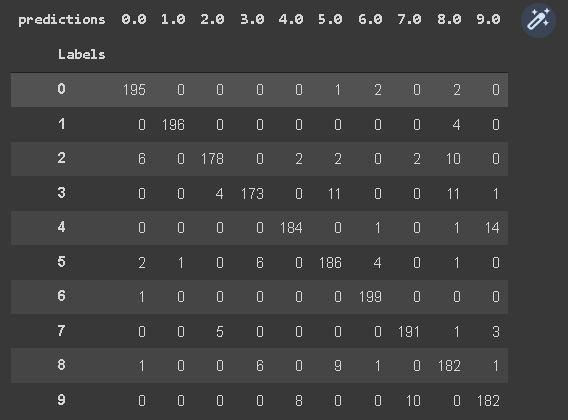


DSP Assignment #3

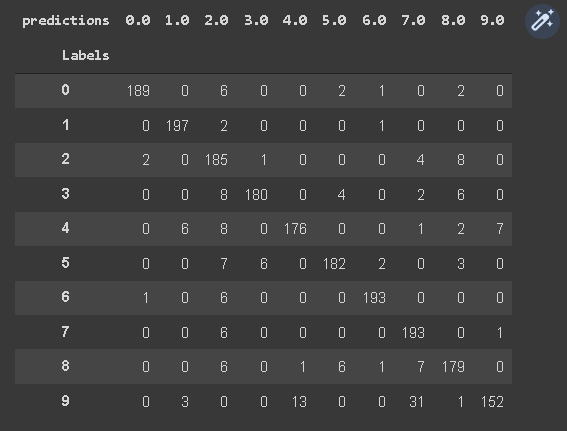
|  |  |  |
| --- | --- | --- |
| Name | Section | BN |
| Ahmed Radwan GadELRab | 1 | 12 |
| Mohamed Ismail Amer | 3 | 42 |
| Moamen Nasser Saad | 3 | 37 |

Submitted to: Dr. Mohsen Rashawn

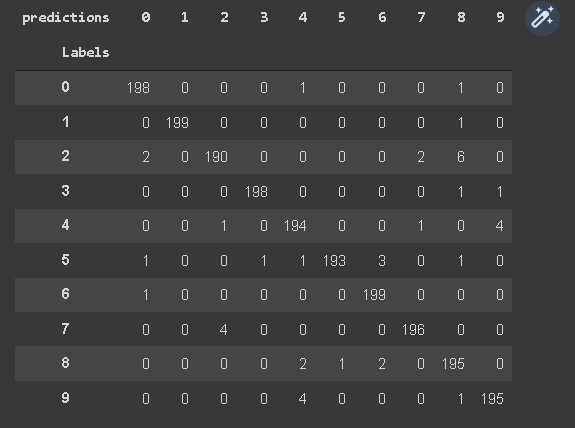
**Confusion Matrix for K-means [16 Cluster per class and DCT]**



**Confusion Matrix for GMM [4 Cluster per class and LDA]**



**Confusion Matrix for SVM [RBF kernel and PCA]**



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | | Features | |  | |
| DCT | | PCA | | Your features | |
| Accuracy | Processing Time | Accuracy | Processing Time | Accuracy | Processing Time |
| Classifier | |  |  |  |  |  |  |
| K-means  Clustering | 1 | 62.85% | 2.15 s | 67.05% | 1.8 s | 89.45% | 0.15 s |
| 4 | 86.5% | 4.2 s | 88.25% | 4.3 s | 89.25% | 1.22 s |
| 16 | 93.3% | 11.1 s | 93% | 8.94 s | 90.2% | 2.5 s |
| 32 | 95% | 18.23 s | 95.35% | 14.96 s | 91.5% | 4.13 s |
| GMM | 1 | 62.1% | 32.26 s | 50.3% | 19.45 s | 86.3% | 2.27 s |
|  | 2 | 75.9% | 32.01 s | 67.55% | 28.2 s | 85.15% | 5.41 s |
|  | 4 | 85.2% | 74.6 s | 80.9% | 51.85 s | 88% | 9.81 s |
| SVM | Linear | 93.9% | 3.32 s | 93.35% | 3.49 s | 89.9% | 2.49 s |
|  | nonlinear\* | 97.6% | 2.46 s | 97.85% | 3.66 s | 91.05% | 0.653 s |

