DSP Assignment 2

March 28, 2018

Introduction

Classification of data is a main problem in machine learning, it is used in day to day problems, it can be very helpful and save time and money, in our problem here it is required to classify the handwritten numbers into its class of number between 0-9.

Clustering:

Clustering is an important tool in machine learning and powerful for getting useful information from data, in this assignment different clustering algorithms were used as: K-Means, GMM or SVM.

Features reduction:

Large size of features can be useful for learning algorithm however it is heavily affected by computational power of the machine so a very helpful tools can be used to reduce amount of input features while keeping most of variance and information in data using DCT or PCA.

Outline

Combining this two powerful concept can led to satisfying results for classification problem and this will be shown in the next handwritten numbers classification problem:

```
In [1]: #Initializing needed libaries import
        scipy as sp
        import numpy as np import pandas
        as pd import matplotlib as plt
        from scipy.fftpack import dct as dct from
        scipy import io as spio
        from scipy import ndimage as img from
        random import randint
        from sklearn.decomposition import PCA from
        sklearn.cluster import KMeans
        from sklearn.mixture import GaussianMixture from
        sklearn.metrics import accuracy_score from
        sklearn.preprocessing import normalize
        from sklearn.preprocessing import StandardScaler from
        sklearn import svm
        import time
        from sklearn.metrics import confusion_matrix from
        scipy import linalg as la
        from numpy.linalg import inv
        %matplotlib inline
```

Core Functions:

-Extract Data

This function handles the input data, first it reads the input features and labels, then it reshapes features into an image pixels format while labels transform it into class number instead of hot one.

-Standardize

To make data in standard form with mean=0 and variance of 1, this function takes the input data and standardize them.
The standardization process decreases the values of input data making information dense in smaller values while keeping all information, due to smaller values computational process be-comes faster which is in favor of the algorithm.

-Unroll

Unrolling of input into a single vector for other functions.

-dct_2D

DCT is a powerful transformation for features which decreases number of features dramatically while keeping most of information and variations in data.

first 2D DCT transformation is applied on the data then Zigzag reading of DCT coefficient to make the most of the transformation.

-pca_fit & pca_trans

PCA is another powerful transformation where it reduces input features into smaller number of feature, it reduces the number of dimensions of the input data while keeping high variations of the data.

pca_fit forms the model on the training data, while pca_trans transforms the test features into the same model of the training data.

-KMeans

K-Means is one of the most known clustering algorithms, in order to use it with our classification problem it was applied on each class training data in order to produce the means of the each class data, then at test time nearest class is assigned to the input vector with the nearest distance from cluster of the assigned class.

-GMM

Gaussian Mixture Model is another important algorithm of clustering, it takes longer time for training however it helps a lot in increasing accuracy when compared to kmeans using same number of clusters

-predict_acc

In this function it estimates which class should be assigned to the test example, and then calculates the accuracy of the predictions according to the true labels.

-accuracy

In this function it calculates the correctly predicted classes ratio to the whole test set and returns the accuracy.

-find_class

This function estimates the class that should be assigned to the currrebt test vector of features, this function is used by predict_acc

```
In [2]: #Functions defination
        #extract data from Mat format and normalize the image def
        extract(Data):
            features=[]
            output=[]
            for i in range (Data.shape[0]):
                    features.append(Data[i,0])
                    output.append(Data[i,1])
            features=np.array(features)
            output=np.array(output)
            features=np.reshape(features,[features.shape[0]*features.shape[1],28,28])/1.0
            output=np.reshape(output,[output.shape[0]*output.shape[1],output.shape[2]])
            output=[np.where(r==1)[0][0] for r in output]
            return features, np.array(output)
```

```
  def standarize(x):
      stnd=[]
      for i in range(x.shape[0]): scaler =
          StandardScaler() scaler.fit(x[i])
          temp=scaler.transform(x[i])
          stnd.append(temp)
      return np.array(stnd) def
  unroll(a):
      a=a.reshape(a.shape[0],a.shape[1]*a.shape[2]) return
      a
#2D dct
 def dct_2D(x):
      a=[]
      for i in range (x.shape[0]):
       x_dct=dct(dct(x[i],norm='ortho').T,norm='ortho').T;
        a.append([x_dct[0,0], x_dct[0,1], x_dct[1,0], x_dct[2,0], x_dct[1,1],\
                x_dct[0,2],  x_dct[0,3],  x_dct[1,2],  x_dct[2,1],  x_dct[3,0],\
                  x_dct[4,0],  x_dct[3,1],  x_dct[2,2],  x_dct[1,3],  x_dct[0,4],\
                  x_dct[0,5], x_dct[1,4], x_dct[2,3], x_dct[3,2], x_dct[4,1]])
return np.array(a) #PCA
def pca_fit(x,n):
    pca = PCA(n_components=n)
    pca.fit(x)
    pca_comp=pca.transform(x)
    var=sum(pca.explained_variance_ratio_)*100 return
    pca_comp,var,pca
def pca_trans(x,pca):
    #x = StandardScaler().fit_transform(x)
    pca_comp=pca.transform(x)
    var=sum(pca.explained_variance_ratio_)*100.0
    return pca_comp,var
#K-Means
def kmeans(clusters,classes_n,Features_Train,class_margin): kmeans_=[]
    for i in range (classes_n):
        kmeans_temp=KMeans(n_clusters=clusters,n_init=10,max_iter=5000,algorithm= 'full',
        fit(Features_Train[i*class_margin:i*class_margin+class_margin-1])
        kmeans_.append(kmeans_temp.cluster_centers_)
    kmeans_=np.array(kmeans_)
    return kmeans_



#GMM
def GMM(Mixtures,classes_n,Features_Train,class_margin): G=[]
    for i in range (classes_n):
        G_temp=GaussianMixture(n_components=Mixtures,n_init=10,max_iter=5000,covariance_ty
        fit(Features_Train[i*class_margin:i*class_margin+class_margin-1])
        G.append(G_temp.means_)
    G=np.array(G)
    return G
```

```
#Predict
def predict_acc(test_features,label_set,model):
    Y_predict=np.zeros_like(label_set)
    for i in range (Y_predict.shape[0]):
        Y_predict[i]=find_class(test_features[i],model)
    acc1=accuracy(label_set,Y_predict)
    return acc1,Y_predict
#accuracy calc
def accuracy(original,predicted):
    acc=original-predicted
      acc[acc != 0] = 1
      acc=(np.count_nonzero(acc == 0)*1.0/original.shape[0])*100.0 return
      acc
  #class decision
  def find_class(x,y):
      min_d=np.ones(y.shape[0])*100000000.0 for i
      in range(y.shape[0]):
          for j in range(y.shape[1]):
              temp=np.linalg.norm(x-y[i][j]) if
              temp<min_d[i]:
                  min_d[i]=temp
      min_class_idx=np.argmin(min_d) return
      min_class_idx
```

