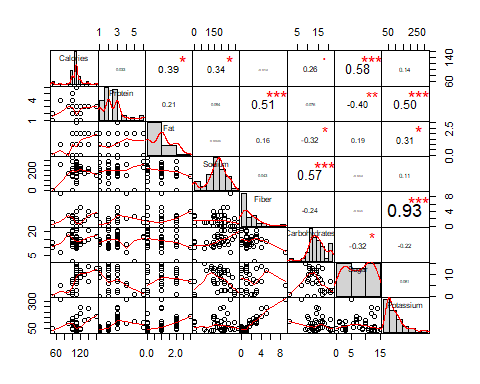
STAT 4360 Mini Project 6

Name: Jaemin Lee

Section 1: Answers to the specific question asked

Question 1

(a)

Figure 1. Pairwise scatter plot of the cereal data

From Figure 1, we see that Fiber and Patassium, Sodium and Carbohydrates, Protein and Patassium, and Calories and Sugar are strongly correlated.

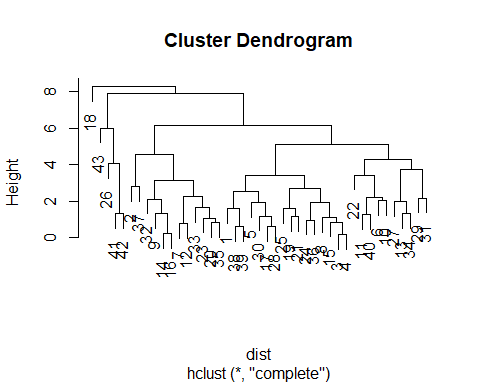
1. Standatdizing the variables before clustering would be a good idea as the predictors are measured on different scales.
2. Metric-based distance would be a good choice as metric-based distance such as euclidean distance gives you how similar or dissimilar clusters are based on all the variables. However, correlation-based distance doesn’t necessarily use all the variables when clustering. For our cereal data set, we want to use all the predictors to cluster cereals.

Figure 2. Hierarchical clustering using complete linkage and Euclidean distance

1. One can either choose 3 or 5 clusters based on Figure 2. One can cut the dendogram slightly above height 6 which results in 3 clusters. One can also cut the dendogram slightly above 4 which results in 5 clusters. If you choose 3 clusters, 18 is in its own cluster. This tells us that cereal 18 is an outlier. If you choose 5 clusters, both 18 and 43 are in its own cluster. In my opinion, I would choose 3 clusters as we don’t want to have too many clusters with only one item.

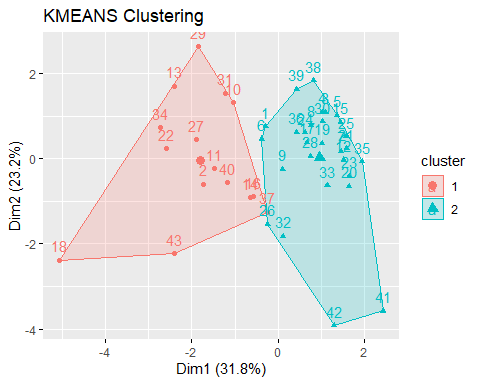
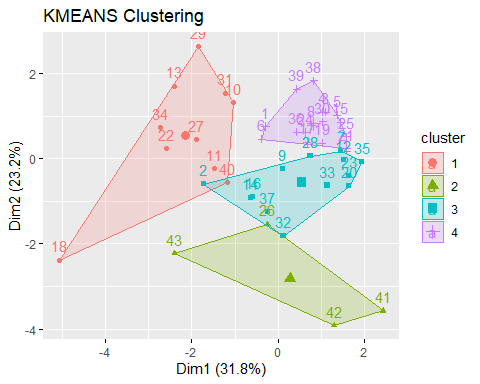
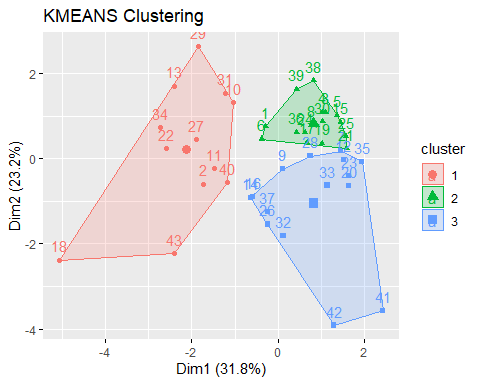


