Applied Data Mining: Homework 4

Anshul Doshi

Anshul Doshi (ad3222)
STAT UN3106 Section 001
Assignment: 4

Problem 1: LDA vs. KNN

i.) Initialize dataset

```
#setwd("~/Desktop/Spring2017/Applied Data Mining/Homeworks/Homework4")
kNNData <- read.csv("kNNData.csv")[,c("X1","X2","Class")]
set.seed(2)
test.index <- sample(1:nrow(kNNData),100,replace=F)
kNNData.test  <- kNNData[test.index, ]
kNNData.train <- kNNData[-test.index, ]
```

ii.) Classify a test point using LDA

```
my.lda <- function(pi.vec,mu,Sigma,x){
  # Inputs:
  # pi.vec  :   vector of class probabilities
  # mu      :   matrix of means per class
  # Sigma   :   covariance matrix
  # x       :   vector of inputs, i.e., query point x = (test.balance,test.income)
  # Outputs:
  # out.vec :   vector of predicted classes
  x.dims <- dim(x)
  n <- x.dims[1]
  Sigma.inv <- solve(Sigma)

  out.prod <- vector(length=n)

  discrim.1 <- apply(x,1,function(y) y %*% Sigma.inv %*% mu[,1] - 0.5*t(mu[,1]) %*% Sigma.inv %*% mu[,1]
  discrim.2 <- apply(x,1,function(y) y %*% Sigma.inv %*% mu[,2] - 0.5*t(mu[,2]) %*% Sigma.inv %*% mu[,2]
  discrim.3 <- apply(x,1,function(y) y %*% Sigma.inv %*% mu[,3] - 0.5*t(mu[,3]) %*% Sigma.inv %*% mu[,3]

  class_names <- c("Group1", "Group2", "Group3")
  for (i in 1:n)
  {
    out.prod[i] <- class_names[which.max(c(discrim.1[i],discrim.2[i],discrim.3[i]))]
  }

  return(out.prod)
}

head(kNNData.train)
```

```
##           X1        X2  Class
## 1  -5.506668  4.493600 Group1
## 5  -2.634081  3.114774 Group1
```

```
## 7  -5.825373 -3.082661 Group1
## 8  -1.255092  4.013510 Group1
## 9  -3.909155  6.979583 Group1
## 11 -1.644468  6.597973 Group1
```

```
unique(kNNData.train$Class)
```

```
## [1] Group1 Group2 Group3
## Levels: Group1 Group2 Group3
```

```
#Estimating Parameters
#pi vector
pi.vec <- c(sum((kNNData.train$Class=="Group1")),sum((kNNData.train$Class=="Group2")),
            sum((kNNData.train$Class=="Group3")))/length(kNNData.train$Class)
#pi.vec

#mean vector
mu <- rbind(tapply(kNNData.train$X1,kNNData.train$Class,mean),
            tapply(kNNData.train$X2,kNNData.train$Class,mean))
rownames(mu)<-c("x1","x2")
mu
```

```
##       Group1   Group2   Group3
## x1 -3.447668 1.882494 3.019406
## x2  3.977015 2.144897 7.616310
```

```
# Variance-covariance matrices for k=1,2,3
Sig.1 <- var(kNNData.train[kNNData.train$Class=="Group1",c("X1","X2")])

Sig.2 <- var(kNNData.train[kNNData.train$Class=="Group2",c("X1","X2")])

Sig.3 <- var(kNNData.train[kNNData.train$Class=="Group3",c("X1","X2")])

# Pooled variance-covariance matrices for k=1,2
Sigma <- ((sum(kNNData.train$Class=="Group1")-1)*Sig.1+
          (sum(kNNData.train$Class=="Group2")-1)*Sig.2+
          (sum(kNNData.train$Class=="Group3")-1)*Sig.3)/
            (length(kNNData.train$Class)-3)
Sigma
```

```
##           X1         X2
## X1 4.8566571  0.6830786
## X2 0.6830786 10.2639655
```

```
#Test Points
#x1=0, x2=10.
x.test <- matrix(c(0,10),nrow=1)
colnames(x.test) <- c("x1","x2")
x <- x.test
cat("At (0, 5) lda predicts: ", my.lda(pi.vec=pi.vec,mu=mu,Sigma=Sigma,x=x.test))
```

```
## At (0, 5) lda predicts:  Group3
```

```
#x1=0, x2=5
x.test[2] = 5
x <- x.test
cat("At (0, 5) lda predicts: ", my.lda(pi.vec=pi.vec,mu=mu,Sigma=Sigma,x=x.test))
```

```
## At (0, 5) lda predicts:  Group2
```

iii.) Compute the prediction error and compare with kNN

```
## kNN function
KNN.decision <- function(x1.new, x2.new, K = 14,
                         x1 = kNNData$X1,
                         x2 = kNNData$X2,
                         Class = kNNData$Class) {
  n <- length(x1)
  stopifnot(length(x2) == n, length(x1.new) == 1,
            length(x2.new) == 1, K <= n)

  dists <- sqrt((x1-x1.new)^2 + (x2-x2.new)^2)

  neighbors  <- order(dists)[1:K]
  neighb.dir <- Class[neighbors]
  choice     <- names(which.max(table(neighb.dir)))
  return(choice)
}
KNN.decision(0, 10)
```

```
## [1] "Group3"
```

```
KNN.decision(0, 5)
```

```
## [1] "Group3"
```