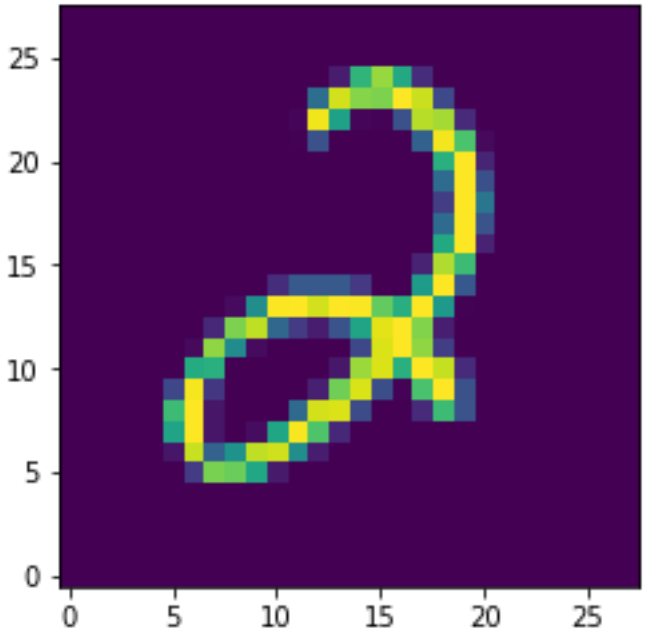
Step 1:

- Read Data
- Extract the features and labels •

Standardize the features

- Unroll the Features
- Show an example of handwritten image

```
In [3]: #Read Data
        R_MNIST=spio.loadmat('./ReducedMNIST.mat')
        R_MINST_Train=R_MNIST['SmallTrainData']
        R_MINST_Test=R_MNIST['SmallTestData'] #extract
        features and labels X_Train,Y_Train=
        extract(R_MINST_Train)
        X_Train_std=standarize(X_Train) X_Test,Y_Test=
        extract(R_MINST_Test) X_Test_std=standarize(X_Test)
        #unroll images
        X_Train_unroll=unroll(X_Train_std)
        X_Test_unroll=unroll(X_Test_std) #show a
        random picture example
        img_num=randint(0,X_Train.shape[0])
        plt.pyplot.imshow(img.rotate(X_Train[img_num],90),origin= 'lower')
        plt.pyplot.show()
        print("Label = "+ str(Y_Train[img_num])+" image = "+ str(img_num))
```




```
Label = 2 image = 2938
```

Step 2:

- Transform input features using DCT

```
In [4]: #dct for features
        X_Train_DCT=dct_2D(X_Train_std)
        X_Test_DCT=dct_2D(X_Test_std)
```

Step 3:

- Transform input features using PCA

```
In [5]: n_comp=110
        X_Train_pca,X_Train_var,pca=pca_fit(X_Train_unroll,n_comp)
        X_Test_pca,X_Test_var=pca_trans(X_Test_unroll,pca)
        print("Variance= "+ "%.2f" %X_Train_var+", PCA #Components = "+ str(n_comp))
```

```
Variance= 90.68, PCA #Components = 110
```

After that, PCA analysis is used to DIAGONALIZE the covariance of the new feature using the formula (S=Uˆ-1 * sigma* U) sigma: covariance matrix of the PCA output U is the matrix of Eigenvectors Sis a diagonal matrix containing the Eigenvalues

```
In [6]: def covariance_diagonalization (x):
            covariance_matrix = (1/x.shape[0]) *
            np.dot(np.transpose(x),x) e_vals, e_vecs =
            la.eig(covariance_matrix)
            diagonal_eignvalues = inv(e_vecs) * covariance_matrix *
            e_vecs return diagonal_eignvalues
        covariance_diagonalization = covariance_diagonalization
        (X_Train_pca) print(covariance_diagonalization)
```

```
[[ 3.85417350e+01 -1.87597666e-45 -4.04078650e-46 ... -4.77611185e-
  36 -1.10178554e-34  8.45323962e-36]
 [-1.00053527e-45  2.79260414e+01  3.06005379e-42 ... -2.78482658e-
  33 1.17642286e-34 -2.40026050e-34]
 [-2.58722898e-46  3.98765521e-42  2.59482388e+01 ... -4.52985837e-
  33 8.30423948e-33 -1.67113382e-33]
 ...
 [-4.77611185e-36 -2.78481084e-33 -4.52971707e-33 ...  1.34207393e-
  06 4.37275696e-10  1.43614914e-10]
 [-1.10178554e-34  1.17648019e-34  8.30472419e-33 ...  4.37275696e-
  10 5.25614607e-08 -3.17300807e-10]
 [ 8.45323962e-36 -2.40046086e-34 -1.67113038e-33 ...  1.43614914e-
  10 -3.17300807e-10  2.60308646e-07]]
```

- The highest values are across the diagonal of the matrix.

K-Means Clustering on DCT Step 4.1:

- perform KMeans clustering with 1 cluster for each class

```
In [7]: #dct kmeans 1
        cluster tic =
        time.time()
        kmeans_dct1=kmeans(1,10,X_Train_DCT,1000
        ) toc = time.time()
        print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 0.1575 sec
```

- Predict KMeans clustering with 1 cluster on test Set

```
In [8]: #dct kmeans 1 cluster
        Prediction tic = time.time()
        acc_kmeans_dct1,Y_dct_KMeans1=predict_acc(X_Test_DCT,Y_Test,kmeans_dct1
        ) toc = time.time()
        print("accuracy
        =",round(acc_kmeans_dct1,2),"%") print("elapsed
        time               =",round(toc-tic,4),"sec\n")
        print("Confusion Matrix:")
        confusion_dct_KMeans1 = pd.crosstab(Y_Test, Y_dct_KMeans1,rownames=[ 'Actual'],
        colnames=[ display(confusion_dct_KMeans1)
```

accuracy = 59.0 %
elapsed time = 0.1835 sec

Confusion Matrix:

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	81	1	6	0	0	6	2	0	3	1	100
1	0	79	10	0	0	0	1	0	3	7	100
2	11	6	52	6	0	2	9	5	5	4	100
3	2	4	8	64	1	2	0	7	2	10	100
4	3	6	0	1	54	1	1	0	13	21	100
5	7	3	4	9	8	32	4	0	20	13	100
6	6	3	3	0	7	3	68	0	10	0	100
7	2	7	8	0	3	0	0	61	5	14	100
8	2	10	2	1	2	2	8	0	48	25	100
9	3	12	1	1	14	3	1	1	13	51	100
All	117	131	94	82	89	51	94	74	122	146	1000

Step 4.2:

- perform KMeans clustering with 2 clusters for each class

```
In [9]: #dct kmeans 2
        cluster tic =
        time.time()
        kmeans_dct2=kmeans(2,10,X_Train_DCT,1000
        ) toc = time.time()
        print("elapsed time =",round(toc-tic,4),"sec")
```

elapsed time = 0.8128 sec

- Predict KMeans clustering with 2 clusters on test Set

```
In [10]: #dct kmeans 2 cluster
         Prediction tic = time.time()
         acc_kmeans_dct2,Y_dct_KMeans2=predict_acc(X_Test_DCT,Y_Test,kmeans_dct2
         ) toc = time.time()
         print("accuracy
         =",round(acc_kmeans_dct2,2),"%") print("elapsed
```

```
        time                ="",round(toc-tic,4),"sec\n")
        print("Confusion Matrix:")
        confusion_dct_KMeans2 = pd.crosstab(Y_Test, Y_dct_KMeans2,rownames=[ 'Actual'],
        colnames= display(confusion_dct_KMeans2)
```

```
accuracy = 63.8 %
elapsed time = 0.332 sec
```

```
Confusion Matrix:
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	81	0	2	3	0	8	2	0	2	2	100
1	0	97	0	0	0	0	1	0	1	1	100
2	6	8	59	8	0	2	5	4	6	2	100
3	1	4	8	67	1	4	1	7	2	5	100
4	1	3	2	0	51	3	1	1	16	22	100
5	4	4	6	12	7	48	2	0	12	5	100
6	4	2	9	0	4	4	72	0	5	0	100
7	1	6	3	2	4	0	0	65	2	17	100
8	4	6	2	1	5	7	2	2	49	22	100
9	1	7	1	1	10	0	0	5	26	49	100
All	103	137	92	94	82	76	86	84	121	125	1000

Step 4.3:

- perform KMeans clustering with 4 clusters for each class

```
In [11]: #dct kmeans 4
         clustertic =
         time.time()
         kmeans_dct4=kmeans(4,10,X_Train_DCT,1000
         ) toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 1.383 sec
```

- Predict KMeans clustering with 4 clusters on test Set

```
In [12]: #dct kmeans 4 cluster
         Prediction tic = time.time()
         acc_kmeans_dct4,Y_dct_KMeans4=predict_acc(X_Test_DCT,Y_Test,kmeans_dct4
         ) toc = time.time()
         print("accuracy
         =",round(acc_kmeans_dct4,2),"%") print("elapsed
         time                =",round(toc-tic,4),"sec\n")
         print("Confusion Matrix:")
         confusion_dct_KMeans4 = pd.crosstab(Y_Test, Y_dct_KMeans4,rownames=[ 'Actual'],
         colnames= display(confusion_dct_KMeans4)
```

```
accuracy = 70.3 %
elapsed time = 0.5454 sec
```

```
Confusion Matrix:
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	86	1	0	2	0	5	2	0	4	0	100
1	0	98	0	0	0	0	1	0	1	0	100
2	8	4	63	10	1	2	4	3	3	2	100
3	1	1	7	74	1	5	0	4	1	6	100
4	0	4	1	0	63	3	1	0	7	21	100
5	5	2	3	12	4	54	1	0	12	7	100
6	3	1	8	0	1	1	84	0	2	0	100
7	0	4	1	0	1	0	0	74	2	18	100
8	10	4	3	6	7	4	4	1	46	15	100
9	0	5	2	1	19	1	0	3	8	61	100
All	113	124	88	105	97	75	97	85	86	130	1000

Step 4.4:

- perform KMeans clustering with 8 clusters for each class

```
In [13]: #dct kmeans 8
         cluster tic =
         time.time()
         kmeans_dct8=kmeans(8,10,X_Train_DCT,1000
         ) toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 1.8654 sec
```

- Predict KMeans clustering with 8 clusters on test Set

```
In [14]: #dct kmeans 8 cluster
         Prediction tic = time.time()
         acc_kmeans_dct8,Y_dct_KMeans8=predict_acc(X_Test_DCT,Y_Test,kmeans_dct8
         ) toc = time.time()
         print("accuracy
         =",round(acc_kmeans_dct8,2),"%") print("elapsed
         time                =",round(toc-tic,4),"sec\n")
         print("Confusion Matrix:")
         confusion_dct_KMeans8 = pd.crosstab(Y_Test, Y_dct_KMeans8,rownames=[ 'Actual'],
         colnames= display(confusion_dct_KMeans8)
```

```
accuracy = 75.0 %
elapsed time = 0.8386 sec
```

```
Confusion Matrix:
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	88	1	2	0	0	3	1	0	5	0	100
1	0	95	0	0	0	0	1	0	4	0	100
2	6	2	72	10	0	3	2	2	2	1	100
3	0	1	5	79	1	7	0	4	1	2	100
4	0	2	1	0	74	2	3	0	5	13	100
5	4	2	2	9	0	62	4	0	14	3	100
6	3	1	1	0	1	3	88	0	3	0	100
7	0	1	1	1	1	0	0	79	1	16	100
8	6	2	3	5	8	9	1	1	51	14	100
9	0	4	1	3	17	0	0	5	8	62	100
All	107	111	88	107	102	89	100	91	94	111	1000

Step 4.5:

- perform KMeans clustering with 16 clusters for each class

```
In [15]: #dct kmeans 16 cluster
         tic = time.time()
         kmeans_dct16=kmeans(16,10,X_Train_DCT,1000)
         toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 2.3383 sec
```

- Predict KMeans clustering with 16 clusters on test Set