Figure 3. K-Means Clustering with K =2, K = 3, and K = 4

Based on the three plots above, I would recommend choosing K = 3 for clusters. If you look at the plot for K = 4, the traingle cluster (on the bottom right) only has 4 points. The number of data in the triangle cluster is quite small. This is happening as it’s trying to catch the outlier and we do not want to focus too much on capturing outliers. The plot for K = 2 seems better than K = 4, but it seems like the more clusters can be formed from the triangle cluster. Therefore, K = 3 would be the most reasonable choice as three clusters are formed reasonably.

1. One can choose K-Means clustering over hierarchical clustering for this particular data set. For hierarchical clustering, if you choose 3 clusters, one of the clusters contains majority of the data whereas one of the other two only contains one data (cereal number 18). Having only one point in a cluster means that the point is an outlier and hierarchical clustering doesn’t catch that outlier till the very end. However, if you choose K-Means clustering with K = 3, all three clusters are fully formed (as no cluster has only one point).

Question 2

(a) PPERSAUT, PBRAND, MOPLLAG are the important variables based on the summary for the unpruned tree.

## Classification tree:  
## tree(formula = Purchase ~ ., data = std.data, subset = -test)  
## Variables actually used in tree construction:  
## [1] "PPERSAUT" "PBRAND" "MOPLLAAG"  
## Number of terminal nodes: 4   
## Residual mean deviance: 0.4066 = 1959 / 4818   
## Misclassification error rate: 0.05993 = 289 / 4822

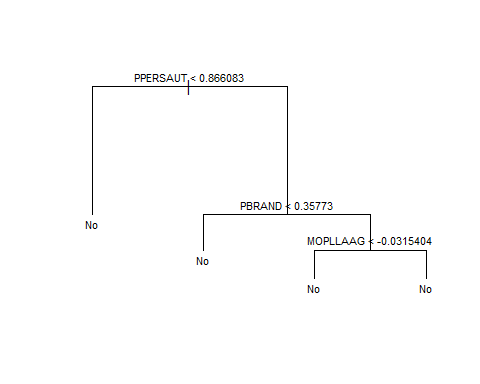


Figure 4. Unpruned decision tree of cereal data

There are 4 regions total. R1: PPERSAUT < 0.866083, R2: PPERSAUT > 0.866083 and PBRAND < 0.3577, R3: PPERSAUT > 0.866083 and PBRAND > 0.3577 and MOPLLAAG < -0.0315404, R4: PPERSAUT > 0.866083 and PBRAND > 0.3577 and MOPLLAAG > -0.0315404.

## test.Y  
## tree.pred No Yes  
## No 941 59  
## Yes 0 0

The above table shows the confusion matrix for the unpruned tree. The error rate is 5.9%.

* 1. Pruning is not helpful as the pruned tree looks exactly the same as the unpruned tree. The optimal size for the pruned tree is 4. Important predictors are PPERSAUT, PBRAND, MOPLLAAG. The table below shows the confusion matrix for the pruned tree. The error rate is also 5.9%.

## test.Y  
## prune.pred No Yes  
## No 941 59  
## Yes 0 0

* 1. Important variables for random forest are MOSTYPE and MOPLLAAG as they appear on the top on both MeanDecreaseAccuracy and MeanDecreaseGini plots. PBRAND might be an important variable as it appears to be the most important variable on the Gini plot, but it doesn’t appear on the MeanDecreaseAccuracy plot. The table below shows the confusion matrix for bagging. The error rate is 8.0%.

## test.Y  
## yhat.bag No Yes  
## No 912 51  
## Yes 29 8

(d) MOSTYPE, MOSHOOFD, and MOPLLAG appear to be the most important based on the VarImpPlots. The table below shows the confusion matrix for random forest. The error rate is 7.2%.

## test.Y  
## yhat.rf No Yes  
## No 924 55  
## Yes 17 4

1. The table below shows the confusion matrix for boosting. The error rate is 5.9%.

##   
## boost.pred 0 1  
## 0 941 59

Based on the summary, the important variables for boosting seems to be PPERSAUT,  
PPLEZIER, and PBRAND as they are the top three variables.

