Apurva Shrivastava
ISEN 613 – 600
925009508

Final_Exam – Part #B

Convention:

- R script (code) is shown in Black box
- Console output is shown in Blue box
- Plots are shown in Red box

***Data preparation steps*:**

- Creating variable “Count01” on the basis of “Count”:

```
count01 = rep("Low", nrow(Bike_data))
count01[Bike_data$count > 724] = "High"
```

- Removing column count and adding the newly created column count01 in the dataset

```
Bike_data = Bike_data[,-11]
Bike_data = cbind(Bike_data, count01)
summary(Bike_data)
```

```
     season          year            month           holiday          weekday        weathersit         temp              atemp               hum        
 Min.   :1.000   Min.   :0.0000   Min.   : 1.000   Min.   :0.00000   Min.   :0.000   Min.   :1.000   Min.   :0.05913   Min.   :0.07907   Min.   :0.0000  
 1st Qu.:2.000   1st Qu.:0.0000   1st Qu.: 4.000   1st Qu.:0.00000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:0.33875   1st Qu.:0.33815   1st Qu.:0.5200  
 Median :3.000   Median :1.0000   Median : 7.000   Median :0.00000   Median :3.000   Median :1.000   Median :0.50250   Median :0.48926   Median :0.6258  
 Mean   :2.502   Mean   :0.5021   Mean   : 6.529   Mean   :0.02954   Mean   :3.003   Mean   :1.392   Mean   :0.49608   Mean   :0.47506   Mean   :0.6272  
 3rd Qu.:3.000   3rd Qu.:1.0000   3rd Qu.:10.000   3rd Qu.:0.00000   3rd Qu.:5.000   3rd Qu.:2.000   3rd Qu.:0.65500   3rd Qu.:0.60798   3rd Qu.:0.7296  
 Max.   :4.000   Max.   :1.0000   Max.   :12.000   Max.   :1.00000   Max.   :6.000   Max.   :3.000   Max.   :0.86167   Max.   :0.84090   Max.   :0.9725  
   windspeed        count01  
 Min.   :0.02239   High:355  
 1st Qu.:0.13434   Low :356  
 Median :0.17972             
 Mean   :0.19027             
 3rd Qu.:0.23321             
 Max.   :0.50746             
> 
```

- Factoring for qualitative predictors

```
Bike_data$season <- factor(Bike_data$season)
Bike_data$year <- factor(Bike_data$year)
Bike_data$month <- factor(Bike_data$month)
Bike_data$holiday <- factor(Bike_data$holiday)
Bike_data$weekday <- factor(Bike_data$weekday)
Bike_data$weathersit <- factor(Bike_data$weathersit)
```

- Splitting data into training and test data sets

```
createTrainingData <- function(dataset, sampleTrain){
  m <- nrow(dataset)
  set.seed(1)
  train <- sample(m, as.integer(sampleTrain*m))
  trainingData <- dataset[train,]
  return(trainingData)
}
createTestData <- function(dataset, sampleTest){
  train <- as.integer(row.names(createTrainingData(dataset, 1 - sampleTest)))
  testData <- dataset[-train,]
  return(testData)
}
sampleTrain <- 600/nrow(Bike_data)
bikeTrain <- createTrainingData(Bike_data, sampleTrain)
bikeTest <- createTestData(Bike_data, 1- sampleTrain)
```

Apurva Shrivastava
ISEN 613 – 600
925009508

- Created method prediction performance as these would be used repeatedly for each type of classification problem

```
errorCheck <- function(prediction, actual){
  confusionTable <- table(prediction,actual)
  print(confusionTable)
  accuracy <- mean(prediction==actual)
  errorRate <- 1 - accuracy
  TruePositive = confusionTable[2,2]
  TrueNegative = confusionTable[1,1]
  FalsePositive = confusionTable[2,1]
  FalseNegative = confusionTable[1,2]
  Positives = FalseNegative + TruePositive
  Negatives = FalsePositive + TrueNegative
  sensitivity <- TruePositive/Positives
  specificity <- 1 - FalsePositive/Negatives
  performanceTable <- matrix(c(errorRate,accuracy,sensitivity,specificity))
  rownames(performanceTable) <- c("Error Rate", "Accuracy", "Sensitivity","Specificity")
  return(performanceTable)
}
```

Explanation for methods:

- The errorCheck method is creating confusion table for the prediction and calculating Accuracy, Error Rate, Sensitivity and Specificity according to the formulas in the method and returning the table as output.

