Pattern Recognition

Mini-Project #3

Instructor: Dr. Yazdi

Student: Hamze Ghaedi

Student-No: 9831419

Abstract

Usually, in real world problems, we are facing with datasets of thousands sample points each having hundreds of features.

In such situations, taking all features into account, leads to extraordinary computational burden. But It is often the case that all features are not equally important, so eliminating some less important features May cause better computational complexity as well as simpler classification scheme

In this project, we are implementing two techniques (so-called **PCA** and **FLDA**) for feature selection (or dimensionality reduction) and comparing their aspects.

Problem #1

Dataset Preparation

Each sample in our dataset is a vector of 784 elements with values up to 255

At the first step, the samples are normalized to [0,1] (by dividing by 255).

Then the samples are z-scored based on below relation

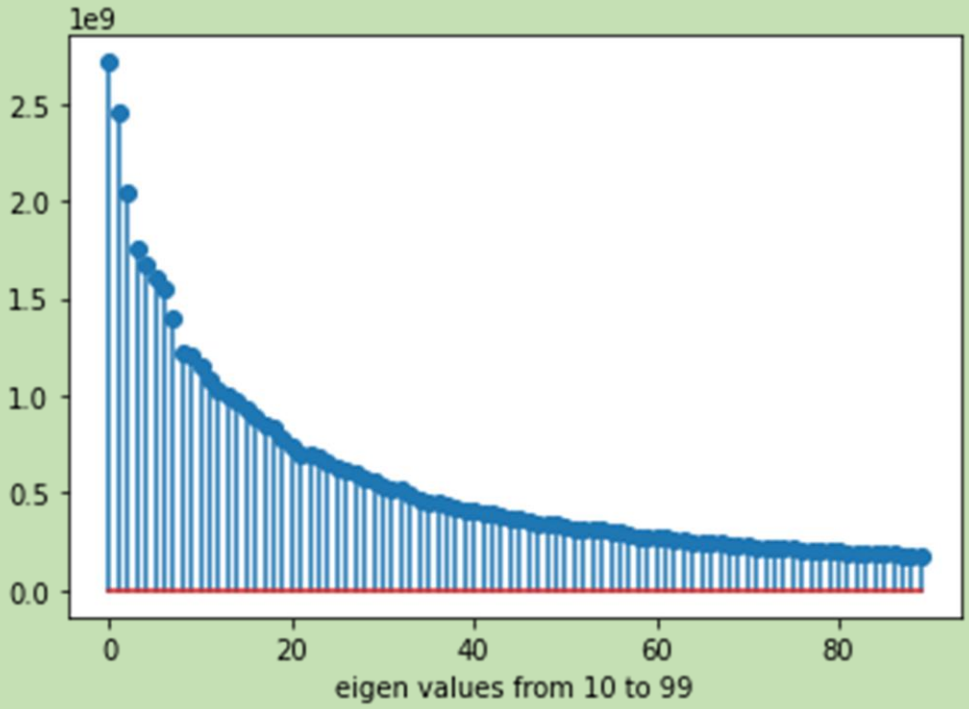
$$x_{zscore} = \frac{x - \mu}{\sigma}$$

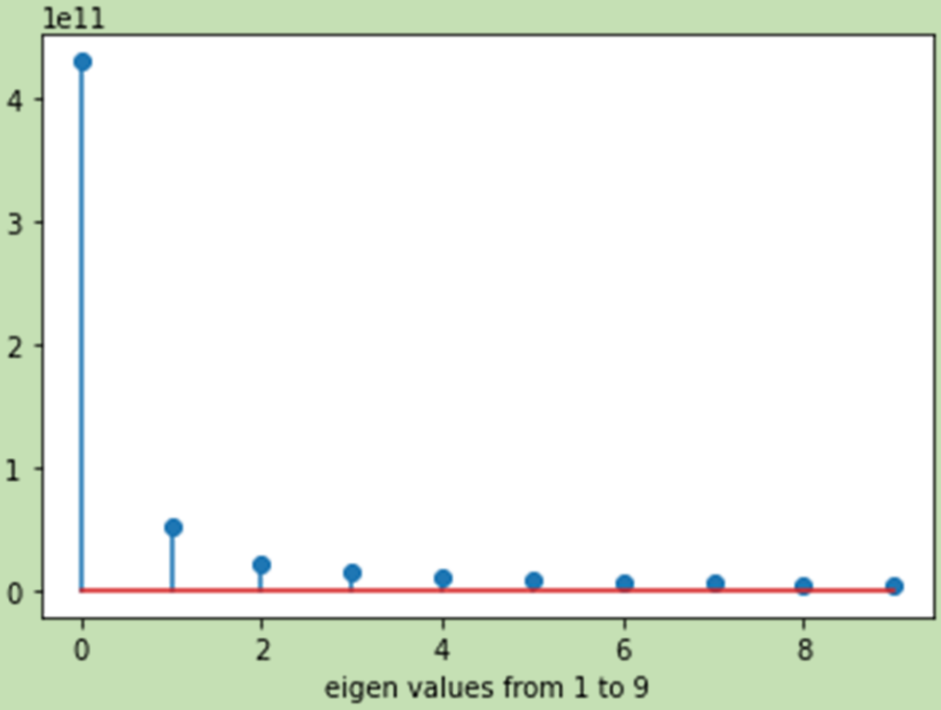
$$\mu: total\ mean\ of\ samples$$

$$\sigma: total\ variance\ of\ samples$$

#1. a

The figure below, illustrates Principal Components of the Scatter Matrix, sorted descending and normalized by the largest component.





#1. b (PCA)

According to part a, only 68 largest components of the scatter matrix are selected and their corresponding eigen vectors are putted together to make a transformation matrix W.

Then, the train set (and test set) are transformed based on relations listed below

$$X_{trans} = W^T X$$

Note that, the columns of W are selected eigen vectors and X is the matrix of the normalized and z-scored version of the primitive dataset (each column is a sample)

Finally, X_{trans} is the matrix of the transformed samples (each column is transformed sample corresponding to sample located at same column in X)

actually, a transformed sample is a liner combination of selected eigen vectors

