Pramod Sivaram Kaushik(201407538)
Mohd Salman Khan(201305513)
Sajal Sharma(201305526)

Project 2 : Data Classification

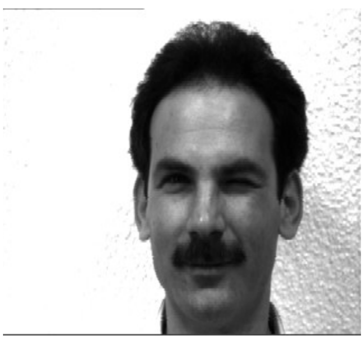
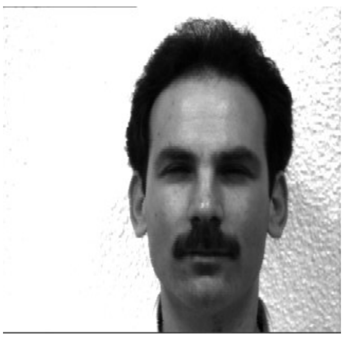
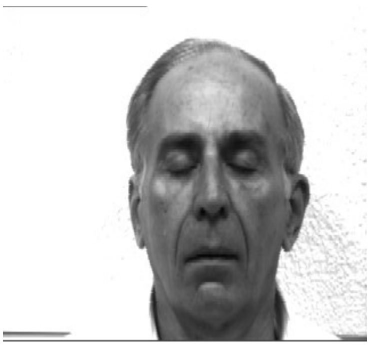
Datasets:

Yale Dataset

The dataset is taken from Original Yale Face Dataset (http://vision.ucsd.edu/content/yale-face-database). The Yale Face Database (size 6.4MB) contains 165 grayscale images in GIF format of 15 individuals. There are 11 images per subject, one per different facial expression or configuration: center-light, w/glasses, happy, left-light, w/no glasses, normal, right-light, sad, sleepy, surprised, and wink.

Some of the images are given below :



Handwriting Recognition Dataset(MNIST)

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

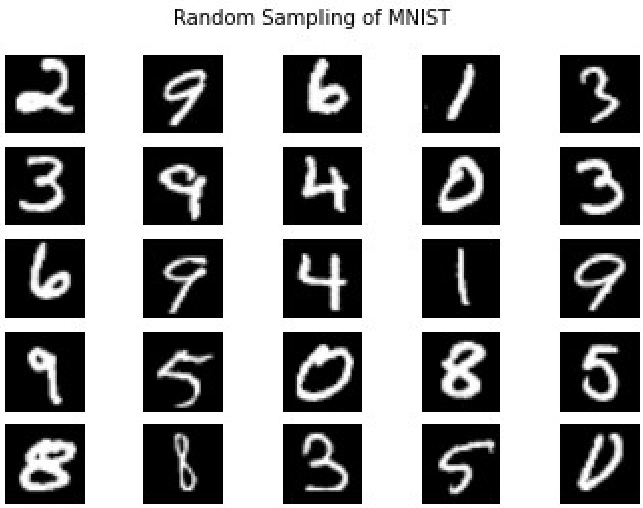
Four files are available on this site:

train-images-idx3-ubyte.gz: training set images (9912422 bytes)

train-labels-idx1-ubyte.gz: training set labels (28881 bytes)

t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)

t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)



Source Codes & their respective Results:

