VIT-AP UNIVERSITY, ANDHRA PRADESH

CSE2047 – Data Analytics - Lab Sheet : 10

Academic year: 2020-2021 **Branch/ Class:** B.Tech/M.Tech
Semester: Fall **Date:**
Faculty Name: Prof. S.Gopikrishnan **School:** SCOPE
Student name: Valiveti Manikanta bhuvanesh **Reg. no.: 19BCD7088**

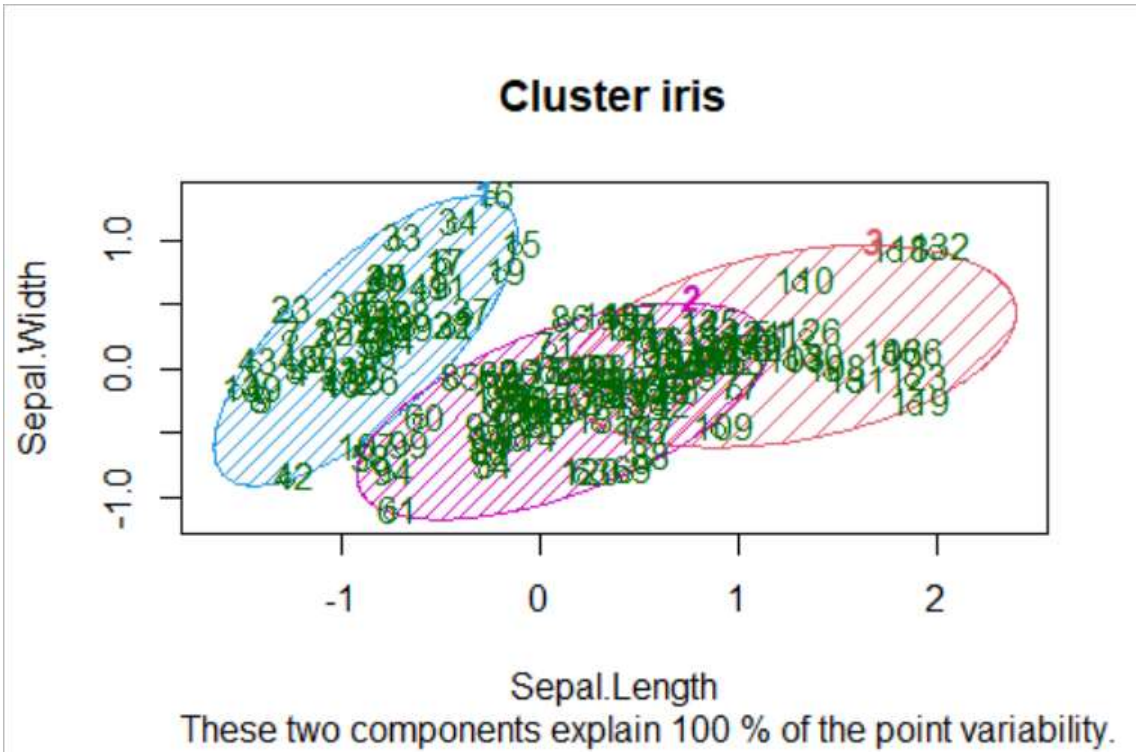
LAB 10

Clustering and Classification Algorithm implementation using R.

