7.0   Min.   : 2.30   Min.   :57.00
##  1st Qu.: 18.0   1st Qu.:113.5   1st Qu.: 7.40   1st Qu.:71.00
##  Median : 31.0   Median :207.0   Median : 9.70   Median :79.00
##  Mean   : 42.1   Mean   :184.8   Mean   : 9.94   Mean   :77.79
##  3rd Qu.: 62.0   3rd Qu.:255.5   3rd Qu.:11.50   3rd Qu.:84.50
##  Max.   :168.0   Max.   :334.0   Max.   :20.70   Max.   :97.00
##      Month            Day
##  Min.   :5.000   Min.   : 1.00
##  1st Qu.:6.000   1st Qu.: 9.00
##  Median :7.000   Median :16.00
##  Mean   :7.216   Mean   :15.95
##  3rd Qu.:9.000   3rd Qu.:22.50
##  Max.   :9.000   Max.   :31.00
```

```
length(airQual)
```

```
## [1] 6
```

```
#table(airQual)


#create 2/3 cut point
cutPoint2_3 <- floor(2* dim(airQual)[1]/3)
#verify cut point
cutPoint2_3
```

```
## [1] 74
```

```
#create training dataset
trainData <- airQual[randIndex[1:cutPoint2_3],]

#create test data set
testData <- airQual[randIndex[(cutPoint2_3+1):dim(airQual)[1]],]
```

Step 3: Build a Model using KSVM & visualize the results

1) Build a model (using the 'ksvm' function, trying to predict onzone). You can use all the possible attributes, or select the attributes that you think would be the most helpful.

2) Test the model on the testing dataset, and compute the Root Mean Squared Error

3) Plot the results. Use a scatter plot. Have the x-axis represent temperature, the y-axis represent wind, the point size and color represent the error, as defined by the actual ozone level minus the predicted ozone level).

4) Compute models and plot the results for 'svm' (in the e1071 package) and 'lm'.Generate similar charts for each model

5) Show all three results (charts) in one window, using the grid.arrange function

```
library(kernlab)
```

```
## Warning: package 'kernlab' was built under R version 4.1.1
```

```
#Build a model (using the 'ksvm' function, trying to predict onzone).
KvsmOzone <-ksvm(Ozone~., data=trainData, kernel="rbfdot",kpar="automatic",C=21, prob.model=TRUE )

KvsmOzone
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: eps-svr  (regression)
##  parameter : epsilon = 0.1  cost C = 21
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.209624626625756
##
## Number of Support Vectors : 62
##
## Objective Function Value : -103.1768
## Training error : 0.02696
## Laplace distr. width : 43.2223
```

```r
#Test the model on the testing dataset, and compute the Root Mean Squared Error

KvsmOzonePred <- predict(KvsmOzone, testData, type="votes")
kvsmError <- (testData$Ozone - KvsmOzonePred)
kvsmSquareRootError <- sqrt(mean(kvsmError^2))
paste("KVSM Square Root Error: ", kvsmSquareRootError, " " )
```

```
## [1] "KVSM Square Root Error:  20.2547296342838  "
```

```r
#Build a model (using the 'ksvm' function, trying to predict onzone).

library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.1.1
```

```r
svmOzone <-svm(Ozone~., data=trainData)

svmOzone
```

```
##
## Call:
## svm(formula = Ozone ~ ., data = trainData)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.2
##     epsilon:  0.1
##
##
## Number of Support Vectors:  64
```

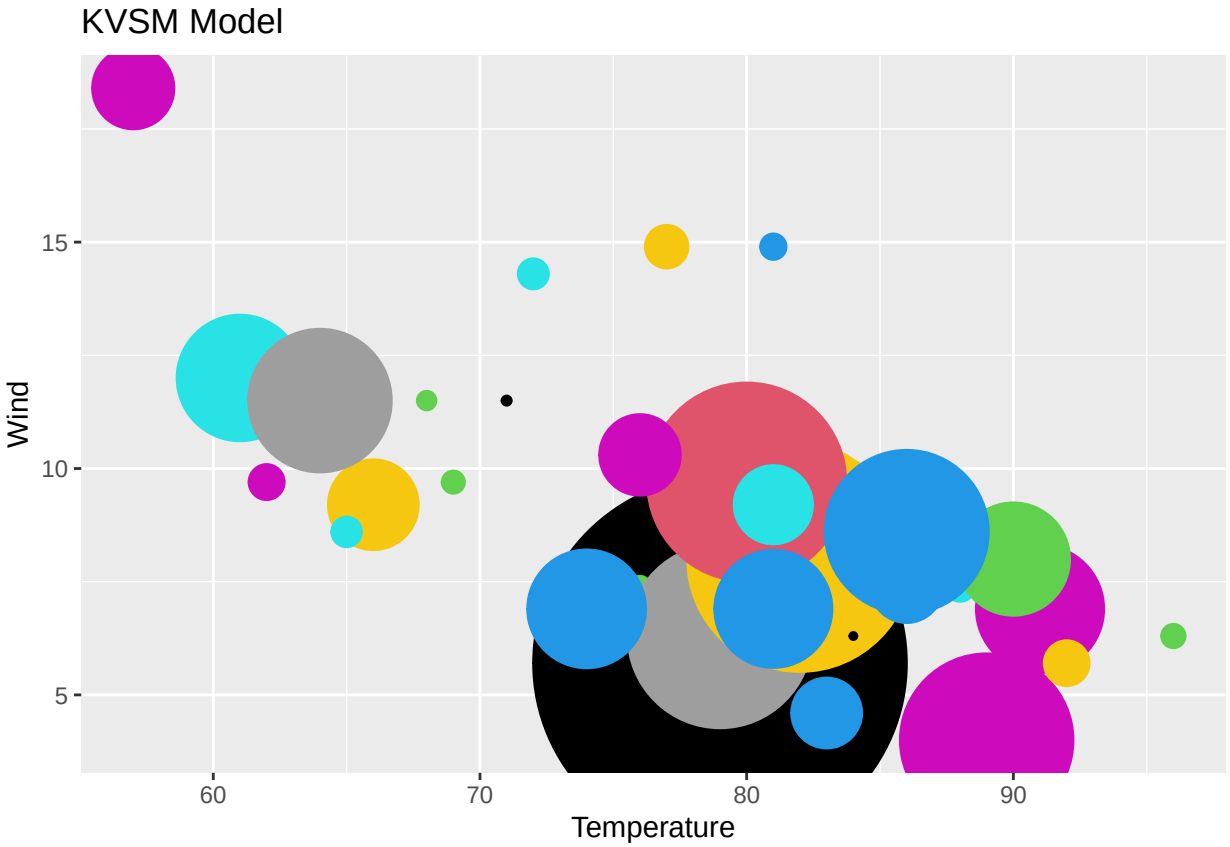
```r
#Test the model on the testing dataset, and compute the Root Mean Squared Error