1. The table below shows the confusion matrix for KNN. The error rate is 5.9%.

## test.Y  
## knn.pred No Yes  
## No 940 58  
## Yes 1 1

For KNN, it’s not possible to find important predictors as it’s a distance-based method.

1. The table below shows the confusion matrix for logistic regression. The error rate is 6.6%.

## test.Y  
## lr.pred No Yes  
## No 934 59  
## Yes 7 0

Using the chi-squared test, we can find the important variables for logistic regression.

Anova(lr.fit, type = "II", test = "Wald")

## Analysis of Deviance Table (Type II tests)  
##   
## Response: Purchase  
## Df Chisq Pr(>Chisq)   
## MOSTYPE 1 2.6867 0.1011878   
## MAANTHUI 1 1.0435 0.3070204   
## MGEMOMV 1 0.4418 0.5062587   
## MGEMLEEF 1 5.5766 0.0182021 \*   
## MOSHOOFD 1 2.5052 0.1134725   
## MGODRK 1 0.1572 0.6917564   
## MGODPR 1 0.0126 0.9106862   
## MGODOV 1 0.3670 0.5446198   
## MGODGE 1 0.1229 0.7258594   
## MRELGE 1 0.9146 0.3388913   
## MRELSA 1 0.0178 0.8937304   
## MRELOV 1 0.1467 0.7016688   
## MFALLEEN 1 0.2594 0.6105535   
## MFGEKIND 1 0.6793 0.4098233   
## MFWEKIND 1 0.5880 0.4431842   
## MOPLHOOG 1 0.0046 0.9458976   
## MOPLMIDD 1 0.0587 0.8085862   
## MOPLLAAG 1 1.6664 0.1967384   
## MBERHOOG 1 1.6021 0.2056078   
## MBERZELF 1 0.5178 0.4717743   
## MBERBOER 1 0.2351 0.6277357   
## MBERMIDD 1 2.0622 0.1509873   
## MBERARBG 1 0.3453 0.5567968   
## MBERARBO 1 1.1896 0.2754102   
## MSKA 1 0.0730 0.7870001   
## MSKB1 1 0.3361 0.5620936   
## MSKB2 1 0.1322 0.7161287   
## MSKC 1 0.3380 0.5609845   
## MSKD 1 0.0079 0.9292677   
## MHHUUR 1 0.0003 0.9866609   
## MHKOOP 1 0.0003 0.9866834   
## MAUT1 1 1.1506 0.2834298   
## MAUT2 1 0.8779 0.3487753   
## MAUT0 1 0.2773 0.5984622   
## MZFONDS 1 0.0002 0.9883119   
## MZPART 1 0.0002 0.9882674   
## MINKM30 1 1.1078 0.2925566   
## MINK3045 1 1.6803 0.1948898   
## MINK4575 1 0.6209 0.4307174   
## MINK7512 1 0.4059 0.5240526   
## MINK123M 1 0.9132 0.3392621   
## MINKGEM 1 2.1993 0.1380703   
## MKOOPKLA 1 0.5940 0.4408617   
## PWAPART 1 1.3480 0.2456308   
## PWABEDR 1 0.6748 0.4113896   
## PWALAND 1 0.0124 0.9112507   
## PPERSAUT 1 15.2722 9.308e-05 \*\*\*  
## PBESAUT 1 0.0010 0.9753254   
## PMOTSCO 1 0.7027 0.4018689   
## PVRAAUT 1 0.0000 0.9994243   
## PAANHANG 1 0.7673 0.3810422   
## PTRACTOR 1 1.6960 0.1928094   
## PWERKT 1 0.0000 0.9989051   
## PBROM 1 0.3291 0.5661677   
## PLEVEN 1 4.2249 0.0398346 \*   
## PPERSONG 1 0.0582 0.8093958   
## PGEZONG 1 0.8839 0.3471425   
## PWAOREG 1 0.9998 0.3173697   
## PBRAND 1 11.2396 0.0008007 \*\*\*  
## PZEILPL 1 0.0000 0.9999792   
## PPLEZIER 1 0.8954 0.3440256   
## PFIETS 1 0.2837 0.5942686   
## PINBOED 1 1.1234 0.2891971   
## PBYSTAND 1 1.3564 0.2441667   
## AWAPART 1 0.9083 0.3405757   
## AWABEDR 1 0.4033 0.5254135   
## AWALAND 1 0.0629 0.8020102   
## APERSAUT 1 1.1600 0.2814688   
## ABESAUT 1 0.0009 0.9754512   
## AMOTSCO 1 0.5547 0.4563871   
## AVRAAUT 1 0.0000 0.9995513   
## AAANHANG 1 0.4327 0.5106650   
## ATRACTOR 1 1.3940 0.2377286   
## AWERKT 1 0.0000 0.9999561   
## ABROM 1 0.6532 0.4189762   
## ALEVEN 1 5.8607 0.0154824 \*   
## APERSONG 1 0.0429 0.8358979   
## AGEZONG 1 0.4648 0.4954053   
## AWAOREG 1 0.5859 0.4439992   
## ABRAND 1 3.3519 0.0671261 .   
## AZEILPL 1 0.0000 0.9988404   
## APLEZIER 1 5.4036 0.0200954 \*   
## AFIETS 1 0.1070 0.7435570   
## AINBOED 1 1.7537 0.1854130   
## ABYSTAND 1 1.9544 0.1621120   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Based on the chi-squared test to test the importance of variables, PPERSAUT and PBRAND are significant as their chi-squared values are less than 0.05.