```
In [16]: #dct kmeans 16 cluster Prediction
         tic = time.time()
         acc_kmeans_dct16,Y_dct_KMeans16=predict_acc(X_Test_DCT,Y_Test,kmeans_dct16)
         toc = time.time()
         print("accuracy =",round(acc_kmeans_dct16,2),"%")
         print("elapsed time =",round(toc-tic,4),"sec\n")
         print("Confusion Matrix:")
         confusion_dct_KMeans16 = pd.crosstab(Y_Test, Y_dct_KMeans16,rownames=[ 'Actual'],
         colnam=[ 'Predicted])
         display(confusion_dct_KMeans16)
```

```
accuracy = 78.2 %
elapsed time = 1.6592 sec
```

```
Confusion Matrix:
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	91	0	2	0	0	2	1	0	4	0	100
1	1	93	0	0	0	0	1	0	4	1	100
2	5	2	75	5	0	1	4	3	5	0	100
3	2	1	3	85	0	5	0	2	0	2	100
4	0	2	0	0	73	2	2	0	6	15	100
5	2	2	0	7	3	69	4	1	9	3	100
6	4	1	2	0	1	1	88	0	3	0	100
7	0	1	1	1	2	0	0	79	1	15	100
8	4	3	3	4	10	9	1	3	55	8	100
9	0	3	0	2	14	0	0	6	1	74	100
All	109	108	86	104	103	89	101	94	88	118	1000

K-Means Clustering on PCA Step 5.1:

- perform KMeans clustering with 1 cluster for each class

```
In [17]: #pca kmeans 1
         Cluster tic =
         time.time()
         kmeans_pca1=kmeans(1,10,X_Train_pca,1000
         ) toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 0.1909 sec
```

- Predict KMeans clustering with 1 cluster on test Set

```
In [18]: #pca kmeans 1 Cluster
         Prediction toc = time.time()
         acc_kmeans_pca1,Y_pca_KMeans1=predict_acc(X_Test_pca,Y_Test,kmeans_pca1)
         toc = time.time()
         print("accuracy
         =",round(acc_kmeans_pca1,2),"%") print("elapsed
         time =",round(toc-tic,4),"sec")
         print("\nConfusion Matrix:")
         confusion_pca_KMeans1 = pd.crosstab(Y_Test, Y_pca_KMeans1,rownames=[ 'Actual'],
         colnames= display(confusion_pca_KMeans1)
```

```
accuracy = 74.5 %
elapsed time = 0.3548 sec
```

```
Confusion Matrix:
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	88	0	1	0	0	3	7	0	1	0	100
1	0	86	8	0	0	3	1	0	1	1	100
2	7	9	69	1	1	1	1	4	5	2	100
3	0	0	4	80	0	6	1	2	5	2	100
4	1	2	1	2	67	1	5	5	2	14	100
5	4	2	5	5	2	74	4	0	4	0	100
6	5	2	3	0	4	5	80	0	1	0	100
7	2	5	8	3	3	1	0	65	0	13	100
8	2	1	3	10	2	4	3	2	66	7	100
9	0	0	4	1	7	0	1	11	6	70	100
All	109	107	106	102	86	98	103	89	91	109	1000

Step 5.2:

- perform KMeans clustering with 2 clusters for each class

```
In [19]: #pca kmeans 2
         Cluster tic =
         time.time()
         kmeans_pca2=kmeans(2,10,X_Train_pca,1000
         ) toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 1.1443 sec
```

- Predict KMeans clustering with 2 clusters on test Set

```
In [20]: #pca kmeans 2 Cluster
         Prediction tic = time.time()
         acc_kmeans_pca2,Y_pca_KMeans2=predict_acc(X_Test_pca,Y_Test,kmeans_pca2)
         toc = time.time()
         print("accuracy
         =",round(acc_kmeans_pca2,2),"%") print("elapsed
         time =",round(toc-tic,4),"sec") print("Confusion
         Matrix:")
         confusion_pca_KMeans2 = pd.crosstab(Y_Test, Y_pca_KMeans2,rownames=[ 'Actual'],
         colnames= display(confusion_pca_KMeans2)
```

```
accuracy = 80.1 %
elapsed time = 0.2988
sec Confusion Matrix:
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	91	0	1	0	0	2	4	0	2	0	100
1	0	98	0	0	0	0	1	0	1	0	100
2	5	7	77	1	1	0	1	3	5	0	100
3	0	0	3	81	0	7	0	2	6	1	100
4	1	2	0	0	65	1	3	7	2	19	100
5	3	2	0	3	3	81	5	0	2	1	100
6	4	0	2	0	2	3	89	0	0	0	100
7	2	6	5	0	5	1	0	71	1	9	100
8	0	0	2	6	2	5	0	0	75	10	100
9	0	1	1	2	6	0	1	14	2	73	100
All	106	116	91	93	84	100	104	97	96	113	1000

Step 5.3:

- perform KMeans clustering with 4 clusters for each class

```
In [21]: #pca kmeans 4
         Cluster tic =
         time.time()
         kmeans_pca4=kmeans(4,10,X_Train_pca,1000
         ) toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 2.0916 sec
```

- Predict KMeans clustering with 4 clusters on test Set

```
In [22]: #pca kmeans 4 Cluster
         Prediction tic = time.time()
         acc_kmeans_pca4,Y_pca_KMeans4=predict_acc(X_Test_pca,Y_Test,kmeans_pca4)
         toc = time.time()
         print("accuracy
         =",round(acc_kmeans_pca4,2),"%") print("elapsed
         time =",round(toc-tic,4),"sec")
         print("\nConfusion Matrix:")
         confusion_pca_KMeans4 = pd.crosstab(Y_Test, Y_pca_KMeans4,rownames=[ 'Actual'],
         colnames= display(confusion_pca_KMeans4)
```

```
accuracy = 84.0 %
elapsed time = 0.4627 sec

Confusion Matrix:
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	97	0	0	0	0	2	0	0	1	0	100
1	0	98	0	0	0	0	2	0	0	0	100
2	4	2	83	2	0	0	1	5	2	1	100
3	0	1	2	87	1	4	0	1	4	0	100
4	0	3	0	0	74	0	3	4	0	16	100
5	0	1	0	0	2	82	8	0	5	2	100
6	5	0	1	0	1	3	89	0	1	0	100
7	1	6	3	0	9	1	0	69	0	11	100
8	0	0	2	3	3	4	0	1	83	4	100
9	1	1	0	0	12	0	0	8	0	78	100
All	108	112	91	92	102	96	103	88	96	112	1000

Step 5.4:

- perform KMeans clustering with 8 clusters for each class

