Task 1

We first defined functions to calculate the error rate of the fitted models.

> #Calculate error rate from observed and predicted

> errorrate <- function(observed, predicted) {

+ tab<-table(observed, predicted)

+ errorrate<-1-sum(diag(tab))/sum(tab)

+ return(errorrate)

+ }

>

> #Calculate error rate from model ouput, training and test data

> model <- function(model.out, train.data, train.target, test.data, test.target) {

+ pred.train <- predict(model.out, train.data)

+ pred.test <- predict(model.out, test.data)

+ train <- errorrate(train.target, pred.train$class)

+ test <- errorrate(test.target, pred.test$class)

+ return(list(train = train, test = test))

+ }

>

> #Tune k for KNN

> tuneknn <- function(train.data, train.target, test.data, test.target, kmax) {

+ knn<-matrix(rep(0,kmax\*2),nrow=kmax)

+ for (j in 1:kmax){

+ predknn.train<- knn(train.data, train.data, train.target, k=j)

+ knn[j,1]<-errorrate(train.target,predknn.train)

+

+ predknn.test<- knn(train.data, test.data, train.target, k=j)

+ knn[j,2]<-errorrate(test.target,predknn.test)

+ }

+ return(knn)

+ }

>

> #Plot KNN as a function of k

> plotknn <- function(knn, kmax) {

+ plot(-10,-10,xlim=c(1,kmax),ylim=c(0,0.15),col="red",type="b",xlab="K",ylab="error")

+ lines(c(1:kmax),knn[,1],col="red")

+ lines(c(1:kmax),knn[,2],col="blue")

+ legend("topright",c("training error", "test error"),col=c("red","blue"),lty=c(1,1))

+ }

>

> #Find the best k for knn, not taking k=1 into account

> knnbest <- function(knn) {

+ best <- which(knn1[,2] == sort(unique(knn1[,2]))[2])

+ return(best = best)

+ }

>

> #Calculate error rate for random forest

> rferror <- function(train.data, train.target, test.data, test.target, mtry, ntree) {

+ rfdata<-data.frame(train.target=factor(train.target),train.data)

+ bag.mod=randomForest(train.target~.,data=rfdata,mtry=mtry,ntree=ntree,importance=TRUE)

+ predrf.train<-predict(bag.mod,newdata=rfdata)

+ train<-errorrate(rfdata$train.target,predrf.train)

+ predrf.test<-predict(bag.mod,newdata=test.data)

+ test<-errorrate(test.target,predrf.test)

+ return(list(train = train, test = test))

+ }

>

> #Calculate error rate for HDDA

> hddaerror <- function(train.data, train.target, test.data, test.target, model, d\_select, threshold) {

+ hdda.out <- hdda(train.data, train.target, model=model, d\_select = d\_select, threshold = threshold)

+ predhdda.train <- predict(hdda.out, train.data, train.target)

+ train<-errorrate(train.target,predhdda.train$class)

+ predhdda.test <- predict(hdda.out, test.data, test.target)

+ test<-errorrate(test.target,predhdda.test$class)

+ return(list(train = train, test = test))

+ }

a)

The principal components were extracted from training data, and were used to transform the train and test data.

> pctraindata <- prcomp(traindata)

> pctestdata <- prcomp(testdata)

>

> totvar<- sum(apply(traindata,2,var))

> eigenvalues<-pctraindata$sdev^2

> propvar<-eigenvalues/totvar

> cumpropvar<-cumsum(propvar)

> #scenario 1: components account for 80% of the variance in the training data

> scen1 <- 43

> pcatrain1<-traindata%\*%pctraindata$rotation[,1:scen1]

> pcatest1<-testdata%\*%pctraindata$rotation[,1:scen1]

> #scenario 2: components account for 90% of the variance in the training data

> scen2 <- 86

> pcatrain2<-traindata%\*%pctraindata$rotation[,1:scen2]

> pcatest2<-testdata%\*%pctraindata$rotation[,1:scen2]

b)

> #LDA PCA scenario 1

> lda1.out<-lda(pcatrain1,train.target)

> lda1 <- model(lda1.out, pcatrain1, train.target, pcatest1, test.target)

>

> #LDA PCA scenario 2

> lda2.out<-lda(pcatrain2,train.target)

> lda2 <- model(lda2.out, pcatrain2, train.target, pcatest2, test.target)

