CSCI454_hw2

Jen Johnson

10/17/2017

Training Data

Read in the training data.

```
training <- c()

for (file in filenames){
  img <- readPNG(paste("/Users/jen/Dropbox/CSCI 454/hw/trainingfaces2", file, sep = "/"))
  v.img <- as.vector(img)
  training <- cbind(training, v.img)
}
```

Store mean of each row in last column in dataset. Subtract mean to center the data. Transpose. Calculate the covariance matrix.

```
training <- cbind(training, apply(training, 1, mean))

X <- training[ , 1:dim(training)[2]-1]-training[ ,dim(training)[2]]

trans.X <- t(X)

C <- X %*% trans.X
```

Calculate eigen vectors and values. Subset to only include the best/first 10 vectors/columns with highest eigen values. Make transpose for use with testing set.

```
EV <- eigen(C)

Top.10 <- EV$vectors[ , 1:10]

trans.EV <- t(Top.10)
```

Testing Data

Read in the testing data.

```
filenames <- list.files(path = "/Users/jen/Dropbox/CSCI 454/hw/testingfaces2")

testing <- c()

for (file in filenames){
  img <- readPNG(paste("/Users/jen/Dropbox/CSCI 454/hw/testingfaces2", file, sep = "/"))
  v.img <- as.vector(img)
  testing <- cbind(testing, v.img)
 }
```

Normalize testing data by subtracting the mean of the training data.

```
testing <- testing - training[ ,dim(training)[2]]
```

Compute the weight for each image in the testing dataset by multiplying the normalized column by the transpose of the top 10 eigenvectors. This will result in a weight vector of length 10 for each image in the testing data. Store each weight vector into a matrix.

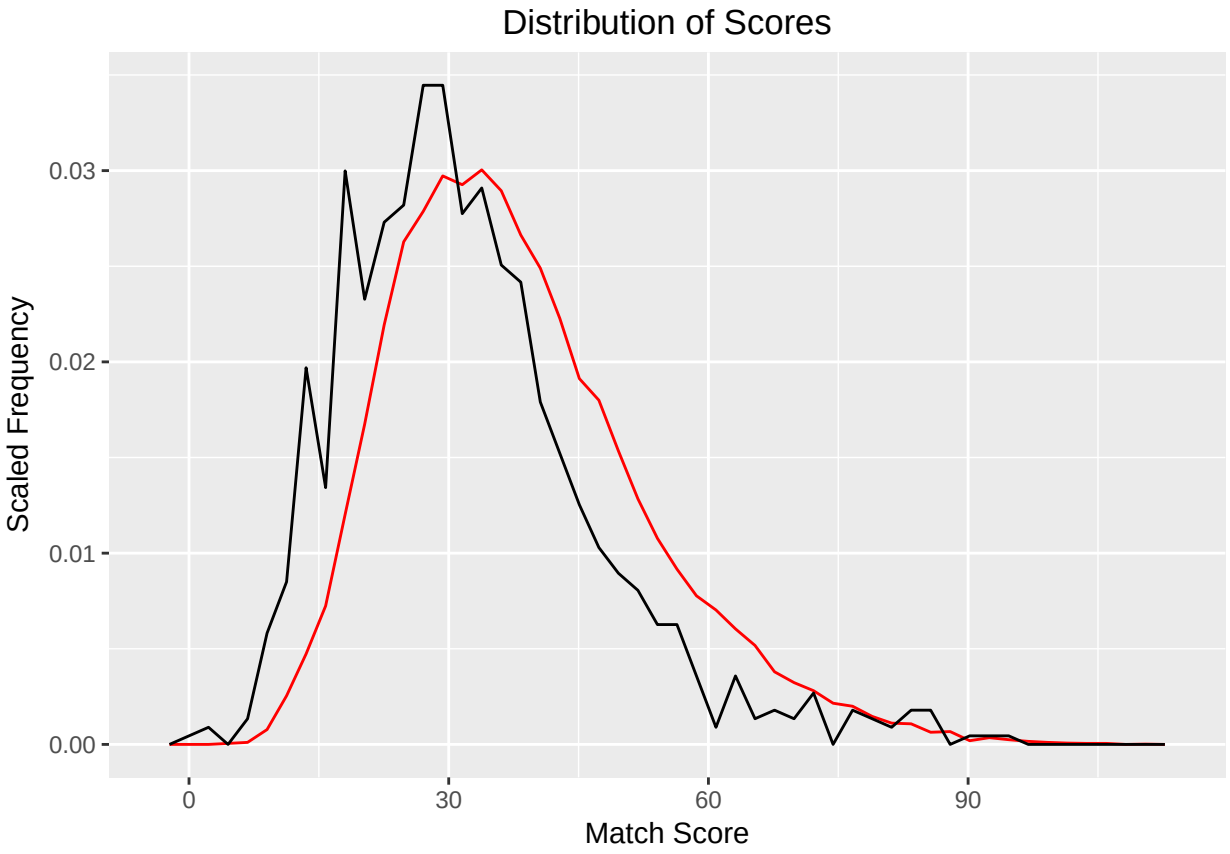
```
weight.matrices <- c()

for (i in 1:dim(testing)[2]){
  current.img <- testing[ ,i]
  current.weight <-  trans.EV %*% current.img
  weight.matrices <- cbind(weight.matrices, current.weight)
}
```