```
In [23]: #pca kmeans 8
         Cluster tic =
         time.time()
         kmeans_pca8=kmeans(8,10,X_Train_pca,1000
         ) toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 2.3143 sec
```

- Predict KMeans clustering with 8 clusters on test Set

```
In [24]: #pca kmeans 8 Cluster
         Prediction tic = time.time()
         acc_kmeans_pca8,Y_pca_KMeans8=predict_acc(X_Test_pca,Y_Test,kmeans_pca8)
         toc = time.time()
         print("accuracy
         =",round(acc_kmeans_pca8,2),"%") print("elapsed
         time =",round(toc-tic,4),"sec")
         print("\nConfusion Matrix:")
         confusion_pca_KMeans8 = pd.crosstab(Y_Test, Y_pca_KMeans8,rownames=[ 'Actual'],
         colnames= display(confusion_pca_KMeans8)
```

```
accuracy = 89.5 %
elapsed time = 0.8857 sec
```

Confusion Matrix:

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	99	0	0	0	0	0	0	0	1	0	100
1	0	99	0	0	0	0	1	0	0	0	100
2	0	2	88	1	1	0	2	2	4	0	100
3	0	1	1	93	0	0	0	1	3	1	100
4	0	4	0	0	83	0	2	2	0	9	100
5	0	0	0	1	2	92	1	0	4	0	100
6	3	0	0	0	0	0	95	0	2	0	100
7	1	3	3	0	1	0	0	80	0	12	100
8	0	0	1	1	1	6	0	0	87	4	100
9	0	1	0	2	9	0	1	8	0	79	100
All	103	110	93	98	97	98	102	93	101	105	1000

Step 5.5:

- perform KMeans clustering with 16 clusters for each class

```
In [25]: #pca kmeans 16
         Cluster tic =
         time.time()
         kmeans_pca16=kmeans(16,10,X_Train_pca,1000
         ) toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 2.6674 sec
```

- Predict KMeans clustering with 16 clusters on test Set

```
In [26]: #pca kmeans 16 Cluster Prediction
         tic = time.time()
         acc_kmeans_pca16,Y_pca_KMeans16=predict_acc(X_Test_pca,Y_Test,kmeans_pca16)
         toc = time.time()
         print("accuracy =",round(acc_kmeans_pca16,2),"%")
         print("elapsed time =",round(toc-tic,4),"sec")
         print("\nConfusion Matrix:")
         confusion_pca_KMeans16 = pd.crosstab(Y_Test, Y_pca_KMeans16,rownames=[ 'Actual'], colnam
         display(confusion_pca_KMeans16)
```

```
accuracy = 91.8 %
elapsed time = 1.6466 sec

Confusion Matrix:
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	98	0	0	0	0	1	1	0	0	0	100
1	0	97	0	0	0	0	2	0	1	0	100
2	0	1	94	0	1	0	1	2	1	0	100
3	0	0	2	92	0	2	0	2	2	0	100
4	0	4	0	0	81	0	2	2	0	11	100
5	0	0	0	2	0	92	3	0	2	1	100
6	0	0	0	0	0	2	98	0	0	0	100
7	0	3	1	0	1	0	0	87	0	8	100
8	0	0	2	2	1	4	0	0	90	1	100
9	0	1	0	2	3	0	1	4	0	89	100
All	98	106	99	98	87	101	108	97	96	110	1000

GMM Clustering on DCT Step 6.1:

- perform KMeans clustering with 1 cluster for each class

```
In [27]: #dct GMM 1
         tic = time.time()
         G_dct1=GMM(1,10,X_Train_DCT,1000)
         toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 0.3606 sec
```

- Predict KMeans clustering with 1 clusters on test Set

```
In [28]: #dct GMM 1 Predictions
         tic = time.time()
         acc_GMM_dct1,Y_dct_GMM1=predict_acc(X_Test_DCT,Y_Test,G_dct1)
         toc = time.time()
         print("accuracy =",round(acc_GMM_dct1,2),"%")
         print("elapsed time =",round(toc-tic,4),"sec")
         print("\nConfusion Matrix:")
         confusion_dct_GMM1 = pd.crosstab(Y_Test, Y_dct_GMM1,rownames=[ 'Actual'], colnames=['Pre
         display(confusion_dct_GMM1)
```

```
accuracy = 59.0 %
elapsed time = 0.1775 sec

Confusion Matrix:
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	81	1	6	0	0	6	2	0	3	1	100
1	0	79	10	0	0	0	1	0	3	7	100
2	11	6	52	6	0	2	9	5	5	4	100
3	2	4	8	64	1	2	0	7	2	10	100
4	3	6	0	1	54	1	1	0	13	21	100
5	7	3	4	9	8	32	4	0	20	13	100
6	6	3	3	0	7	3	68	0	10	0	100
7	2	7	8	0	3	0	0	61	5	14	100
8	2	10	2	1	2	2	8	0	48	25	100
9	3	12	1	1	14	3	1	1	13	51	100
All	117	131	94	82	89	51	94	74	122	146	1000

Step 6.2:

- perform KMeans clustering with 2 cluster for each class

```
In [29]: #dct GMM 2
         tic = time.time()
         G_dct2=GMM(2,10,X_Train_DCT,1000)
         toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 1.7436 sec
```

- Predict KMeans clustering with 2 clusters on test Set

```
In [30]: #dct GMM 2 Predictions
         tic = time.time()
         acc_GMM_dct2,Y_dct_GMM2=predict_acc(X_Test_DCT,Y_Test,G_dct2)
         toc = time.time()
         print("accuracy =",round(acc_GMM_dct2,2),"%")
         print("elapsed time =",round(toc-tic,4),"sec")
         print("\nConfusion Matrix:")
         confusion_dct_GMM2 = pd.crosstab(Y_Test, Y_dct_GMM2,rownames=[ 'Actual']
         , colnames=['Predicted] )
          display(confusion_dct_GMM2)
```