**Notes:**

*We used LDA to extract 9 components of features.*

*We used RBF kernel for SVM.*

**CODE:**

|  |
| --- |
| *# mount drive that contain dataset #Data is on Drive* drive.mount(**"/content/drive"**, force\_remount=**True**) **from** google.colab **import** drive drive.mount(**'/content/drive'**)  *# copy data from drive into colab #!cp -av '/content/drive/MyDrive/dataset' '/content/dataset'  # unzip data* **import** zipfile **with** zipfile.ZipFile(**"/content/drive/MyDrive/dataset/MNIST.zip"**, **'r'**) **as** zip\_ref:  zip\_ref.extractall(**"/content/drive/MyDrive/dataset/"**)  **import** glob **import** numpy **as** np train\_dir = **"/content/drive/MyDrive/dataset/Reduced MNIST Data/Reduced Trainging data"** test\_dir = **"/content/drive/MyDrive/dataset/Reduced MNIST Data/Reduced Testing data"** *# lists contains images paths* train\_list = [] test\_list = []  **for** i **in** range(10):  train\_list.append(glob.glob(**'{}/{}/\*.jpg'**.format(train\_dir,i)))  test\_list.append(glob.glob(**'{}/{}/\*.jpg'**.format(test\_dir,i)))  train\_list = [item **for** sublist **in** train\_list **for** item **in** sublist] test\_list = [item **for** sublist **in** test\_list **for** item **in** sublist]  *# create training and test datasets* **import** PIL **import** matplotlib.pyplot **as** plt  train\_data = np.array([np.array(PIL.Image.open(fname)) \  **for** fname **in** train\_list]) test\_data = np.array([np.array(PIL.Image.open(fname)) \  **for** fname **in** test\_list])  *# create training and test data* **import** re  train\_label = np.array([x **for** x **in** range(10) **for** y **in** range(1000)]) test\_label = np.array([x **for** x **in** range(10) **for** y **in** range(200)])  *# Shuffle training and test data* **from** sklearn.utils **import** shuffle  train\_data, train\_label = shuffle(train\_data/255, train\_label) test\_data, test\_label = shuffle(test\_data/255, test\_label)  *# plot first 30 images in MNIST after being shuffled* fig, ax = plt.subplots(6, 5, figsize = (12, 12)) fig.suptitle(**'First 30 images in MNIST'**) fig.tight\_layout(pad = 0.3, rect = [0, 0, 0.9, 0.9]) **for** x, y **in** [(i, j) **for** i **in** range(6) **for** j **in** range(5)]:  ax[x, y].imshow(train\_data[x + y \* 6].reshape((28, 28)), cmap = **'gray'**)  ax[x, y].set\_title(train\_label[x + y \* 6])  *# DCT Features* **from** scipy.fftpack **import** dct ,idct  **def** dct\_(a):  **return** dct(dct(a.T, norm=**'ortho'**).T, norm=**'ortho'**)  **def** zigzag(a):  x=np.concatenate([np.diagonal(a[::-1,:], i)[::(2\*(i % 2)-1)] \  **for** i **in** range(1-a.shape[0], a.shape[0])])  **return** x[0:200]  **def** idct\_(a):  **return** idct(idct(a.T, norm=**'ortho'**).T, norm=**'ortho'**)  **def** get\_dct\_features(a):  DCT\_features=np.zeros((a.shape[0],200))  **for** i **in** range(a.shape[0]):  DCT\_ordered = zigzag(dct\_(a[i]))  DCT\_features[i] = DCT\_ordered   **return** DCT\_features.reshape((a.shape[0],-1))  DCT\_features\_train = get\_dct\_features(train\_data) print(**"The Size of DCT features for Training are now {} "**.format(DCT\_features\_train.shape)) DCT\_features\_test = get\_dct\_features(test\_data) print(**"The Size of DCT features for Testing are now {} using "**.format(DCT\_features\_test.shape))  *# PCA Feature* **from** sklearn.decomposition **import** PCA **from** sklearn.cluster **import** KMeans  pca = PCA(0.9) pca.fit(train\_data.reshape((train\_data.shape[0],784))) train\_pca = pca.transform(train\_data.reshape((train\_data.shape[0],784))) test\_pca = pca.transform(test\_data.reshape((test\_data.shape[0],784))) print(**"The number of components for 90% varinace is "**, pca.n\_components\_)  *# LDA Features* **from** sklearn.discriminant\_analysis **import** LinearDiscriminantAnalysis **as** LDA  lda = LDA(n\_components=9) lda\_train = lda.fit\_transform(train\_data.reshape((train\_data.shape[0],784)), train\_label) lda\_test = lda.transform(test\_data.reshape((test\_data.shape[0],784)))  *# Kmeans* **def** pred\_labeled(y\_true, y\_pred):  y\_voted\_labels = np.zeros(y\_true.shape)  labels = np.unique(y\_true)  ordered\_labels = np.arange(labels.shape[0])  **for** k **in** range(labels.shape[0]):  y\_true[y\_true==labels[k]] = ordered\_labels[k]  *# Update unique labels* labels = np.unique(y\_true)  *# We set the number of bins to be n\_classes+2 so that   # we count the actual occurence of classes between two consecutive bins  # the bigger being excluded [bin\_i, bin\_i+1[* bins = np.concatenate((labels, [np.max(labels)+1]), axis=0)   **for** cluster **in** np.unique(y\_pred):  hist, \_ = np.histogram(y\_true[y\_pred==cluster], bins=bins)  *# Find the most present label in the cluster* winner = np.argmax(hist)  y\_voted\_labels[y\_pred==cluster] = winner  **return** y\_voted\_labels  **def** purity\_score(y\_true, y\_pred):  *# matrix which will hold the majority-voted labels* y\_voted\_labels = np.zeros(y\_true.shape)  labels = np.unique(y\_true)  ordered\_labels = np.arange(labels.shape[0])  **for** k **in** range(labels.shape[0]):  y\_true[y\_true==labels[k]] = ordered\_labels[k]  *# Update unique labels* labels = np.unique(y\_true)  *# We set the number of bins to be n\_classes+2 so that* bins = np.concatenate((labels, [np.max(labels)+1]), axis=0)  **for** cluster **in** np.unique(y\_pred):  hist, \_ = np.histogram(y\_true[y\_pred==cluster], bins=bins)  *# Find the most present label in the cluster* winner = np.argmax(hist)  y\_voted\_labels[y\_pred==cluster] = winner  **return** accuracy\_score(y\_true, y\_voted\_labels)  *# Function to calculate kmean clusters for required cluster numbers* **import** time **from** sklearn.metrics **import** accuracy\_score **from** sklearn.metrics **import** confusion\_matrix, classification\_report **from** sklearn.metrics **import** multilabel\_confusion\_matrix   **def** kmean\_cluster(train\_data,test\_data,test\_label):  *# multiplying the numbers by 10* cluster\_number = [10,40,160,320]  confusionMatrix = []  **for** i **in** cluster\_number:  print(**"Number of clusters per class ="**,int(i)/10)  *# Initialize the K-Means model* kmeans = KMeans(n\_clusters = i,n\_init=5,max\_iter=10000,algorithm=**'full'**,random\_state=0)  *# Fitting the model to training set* tic = time.time()  kmeans.fit(train\_data)  toc=time.time()  print(**"Training time ="**,round(toc-tic,4))  tic = time.time()  pred\_labels=kmeans.predict(test\_data)  toc=time.time()  print(**"Testing time ="**,round(toc-tic,4))  accuracy=purity\_score(test\_label, pred\_labels)  print(**"Testing accuracy ="**,accuracy)  print(**"\n"**)  print(**"-------K-means-------"**)  kmean\_cluster(DCT\_features\_train,DCT\_features\_test,test\_label)  