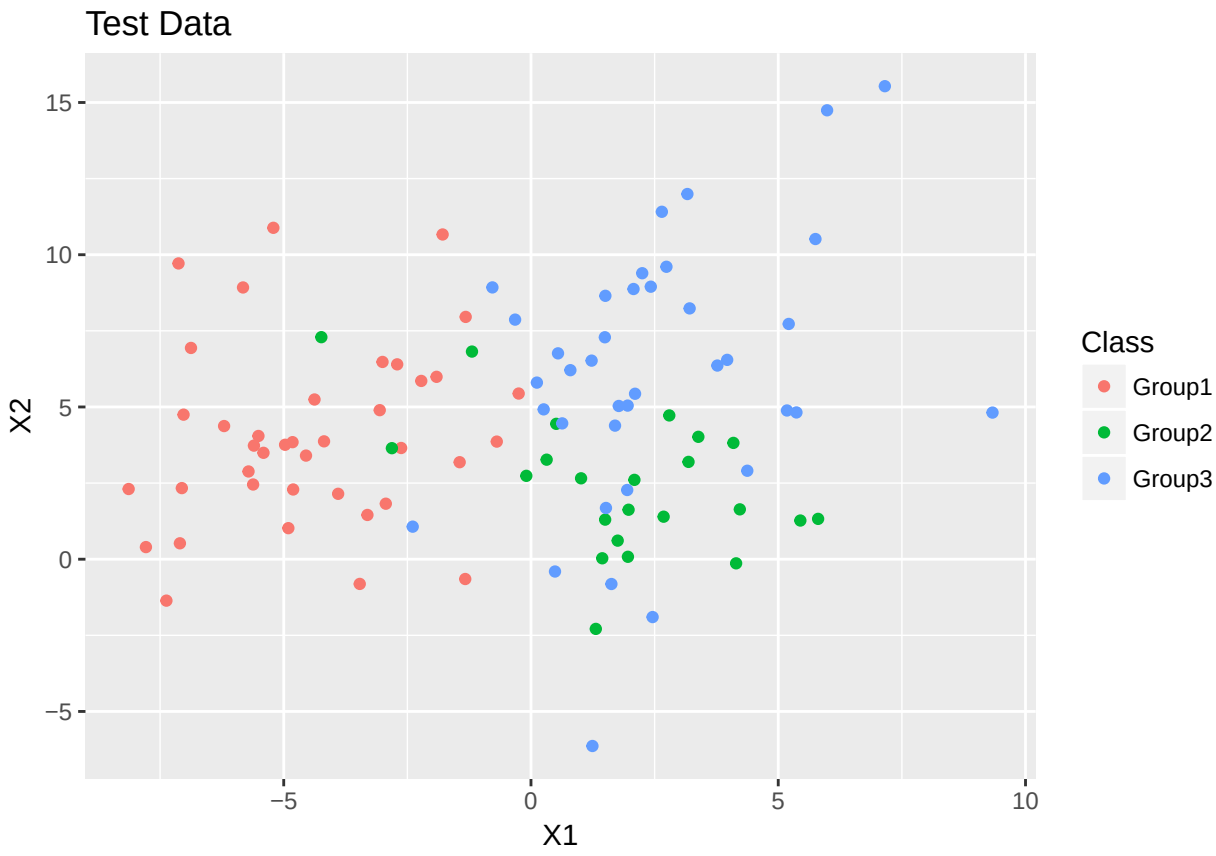
```
#Prediction Error when k = 14
n.test <- nrow(kNNData.test)
predictions <- rep(NA, n.test)

for (i in 1:n.test){
  predictions[i] <- KNN.decision(kNNData.test$X1[i],kNNData.test$X2[i],
                                 x1 = kNNData.train$X1, x2 = kNNData.train$X2,
                                 Class = kNNData.train$Class)
}
test.error <- sum(predictions != kNNData.test$Class)/n.test
cat("Prediction error for k=14 in KNN:",test.error)
```

```
## Prediction error for k=14 in KNN: 0.17
```

