==

National University of Singapore

Pattern Recognition (EE5907)

Assignment CA2

Matriculation: A0179741U

Name: Rogatiya Mohmad Aspak Arif

==

Important necessary python packages

In [1]:
```
import numpy as np
from pylab import imshow, show, savefig, cm
import matplotlib.pylab as plt
from sklearn.preprocessing import StandardScaler
from itertools import combinations_with_replacement
```

Data file and folder names

In [2]:
```
FILES_DIR = '.\\MNIST_Data\\'
TRAIN_FILE = 'train-images.idx3-ubyte'
TRAIN_LABEL = 'train-labels-idx1-ubyte'
TEST_FILE = 't10k-images-idx3-ubyte'
TEST_LABEL = 't10k-labels-idx1-ubyte'
```

Global constants

In [3]:
```
POLYNOMIAL_ORDER = 2
TOP_EIGENVECTORS = 30
NUM_CLASSES = 10
NUM_FEATURES = 784
```

DigitRecognition class with necessary functions

In [4]:
```
class DigitRecognition:
    def __init__(self):
            with open(FILES_DIR + TRAIN_FILE,'rb') as ftemp:
                datatemp = np.fromfile(ftemp, dtype = np.ubyte)
                self.trainingData=datatemp[16::].reshape(60000,784)
                print('Size of the training set: ',self.trainingData.shape)

            with open(FILES_DIR + TRAIN_LABEL,'rb') as ftemp:
                datatemp = np.fromfile(ftemp, dtype = np.ubyte)
                self.trainingLabels=datatemp[8::]
                print('Size of the training labels: ',self.trainingLabels.shape)

            with open(FILES_DIR + TEST_FILE) as ftemp:
                datatemp = np.fromfile(ftemp, dtype = np.ubyte)
                self.testData=datatemp[16::].reshape(10000,784)
                print('Size of the test set: ',self.testData.shape)

            with open(FILES_DIR + TEST_LABEL,'rb') as ftemp:
                datatemp = np.fromfile(ftemp, dtype = np.ubyte)
                self.testLabels=datatemp[8::]
                print('Size of the test labels: ',self.testLabels.shape)


    def showDigit(self,image):
        """
        Function to show a single image
        """
        imshow(image)
        show()

    def covarianceMatrix(self):
        """
        Function to calculate covariance matrix
        """
        trainingStd = StandardScaler().fit_transform(self.trainingData)
        trainingMean = np.mean(trainingStd, axis=0)
        self.covMatx = np.cov(trainingStd.T)

    def calcEigenValVec(self):
        """
        Function to calculate eigen values and eigen vectors
        """
        eigenVals, eigenVecs = np.linalg.eig(self.covMatx)
        print ("Eigenvals shape: " + str(eigenVals.shape))
        print ("Eigenvecs shape: " + str(eigenVecs.shape))
        self.eigenValVec = [(np.abs(eigenVals[i]), eigenVecs[:,i]) for i in range
        self.eigenValVec = sorted(self.eigenValVec, reverse=True, key=lambda x:x

    def plotEigenVectors(self,numOfEigenVecs):
        """
        Function to plot selected eigen vectors
        """
        f,ax = plt.subplots(1,10,figsize=(50,50))
        for i in range(10):
            ax[i].imshow(self.eigenValVec[i][1].reshape(28,28))
        plt.show()
```

```
    def plotTrainingDigits(self):
        """
        Function to plot selected training digits
        """
        trainDigitList = np.arange(2,21,2)
        f,ax = plt.subplots(1,10,figsize=(50,50))
        for idx,digit in enumerate(trainDigitList):
            ax[idx].imshow(self.trainingData[digit-1].reshape(28,28))
        plt.show()

    def plotTestDigits(self):
        """
        Function to plot selected test digits
        """

        testDigitList = np.asarray([4, 3, 2, 19, 5, 9, 12, 1, 62, 8])
        f,ax = plt.subplots(1,10,figsize=(50,50))
        for idx,digit in enumerate(testDigitList):
            ax[idx].imshow(self.testData[digit-1].reshape(28,28))
        plt.show()

    def plotReconstructedDigits(self):
        """
        Function to reconstruct selected test digits and plot the same
        """
        testDigitList = np.asarray([4, 3, 2, 19, 5, 9, 12, 1, 62, 8])
        reconstructedTestDigits = []
        for idx, digit in enumerate(testDigitList):
            lambdas = []
            for i in range(TOP_EIGENVECTORS):
                lambdas.append(self.testData[digit-1].T.dot(self.eigenValVec[i][
            digitNew = np.zeros(784)
            for i in range(TOP_EIGENVECTORS):
                digitNew += lambdas[i]*self.eigenValVec[i][1]
            reconstructedTestDigits.append(digitNew)

        f,ax = plt.subplots(1,10,figsize=(50,50))
        for idx in range(10):
            ax[idx].imshow(reconstructedTestDigits[idx].reshape(28,28))
        plt.show()

    def calcMatrixY(self):
        """
        Function to calculate Y matrix for linear Regression
        """
        trainingStd = StandardScaler().fit_transform(self.trainingData)
        trainingMean = np.mean(trainingStd, axis=0)