svmOzonePred <- predict(svmOzone, testData, type="votes")
svmError <- (testData$Ozone - svmOzonePred)
ozoneSvmsquareRootError <- sqrt(mean(svmError^2))
paste("SVM Square Root Error: ", ozoneSvmsquareRootError, " " )
```

```
## [1] "SVM Square Root Error:  19.1995106686193  "
```

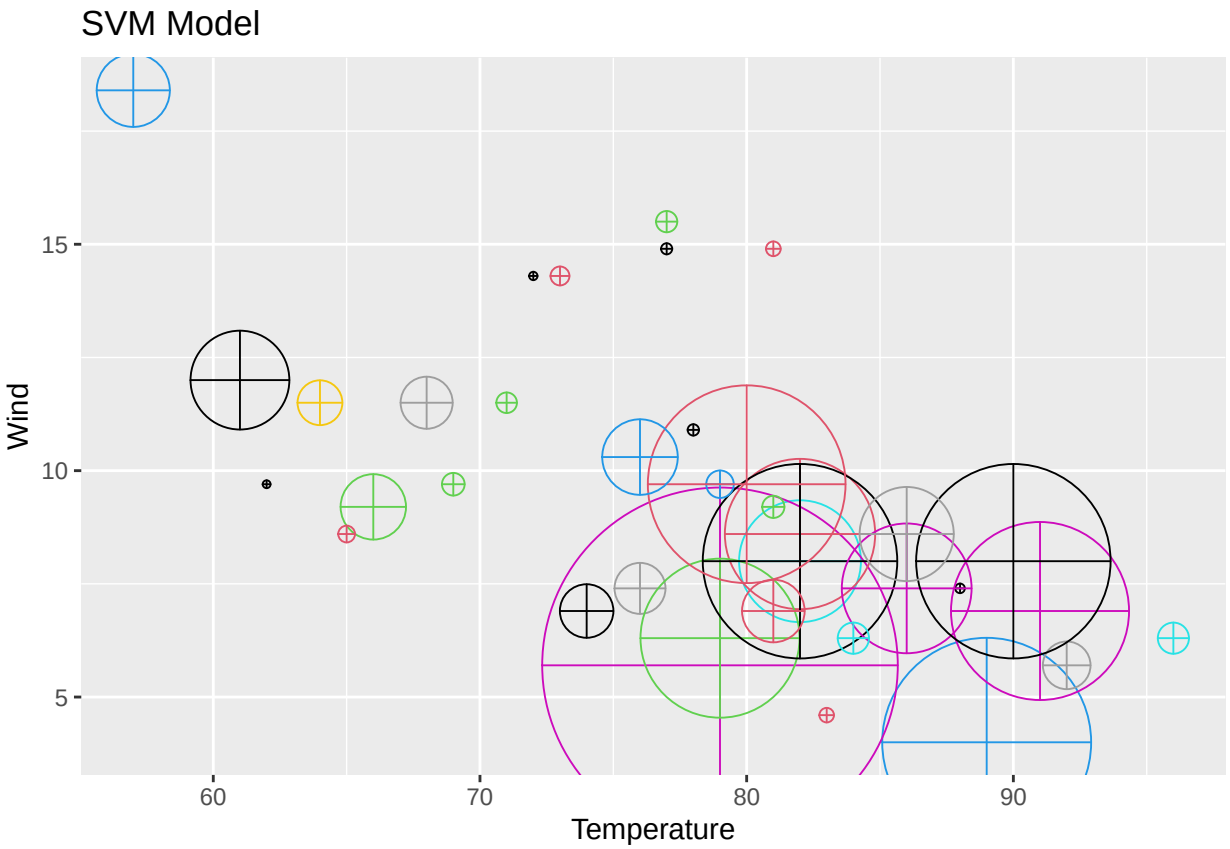
```r
#Plot the results. Use a scatter plot.
#Have the
#x-axis represent temperature,
#the y-axis represent wind,
#the point size and color represent the error,
#as defined by the actual ozone level minus the predicted ozone level).