K-Nearest Neighbours (K = 3) :
(for Yale Dataset)

```
import os
import sys
import pdb
import glob
import math
import numpy
import random
import operator
from sets import Set
import scipy as scipy
from scipy.misc import *
from scipy import linalg

# This function loads the images, divides the data intro training and test set
def getNeighbors(trainingSet, testInstance, trainingLabels, k):
    distances = []
    length = len(testInstance)-1

    for x in range(len(trainingSet)):
        #print(trainingSet[x])
        dist = EuclideanDistance(testInstance, trainingSet[x], length)
        distances.append((trainingSet[x], dist, trainingLabels[x]))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        #neighbors.append(distances[x][0])
        neighbors.append(distances[x][2])
    return neighbors

def getResponse(neighbors):
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x]
        #print('response'+response)
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)
    return sortedVotes[0][0]

def LoadImages(directory, split):
    # get a list of all the picture filenames
    gifs = glob.glob(directory + '/*.gif')
    # uncomment the below line when trying an unknown file
    #extraGif = glob.glob("/media/cosmos/Data/College Notes/M.Tech/Semester 4/Statistical Methods in AI/Project -
Face Recognition/2.gif")
    classMap = {}
    testGIF = []
    allLabels = []
    testLabels = []
    trainingGIF = []
    trainingLabels = []
    for i in range(len(gifs)):
        if random.random() < split:
            trainingGIF.append(gifs[i])
            l = gifs[i].split("/");
```

```
            labelName = l[len(l)-1].split(".")[0][-2:]
            trainingLabels.append(labelName)
            allLabels.append(labelName)
        else:
            testGIF.append(gifs[i])
            l = gifs[i].split("/");
            labelName = l[len(l)-1].split(".")[0][-2:]
            testLabels.append(labelName)
            allLabels.append(labelName)
    # uncomment the below 2 lines when trying an unknown file
    #testGIF.append(extraGif[0])
    #testLabels.append("un")
    #allLabels.append("un")
    trainingImgs = numpy.array([imread(i, True).flatten() for i in trainingGIF])
    testImgs = numpy.array([imread(i, True).flatten() for i in testGIF])
    # creating a list of class labels
    allLabels = set(allLabels)
    noOfClasses = len(allLabels)
    sortedLabels = []
    for i in allLabels:
        sortedLabels.append(i)
    sortedLabels = sorted(sortedLabels)
    # creating a mapping for confusion matrix
    j = 0
    for i in sortedLabels:
        classMap[i] = j
        j = j + 1
    return trainingImgs,testImgs,trainingLabels,testLabels,noOfClasses,classMap

def EuclideanDistance(instance1, instance2, length):
        distance = 0
        for x in range(length):
                distance += pow((instance1[x] - instance2[x]), 2)
        return math.sqrt(distance)

def Mahanalobis(x, y):
    return scipy.spatial.distance.mahalanobis(x,y,np.linalg.inv(np.cov(x,y)))

# Run Principal Component Analysis on the input data.
# INPUT  : data    - an n x p matrix
# OUTPUT : e_faces -
#          weights -
#          mu      -
def PCA(data):
    # mean
    mu = numpy.mean(data, 0)
    # mean adjust the data
    ma_data = data - mu
    # run SVD
    e_faces, sigma, v = linalg.svd(ma_data.transpose(), full_matrices=False)
    # compute weights for each image
    weights = numpy.dot(ma_data, e_faces)
    return e_faces, weights, mu

# This function calculates the weight of the test data
def InputWeight(testData, mu, e_faces):
    ma_data = testData - mu
    weights = numpy.dot(ma_data, e_faces)
    return weights

# Reconstruct an image using the given number of principal components.
def Reconstruct(imgIDx, e_faces, weights, mu, npcs):
        # dot weights with the eigenfaces and add to mean
```

```
        recon = mu + numpy.dot(weights[imgIDx, 0:npcs], e_faces[:, 0:npcs].T)
        return recon

# Saves the image in the given directory
def SaveImage(outDIR, subdir, imgID, imgDims, data):
        directory = outDIR + "/" + subdir
        if not os.path.exists(directory): os.makedirs(directory)
        imsave(directory + "/image_" + str(imgID) + ".jpg", data.reshape(imgDims))

# Prints the final results
def PrintResults(wrongPredictedClassCount, unknownLabels, accuracy, correctlyClassifiedDistances, maxDist,
confusionMatrix):
  print "Number of Wrongly Predicted Labels:",wrongPredictedClassCount
  print "Number of Unknown Labels:",unknownLabels
  print "Accuracy:",accuracy,"%"
  print "Max Distances among correcltly classified:",correctlyClassifiedDistances[len(correctlyClassifiedDistances) -
1]
  print "Max Distances among all:",maxDist
  PrintConfusionMatrix(confusionMatrix)

# Prints the confustion matrix
def PrintConfusionMatrix(confusionMatrix):
  print "Confusion Matrix:"
  for i in range(0, len(confusionMatrix)):
    print confusionMatrix[i]

# Predicts the class of test data
def PredictLabelsFromTestData(testData, noOfClasses, mu, e_faces, trainingWeights, testLabels, trainingLabels,
classMap, thresholdDistance, noOfDimensions, k):
  correctlyClassifiedDistances = []
  confusionMatrix = [[0 for i in xrange(noOfClasses)] for i in xrange(noOfClasses)]
  wrongPredictedClassCount = 0
  unknownLabels = 0
  for i in range(len(testData)):
    testWeight=InputWeight(testData[i],mu,e_faces)
    distances = []
    for x in range(len(trainingWeights)):
      dist = EuclideanDistance(testWeight, trainingWeights[x], noOfDimensions)

      #dist = Mahanalobis(testWeight, trainingWeights[x])
      distances.append(dist)
    neighbors = getNeighbors(trainingWeights,testWeight, trainingLabels, k)
    result = getResponse(neighbors)
    predictedLabel = result
    actualLabel = testLabels[i]
    #predictedLabel = ""
    minDist = sys.maxint
    maxDist = -sys.maxint
    for j in range(len(distances)):
      if minDist > distances[j]:
        minDist = distances[j]
        predictedLabel = trainingLabels[j]
      if maxDist < distances[j]:
        maxDist = distances[j]
    confusionMatrix[classMap[actualLabel]][classMap[predictedLabel]] = confusionMatrix[classMap[actualLabel]]
[classMap[predictedLabel]] + 1
    #print "Actual class:",actualLabel
    #print "Predicted class:",predictedLabel
    #print "Min Dist:",minDist
    #print "---------"
    if minDist >= thresholdDistance:
      predictedLabel = "Unknown"
      unknownLabels = unknownLabels + 1
```

```
        elif predictedLabel != actualLabel:
            wrongPredictedClassCount = wrongPredictedClassCount + 1
        else:
            correctlyClassifiedDistances.append(minDist)
    # calculate accuracy
    accuracy = (1 - wrongPredictedClassCount / float(len(testData))) * 100
    correctlyClassifiedDistances.sort()
    PrintResults(wrongPredictedClassCount, unknownLabels, accuracy, correctlyClassifiedDistances, maxDist,
confusionMatrix)
    return accuracy

def main(arg):
    inDIR  = "/home/cosmos/CSStuff/SMAI/Project 2 - Classification/input/yalefaces"
    outDIR = "/home/cosmos/CSStuff/SMAI/Project 2 - Classification/output/Yalefaces Output"
    k = int(arg[0])
    imgDims = (243, 320)
    split = 0.8
    noOfDimensions = 100
    thresholdDistance = 18000.0
    trainingData, testData, trainingLabels, testLabels, noOfClasses, classMap = LoadImages(inDIR, split)
    e_faces, trainingWeights, mu = PCA(trainingData)
                # save mean photo
    imsave(outDIR + "/mean.gif", mu.reshape(imgDims))
    # save each eigenface as an image
    for i in range(e_faces.shape[1]):
                        SaveImage(outDIR, "eigenfaces", i, imgDims, e_faces[:,i])
                # reconstruct each face image using an increasing number of principal components
    reconstructed = []
    for p in range(trainingData.shape[0]):
        reconstructed.append(Reconstruct(p, e_faces, trainingWeights, mu, noOfDimensions))
        imgID = p
        SaveImage(outDIR, "reconstructed/" + str(p), imgID,imgDims, reconstructed[p])
    # Predicting Classes for test data
    accuracy = PredictLabelsFromTestData(testData, noOfClasses, mu, e_faces, trainingWeights, testLabels,
trainingLabels, classMap, thresholdDistance, noOfDimensions, k)
    return accuracy

total = 0.0
for i in range(0, 5):
    total = total + main(sys.argv[1:])
    print "_____________________________________________"
print "Mean Accuracy:",total / 5,"%"
```

Results:

Number of Wrongly Predicted Labels: 5
Number of Unknown Labels: 0
Accuracy: 84.8484848485 %
Max Distances among correcltly classified: 13029.804586
Max Distances among all: 44376.4745935
Confusion Matrix:
[2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 3, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2]

Number of Wrongly Predicted Labels: 3
Number of Unknown Labels: 0
Accuracy: 90.3225806452 %
Max Distances among correcltly classified: 14559.4523311
Max Distances among all: 37696.4649247
Confusion Matrix:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 3, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3]

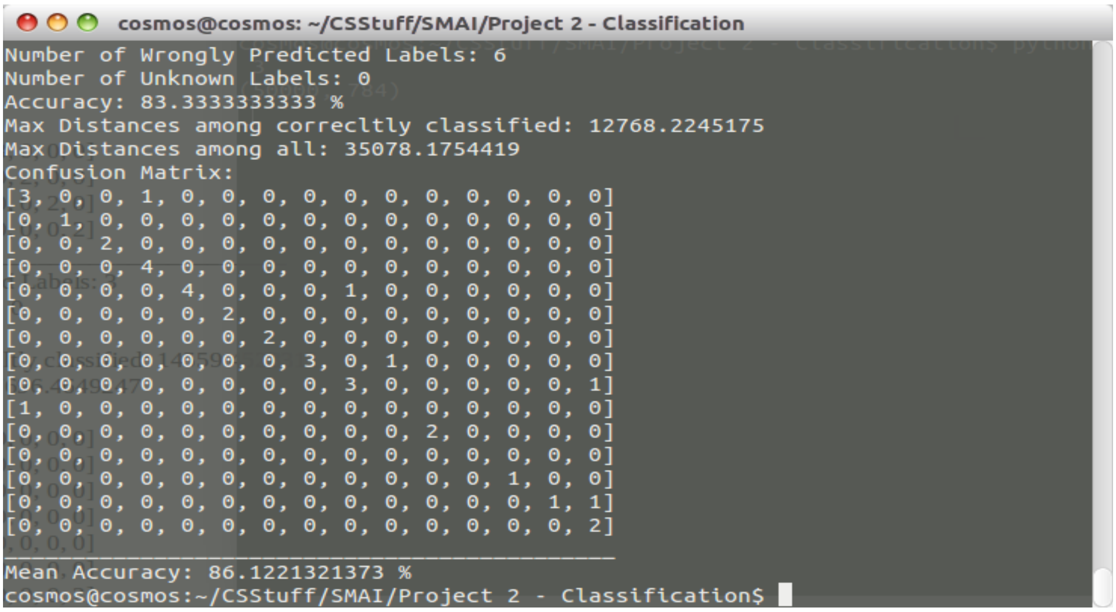
Number of Wrongly Predicted Labels: 4
Number of Unknown Labels: 0
Accuracy: 88.2352941176 %
Max Distances among correcltly classified: 17670.4641544
Max Distances among all: 37367.8789651
Confusion Matrix:
[2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3]