1. Based on the error rates, the unpruned decision tree, the pruned decision tree, boosting, and KNN all had the lowest error rates as 5.9%. However, if you look at their confusion matrices, none of them is doing a good job capturing the true ‘Yes’s. However, although bagging has the highest error rate as 8.0%, it’s doing a good job of capturing true ‘Yes’s. Therefore, I believe having a highest error rate can be a good trade off. So, I would say bagging would be a good choice.

Question 3

1. Based on the fit, the optimal cost is 1. When it is tested against the test data, the error rate came out to be 20%. The table below is the confusion matrix for the support vector classifier.

## truth  
## predict 1 2 3  
## 1 4 0 0  
## 2 0 3 0  
## 3 1 2 5

1. Based on the fit, the optimal cost is 5. When it was tested against the test data, the error rate came out to be 53.3% The confusion matrix for the support vector machine with a polynomial kernel of degree two is shown below.

## truth  
## predict 1 2 3  
## 1 2 5 0  
## 2 2 0 0  
## 3 1 0 5

1. Based on the fit, the optimal cost is 0.5 and the optimal gamma is 0.5. When it was tested against the test data, the error rate came out to be 20%. The confusion matrix is shown below.

## truth  
## predict 1 2 3  
## 1 4 0 0  
## 2 0 3 0  
## 3 1 2 5

1. The error rates for support vector classifier, support vector machine with a polynomial degree = 2, and support vector machine with a radial came out to be 20%, 53.5% and 20%, respectively. From mini project 3, we observed that the error rates for LDA and QDA were 20% and 13%, respectively. Therefore, due to its lowest error rate, QDA is the best method amongst all of them. The decision boundary for QDA is shown in figure 5.

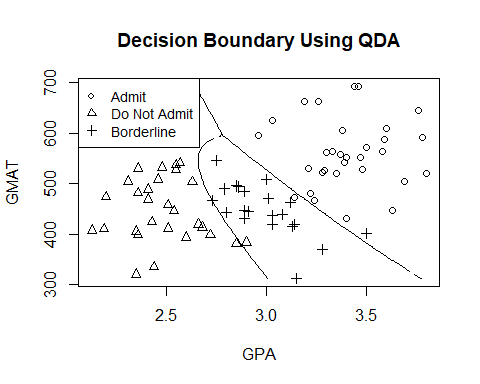


Figure 5. Decision boundary using QDA on the admission data set

Section 2: R Code

######### 1 a)

# exploratory analysis

Cereal = read.csv("C:/Users/jaemi/Desktop/STAT 4360 (Stat Learning)/Projects/Project 6/Cereal.csv", header = T)

str(Cereal) # look at the structure of the data

head(Cereal)

library(PerformanceAnalytics)

chart.Correlation(Cereal[,-(1:3)])

########## 1 b)

# standardizing data

std.cereal = scale(Cereal[,-(1:3)])

str(std.cereal)

head(std.cereal)

var(Cereal[,-(1:3)][,1]) # check the variance of the 1st variable before it's standardized -> 359

var(std.cereal[,1]) # notice it's standardized to 1

var(std.cereal[,2]) # standardized to 1

######### 1 d)