#KVSM Model Scatter Plot
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:kernlab':
##
##     alpha
```

```
abskvsmError<- abs(kvsmError)
kvsmOzonePlot <- ggplot(data = testData, aes(x=testData$Temp, y=testData$Wind)) + geom_point(color=absk

kvsmOzonePlot <- kvsmOzonePlot + ggtitle("KVSM Model ") + xlab("Temperature") + ylab("Wind")
kvsmOzonePlot
```



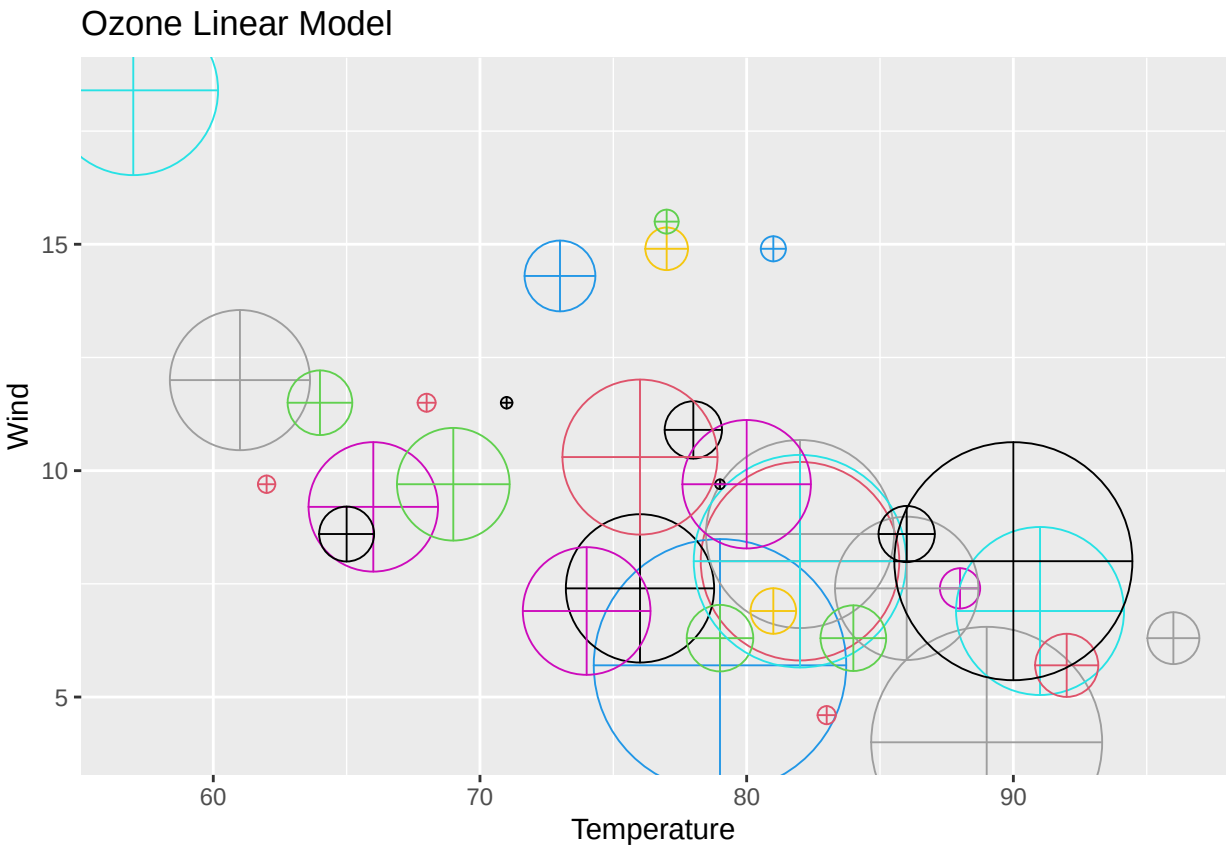

```
#Plot with Error as point color and size
absSvmError <- abs(svmError)
ozoneSvmModel <- ggplot(data = testData, aes(x=testData$Temp, y=testData$Wind)) + geom_point(shape=10,s

ozoneSvmModel <- ozoneSvmModel + ggtitle("SVM Model") + xlab("Temperature") + ylab("Wind")
ozoneSvmModel
```




```
#Train the LM Model
lmModel <- lm(Ozone~., data=airQual)
OzoneLmPredict <- predict(lmModel, testData)

#Find the Error
ozoneLinearModelError <- (testData$Ozone - OzoneLmPredict)

#Find the Square Root Error
OzoneLmsquareRootError <- sqrt(mean(ozoneLinearModelError^2))
paste("Linear Model Square Root Error: ", OzoneLmsquareRootError, " " )
```