and the coefficients of the combination are elements of corresponding column in X_{trans} ,so for a transformed sample $\tilde{x_j}$ we have

$$\tilde{x_j} = \Sigma_{i=1}^{p} \tilde{X}(i,j) * W(:,i)$$

Where p is the number of selected eigen vectors.

If d be the number of all eigen vectors, usually (p < d) so dimensionality reduction takes place here.

We define the transform error as below:

$$J(W) = \Sigma_{k=1}^{N} (x - \tilde{x})^2$$

$$\boldsymbol{J(W) \approx 14982864.641702838}$$

#1. b (Bayesian Classifier)

Our dataset consists of 60,000 samples each having 784 features, since the computational complexity needed for fitting a Bayesian classifier is in the order of O(Nd^2) so,

Without feature selection

$$O(Nd^2) \approx 36{,}879{,}360{,}000$$

With PCA feature selection (selected top 68 features)

$$O(Nd^2) \approx 277{,}440{,}000$$

Fitting a Bayesian classifier to our transformed version of the dataset follows the procedures discussed in Mini-Project #2 (and listed below), for the sake of conciseness, the discussions are omitted.

1) First computing priori, mean and Covariance matrix for each class
2) Covariance matrices are singular. Maximum entropy estimation is addressed this problem
3) According to handbook, a set of discriminator functions are defined
4) Classification and evaluation are conducted based on the the discriminator functions

following this approach, the overall accuracy of 84% is achieved

$$acc = 84.2\%$$

Problem #2

FLD

Same as PCA, we seek a linear transformation such that:

$$\tilde{X} = W^T X$$

$$J(W) = \frac{W^T S_B W}{W^T S_W W} \rightarrow max$$

Where as X~ is matrix of samples with reduced dimension and

- **Total mean vector:**

$$m = \frac{1}{N}\sum_{k=1}^{c} X_k = \frac{1}{N}\sum_{k=1}^{c} n_k m_k$$

- **Total scatter matrix:**

$$S_T = \sum_{k=1}^{N} (X_k - m)(X_k - m)^T$$

- **Within-class scatter matrix:**

$$S_W = \sum_{k=1}^{c} S_k = \sum_{k=1}^{c} \sum_{X \in D_k} (X - m_k)(X - m_k)^T$$

