# PCA SVM , GMM and KMeans

|  |
| --- |
| !pip uninstall scikit-learn -y    !pip install -U scikit-learn  import time  import sys  import pandas as pd  import numpy as np  import os  import tensorflow as tf  import cv2  import sklearn  import matplotlib.image as mpimg  import torch  import scipy  from sklearn import mixture  from sklearn.cluster import KMeans  from tensorflow import keras  from keras.layers.core import Dense  from tensorflow.keras.layers import InputLayer, Flatten  from tensorflow.keras import layers, Input  from tensorflow.keras.models import Sequential, Model  from matplotlib import pyplot as plt  from sklearn.decomposition import PCA  from sklearn import svm  from PIL import Image  from numpy import array  from sklearn.pipeline import make\_pipeline  from sklearn.svm import SVC  from sklearn.datasets import make\_classification  from sklearn.svm import LinearSVC  from sklearn.preprocessing import StandardScaler  from sklearn import metrics  from sklearn.metrics import cluster  from scipy import misc  from scipy.special import comb  from scipy.stats import mode  from sklearn.metrics import accuracy\_score  np.set\_printoptions(threshold=sys.maxsize)  import warnings  warnings.filterwarnings('ignore')  from sklearn.metrics import confusion\_matrix    %matplotlib inline  !unzip '/content/Reduced MNIST Data.zip'  def create\_dataset(img\_folder):    img\_data\_array=[]  class\_name=[]    for dir1 in os.listdir(img\_folder):  for file in os.listdir(os.path.join(img\_folder, dir1)):  image\_path= os.path.join(img\_folder, dir1, file)  image= cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)  image=np.array(image)  image = image.astype('double')  image /= 255  img\_data\_array.append(image)  class\_name.append(dir1)  return img\_data\_array, class\_name  def get\_labels\_from\_clustring(true\_labels, predicted\_labels, num\_clusters):  labels = np.zeros\_like(predicted\_labels)    for i in range(num\_clusters):  mask = (predicted\_labels == i)  labels[mask] = mode(true\_labels[mask])[0]  return labels  **# Training data set part**  img\_data, class\_name = create\_dataset (r'/content/Reduced MNIST Data/Reduced Trainging data')  target\_dict={k: v for v, k in enumerate(np.unique(class\_name))}  target\_val= [target\_dict[class\_name[i]] for i in range(len(class\_name))]  X = np.array(img\_data).astype('double')  y = np.array(list(map(int,target\_val)))  # X matrix needs to be reshaped into (Nsamples \* (HxW))  nsamples, nx, ny = X.shape  X\_d2 = X.reshape((nsamples,nx\*ny))  pca = PCA(.95)  pca.fit(X\_d2)  eig\_scores = pca.transform(X\_d2)  # SVM regularization parameter  C = 1.0    # # SVM kernels: Linear, poly, rbf, sigmoid  print("SVM Training Timing details")  t0= time.process\_time ()  clf\_linear = svm.SVC(kernel='linear', C=C).fit(eig\_scores, y)  t1\_svm\_linear = time.process\_time () - t0  print("Time elapsed for SVM Linear:: ", t1\_svm\_linear, " sec")  t0= time.process\_time ()  clf\_poly = svm.SVC(kernel='poly', C=C).fit(eig\_scores, y)  t1\_svm\_poly = time.process\_time () - t0  print("Time elapsed for SVM Poly:: ", t1\_svm\_poly, " sec")  t0= time.process\_time ()  clf\_rbf = svm.SVC(kernel='rbf', C=C).fit(eig\_scores, y)  t1\_svm\_rbf = time.process\_time () - t0  print("Time elapsed for SVM RBF:: ", t1\_svm\_rbf, " sec")  t0= time.process\_time ()  clf\_sigmoid = svm.SVC(kernel='sigmoid', C=C).fit(eig\_scores, y)  t1\_svm\_sigmoid = time.process\_time () - t0  print("Time elapsed for SVM Sigmoid:: ", t1\_svm\_sigmoid, " sec")  # # # # # # K Means classifier  print("KMeans Training Timing details")  t0= time.process\_time ()  kmeans16 = KMeans(n\_clusters=160, init='k-means++', max\_iter=300, n\_init=10, random\_state=0).fit(eig\_scores)  t1\_km16 = time.process\_time () - t0  print("Time elapsed for KMeans 16 Clusters:: ", t1\_km16, " sec")  t0= time.process\_time ()  kmeans4 = KMeans(n\_clusters=40, init='k-means++', max\_iter=300, n\_init=10, random\_state=0).fit(eig\_scores)  t1\_km4 = time.process\_time () - t0  print("Time elapsed for KMeans 4 Clusters:: ", t1\_km4, " sec")  t0= time.process\_time()  kmeans1 = KMeans(n\_clusters=10, init='k-means++', max\_iter=300, n\_init=10, random\_state=0).fit(eig\_scores)  t1\_km1 = time.process\_time() - t0  print("Time elapsed for KMeans 1 Clusters:: ", t1\_km1, " sec")  # # # # # #GMM classifier  print("GMM Training Timing details")  t0= time.process\_time ()  GMM\_1 = sklearn.mixture.GaussianMixture(n\_components=10, max\_iter=300, tol=1e-4, random\_state=42, init\_params='kmeans', covariance\_type = 'spherical').fit(eig\_scores)  t1\_gmm1 = time.process\_time () - t0  print("Time elapsed for GaussianMixture 1 Component:: ", t1\_gmm1, " sec")  t0= time.process\_time ()  GMM\_4 = sklearn.mixture.GaussianMixture(n\_components=40, max\_iter=300, tol=1e-4, random\_state=42, init\_params='kmeans', covariance\_type = 'spherical').fit(eig\_scores)  t1\_gmm4 = time.process\_time () - t0  print("Time elapsed for GaussianMixture 4 Component:: ", t1\_gmm4, " sec")  t0= time.process\_time ()  GMM\_1\_6 = sklearn.mixture.GaussianMixture(n\_components=160, max\_iter=300, tol=1e-4, random\_state=42, init\_params='kmeans', covariance\_type = 'spherical').fit(eig\_scores)  t1\_gmm16 = time.process\_time () - t0  print("Time elapsed for GaussianMixture 16 Component:: ", t1\_gmm16, " sec")  **## Test dataset part**  img\_data\_test, class\_name\_test = create\_dataset(r'/content/Reduced MNIST Data/Reduced Testing data')  target\_dict\_test={k: v for v, k in enumerate(np.unique(class\_name\_test))}  target\_val\_test= [target\_dict\_test[class\_name\_test[i]] for i in range(len(class\_name\_test))]  X\_test = np.array(img\_data\_test)  y = np.array(list(map(int, target\_val\_test)))  # Reshape data to get ready to get the eigen scores  nsamples, nx, ny = X\_test.shape  X\_d2\_test = X\_test.reshape((nsamples,nx\*ny))  eig\_scores = pca.transform(X\_d2\_test)  print("--------------------------------------------------------")  print("------------- Outputs for PCA features -----------------")  print("--------------------------------------------------------")  print("SVM\_LINEAR ModelScore = ", clf\_linear.score(eig\_scores, y) \* 100, "%")  print("Time elapsed for SVM Linear:: ", t1\_svm\_linear, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,clf\_linear.predict(eig\_scores))  print(cm)  print(" ")  # plt.imshow(cm, cmap='binary')  print("SVM\_POLY kernel Score = ", clf\_poly.score(eig\_scores, y) \* 100, "%")  print("Time elapsed for SVM Poly:: ", t1\_svm\_poly, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,clf\_poly.predict(eig\_scores))  print(cm)  print(" ")  print("SVM\_RBF kernel Score = ", clf\_rbf.score(eig\_scores, y) \* 100, "%")  print("Time elapsed for SVM RBF:: ", t1\_svm\_rbf, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,clf\_poly.predict(eig\_scores))  print(cm)  print(" ")  print("SVM\_SIGMOID kernel Score = ", clf\_sigmoid.score(eig\_scores, y) \* 100, "%")  print("Time elapsed for SVM Sigmoid:: ", t1\_svm\_sigmoid, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,clf\_sigmoid.predict(eig\_scores))  print(cm)  print(" ")  predicted\_labels16 = get\_labels\_from\_clustring(y, kmeans16.predict(eig\_scores), 160)  predicted\_labels4 = get\_labels\_from\_clustring(y, kmeans4.predict(eig\_scores), 40)  predicted\_labels1 = get\_labels\_from\_clustring(y, kmeans1.predict(eig\_scores), 10)  print("Kmeans 1 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels1) \* 100, "%")  print("Time elapsed for KMeans 1 Clusters:: ", t1\_km1, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels1)  print(cm)  print(" ")  print("Kmeans 4 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels4) \* 100, "%")  print("Time elapsed for KMeans 4 Clusters:: ", t1\_km4, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels4)  print(cm)  print(" ")  print("Kmeans 16 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels16) \* 100, "%")  print("Time elapsed for KMeans 16 Clusters:: ", t1\_km16, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels16)  print(cm)  print(" ")  predicted\_labels16 = get\_labels\_from\_clustring(y, GMM\_1\_6.predict(eig\_scores), 160)  predicted\_labels4 = get\_labels\_from\_clustring(y, GMM\_4.predict(eig\_scores), 40)  predicted\_labels1 = get\_labels\_from\_clustring(y, GMM\_1.predict(eig\_scores), 10)  print("GMM 1 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels1) \* 100, "%")  print("Time elapsed for GaussianMixture 1 Component:: ", t1\_gmm1, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels1)  print(cm)  print(" ")  print("GMM 4 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels4) \* 100, "%")  print("Time elapsed for GaussianMixture 4 Component:: ", t1\_gmm4, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels4)  print(cm)  print(" ")  print("GMM 16 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels16) \* 100, "%")  print("Time elapsed for GaussianMixture 16 Component:: ", t1\_gmm16, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels16)  print(cm)  print(" ") |

# DCT features

|  |
| --- |
| !pip uninstall scikit-learn -y  !pip install -U scikit-learn  import sys  import pandas as pd  import numpy as np  import os  import tensorflow as tf  import cv2  import sklearn  import matplotlib.image as mpimg  import torch  import scipy  np.set\_printoptions(threshold=sys.maxsize)  import time  import itertools  from sklearn import mixture  from sklearn.cluster import KMeans  from tensorflow import keras  from keras.layers.core import Dense  from tensorflow.keras.layers import InputLayer, Flatten  from tensorflow.keras import layers, Input  from tensorflow.keras.models import Sequential, Model  from matplotlib import pyplot as plt  from sklearn.decomposition import PCA  from sklearn import svm  from PIL import Image  from numpy import array  from sklearn.pipeline import make\_pipeline  from sklearn.svm import SVC  from sklearn.datasets import make\_classification  from sklearn.svm import LinearSVC  from sklearn.preprocessing import StandardScaler  from sklearn import metrics  from sklearn.metrics import cluster  from scipy import misc  from scipy.special import comb  from sklearn.metrics import accuracy\_score  from scipy.stats import mode  from scipy.fftpack import fft, dct  from sklearn.metrics import confusion\_matrix  %matplotlib inline  !unzip '/content/Reduced MNIST Data.zip'  def create\_dataset(img\_folder):    img\_data\_array=[]  class\_name=[]    for dir1 in os.listdir(img\_folder):  for file in os.listdir(os.path.join(img\_folder, dir1)):  image\_path= os.path.join(img\_folder, dir1, file)  image= cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)  image=np.array(image)  image = image.astype('double')  image /= 255  img\_data\_array.append(image)  class\_name.append(dir1)  return img\_data\_array, class\_name  def get\_labels\_from\_clustring(true\_labels, predicted\_labels, num\_clusters):  labels = np.zeros\_like(predicted\_labels)    for i in range(num\_clusters):  mask = (predicted\_labels == i)  labels[mask] = mode(true\_labels[mask])[0]  return labels  def zigzag(input):  #initializing the variables  #----------------------------------  h = 0  v = 0  vmin = 0  hmin = 0  vmax = input.shape[0]  hmax = input.shape[1]    #print(vmax ,hmax )  i = 0  output = np.zeros(( vmax \* hmax))  #----------------------------------  while ((v < vmax) and (h < hmax)):    if ((h + v) % 2) == 0: # going up    if (v == vmin):  #print(1)  output[i] = input[v, h] # if we got to the first line  if (h == hmax):  v = v + 1  else:  h = h + 1  i = i + 1  elif ((h == hmax -1 ) and (v < vmax)): # if we got to the last column  #print(2)  output[i] = input[v, h]  v = v + 1  i = i + 1  elif ((v > vmin) and (h < hmax -1 )): # all other cases  #print(3)  output[i] = input[v, h]  v = v - 1  h = h + 1  i = i + 1    else: # going down  if ((v == vmax -1) and (h <= hmax -1)): # if we got to the last line  #print(4)  output[i] = input[v, h]  h = h + 1  i = i + 1    elif (h == hmin): # if we got to the first column  #print(5)  output[i] = input[v, h]  if (v == vmax -1):  h = h + 1  else:  v = v + 1  i = i + 1  elif ((v < vmax -1) and (h > hmin)): # all other cases  #print(6)  output[i] = input[v, h]  v = v + 1  h = h - 1  i = i + 1  if ((v == vmax-1) and (h == hmax-1)): # bottom right element  #print(7)  output[i] = input[v, h]  break  #print ('v:',v,', h:',h,', i:',i)  return output  # implement 2D DCT  def dct2(a):  return dct(dct(a.T, norm='ortho').T, norm='ortho')  **# Training data set part**  img\_data, class\_name = create\_dataset (r'/content/Reduced MNIST Data/Reduced Trainging data')  target\_dict={k: v for v, k in enumerate(np.unique(class\_name))}  target\_val= [target\_dict[class\_name[i]] for i in range(len(class\_name))]  X = np.array(img\_data).astype('double')  y = np.array(list(map(int,target\_val)))  # X matrix needs to be reshaped into (Nsamples \* (HxW))  nsamples, nx, ny = X.shape  X\_d2 = X.reshape((nsamples,nx\*ny))  eig\_scores = np.empty((0,180), int)  for i in range(0, 10000):  dct\_out = dct2(X\_d2[i, :].reshape((1, 784)))  z\_out = np.array(zigzag(dct\_out))  z\_out1 = z\_out[:180].reshape(1, 180)  eig\_scores = np.append(eig\_scores, z\_out1, axis = 0)  # # SVM regularization parameter  C = 1.0    # # SVM kernels: Linear, poly, rbf, sigmoid  print("SVM Training Timing details")  t0= time.process\_time ()  clf\_linear = svm.SVC(kernel='linear', C=C).fit(eig\_scores, y)  t1\_svm\_linear = time.process\_time () - t0  print("Time elapsed for SVM Linear:: ", t1\_svm\_linear, " sec")  t0= time.process\_time ()  clf\_poly = svm.SVC(kernel='poly', C=C).fit(eig\_scores, y)  t1\_svm\_poly = time.process\_time () - t0  print("Time elapsed for SVM Poly:: ", t1\_svm\_poly, " sec")  t0= time.process\_time ()  clf\_rbf = svm.SVC(kernel='rbf', C=C).fit(eig\_scores, y)  t1\_svm\_rbf = time.process\_time () - t0  print("Time elapsed for SVM RBF:: ", t1\_svm\_rbf, " sec")  t0= time.process\_time ()  clf\_sigmoid = svm.SVC(kernel='sigmoid', C=C).fit(eig\_scores, y)  t1\_svm\_sigmoid = time.process\_time () - t0  print("Time elapsed for SVM Sigmoid:: ", t1\_svm\_sigmoid, " sec")  # # # # # # K Means classifier  print("KMeans Training Timing details")  t0= time.process\_time ()  kmeans16 = KMeans(n\_clusters=160, init='k-means++', max\_iter=300, n\_init=10, random\_state=0).fit(eig\_scores)  t1\_km16 = time.process\_time () - t0  print("Time elapsed for KMeans 16 Clusters:: ", t1\_km16, " sec")  t0= time.process\_time ()  kmeans4 = KMeans(n\_clusters=40, init='k-means++', max\_iter=300, n\_init=10, random\_state=0).fit(eig\_scores)  t1\_km4 = time.process\_time () - t0  print("Time elapsed for KMeans 4 Clusters:: ", t1\_km4, " sec")  