```
accuracy = 65.1 %
elapsed time = 0.2937 sec

Confusion Matrix:
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	82	0	3	2	0	7	2	0	2	2	100
1	0	97	1	0	0	0	1	0	0	1	100
2	5	8	55	12	0	2	7	4	5	2	100
3	1	4	6	71	2	3	0	6	2	5	100
4	1	2	2	0	55	3	1	1	17	18	100
5	5	4	4	10	8	47	3	0	12	7	100
6	4	1	6	0	3	4	76	0	6	0	100
7	1	7	3	2	2	0	0	67	2	16	100
8	3	6	1	1	7	8	2	3	49	20	100
9	0	7	1	1	8	0	1	3	27	52	100
All	102	136	82	99	85	74	93	84	122	123	1000

Step 6.3:

- perform KMeans clustering with 4 cluster for each class

```
In [31]: #dct GMM 4
         tic = time.time()
         G_dct4=GMM(4,10,X_Train_DCT,1000)
         toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 5.66 sec
```

- Predict KMeans clustering with 4 clusters on test Set

```
In [32]: #dct GMM 4 Predictions
         tic = time.time()
         acc_GMM_dct4,Y_dct_GMM4=predict_acc(X_Test_DCT,Y_Test,G_dct4)
         toc = time.time()
         print("accuracy =",round(acc_GMM_dct4,2),"%")
         print("elapsed time =",round(toc-tic,4),"sec")
         print("\nConfusion Matrix:")
         confusion_dct_GMM4 = pd.crosstab(Y_Test, Y_dct_GMM4,rownames=[ 'Actual'],
          colnames=['Predicted'])
         display(confusion_dct_GMM4)
```

```
accuracy = 70.9 %
elapsed time = 0.4784 sec

Confusion Matrix:
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	86	1	2	2	0	3	0	0	5	1	100
1	0	97	0	0	0	0	1	0	2	0	100
2	5	5	61	9	0	2	7	5	4	2	100
3	1	2	5	76	1	3	0	4	2	6	100
4	0	3	1	1	62	5	1	0	9	18	100
5	1	3	2	13	4	58	3	0	12	4	100
6	3	1	2	0	2	0	87	0	5	0	100
7	0	3	2	0	2	0	0	74	1	18	100
8	3	7	2	6	9	14	2	1	41	15	100
9	0	4	1	0	15	1	1	2	9	67	100
All	99	126	78	107	95	86	102	86	90	131	1000

GMM Clustering on PCA Step 7.1:

- perform KMeans clustering with 1 cluster for each class

```
In [33]: #pca GMM 1
         tic = time.time()
         G_pca1=GMM(1,10,X_Train_pca,1000)
         toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 0.9161 sec
```

- Predict KMeans clustering with 1 clusters on test Set

```
In [34]: #pca GMM 1 Predictions
         tic = time.time()
         acc_GMM_pca1,Y_pca_GMM1=predict_acc(X_Test_pca,Y_Test,G_pca1)
         toc = time.time()
         print("accuracy =",round(acc_GMM_pca1,2),"%")
         print("elapsed time =",round(toc-tic,4),"sec")
         print("\nConfusion Matrix:")
         confusion_pca_GMM1 = pd.crosstab(Y_Test, Y_pca_GMM1,rownames=[ 'Actual'], colnames=['Pre
         display(confusion_pca_GMM1)
```

```
accuracy = 74.5 %
elapsed time = 0.1598 sec

Confusion Matrix:
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	88	0	1	0	0	3	7	0	1	0	100
1	0	86	8	0	0	3	1	0	1	1	100
2	7	9	69	1	1	1	1	4	5	2	100
3	0	0	4	80	0	6	1	2	5	2	100
4	1	2	1	2	67	1	5	5	2	14	100
5	4	2	5	5	2	74	4	0	4	0	100
6	5	2	3	0	4	5	80	0	1	0	100
7	2	5	8	3	3	1	0	65	0	13	100
8	2	1	3	10	2	4	3	2	66	7	100
9	0	0	4	1	7	0	1	11	6	70	100
All	109	107	106	102	86	98	103	89	91	109	1000

Step 7.2:

- perform KMeans clustering with 2 cluster for each class

```
In [35]: #pca GMM 2
         tic = time.time()
         G_pca2=GMM(2,10,X_Train_
         pca,1000) toc =
         time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 5.0936 sec
```

- Predict KMeans clustering with 2 clusters on test Set

```
In [36]: #pca GMM 2
         Predictions tic
         = time.time()
         acc_GMM_pca2,Y_pca_GMM2=predict_acc(X_Test_pca,Y_Test
         ,G_pca2) toc = time.time()
         print("accuracy
         =",round(acc_GMM_pca2,2),"%")
         print("elapsed time =",round(toc-
         tic,4),"sec") print("\nConfusion
         Matrix:")
         confusion_pca_GMM2 = pd.crosstab(Y_Test, Y_pca_GMM2,rownames=[ 'Actual'],
         colnames=['Pre display(confusion_pca_GMM2)
```

```
accuracy = 80.0 %
elapsed time = 0.2963 sec
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	91	0	1	0	0	1	4	0	3	0	100
1	0	98	0	0	0	0	1	0	1	0	100
2	5	7	76	1	1	0	1	3	6	0	100
3	0	0	3	82	0	6	0	2	6	1	100
4	1	2	0	2	66	2	3	4	2	18	100
5	4	2	0	3	2	81	5	0	2	1	100
6	4	0	2	0	2	3	89	0	0	0	100
7	2	6	5	0	4	1	0	69	1	12	100
8	0	0	2	5	2	5	0	0	76	10	100
9	0	1	1	1	2	0	2	18	3	72	100
All	107	116	90	94	79	99	105	96	100	114	1000

Step 7.3:

- perform KMeans clustering with 4 cluster for each class

```
In [37]: #pca GMM 4
         tic = time.time()
         G_pca4=GMM(4,10,X_Train_
         pca,1000) toc =
         time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 9.4406 sec
```

- Predict KMeans clustering with 4 clusters on test Set

```
In [38]: #pca GMM 4
         Predictions tic
         = time.time()
         acc_GMM_pca4,Y_pca_GMM4=predict_acc(X_Test_pca,Y_Test
         ,G_pca4) toc = time.time()
         print("accuracy
         =",round(acc_GMM_pca4,2),"%")
         print("elapsed time =",round(toc-
         tic,4),"sec") print("\nConfusion
         Matrix:")
```