```
## [1] "Linear Model Square Root Error:  20.9867081275873  "
```

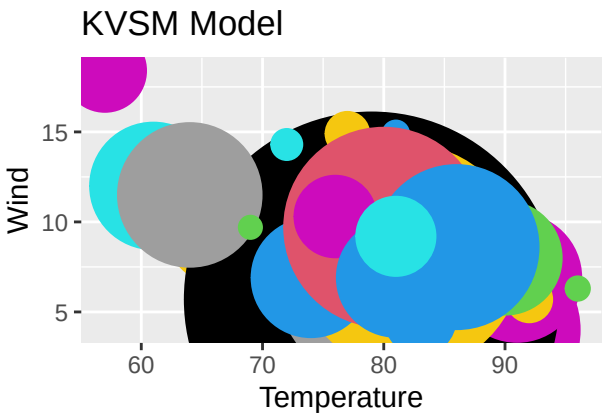
```
#Plot the Linear Model

#Get the Absolute Value of the Error
linearModelErrorAbs <- abs(ozoneLinearModelError)

OzoneLinearModel <- ggplot(data = testData, aes(x=testData$Temp, y=testData$Wind)) + geom_point(shape=1
OzoneLinearModel
```

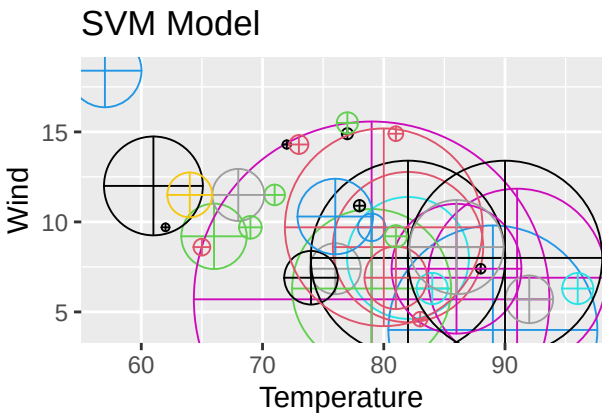


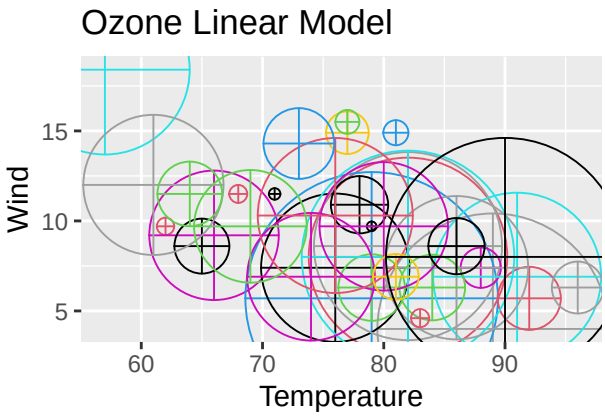

```
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 4.1.1
```

```
library(grid)
grid.arrange(kvsmOzonePlot, ozoneSvmModel, OzoneLinearModel, ncol=2)
```







Step 4: Create a 'goodOzone' variable

This variable should be either 0 or 1. It should be 0 if the ozone is below the average for all the data observations, and 1 if it is equal to or above the average ozone observed.

```
meanAirOzone <- mean(airQual$Ozone)
paste("Mean of overall Ozone is: ", meanAirOzone, " ")
```

```
## [1] "Mean of overall Ozone is:  42.0990990990991  "
```

```
for(i in 1:nrow(airQual)){

  if(airQual$Ozone[i]<meanAirOzone){
    airQual$goodOzone[i] <-  0
  }else if(airQual$Ozone[i]>=meanAirOzone){
    airQual$goodOzone[i] <- 1
  }
}


airQual$goodOzone <- as.numeric(airQual$goodOzone)
```

Step 5: See if we can do a better job predicting 'good' and 'bad' days

1) Build a model (using the 'ksvm' function, trying to predict 'goodOzone'). You can use all the possible attributes, or select the attributes that you think would be the most helpful.

2) Test the model on the testing dataset, and compute the percent of 'goodOzone' that was correctly predicted.

3) Plot the results. Use a scatter plot. Have the x-axis represent temperature, the y-axis represent wind, the shape representing what was predicted (good or bad day), the color representing the actual value of 'goodOzone' (i.e. if the actual ozone level was good) and the size represent if the prediction was correct (larger symbols should bethe observations the model got wrong).

4) Compute models and plot the results for 'svm' (in the e1071 package) and 'nb' (Naive Bayes, also in the e1071 package).

5) Show all three results (charts) in one window, using the grid.arrange function (have two charts in one row).