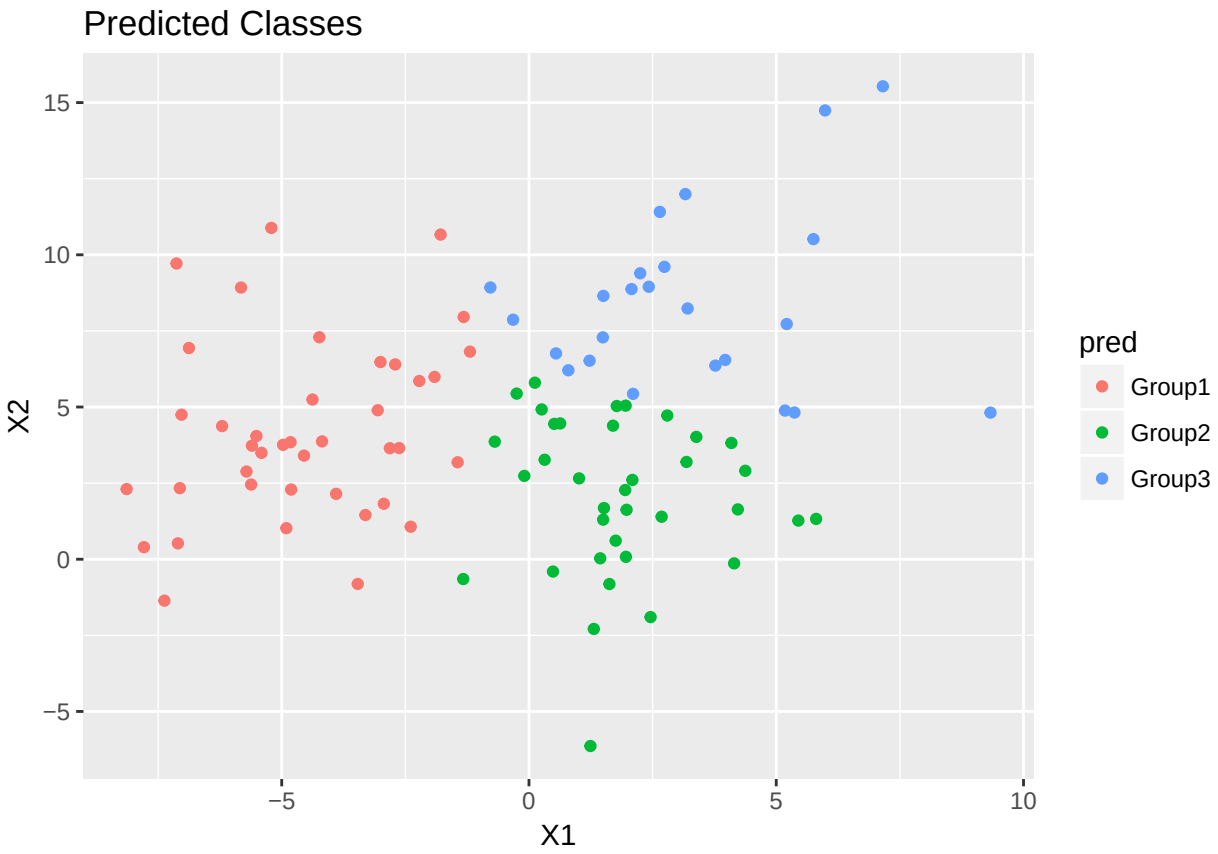
```
#Predicition Error for LDA
x.test = kNNData.test[, c("X1","X2")]
y_hat <- my.lda(pi.vec=pi.vec,mu=mu,Sigma=Sigma,x=x.test)
#y_hat

pred_error <- mean(kNNData.test$Class!=y_hat)
cat("Prediction error for LDA:",pred_error)
```

```
## Prediction error for LDA: 0.2
```

```
#Plot data
library(ggplot2)
ggplot(data=kNNData.test)+
  geom_point(mapping=aes(x=X1,y=X2,color=Class))+
  labs(title="Test Data")
```




```
# Plot the data
kNNData.test$pred <- y_hat
ggplot(data=kNNData.test)+
  geom_point(mapping=aes(x=X1,y=X2,color=pred))+
  labs(title="Predicted Classes")
```



Thus we get that using k=14 with the kNN model gives a prediction error slightly lower than lda.