```
confusion_pca_GMM4 = pd.crosstab(Y_Test, Y_pca_GMM4,rownames=[ 'Actual'],
colnames=['Pre display(confusion_pca_GMM4)
```

```
accuracy = 83.0 %
elapsed time = 0.5454 sec
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	96	0	0	0	0	2	0	0	2	0	100
1	0	98	0	0	0	0	2	0	0	0	100
2	4	2	83	1	1	0	1	5	3	0	100
3	1	0	2	86	0	3	0	1	6	1	100
4	0	3	0	0	70	0	3	4	0	20	100
5	2	1	0	0	2	81	4	0	7	3	100
6	5	0	1	0	2	3	88	0	1	0	100
7	1	6	3	0	6	1	0	67	0	16	100
8	0	0	2	2	2	5	0	1	86	2	100
9	1	1	0	0	15	0	0	8	0	75	100
All	110	111	91	89	98	95	98	86	105	117	1000

Classification using Linear SVM with DCT Step 8:

- perform Linear SVM Classification

```
In [39]: #dct
         Linear
         SVMtic =
         time.time
         ()
         svm_dct_lin = svm.SVC(kernel=
         'linear')
         svm_dct_lin.fit(X_Train_DCT,Y_
         Train) toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 2.4367 sec
```

- Predict with Linear SVM on test Set

```
In [40]: #dct Linear SVM
Prediction tic = time.time()
```

```
acc_dct_svm=accuracy_score(Y_Test,svm_dct_lin.predict(X_Test_D
CT))*100 toc = time.time()
print('accuracy =
',round(acc_dct_svm,2),"%")
print("elapsed time =",round(toc-
tic,4),"sec") print("\nConfusion
Matrix:")
confusion_svm_dct_lin = pd.crosstab(Y_Test, svm_dct_lin.predict(X_Test_DCT),\
                                    rownames=['Actual'],
                                    colnames=['Predicted'], margin
display(confusion_svm_dct_lin)
```

```
accuracy =  82.0
%elapsed time =
0.1107 sec
```

```
Confusion Matrix:
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	93	0	1	0	0	1	0	0	5	0	100
1	0	92	2	0	1	2	0	0	3	0	100
2	4	1	82	5	1	1	1	1	4	0	100
3	0	1	1	85	0	1	0	7	2	3	100
4	0	2	0	1	83	3	1	0	1	9	100
5	0	5	3	2	8	74	2	1	5	0	100
6	0	1	3	0	3	4	88	0	1	0	100
7	0	1	3	2	0	1	0	83	0	10	100
8	3	0	2	6	5	5	9	0	64	6	100
9	0	3	1	2	9	2	0	5	2	76	100
All	100	106	98	103	110	94	101	97	87	104	1000

Classification using Non-Linear kernel SVM with DCT Step 9:

- perform Non-Linear SVM Classification

```
In [41]: #dct non-
         Linear SVM
         tic =
         time.time()
         svm_dct_nlin = svm.SVC(kernel=
         'poly')
         svm_dct_nlin.fit(X_Train_DCT,Y
         _Train) toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 2.3865 sec
```

- Predict with Non-Linear SVM on test Set

```
In [42]: #dct non-Linear SVM
         Prediction tic =
         time.time()
         acc_dct_nsvm=accuracy_score(Y_Test,svm_dct_nlin.predict(X_Test_D
         CT))*100 toc = time.time()
         print('accuracy =
         ',round(acc_dct_nsvm,2),"%")
         print("elapsed time =",round(toc-
         tic,4),"sec") print("\nConfusion
         Matrix:")
         confusion_svm_dct_nlin = pd.crosstab(Y_Test,svm_dct_nlin.predict(X_Test_DCT),\
                                              rownames=['Actual'],
                                              colnames=['Predicted'], margin
         display(confusion_svm_dct_nlin)
accuracy =  92.1
%elapsed time =
0.0741 sec
```

Confusion Matrix:

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	96	0	1	1	0	1	0	0	1	0	100
1	1	98	1	0	0	0	0	0	0	0	100
2	3	1	92	1	0	0	0	0	3	0	100
3	0	0	2	95	0	0	0	2	0	1	100
4	0	1	0	1	88	0	3	0	1	6	100
5	0	0	1	1	2	93	0	0	3	0	100
6	2	0	0	0	1	0	97	0	0	0	100
7	0	0	0	0	1	0	0	92	0	7	100
8	4	0	2	1	3	4	0	1	84	1	100
9	0	2	0	2	7	0	0	3	0	86	100
All	106	102	99	102	102	98	100	98	92	101	1000

Classification using Linear SVM with PCA Step 10:

- perform Linear SVM Classification

```
In [43]: #pca
         Linear
         SVMtic =
         time.time
         ()
         svm_pca_lin = svm.SVC(kernel=
         'linear')
         svm_pca_lin.fit(X_Train_pca,Y_T
         rain) toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")

elapsed time = 5.1712 sec
```

- Predict with Linear SVM on test Set

```
  In [44]: #pca Linear SVM
Prediction tic = time.time()
         acc_pca_svm=accuracy_score(Y_Test,svm_pca_lin.predict(X_Test_p
         ca))*100 toc = time.time()
         print('accuracy =
         ',round(acc_pca_svm,2),"%")
         print("elapsed time =",round(toc-
         tic,4),"sec") print("\nConfusion
         Matrix:")
         confusion_svm_pca_lin = pd.crosstab(Y_Test,svm_pca_lin.predict(X_Test_pca),\
                                             rownames=['Actual'],
                                             colnames=['Predicted'], margin
         display(confusion_svm_pca_lin)
accuracy =  90.3
%elapsed time =
0.2524 sec
```

Confusion Matrix:

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	97	0	0	0	0	1	1	0	1	0	100
1	0	96	1	0	0	1	1	1	0	0	100
2	4	2	84	2	2	1	1	2	2	0	100
3	0	0	1	92	0	4	0	2	1	0	100
4	0	0	1	1	93	0	1	1	0	3	100
5	0	2	0	2	3	87	4	0	1	1	100
6	0	0	3	0	1	1	95	0	0	0	100
7	0	0	3	1	1	0	0	91	1	3	100
8	2	0	2	1	5	2	2	2	81	3	100
9	0	1	0	0	9	0	0	1	2	87	100
All	103	101	95	99	114	97	105	100	89	97	1000

Classification using Non-Linear Kernel SVM with PCA Step 11:

- perform Non-Linear SVM Classification