---

1. Fit a logistic regression model for the training data. Interpret the fitted model. Find its prediction performance (prediction accuracy, sensitivity, specificity) on the test data. (Note: Show formulas and calculations on each performance measure.)

```
Count01Fit <- glm(count01 ~ ., data = bikeTrain,  family = binomial)
summary(Count01Fit)
```

```
Call:
glm(formula = count01 ~ ., family = binomial, data = bikeTrain)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.39008  -0.28009  -0.00804   0.13082   2.88026

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)    5.5328     1.8837   2.937 0.003312 **
season2       -0.9936     0.9196  -1.081 0.279914
season3       -3.4427     1.4928  -2.306 0.021097 *
season4       -3.9308     1.6971  -2.316 0.020545 *
year1         -2.8961     0.4018  -7.209 5.65e-13 ***
month2        -0.9351     1.7586  -0.532 0.594898
month3        -6.1155     1.5046  -4.064 4.82e-05 ***
month4        -6.2123     1.7678  -3.514 0.000441 ***
month5        -5.5537     1.8438  -3.012 0.002594 **
month6        -5.1444     1.8968  -2.712 0.006685 **
month7        -1.6007     2.2246  -0.720 0.471788
month8        -3.1405     2.2150  -1.418 0.156233
month9        -2.7350     2.1251  -1.287 0.198098
month10       -3.3519     2.2021  -1.522 0.127974
month11       -1.2015     2.1595  -0.556 0.577956
month12        0.5490     2.1518   0.255 0.798635
holiday1      -1.9271     1.0927  -1.764 0.077784 .
weekday1       4.7023     0.7977   5.895 3.75e-09 ***
weekday2       5.7954     0.8592   6.745 1.53e-11 ***
weekday3       6.3662     0.8999   7.074 1.50e-12 ***
weekday4       5.6281     0.8449   6.661 2.71e-11 ***
weekday5       3.8248     0.7583   5.044 4.56e-07 ***
```

```
weekday5       3.8248     0.7583   5.044 4.56e-07 ***
weekday6      -1.1014     0.7663  -1.437 0.150647
weathersit2    1.3542     0.4479   3.023 0.002499 **
weathersit3   16.0805   717.2471   0.022 0.982113
temp         -12.1763    10.8907  -1.118 0.263550
atemp         -2.6124    11.9534  -0.219 0.827002
hum            5.1799     1.8405   2.814 0.004886 **
windspeed      8.0661     2.5342   3.183 0.001458 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 831.45  on 599  degrees of freedom
Residual deviance: 265.18  on 571  degrees of freedom
AIC: 323.18

Number of Fisher Scoring iterations: 16
```

Apurva Shrivastava
ISEN 613 – 600
925009508

```
Count01Fit <- glm(count01 ~ ., data = bikeTrain,  family = binomial)
summary(Count01Fit)
Count01cProbs <- predict(Count01Fit, bikeTest, type = 'response')
Count01Prediction <- ifelse(Count01cProbs > 0.5, "High", "Low")
Count01LogisticErrorTable <- errorCheck(Count01Prediction, bikeTest$count01)
colnames(Count01LogisticErrorTable) <- "Logistic"
print(Count01LogisticErrorTable)
```

Prediction using formulas in errorCheck() method:

```
          actual
prediction High Low
      High    6  57
      Low    42   6
> colnames(Count01LogisticErrorTable) <- "Logistic"
> print(Count01LogisticErrorTable)
             Logistic
Error Rate  0.8918919
Accuracy    0.1081081
Sensitivity 0.0952381
Specificity 0.1250000
>
```

Interpretation:

We can see that in the model fit **AIC (Akaike Information Criterian)** is **323** which means the test error rate is very high and the model is not good. This becomes evident from the prediction as Accuracy for logistic regression is **10.8%,** and error rate is **89%** which is very high.