1. **Use iris dataset for k means clustering in R**

```
df <- iris[, -5]
set.seed(240)
kmeans.re <- kmeans(df, centers = 3, nstart = 20)
kmeans.re
kmeans.re$cluster
plot(df[c("Sepal.Length", "Sepal.Width")])
plot(df[c("Sepal.Length", "Sepal.Width")], col = kmeans.re$cluster)
plot(df[c("Sepal.Length", "Sepal.Width")], col = kmeans.re$cluster, main = "K-
means with 3 clusters")
kmeans.re$centers
kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")]
points(kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")], col = 1:3, pch =
8, cex = 3)
y_kmeans <- kmeans.re$cluster
clusplot(df[, c("Sepal.Length", "Sepal.Width")],y_kmeans,lines = 0,shade =
TRUE,color = TRUE,labels = 2,
        plotchar = FALSE,
        span = TRUE,
        main = paste("Cluster iris"),
        xlab = 'Sepal.Length',
        ylab = 'Sepal.Width')
```

```
> df <- iris[, -5]
> set.seed(240)
> kmeans.re <- kmeans(df, centers = 3, nstart = 20)
> kmeans.re
K-means clustering with 3 clusters of sizes 50, 62, 38

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053

Clustering vector:
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [42] 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2
 [83] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3 2 2 3 3 3 2 3 2 3
[124] 2 3 3 2 2 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3 3 2

Within cluster sum of squares by cluster:
[1] 15.15100 39.82097 23.87947
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
> kmeans.re$cluster
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [42] 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2
 [83] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3 2 2 3 3 3 2 3 2 3
[124] 2 3 3 2 2 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3 3 2
> plot(df[c("Sepal.Length", "Sepal.Width")])
> plot(df[c("Sepal.Length", "Sepal.Width")], col = kmeans.re$cluster)
> plot(df[c("Sepal.Length", "Sepal.Width")], col = kmeans.re$cluster, main = "K-means with 3 clusters")
> kmeans.re$centers
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053
> kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")]
  Sepal.Length Sepal.Width
1     5.006000    3.428000
2     5.901613    2.748387
3     6.850000    3.073684
> points(kmeans.re$centers[, c("Sepal.Length", "Sepal.Width")], col = 1:3, pch = 8, cex = 3)
> y_kmeans <- kmeans.re$cluster
> clusplot(df[, c("Sepal.Length", "Sepal.Width")],y_kmeans,lines = 0,shade = TRUE,color = TRUE,labels = 2,
+          plotchar = FALSE,
+          span = TRUE,
+          main = paste("Cluster iris"),
+          xlab = 'Sepal.Length',
+          ylab = 'Sepal.Width')
> |
```