t0= time.process\_time()  kmeans1 = KMeans(n\_clusters=10, init='k-means++', max\_iter=300, n\_init=10, random\_state=0).fit(eig\_scores)  t1\_km1 = time.process\_time() - t0  print("Time elapsed for KMeans 1 Clusters:: ", t1\_km1, " sec")  # # # # # #GMM classifier  print("GMM Training Timing details")  t0= time.process\_time ()  GMM\_1 = sklearn.mixture.GaussianMixture(n\_components=10, max\_iter=300, tol=1e-4, random\_state=42, init\_params='kmeans', covariance\_type = 'spherical').fit(eig\_scores)  t1\_gmm1 = time.process\_time () - t0  print("Time elapsed for GaussianMixture 1 Component:: ", t1\_gmm1, " sec")  t0= time.process\_time ()  GMM\_4 = sklearn.mixture.GaussianMixture(n\_components=40, max\_iter=300, tol=1e-4, random\_state=42, init\_params='kmeans', covariance\_type = 'spherical').fit(eig\_scores)  t1\_gmm4 = time.process\_time () - t0  print("Time elapsed for GaussianMixture 4 Component:: ", t1\_gmm4, " sec")  t0= time.process\_time ()  GMM\_1\_6 = sklearn.mixture.GaussianMixture(n\_components=160, max\_iter=300, tol=1e-4, random\_state=42, init\_params='kmeans', covariance\_type = 'spherical').fit(eig\_scores)  t1\_gmm16 = time.process\_time () - t0  print("Time elapsed for GaussianMixture 16 Component:: ", t1\_gmm16, " sec")  **## Test dataset part**  img\_data\_test, class\_name\_test = create\_dataset(r'/content/Reduced MNIST Data/Reduced Testing data')  target\_dict\_test={k: v for v, k in enumerate(np.unique(class\_name\_test))}  target\_val\_test= [target\_dict\_test[class\_name\_test[i]] for i in range(len(class\_name\_test))]  X\_test = np.array(img\_data\_test)  y = np.array(list(map(int, target\_val\_test)))  # Reshape data to get ready to get the eigen scores  nsamples, nx, ny = X\_test.shape  X\_d2\_test = X\_test.reshape((nsamples,nx\*ny))  eig\_scores = np.empty((0,180), int)  for i in range(0, 2000):  dct\_out = dct2(X\_d2\_test[i, :].reshape((1, 784)))  z\_out = np.array(zigzag(dct\_out))  z\_out1 = z\_out[:180].reshape(1, 180)  eig\_scores = np.append(eig\_scores, z\_out1, axis = 0)  print("--------------------------------------------------------")  print("------------- Outputs for DCT features -----------------")  print("--------------------------------------------------------")  print("SVM\_LINEAR ModelScore = ", clf\_linear.score(eig\_scores, y) \* 100, "%")  print("Time elapsed for SVM Linear:: ", t1\_svm\_linear, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,clf\_linear.predict(eig\_scores))  print(cm)  print(" ")  # plt.imshow(cm, cmap='binary')  print("SVM\_POLY kernel Score = ", clf\_poly.score(eig\_scores, y) \* 100, "%")  print("Time elapsed for SVM Poly:: ", t1\_svm\_poly, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,clf\_poly.predict(eig\_scores))  print(cm)  print(" ")  print("SVM\_RBF kernel Score = ", clf\_rbf.score(eig\_scores, y) \* 100, "%")  print("Time elapsed for SVM RBF:: ", t1\_svm\_rbf, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,clf\_poly.predict(eig\_scores))  print(cm)  print(" ")  print("SVM\_SIGMOID kernel Score = ", clf\_sigmoid.score(eig\_scores, y) \* 100, "%")  print("Time elapsed for SVM Sigmoid:: ", t1\_svm\_sigmoid, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,clf\_sigmoid.predict(eig\_scores))  print(cm)  print(" ")  predicted\_labels16 = get\_labels\_from\_clustring(y, kmeans16.predict(eig\_scores), 160)  predicted\_labels4 = get\_labels\_from\_clustring(y, kmeans4.predict(eig\_scores), 40)  predicted\_labels1 = get\_labels\_from\_clustring(y, kmeans1.predict(eig\_scores), 10)  print("Kmeans 1 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels1) \* 100, "%")  print("Time elapsed for KMeans 1 