# apply hierarchical clustering using complete linkage and Euclidean distance

dist = dist(std.cereal, method = "euclidean")

hc.complete = hclust(dist, method = "complete")

plot(hc.complete)

######### 1 e)

# Use clusplot function to plot cluster when K = 2

library(factoextra)

km.cluster2.plot = eclust(x = std.cereal, FUNcluster = "kmeans", k = 2, graph = TRUE)

# Use clusplot function to plot cluster when K = 3

library(factoextra)

km.cluster2.plot = eclust(x = std.cereal, FUNcluster = "kmeans", k = 3, graph = TRUE)

# Use clusplot function to plot cluster when K = 4

library(factoextra)

km.cluster2.plot = eclust(x = std.cereal, FUNcluster = "kmeans", k = 4, graph = TRUE)

############## 2 (a)

library(ISLR)

dim(Caravan) # 5822 observations with 86 variables (one of them being response)

str(Caravan)

head(Caravan$Purchase)

# standardize the data

standardized.X = scale(Caravan[,-86])

str(standardized.X)

# add back the response (Purchase) to the standardized data

std.data = cbind(standardized.X, Caravan$Purchase)

std.data = data.frame(std.data)

colnames(std.data)[which(names(std.data) == "V86")] <- "Purchase"

str(std.data$Purchase)

# switch the purchase back to factor (Yes and no)

old.purchase = c('1','2')

new.purchase = factor(c('No', 'Yes'))

std.data$Purchase = new.purchase[match(std.data$Purchase, old.purchase)]

std.data$Purchase

# split into test and training data

test = 1:1000 # test ID

train.X = standardized.X[-test ,] # train X

str(train.X)

test.X = standardized.X[test ,] # test x

str(test.X)

train.Y = Caravan$Purchase[-test] # train y

test.Y = Caravan$Purchase[test] # test y

############ 2 a)

# decision tree

library(tree)

# grow a tree using training data

tree.caravan = tree(Purchase ~., std.data, subset = -test)

# see the default output

tree.caravan

# see the summary of the output

summary(tree.caravan)

# plot the tree

plot(tree.caravan)

text(tree.caravan, pretty = 0 , cex = 0.7)

# get predictions on the test set

tree.pred = predict(tree.caravan, data.frame(test.X), type = "class")

# compute the confusion matrix

tree.conf.mat = table(tree.pred, test.Y);tree.conf.mat

# misclassificaion rate

misclass.tree.unpruned = (59/1000) \* 100

####### 2 (b)

# prune the tree using CV

set.seed(1)

cv.caravan = cv.tree(tree.caravan, FUN = prune.misclass)

cv.caravan

names(cv.caravan)

# plot the estimated error rate

par(mfrow = c(1, 2))

plot(cv.caravan$size, cv.caravan$dev, type = "b")

plot(cv.caravan$k, cv.caravan$dev, type = "b")

# get the best size

cv.caravan$size[which.min(cv.caravan$dev)]

# get the pruned tree of the best size

prune.caravan = prune.misclass(tree.caravan, best = 4)

# plot the pruned tree

plot(prune.caravan)

text(prune.caravan, pretty = 0 , cex = 0.7)

library(tree)

# get predictions on the test set

prune.pred = predict(prune.caravan, data.frame(test.X), type = "class")

# compute the confusion matrix

conf.mat.pruned = table(prune.pred, test.Y);conf.mat.pruned

# misclassificaion rate

misclass.tree.pruned = (59/1000) \* 100 # 5.9%

######## 2 c)

# perform bagging using training data

library(randomForest)

set.seed(1)

bag.caravan <- randomForest(Purchase ~., data = std.data, subset = -test, mtry = 85, ntree = 1000, importance = TRUE)

bag.caravan

# estimate test error rate

yhat.bag = predict(bag.caravan, newdata = std.data[test, ])

# confusion matrix for bagging

table(yhat.bag, test.Y)

# error rate for bagging

err.bag = ((51+29)/1000)\*100 # 8 %

# look for the important variables for bagging

varImpPlot(bag.caravan)

####### 2 (d)