Number of Wrongly Predicted Labels: 5
Number of Unknown Labels: 0
Accuracy: 83.8709677419 %
Max Distances among correcltly classified: 10415.4792513
Max Distances among all: 37176.1344234
Confusion Matrix:
[4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 2, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3]

Number of Wrongly Predicted Labels: 6
Number of Unknown Labels: 0
Accuracy: 83.3333333333 %
Max Distances among correcltly classified: 12768.2245175
Max Distances among all: 35078.1754419
Confusion Matrix:
[3, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 4, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 3, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2]

Mean Accuracy: 86.1221321373 %



For MNIST Dataset:

```
import mnist_loader
import os
import sys
import pdb
import glob
import math
import numpy
import random
import operator
from sets import Set
import scipy as scipy
from scipy.misc import *
from scipy import linalg

# Prints the confustion matrix
def getNeighbors(trainingSet, testInstance, trainingLabels, k):
    distances = []
    length = len(testInstance)-1

    for x in range(len(trainingSet)):
        #print(trainingSet[x])
        dist = EuclideanDistance(testInstance, trainingSet[x], length)
        distances.append((trainingSet[x], dist, trainingLabels[x]))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][2])
    return neighbors

def getResponse(neighbors):
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x]
        #print('response'+response)
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)
    return sortedVotes[0][0]

def CreateConfusionMatrix(predictions, testSet):
    noOfClasses = 10
    confusionMatrix = [[0 for i in xrange(noOfClasses)] for i in xrange(noOfClasses)]
    s=set()
    for i in testSet:
        s.add(i)
    l=list(s)
    d={}
    for x in range(len(l)):
        d[l[x]]=x

    for x in range(len(testSet)):
        if predictions[x]==testSet[x]:
            confusionMatrix[d[testSet[x]]][d[testSet[x]]] = confusionMatrix[d[testSet[x]]][d[testSet[x]]] + 1
        else:
            confusionMatrix[d[testSet[x]]][d[predictions[x]]] = confusionMatrix[d[testSet[x]]][d[predictions[x]]] + 1
    return confusionMatrix

def CalculatePrecisionAndRecall(confusionMatrix, noOfClasses, noOfTestSamples):
            totalRecall = 0.0
```

```
        totalPrecision = 0.0
        totalSpecificity = 0.0
        recall = []
        precision = []
        specificity = []
        #print "Precisions for Different Classes"
        #print "__________________________________"
        for i in range(0, len(confusionMatrix[0])):
            classPrecision = 0.0
            for j in range(0, len(confusionMatrix)):
                classPrecision = classPrecision + confusionMatrix[j][i]
            if classPrecision != 0.0:
                classPrecision = (confusionMatrix[i][i] / float(classPrecision)) * 100
            else:
                classPrecision = 0.0
            precision.append(classPrecision)
            #print "Class Precision for class", i + 1,":", classPrecision
            totalPrecision = totalPrecision + classPrecision
        #print "Recalls for Different Classes"
        #print "__________________________________"
        for i in range(0, len(confusionMatrix)):
            classRecall = 0.0
            for j in range(0, len(confusionMatrix[i])):
                classRecall = classRecall + confusionMatrix[i][j]
            if classRecall != 0.0:
                classRecall = (confusionMatrix[i][i] / float(classRecall)) * 100
            else:
                classRecall = 0.0
            recall.append(classRecall)
            #print "Class Recall for class", i + 1,":", classRecall
            totalRecall = totalRecall + classRecall
        for i in range(0, len(confusionMatrix[0])):
            numerator = noOfTestSamples - confusionMatrix[i][i]
            denominator = numerator
            for j in range(0, len(confusionMatrix)):
                if i != j:
                    denominator = denominator + confusionMatrix[j][i]
            classSpecificity = (numerator / float(denominator)) * 100
            totalSpecificity = totalSpecificity + classSpecificity
            specificity.append(classSpecificity)

        avgRecall = (totalRecall / float(noOfClasses))
        avgPrecision = (totalPrecision / float(noOfClasses))
        avgSpecificity = (totalSpecificity / float(noOfClasses))
        return avgPrecision, avgRecall, avgSpecificity, precision, recall, specificity

def PrintResults(confusionMatrix, avgPrecision, avgRecall, avgSpecificity, precision, recall, specificity):
        print "Confusion Matrix:"
        for i in range(0, len(confusionMatrix)):
            print confusionMatrix[i]
        for i in range(0, len(precision)):
            print "Class", i + 1
            print "--------"
            print "Precision :", precision[i]
            print "Recall :", recall[i]
            print "Specificity :", specificity[i]
            print "\n"
        print "------------------"
        print "Average Recall:", avgRecall
        print "Average Precision:", avgPrecision
        print "Average Specificity:", avgSpecificity

def EuclideanDistance(instance1, instance2, length):
```

```
        distance = 0
        for x in range(length):
            distance += pow((instance1[x] - instance2[x]), 2)
        return math.sqrt(distance)

def GetAccuracy(testLabels, predictions):
   correct = 0

   for x in range(len(testLabels)):
      if testLabels[x] == predictions[x]:
          correct += 1

   return (correct/float(len(testLabels))) * 100.0

def main(k):
        training_data, validation_data, test_data = mnist_loader.load_data()
        k = int(k[0])
        predictions=[]
        for i in range(  len(test_data[0])):
            neighbors=getNeighbors(training_data[0],test_data[0][i],training_data[1],k)
            result=getResponse(neighbors)
            predictions.append(result)
        confusionMatrix = CreateConfusionMatrix(predictions,test_data[1])
        avgPrecision, avgRecall, avgSpecificity, precision, recall, specificity = 
CalculatePrecisionAndRecall(confusionMatrix, 10, len(test_data[1]))
        PrintResults(confusionMatrix, avgPrecision, avgRecall, avgSpecificity, precision, recall, specificity)
        accuracy = GetAccuracy(test_data[1], predictions)
        print "Accuracy :", accuracy

main(sys.argv[1:])
```

Results:

Confusion Matrix:
[967, 0, 1, 0, 0, 5, 4, 1, 2, 0]
[0, 1120, 2, 3, 0, 1, 3, 1, 5, 0]
[9, 1, 962, 7, 10, 1, 13, 11, 16, 2]
[1, 1, 14, 950, 1, 17, 1, 10, 11, 4]
[1, 1, 7, 0, 937, 0, 7, 2, 2, 25]
[7, 4, 5, 33, 7, 808, 11, 2, 10, 5]
[10, 3, 4, 1, 5, 10, 924, 0, 1, 0]
[2, 13, 22, 5, 7, 1, 0, 954, 4, 20]
[4, 6, 6, 14, 8, 24, 10, 8, 891, 3]
[10, 6, 0, 12, 33, 5, 1, 14, 6, 922]
Class 1

