

## Machine Learning - Coursework 1

Calculate and plot the average face of the training set, then write a function to find a PCA basis of size M, where the inputs will be M and X, the matrix containing the training set. Clearly describe all aspects of your function, then use it to plot the first 5 eigenfaces of the training set.

To calculate the average face of the training set, I simply calculate the average pixel values for each pixel across the trainings set.

```

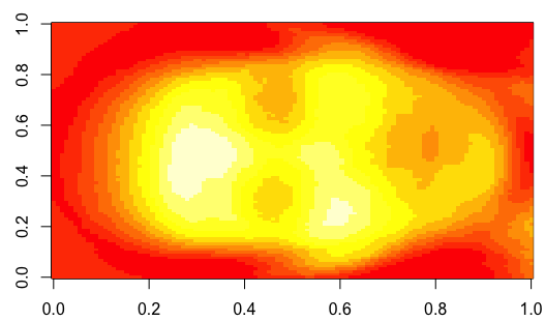
1 library(rARPACK)
2 library(philentropy)
3 # I load the raw csv's
4 faces.train.inputs <- read.csv("./2018_ML_Assessed_Coursework_1_Data/
5                               Faces_Train_Inputs.csv", head=FALSE)
6 faces.train.label <- read.csv("./2018_ML_Assessed_Coursework_1_Data/
7                               Faces_Train_Labels.csv", head=FALSE)
8 faces.test.inputs <- read.csv("./2018_ML_Assessed_Coursework_1_Data/
9                               Faces_Test_Inputs.csv", head=FALSE)
10 faces.test.label <- read.csv("./2018_ML_Assessed_Coursework_1_Data/
11                               Faces_Test_Labels.csv", head=FALSE)
12 # I turn the input values into a list of 320 matrices, each matrix a 112 x 92 value
13 # of pixels corresponding to each image .. I need to use lapply again on the
14 #result because apply gives the matrices in a weird form
15 faces.train.inputs.cleaned <- lapply(apply(X=faces.train.inputs,
16                                           MARGIN=1,
17                                           function(x) list(matrix(as.numeric(x),
18                                           nrow = 112))), "[", 1)
19 # Here I calculate the average face
20 avg.face <- Reduce('+', faces.train.inputs.cleaned) /
21   length(faces.train.inputs.cleaned)
22 image(avg.face)

```

Please see figure 1 for results.

The following function returns the PCA basis of size M as specified. I added a default parameter which also allows access to the eigenvalues associated with each vector of the eigenbasis.

Figure 1: Average face of the training set



The function works exactly as the PCA described in lectures. Calculate the centralised data matrix,  $X$ , and then calculate the first  $M$  eigenvectors for the covariance matrix  $\frac{XX^T}{n}$ .

```

1 find.pca.basis <- function(M,X, return.full.results = FALSE){
2   n <- dim(X)[1] # The number of images
3   # Turn the input data into a matrix and transpose it
4   X.data.matrix <- data.matrix(t(X))
5   # Centralise the data matrix
6   means <- rowMeans(X.data.matrix) # calculate row means
7   data.matrix.centralised <- X.data.matrix - means %*% t(rep(1,n)) # and subtract
8   # Calculate the covariance matrix as defined in lectures
9   covariance.matrix <- (data.matrix.centralised %*% t(data.matrix.centralised)) / n
10  # Now I need to compute the first M eigenvectors/ eigenvalues using the
11  # R package rARPACK
12  results <- eigs.sym(covariance.matrix,k=M,which="LM")
13  if (return.full.results){
14    return(results)
15  } else{
16    return(results$vectors)
17  }
18 }

```

The code below then plots the first 5 eigenfaces of the training set. This means simply calculating the eigenbasis for  $M = 5$ , and then plotting the vectors in the resulting eigenbasis.

```

1 eigenbasis <- find.pca.basis(5,faces.train.inputs)
2 par(mfrow=c(2,3))
3 for (i in 1:5){
4   # eigenbasis[,i] corresponds to the i'th eigenvector.
5   image(matrix(eigenbasis[,i], nrow = 112),useRaster=TRUE, axes=FALSE,main=i)
6 }
7 par(mfrow=c(1,1))

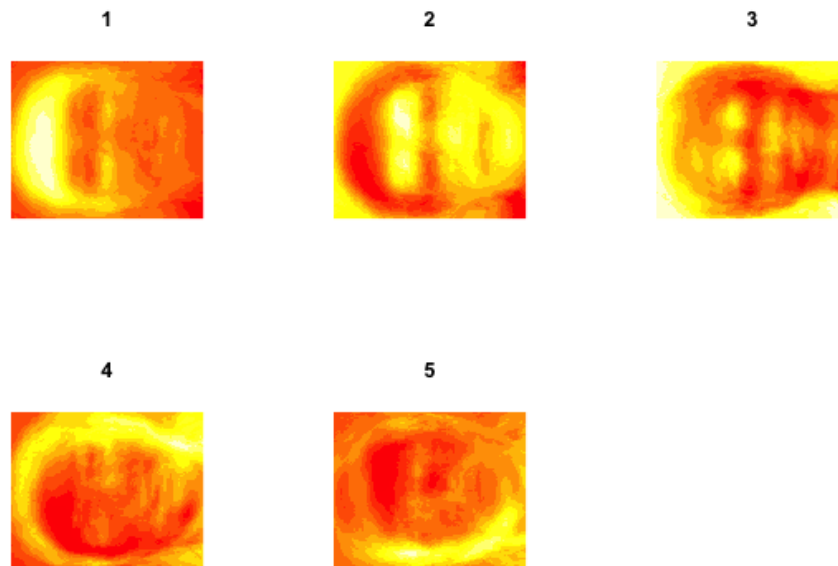
```

The results can be seen in figure 2.