```
In [45]: #pca non-
         Linear SVM
         tic =
         time.time()
         svm_pca_nlin = svm.SVC(kernel=
         'poly')
         svm_pca_nlin.fit(X_Train_pca,Y
         _Train) toc = time.time()
         print("elapsed time =",round(toc-tic,4),"sec")
```

elapsed time = 5.5565 sec

- Predict with Non-Linear SVM on test Set

```
In [46]: #pca non-Linear SVM
         Prediction tic =
         time.time()
         acc_pca_nsvm=accuracy_score(Y_Test,svm_pca_nlin.predict(X_Test_p
         ca))*100 toc = time.time()
         print('accuracy =
         ',round(acc_pca_nsvm,2),"%")
         print("elapsed time =",round(toc-
```

```
tic,4),"sec") print("\nConfusion
Matrix:")
confusion_svm_pca_nlin = pd.crosstab(Y_Test,svm_pca_nlin.predict(X_Test_pca),\
                                     rownames=['Actual'],
                                     colnames=['Predicted'], margin
display(confusion_svm_pca_nlin)
```

```
accuracy =  97.4
%elapsed time =
0.5134 sec
Confusion Matrix:
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	98	0	0	0	0	1	1	0	0	0	100
1	0	98	0	0	1	0	1	0	0	0	100
2	0	0	99	0	0	0	1	0	0	0	100
3	0	0	2	96	0	0	0	1	1	0	100
4	0	0	0	0	95	0	2	0	1	2	100
5	0	0	0	2	0	96	0	0	2	0	100
6	0	0	0	0	0	0	100	0	0	0	100
7	0	0	0	0	1	0	0	97	0	2	100
8	0	0	0	1	0	0	0	0	99	0	100
9	0	1	0	0	3	0	0	0	0	96	100
All	98	99	101	99	100	97	105	98	103	100	1000

Concatenation of DCT and PCA features Step 12:

- perform Concatenation

In [47]:
```
X_Train_conc=np.concatenate((X_Train_pca,X_Train_DCT)
,axis=1)
X_Test_conc=np.concatenate((X_Test_pca,X_Test_DCT),ax
is=1)
```

- Perform K-Means clustering with 16 cluster

In [48]:
```
#concatenated
K-Means tic =
time.time()
kmeans_conc=kmeans(16,10,X_Train_co
nc,1000) toc = time.time()
print("elapsed time =",round(toc-tic,4),"sec")
```

```
elapsed time = 2.7132 sec
```

- Predict on the conactenated features

```
In [49]: #concatenated kmeans
         Predictions tic =
         time.time()
         acc_kmeans_conc,Y_conc=predict_acc(X_Test_conc,Y_Test,kmea
         ns_conc) toc = time.time()
         print("accuracy
         =",round(acc_kmeans_conc,2),"%")
         print("elapsed time =",round(toc-
         tic,4),"sec") print("\nConfusion
         Matrix:")
         confusion_kmeans_conc = pd.crosstab(Y_Test,Y_conc,rownames=[ 'Actual'],
         colnames=['Predi display(confusion_kmeans_conc)
```

```
accuracy = 92.0 %
elapsed time = 1.8375 sec
```

Confusion Matrix:

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	98	0	0	0	0	1	0	0	1	0	100
1	0	98	0	0	0	0	1	0	0	1	100
2	0	2	91	0	1	0	1	2	3	0	100
3	0	0	1	92	0	2	0	2	3	0	100
4	0	3	0	0	82	0	2	1	0	12	100
5	0	0	0	1	1	96	1	0	1	0	100
6	0	0	0	0	0	1	99	0	0	0	100
7	0	2	1	0	2	0	0	84	0	11	100
8	1	0	2	2	1	3	0	0	89	2	100
9	0	1	0	1	6	0	0	1	0	91	100
All	99	106	95	96	93	103	104	90	97	117	1000

Comparison of Models

- Different Models Processing Time and accuracies are shown in the following table:

Model type	Time of training	Time of prediction	Accuracy
dct kmeans 1 cluster	0.1575 sec	0.1835 sec	59.0 %
dct kmeans 2 cluster	0.8128 sec	0.3320 sec	63.8 %
dct kmeans 4 cluster	1.3830 sec	0.5454 sec	70.3 %
dct kmeans 8 cluster	1.8654 sec	0.8386 sec	75.0 %
dct kmeans 16 cluster	2.2383 sec	1.6592 sec	78.2 %
pca kmeans 1 Cluster	0.1909 sec	0.3548 sec	74.6 %
pca kmeans 2 Cluster	1.1443 sec	0.2988 sec	80.1 %
pca kmeans 4 Cluster	2.0916 sec	0.4627 sec	84.0 %
pca kmeans 8 Cluster	2.3143 sec	0.8857 sec	89.5 %
pca kmeans 16 Cluster	2.6674 sec	1.6466 sec	91.8 %
dct GMM 1	0.3606 sec	0.1775 sec	59.0 %
dct GMM 2	1.7436 sec	0.2937 sec	65.1 %
dct GMM 4	5.6600 sec	0.7784 sec	70.9 %
pca GMM 1	0.9161 sec	0.1598 sec	74.5 %
pca GMM 2	5.0936 sec	0.2963sec	80.0 %
pca GMM 4	9.4406 sec	0.5454 sec	83.0 %
dct Linear SVM	2.4367 sec	0.1107 sec	82.0 %
dct non-Linear SVM	2.3865 sec	0.0749 sec	92.1 %
pca Linear SVM	5.1712 sec	0.2524 sec	90.3 %
pca non-Linear SVM	5.5565 sec	0.5134 sec	97.4 %
concatenated K-Means	2.7132 sec	1.8375 sec	92.0 %

Conclusion:

Classification of Handwritten numbers is an important part of day to day services and industries, using machines can increase efficiency of this process, descision of which algorithm to use depends on accuracy of this algorithm and processing time of it in our problem next points were noticed:

- As the number of clusters increases the accuracy increases.
- As the number of clusters increases computational time increases.
- GMM gives better accuracy for the same number of clusters over the KMeans, with more computational time.
- Concatenation of the PCA and DCT didn't increase the accuracy dramatically, it almost remained the same as using PCA only.
- SVM with non-linear kernel was the most powerful algorithm of classification among the algorithms for this problem regarding accuracy