```
#create training dataset with the new column
trainData <- airQual[randIndex[1:cutPoint2_3],]

#create test data set with the new column
testData <- airQual[randIndex[(cutPoint2_3+1):dim(airQual)[1]],]


#Build a model (using the 'ksvm' function, trying to predict 'goodOzone')

kvsmGoodDayModel <- ksvm(goodOzone~., data=trainData )
kvsmGoodDayModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: eps-svr  (regression)
##  parameter : epsilon = 0.1  cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.130226434000133
##
## Number of Support Vectors : 48
##
## Objective Function Value : -17.9944
## Training error : 0.176711
```

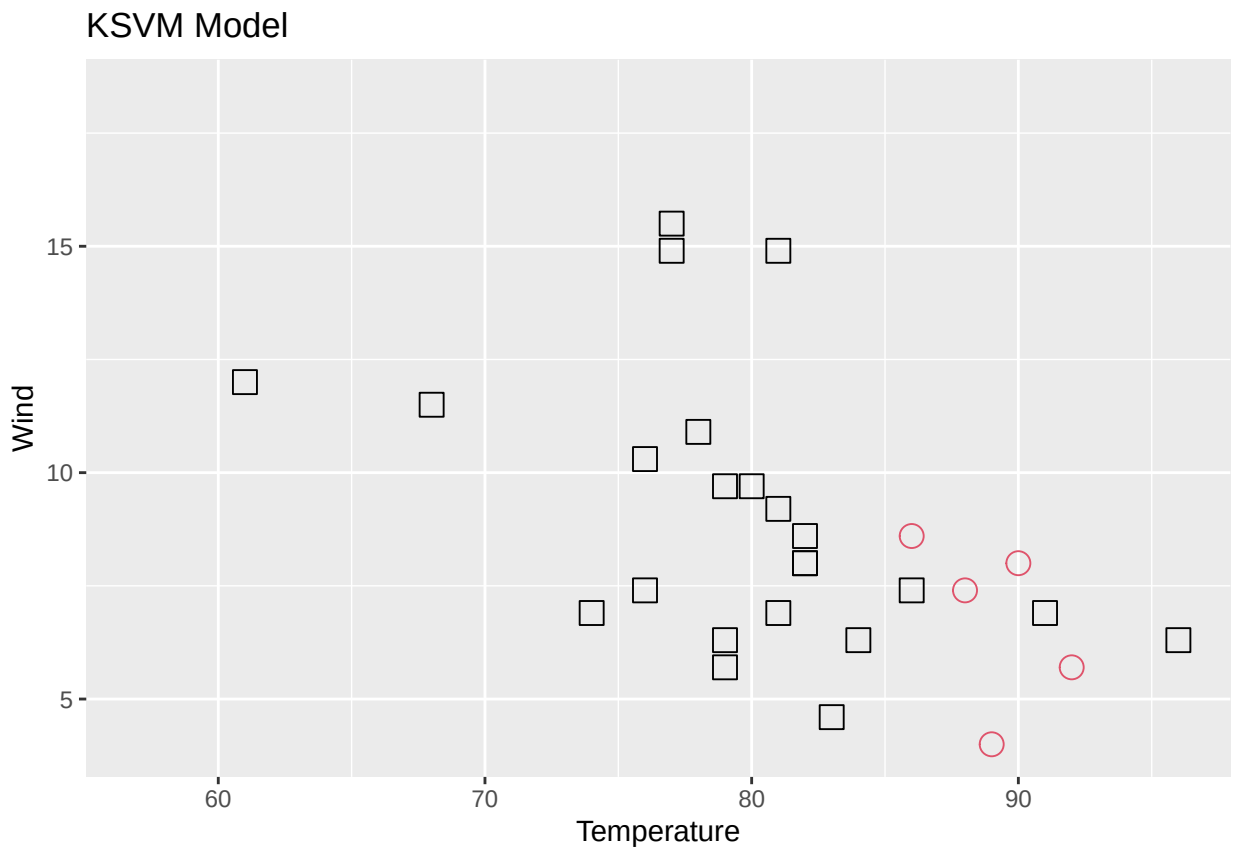
```
#Test the model on the testing dataset, and compute the percent of 'goodOzone' that was cor
goodDayPredict <- predict(kvsmGoodDayModel, testData, type="votes")
```

```
goodDayCorrectPercentValues <-  (as.numeric(testData$goodOzone) - as.numeric(goodDayPredict))

#Get the Absolute Value
goodDayCorrectPercentValues <- abs(round(goodDayCorrectPercentValues))

goodDayCorrectPercent <- (1 - sum(goodDayCorrectPercentValues[goodDayCorrectPercentValues != 0])/length
paste("Percent Correct KSVM: ", goodDayCorrectPercent, " ")
```

```
## [1] "Percent Correct KSVM:  0.918918918918919  "
```

```
correctValues <- vector()


for(i in 1:length(goodDayCorrectPercentValues)){
   if(goodDayCorrectPercentValues[i]==0){
     correctValues[i] <- "correct"
   }else{
     correctValues[i] <- "incorrect"
   }

}




#Plot the KVS Mthe  shape   representing    what    was predicted   (good   or  bad day),   the color

goodDayKsvmModelPlot <- ggplot(data = testData, aes(x=testData$Temp, y=testData$Wind)) + geom_point(siz

goodDayKsvmModelPlot <- goodDayKsvmModelPlot + ggtitle("KSVM Model ") + xlab("Temperature") + ylab("Win
goodDayKsvmModelPlot
```

```
## Warning in Ops.factor(coords$size, .pt): '*' not meaningful for factors
```