- **Between-class scatter matrix:**

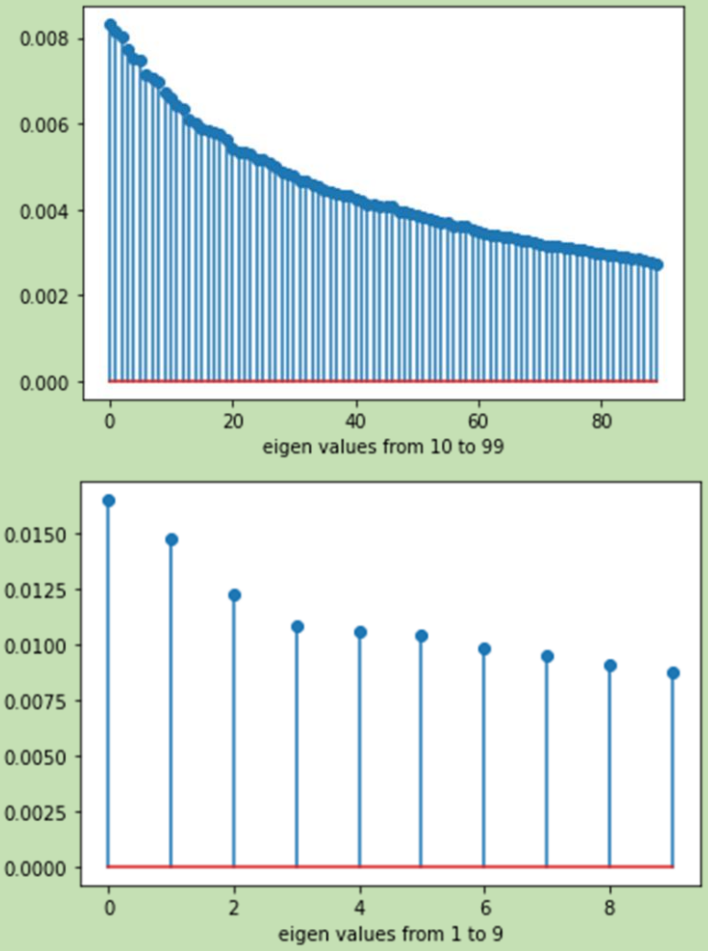
$$S_B = \sum_{k=1}^{c} (m_k - m)(m_k - m)^T$$