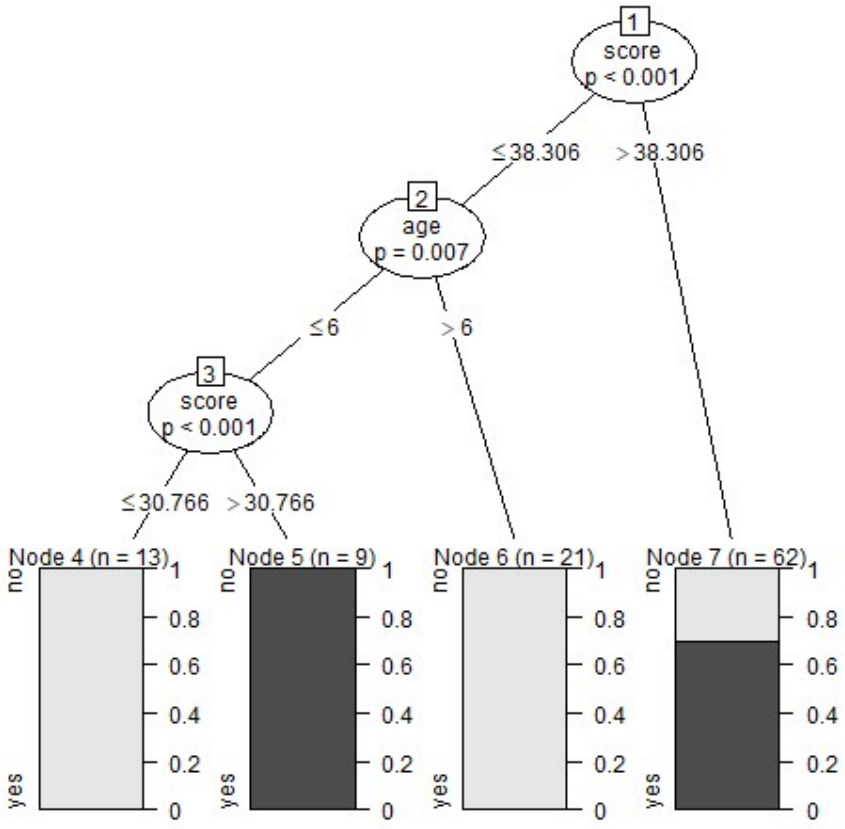
2. **Use readingSkills dataset for all classification practice which is default in party package**
   a. **Logistic regression**

```
df<-readingSkills[c(1:105), ]
split <- sample.split(df, SplitRatio = 0.8)
split
train_reg <- subset(df, split == "TRUE")
test_reg <- subset(df, split == "FALSE")
logistic_model <- glm(nativeSpeaker ~ age + shoeSize + score, data =
train_reg, family = "binomial")
summary(logistic_model)
predict_reg <- predict(logistic_model,test_reg, type = "response")
predict_reg
predict_reg <- ifelse(predict_reg >0.75, 1, 0)
table(test_reg$nativeSpeaker, predict_reg)
missing_classerr <- mean(predict_reg != test_reg$nativeSpeaker)
print(paste('Accuracy =', 1 - missing_classerr))
```

```
> df<-readingSkills[c(1:105), ]
> split <- sample.split(df, SplitRatio = 0.8)
> split
[1]  TRUE  TRUE FALSE  TRUE
> train_reg <- subset(df, split == "TRUE")
> test_reg <- subset(df, split == "FALSE")
> logistic_model <- glm(nativeSpeaker ~ age + shoeSize + score, data = train_reg, family = "binomial")
Warning messages:
1: glm.fit: algorithm did not converge
2: glm.fit: fitted probabilities numerically 0 or 1 occurred
> summary(logistic_model)

Call:
glm(formula = nativeSpeaker ~ age + shoeSize + score, family = "binomial",
    data = train_reg)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-1.862e-05  -2.110e-08   2.110e-08   2.110e-08   1.891e-05

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -4.442e+01  1.289e+06   0.000    1.000
age         -4.503e+01  8.619e+04  -0.001    1.000
shoeSize    -2.772e+00  6.245e+04   0.000    1.000
score        1.178e+01  1.453e+04   0.001    0.999

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1.0920e+02  on 78  degrees of freedom
Residual deviance: 9.2204e-10  on 75  degrees of freedom
AIC: 8

Number of Fisher Scoring iterations: 25

> predict_reg <- predict(logistic_model,test_reg, type = "response")
> predict_reg
           3            7           11           15           19           23
2.220446e-16 2.220446e-16 1.000000e+00 2.220446e-16 2.220446e-16 2.220446e-16
          27           31           35           39           43           47
1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 2.220446e-16 1.000000e+00
          51           55           59           63           67           71
2.220446e-16 1.000000e+00 2.220446e-16 2.220446e-16 2.220446e-16 1.000000e+00
          75           79           83           87           91           95
1.320527e-13 2.220446e-16 2.220446e-16 3.556603e-13 2.220446e-16 1.000000e+00
          99          103
1.000000e+00 2.220446e-16
> predict_reg <- ifelse(predict_reg >0.75, 1, 0)
> table(test_reg$nativeSpeaker, predict_reg)
     predict_reg
       0  1
  no  16  0
  yes  0 10
> missing_classerr <- mean(predict_reg != test_reg$nativeSpeaker)
> print(paste('Accuracy =', 1 - missing_classerr))
[1] "Accuracy = 0"
>
```