Precision : 95.6478733927
Recall : 98.6734693878
Specificity : 99.5152583453

Class 2

Precision : 96.9696969697
Recall : 98.6784140969
Specificity : 99.6074032529

Class 3

Precision : 94.03714565
Recall : 93.2170542636

Specificity : 99.329596659

Class 4

Precision : 92.6829268293
Recall : 94.0594059406
Specificity : 99.1780821918

Class 5

Precision : 92.9563492063
Recall : 95.4175152749
Specificity : 99.2226844756

Class 6

Precision : 92.6605504587
Recall : 90.5829596413
Specificity : 99.3085566119

Class 7

Precision : 94.8665297741
Recall : 96.4509394572
Specificity : 99.4521148367

Class 8

Precision : 95.1146560319
Recall : 92.8015564202
Specificity : 99.4612424409

Class 9

Precision : 93.9873417722
Recall : 91.4784394251
Specificity : 99.3781365918

Class 10

Precision : 93.9857288481
Recall : 91.3776015857
Specificity : 99.3542738317

Average Recall: 94.2737355493
Average Precision: 94.2908798933
Average Specificity: 99.3807349238
Accuracy : 82.15

SVM:

For Yale Dataset:
1. Source Code

```
import sys
import glob
import numpy
import random
from scipy.misc import *
from scipy import linalg
from subprocess import call

def SplitData(directory):
        split = 0.8
        gifs = glob.glob(directory + '/*.gif')
        classMap = {}
        testGIF = []
        allLabels = []
        testLabels = []
        trainingGIF = []
        trainingLabels = []
        for i in range(len(gifs)):
                if random.random() < split:
                        trainingGIF.append(gifs[i])
                        l = gifs[i].split("/");
                        labelName = l[len(l)-1].split(".")[0][-2:]
                        trainingLabels.append(labelName)
                        allLabels.append(labelName)
                else:
                        testGIF.append(gifs[i])
                        l = gifs[i].split("/");
                        labelName = l[len(l)-1].split(".")[0][-2:]
                        testLabels.append(labelName)
                        allLabels.append(labelName)
        trainingImgs = numpy.array([imread(i, True).flatten() for i in trainingGIF])
        testImgs = numpy.array([imread(i, True).flatten() for i in testGIF])
        allLabels = set(allLabels)
        noOfClasses = len(allLabels)
        sortedLabels = []
        for i in allLabels:
                sortedLabels.append(i)
        sortedLabels = sorted(sortedLabels)
    # creating a mapping for confusion matrix
        j = 0
        for i in sortedLabels:
                if i[0] == '0':
                        i = i[1 : ]
                classMap[i] = j
                j = j + 1
        return trainingImgs, testImgs, trainingLabels, testLabels, noOfClasses, classMap

def PCA(data):
    # mean
        mu = numpy.mean(data, 0)
    # mean adjust the data
        ma_data = data - mu
    # run SVD
        e_faces, sigma, v = linalg.svd(ma_data.transpose(), full_matrices=False)
    # compute weights for each image
        weights = numpy.dot(ma_data, e_faces)
        return e_faces, weights, mu
```

```
def WriteIntoFile(data, labels, directory, fileType):
	if fileType == "train":
		fd = open(directory + "/training.txt", 'w+')
	else:
		fd = open(directory + "/test.txt", 'w+')
	for i in range(0, len(data)):
		line = labels[i]
		for j in range(0, len(data[i])):
			line = line + " " + str(j + 1) + ":" + str(data[i][j])
		fd.write(line + "\n")
	fd.close()

def TrainUsingSVM(directory):
	filePath = directory + "/training.txt"
	command = directory + "/svm-train"
	call([command, "-t", "0", filePath])

def InputWeight(testData, mu, e_faces):
   ma_data = testData - mu
   weights = numpy.dot(ma_data, e_faces)
   return weights

def PredictLabels(directory):
	command = directory + "output/svm-predict"
	testFilePath = directory + "output/test.txt"
	modelPath = directory + "/training.txt.model"
	fd = open(directory + "/output/result", 'w+')
	resultFilePath = directory + "output/result"
	call([command, testFilePath, modelPath, resultFilePath])
	fd.close()

def CalculateAccuracy(directory, actualLabels):
	noOfCorrectlyClassifiedSamples = 0
	fd = open(directory + "/result", "r")
	predictedLabels = []
	for line in fd:
		predictedLabels.append(line)
	for i in range(0, len(predictedLabels)):
		predictedLabels[i] = predictedLabels[i][0 : len(predictedLabels[i]) - 1]
	for i in range(0, len(actualLabels)):
		if actualLabels[i][0] == '0':
			actualLabels[i] = actualLabels[i][1 : ]
	for i in range(0, len(actualLabels)):
		if actualLabels[i] == predictedLabels[i]:
			noOfCorrectlyClassifiedSamples = noOfCorrectlyClassifiedSamples + 1
	accuracy = (noOfCorrectlyClassifiedSamples / float(len(actualLabels))) * 100
	return actualLabels, predictedLabels, accuracy

def CalculateConfusionMatrix(actualLabels, predictedLabels, classMap, noOfClasses):
	confusionMatrix = [[0 for i in xrange(noOfClasses)] for i in xrange(noOfClasses)]
	for i in range(0, len(actualLabels)):
		confusionMatrix[classMap[actualLabels[i]]][classMap[predictedLabels[i]]] =
confusionMatrix[classMap[actualLabels[i]]][classMap[predictedLabels[i]]] + 1
	return confusionMatrix

def CalculatePrecisionAndRecall(confusionMatrix, noOfClasses, noOfTestSamples):
	totalRecall = 0.0
	totalPrecision = 0.0
	totalSpecificity = 0.0
	recall = []
	precision = []
	specificity = []
```

```
	#print "Precisions for Different Classes"
	#print "____________________________________"
	for i in range(0, len(confusionMatrix[0])):
		classPrecision = 0.0
		for j in range(0, len(confusionMatrix)):
			classPrecision = classPrecision + confusionMatrix[j][i]
		if classPrecision != 0.0:
			classPrecision = (confusionMatrix[i][i] / float(classPrecision)) * 100
		else:
			classPrecision = 0.0
		precision.append(classPrecision)
		#print "Class Precision for class", i + 1,":", classPrecision
		totalPrecision = totalPrecision + classPrecision
	#print "Recalls for Different Classes"
	#print "____________________________________"
	for i in range(0, len(confusionMatrix)):
		classRecall = 0.0
		for j in range(0, len(confusionMatrix[i])):
			classRecall = classRecall + confusionMatrix[i][j]
		if classRecall != 0.0:
			classRecall = (confusionMatrix[i][i] / float(classRecall)) * 100
		else:
			classRecall = 0.0
		recall.append(classRecall)
		#print "Class Recall for class", i + 1,":", classRecall
		totalRecall = totalRecall + classRecall
	for i in range(0, len(confusionMatrix[0])):
		numerator = noOfTestSamples - confusionMatrix[i][i]
		denominator = numerator
		for j in range(0, len(confusionMatrix)):
			if i != j:
				denominator = denominator + confusionMatrix[j][i]
		classSpecificity = (numerator / float(denominator)) * 100
		totalSpecificity = totalSpecificity + classSpecificity
		specificity.append(classSpecificity)

	avgRecall = (totalRecall / float(noOfClasses))
	avgPrecision = (totalPrecision / float(noOfClasses))
	avgSpecificity = (totalSpecificity / float(noOfClasses))
	return avgPrecision, avgRecall, avgSpecificity, precision, recall, specificity

def PrintResults(confusionMatrix, avgPrecision, avgRecall, avgSpecificity, precision, recall, specificity, accuracy, 
iterationNo):
	print "Confusion Matrix:"
	for i in range(0, len(confusionMatrix)):
		print confusionMatrix[i]
	for i in range(0, len(precision)):
		print "Class", i + 1
		print "--------"
		print "Precision :", precision[i]
		print "Recall :", recall[i]
		print "Specificity :", specificity[i]
		print "\n"
	print "-------------------"
	print "Average Recall:", avgRecall
	print "Average Precision:", avgPrecision
	print "Average Specificity:", avgSpecificity
	print "Accuracy for iteration number", iterationNo + 1,":", accuracy

def main(argv, iterationNo):
	directory = argv[0]
	inDIR = directory + "input/yalefaces"
	outDIR = directory + "output"
```

```
	trainingData, testData, trainingLabels, testLabels, noOfClasses, classMap = SplitData(inDIR)
	e_faces, trainingWeights, mu = PCA(trainingData)
	WriteIntoFile(trainingWeights, trainingLabels, outDIR, "train")
	TrainUsingSVM(outDIR)
	testWeights = InputWeight(testData, mu, e_faces)
	WriteIntoFile(testWeights, testLabels, outDIR, "test")
	PredictLabels(directory)
	actualLabels, predictedLabels, accuracy = CalculateAccuracy(outDIR, testLabels)
	confusionMatrix = CalculateConfusionMatrix(actualLabels, predictedLabels, classMap, noOfClasses)
	avgPrecision, avgRecall, avgSpecificity, precision, recall, specificity =
CalculatePrecisionAndRecall(confusionMatrix, noOfClasses, len(testData))
	PrintResults(confusionMatrix, avgPrecision, avgRecall, avgSpecificity, precision, recall, specificity, accuracy,
iterationNo)
	return accuracy

total = 0.0
for i in range(0, 5):
	accuracy = main(sys.argv[1:], i)
	total = total + accuracy
	print "______________________________________________"
print "Average Accuracy : ",total / 5.0,"%"
```