Problem 2: Yale Faces - 1NN and PCA

a.) Load the data

```
pic_list <- c(paste("0", 1:9, sep=""), 10:13, 15:39)
views_2a = c('P00A+000E+00', 'P00A+005E+10', 'P00A+005E-10', 'P00A+010E+00' )

# load the data and save it as a matrix with the name face_matrix_2a

face_matrix_2a <- NULL
id_2a <- NULL
view_2a <- NULL

for ( i in 1:length(pic_list) ){
  for ( j in 1:length(views_2a) ){
    this_filename <- paste("CroppedYale/yaleB", pic_list[i], "/yaleB", pic_list[i], "_", views_2a[j], "
    this_face = read.pnm(file = this_filename)
    this_face_matrix = getChannels(this_face)               # matrix of the pic; dim = 192 X 168
    face_row_2a <- c(this_face_matrix)                      # convert the matrix into row vector by the
    face_matrix_2a <- rbind(face_matrix_2a, face_row_2a)
    id_2a <- c(id_2a, i)                                            # subject id number: 1:38
    view_2a <- c(view_2a, j)                                        # view number: 1:4
    # When knitting the R Markdown file, make sure to comment the code out below!
    #cat("id=", pic_list[i], "view=", views_2a[j], "\n")        # keep tract of loading the pics
  }
```

```
}



# Construct test data and training data below

fm_2a_size = dim(face_matrix_2a)            # Get the size of the matrix for use later
# Use 4/5 of the data for training, 1/5 for testing
ntrain_2a = floor(fm_2a_size[1]*4/5)        # Number of training obs
ntest_2a = fm_2a_size[1]-ntrain_2a          # Number of testing obs
set.seed(1)                                 # Set pseudo-random numbers so everyone gets the same outpu
ind_train_2a = sample(1:fm_2a_size[1],ntrain_2a)    # Training indices
ind_test_2a = c(1:fm_2a_size[1])[-ind_train_2a]     # Testing indices

# Check the results

dat.train_2a <- face_matrix_2a[ind_train_2a,]                                # training data
dat.test_2a <- face_matrix_2a[ind_test_2a,]                                  # test data
cbind(id_2a=id_2a[ind_train_2a][1:5], view_2a=view_2a[ind_train_2a][1:5])   # the first 5 files in trai
```

```
##      id_2a view_2a
## [1,]    11       1
## [2,]    15       1
## [3,]    22       2
## [4,]    34       4
## [5,]     8       2
```

```
cbind(id_2a=id_2a[ind_test_2a][1:5], view_2a=view_2a[ind_test_2a][1:5])     # the first 5 files in test
```

```
##      id_2a view_2a
## [1,]     2       1
## [2,]     3       4
## [3,]     5       2
## [4,]     5       4
## [5,]     6       1
```

b.) Run PCA on first 25 scores of training data and project test data onto these 25 scores.

```
mean.train_2a <- colMeans(dat.train_2a)                                      # mean face of training dat
dat.train_demean_2a <- scale(dat.train_2a, center=T, scale=F)                # subtract training mean fa
pca_face_2a <- prcomp(dat.train_demean_2a)                                   # run PCA on the de-meaned
loading.train_2a <- pca_face_2a$rotation[,1:25]                              # loadings of the first 25
X.train_2a <- pca_face_2a$x[,1:25]                                           # use the first 25 scores a
Y.train_2a <- id_2a[ind_train_2a]                                            # Y.train_2a = the differen

dat.test_demean_2a <- t(t(dat.test_2a) - mean.train_2a)                      # subtract training mean fa
X.test_2a <- dat.test_demean_2a %*% loading.train_2a                         # project de-meaned test da
Y.test_2a <- id_2a[ind_test_2a]

# Apply the kNN classification method to the above data set. Use k=1, i.e., 1NN classification method.

#----- START YOUR CODE BLOCK HERE -----#
KNN.decision <- function(testData, K = 1,
                         trainingData=X.train_2a,
                         Class=Y.train_2a) {
```

```
  n <- nrow(trainingData)

  dists <- rep(0, n)
  for (i in 1:ncol(trainingData))
  {
    dists = dists + (trainingData[,i]-testData[i])^2
  }
  dists <- sqrt(dists)

  #dists <- sqrt((x1-x1.new)^2 + (x2-x2.new)^2)

  neighbors  <- order(dists)[1]
  neighb.dir <- Class[neighbors]
  return(neighb.dir)
}

n.test <- length(Y.test_2a)
predictions <- rep(NA, n.test)

testing <- KNN.decision(X.test_2a[1,])
testing
```

```
## [1] 2
```

```
for (i in 1:n.test){
  predictions[i] <- KNN.decision(X.test_2a[i,])
}
test.error <- sum(predictions != Y.test_2a)/n.test
test.error
```

```
## [1] 0
```

```
Y.test_2a
```

```
##  [1]  2  3  5  5  6  6  8  9  9 10 10 14 14 17 18 19 19 20 24 24 25 27 29
## [24] 29 30 30 30 31 32 36 38
```

```
predictions
```

```
##  [1]  2  3  5  5  6  6  8  9  9 10 10 14 14 17 18 19 19 20 24 24 25 27 29
## [24] 29 30 30 30 31 32 36 38
```

```
#----- END YOUR CODE BLOCK HERE -----#
```