b. Decision trees

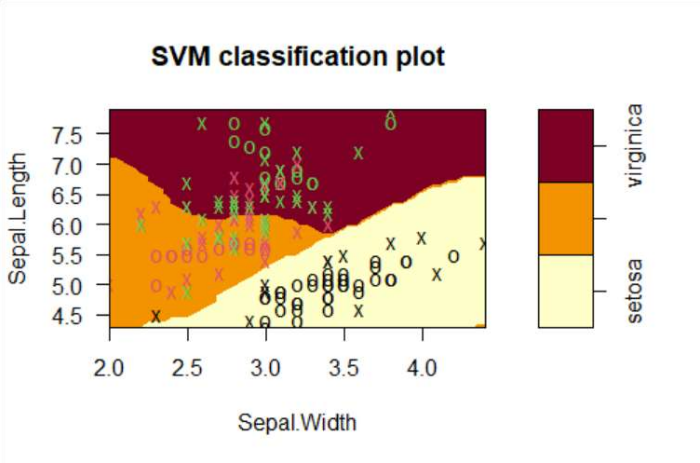
```
png(file = "decision_tree.png")
output.tree <- ctree(nativeSpeaker ~ age + shoeSize + score,data = df)
plot(output.tree)
dev.off()
```



c. Support Vector Machines (iris data – default)

df = iris[,c(1,2,5)]
model <- svm(Species ~ ., data=df)
summary(model)
final_svm <- svm(Species ~ ., data=df, kernel="radial", cost=1,gamma=1)
plot(final_svm , df)

```
> df = iris[,c(1,2,5)]
> model <- svm(Species ~ ., data=df)
> summary(model)

Call:
svm(formula = Species ~ ., data = df)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  86

 ( 10 40 36 )


Number of Classes:  3

Levels:
 setosa versicolor virginica



> final_svm <- svm(Species ~ ., data=df, kernel="radial", cost=1,gamma=1)
> plot(final_svm , df)
>
```



d. Naive Bayes Classifier (use hsbdata.csv)

```
df<-hsb
set.seed(7267166)
trainIndex=createDataPartition(df$prog, p=0.7)$Resample1
train=df[trainIndex, ]
test=df[-trainIndex, ]
print(table(df$prog))
NBclassfier=naiveBayes(prog~science+socst, data=train)
print(NBclassfier)
Print=function(model){
  trainPred=predict(model, newdata = train, type = "class")
  trainTable=table(train$prog, trainPred)
  testPred=predict(NBclassfier, newdata=test, type="class")
  testTable=table(test$prog, testPred)

trainAcc=(trainTable[1,1]+trainTable[2,2]+trainTable[3,3])/sum(trainTable)
  testAcc=(testTable[1,1]+testTable[2,2]+testTable[3,3])/sum(testTable)
  message("Contingency Table for Training Data")
  print(trainTable)
  message("Contingency Table for Test Data")
  print(testTable)
  message("Accuracy")
  print(round(cbind(trainAccuracy=trainAcc, testAccuracy=testAcc),3))
}
Print(NBclassfier)
print(table(train$prog))
newNBclassifier=naive_bayes(prog~ses+science+socst,usekernel=T,data
=train)
Print(newNBclassifier)
```

```
> df<-hsb
> set.seed(7267166)
> trainIndex=createDataPartition(df$prog, p=0.7)$Resample1
> train=df[trainIndex, ]
> test=df[-trainIndex, ]
> print(table(df$prog))

  1   2   3
 45 105  50
> NBclassfier=naiveBayes(prog~science+socst, data=train)
> print(NBclassfier)

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
        1         2         3
0.2269504 0.5248227 0.2482270

Conditional probabilities:
   science
Y       [,1]     [,2]
  1 51.50000 9.758371
  2 53.16216 9.391087
  3 47.28571 9.341783

   socst
Y       [,1]     [,2]
  1 49.96875 9.351503
  2 56.91892 9.338031
  3 46.85714 9.213618

> Print=function(model){
+   trainPred=predict(model, newdata = train, type = "class")
+   trainTable=table(train$prog, trainPred)
+   testPred=predict(NBclassfier, newdata=test, type="class")
+   testTable=table(test$prog, testPred)
+   trainAcc=(trainTable[1,1]+trainTable[2,2]+trainTable[3,3])/sum(trainTable)
+   testAcc=(testTable[1,1]+testTable[2,2]+testTable[3,3])/sum(testTable)
+   message("Contingency Table for Training Data")
+   print(trainTable)
+   message("Contingency Table for Test Data")
+   print(testTable)
+   message("Accuracy")
+   print(round(cbind(trainAccuracy=trainAcc, testAccuracy=testAcc),3))
+ }
> Print(NBclassfier)
Contingency Table for Training Data
    trainPred
      1  2  3
  1   0 21 11
  2   1 60 13
  3   1 15 19
Contingency Table for Test Data
    testPred
      1  2  3
  1   0 10  3
  2   0 26  5
  3   1  4 10
Accuracy
     trainAccuracy testAccuracy
[1,]          0.56         0.61
> print(table(train$prog))

 1  2  3
32 74 35
> newNBclassifier=naive_bayes(prog~ses+science+socst,usekernel=T,data=train)
> Print(newNBclassifier)
Contingency Table for Training Data
    trainPred
      1  2  3
  1   6 20  6
  2   4 59 11
  3   5 11 19
Contingency Table for Test Data
    testPred
      1  2  3
  1   0 10  3
  2   0 26  5
  3   1  4 10
Accuracy
     trainAccuracy testAccuracy
[1,]         0.596         0.61
Warning message:
predict.naive_bayes(): more features in the newdata are provided as there are
s performed based on features to be found in the tables.
```