2. Results

Accuracy = 92.1053% (35/38) (classification)
Confusion Matrix:
[5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 2, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2]
Class 1

Precision : 100.0
Recall : 100.0
Specificity : 100.0

Class 2

Precision : 100.0
Recall : 100.0
Specificity : 100.0

Class 3

Precision : 66.6666666667
Recall : 100.0
Specificity : 97.2972972973

Class 4

Precision : 100.0
Recall : 100.0
Specificity : 100.0

Class 5

Precision : 100.0
Recall : 33.3333333333
Specificity : 100.0

Class 6

Precision : 100.0
Recall : 100.0
Specificity : 100.0

Class 7

Precision : 50.0
Recall : 100.0
Specificity : 97.3684210526

Class 8

Precision : 100.0
Recall : 100.0
Specificity : 100.0

Class 9

Precision : 100.0
Recall : 100.0
Specificity : 100.0

Class 10

Precision : 100.0
Recall : 100.0
Specificity : 100.0

Class 11

Precision : 100.0
Recall : 100.0
Specificity : 100.0

Class 12

Precision : 50.0
Recall : 100.0
Specificity : 97.3684210526

Class 13

Precision : 100.0
Recall : 66.6666666667
Specificity : 100.0

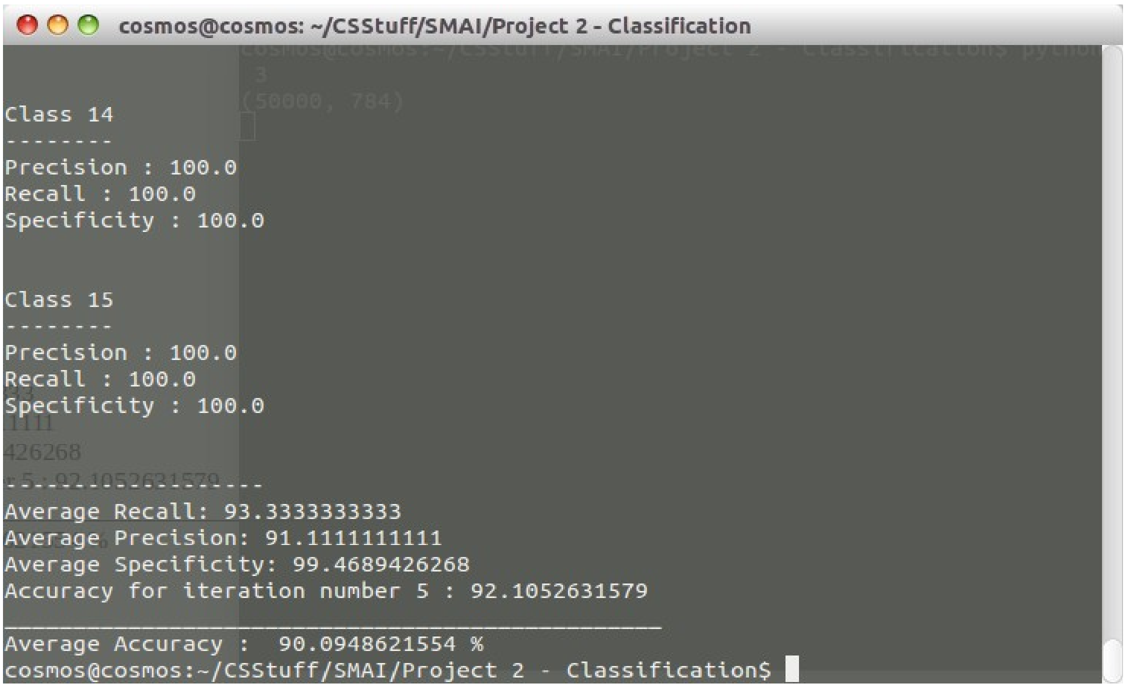
Class 14

Precision : 100.0
Recall : 100.0
Specificity : 100.0

Class 15

Precision : 100.0
Recall : 100.0
Specificity : 100.0

Average Recall: 93.3333333333
Average Precision: 91.1111111111
Average Specificity: 99.4689426268
Accuracy for iteration number 5 : 92.1052631579
____
Average Accuracy : 90.0948621554 %



For MNIST Dataset:

1. Source Code

```
"""
mnist_svm
~~~~~~~~~

A classifier program for recognizing handwritten digits from the MNIST
data set, using an SVM classifier."""

#### Libraries
# My libraries
import mnist_loader

# Third-party libraries
from sklearn import svm

def CreateConfusionMatrix(predictions, testSet):
  noOfClasses = 10
  confusionMatrix = [[0 for i in xrange(noOfClasses)] for i in xrange(noOfClasses)]
  s=set()
  for i in testSet:
      s.add(i)
  l=list(s)
  d={}
  for x in range(len(l)):
    d[l[x]]=x

  for x in range(len(testSet)):
    if predictions[x]==testSet[x]:
      confusionMatrix[d[testSet[x]]][d[testSet[x]]] = confusionMatrix[d[testSet[x]]][d[testSet[x]]] + 1
    else:
      confusionMatrix[d[testSet[x]]][d[predictions[x]]] = confusionMatrix[d[testSet[x]]][d[predictions[x]]] + 1
  return confusionMatrix

def CalculatePrecisionAndRecall(confusionMatrix, noOfClasses, noOfTestSamples):
  totalRecall = 0.0
  totalPrecision = 0.0
  totalSpecificity = 0.0
  recall = []
  precision = []
  specificity = []
  #print "Precisions for Different Classes"
  #print "________________________________"
  for i in range(0, len(confusionMatrix[0])):
    classPrecision = 0.0
    for j in range(0, len(confusionMatrix)):
      classPrecision = classPrecision + confusionMatrix[j][i]
    if classPrecision != 0.0:
      classPrecision = (confusionMatrix[i][i] / float(classPrecision)) * 100
    else:
      classPrecision = 0.0
    precision.append(classPrecision)
    #print "Class Precision for class", i + 1,":", classPrecision
    totalPrecision = totalPrecision + classPrecision
  #print "Recalls for Different Classes"
  #print "________________________________"
  for i in range(0, len(confusionMatrix)):
    classRecall = 0.0
    for j in range(0, len(confusionMatrix[i])):
      classRecall = classRecall + confusionMatrix[i][j]
```

