ENEE633 Statistical Pattern Recognition Project 1

UID:118339238 Young-Shiuan Hsu

1. Dataset (Training and testing dataset)

Throughout this project, I only use the first dataset which has 3 different pictures (neutral, facial expression and illumination) for each individual.

- For the first four classifiers (Bayes', K-NN, PCA, MDA), I separate the given data into 200 different individuals. Each contains three different pictures. By doing facial recognition, the neutral and facial expression data are the training data and the illumination picture is the testing data. Since the training data is not sufficient to form a good classifier, the expected training error rate would be very high.
- For Kernel SVM and Adaboost classifier, the goal is to identify whether the picture is neutral or with facial expression. I use the first 150 data of neutral faces and facial expression as my training data and the last 50 pictures of each of them as the testing data.

2. Classifier Results and methods

- **Bayes' Classifier**

  To use the first two picture to construct 200 different gaussians. First, we have to find the maximum likelihood (the formula given below) of the given data so that we can treat it as the mean and variance for our gaussian distribution.

$$\mu_n = \frac{1}{N}\sum_{x_i \epsilon w_n} x_i \quad \text{Cov(X)} = (x_n - \mu_n)(x_n - \mu_n)^T$$

  The data is passed into the function (ML_estimation.m) to calculate the mean (504*1 matrix) and variance (504 * 504 matrix) with the formula given in the lecture notes. Then we can test the given testing dataset and that getting the result of error rate equals to 0.365 which is better than what I expected since the training set is not sufficient to train a good model.

- **K-NN Classifier**

  The goal for K-NN classifier, we have to try out different Ks to find the minimum testing error rate. First, I calculate the Euclidian distance (Eucli_dis.m) for each point to all the other points. Then I find the K nearest neighbors for the testing data and poll among those chosen values. By fitting in different K value, I found out that when K=1 has the best performance. It is logical that K=1 is a better choice in my scenario because there are only two training data for every individual in my training set.

K values	K=1	K=2	K=3	K=5	K=10	K=20
Error rate	0.405	0.535	0.545	0.54	0.64	0.77

- **PCA**

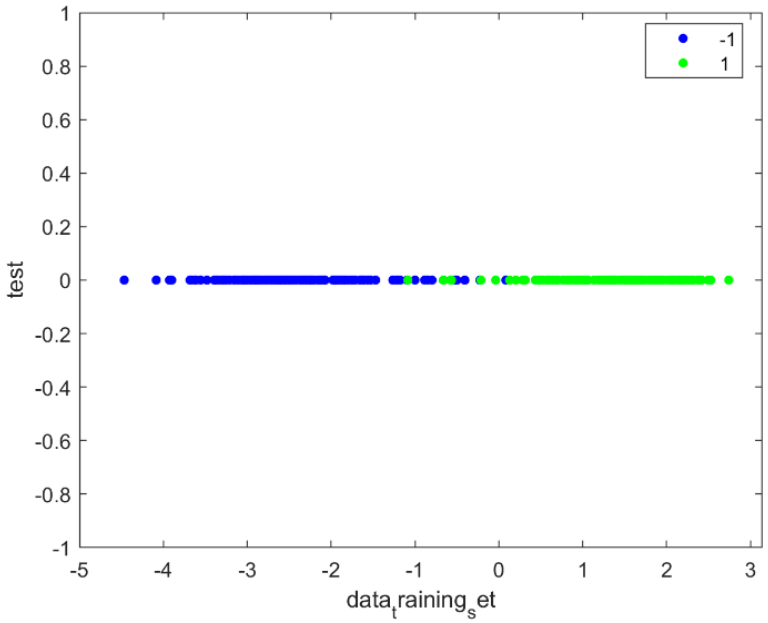
  The goal for PCA is to reduce the dimensionality to avoid curse of dimensionality. The main difference between PCA and MDA is that we find the data that best fits the data and all the calculated eigen values are orthogonal. In my experiment, I projected my original data (504 dimensions) onto approximately 300 dimensions by setting a threshold for the extracted eigenvalues. Then by passing it into ML estimation and Bayes' classifier with the lower dimension data, I am able to obtain an error rate of 0.76. The error rate is extremely high since the training data is not enough and we reduce the dimensionality which indicates that there is information loss. However, it is still better than guessing the person if given an image (the probability of guessing it wrong is 199/200 = 0.995).