e. **k-Nearest Neighbour (iris data)**

```
df = iris
df = df[-c(1,8)]
iris_tr_feat <- df[,1:4]
set.seed(1)
train_pred <- knn(iris_tr_feat, iris_tr_feat, df$Species, k=3)
train_pred[1:10]
accuracy <- mean(train_pred == df$Species)
cat("Training Accuracy: ", accuracy, sep='')
```

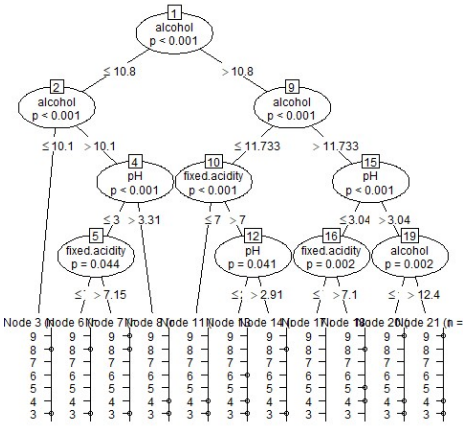
3. **Use winequality dataset for all classification practice and use quality as predictor variable**
   a. **Logistic regression**

```
df <- read.csv('winequality.csv')
df <- df[,c(1,9,11,12)]
split <- sample.split(df, SplitRatio = 0.8)
split
train_reg <- subset(df, split == "TRUE")
test_reg <- subset(df, split == "FALSE")
logistic_model <- glm( quality ~ fixed.acidity+pH + alcohol,data = df)
logistic_model
summary(logistic_model)
predict_reg <- predict(logistic_model,test_reg, type = "response")
predict_reg
predict_reg <- ifelse(predict_reg >0.5, 1, 0)
table(test_reg$quality, predict_reg)
missing_classerr <- mean(predict_reg != test_reg$quality)
print(paste('Accuracy =', 1 - missing_classerr))
```

```
> df <- read.csv('winequality.csv')
> df <- df[,c(1,9,11,12)]
> split <- sample.split(df, SplitRatio = 0.8)
> split
[1]  TRUE  TRUE  TRUE FALSE
> train_reg <- subset(df, split == "TRUE")
> test_reg <- subset(df, split == "FALSE")
> logistic_model <- glm( quality ~ fixed.acidity+pH + alcohol,data = df)
> logistic_model

Call:  glm(formula = quality ~ fixed.acidity + pH + alcohol, data = df)

Coefficients:
  (Intercept)  fixed.acidity             pH        alcohol
      2.53585       -0.05364        0.15192        0.30676

Degrees of Freedom: 4897 Total (i.e. Null);  4894 Residual
Null Deviance:      3841
Residual Deviance: 3096         AIC: 11660
> summary(logistic_model)

Call:
glm(formula = quality ~ fixed.acidity + pH + alcohol, data = df)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.5338  -0.5211  -0.0139   0.4882   3.2758

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   2.535850   0.333473   7.604 3.41e-14 ***
fixed.acidity -0.053636   0.014930  -3.593 0.000331 ***
pH             0.151922   0.083440   1.821 0.068708 .
alcohol        0.306760   0.009332  32.872  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 0.63254)

    Null deviance: 3841.0  on 4897  degrees of freedom
Residual deviance: 3095.7  on 4894  degrees of freedom
AIC: 11663