```
        if classRecall != 0.0:
            classRecall = (confusionMatrix[i][i] / float(classRecall)) * 100
        else:
            classRecall = 0.0
        recall.append(classRecall)
        #print "Class Recall for class", i + 1,":", classRecall
        totalRecall = totalRecall + classRecall
    for i in range(0, len(confusionMatrix[0])):
        numerator = noOfTestSamples - confusionMatrix[i][i]
        denominator = numerator
        for j in range(0, len(confusionMatrix)):
            if i != j:
                denominator = denominator + confusionMatrix[j][i]
        classSpecificity = (numerator / float(denominator)) * 100
        totalSpecificity = totalSpecificity + classSpecificity
        specificity.append(classSpecificity)

    avgRecall = (totalRecall / float(noOfClasses))
    avgPrecision = (totalPrecision / float(noOfClasses))
    avgSpecificity = (totalSpecificity / float(noOfClasses))
    return avgPrecision, avgRecall, avgSpecificity, precision, recall, specificity

def PrintResults(confusionMatrix, avgPrecision, avgRecall, avgSpecificity, precision, recall, specificity):
    print "Confusion Matrix:"
    for i in range(0, len(confusionMatrix)):
        print confusionMatrix[i]
    for i in range(0, len(precision)):
        print "Class", i + 1
        print "--------"
        print "Precision :", precision[i]
        print "Recall :", recall[i]
        print "Specificity :", specificity[i]
        print "\n"
    print "-------------------"
    print "Average Recall:", avgRecall
    print "Average Precision:", avgPrecision
    print "Average Specificity:", avgSpecificity

def svm_baseline():
    training_data, validation_data, test_data = mnist_loader.load_data()

    # train
    clf = svm.SVC()
    clf.fit(training_data[0], training_data[1])
    # test
    predictions = [int(a) for a in clf.predict(test_data[0])]
    num_correct = sum(int(a == y) for a, y in zip(predictions, test_data[1]))
    confusionMatrix = CreateConfusionMatrix(predictions, test_data[1])
    avgPrecision, avgRecall, avgSpecificity, precision, recall, specificity =
CalculatePrecisionAndRecall(confusionMatrix, 10, len(test_data[1]))
    PrintResults(confusionMatrix, avgPrecision, avgRecall, avgSpecificity, precision, recall, specificity)
    print "Baseline classifier using an SVM."
    print "%s of %s values correct." % (num_correct, len(test_data[1]))


