Lab 6: Facial Recognition Using Nearest Neighbor

Jared Gridley

March 2021

Prelude: Helper Functions from Prelab6

`faceplot` makes a plot of faces and arranges them on a grid. The code is below:

```
# A function to help plot faces
faceplot <- function(xx,width=64,midcolor="grey10",gcols=2,labels=row.names(xx)) {
# Note require(ggplot2); require(reshape2); require(gridExtra)
 if (is.vector(xx)) {
   xx <- matrix(xx,nrow=1)
}
  pl <- vector("list",nrow(xx))
  for(i in 1:nrow(xx)){
    face <- matrix(xx[i,], nrow=width)
    face.m <- melt(apply(face, 2, function(x) as.numeric(rev(x))))
    pl[[i]] <- ggplot(data=face.m) + geom_raster(aes(x=Var2, y=Var1, fill=value)) +
        theme(axis.text=element_blank(), axis.title=element_blank()) + guides(fill=FALSE) +
        scale_fill_gradient2(low="black", mid=midcolor, high="white") +ggtitle(labels[i])
  }
  grid.arrange(grobs=pl,ncol=gcols)
}
```

`norm2` computes the norm of a vector.

`Matchimage` finds the index of the image in the matrix `image.matrix` that most nearly matches `refimage`.

```
# Function to compute 2-norm
norm2 <- function(x) sqrt(as.vector(x)%*%as.vector(x))

# Function to find closest match
Matchimage <- function(image.matrix, refimage)
  # image.matrix contains the imaages
  # refimage contains the image
  # returns the index of the closest imaige in image.matrix
  {
  dist2myimage<-apply(image.matrix,1,function(x)norm2(x-refimage))
  q <- which.min(dist2myimage)
}
```

Overview

WeCU wants a system for facial recognition that uses the **Nearest Neighbor** algorithm (NN). NN works by taking an image, computing the closest point in the faces dataset, and then returning that closest point as the predicted label of the image. If the predicted label matches the true label of the person, then the system

has correctly identified the person. *You have been hired to improve their current system.*

Preparing Data

WeCU has scaled the images to have unit length and created a train and test partition of the dataset.

```
# Read in data
F.df <- read.csv('~/MATP-4400/data/faces.csv')

# Save first column as labels
labels <- as.numeric(F.df[,1])
F.matrix<-as.matrix(F.df[,-1])

# Assign the row names
row.names(F.matrix)<-1:(nrow(F.matrix))

# Scale the images to have unit length
rownorms<-apply(F.matrix, 1, function(x){sqrt(sum(x^2))})
Fbar.matrix<-diag(1/rownorms) %*% F.matrix

# Divide into trainface and  testface
# Since images occur in blocks of 10, work mod 10 (using %%10) and get  images 4-9 as train and 0-3 as
gg <- c(1:400)
h <- gg[ gg%%10 >3]

#h is the list of numbers of faces to extract
trainface.uncentered<- Fbar.matrix[h,]
trainfacelabels<-labels[h]

#Test face
gg <- c(1:400)
h <- gg[ gg%%10 <= 3  ]

#h is the list of numbers of faces to extract
testface.uncentered<- Fbar.matrix[h,]
testfacelabels<-labels[h]
```

Centering the Data

The *training mean image* is calculated as `train.mean`, then both the training and test data are centered using the same train mean.

```
# Compute the train mean face and save in train.mean
train.mean <- colMeans(trainface.uncentered)

# Mean center the training data and save in trainface.
trainface.centered <- trainface.uncentered-matrix(1,ncol=1,nrow=nrow(trainface.uncentered)) %*% train.m
```

Nearest Neighbor Facial Recognition System

To improve WeCU's current system, you must now write a program that computes the number of images that are correctly classified by the original NN system based on `trainface.centered`. An image is "correct""

if the NN label of the image is equal to the label of the original image. The following code calculates the nearest neighbor of the images in `testface.uncentered` and reports the percentage correctly identified.

```
nimages<-nrow(testface.uncentered)
correctcount<-0
for (j in 1:nimages) {

  # find the closest image that isn't me
  match<-Matchimage(trainface.centered,testface.uncentered[j,])
  if (trainfacelabels[match]==testfacelabels[j])
  {
  correctcount<-correctcount+1
  }
}
# Report percentage correct
correctcount/nimages*100
```

```
## [1] 45
```

WeCU is not happy because there system is only identifies 45% of the images correctly. *Can you help?*

Exercise 1

Bingo, you have an idea. Since you are a good data scientist, you know that the train and test sets should be scaled identically. `testface.uncentered` is not centered but `trainface` is. Revise the algorithm so that each image in testface.uncentered is first centered by substracting train.mean and the nearest neigbor in `trainface` is calculated. *Evaluate the accuracy of the revised algorithm.*

```
# Centering the test faces (uncentered - mean)
testface.centered <- testface.uncentered-matrix(1,ncol=1,nrow=nrow(testface.uncentered)) %*% train.mean
nimages<-nrow(testface.centered)

correctcount <- 0
# Report percentage correct
for (j in 1:nimages) {

  # find the closest image that isn't me
  match<-Matchimage(trainface.centered,testface.centered[j,])
  if (trainfacelabels[match]==testfacelabels[j])
  {
  correctcount<-correctcount+1
  }
}
# Report percentage correct
correctcount/nimages*100
```

```
## [1] 85.625
```

Discuss the accuracy of the new system you developed. The accuracy for our system is much better than on the uncentered test data. This makes sense that it would be better on the centered test data because it would emphasize the more distinct features about the face, i.e. with the mean face removed, there is less similarity so the centered data will be scaled to match the distinct features more closely of the training set.

