Assignment 07

Group Members

1. RAJENDRA KUJUR (214161008)
2. ROHIT RAJ SINGH CHAUHAN (214161009)

In [1]:
```
# importing all necessary libraries
from numpy.linalg import eig
import csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Reading the data and making it ready to perform PCA

In [2]:
```
# df reads the first column as labels
df = pd.read_csv('./train.csv')

# taking first row as test data
test_df = df.iloc[0,:]
test_label = test_df['label']
test_df = test_df[1:].to_numpy().reshape(1, -1)

labels = df[df.columns[0]]

# delete the first column i.e. the labels
df.drop(columns=df.columns[0], axis=1, inplace=True)

X = df
X.head(5)
# Calculate mean
mu = np.mean(X, axis =0)
mu.shape
mu = mu.to_numpy().reshape((1,784))

# mean adjusted
Y = X - mu

# calculate covariance matrix
C = np.dot(Y.T, Y)
C = np.cov(Y.T)
EigenValues, EigenVector = eig(C)

# sort and reverse the eigen values indices
indices = np.flipud(np.argsort(EigenValues))
```

function that returns the precision % after giving certain parameters

In [3]:

```
def performanceEvaluation(PCA_d, depth, leaf_threshold, top_k):
  # select first 300 eigen values and corresponding eigen vectors
  A = np.array([EigenValues[indices[i]] for i in range(PCA_d)])
  E = np.array([EigenVector[indices[i]] for i in range(PCA_d)])

  # dimension reduced matrix
  D = np.dot(Y, E.T)

  # find index with maximum variance and calculate its median
  index = np.argmax(np.var(D, axis = 0))
  column = np.array([D[i][index] for i in range(len(D))])
  mvalue = np.median(column)

  # make a node with data at 0th index, index with maxvariance at 1st index, m
  node = [D, index, mvalue, labels]
  f = [node]

  r = 2**(depth-1)-1

  # continue untill we make it to the leaf node
  for index_1 in range(r):
    lchild = []
    rchild = []
    l_label = []
    r_label = []
    # extract values from node
    matrix = f[index_1][0]
    check_index = f[index_1][1]
    check_value = f[index_1][2]
    mat_label = f[index_1][3]

    # check for each matrix's pixel if its value greater than median value the
    for index_2 in range(len(matrix)):
      if matrix[index_2][check_index] < check_value:
        lchild.append(matrix[index_2])
        l_label.append(mat_label[index_2])
      else:
        rchild.append(matrix[index_2])
        r_label.append(mat_label[index_2])

    lchild = np.array(lchild)
    rchild = np.array(rchild)
    l_label = np.array(l_label)
    r_label = np.array(r_label)

    m_thres = 20

    # make a node as left child with data at 0th index, index with maxvariance
    # calculate max variance index and median and make a node and append it to
    if len(lchild) == 0 or len(lchild) < m_thres:
      index_lchild = -1
      median_lchild = -1
      label_l = -1
    else:
      index_lchild = np.argmax(np.var(lchild, axis = 0))
      column = np.array([D[i][index] for i in range(len(lchild))])
      median_lchild = np.median(column)

    left_child = [lchild, index_lchild, median_lchild, l_label]

      # make a node as right child with data at 0th index, index with maxvaria
      # calculate max variance index and median and make a node and append it
    if len(rchild) == 0 or len(lchild) < m_thres:
      index_rchild = -1
      median_rchild = -1
```

```
      label_l = -1
    else:
      index_rchild = np.argmax(np.var(rchild, axis = 0))
      column = np.array([D[i][index] for i in range(len(rchild))])

      median_rchild = 0
      label_r = labels[index_rchild]
    right_child = [rchild, index_rchild, median_rchild, r_label]

    f.append(left_child)
    f.append(right_child)

  # adjust test data according to need
  y = test_df - mu
  y = np.matmul(y, E.T)

  # Searching in KDTree
  KDTree = f
  max_index = 2**(depth)-1
  current_index = 0

  while ((current_index*2+1) < max_index) and (KDTree[current_index*2+1][1]!=
    if y[0, KDTree[current_index][1]] < KDTree[current_index][2]:
      current_index = current_index*2 + 1
    else:
      current_index = current_index*2 + 2

  distance = []
  for i in range(len(KDTree[current_index][0])):
    distance.append(np.linalg.norm(KDTree[current_index][0][i, :] - y))

  ind = np.argpartition(distance, top_k)[ :top_k]
  final_label = np.array(KDTree[current_index][3])[ind]

  # calculate precision according to given formula
  precision = (np.sum((final_label == test_label).astype(int))/top_k)*100
  return precision
```

Test Case 01

In [4]:
```
PCA_d = 48
max_depth = 20
leaf_threshold = 5000
top_k = 20
searchPrecision = performanceEvaluation(PCA_d, max_depth, leaf_threshold, top_
print(f'Precision with given parameters : {searchPrecision}%')
```

```
Precision with given parameters : 95.0%
```

Test Case 02

In [5]:
```
PCA_d = 300
max_depth = 15
leaf_threshold = 2500
top_k = 50
searchPrecision = performanceEvaluation(PCA_d, max_depth, leaf_threshold, top_
print(f'Precision with given parameters : {searchPrecision}%')
```

```
Precision with given parameters : 100.0%
```

Test Case 03

In [6]:
```
PCA_d = 120
max_depth = 9
leaf_threshold = 2890
top_k = 8
searchPrecision = performanceEvaluation(PCA_d, max_depth, leaf_threshold, top_
print(f'Precision with given parameters : {searchPrecision}%')
```

```
Precision with given parameters : 100.0%
```

Test Case 04

In [7]:
```
PCA_d = 70
max_depth = 12
leaf_threshold = 4000
top_k = 30
searchPrecision = performanceEvaluation(PCA_d, max_depth, leaf_threshold, top_
print(f'Precision with given parameters : {searchPrecision}%')
```

```
Precision with given parameters : 93.33333333333333%
```