Clusters:: ", t1\_km1, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels1)  print(cm)  print(" ")  print("Kmeans 4 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels4) \* 100, "%")  print("Time elapsed for KMeans 4 Clusters:: ", t1\_km4, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels4)  print(cm)  print(" ")  print("Kmeans 16 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels16) \* 100, "%")  print("Time elapsed for KMeans 16 Clusters:: ", t1\_km16, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels16)  print(cm)  print(" ")  predicted\_labels16 = get\_labels\_from\_clustring(y, GMM\_1\_6.predict(eig\_scores), 160)  predicted\_labels4 = get\_labels\_from\_clustring(y, GMM\_4.predict(eig\_scores), 40)  predicted\_labels1 = get\_labels\_from\_clustring(y, GMM\_1.predict(eig\_scores), 10)  print("GMM 1 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels1) \* 100, "%")  print("Time elapsed for GaussianMixture 1 Component:: ", t1\_gmm1, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels1)  print(cm)  print(" ")  print("GMM 4 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels4) \* 100, "%")  print("Time elapsed for GaussianMixture 4 Component:: ", t1\_gmm4, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels4)  print(cm)  print(" ")  print("GMM 16 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels16) \* 100, "%")  print("Time elapsed for GaussianMixture 16 Component:: ", t1\_gmm16, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels16)  print(cm)  print(" ") |

# ExtraTrees Features

|  |
| --- |
| !pip uninstall scikit-learn -y  !pip install -U scikit-learn  import time  import sys  import pandas as pd  import numpy as np  import os  import tensorflow as tf  import cv2  import sklearn  import matplotlib.image as mpimg  import torch  import scipy  from sklearn import mixture  from sklearn.cluster import KMeans  from tensorflow import keras  from keras.layers.core import Dense  from tensorflow.keras.layers import InputLayer, Flatten  from tensorflow.keras import layers, Input  from tensorflow.keras.models import Sequential, Model  from matplotlib import pyplot as plt  from sklearn.decomposition import PCA  from sklearn import svm  from PIL import Image  from numpy import array  from sklearn.pipeline import make\_pipeline  from sklearn.svm import SVC  from sklearn.datasets import make\_classification  from sklearn.svm import LinearSVC  from sklearn.preprocessing import StandardScaler  from sklearn import metrics  from sklearn.metrics import cluster  from scipy import misc  from scipy.special import comb  from scipy.stats import mode  from sklearn.metrics import accuracy\_score  np.set\_printoptions(threshold=sys.maxsize)  import warnings  warnings.filterwarnings('ignore')  from sklearn.feature\_selection import SelectPercentile  from sklearn.feature\_selection import chi2 , f\_classif  from sklearn.metrics import confusion\_matrix  %matplotlib inline  !unzip '/content/Reduced MNIST Data.zip'  def create\_dataset(img\_folder):    img\_data\_array=[]  class\_name=[]    for dir1 in os.listdir(img\_folder):  for file in os.listdir(os.path.join(img\_folder, dir1)):  image\_path= os.path.join(img\_folder, dir1, file)  image= cv2.imread(image\_path, -1)  image=np.array(image)  image = image.astype('double')  image /= 255  img\_data\_array.append(image)  class\_name.append(dir1)  return img\_data\_array, class\_name  def get\_labels\_from\_clustring(true\_labels, predicted\_labels, num\_clusters):  labels = np.zeros\_like(predicted\_labels)    for i in range(num\_clusters):  mask = (predicted\_labels == i)  labels[mask] = mode(true\_labels[mask])[0]  return labels  **# Training