- **Choose a single face and project it into a PCA basis for dimension  $M = 5, 10, 50$ , then plot the results.**

Here is the code to project the first image of the training set onto the PCA basis for dimensions 5,10 and 50.

Figure 2: First 5 Eigenfaces of the training set



```

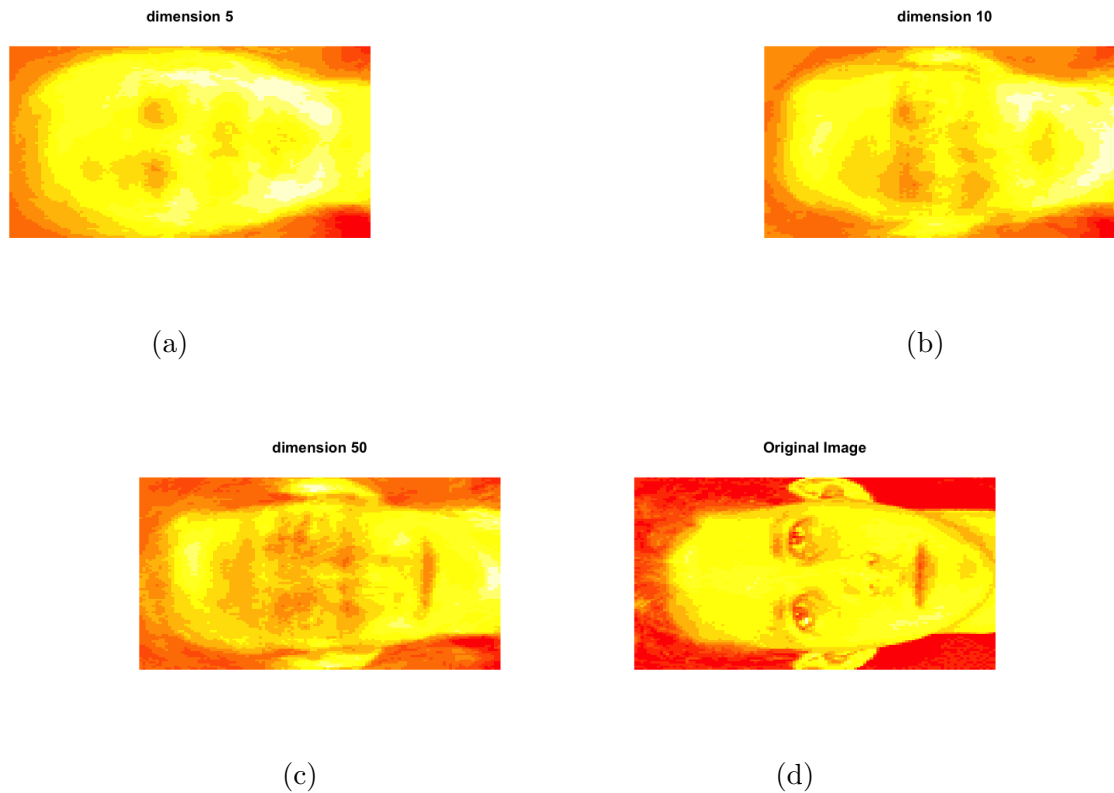
1 # initialise the dimensions and face used.
2 dimensions <- c(5,10,50)
3 single.face <- 1
4 means <- as.vector(avg.face) # convert the mean face back into a vector
5
6 # iterate through the dimensions considered
7 for (i in dimensions){
8
9   # compute the eigenbasis using the function created
10  eigenbasis <- find.pca.basis(i,faces.train.inputs)
11  # calculate the projection values in this pca basis
12  projection.vals <- t(as.numeric(faces.train.inputs[single.face,]) -
13                      means) %*% eigenbasis
14  # calculate the actual face in this basis
15  projection.vector <- eigenbasis %*% as.numeric(as.list(projection.vals))
16
17  # carry out the plot
18  image(matrix(projection.vector, nrow = 112),useRaster=TRUE,
19        axes=FALSE,main=paste("dimension",i,sep=" "))
20 }
21
22 # and here's the original image
23 image(matrix(as.numeric(faces.train.inputs[single.face,]), nrow = 112),
24        useRaster=TRUE, axes=FALSE, main="Original Image")

```

Please see Figure 3 for the results of this code.

- Plot a graph of the mean squared error of each lower dimensional approximation of this chosen face, with the dimensionality plotted along the x-axis. Is there a clear point at which we can choose a good approximation? Discuss how we should choose the appropriate dimensionality

Figure 3: The projection of training image 1 onto PCA bases of different dimensions



of the approximation.