        # Generate normalized training data
        normedTrain = np.subtract(trainingStd,trainingMean)
        # Reconstruct training data
        for idx, digit in enumerate(normedTrain):
            lambdas = []
            for i in range(TOP_EIGENVECTORS):
                lambdas.append(normedTrain[idx].T.dot(self.eigenValVec[i][1]))
            digitNew = np.zeros(784)
            for i in range(TOP_EIGENVECTORS):
```

```
                digitNew += lambdas[i]*self.eigenValVec[i][1]
            normedTrain[idx] = digitNew

        # sort the training data based on class label
        sortedNormedTrain = []
        for i in range(0,60000,1):
            sortedNormedTrain += (normedTrain[self.trainingLabels==i].tolist())
        self.sortedNormedTrain = np.asarray(sortedNormedTrain)

        # sort the class labels
        self.sortedTrainLabels = np.sort(self.trainingLabels)

        # use few top eigenvectors
        top30EigenVecs = []
        for i in range(TOP_EIGENVECTORS):
            top30EigenVecs.append(self.eigenValVec[i][1])
        self.top30EigenVecs = np.asarray(top30EigenVecs)

        # initial Y matrix before appending ones
        initY = np.dot(self.sortedNormedTrain,self.top30EigenVecs.T)

        # append ones
        self.Y = np.append(initY,np.ones(60000,dtype=int).reshape(60000,1),axis=
        print (self.Y.shape)

    def calcMatrixB(self):
        """
        Function to calculate one-hot encoded B matrix for linear and polynomial
        """
        a = np.asarray(self.sortedTrainLabels)
        self.B = np.zeros((a.size, a.max()+1))
        self.B[np.arange(a.size),a] = 1
        print (self.B.shape)

    def calcMatrixA(self):
        """
        Function to calculate matrix A for linear regression
        """
        pY = np.linalg.pinv(self.Y)
        A = pY.dot(self.B)
        print (A.shape)
        self.AT = A.T

    def LinearRegression(self,featureData,labels):
        """
        Linear Regression Classifier Function
        """
        predictedLabel = []
        for j in range(len(labels)):
            result = []
            for i in range(NUM_CLASSES):
                result.append(self.AT[i].T.dot(featureData[j]))
            predictedLabel.append(result.index(max(result)))
        predictedLabels = np.asarray(predictedLabel)
        accuratePredictions = np.sum(labels==predictedLabels)
        accuracy = (accuratePredictions/len(labels))*100
        print ("Accuracy:",accuracy)
```

```
    def prepareTestDataForLinearRegression(self):
        """
        Function to prepare test dataset for linear regression
        """
        trainingStd = StandardScaler().fit_transform(self.trainingData)
        trainingMean = np.mean(trainingStd, axis=0)
        testStd = StandardScaler().fit_transform(self.testData)

        # normalize test data
        normedTest = np.subtract(testStd,trainingMean)

        for idx, digit in enumerate(normedTest):
            lambdas = []
            for i in range(TOP_EIGENVECTORS):
                lambdas.append(normedTest[idx].T.dot(self.eigenValVec[i][1]))
            digitNew = np.zeros(NUM_FEATURES)
            for i in range(TOP_EIGENVECTORS):
                digitNew += lambdas[i]*self.eigenValVec[i][1]
            normedTest[idx] = digitNew

        # Change to 30-dimensional
        shortedTest = np.dot(normedTest,self.top30EigenVecs.T)

        # append ones
        self.normedTest = np.append(shortedTest,np.ones(10000,dtype=int).reshape

        print (self.normedTest.shape)

    def yMatrixPolRegression(self,yMatrix):
        """
        Function to calculate Y matrix for polynomial regression
        """
        newY = []
        for y in (yMatrix):
            combinations = np.asarray(list(combinations_with_replacement(y,POLYN
            prodCombinations = np.prod(combinations,axis=1)
            newY.append(prodCombinations)
        newY = np.asarray(newY)
        return newY

    def aMatrixPolRegression(self,newY):
        """
        Function to calculate A matrix for polynomial regression
        """
        pY = np.linalg.pinv(newY)
        A = pY.dot(self.B)
        self.polAT = A.T

    def polRegression(self,dataset,labels):
        """
        Polynomial Regression Classifier Function
        """
        predictedLabel = []
        for j in range(len(labels)):
            result = []
            for i in range(NUM_CLASSES):
```

```
                result.append(self.polAT[i].T.dot(dataset[j]))
            predictedLabel.append(result.index(max(result)))
        predictedLabels = np.asarray(predictedLabel)
        accuratePredictions = np.sum(labels==predictedLabels)
        accuracy = (accuratePredictions/len(labels))*100
        print ("Accuracy: ", accuracy)
```