kmean\_cluster(train\_pca,test\_pca,test\_label)  kmean\_cluster(lda\_train,lda\_test,test\_label)  **from** sklearn.mixture **import** GaussianMixture **def** GMM\_mix(train\_data,test\_data,test\_label):  Mix\_number = [10,20,40]  **for** i **in** Mix\_number:  print(**"Number of clusters per class is :"**,int(i/10))  *# Initialize the GMM model* GMM = GaussianMixture(n\_components=i, n\_init = 10, max\_iter = 6000, covariance\_type = **'diag'**)  *# Fitting the model to training set* tic = time.time()  GMM.fit(train\_data)   toc=time.time()  print(**"Training time ="**,round(toc-tic,4))  tic = time.time()  pred\_labels=GMM.predict(test\_data)  toc=time.time()  print(**"Testing time ="**,round(toc-tic,4))  accuracy=purity\_score(test\_label, pred\_labels)  print(**"Testing accuracy ="**,accuracy)  print(**"\n"**)  print(**"-------GMM-------"**)  GMM\_mix(DCT\_features\_train,DCT\_features\_test,test\_label)  GMM\_mix(train\_pca,test\_pca,test\_label)  GMM\_mix(lda\_train,lda\_test,test\_label)  **from** sklearn **import** svm **def** svm\_models(training\_data,training\_labels,testing\_data,testing\_labels):  **for** kernel **in** (**'linear'**, **'rbf'**):  classifier\_svm = svm.SVC(kernel=kernel, C=6)  tic = time.time()  classifier\_svm.fit(training\_data, training\_labels)  toc = time.time()  print(**"Training time ="**,round(toc-tic,4))  predicted\_labels\_train = classifier\_svm.predict(training\_data)  tic = time.time()  predicted\_labels\_test= classifier\_svm.predict(testing\_data)  toc = time.time()  print(**"Test time ="**,round(toc-tic,4))  print(**"Accuracy using "**+ kernel + **"kernel ="** + str(accuracy\_score(predicted\_labels\_test,testing\_labels)))  print(**'\n'**)  print(**"-------SVM-------"**)  svm\_models(DCT\_features\_train,train\_label,DCT\_features\_test,test\_label)  svm\_models(train\_pca,train\_label,test\_pca,test\_label)  svm\_models(lda\_train,train\_label,lda\_test,test\_label)  **import** pandas **as** pd **def** confusion\_matrix(labels,pred):  *# Create a DataFrame with labels and varieties as columns: df* df = pd.DataFrame({**'Labels'**: labels, **'predictions'**: pred})   *# Create crosstab: ct* ct = pd.crosstab(df[**'Labels'**], df[**'predictions'**])   *# Display ct* display(ct)  *#kmeans\_16 confusion matrix using DCT features* kmeans = KMeans(n\_clusters =160,n\_init=5,max\_iter=10000,algorithm=**'full'**,random\_state=0) kmeans.fit(DCT\_features\_train) pred\_labels=kmeans.predict(DCT\_features\_test) y\_pred=pred\_labeled(test\_label, pred\_labels) confusion\_matrix(test\_label,y\_pred)  *#GMM\_16 confusion matrix using LDA features* GMM = GaussianMixture(n\_components=40, n\_init = 10, max\_iter = 6000, covariance\_type = **'diag'**) GMM.fit(lda\_train) pred\_labels=GMM.predict(lda\_test) y\_pred=pred\_labeled(test\_label, pred\_labels) confusion\_matrix(test\_label,y\_pred)  *#svm\_rbf confusion matrix using pca features* classifier\_svm = svm.SVC(kernel=**'rbf'**, C=5) classifier\_svm.fit(train\_pca, train\_label) predicted\_labels\_test= classifier\_svm.predict(test\_pca) confusion\_matrix(test\_label,predicted\_labels\_test) |

**You can find the code on Google Colab with outputs in the following link:**

<https://colab.research.google.com/drive/1ocsSb0lPRGAuSG108wrK6cl3dmN76-eJ?usp=sharing>