```
#Second Model

#Build  a   model   (using  the 'svm'   function,   trying  to  predict 'goodOzone')

svmGoodDayModel <- svm(goodOzone~., data=trainData )
svmGoodDayModel
```

```
##
## Call:
## svm(formula = goodOzone ~ ., data = trainData)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.1666667
##     epsilon:  0.1
##
##
## Number of Support Vectors:  49
```

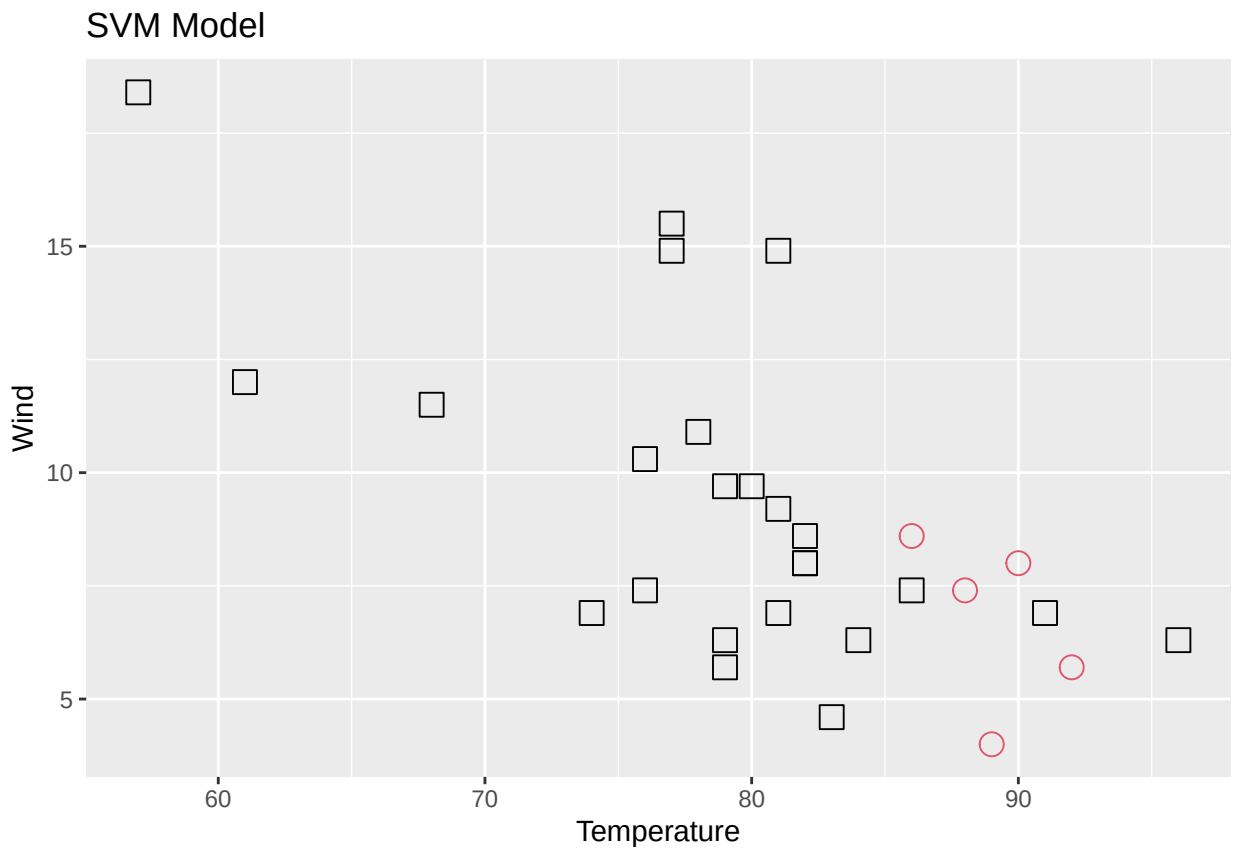
```
#Test   the model   on  the testing dataset,    and compute the percent of  'goodOzone' that    was cor
svmGoodDayPredict <- predict(svmGoodDayModel, testData, type="class")
svmGoodDayCorrectPercentValues <-  (as.numeric(testData$goodOzone) - as.numeric(svmGoodDayPredict))
```

```r
#Get the Absolute Value
svmGoodDayCorrectPercentValues <- abs(round(svmGoodDayCorrectPercentValues))

svmGoodDayCorrectPercent <- (1 - sum(svmGoodDayCorrectPercentValues[svmGoodDayCorrectPercentValues != 0]
paste("Percent Correct SVM: ", svmGoodDayCorrectPercent, " ")
```

```
## [1] "Percent Correct SVM:  0.945945945945946  "
```

```r
svmCorrectValues <- vector()


for(i in 1:length(svmGoodDayCorrectPercentValues)){
   if(svmGoodDayCorrectPercentValues[i]==0){
     svmCorrectValues[i] <- "correct"
   }else{
     svmCorrectValues[i] <- "incorrect"
   }

}


goodDaySvmModelPlot <- ggplot(data = testData, aes(x=testData$Temp, y=testData$Wind)) + geom_point(size=

goodDaySvmModelPlot <- goodDaySvmModelPlot + ggtitle("SVM Model ") + xlab("Temperature") + ylab("Wind")
goodDaySvmModelPlot
```