PCA Based Facial Recognition

WeCU's current facial recognition system requires storage of a 400x4096 matrix which contains 1,638,400 numbers. They have hired you to make a more efficient system that requires less storage. You decide to use PCA to compress the data using the scalar projections. The new system will compute the Nearest Neighbor using the scalar projections on the first K components of PCA instead of the original data.

From `Prelab6` we compute the principal components of the centered training data in `trainface`. Note that we precentered and scaled the data, so we didn't let `prcomp` rescale it. Let `V` be a matrix containing the first 50 principal components.

```
# Running PCA on test in train data
my.pca<- prcomp(trainface.centered,retx=TRUE, scale=FALSE,center=FALSE)

#Getting the first 50 Principal Components
#head(t(summary(my.pca)$importance),n=50)
V <- as.matrix(my.pca$rotation[,1:50])
```

Exercise 2

From the last exercise, you know that it is very important that you center and scale the data used for training and testing exactly the same way. Verify that `(x-train.mean) %*% V` (here x means an arbitrary point) computes the scalar projections on `PC1` through `PC50` by comparing the computed scalar projections for the first point in `trainface.uncentered` with the scalar projections for the first point in `my.pca` by computing the 2-norm of their difference. In this case, 'x<-trainface.uncentered[1,]'.

```
# Center the test data
test_data <- testface.centered %*% V

#Getting the projection from my pca.
pca.proj <- my.pca$rotation[1,1:50]

#Computin 2norm of their difference.
diff <- norm2(test_data[1,] - pca.proj)
diff
```

```
##           [,1]
## [1,] 0.2563187
```

Exercise 3

Compute the accuracy of the new PCA NN system on the testing data for `K=50` (use the programs you wrote in Exercise 1,2) on the total. The accuracy is defined as the number of correctly classified images divided by the total number of images. *Make sure you correctly center and compute the scalar projections of the testing data!*

```
# Making the test data
C <- trainface.centered %*% V[,1:50]

nimages<-nrow(test_data)
correctcount <- 0
# Report percentage correct
for (j in 1:nimages) {

  # find the closest image that isn't me
```

```
  match<-Matchimage(C,test_data[j,])
  if (trainfacelabels[match]==testfacelabels[j])
  {
  correctcount<-correctcount+1
  }
}
# Report percentage correct
correctcount/nimages*100
```

```
## [1] 83.75
```

Compare the performance of the `K = 50` PCA-based system with the performance of the original system using all of the data. Specifically,

- How do the number of points correctly classified differ?

- How much storage is required by your proposed system versus the system provided.
- When might the NN system based on PCA be more advantageous than a NN system based on the original data?

The PCA based system performed very similarly (but slightly worse) than the original system based on the data. The PCA based system was 83.75% accuracte while the regular system was only 85.625% accurate. So overall the system based on all the data performed slightly better, which is what we expected because the PCA method relies on projecting the original data, so it loses out on some of the features. However, the PCA System was noticably faster when running it. In terms of space complexity, the PCA system is much better because it does not store all of the images, it only stores their PCA projections which still yeilds similar accuracy. The PCA system would be better when there is a lot more data to train and test on. This is because it would significantly reduce the storage required and would run with a faster runtime, while still getting a similar accuracy to that of all the original data.

Bonus Exercise 3pts

Do a computational study to see what number of PCA components yields the most accurate results.

```
#Trying PCA with 100 components.
V_109 <- as.matrix(my.pca$rotation[,1:109])
test109 <- testface.centered %*% V_109
C_109 <- trainface.centered %*% V_109[,1:109]

#Testing on 100 components
nimages<-nrow(test109)
correctcount <- 0

for (j in 1:nimages) {
  match<-Matchimage(C_109,test109[j,])
  if (trainfacelabels[match]==testfacelabels[j])
  {
  correctcount<-correctcount+1
  }
}
correctcount/nimages*100
```

```
## [1] 85
```

```
#Testing on 150 components
V_test <- as.matrix(my.pca$rotation[,1:110])
```

```
test_d <- testface.centered %*% V_test
C_test <- trainface.centered %*% V_test[,1:110]

nimages<-nrow(test_d)
correctcount <- 0

for (j in 1:nimages) {
  match<-Matchimage(C_test,test_d[j,])
  if (trainfacelabels[match]==testfacelabels[j])
  {
  correctcount<-correctcount+1
  }
}
correctcount/nimages*100
```

```
## [1] 85.625
```

Thus we can see that, obviously the more training data we have the better the algorithms will be, which was seen with an increase in the accuracy when tested with more training data. However, the accuracy converged to the same accuracy we got from testing on the method with all of the data. I tested it on 160, 150, 140, 130, 120, and 110, and I saw that the accuracy was the same as when using all the data. So to get the biggest bang for you buck on this dataset, using 110 principle components will give you the same accuracy as all the data and it will shave down teh runtime a bit when compared to running on all of the data.