data set part**  from sklearn.manifold import TSNE  from sklearn.feature\_selection import SelectKBest  from sklearn.ensemble import ExtraTreesClassifier  from sklearn.feature\_selection import SelectFromModel  img\_data, class\_name = create\_dataset (r'/content/Reduced MNIST Data/Reduced Trainging data')  target\_dict={k: v for v, k in enumerate(np.unique(class\_name))}  target\_val= [target\_dict[class\_name[i]] for i in range(len(class\_name))]  X = np.array(img\_data).astype('double')  y = np.array(list(map(int,target\_val)))  # X matrix needs to be reshaped into (Nsamples \* (HxW))  nsamples, nx, ny = X.shape  X\_d2 = X.reshape((nsamples,nx\*ny))  clf = ExtraTreesClassifier(n\_estimators=50)  clf = clf.fit(X\_d2, y)  clf.feature\_importances\_  model = SelectFromModel(clf, prefit=True)  X = model.transform(X\_d2)    eig\_scores = X  # SVM regularization parameter  C = 1.0    # # SVM kernels: Linear, poly, rbf, sigmoid  print("SVM Training Timing details")  t0= time.process\_time ()  clf\_linear = svm.SVC(kernel='linear', C=C).fit(eig\_scores, y)  t1\_svm\_linear = time.process\_time () - t0  print("Time elapsed for SVM Linear:: ", t1\_svm\_linear, " sec")  t0= time.process\_time ()  clf\_poly = svm.SVC(kernel='poly', C=C).fit(eig\_scores, y)  t1\_svm\_poly = time.process\_time () - t0  print("Time elapsed for SVM Poly:: ", t1\_svm\_poly, " sec")  t0= time.process\_time ()  clf\_rbf = svm.SVC(kernel='rbf', C=C).fit(eig\_scores, y)  t1\_svm\_rbf = time.process\_time () - t0  print("Time elapsed for SVM RBF:: ", t1\_svm\_rbf, " sec")  t0= time.process\_time ()  clf\_sigmoid = svm.SVC(kernel='sigmoid', C=C).fit(eig\_scores, y)  t1\_svm\_sigmoid = time.process\_time () - t0  print("Time elapsed for SVM Sigmoid:: ", t1\_svm\_sigmoid, " sec")  # # # # # # K Means classifier  print("KMeans Training Timing details")  t0= time.process\_time ()  kmeans16 = KMeans(n\_clusters=160, init='k-means++', max\_iter=300, n\_init=10, random\_state=0).fit(eig\_scores)  t1\_km16 = time.process\_time () - t0  print("Time elapsed for KMeans 16 Clusters:: ", t1\_km16, " sec")  t0= time.process\_time ()  kmeans4 = KMeans(n\_clusters=40, init='k-means++', max\_iter=300, n\_init=10, random\_state=0).fit(eig\_scores)  t1\_km4 = time.process\_time () - t0  print("Time elapsed for KMeans 4 Clusters:: ", t1\_km4, " sec")  t0= time.process\_time()  kmeans1 = KMeans(n\_clusters=10, init='k-means++', max\_iter=300, n\_init=10, random\_state=0).fit(eig\_scores)  t1\_km1 = time.process\_time() - t0  print("Time elapsed for KMeans 1 Clusters:: ", t1\_km1, " sec")  # # # # # #GMM classifier  print("GMM Training Timing details")  t0= time.process\_time ()  GMM\_1 = sklearn.mixture.GaussianMixture(n\_components=10, max\_iter=300, tol=1e-4, random\_state=42, init\_params='kmeans', covariance\_type = 'spherical').fit(eig\_scores)  t1\_gmm1 = time.process\_time () - t0  print("Time elapsed for GaussianMixture 1 Component:: ", t1\_gmm1, " sec")  t0= time.process\_time ()  GMM\_4 = sklearn.mixture.GaussianMixture(n\_components=40, max\_iter=300, tol=1e-4, random\_state=42, init\_params='kmeans', covariance\_type = 'spherical').fit(eig\_scores)  t1\_gmm4 = time.process\_time () - t0  print("Time elapsed for GaussianMixture 4 Component:: ", t1\_gmm4, " sec")  t0= time.process\_time ()  GMM\_1\_6 = sklearn.mixture.GaussianMixture(n\_components=160, max\_iter=300, tol=1e-4, random\_state=42, init\_params='kmeans', covariance\_type = 'spherical').fit(eig\_scores)  t1\_gmm16 = time.process\_time () - t0  print("Time elapsed for GaussianMixture 16 Component:: ", t1\_gmm16, " sec")  **## Test dataset part**  img\_data\_test, class\_name\_test = create\_dataset(r'/content/Reduced