>

> #QDA PCA scenario 1

> qda1.out<-qda(pcatrain1,train.target)

> qda1 <- model(qda1.out, pcatrain1, train.target, pcatest1, test.target)

>

> #QDA PCA scenario 2

> qda2.out<-qda(pcatrain2,train.target)

> qda2 <- model(qda2.out, pcatrain2, train.target, pcatest2, test.target)

>

> #KNN PCA scenario 1 with tuning

> knn1 <- tuneknn(pcatrain1, train.target, pcatest1, test.target, 20)

> plotknn(knn1, 20)

>

> #KNN PCA scenario 2 with tuning

> knn2 <- tuneknn(pcatrain2, train.target, pcatest2, test.target, 20)

> plotknn(knn2, 20)

>

> #Random Forest PCA scenario 1 (p = 43)

> rf1<-rferror(pcatrain1, train.target, pcatest1, test.target, 5, 500)

>

> #Random Forest PCA scenario 2 (p = 86)

> rf2<-rferror(pcatrain2, train.target, pcatest2, test.target, 5, 500)

>

> # HDDA AKJBKQKD (raw data)

> hdda1<-hddaerror(traindata, train.target, testdata, test.target, model="AKJBKQKD", d\_select = "Cattell", threshold = 0.05)

>

> # HDDA AKJBQKD (raw data)

> hdda2<-hddaerror(traindata, train.target, testdata, test.target, model="AKJBQKD", d\_select = "Cattell", threshold = 0.05)

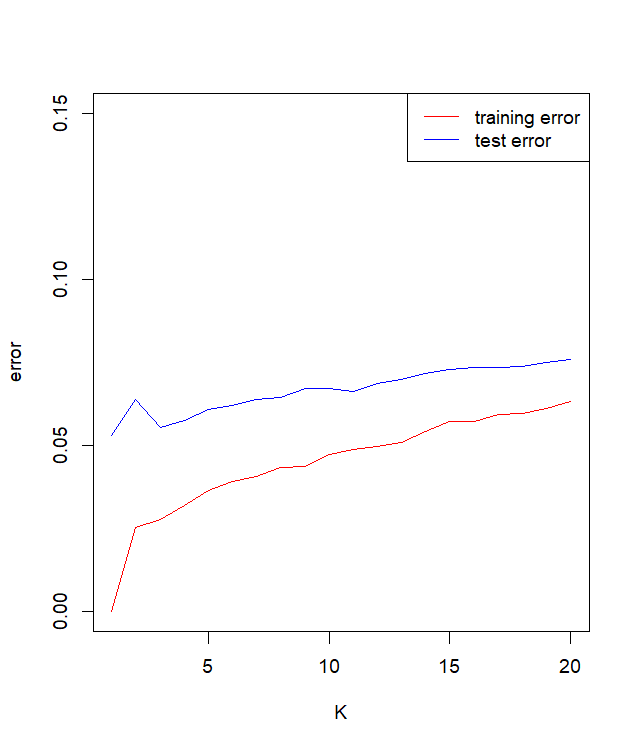
The two KNN models for scenarios 1 and 2 were tuned for the best number of k. Training and test error were plotted as a function of k, as shown in the figures below. We do not take k=1 as the best model since its perfect fit is expected. Hence, the second best model were chosen, here both k are equal to 3.

Figure .2. KNN2 performance

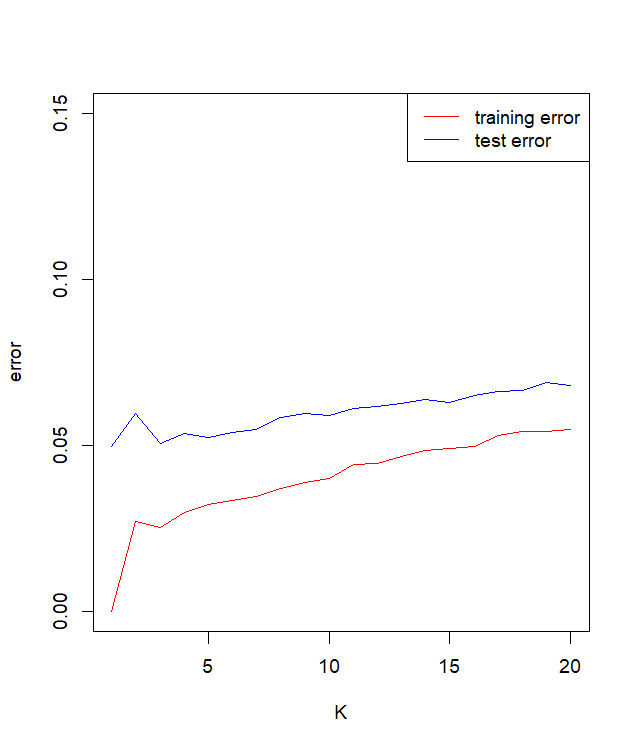
We tuned the number of variables to be considered, *mtry*, in the random forest models, resulting in mtry=5 for both. We selected 500 as the number of trees, since this is reasonably large with acceptable running time.

