Stevens Institute of Technology

Department of Computer Science

CS559:Machine Learning Fundamentals and Applications

Prof. Tian Han

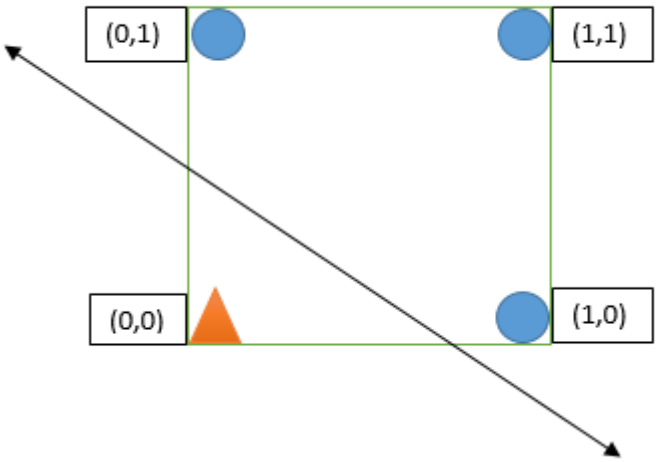
Homework Assignment 2

Submitted by: **Malav Shastri**

CWID**: 10456244**

Problem 1 (30pt): [Perceptron Algorithm] In this problem, we learn the linear discriminant function for boolean OR function. Suppose we have two dimensional x = (x1,x2), x1 and x2 can be either 0 (false) or 1 (true). The boolean OR function is defined as: f(x1,x2) = x1 OR x2. Specifically, f(0,0) = false, f(1,0) = true, f(0,1) = true, and f(1,1) = true where true can be treated as positive class and false can be treated as negative class. You can think of this function as having 4 points on the 2D plane (x1 being the horizontal axis and x2 being the vertical axis): P1 = (0,1),P2 = (1,1),P3 = (1,0),P4 = (0,0), P1,P2,P3 in positive class and P4 in negative class.

(1) [5pt] For boolean OR function, is the negative class and positive class linearly separable? (2) **Solution:** Boolean OR specifies the input in negative class only when both the input features are negative i.e (0,0). For all other cases it classifies the output in the positive class so yes it is linearly separable. Also from the figure given below we can conclude the same.



(2) [25pt] Is it possible to apply the perceptron algorithm to obtain the linear decision boundary that correctly classify both the positive and negative classes? If so, write down the updation steps and the obtained linear decision boundary. (You may assume the initial decision boundary is $x_2 = 1/2$, and sweep the 4 points in clockwise order, i.e., (P1,P2,P3,P4,P1,P2,...), note that you cannot write down the arbitrary linear boundary without updation steps.)

Ans: Yes it is possible to apply the perceptron algorithm to obtain the linear decision boundary that correctly classify both the positive and negative classes. Decision boundary is given by $w_0 + w_1x_{1+}w_2x_2 = 0$. From the question we are supposed to consider $x_2 = \frac{1}{2}$. That means we can consider our initial weights as $w_0 = -1$, $w_1 = 0$ and $w_2 = 2$.

We want our perceptron algorithm to work for OR function, so truth table of OR is given below.

X_1	X_2	T (Output)
0	1	1
1	1	1
1	0	1
0	0	0

So according to Perceptron algorithm,

For P1, w_0 = -1, w_1 = 0, w_2 = 2, t = 1,

$Y_{in} = w_0 + w_1x_{1+}w_2x_2$

$Y_{in} = -1 - 0x_1 + 2x_2$

$= -1 - 0(0) + 2(1)$

$= 1$

We check, $T^*(Y_{in})$ if it is positive then our predicted output is correct. Here in this case , $T^*(Y_{in})>0$ thus predicted output is correct. So no need to change the weights.

For P2, w_0 = -1, w_1 = 0, w_2 = 2, t = 1,

$Y_{in} = w_0 + w_1x_{1+}w_2x_2$

$Y_{in} = -1 - 0x_1 + 2x_2$

$Y_{in} = -1 - 0(1) + 2(1)$

$Y_{in} = -1 - 0 + 2$

$Y_{in} = 1$

We check, $T^*(Y_{in})$ if it is positive then our predicted output is correct. Here in this case, $T^*(Y_{in})>0$ thus predicted output is correct so no need to change the weights.