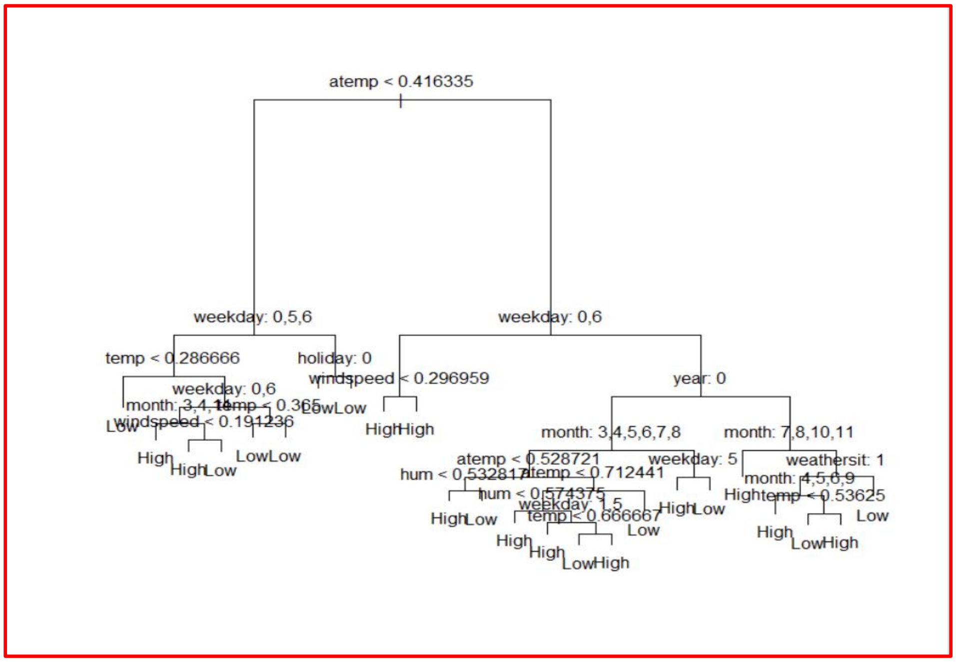
----

2. Fit a tree model for the training data considering both prediction performance and interpretability (that is, your model must be good in prediction and easy to interpret). Interpret the fitted model. Find its prediction performance (prediction accuracy, sensitivity, specificity) on the test data. (Note: Justify your choices, if any, in the model building process.)

```
library(tree)
library(boot)
library(randomForest)
treeBikes <- tree(count01 ~ ., data = bikeTrain)
summary(treeBikes)
plot(treeBikes)
text(treeBikes, pretty = FALSE)
```

```
Classification tree:
tree(formula = count01 ~ ., data = bikeTrain)
Variables actually used in tree construction:
[1] "atemp"      "weekday"    "temp"       "month"      "windspeed"  "holiday"    "year"       "hum"        "weathersit"
Number of terminal nodes:  24
Residual mean deviance:  0.3213 = 185.1 / 576
Misclassification error rate: 0.075 = 45 / 600
```

Apurva Shrivastava
ISEN 613 – 600
925009508



Prediction using formulas in <u>errorCheck()</u> method:

```
treePreds1 <- predict(treeBikes, bikeTest, type = "class")
treeErrorTable <- errorCheck(treePreds1, bikeTest$count01)
colnames(treeErrorTable) <- "Tree"
print(treeErrorTable)
```

```
          actual
prediction High Low
      High   40  12
      Low     8  51
> colnames(treeErrorTable) <- "Tree"
> print(treeErrorTable)
                 Tree
Error Rate  0.1801802
Accuracy    0.8198198
Sensitivity 0.8095238
Specificity 0.8333333
>
```