```
## Warning in Ops.factor(coords$size, .pt): '*' not meaningful for factors
```




```
#Third Model

GoodDayModelThree <- naiveBayes(goodOzone~., data=trainData )
GoodDayModelThree
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##         0         1
## 0.6351351 0.3648649
##
## Conditional probabilities:
##    Ozone
## Y       [,1]     [,2]
##   0 21.19149 10.23345
##   1 75.88889 30.35854
##
##    Solar.R
## Y       [,1]      [,2]
##   0 159.2340 104.78678
##   1 215.5556  48.59514
```

```
##
##    Wind
## Y       [,1]     [,2]
##   0 11.47447 3.106058
##   1  7.97037 3.542144
##
##    Temp
## Y       [,1]     [,2]
##   0 72.59574 7.609089
##   1 86.88889 5.146570
##
##    Month
## Y       [,1]     [,2]
##   0 7.297872 1.730982
##   1 7.629630 1.005682
##
##    Day
## Y       [,1]      [,2]
##   0 14.72340  8.139282
##   1 15.66667 10.209347
```

```
#Test   the model   on  the testing dataset,    and compute the percent of  'goodOzone' that    was cor
ModelThreePredict <- predict(GoodDayModelThree, testData, type="class")
modelThreeCorrectPercentValues <-  (as.numeric(testData$goodOzone) - as.numeric(ModelThreePredict))
ModelThreePredict
```

```
##  [1] 0 0 0 1 0 1 0 0 0 1 1 1 1 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1
## Levels: 0 1
```

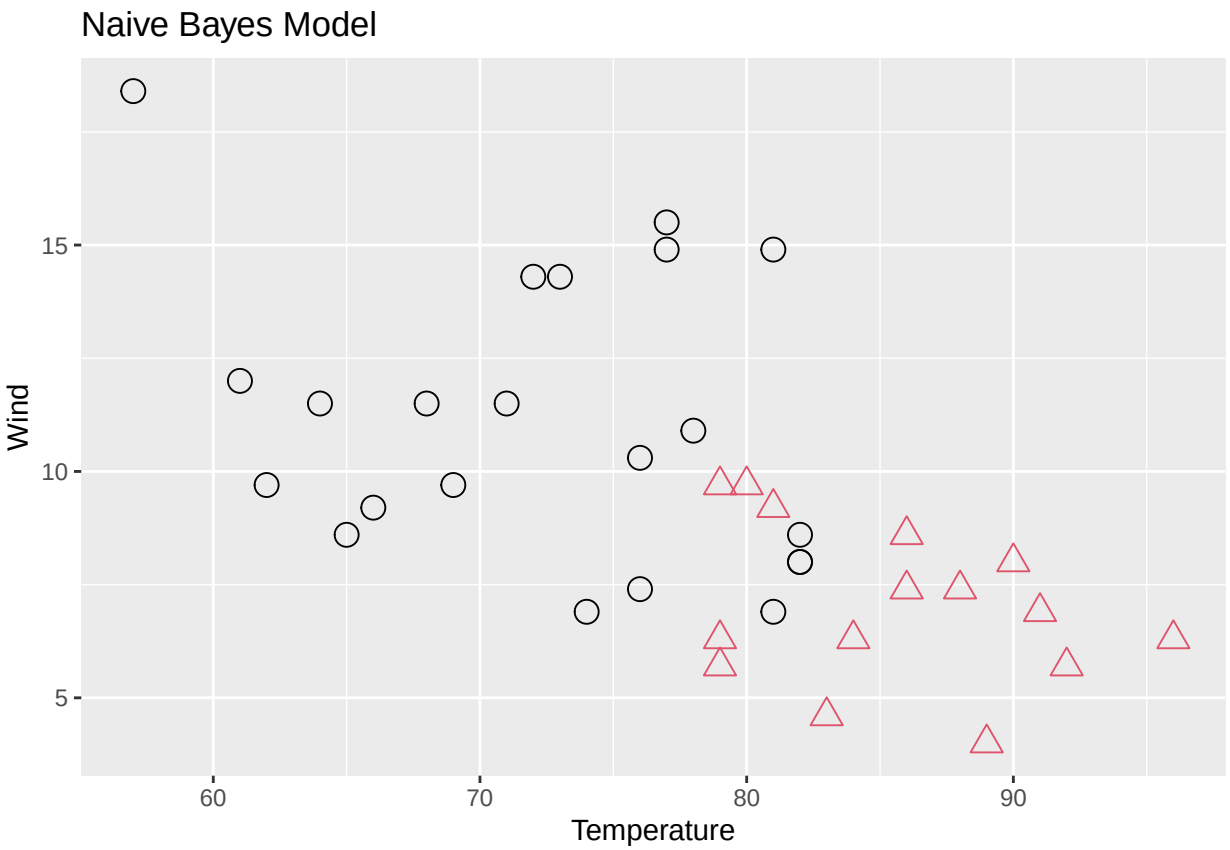
```
#Get the Absolute Value
modelThreeCorrectPercentValues <- abs(round(modelThreeCorrectPercentValues))

modelThreeCorrectPercent <- (1 - sum(modelThreeCorrectPercentValues[modelThreeCorrectPercentValues != 0
paste("Third Model Percent Correct: ", modelThreeCorrectPercent, " ")
```