- **FLD Criterion function:**

$$J(w) = \frac{w^T S_B w}{w^T S_W w}$$

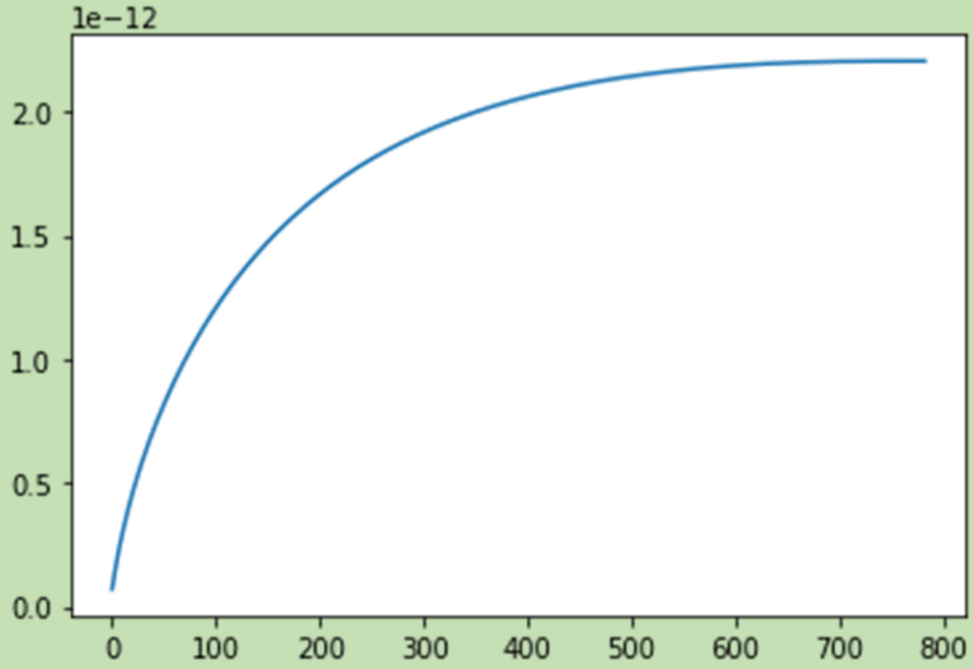
#2. a

The figure below, illustrates Principal Components of the Scatter Matrix, sorted descending and normalized by the largest component.



#2. b

The figure below, illustrates the separability measure vs number of components (normalized by largest component excluding the largest value)



#2. c (Bayesian classifier)

After transforming samples using FLD method and following same procedures as PROBLEM #1.b :

the overall accuracy of 46% is achieved

$$acc = 46\%$$

Results

considering the eigen values plot for both of two methods (PCA and FLD) we find that the eigen values for scatter matrix in PCA is more compact than eigen values of separability matrix in the case of FLD.

In the other words, the PCA method produces less important eigen values than the FLD method or it can be said that the PCA transformation distributed the energy of a signal more compactly than FLD (energy compaction)

In this project, I reached the overall accuracy of 84% by using PCA method for feature selection and about a half of that (46%) using the FLD method whereas the number of selected features is nearly equal (68 for PCA and 70 for FLD) so, approximately both methods have same computational complexity.

PCA :

$$acc = 84\%$$

FLD :

$$acc = 46\%$$

Appendix

The scripts are written in python and the following libraries are used:

1) numpy for matrix manipulation

2) Pandas for reading CSV

3) matplotlib for plotting

Function for PCA feature selection

```
def pca_feature_selection(x_train,x_test,n_selected_f):

  n_features = x_train.shape[1]
  n_samples = x_train.shape[0]
  #---------------COMPUTING SCATTER MATRIX---------------------
  scatter_matrix = np.zeros((n_features,n_features));
  for i in range(0,n_samples):
    temp = np.matmul(x_train[i,:].reshape(n_features,1),x_train[i,:].reshape(1,
n_features))
    scatter_matrix += temp
  #------------------------------------------------------------

  #finding eigen values and eigen vectors
  W,V = np.linalg.eig(scatter_matrix)

  #reaarange indices of elements of W based on values of the elements ascending
ly!
  indices = np.argsort(W)
  #reverse the elements of indices (we need them in descending order)
  indices = indices[::-1]

  top = V[:,indices[0:n_selected_f]] #select top n_selected_f vectors of V

  #apply transformation on x_train and x_test
  x_train_trans = np.matmul(x_train,top)
  x_test_trans = np.matmul(x_test,top)
  #----------------------------------------

  return W,top,x_train_trans,x_test_trans
```