For P3, w_0 = -1, w_1 = 0, w_2 = 2, t = 1,

$Y_{in} = w_0 + w_1x_{1+}w_2x_2$

$Y_{in} = -1 - 0x_1 + 2x_2$

$Y_{in} = -1 - 0(1) + 2(0)$

$Y_{in} = -1$

We check, $T^*(Y_{in})$ if it is positive then our predicted output is correct. Here in this case, $T^*(Y_{in})<=0$ thus predicted output is incorrect. So we need to change the weights.

Weight Updating Algorithm,

$W_{0(new)} = W_{0(old)} + T$

$= -1 + 1$

$= 0$

$W_{1(new)} = W_{1(old)} + T(X_1)$

$= 0 + 1(1)$

$= 1$

$W_{2(new)} = W_{2(old)} + T(X_2)$

$= 2 + 1(0)$

= 2

For P4, $w_0 = 0$, $w_1 = 1$, $w_2 = 2$, t = -1,

$Y_{in} = w_0 + w_1x_{1+}w_2x_2$

$Y_{in} = 0 - 0x_1 + 2x_2$

$Y_{in} = 0 - 0(0) + 2(0)$

$Y_{in} = 0$

We check, $T^*(Y_{in})$ if it is positive then our predicted output is correct. Here in this case, $T^*(Y_{in})<=0$ thus predicted output is incorrect. So we need to change the weights.

Weight Updating Algorithm,

$W_{0(new)} = W_{0(old)} + T$

= 0 - 1

= -1

$W_{1(new)} = W_{1(old)} + T(X_1)$

= 1 - 1(0)

= 1

$W_{2(new)} = W_{2(old)} + T(X_2)$

= 2 + -1(0)

= 2

For P1, $w_0 = -1$, $w_1 = 1$, $w_2 = 2$, t = 1,

$Y_{in} = w_0 + w_1x_{1+}w_2x_2$

$Y_{in} = -1 - 1x_1 + 2x_2$

$Y_{in} = -1 - 1(0) + 2(1)$

$Y_{in} = 1$

We check, $T^*(Y_{in})$ if it is positive then our predicted output is correct. Here in this case, $T^*(Y_{in})>0$ thus predicted output is correct. So no need to change the weights.

For P2, $w_0 = -1$, $w_1 = 1$, $w_2 = 2$, t = 1,

$Y_{in} = w_0 + w_1x_{1+}w_2x_2$

$Y_{in} = -1 + 1x_1 + 2x_2$

$Y_{in} = -1 + 1(1) + 2(1)$

$Y_{in} = 2$

We check, $T^*(Y_{in})$ if it is positive then our predicted output is correct. Here in this case, $T^*(Y_{in})>0$ thus predicted output is incorrect. So no need to change the weights.

For P3, w_0 = -1, w_1 = 1, w_2 = 2, t = 1,

$Y_{in} = w_0 + w_1x_{1+}w_2x_2$

$Y_{in} = -1 + 1x_1 + 2x_2$

$Y_{in} = -1 + 1(1) + 2(0)$

$Y_{in} = 0$

We check, $T^*(Y_{in})$ if it is positive then our predicted output is correct. Here in this case, $T^*(Y_{in})<=0$ thus predicted output is incorrect. So we need to change the weights.

Weight Updating Algorithm,

$W_{0(new)} = W_{0(old)} + T$

= -1 + 1

= 0

$W_{1(new)} = W_{1(old)} + T(X_1)$

= 1 + 1(1)

= 2

$W_{2(new)} = W_{2(old)} + T(X_2)$

= 2 + 1(0)

= 2

For P4, w_0 = 0, w_1 = 2, w_2 = 2, t = -1,

$Y_{in} = w_0 + w_1x_{1+}w_2x_2$

$Y_{in} = 0 + 2x_1 + 2x_2$

$Y_{in} = 0 + 2(0) + 2(0)$

$Y_{in} = 0$

Weight Updating Algorithm,

$W_{0(new)} = W_{0(old)} + T$

= 0 - 1

= -1

$W_{1(new)} = W_{1(old)} + T(X_1)$

= 2 - 1(0)