Number of Fisher Scoring iterations: 2

> predict_reg <- predict(logistic_model,test_reg, type = "response")
> predict_reg
       4        8       12       16       20       24       28       32       36
5.671225 5.315651 5.527186 6.172661 5.590623 5.444445 5.894873 5.746791 6.555464
      40       44       48       52       56       60       64       68       72
5.270112 5.585260 5.703300 6.063630 6.015636 5.686324 5.783280 5.659770 5.589725
      76       80       84       88       92       96      100      104      108
5.528678 5.434113 5.564426 5.635885 5.443506 5.439068 5.390556 5.348666 5.426623
     112      116      120      124      128      132      136      140      144
5.341676 5.767899 5.428341 5.457100 5.686918 5.686918 5.616649 6.211873 5.538627
     148      152      156      160      164      168      172      176      180
5.466915 5.642174 5.308161 6.479194 5.308161 6.509582 5.962000 6.135301 5.499401
     184      188      192      196      200      204      208      212      216
5.593768 6.072310 5.285067 5.442899 5.376804 6.358143 5.479932 6.111216 5.611166
     220      224      228      232      236      240      244      248      252
5.969781 5.498304 5.428142 5.542985 5.379949 5.676890 5.648038 6.062190 5.544995
     4000
5.870843
 [ reached getOption("max.print") -- omitted 224 entries ]
> predict_reg <- ifelse(predict_reg >0.5, 1, 0)
> table(test_reg$quality, predict_reg)
   predict_reg
      1
  3   4
  4  41
  5 349
  6 558
  7 230
  8  41
  9   1
> missing_classerr <- mean(predict_reg != test_reg$quality)
> print(paste('Accuracy =', 1 - missing_classerr))
[1] "Accuracy = 0"
>
```

b. Decision trees

```
png(file = "decision_tree1.png")
output.tree <- ctree(quality ~ fixed.acidity + pH + alcohol,data = df)
plot(output.tree)
dev.off()
```



c. Support Vector Machines

model <- svm(quality ~., data=df)
summary(model)
final_svm <- svm(quality ~., data=df, kernel="radial", cost=1,gamma=1)
plot(final_svm , df)

```
> df <- read.csv('winequality.csv')
> model <- svm(quality ~., data=df)
> summary(model)

Call:
svm(formula = quality ~ ., data = df)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.09090909
    epsilon:  0.1


Number of Support Vectors:  4176




> final_svm <- svm(quality ~., data=df, kernel="radial", cost=1,gamma=1)
> plot(final_svm , df)
```