MNIST Data/Reduced Testing data')  target\_dict\_test={k: v for v, k in enumerate(np.unique(class\_name\_test))}  target\_val\_test= [target\_dict\_test[class\_name\_test[i]] for i in range(len(class\_name\_test))]  X\_test = np.array(img\_data\_test)  y = np.array(list(map(int, target\_val\_test)))  # Reshape data to get ready to get the eigen scores  nsamples, nx, ny = X\_test.shape  X\_d2\_test = X\_test.reshape((nsamples,nx\*ny))  X = model.transform(X\_d2\_test)  eig\_scores=X  print("--------------------------------------------------------")  print("---------- Outputs for ExtraTree features --------------")  print("--------------------------------------------------------")  print("SVM\_LINEAR ModelScore = ", clf\_linear.score(eig\_scores, y) \* 100, "%")  print("Time elapsed for SVM Linear:: ", t1\_svm\_linear, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,clf\_linear.predict(eig\_scores))  print(cm)  print(" ")  # plt.imshow(cm, cmap='binary')  print("SVM\_POLY kernel Score = ", clf\_poly.score(eig\_scores, y) \* 100, "%")  print("Time elapsed for SVM Poly:: ", t1\_svm\_poly, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,clf\_poly.predict(eig\_scores))  print(cm)  print(" ")  print("SVM\_RBF kernel Score = ", clf\_rbf.score(eig\_scores, y) \* 100, "%")  print("Time elapsed for SVM RBF:: ", t1\_svm\_rbf, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,clf\_poly.predict(eig\_scores))  print(cm)  print(" ")  print("SVM\_SIGMOID kernel Score = ", clf\_sigmoid.score(eig\_scores, y) \* 100, "%")  print("Time elapsed for SVM Sigmoid:: ", t1\_svm\_sigmoid, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,clf\_sigmoid.predict(eig\_scores))  print(cm)  print(" ")  predicted\_labels16 = get\_labels\_from\_clustring(y, kmeans16.predict(eig\_scores), 160)  predicted\_labels4 = get\_labels\_from\_clustring(y, kmeans4.predict(eig\_scores), 40)  predicted\_labels1 = get\_labels\_from\_clustring(y, kmeans1.predict(eig\_scores), 10)  print("Kmeans 1 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels1) \* 100, "%")  print("Time elapsed for KMeans 1 Clusters:: ", t1\_km1, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels1)  print(cm)  print(" ")  print("Kmeans 4 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels4) \* 100, "%")  print("Time elapsed for KMeans 4 Clusters:: ", t1\_km4, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels4)  print(cm)  print(" ")  print("Kmeans 16 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels16) \* 100, "%")  print("Time elapsed for KMeans 16 Clusters:: ", t1\_km16, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels16)  print(cm)  print(" ")  predicted\_labels16 = get\_labels\_from\_clustring(y, GMM\_1\_6.predict(eig\_scores), 160)  predicted\_labels4 = get\_labels\_from\_clustring(y, GMM\_4.predict(eig\_scores), 40)  predicted\_labels1 = get\_labels\_from\_clustring(y, GMM\_1.predict(eig\_scores), 10)  print("GMM 1 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels1) \* 100, "%")  print("Time elapsed for GaussianMixture 1 Component:: ", t1\_gmm1, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels1)  print(cm)  print(" ")  print("GMM 4 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels4) \* 100, "%")  print("Time elapsed for GaussianMixture 4 Component:: ", t1\_gmm4, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels4)  print(cm)  print(" ")  print("GMM 16 ModelScore = ", metrics.accuracy\_score(y, predicted\_labels16) \* 100, "%")  print("Time elapsed for GaussianMixture 16 Component:: ", t1\_gmm16, " sec")  print("Confusion Matix :")  cm = confusion\_matrix(y,predicted\_labels16)  print(cm)  print(" ") |