= 2

$W_{2(new)} = W_{2(old)} + T(X_2)$

= 2 - 1(0)

= 2

For P1, w_0 = -1, w_1 = 2, w_2 = 2, t = 1,

$Y_{in} = w_0 + w_1x_{1+}w_2x_2$

$Y_{in} = -1 + 2x_1 + 2x_2$

$Y_{in} = -1 + 2(0) + 2(1)$

$Y_{in} = 1$

We check, $T^*(Y_{in})$ if it is positive then our predicted output is correct. Here in this case, $T^*(Y_{in})>0$ thus predicted output is incorrect. So no need to change the weights.

For P2, w_0 = -1, w_1 = 2, w_2 = 2, t = 1,

$Y_{in} = w_0 + w_1x_{1+}w_2x_2$

$Y_{in} = -1 + 2x_1 + 2x_2$

$Y_{in} = -1 + 2(1) + 2(1)$

$Y_{in} = 3$

We check, $T^*(Y_{in})$ if it is positive then our predicted output is correct. Here in this case, $T^*(Y_{in})>0$ thus predicted output is incorrect. So no need to change the weights.

For P3, w_0 = -1, w_1 = 2, w_2 = 2, t = 1,

$Y_{in} = w_0 + w_1x_{1+}w_2x_2$

$Y_{in} = -1 + 2x_1 + 2x_2$

$Y_{in} = -1 + 2(1) + 2(0)$

$Y_{in} = 2$

We check, $T^*(Y_{in})$ if it is positive then our predicted output is correct. Here in this case, $T^*(Y_{in})>0$ thus predicted output is incorrect. So no need to change the weights.

For P4, w_0 = -1, w_1 = 2, w_2 = 2, t = 1,

$Y_{in} = w_0 + w_1x_{1+}w_2x_2$

$Y_{in} = -1 + 2x_1 + 2x_2$

$Y_{in} = -1 + 2(0) + 2(0)$

$Y_{in} = -1$

We check, $T^*(Y_{in})$ if it is positive then our predicted output is correct. Here in this case, $T^*(Y_{in})>0$ thus predicted output is incorrect. So no need to change the weights.

So here weights W_0, W_1, W_2 correctly classify all 4 inputs so out final decision boundary is

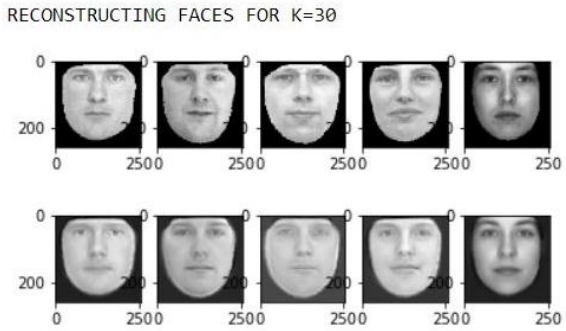
$2X_1+2X_2 = 1$

Problem 2(70pt): [Principal Component Analysis, Dimension Reduction, Eigenface] Face modelling serves as one of the most fundamental problems in modern artificial intelligence and computer vision, and can be useful in various applications like face recognition, identification etc. However, face images are usually of high-dimensional (e.g., a small 100×100 gray-scaled face image has dimension 100×100 = 10,000), therefore, find a suitable representation is utterly important. In this problem, we apply the linear model, principal component analysis (PCA), on face images to reduce the dimension and obtain eigenface representations. Dataset: we use the dataset† which contains 177 face images. Each image contains 256×256 pixels and is gray-scaled (i.e., the value for each pixel is stored as unsigned integer between [0,255], typically, 0 is taken to be black and 255 is taken to be white). You need to split the dataset to be train/test set, e.g., you could use the first 157 images for training, and the rest 20 faces for testing.