- **MDA**

  The goal for MDA is also to reduce the dimensionality of the given data to avoid curse of dimensionality. The difference between PCD is that we find the project vectors that perfectly separates the training data set and the vectors are not necessarily orthogonal. The dimension of the projected data should be less or equal to the number of classes-1. Since we desire to classify the data into 200 classes, therefore we reduce the data dimension to 199. After projecting the training data onto the selected eigen vectors, we pass it into the ML estimation function and Bayes' classifier (same step as PCA). We can obtain an error rate of 0.885. The error rate is even higher than PCA because we reduce the dimensionality greater that PCA which means we loss more information from the data. However, it is still better than purely guessing the person if we are given an image.

- **Kernel SVM**

  By setting the label of the dataset y to +1 and -1 it is obvious that Kernel SVM can only classify a dataset into 2 different cases. As a result, I changed my training dataset and testing data set to distinguish whether the given image is neutral or if it has a facial expression instead of working on face recognition. By implementing MDA and reduce the data's dimension to 1 (the theory of MDA is to reduce the dimension less than the number of classes). To verify the projected data is logical, the plot below is the output after implementing MDA. It is obvious that the vector is able to separates the two datasets.



After checking the projected data, we pass the data into the specified kernel as below to get higher dimension data.

$$\text{RBF: K(x, y)} = \exp\left(\frac{-\|x-y\|^2}{\sigma^2}\right) \quad Polynomial: K(x,y) = (x^T y + 1)^r$$

The constructed kernel (Kernel_construction.m) is a matrix shown below

$$\text{K} = \begin{bmatrix} K(x_1,x_1) & K(x_1,x_2) & \dots & K(x_1,x_{N-1}) & K(x_1,x_N) \\ \vdots & & \ddots & \vdots & \\ K(x_N,x_1) & K(x_N,x_2) & \dots & K(x_N,x_{N-1}) & K(x_N,x_N) \end{bmatrix}$$

After obtaining the K matrix, we can find the optimal μ vector from the dual problem in SVMs by using the function quadprog(). We will also have to tune the value of σ^2, r, C where C is the value of upper bound of the optimal μ. The results are below:

RBF Kernels error rate:

	$\sigma^2 = 1$	$\sigma^2 = 0.1$	$\sigma^2 = 0.001$	$\sigma^2 = 0.000001$
C = 1	9	9	7	0
C = 2	9	9	7	0
C = 7	10	9	7	0

Polynomial Kernel error rate:

	$r = 1$	$r = 2$	$r = 10$	$r = 20$
C = 1	10	10	14	11
C = 2	10	10	14	11
C = 7	10	10	14	10

- **Adaboost (Linear SVM)**

  Linear SVMs required in this problem is only to substitute the kernel in Kernel SVM with the inner product of two column vector. The tricky part is the second step of the algorithm where we have to find $\theta_i \; s.t. \sum_{n=1}^{N} P_n \; (if \; y_n \neq \varphi(x_n, \; \theta_i)) \leq \frac{1}{2}$. To find θ_i we want, I use the function randsample() which selects the sample points using the weights that are calculated(the P matrix which indicates that the weight of each training data supports the vector). If we select $2/3$ of the data that has higher weights and pass it into the linear kernel that classifies those data samples correctly is equivalent to find θ_i that has the ability to classify data with error rate lower than $1/2$. By doing so, we are able to obtain μ for every iteration and that we have to store every value into a matrix to construct the decision function. In my code, I ran the iteration for 200 times to obtain the piecewise linear decision function and getting an error rate of 10 which is similar to what I expected. After we construct the decision function, we only have to pass the testing dataset into it and calculate and take the sign of it to determine which class it belongs to. In my result, I am able to obtain an error rate of 0.1 which is surprisingly lower than I expected. As we reduce the number of iterations will only increase the error rate.