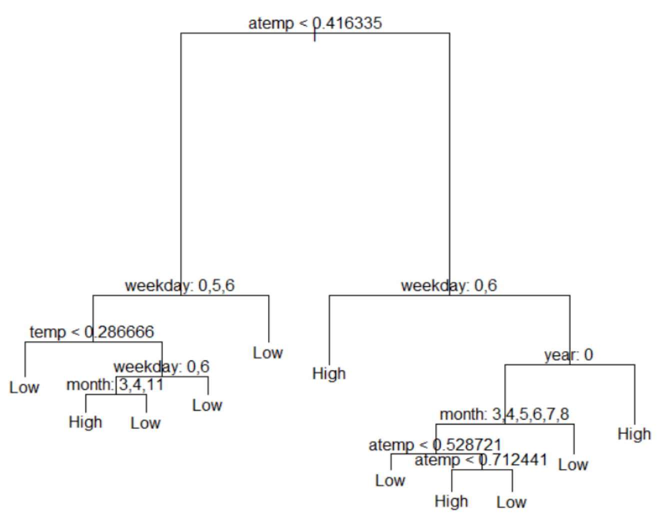
We can see that accuracy of tree is **81.98%**

To improve predictability and interpretability, we will **prune** the tree and analyze if it is giving better results:

```
cvbikeTree <- cv.tree(treeBikes, FUN = prune.misclass)
bestTree <- cvbikeTree$size[which.min(cvbikeTree$dev)]
prunedTreeBikes <- prune.misclass(treeBikes, best = bestTree)
summary(prunedTreeBikes)
plot(prunedTreeBikes)
text(prunedTreeBikes, pretty = FALSE)
```

Apurva Shrivastava
ISEN 613 – 600
925009508

```
Classification tree:
snip.tree(tree = treeBikes, nodes = c(5L, 6L, 19L, 29L, 56L,
15L, 114L, 37L))
Variables actually used in tree construction:
[1] "atemp"   "weekday" "temp"    "month"   "year"
Number of terminal nodes:  11
Residual mean deviance:  0.598 = 352.2 / 589
Misclassification error rate: 0.1067 = 64 / 600
```



Prediction using formulas in errorCheck() method:

```
treePrunedErrorTable = predict(prunedTreeBikes, bikeTest, type = "class")
treePruneErrorTable <- errorCheck(treePrunedErrorTable, bikeTest$count01)
colnames(treePruneErrorTable) <- "TreePrune"
print(treePruneErrorTable)
```

```
          actual
prediction High Low
      High   43  12
      Low     5  51
> colnames(treePruneErrorTable) <- "TreePrune"
> print(treePruneErrorTable)
            TreePrune
Error Rate  0.1531532
Accuracy    0.8468468
Sensitivity 0.8095238
Specificity 0.8958333
>
```

Interpretation:

The tree with all predictors is difficult to interpret however the pruned version has only 11 nodes and hence **better interpretability**. Also, the **prediction accuracy** for pruned tree is **84.6% as compared to 81.98%** for normal tree. As a result, from the pruned tree it can be clearly seen that on weekends (weekday: 0,6 -> Saturday and Sunday) when normalized temperature is > 0.416 the user count for bike sharing is high. However, when temperature is < 0.2866, even on weekends (weekday: 0,5,6 -> Friday to Sunday) the user count for bike sharing is low. The other nodes can be read in a similar way.

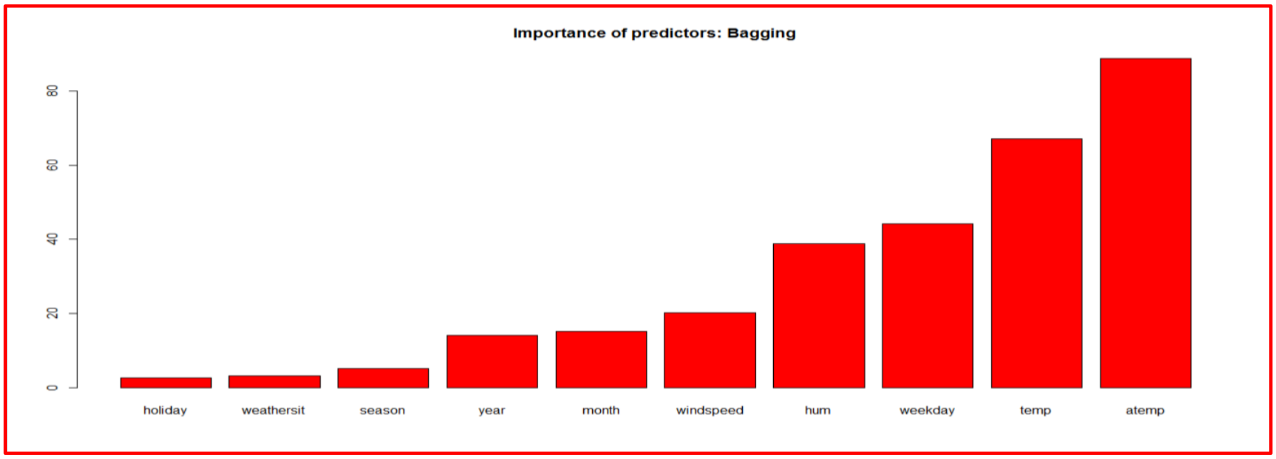
To improve predictability and interpretability further, we can apply advanced ensemble tree methods of Bagging and Random Forest:

#Bagging

```
set.seed(1)
p <- ncol(Bike_data) - 1
set.seed(1)
bagBikes <- randomForest(count01 ~ ., data = bikeTrain, ntree = 100, mtry = p, importance = TRUE)
bagPreds <- predict(bagBikes, bikeTest)
bagErrorTable <- errorCheck(bagPreds, bikeTest$count01)
colnames(bagErrorTable) <- "Bagging"
print(bagErrorTable)
```

```
          actual
prediction High Low
      High   43   7
      Low     5  56
> colnames(bagErrorTable) <- "Bagging"
> print(bagErrorTable)
              Bagging
Error Rate  0.1081081
Accuracy    0.8918919
Sensitivity 0.8888889
Specificity 0.8958333
>
```

```
barplot(sort(bagBikes$importance[,"MeanDecreaseGini"]), col = 'Red',main = "Importance of predictors: Bagging" )
```



Interpretation:

After bagging, we can see that the **prediction accuracy** has increased to **89.18%** which is an improvement over the pruned tree. However, bagging averages over bootstrapped trees and hence **interpretability is low**. From the bar plot it could be seen that the normalized freezing temperature and the normal temperature are the most important predictors.

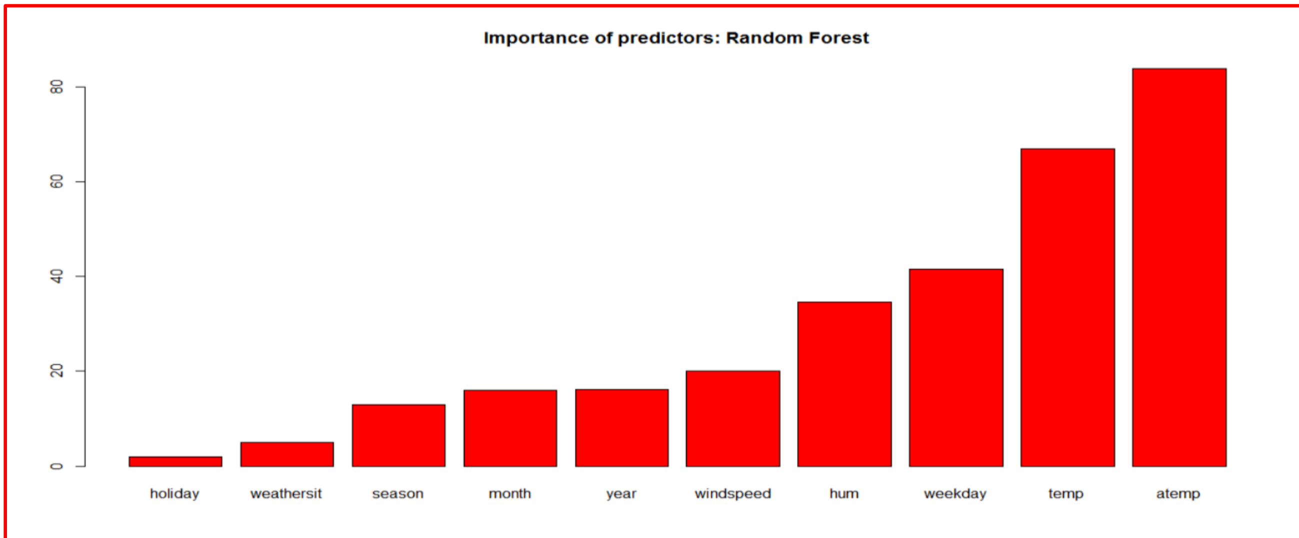
Apurva Shrivastava
ISEN 613 – 600
925009508