With the prediction error being 0, all subjects were identified correctly

c.) Rerun above with different views

```
# Use different lighting conditions
pic_list <- c(paste("0", 1:9, sep=""), 10:13, 15:39)
views_2c = c('P00A-035E+15', 'P00A-050E+00', 'P00A+035E+15', 'P00A+050E+00')
pic_list
```

```
##  [1] "01" "02" "03" "04" "05" "06" "07" "08" "09" "10" "11" "12" "13" "15"
## [15] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28" "29"
## [29] "30" "31" "32" "33" "34" "35" "36" "37" "38" "39"
```

```
views_2c
```

```
## [1] "P00A-035E+15" "P00A-050E+00" "P00A+035E+15" "P00A+050E+00"
```

```
# Several steps follow below

# load the data and save it as a matrix with the name face_matrix_2a

face_matrix_2c <- NULL
id_2c <- NULL
view_2c <- NULL

for ( i in 1:length(pic_list) ){
  for ( j in 1:length(views_2c) ){
    this_filename <- paste("CroppedYale/yaleB", pic_list[i], "/yaleB", pic_list[i], "_", views_2c[j], "
    this_face = read.pnm(file = this_filename)
    this_face_matrix = getChannels(this_face)               # matrix of the pic; dim = 192 X 168
    face_row_2c <- c(this_face_matrix)                      # convert the matrix into row vector by the
    face_matrix_2c <- rbind(face_matrix_2c, face_row_2c)
    id_2c <- c(id_2c, i)                                          # subject id number: 1:38
    view_2c <- c(view_2c, j)                                      # view number: 1:4
    # When knitting the R Markdown file, make sure to comment the code out below!
    #cat("id=", pic_list[i], "view=", views_2c[j], "\n")        # keep tract of loading the pics
  }
}

# Construct test data and training data below

fm_2c_size = dim(face_matrix_2c)            # Get the size of the matrix for use later
# Use 4/5 of the data for training, 1/5 for testing
ntrain_2c = floor(fm_2c_size[1]*4/5)        # Number of training obs
ntest_2c = fm_2c_size[1]-ntrain_2c          # Number of testing obs
set.seed(2)                                 # Set pseudo-random numbers so everyone gets the same outpu
ind_train_2c = sample(1:fm_2c_size[1],ntrain_2c)    # Training indices
ind_test_2c = c(1:fm_2c_size[1])[-ind_train_2c]     # Testing indices

# Check the results

dat.train_2c <- face_matrix_2c[ind_train_2c,]                                 # training data
dat.test_2c <- face_matrix_2c[ind_test_2c,]                                   # test data
cbind(id_2c=id_2c[ind_train_2c][1:5], view_2c=view_2c[ind_train_2c][1:5])   # the first 5 files in trai
```

```
##      id_2c view_2c
## [1,]     8       1
## [2,]    27       3
## [3,]    22       2
## [4,]     7       2
## [5,]    35       4
```

```
cbind(id_2c=id_2c[ind_test_2c][1:5], view_2c=view_2c[ind_test_2c][1:5])     # the first 5 files in test
```

```
##      id_2c view_2c
## [1,]     1       3
## [2,]     2       1
## [3,]     2       4
## [4,]     3       1
## [5,]     4       4
```

```
view_2c[ind_test_2c]
```

```