if __name__ == "__main__":
    svm_baseline()
```

2. Results

Confusion Matrix:
[967, 0, 1, 0, 0, 5, 4, 1, 2, 0]
[0, 1120, 2, 3, 0, 1, 3, 1, 5, 0]
[9, 1, 962, 7, 10, 1, 13, 11, 16, 2]
[1, 1, 14, 950, 1, 17, 1, 10, 11, 4]
[1, 1, 7, 0, 937, 0, 7, 2, 2, 25]
[7, 4, 5, 33, 7, 808, 11, 2, 10, 5]
[10, 3, 4, 1, 5, 10, 924, 0, 1, 0]
[2, 13, 22, 5, 7, 1, 0, 954, 4, 20]
[4, 6, 6, 14, 8, 24, 10, 8, 891, 3]
[10, 6, 0, 12, 33, 5, 1, 14, 6, 922]
Class 1

Precision : 95.6478733927
Recall : 98.6734693878
Specificity : 99.5152583453

Class 2

Precision : 96.9696969697
Recall : 98.6784140969
Specificity : 99.6074032529

Class 3

Precision : 94.03714565
Recall : 93.2170542636
Specificity : 99.329596659

Class 4

Precision : 92.6829268293
Recall : 94.0594059406
Specificity : 99.1780821918

Class 5

Precision : 92.9563492063
Recall : 95.4175152749
Specificity : 99.2226844756

Class 6

Precision : 92.6605504587
Recall : 90.5829596413
Specificity : 99.3085566119

Class 7

Precision : 94.8665297741
Recall : 96.4509394572
Specificity : 99.4521148367

Class 8

Precision : 95.1146560319
Recall : 92.8015564202
Specificity : 99.4612424409

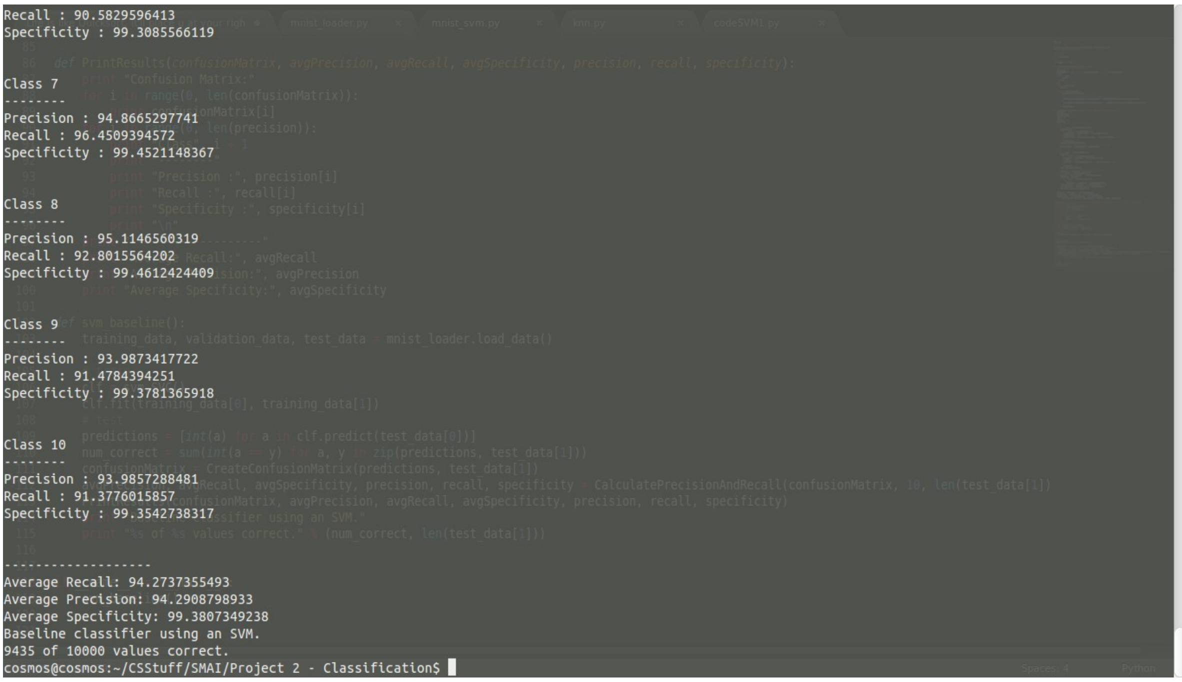
Class 9

Precision : 93.9873417722
Recall : 91.4784394251
Specificity : 99.3781365918

Class 10

Precision : 93.9857288481
Recall : 91.3776015857
Specificity : 99.3542738317

Average Recall: 94.2737355493
Average Precision: 94.2908798933
Average Specificity: 99.3807349238
Baseline classifier using an SVM.
9435 of 10000 values correct.



Multilayer Feedforward Neural Network

For Yale Dataset:
1. Source Code

eigen_face_loader

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr  9 22:59:21 2015

@author: pramod
"""
import numpy as numpy
import random
import glob
from scipy.misc import *

def LoadImages(directory, split):
   # get a list of all the picture filenames
   gifs = glob.glob(directory + '/*.gif')
   # uncomment the below line when trying an unknown file
   #extraGif = glob.glob("/media/cosmos/Data/College Notes/M.Tech/Semester 4/Statistical Methods in AI/Project - Face Recognition/2.gif")
   classMap = {}
   testGIF = []
   allLabels = []
   testLabels = []
   trainingGIF = []
   trainingLabels = []
   for i in range(len(gifs)):
      if random.random() < split:
         trainingGIF.append(gifs[i])
         l = gifs[i].split("/");
         labelName = l[len(l)-1].split(".")[0][-2:]
         trainingLabels.append(labelName)
         allLabels.append(labelName)
      else:
         testGIF.append(gifs[i])
         l = gifs[i].split("/");
         labelName = l[len(l)-1].split(".")[0][-2:]
         testLabels.append(labelName)
         allLabels.append(labelName)
   # uncomment the below 2 lines when trying an unknown file
   #testGIF.append(extraGif[0])
   #testLabels.append("un")
   #allLabels.append("un")
   trainingImgs = numpy.array([imread(i, True).flatten() for i in trainingGIF])
   testImgs = numpy.array([imread(i, True).flatten() for i in testGIF])
   # creating a list of class labels
   allLabels = set(allLabels)
   noOfClasses = len(allLabels)
   sortedLabels = []
   for i in allLabels:
      sortedLabels.append(i)
   sortedLabels = sorted(sortedLabels)
   # creating a mapping for confusion matrix
   j = 0
   for i in sortedLabels:
      classMap[i] = j
      j = j + 1
   return trainingImgs,testImgs,trainingLabels,testLabels,noOfClasses,classMap
   # mean
```

```
    mu = numpy.mean(data, 0)
    # mean adjust the data
    ma_data = data - mu
    # run SVD
    e_faces, sigma, v = numpy.linalg.svd(ma_data.transpose(), full_matrices=False)
    # compute weights for each image
    weights = numpy.dot(ma_data, e_faces)
    return e_faces, weights, mu
def PCA(data):
    # mean
    mu = numpy.mean(data, 0)
    # mean adjust the data
    ma_data = data - mu
    # run SVD
    e_faces, sigma, v = numpy.linalg.svd(ma_data.transpose(), full_matrices=False)
    # compute weights for each image
    weights = numpy.dot(ma_data, e_faces)
    return e_faces, weights, mu
def InputWeight(testData, mu, e_faces):
    ma_data = testData - mu
    weights = numpy.dot(ma_data, e_faces)
    return weights
def load_data():
    inDIR  = "/home/cosmos/CSStuff/SMAI/Project 2 - Classification/input/yalefaces"
    outDIR = "/home/cosmos/CSStuff/SMAI/Project 2 - Classification/output"
    imgDims = (243, 320)

    split = 0.8
    trainingData, testData, trainingLabels, testLabels, noOfClasses, classMap = LoadImages(inDIR, split)
    e_faces, trainingWeights, mu = PCA(trainingData)
    #print trainingWeights.shape[0]
    #print trainingWeights.shape
    #print mu.shape
    #print e_faces.shape
    '''
    tr_d, va_d, te_d = load_data()
    training_inputs = [np.reshape(x, (784, 1)) for x in tr_d[0]]
    print tr_d[0].shape
    training_results = [vectorized_result(y) for y in tr_d[1]]
    training_data = zip(training_inputs, training_results)
    validation_inputs = [np.reshape(x, (784, 1)) for x in va_d[0]]
    validation_data = zip(validation_inputs, va_d[1])
    test_inputs = [np.reshape(x, (784, 1)) for x in te_d[0]]
    test_data = zip(test_inputs, te_d[1])
    return (training_data, validation_data, test_data)
    '''
    testWeights=numpy.zeros((len(testData),trainingWeights.shape[1]))
    #print testWeights.shape
    #print trainingWeights.shape
    #print len(testWeights)
    for i in range(len(testWeights)):
        #testWeight =InputWeight(testData[i],mu,e_faces)
        testWeights[i]=InputWeight(testData[i],mu,e_faces)

    #formatting weights
        training_inputs = [numpy.reshape(x, (len(trainingWeights), 1)) for x in trainingWeights]
        test_inputs= [numpy.reshape(x,(len(trainingWeights) , 1)) for x in testWeights]
    #Convert training labels to vectors
    trainingLabels=numpy.asarray(trainingLabels)
    trainingLabels=trainingLabels.astype(numpy.float)
    testLabels=numpy.asarray(testLabels)
    testLabels=testLabels.astype(numpy.float)
    #decrementing subject labels
```

```
    for i in range(len(trainingLabels)):
        trainingLabels[i]=trainingLabels[i]-1
    for i in range(len(testLabels)):
        testLabels[i]=testLabels[i]-1
    #print testLabels
    #print vectorized_result(trainingLabels[0])
    #print trainingWeights.shape
    training_results = [vectorized_result(y) for y in trainingLabels]
    #test_results= [vectorized_result(y) for y in testLabels]

    tr_d=zip(training_inputs, training_results)
    te_d=zip(test_inputs,testLabels)
    #print te_d
    return (tr_d,te_d)
def vectorized_result(j):
    """Return a 10-dimensional unit vector with a 1.0 in the jth
    position and zeroes elsewhere.  This is used to convert a digit
    (0...9) into a corresponding desired output from the neural
    network."""
    e = numpy.zeros((15, 1))
    e[j-1] = 1.0
    return e
#load_data()



# -*- coding: utf-8 -*-
"""
Created on Thu Apr  9 22:59:21 2015

@author: pramod
"""
import numpy as numpy
import random
import glob
from scipy.misc import *

def LoadImages(directory, split):
    # get a list of all the picture filenames
    gifs = glob.glob(directory + '/*.gif')
    # uncomment the below line when trying an unknown file
    #extraGif = glob.glob("/media/cosmos/Data/College Notes/M.Tech/Semester 4/Statistical Methods in AI/Project -
Face Recognition/2.gif")
    classMap = {}
    testGIF = []
    allLabels = []
    testLabels = []
    trainingGIF = []
    trainingLabels = []
    for i in range(len(gifs)):
        if random.random() < split:
            trainingGIF.append(gifs[i])
            l = gifs[i].split("/");
            labelName = l[len(l)-1].split(".")[0][-2:]
            trainingLabels.append(labelName)
            allLabels.append(labelName)
        else:
            testGIF.append(gifs[i])
            l = gifs[i].split("/");
            labelName = l[len(l)-1].split(".")[0][-2:]
            testLabels.append(labelName)
            allLabels.append(labelName)
    # uncomment the below 2 lines when trying an unknown file
```

```
    #testGIF.append(extraGif[0])
    #testLabels.append("un")
    #allLabels.append("un")
    trainingImgs = numpy.array([imread(i, True).flatten() for i in trainingGIF])
    testImgs = numpy.array([imread(i, True).flatten() for i in testGIF])
    # creating a list of class labels
    allLabels = set(allLabels)
    noOfClasses = len(allLabels)
    sortedLabels = []
    for i in allLabels:
        sortedLabels.append(i)
    sortedLabels = sorted(sortedLabels)
    # creating a mapping for confusion matrix
    j = 0
    for i in sortedLabels:
        classMap[i] = j
        j = j + 1
    return trainingImgs,testImgs,trainingLabels,testLabels,noOfClasses,classMap
    # mean
    mu = numpy.mean(data, 0)
    # mean adjust the data
    ma_data = data - mu
    # run SVD
    e_faces, sigma, v = numpy.linalg.svd(ma_data.transpose(), full_matrices=False)
    # compute weights for each image
    weights = numpy.dot(ma_data, e_faces)
    return e_faces, weights, mu
def PCA(data):
    # mean
    mu = numpy.mean(data, 0)
    # mean adjust the data
    ma_data = data - mu
    # run SVD
    e_faces, sigma, v = numpy.linalg.svd(ma_data.transpose(), full_matrices=False)
    # compute weights for each image
    weights = numpy.dot(ma_data, e_faces)
    return e_faces, weights, mu
def InputWeight(testData, mu, e_faces):
    ma_data = testData - mu
    weights = numpy.dot(ma_data, e_faces)
    return weights
def load_data():
    inDIR  = "/home/cosmos/CSStuff/SMAI/Project 2 - Classification/input/yalefaces"
    outDIR = "/home/cosmos/CSStuff/SMAI/Project 2 - Classification/output"
    imgDims = (243, 320)

    split = 0.8
    trainingData, testData, trainingLabels, testLabels, noOfClasses, classMap = LoadImages(inDIR, split)
    e_faces, trainingWeights, mu = PCA(trainingData)
    #print trainingWeights.shape[0]
    #print trainingWeights.shape
    #print mu.shape
    #print e_faces.shape
    '''
    tr_d, va_d, te_d = load_data()
    training_inputs = [np.reshape(x, (784, 1)) for x in tr_d[0]]
    print tr_d[0].shape
    training_results = [vectorized_result(y) for y in tr_d[1]]
    training_data = zip(training_inputs, training_results)
    validation_inputs = [np.reshape(x, (784, 1)) for x in va_d[0]]
    validation_data = zip(validation_inputs, va_d[1])
    test_inputs = [np.reshape(x, (784, 1)) for x in te_d[0]]
    test_data = zip(test_inputs, te_d[1])
```

```
    return (training_data, validation_data, test_data)
    '''
    testWeights=numpy.zeros((len(testData),trainingWeights.shape[1]))
    #print testWeights.shape
    #print trainingWeights.shape
    #print len(testWeights)
    for i in range(len(testWeights)):
        #testWeight =InputWeight(testData[i],mu,e_faces)
        testWeights[i]=InputWeight(testData[i],mu,e_faces)

    #formatting weights
        training_inputs = [numpy.reshape(x, (len(trainingWeights), 1)) for x in trainingWeights]
        test_inputs= [numpy.reshape(x,(len(trainingWeights) , 1)) for x in testWeights]
    #Convert training labels to vectors
    trainingLabels=numpy.asarray(trainingLabels)
    trainingLabels=trainingLabels.astype(numpy.float)
    testLabels=numpy.asarray(testLabels)
    testLabels=testLabels.astype(numpy.float)
    #decrementing subject labels
    for i in range(len(trainingLabels)):
        trainingLabels[i]=trainingLabels[i]-1
    for i in range(len(testLabels)):
        testLabels[i]=testLabels[i]-1
    #print testLabels
    #print vectorized_result(trainingLabels[0])
    #print trainingWeights.shape
    training_results = [vectorized_result(y) for y in trainingLabels]
    #test_results= [vectorized_result(y) for y in testLabels]

    tr_d=zip(training_inputs, training_results)
    te_d=zip(test_inputs,testLabels)
    #print te_d
    return (tr_d,te_d)
def vectorized_result(j):
    """Return a 10-dimensional unit vector with a 1.0 in the jth
    position and zeroes elsewhere.  This is used to convert a digit
    (0...9) into a corresponding desired output from the neural
    network."""
    e = numpy.zeros((15, 1))
    e[j-1] = 1.0
    return e
#load_data()
```