d. Naive Bayes Classifier

split <- sample.split(df, SplitRatio = 0.7)
trainl <- subset(df, split == "TRUE")
testl <- subset(df, split == "FALSE")
train_scale <- scale(trainl[, 1:4])
test_scale <- scale(testl[, 1:4])
set.seed(120)
classifier_cl <- naiveBayes(quality ~ ., data = trainl)
classifier_cl
y_pred <- predict(classifier_cl, newdata = testl)
cm <- table(testl$quality, y_pred)
cm

```
> split <- sample.split(df, SplitRatio = 0.7)
> trainl <- subset(df, split == "TRUE")
> testl <- subset(df, split == "FALSE")
> train_scale <- scale(trainl[, 1:4])
> test_scale <- scale(testl[, 1:4])
> set.seed(120)
> classifier_cl <- naiveBayes(quality ~ ., data = trainl)
> classifier_cl

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
          3           4           5           6           7           8           9
0.003676471 0.032781863 0.293198529 0.451286765 0.181372549 0.036458333 0.001225490

Conditional probabilities:
   fixed.acidity
Y       [,1]      [,2]
  3 8.025000 1.6613932
  4 7.177570 1.0568115
  5 6.933856 0.8316931
  6 6.850373 0.8627391
  7 6.705912 0.7534796
  8 6.642017 0.8393395
  9 7.550000 1.0847427

   volatile.acidity
Y         [,1]       [,2]
  3 0.3566667 0.15830542
  4 0.3708411 0.16064212
  5 0.2996604 0.09665111
  6 0.2596402 0.08775994
  7 0.2632855 0.09312251
  8 0.2747059 0.10874729
  9 0.2825000 0.05315073

   citric.acid
Y         [,1]       [,2]
  3 0.3391667 0.08877534
  4 0.3149533 0.15939966
  5 0.3361338 0.14032276
  6 0.3374881 0.11643570
  7 0.3222635 0.08256635
  8 0.3285714 0.08995560
  9 0.3975000 0.08995369

   residual.sugar
Y         [,1]       [,2]
  3 6.620833 5.599735
  4 4.738785 4.174707
  5 7.339133 5.377164

   density
Y         [,1]        [,2]
  3 0.9954925 0.002582026
  4 0.9942604 0.002565421
  5 0.9952415 0.002596989
  6 0.9939431 0.003060798
  7 0.9924847 0.002779174
  8 0.9924671 0.002791154
  9 0.9918750 0.003437659

   pH
Y         [,1]       [,2]
  3 3.134167 0.19695216
  4 3.173925 0.16515149
  5 3.167994 0.13826687
  6 3.191127 0.15414149
  7 3.211807 0.15894409
  8 3.220336 0.15287600
  9 3.315000 0.09398581

   sulphates
Y         [,1]       [,2]
  3 0.4850000 0.12667065
  4 0.4842991 0.12550361
  5 0.4835528 0.10160079
  6 0.4928649 0.11635284
  7 0.5067399 0.13440228
  8 0.4812605 0.15096612
  9 0.4925000 0.08220908

   alcohol
Y         [,1]      [,2]
  3  9.983333 1.0743567
  4 10.184112 1.0062212
  5  9.822114 0.8658722
  6 10.581444 1.1457502
  7 11.332303 1.2497459
  8 11.493277 1.2868784
  9 12.050000 1.1210114

> y_pred <- predict(classifier_cl, newdata = testl)
> cm <- table(testl$quality, y_pred)
> cm
   y_pred
     3   4   5   6   7   8   9
  3  3   0   0   2   3   0   0
  4  0  17  18  17   4   0   0
  5  6  20 272 151  50   1   0
  6  8   9 191 261 253   3   0
  7  0   0  29  70 188   1   0
  8  0   0   7   9  38   2   0
  9  0   0   0   0   1   0   0
>
```

e. k-Nearest Neighbour

```
df1<- df[,1:4]
set.seed(1)
train_pred <- knn(df1, df1, df$quality, k=3)
train_pred[1:10]
accuracy <- mean(train_pred == df$quality)
cat("Training Accuracy: ", accuracy, sep='')
```

```
> df1<- df[,1:4]
> set.seed(1)
> train_pred <- knn(df1, df1, df$quality, k=3)
> train_pred[1:10]
 [1] 6 6 6 6 6 6 6 6 6 6
Levels: 3 4 5 6 7 8 9
> accuracy <- mean(train_pred == df$quality)
> cat("Training Accuracy: ", accuracy, sep='')
Training Accuracy: 0.7111066
>
```