##  [1] 3 1 4 1 4 3 4 3 4 4 4 2 4 4 1 3 2 4 3 4 1 2 4 4 2 3 1 2 3 4 1 1
```

```
#----- START YOUR CODE BLOCK HERE -----#
mean.train_2c <- colMeans(dat.train_2c)                                    # mean face of training dat
dat.train_demean_2c <- scale(dat.train_2c, center=T, scale=F)              # subtract training mean fa
pca_face_2c <- prcomp(dat.train_demean_2c)                                 # run PCA on the de-meaned
loading.train_2c <- pca_face_2c$rotation[,1:25]                            # loadings of the first 25
X.train_2c <- pca_face_2c$x[,1:25]                                         # use the first 25 scores a
Y.train_2c <- id_2c[ind_train_2c]                                          # Y.train_2a = the differen

dat.test_demean_2c <- t(t(dat.test_2c) - mean.train_2c)                    # subtract training mean fa
X.test_2c <- dat.test_demean_2c %*% loading.train_2c                       # project de-meaned test da
Y.test_2c <- id_2c[ind_test_2c]

# Apply the kNN classification method to the above data set. Use k=1, i.e., 1NN classification method.

#----- START YOUR CODE BLOCK HERE -----#
KNN.decision <- function(testData, K = 1,
                         trainingData = X.train_2c,
                         Class = Y.train_2c) {
  n <- nrow(trainingData)

  dists <- rep(0, n)
  for (i in 1:ncol(trainingData))
  {
    dists = dists + (trainingData[,i]-testData[i])^2
  }
  dists <- sqrt(dists)

  #dists <- sqrt((x1-x1.new)^2 + (x2-x2.new)^2)

  neighbors  <- order(dists)[1]
  neighb.dir <- Class[neighbors]
  #choice <- names(which.max(table(neighb.dir)))
  return(neighb.dir)
}

n.test <- length(Y.test_2c)
predictions <- rep(NA, n.test)

testing <- KNN.decision(X.test_2c[1,])
testing
```

```
## [1] 3
```

```
for (i in 1:n.test){
  predictions[i] <- KNN.decision(testData = X.test_2c[i,])
}
test.error <- sum(predictions != Y.test_2c)/n.test
test.error
```

```
## [1] 0.8709677
```

```
Y.test_2c
```

```
##  [1]  1  2  2  3  4  9  9 11 11 13 16 16 18 22 22 23 23 25 25 27 27 27 28
## [24] 29 31 32 33 33 34 35 37
```

```
predictions
```

```
##  [1]  3  2 33 10  7  8  8 27 19 10 22  6  7 11 22 17 17  6  6 19 29 22 10
## [24]  4 31 33 35 32  8 33 37
```

```
faces_matrix <- vector()

#Plot the mismatched faces
for (i in 1:5) {
  if(predictions[i] != Y.test_2c[i]) {
    this_filename <- paste("CroppedYale/yaleB", pic_list[predictions[i]], "/yaleB", pic_list[predictions
    pred_face = read.pnm(file = this_filename)
    this_filename <- paste("CroppedYale/yaleB", pic_list[Y.test_2c[i]], "/yaleB", pic_list[Y.test_2c[i]
    test_face = read.pnm(file = this_filename)

    pred_face_matrix <- getChannels(pred_face)
    test_face_matrix <- getChannels(test_face)
    this_face_row <- cbind(pred_face_matrix, test_face_matrix)
    faces_matrix <- rbind(faces_matrix, this_face_row)
  }
}
faces <- pixmapGrey(faces_matrix)
plot(faces, main="1NNPrediction vs Test Face")
```