Figure 1.1. KNN1 performance

c)

> results.out<-results()

train test

LDA1 0.1300 0.1350

LDA2 0.1196 0.1274

QDA1 0.0230 0.0462

QDA2 0.0130 0.0564

KNN1 0.0254 0.0506

KNN2 0.0278 0.0556

RF1 0.0000 0.0690

RF2 0.0000 0.0702

HDDA1 0.0262 0.0526

HDDA2 0.0256 0.0508

[1] "The best KNN1 has k = "

[1] 3

[1] "The best KNN2 has k = "

[1] 3

[1] "The best model is: "

[1] "QDA1"

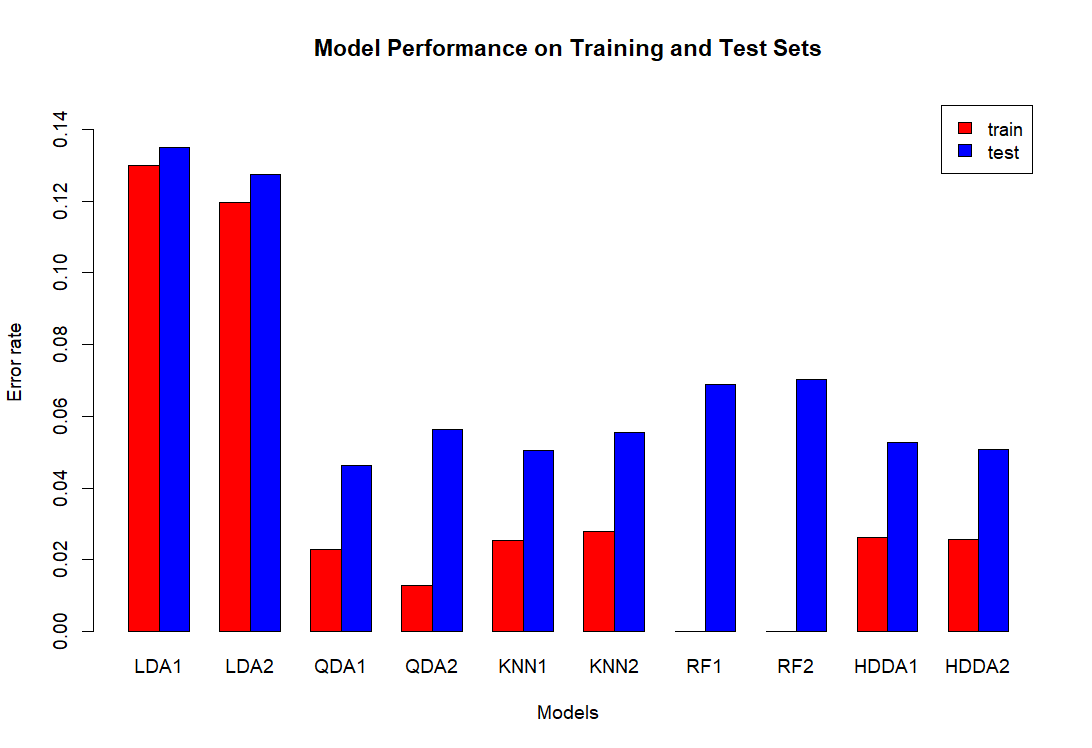


Figure 1.3. Model Performance

Overall, QDA, KNN, and HDDA are the best-performing models based on the error rate, with Random Forest performing perfectly on the training set but exhibiting slightly higher error rates on the test set compared to other models. LDA performs the worst, with a high error rate in both train and test set.