# perform randomForest using training data

set.seed(1)

rf.caravan <- randomForest(Purchase ~., data = std.data, subset = -test, mtry = sqrt(85), ntree = 1000, importance = TRUE)

rf.caravan

# estimate test error rate for random forest

yhat.rf = predict(rf.caravan, newdata = std.data[test, ])

# confusion matrix for random forest

table(yhat.rf, test.Y)

# error rate for random forest

err.rf = ((55+17)/1000)\*100 # 7.2 %

# look for the important variables for random forest

varImpPlot(rf.caravan)

###### 2 (e)

library(gbm)

Caravan[,1:(ncol(Caravan)-1)] =as.data.frame(scale(Caravan[,1:(ncol(Caravan)-1)]))

train.data = Caravan[1001:nrow(Caravan),]

test.data = Caravan[1:1000, ]

train.data[, "Purchase"] = ifelse(train.data[, "Purchase"] == "No", 0,1)

### fit a boosted regression tree using training data

set.seed(1)

boost.caravan = gbm(Purchase ~., data = train.data, distribution = "bernoulli", n.trees = 1000, interaction.depth = 1, shrinkage = 0.01, verbose = F)

summary(boost.caravan)

# estimate test error rate

test.data[,"Purchase"] = ifelse(test.data[, "Purchase"] == "No", 0, 1)

boost.prob = predict(boost.caravan, newdata = test.data, n.trees = 1000, type = "response")

summary(boost.prob)

boost.pred = ifelse(boost.prob > 0.5, 1, 0)

# confusion matrix for boosting

conf.mat.boost = table(boost.pred, test.data[,'Purchase']);conf.mat.boost

err.boost = (1-sum(diag(conf.mat.boost))/sum(conf.mat.boost))\*100;err.boost

######## 2 f)

# KNN

library(class)

library(crossval)

knn.cv.predfun <- function(Xtrain, Ytrain, Xtest, Ytest, k) {

fit <- knn(Xtrain, Xtest, Ytrain, k = k)

return(mean(fit != Ytest))

}

ks <- c(seq(1,50, by = 2))

knn.cv.err = c()

for (k in ks) {

set.seed(1)

knn.cv.err[k] <- crossval(knn.cv.predfun, k = k,

X = test.X, Y = test.Y, K = 10,

B = 1, verbose = F)$stat

}

k.opt <- which.min(knn.cv.err) # k.opt = 11

# Error rates for optimal k

set.seed(1)

knn.pred = knn(test.X, test.X, test.Y, k = k.opt)

# confusion matrix for knn

table(knn.pred, test.Y)

# knn error rate

err.knn = (59/1000)\*100 # 5.9

# fit a logistic regression model with training data

lr.fit = glm(Purchase~., data = std.data, family = binomial, subset = -test)

lr.prob = predict(lr.fit, std.data[test, ], type = "response")

lr.pred = ifelse(lr.prob > 0.5, "Yes", "No")

# confusion matrix for KNN

table(lr.pred, test.Y)

# KNN error rate

err.knn = (66/1000)\*100

library(car)

Anova(lr.fit, type = "II", test = "Wald")

############### 3 (a)

# Read data and define X and Y

Admission = read.csv("C:/Users/jaemi/Desktop/STAT 4360 (Stat Learning)/Projects/Project 6/admission.csv")[,-(4:7)]

X=Admission[,1:2]

Y=as.factor(Admission[,3])

levels(Y)=c("admit","not admit","borderline")

Admission$Group = as.factor(Admission$Group)

test = vector("logical", nrow(Admission))

test[c(27:31, 55:59, 81:85)] = TRUE

# split into test and training set

Admission.test <- Admission[test, ]

Admission.train = Admission[!test, ]

# support vector classifier

library(e1071)

library(caret)

set.seed(1)

svc.train = tune(svm, Group~., data =Admission.train, kernel = "linear", ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10)))

summary(svc.train)

# get results for the best model

best.mod = svc.train$best.model

summary(best.mod)

# predict the test responses using the best model

ypred = predict(best.mod, Admission.test)

# confusion matrix for SVC

conf.mat.svc = table(predict = ypred, truth = Admission.test$Group);conf.mat.svc

err.svc = (1-sum(diag(conf.mat.svc))/sum(conf.mat.svc))\*100;err.svc

############## 3 (b)