1NNPrediction vs Test Face

```
#----- END YOUR CODE BLOCK HERE -----#
```

Above is a plot of the first 4 mismatched faces. In total there are 27 mismatched faces.

Problem 3: Naive Bayes

i.) Load the titanic dataset and remove NA rows

```
titanic <- read.csv("Titanic.txt", header = TRUE, as.is = TRUE)
titanic <- titanic[!is.na(titanic$Age),]
```

ii.) Fit a gamma distribution to histogram of Age

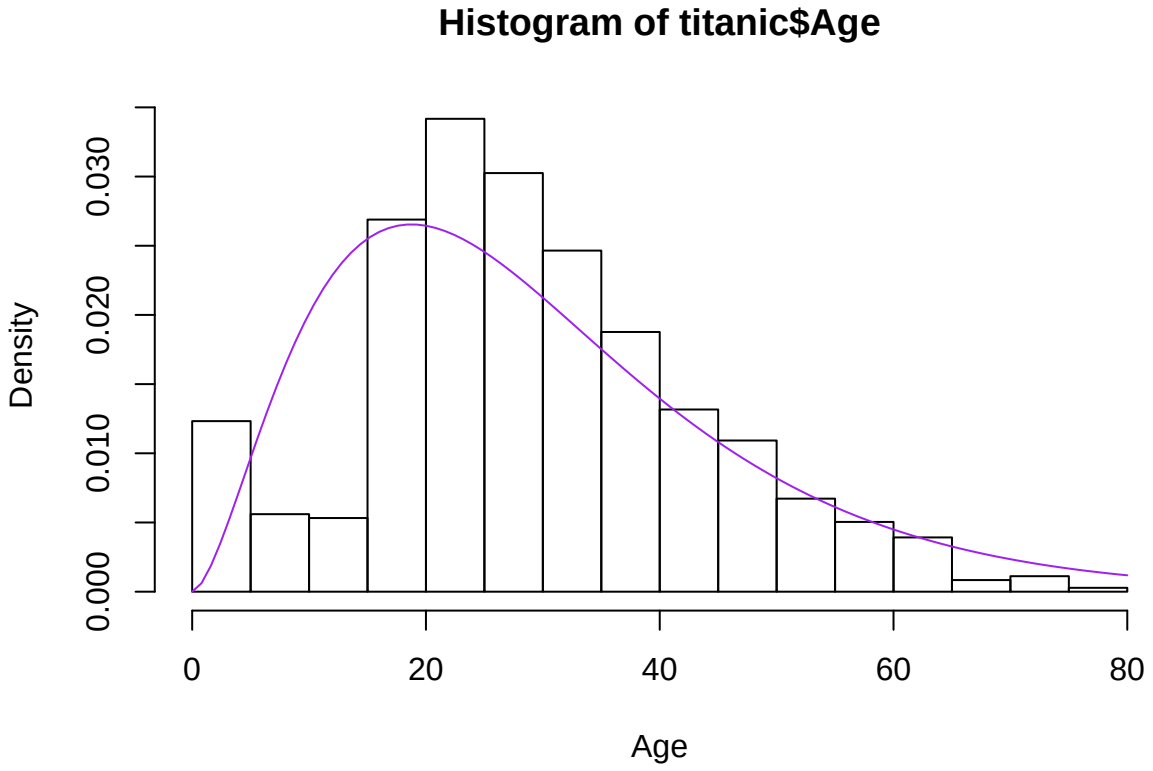
```
# Define negative log-likelihood
neg.gamma.ll <- function(params, data) {
  a <- params[1]
  s <- params[2]
  return(-sum(dgamma(data, shape = a,
                     scale = s, log = TRUE)))
}

# Run optimization procedure
my_params <- c(1,1)
# Extract minimum
suppressWarnings(nlm(neg.gamma.ll, my_params, data = titanic$Age)$minimum)
```

```
## [1] 2981.79
```

```
# Extract estimates
suppressWarnings(titanic.MLE <- nlm(neg.gamma.ll, my_params, data = titanic$Age)$estimate)
titanic.MLE
```

```
## [1]  2.715153 10.938291
```

```
hist(titanic$Age, breaks=20,xlab="Age",probability = T)
curve(dgamma(x, shape = titanic.MLE[1], scale = titanic.MLE[2]),add = TRUE, col = "purple")
```



```
titanic.MLE[2]
```

```
## [1] 10.93829
```

iii.) Code up the Naive Bayes Model

```
library(tidyverse)
```

```
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr
```

```
## Conflicts with tidy packages ----------------------------------------------
```

```
## filter(): dplyr, stats
## lag():    dplyr, stats
```

```
naiveBayesClassifier <- function(titanic_data, age, sex, pclass) {
  model_data <- titanic_data[, c("Age", "Sex", "Pclass", "Survived")]
  #head(model_data)

  #Get class probabilities
  class_prob <- model_data %>%
    group_by(Survived) %>%
    summarize(freq = n(), probClass = n() / nrow(model_data))
  ungroup(model_data)
  head(class_prob)

  #Conditional Sex Probability
  sex_cond_prob <- model_data %>%
    group_by(Sex, Survived) %>%
    summarize(count = n()) %>%
    left_join(class_prob[, c("Survived", "freq")], by="Survived") %>%
    mutate(probSex = count / freq)
  ungroup(model_data)
  head(sex_cond_prob)

  #Conditional Pclass probability
  pclass_cond_prob <- model_data %>%
    group_by(Pclass, Survived) %>%
    summarize(count = n()) %>%
    left_join(class_prob[, c("Survived", "freq")], by="Survived") %>%
    mutate(probPassenger = count / freq)
  ungroup(model_data)
  head(pclass_cond_prob)

  #Get Age MLE estimates
  suppressWarnings(titanic.MLE <- nlm(neg.gamma.ll, my_params, data =
                                        titanic_data$Age)$estimate)

  classify <- class_prob %>%
    left_join(filter(sex_cond_prob, Sex == sex)[, c("Survived", "probSex")],
              by="Survived") %>%
    left_join(filter(pclass_cond_prob, Pclass == pclass)[, c("Survived", "probPassenger")],
```

```
              by="Survived") %>%
    mutate(result = probClass*dgamma(age, shape = titanic.MLE[1], scale = titanic.MLE[2])*
              probSex, probPassenger)
  return(which.max(as.numeric(unlist(classify[, "result" ]))))-1)
}

naiveBayesClassifier(titanic, 23, "male", "1")
```

```
## [1] 0
```