Function for FLD feature selection

```
def fld_feature_selection(x_train,y_train,x_test,n_class,n_sf):
  n_features = x_train.shape[1]
  n_samples = x_train.shape[0]

  #---------------COMPUTING SCATTER MATRIX---------------------
  scatter_matrix = np.zeros((n_features,n_features));
  for i in range(0,n_samples):
    temp = np.matmul(x_train[i,:].reshape(n_features,1),x_train[i,:].reshape(1,n_features))
    scatter_matrix += temp
  #------------------------------------------------------------

  freqs,_,means,covs = compute_stats(x_train,y_train,10)

  #placeholders-------------------------------
  tot_mean = np.zeros([1,n_features])
  wic_scat_mat = np.zeros([n_features,n_features]) #within class scatter matrix
  bc_scat_mat = np.zeros([n_features,n_features]) #between class scatter matrix
  #-------------------------------------------

  #compute total mean and within class scatter matix
  for i in range(n_class):
    tot_mean += (freqs[i,0] * means[i,:]).reshape((1,n_features))
    wic_scat_mat += (freqs[i,0] * covs[:,:,i]).reshape((n_features,n_features))

  tot_mean /= n_samples
  #-------------------------------------------

  #compute between class scatter matrix----------
  for i in range(n_class):
    bc_scat_mat += np.matmul((means[i,:]-tot_mean).reshape((n_features,1)),(means[i,:]-tot_mean))
  #-------------------------------------------

  #apply FLD-------------------------------------
  wic_scat_mat = wic_scat_mat.astype('double')
  s_w_inv = np.linalg.inv(wic_scat_mat)
  s_ii = s_w_inv * bc_scat_mat
  W,V = np.linalg.eig(s_ii)
  indices = np.argsort(W)
  indices = indices[::-1] #reverse order
  top = V[:,indices[0:n_sf]]
  x_train_trans = np.matmul(x_train,top)
  x_test_trans = np.matmul(x_test,top)
  #-----------------------------------------------
  return W,top,x_train_trans,x_test_trans
```

Computes Frequency of occurrence, Priories, Means and covariances of classes according to given dataset

```
def compute_stats(x_train,y_train,n_class):

  n_samples = x_train.shape[0]
  n_features = x_train.shape[1]

  #define placeholders----------------
  priories = np.zeros([n_class,1])
  freqs = np.zeros([n_class,1])
  means = np.zeros([n_class,n_features])
  covs = np.zeros([n_features,n_features,n_class]);
  #-----------------------------------

  #computing frequencies,means and priories
  for i in range(n_samples):
    c = y_train[i][0]
    freqs[c]+=1
    means[c,:] += x_train[i,:]
  priories = freqs / n_samples;

  for i in range(n_class):
    means[i,:] /= freqs[i]
  #-------------------------------------

  #computing covariances---------------
  for i in range(n_samples):
    c = y_train[i,0]
    x_i = x_train[i,:]
    covs[:,:,c] += np.matmul((x_i - means[c,:]).reshape((n_features,1)), (x_i -
 means[c,:]).reshape((1,n_features)))

  for i in range(10):
    covs[:,:,i] /= (freqs[i]-1)
  #-------------------------------------

  return freqs,priories,means,covs
```

disc_params() function Calculates parameters of discriminant function and classify() function assigns a given sample to a the most probable class.

```
def disc_params(priories,means,covs):
  n_selected_f = covs.shape[0]

  #define placeholders
  W_is = np.zeros([n_selected_f,n_selected_f,10])
  w_is = np.zeros([10,n_selected_f])
  w_i0 = np.zeros([10,1])
  #-----------------------------------------

  cov_inv = np.zeros([n_selected_f,n_selected_f,10])
  covs = covs.astype('double')

  for i in range(10):
    cov_inv[:,:,i] = np.linalg.inv(covs[:,:,i])


  W_is = (-1/2)*cov_inv
  cov_inv = cov_inv.astype('double')
  for i in range(10):
    w_is[i,:] = np.matmul(cov_inv[:,:,i],(means[i,:]).reshape((n_selected_f,1))).reshape
((n_selected_f))
    w_i0[i,0] = (-
1/2)*np.dot((means[i,:]).reshape((1,n_selected_f)),(w_is[i,:]).reshape((n_selected_f,1))
)[0,0] -(1/2)*np.log(np.linalg.det(covs[:,:,i]))  + np.log(priories[i,0])

  return W_is,w_is,w_i0



def classify(x,W_i,w_i,w_i0):
  disc = np.zeros([10,1])
  n_selected_f = W_i.shape[0]
  for i in range(10):
    temp = np.matmul(x.reshape((1,n_selected_f)),W_i[:,:,i].reshape((n_selected
_f,n_selected_f)))
    temp = np.matmul(temp.reshape((1,n_selected_f)),x.reshape((n_selected_f,1))
)
    disc[i,0] =  temp + np.dot(w_i[i,:],x) + w_i0[i,0]
  return np.argmax(disc)
```