# SVM with polynomial kernel of degree two

svm.poly.train = tune(svm, Group~., data =Admission.train, kernel = "polynomial", degree = 2, ranges = list(cost = c(0.001, 0.01, 0.1, 0.2, 0.3, 0.5, 1, 5, 10)))

summary(svm.poly.train)

# get the results for the best model

best.mod.poly = svm.poly.train$best.model

summary(best.mod.poly)

# evaluate test set performance

ypred.poly = predict(best.mod.poly, Admission.test)

# confusion matrix for SVC

conf.mat.svm = table(predict = ypred.poly, truth = Admission.test$Group);conf.mat.svm

# error rate for polynomial degree of 2

err.svm = (1-sum(diag(conf.mat.svm))/sum(conf.mat.svm))\*100;err.svm

########## 3 (c)

# SVM with radial

svm.rad.train = tune(svm, Group~., data =Admission.train, kernel = "radial", ranges = list(cost = c(0.001, 0.01, 0.1, 0.2, 0.3, 0.5, 1, 5, 10), gamma = c(0.001, 0.01, 0.5, 1, 2, 3, 4)))

summary(svm.rad.train)

# get the results for the best model

best.mod.rad = svm.rad.train$best.model

summary(best.mod.rad)

# evaluate test set performance for radial

ypred.rad = predict(best.mod.rad, Admission.test)

# confusion matrix for SVC

conf.mat.rad = table(predict = ypred.rad, truth = Admission.test$Group);conf.mat.rad

# error rate for polynomial degree of 2

err.rad = (1-sum(diag(conf.mat.rad))/sum(conf.mat.rad))\*100;err.rad

plot.out= svm(Group~., data = Admission, kernel = "radial", cost = 0.5)

plot(plot.out, Admission)

admission = read.csv("C:/Users/jaemi/Desktop/STAT 4360 (Stat Learning)/Projects/Project 3/admission.csv")

adm = data.frame(admission)

# training data (row 1-80 from col 1,2,3)

train = adm[1:80, ]

train.x = adm[1:80, 1:2]

train.y = adm[1:80, 3]

train.id <- logical(85) # creates a logical vector of the specified length.

# each element of the vector is equal to FALSE

train.id[1:80] <- TRUE # set 1:80 (train data) to be true

# test data (row 81-85 from col 1, 2, 3)

test.x = adm[81:85, 1:2]

test.y = adm[81:85, 3]

# fit qda

qda.train = qda(Group ~ GPA + GMAT, data = adm, subset = train.id)

# get predictions for test data

qda.pred = predict(qda.train, adm[!train.id,])

# confusion matrix for test data

table(qda.pred$class, test.y) # no missclassification

# error rate for test data

mean(qda.pred$class != test.y) # 0 %

# get predictions for train data

qda.pred2 = predict(qda.train, adm[train.id,])

## Decision boundary

# set up a dense grid and compute posterior probability on the grid

n.grid = 100

x1.grid = seq(f = min(train.x[, 1]), t = max(train.x[, 1]), l = n.grid)

x2.grid = seq(f = min(train.x[, 2]), t = max(train.x[, 2]), l = n.grid)

grid = expand.grid(x1.grid, x2.grid)

colnames(grid) = colnames(train.x)

qda.pred.grid = predict(qda.train, grid)

# p\*(x) for class boundaries

p1star.qda = qda.pred.grid$posterior[,1] - pmax(qda.pred.grid$posterior[,2], qda.pred.grid$posterior[,3])

p2star.qda = qda.pred.grid$posterior[,2] - pmax(qda.pred.grid$posterior[,1], qda.pred.grid$posterior[,3])

prob1.qda = matrix(p1star.qda, nrow = n.grid, ncol = n.grid, byrow = F)

prob2.qda = matrix(p2star.qda, nrow = n.grid, ncol = n.grid, byrow = F)

# QDA Decision boundary

plot(train.x, pch = train.y, main = 'Decision Boundary Using QDA')

contour(x1.grid, x2.grid, prob1.qda, levels = 0, labels = "", xlab = "", ylab = "", main = "", add = T)

contour(x1.grid, x2.grid, prob2.qda, levels = 0, labels = "", xlab = "", ylab = "", main = "", add = T)

legend("topleft", legend = c("Admit", "Do Not Admit", "Borderline"), pch = c(1, 2, 3), cex = 0.9)