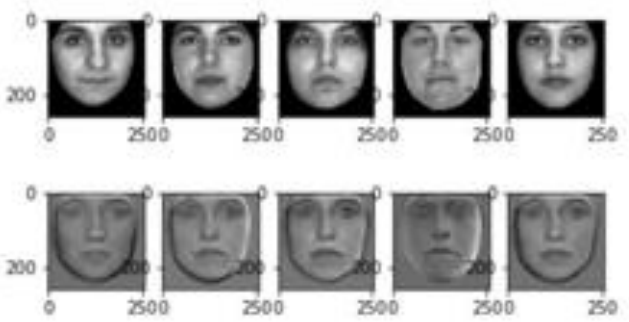
(1) [30pt] Write the PCA codes to compute K = 30 eigenfaces and visualize the top 10 eigenfaces.

Solution: For full Code please refer to attached code file.

```
testing = testing - meanImg
k=30
print("RECONSTRUCTING FACES FOR K=30")
reconstructed30 = reconstructing_faces(k,meanImg,testing,V)

plot_gallery(check, 1, 5)
plot_gallery(reconstructed30, 1, 5)
```



(2) [20pt] Use the learned K eigenfaces from (1) to reconstruct testing images, and record the reconstruction error. The reconstruction error can be defined as ||ˆ Y−Y||2 N , where ˆ Y is the reconstructed face using the learned eigenfaces, Y is the testing faces and N is the total number of testing data. Please show 5 testing images and their reconstructed ones.



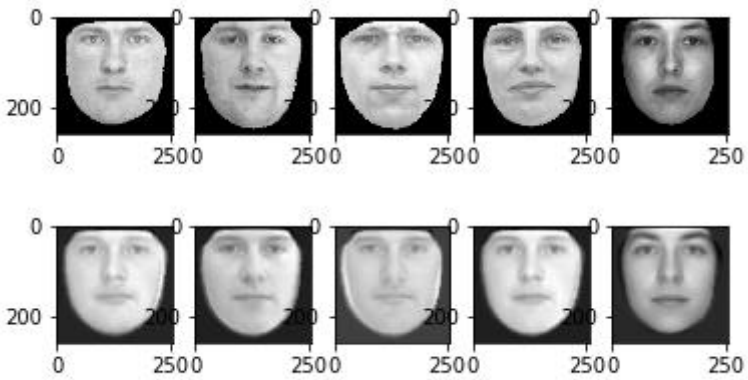
```
#K=30
eig_facesk30 = select_faces(30,sortedval , train,157)
print(eig_facesk30.shape)
eig_facesk10 = select_faces(20,sortedval, train,157)
plot_gallery(eig_facesk10, 2, 5)

#Reconstruction Part
mean_test = np.mean(test, axis = 0)
testing30 = test - mean_temp_test
weight_vector30 = np.dot(eig_facesk30,testing30.T)
final_images30 = np.dot(eig_facesk30.T,weight_vector30)
final_images30 = final_images30.T
for i in range(0,len(final_images30)):
    final_images30[i] += mean_test
    error = (final_images30[i] - test[i])
    reconstruction_error30 = np.linalg.norm(error)/len(train)
    print(reconstruction_error30)
plot_gallery(test[5:10], 1, 5)
plot_gallery(final_images30[5:10], 1, 5)
#showImage(final_images30[3])
#showImage(test[3])
```

Reconstruction Error

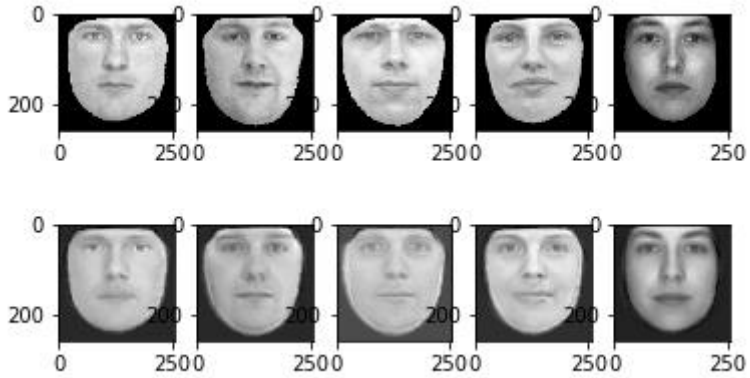
```
Errors for reconstruction in percentage
[24.929919315922724, 29.565734480327546, 31.045727179589356, 30.58402597035898, 24.679070365646716, 25.839396064187568, 28.4
2651634811332, 27.679063788313435, 27.435862328203484, 36.67004378434582, 33.42971385136976, 33.551198454728755, 26.14178313
6067374, 33.447123406193036, 28.567967142087948, 31.528232373633646, 28.97805909524499, 26.822242774957537, 30.5964581998150
14, 28.808222490097947]
```

