Faculty of Engineering
Cairo University

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]

predictions	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
Labels										
0	195	0	0	0	0	1	2	0	2	0
1	0	196	0	0	0	0	0	0	4	0
2	6	0	178	0	2	2	0	2	10	0
3	0	0	4	173	0	11	0	0	11	1
4	0	0	0	0	184	0	1	0	1	14
5	2	1	0	6	0	186	4	0	1	0
6	1	0	0	0	0	0	199	0	0	0
7	0	0	5	0	0	0	0	191	1	3
8	1	0	0	6	0	9	1	0	182	1
9	0	0	0	0	8	0	0	10	0	182

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

predictions	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
Labels										
0	189	0	6	0	0	2	1	0	2	0
1	0	197	2	0	0	0	1	0	0	0
2	2	0	185	1	0	0	0	4	8	0
3	0	0	8	180	0	4	0	2	6	0
4	0	6	8	0	176	0	0	1	2	7
5	0	0	7	6	0	182	2	0	3	0
6	1	0	6	0	0	0	193	0	0	0
7	0	0	6	0	0	0	0	193	0	1
8	0	0	6	0	1	6	1	7	179	0
9	0	3	0	0	13	0	0	31	1	152

Confusion Matrix for SVM [RBF kernel and PCA]

predictions	0	1	2	3	4	5	6	7	8	9
Labels										
0	198	0	0	0	1	0	0	0	1	0
1	0	199	0	0	0	0	0	0	1	0
2	2	0	190	0	0	0	0	2	6	0
3	0	0	0	198	0	0	0	0	1	1
4	0	0	1	0	194	0	0	1	0	4
5	1	0	0	1	1	193	3	0	1	0
6	1	0	0	0	0	0	199	0	0	0
7	0	0	4	0	0	0	0	196	0	0
8	0	0	0	0	2	1	2	0	195	0
9	0	0	0	0	4	0	0	0	1	195

		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)
```

```python
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