#Random Forest

```
p <- ncol(Bike_data) - 1
set.seed(1)
rfBikes <- randomForest(count01 ~ ., data = bikeTrain, ntree = 100, mtry = sqrt(p), importance = TRUE)
rfPreds <- predict(rfBikes, bikeTest)
rfErrorTable <- errorCheck(rfPreds, bikeTest$count01)
colnames(rfErrorTable) <- "Random Forest"
print(rfErrorTable)
```

```
          actual
prediction High Low
      High   44  10
      Low     4  53
> colnames(rfErrorTable) <- "Random Forest"
> print(rfErrorTable)
            Random Forest
Error Rate      0.1261261
Accuracy        0.8738739
Sensitivity     0.8412698
Specificity     0.9166667
>
```

```
barplot(sort(rfBikes$importance[,"MeanDecreaseGini"]), col = 'Red', main = "Importance of predictors: Random Forest")
```



Interpretation:

Random Forest also improves the **prediction accuracy** over the pruned tree (87.38%). Although, it's accuracy is less than bagging by a small margin, Random Forests tends to de-correlates the data and hence it's **interpretability is more** than bagging. For Random Forest also, the normalized freezing temperature and the normal temperature are the most important predictors.

---

3. List all the other methods you have learned in this course that can be used for this dataset. For each of those methods, apply it on the training data and then find its prediction performance (prediction accuracy, sensitivity, specificity) on the test data.

Name of methods learned that can be used for this dataset:

1. LDA

2. QDA
3. Support Vector Classifier (SVC)
4. Support Vector Machine (SVM) – Radial Kernel
5. Support Vector Machine (SVM) – Poly Kernel

#LDA

```
library(MASS)
set.seed(1)
ldaBikes <- lda(count01 ~ ., data = bikeTrain)
ldaProbs <- predict(ldaBikes, bikeTest, type = 'response')
ldaPreds <- ldaProbs$class
ldaErrorTable <- errorCheck(ldaPreds, bikeTest$count01)
colnames(ldaErrorTable) <- "LDA"
print(ldaErrorTable)
```

```
           actual
prediction High Low
      High   45   9
      Low     3  54
> colnames(ldaErrorTable) <- "LDA"
> print(ldaErrorTable)
                  LDA
Error Rate  0.1081081
Accuracy    0.8918919
Sensitivity 0.8571429
Specificity 0.9375000
>
```

#QDA

```
set.seed(1)
qdaBikes <- qda(count01 ~ ., data = bikeTrain)
qdaProbs <- predict(qdaBikes, bikeTest, type = 'response')
qdaPreds <- qdaProbs$class
qdaErrorTable <- errorCheck(qdaPreds, bikeTest$count01)
colnames(qdaErrorTable) <- "QDA"
print(qdaErrorTable)
```

```
           actual
prediction High Low
      High   38  11
      Low    10  52
> colnames(qdaErrorTable) <- "QDA"
> print(qdaErrorTable)
                  QDA
Error Rate  0.1891892
Accuracy    0.8108108
Sensitivity 0.8253968
Specificity 0.7916667
>
```

#SVC

Apurva Shrivastava
ISEN 613 – 600
925009508

```
library(e1071)
i <- -3:2
costs <- 10^i
gammas <- seq(0.5,5,by = 0.5)
degrees <- i[5:6]
svcTune <- tune(svm, count01 ~ ., data = bikeTrain, kernel = 'linear', ranges = list(cost = costs))
print(summary(svcTune))
```

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
   10

- best performance: 0.1233333
```

```
svcBikes <- svm(count01 ~ ., data = bikeTrain,kernel = 'linear', cost = 10, scale = FALSE)
print(summary(svcBikes))
```