Compare all images in the testing set to each other by subtracting their weigth matrices from each other, taking the absolute value, and then summing over all 10 values. Store all comparisons (with labels, in the form of a 3 column data frame) in all.data and store the score in genuine and imposter lists based on their filename.

```
genuine <- c()
imposter <- c()
all.data <- matrix(ncol = 3)
colnames(all.data) <- c("image1", "image2", "score")

for (i in 1:length(filenames)){
  image1 <- filenames[i]
  subject1 <- substr(image1, 2, 3)

  j <- i+1
  while (j <= length(filenames)){
    image2 <- filenames[j]
    subject2 <- substr(image2, 2, 3)

    weight1 <- weight.matrices[, i]
    weight2 <- weight.matrices[, j]
    weight.diff <- sum(abs(weight1-weight2))

    current.row <- cbind(image1, image2, weight.diff)
    all.data <- rbind(all.data, current.row)

    if (subject1 == subject2) {
      genuine <- c(genuine, weight.diff)
    } else {
      imposter <- c(imposter, weight.diff)
    }

    j <- j+1
  }
}

# write.csv(all.data, "JJohnsonMatchingScores.csv")
```

Plot distribution from HW1

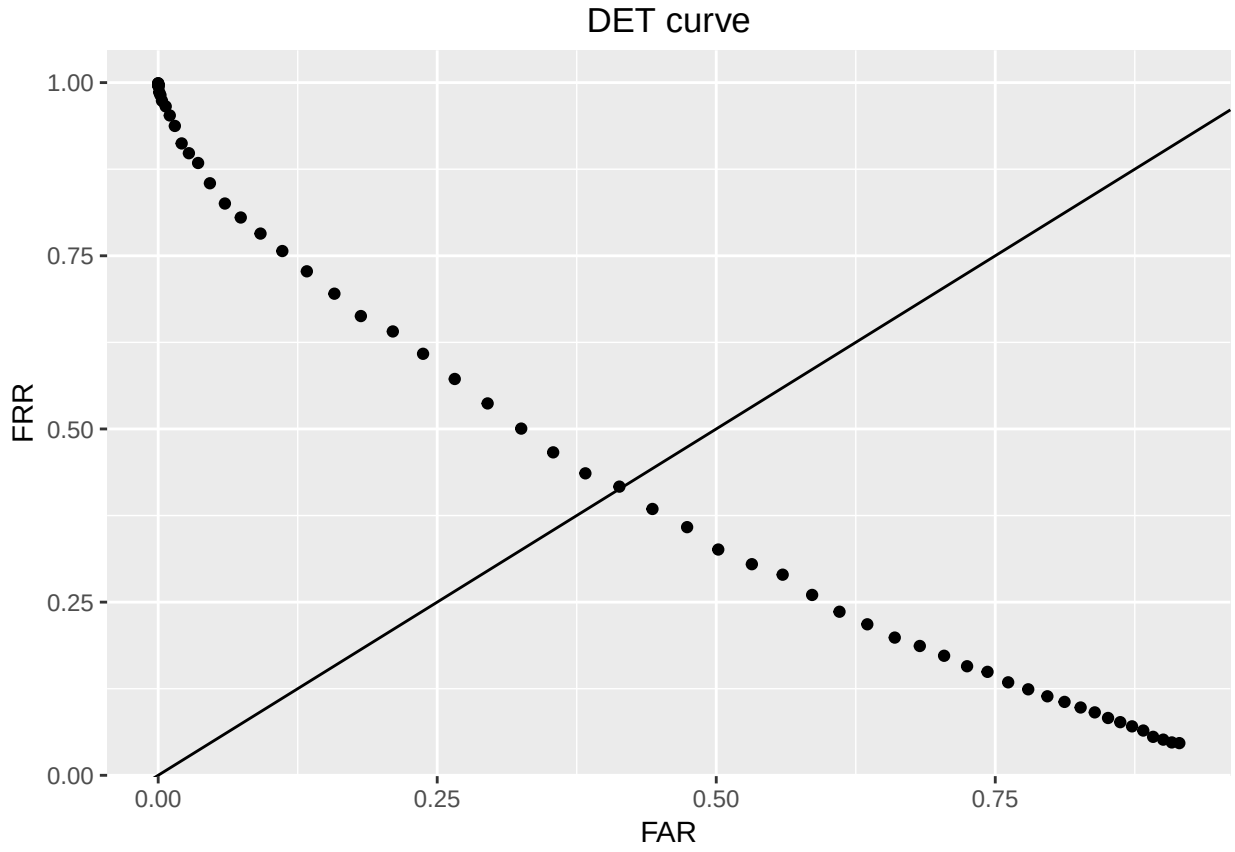
```
library(ggplot2)
theme_update(plot.title = element_text(hjust = 0.5))

imposter <- as.data.frame(imposter)
genuine <- as.data.frame(genuine)

#Scale using the density function and plot using ggplot's geom_freqpoly.