iv.) Test the above Naive Bayes model using LOOCV and 10-fold cross validation

```
#Leave one out cross validation
test_err <- c()
for (k in 1:nrow(titanic)) {
  train_titanic <- titanic[-k, ]
  test_titanic <- titanic[k, ]
  test_err[k] <- abs(naiveBayesClassifier(train_titanic, test_titanic[1, "Age"], test_titanic[1, "Sex"]
}
loocv_err <- mean(test_err)
cat("Test Error from Leave One out Cross Validation is: ", loocv_err)
```

```
## Test Error from Leave One out Cross Validation is:  0.219888
```

```
#10-fold cross validation
set.seed(42)
num_folds <- 10
num_Passengers <- nrow(titanic)

ndx <- sample(1:num_Passengers, num_Passengers, replace=F)

titanic_folds <- titanic[ndx, ] %>%
  mutate(fold = (row_number() %% num_folds) + 1)

#head(titanic_folds)
test_err <- c()
fold_test_err <- c()
for (f in 1:num_folds) {
  titanic_train <- filter(titanic_folds, fold != f)
  titanic_test <- filter(titanic_folds, fold == f)
  for (v in 1:nrow(titanic_test)) {
    fold_test_err[v] <- abs(naiveBayesClassifier(titanic_train, titanic_test[v, "Age"], titanic_test[v,
  }
  test_err[f] <- mean(fold_test_err)
}
avg_test_err <- mean(test_err)
cat("Test Error from 10-fold Cross Validation is: ", avg_test_err)
```

```
## Test Error from 10-fold Cross Validation is:  0.2252934
```

v.) Run a logistic model on the Titanic data set. The code follows

```
titanic <- read.csv("Titanic.txt", header = TRUE, as.is = TRUE)
titanic <- titanic[!is.na(titanic$Age),]
model <- glm(formula = factor(Survived) ~ factor(Pclass)+Sex+Age,
family = binomial(link = "logit"), data = titanic)
summary(model)
```

```
##
## Call:
## glm(formula = factor(Survived) ~ factor(Pclass) + Sex + Age,
##     family = binomial(link = "logit"), data = titanic)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7303  -0.6780  -0.3953   0.6485   2.4657
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)       3.777013   0.401123   9.416  < 2e-16 ***
## factor(Pclass)2 -1.309799   0.278066  -4.710 2.47e-06 ***
## factor(Pclass)3 -2.580625   0.281442  -9.169  < 2e-16 ***
## Sexmale         -2.522781   0.207391 -12.164  < 2e-16 ***
## Age             -0.036985   0.007656  -4.831 1.36e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 964.52  on 713  degrees of freedom
## Residual deviance: 647.28  on 709  degrees of freedom
## AIC: 657.28
##
## Number of Fisher Scoring iterations: 5
```

vi.) Consider classifying (predicting) the two levels of Survived by labeling a case survived = 1 when its predicted success probability is above 0.5.

```
p <- .5
linear.prediction <- predict(model)
predicted.classes <- exp(linear.prediction)/(1+exp(linear.prediction))>p

#Leave one out cross validation
test_err <- c()
for (k in 1:nrow(titanic)) {
  train_titanic <- titanic[-k, ]
  test_titanic <- titanic[k, ]
  model <- glm(formula = factor(Survived) ~ factor(Pclass)+Sex+Age,
family = binomial(link = "logit"), data = train_titanic)
  linear.prediction <- predict(model, test_titanic)
predicted.classes <- exp(linear.prediction)/(1+exp(linear.prediction))>p
  test_err[k] <- abs(as.numeric(predicted.classes) - test_titanic[1,"Survived"])
}
loocv_err <- mean(test_err)
cat("Test Error from Leave One out Cross Validation is: ", loocv_err)
```

```
## Test Error from Leave One out Cross Validation is:  0.2114846
```

```
#10-fold cross validation
set.seed(42)
num_folds <- 10
num_Passengers <- nrow(titanic)

ndx <- sample(1:num_Passengers, num_Passengers, replace=F)
```

```
titanic_folds <- titanic[ndx, ] %>%
  mutate(fold = (row_number() %% num_folds) + 1)

test_err <- c()
for (f in 1:num_folds) {
  titanic_train <- filter(titanic_folds, fold != f)
  model <- glm(formula = factor(Survived) ~ factor(Pclass)+Sex+Age,
family = binomial(link = "logit"), data = titanic_train)
  titanic_test <- filter(titanic_folds, fold == f)

  linear.prediction <- predict(model, titanic_test)
  predicted.classes <- exp(linear.prediction)/(1+exp(linear.prediction))>p
  test_err[f] <- mean(abs(as.numeric(predicted.classes) - titanic_test$Survived))
}
#test_err
avg_test_err <- mean(test_err)
cat("Test Error from 10-fold Cross Validation is: ", avg_test_err)
```

```
## Test Error from 10-fold Cross Validation is:  0.2086463
```

vi.) Compare the test erros from your methods

Error Type	NaiveBayes	Logistic Regression
LOOCV	.220	.211
10-fold Cross	.225	.209

As we can see, all the errors are pretty close to each other for the different classification methods. The errors from logisitc regression is slightly better but the ease of programming and using Naive Bayes might be something to consider when determining which model to use. In addition, the two validation methods yeild similar results so either can be used.