Computes accuracy according to a set of predictions

```
def evaluation(x_test,y_test,W_i,w_i,w_i0):

  n_selected_f = x_test.shape[1]
  preds = np.zeros([len(x_test),1])
  for i in range(len(x_test)):
    preds[i,0] = classify(x_test[i,:].reshape((n_selected_f,1)),W_i,w_i,w_i0)
  #compute acc
  c = 0
  for i in range(len(x_test)):
    if(y_test[i,0] == preds[i,0]):
      c+=1
  acc = c/len(x_test)
  return acc
```

Problem #1

```
x_train,y_train,x_test,y_test = load_dataset()
prepare_dataset(x_train)
prepare_dataset(x_test)


#-----------------------PROBLEM #1-----------------------------
#PROBLEM #1.PART b---------------------------------
n_selected_features_pca = 68
pca_W,pca_top,pca_x_train_trans,pca_x_test_trans = pca_feature_selection(x_train,x_test,n_selected_features_pca)
_,pca_priories,pca_means,pca_covs = compute_stats(pca_x_train_trans,y_train,10)
pca_mecs = max_ent_cov_estimator(pca_covs)
pca_W_i,pca_w_i,pca_w_i0 = disc_params(pca_priories,pca_means,pca_mecs)
pca_acc = evaluation(pca_x_test_trans,y_test,pca_W_i,pca_w_i,pca_w_i0)
print('\nTHE OVERALL ACCURACY IS:')
print (pca_acc)
print('-----------SQUARE ERROR--------------')
#PCA SQUARE ERROR
pca_err = 0
for i in range(x_train.shape[0]):
  x_tilde = np.matmul(pca_x_train_trans[i,:],pca_top.reshape(n_selected_features_pca,784))
  pca_err += np.linalg.norm((x_tilde - x_train[i,:]))**2
print(pca_err / 60000)
#PROBLEM #1.PART a---------------------------------
w_sorted = np.sort(pca_W)
w_sorted = w_sorted[::-1]
w_norm = w_sorted / w_sorted[0]


plt.stem(w_sorted[0:10])
plt.xlabel('eigen values from 1 to 9')
plt.figure()
plt.stem(w_sorted[10:100])
plt.xlabel('eigen values from 10 to 99')
#-------------------------------------------------
```

Problem 2

```
#2.c-----------------------------------

n_selected_features_fld = 300
fld_W,fld_top,fld_x_train_trans,fld_x_test_trans = fld_feature_selection(x_train,y_train
,x_test,10,n_selected_features_fld)
_,fld_priories,fld_means,fld_covs = compute_stats(fld_x_train_trans,y_train,10)
fld_mecs = max_ent_cov_estimator(fld_covs)
fld_W_i,fld_w_i,fld_w_i0 = disc_params(fld_priories,fld_means,fld_mecs)
fld_acc = evaluation(fld_x_test_trans,y_test,fld_W_i,fld_w_i,fld_w_i0)

print('\nTHE OVERALL ACCURACY IS:')
print (fld_acc)
#----------------------------------------

#2.a-------------------------------------
fld_w_sorted = np.sort(fld_W)
fld_w_sorted = fld_w_sorted[::-1]
fld_w_norm = fld_w_sorted / fld_w_sorted[0]

plt.stem(fld_w_sorted[0:10])
plt.xlabel('eigen values from 1 to 9')
plt.figure()
plt.stem(fld_w_sorted[10:100])
plt.xlabel('eigen values from 10 to 99')
#----------------------------------------

#2.b-------------------------------------
comps = fld_w_norm
for i in range(1,784):
  comps[i] += comps[i-1]

plt.figure()
plt.plot(comps[1:])
#----------------------------------------
```