ggplot() + geom_freqpoly(data = imposter, aes(x = imposter, y = ..density..), bins = 50, color = "red")
```



DET from HW1

```
FAR_vs_FRR <- NULL

#For each value of t, calculate FAR and FRR and add to dataset.

for (t in seq(from = 0, to = 60.0, by = 1)){

  false_accept_count <- sum(imposter < t)
  false_accept_rate <- false_accept_count/dim(imposter)[1]
  false_reject_count <- sum(genuine > t)
  false_reject_rate <- false_reject_count/dim(genuine)[1]
```

```
  current_row <- c(false_accept_rate, false_reject_rate)
  FAR_vs_FRR<- rbind(FAR_vs_FRR, current_row)
}

rates_data_frame <- as.data.frame(FAR_vs_FRR)
colnames(rates_data_frame) <- c("FAR", "FRR")
ggplot(rates_data_frame, aes(x=FAR, y = FRR)) + geom_point() + geom_abline(slope = 1, intercept = 0) + l
```



EER from HW1

```
#Make new column containing boolean FAR > FRR.
rates_data_frame$larger <- rates_data_frame$FAR > rates_data_frame$FRR

#Find where FAR becomes less than FRR. Use these as upper and lower boundaries to estimate the EER.
far_is_smaller <- rates_data_frame[rates_data_frame$larger=="FALSE", ]
lower_bound <- max(far_is_smaller$FAR)
far_is_larger <- rates_data_frame[rates_data_frame$larger=="TRUE", ]
upper_bound <- min(far_is_larger$FAR)

EER <- mean(lower_bound, upper_bound)
print(EER)
```

```
## [1] 0.4132192
```