```
## [1] "Third Model Percent Correct:  0  "
```

```
modelThreeCorrectValues <- vector()


for(i in 1:length(modelThreeCorrectPercentValues)){
   if(goodDayCorrectPercentValues[i]==0){
     modelThreeCorrectValues[i] <- "correct"
   }else{
     modelThreeCorrectValues[i] <- "incorrect"
   }

}


modelThreePlot <- ggplot(data = testData, aes(x=testData$Temp, y=testData$Wind))
modelThreePlot <- modelThreePlot + geom_point(size=as.factor(modelThreeCorrectValues), color=abs(as.num
```

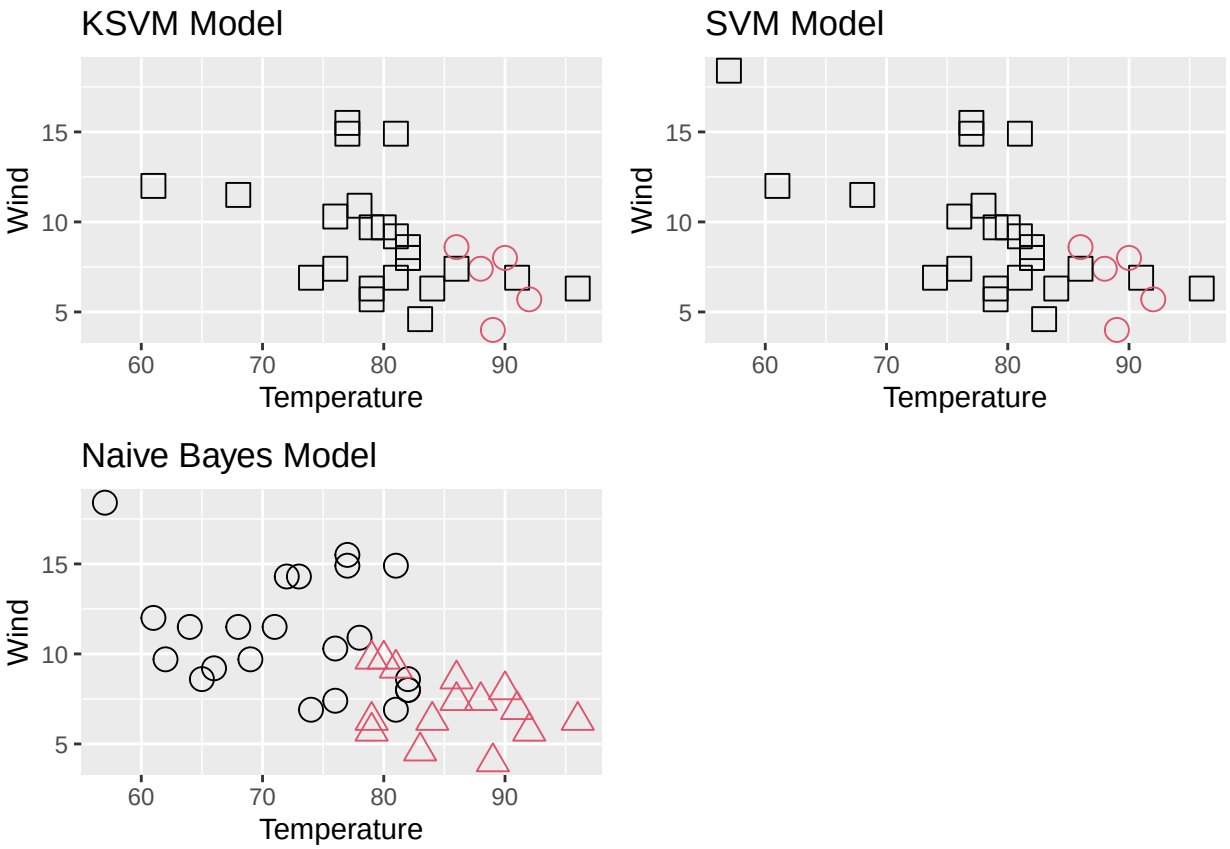
```
modelThreePlot <- modelThreePlot + ggtitle("Naive Bayes Model") + xlab("Temperature") + ylab("Wind")
modelThreePlot
```

```
## Warning in Ops.factor(coords$size, .pt): '*' not meaningful for factors
```




```
grid.arrange(goodDayKsvmModelPlot, goodDaySvmModelPlot, modelThreePlot, ncol=2)
```

```
## Warning in Ops.factor(coords$size, .pt): '*' not meaningful for factors

## Warning in Ops.factor(coords$size, .pt): '*' not meaningful for factors

## Warning in Ops.factor(coords$size, .pt): '*' not meaningful for factors
```



Step 6: Which are the best Models for this data?

Regarding the prediction of a "good" or "bad" ozone level on any day, the SVM (KSVM & SVM) models are where I place most trust. This is because SVM models search for the interactions between features. Within the context of dynamic weather, data features are interdependent. Understanding the relationships/patterns of interaction between our columns and rows; if even only from a general scope, will allow a better telling of the data's story. The Naïve Bayes model treats features as independent, which in this context and with this data, places us at a disadvantaged understanding.