(3) [20pt] Try different values of K, e.g., try K = 10,30,50,100,150..., and draw the curve to indicate the corresponding testing reconstruction errors. The x-axis of the curve can be different K values, and the y-axis can be testing reconstruction error defined in (2).

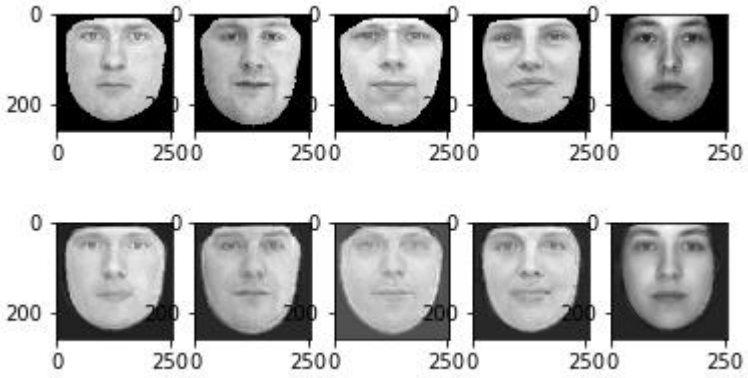
K = 10



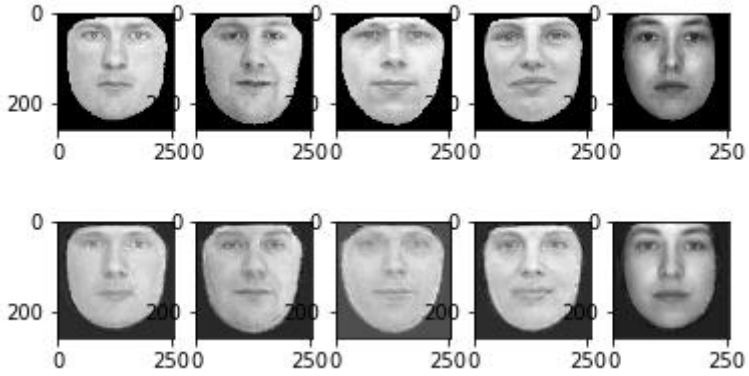
K = 50



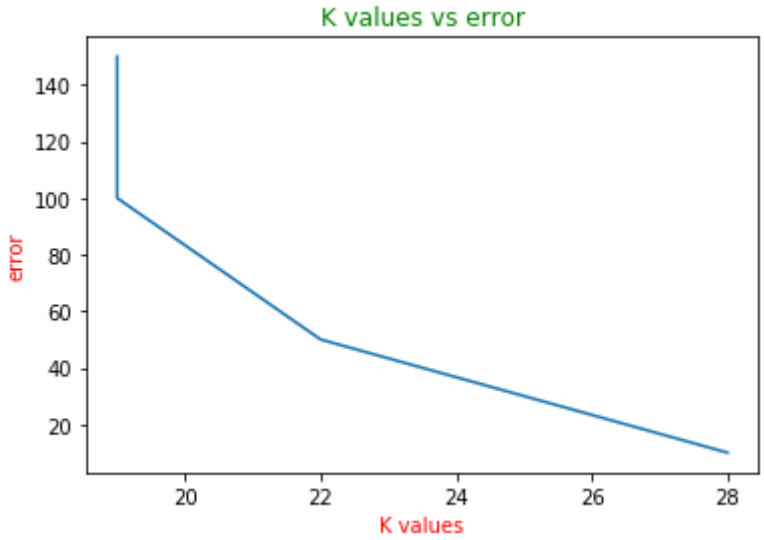
K = 100



K = 150



Plot:


K values vs error
140
120
100
80
60
40
20
error
20
22
24
26
28
K values