```
svcPreds <- predict(svcBikes, bikeTest)
svcErrorTable <- errorCheck(svcPreds, bikeTest$count01)
colnames(svcErrorTable) <- "SVC"
print(svcErrorTable)
```

```
          actual
prediction High Low
      High   44   6
      Low     4  57
> colnames(svcErrorTable) <- "SVC"
> print(svcErrorTable)
                   SVC
Error Rate  0.09009009
Accuracy    0.90990991
Sensitivity 0.90476190
Specificity 0.91666667
>
```

#SVM Radial

```
svmRadialTune <- tune(svm, count01 ~ ., data = bikeTrain,kernel = 'radial', ranges = list(cost = costs, gamma = gammas))
print(summary(svmRadialTune))
```

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    1   0.5

- best performance: 0.1416667
```

```
svmRadialBikes <- svm(count01 ~ ., data = bikeTrain,kernel = 'radial', gamma = 0.5, cost = 1, scale = FALSE)
svmRadialPreds <- predict(svmRadialBikes, bikeTest)
svmRadialErrorTable <- errorCheck(svmRadialPreds, bikeTest$count01)
colnames(svmRadialErrorTable) <- "SVM Radial"
print(svmRadialErrorTable)
```

Apurva Shrivastava
ISEN 613 – 600
925009508

```
           actual
prediction High Low
      High   41  12
      Low     7  51
> colnames(svmRadialErrorTable) <- "SVM Radial"
> print(svmRadialErrorTable)
            SVM Radial
Error Rate   0.1711712
Accuracy     0.8288288
Sensitivity  0.8095238
Specificity  0.8541667
>
```

#SVM Poly

```
svmPolyTune <- tune(svm, count01 ~ ., data = bikeTrain, kernel = 'polynomial', ranges = list(cost = costs, degree = degrees))
print(summary(svmPolyTune))
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost degree
  100      1

- best performance: 0.12
```

```
svmPolyBikes <- svm(count01 ~ ., data = bikeTrain,kernel = 'polynomial', degree = 1, cost = 10, scale = FALSE)
svmPolyPreds <- predict(svmPolyBikes, bikeTest)
svmPolyErrorTable <- errorCheck(svmPolyPreds, bikeTest$count01)
colnames(svmPolyErrorTable) <- "SVM Poly"
print(svmPolyErrorTable)
```

```
           actual
prediction High Low
      High   45   8
      Low     3  55
> colnames(svmPolyErrorTable) <- "SVM Poly"
> print(svmPolyErrorTable)
             SVM Poly
Error Rate  0.0990991
Accuracy    0.9009009
Sensitivity 0.8730159
Specificity 0.9375000
>
```

4. Summarize the prediction performance of all methods in a table. Which method is the best? Why?

```
SummaryErrorTable <- cbind(Count01LogisticErrorTable, treeErrorTable, treePruneErrorTable, rfErrorTable, bagErrorTable,
                           ldaErrorTable, qdaErrorTable, svcErrorTable, svmRadialErrorTable, svmPolyErrorTable)
print(SummaryErrorTable)
```

	Logistic	Tree	TreePrune	Random Forest	Bagging	LDA	QDA	SVC	SVM Radial	SVM Poly
Error Rate	0.8918919	0.1801802	0.1531532	0.1261261	0.1081081	0.1081081	0.1891892	0.09009009	0.1891892	0.0990991
Accuracy	0.1081081	0.8198198	0.8468468	0.8738739	0.8918919	0.8918919	0.8108108	0.90990991	0.8108108	0.9009009
Sensitivity	0.0952381	0.8095238	0.8095238	0.8412698	0.8888889	0.8571429	0.8253968	0.90476190	0.8095238	0.8730159
Specificity	0.1250000	0.8333333	0.8958333	0.9166667	0.8958333	0.9375000	0.7916667	0.91666667	0.8125000	0.9375000

As it can be seen, **SVC** has the least error rate (9%) among all the models and has a sensitivity of 90.4% and specificity of 91.6%. Hence, it is the best model for the given data set.