**setup:** We can derive a nice form for the mean square error. Let  $X$  be our **centralised** data matrix with  $n$  observations of  $p$  variables. Consider PCA of  $k$  dimensions. Let  $\hat{X}$  be the projection of  $X$  onto this basis of dimension  $k$ . Denote the  $i$ 'th eigenvector of the covariance matrix of  $X$  by  $Y_i$ .

**form of MSE:** Now we have  $E[||X||^2] = \sum_{i=1}^n \text{Var}(X_i)$ . From the spectral decomposition of the covariance matrix we have that  $\sum_{i=1}^n \text{Var}(X_i) = \sum_{i=1}^n \lambda_i$ , where the  $\lambda_i$  correspond to the eigenvalues of the covariance matrix we calculate during the PCA algorithm. Then we have that the  $MSE := E[||X - \hat{X}||^2] = \sum_{i=1}^n \lambda_i - \sum_{i=1}^k \text{Var}(Y_i) = \sum_{i=1}^n \lambda_i - \sum_{i=1}^k \lambda_i = \sum_{i=k+1}^n \lambda_i$

We approximate the true total variance here by the sum of the first 320 eigenvalues. Please see figure 4 for the results of this code.

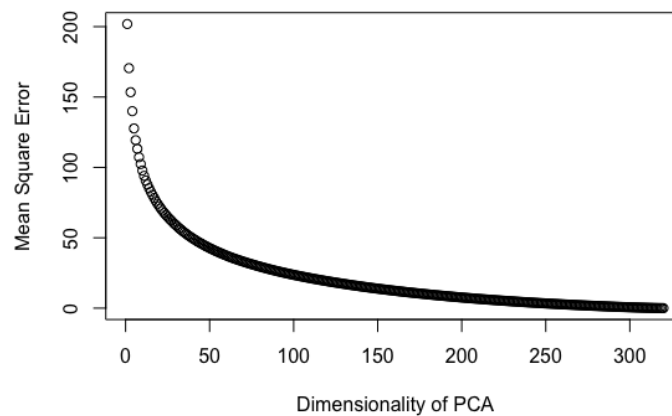
```

1 # First I calculate the full pca basis (and all the associated eigenvalues)
2 full.results <- find.pca.basis(320,faces.train.inputs,return.full.results = TRUE)
3
4 # the mean square error is equal to the total variance minus the sum of the
5 # eigenvalues for components not used
6 mses <- (cumsum(full.results$values)[length(full.results$values)] -
7          cumsum(full.results$values) )
8 plot(mses,xlab="Dimensionality of PCA",ylab="Mean Square Error")

```

From the graph in figure 4, we see that the mean square error rapidly decreases in linear steps of dimensionality up to around  $k = 50$ . After this point, the rate of decay is slower

Figure 4: Mean Squared Error as a function of PCA dimension



and it suggests that taking further dimensions does not improve the mean square error at the same rate. This "elbow point" of 50 is where we should choose our appropriate dimension.

- Write a function implementing a K-nearest neighbour classifier and investigate its use on the face recognition dataset. Make some recommendations regarding how to best set up this algorithm for this particular application.

Here is the code for my K-nearest neighbour classifier

```

1 k.nearest.neighbours <- function(training.data.matrix, training.data.labels,
2                                   testing.data.matrix, K = 4,
3                                   distance.type = "squared.euclidean"){
4   # Make sure all the data is numeric
5   training.data.matrix <- data.matrix(training.data.matrix)
6   testing.data.matrix <- data.matrix(testing.data.matrix)
7   training.data.labels <- as.numeric(training.data.labels)
8   # Initialise the list which will take the classifications
9   classifiers <- c()
10  # iterate through every row of the testing matrix
11  for (i in 1:dim(testing.data.matrix)[1]){
12    # compute the distance of this row of the testing matrix to every other
13    # row in the training set
14    all.distances <- apply(training.data.matrix, MARGIN=1,
15                           function(x) distance(rbind(testing.data.matrix[i,],x),
16                                                    method=distance.type))
17    # sort these distances in increasing order.
18    sorted.distances <- sort(all.distances, index.return=TRUE)
19
20    # Look at the k closest rows in the training set to this testing row.
21    # Whichever classification comes up the most - is the classification we
22    # will give this particular row.
23    all.counts <- tabulate(training.data.labels[sorted.distances$ix[1:K]])
24    sorted.counts <- sort(all.counts, index.return=TRUE, decreasing=TRUE)

```

```
25     ## these are all the voters which share the maximum score
26     max.votes <- which(sorted.counts$x == sorted.counts$x[1])
27     if (length(max.votes) > 1){
28         # if it's a split vote, randomly select one.
29         voter <- sample(1:length(max.votes),size=1)
30     } else {
31         # else, there's only one.
32         voter <- 1
33     }
34     # set the classification.
35     classification <- sorted.counts$ix[voter]
36     classifiers <- c(classifiers,classification)
37
38 }
39
40 return(classifiers)
41
42 }
```