In [5]:
```
digRec = DigitRecognition()
```

```
Size of the training set:  (60000, 784)
Size of the training labels:  (60000,)
Size of the test set:  (10000, 784)
Size of the test labels:  (10000,)
```

Calculate Mean and Covariance Matrix

In [6]:
```
digRec.covarianceMatrix()
```

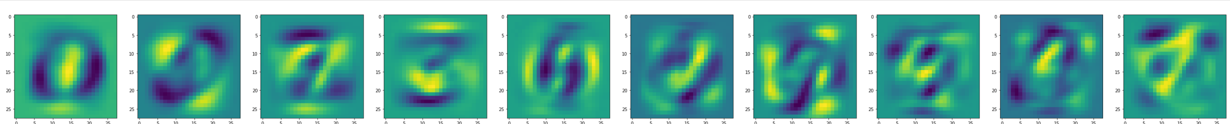
Calculate Eigenvalues and Eigenvectors

In [7]:
```
digRec.calcEigenValVec()
```

```
Eigenvals shape: (784,)
Eigenvecs shape: (784, 784)
```

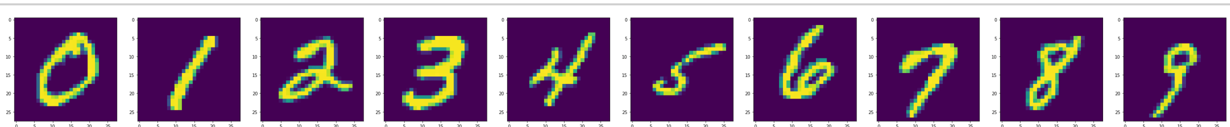
Plot top 10 Eigenvectors

In [8]:
```
digRec.plotEigenVectors(10)
```



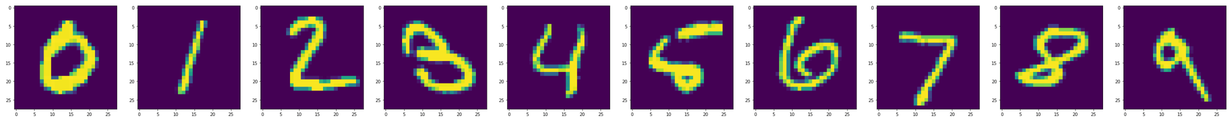
Plot selected training digits

In [9]:
```
digRec.plotTrainingDigits()
```



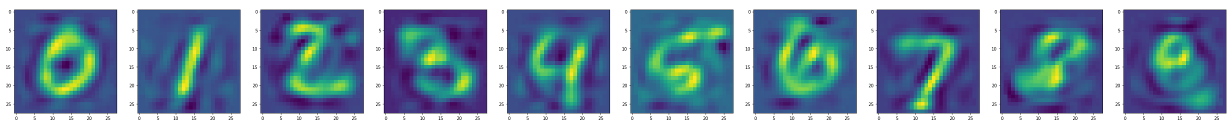
Plot selected test digits

In [10]:
```
digRec.plotTestDigits()
```



Reconstruct the selected 10 test digits and plot the same

In [11]:
```
digRec.plotReconstructedDigits()
```



Calculate Y matrix for linear regression

In [12]:
```
digRec.calcMatrixY()
```

```
(60000, 31)
```

Calculate B matrix for linear and polynomial regression

In [13]:
```
digRec.calcMatrixB()
```

```
(60000, 10)
```

Calculate A matrix for linear regression

In [14]:
```
digRec.calcMatrixA()
```

```
(31, 10)
```

Classify training data using Linear Regression

In [15]:
```
digRec.LinearRegression(digRec.Y,digRec.sortedTrainLabels)
```

```
Accuracy: 80.67333333333333
```

Prepare test data to run linear regression

classifier

In [16]:
```
digRec.prepareTestDataForLinearRegression()
```

```
(10000, 31)
```

Classify test data using Linear Regression

In [17]:
```
digRec.LinearRegression(digRec.normedTest, digRec.testLabels)
```

```
Accuracy: 81.44
```

Classify training data using Polynomial Regression

In [18]:
```
newY = digRec.yMatrixPolRegression(digRec.Y)
digRec.aMatrixPolRegression(newY)
digRec.polRegression(newY,digRec.sortedTrainLabels)
```

```
Accuracy:  94.00333333333334
```

Classify test data using Polynomial Regression

In [19]:
```
newY = digRec.yMatrixPolRegression(digRec.normedTest)
print (newY.shape)

digRec.polRegression(newY,digRec.testLabels)
```

```
(10000, 496)
Accuracy:  93.89999999999999
```

How to improve accuracy?

The test accuracy can be improved with following techniques.

1. By increasing the number of eigenvectors. This will increase the accuracy with a cost of increased execution time.
2. By using higher order polynomial regression. This will also increase the accuracy, but at the same time increasing the execution time drastically.
3. By using a bias term. The accuracy is better when bias term is included in the linear and polynomial classifier.