Neural Network Eigen Utility Code:

```
import eigen_face_loader
trainingData, testData = eigen_face_loader.load_data()
import neuralnetwork
net = neuralnetwork.Network([len(trainingData), 200, 15])
net.SGD(trainingData, 50, 5, .09, 15, test_data = testData)
```

2. Results:

Epoch 0: 3 / 27
Epoch 1: 2 / 27
Epoch 2: 2 / 27
Epoch 3: 3 / 27
Epoch 4: 4 / 27
Epoch 5: 4 / 27
Epoch 6: 4 / 27
Epoch 7: 4 / 27
Epoch 8: 3 / 27

Epoch 9: 3 / 27
Epoch 10: 3 / 27
Epoch 11: 3 / 27
Epoch 12: 3 / 27
Epoch 13: 3 / 27
Epoch 14: 3 / 27
Epoch 15: 3 / 27
Epoch 16: 4 / 27
Epoch 17: 4 / 27
Epoch 18: 4 / 27
Epoch 19: 4 / 27
Epoch 20: 4 / 27
Epoch 21: 4 / 27
Epoch 22: 4 / 27
Epoch 23: 4 / 27
Epoch 24: 4 / 27
Epoch 25: 4 / 27
Epoch 26: 4 / 27
Epoch 27: 4 / 27
Epoch 28: 4 / 27
Epoch 29: 4 / 27
Epoch 30: 4 / 27
Epoch 31: 4 / 27
Epoch 32: 4 / 27
Epoch 33: 4 / 27
Epoch 34: 4 / 27
Epoch 35: 4 / 27
Epoch 36: 3 / 27
Epoch 37: 3 / 27
Epoch 38: 3 / 27
Epoch 39: 3 / 27
Epoch 40: 3 / 27
Epoch 41: 3 / 27
Epoch 42: 3 / 27
Epoch 43: 3 / 27
Epoch 44: 3 / 27
Epoch 45: 3 / 27
Epoch 46: 3 / 27
Epoch 47: 3 / 27
Epoch 48: 3 / 27
Epoch 49: 3 / 27
Confusion Matrix:
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Class 1

Precision : 0.0
Recall : 0.0
Specificity : 93.1034482759

Class 2

Precision : 0.0
Recall : 0.0
Specificity : 100.0

Class 3

Precision : 0.0
Recall : 0.0
Specificity : 96.4285714286

Class 4

Precision : 25.0
Recall : 33.3333333333
Specificity : 89.6551724138

Class 5

Precision : 0.0
Recall : 0.0
Specificity : 96.4285714286

Class 6

Precision : 0.0
Recall : 0.0
Specificity : 93.1034482759

Class 7

Precision : 33.3333333333
Recall : 33.3333333333
Specificity : 92.8571428571

Class 8

Precision : 0.0
Recall : 0.0
Specificity : 96.4285714286

Class 9

Precision : 50.0
Recall : 50.0
Specificity : 96.2962962963

Class 10

Precision : 0.0
Recall : 0.0
Specificity : 93.1034482759

Class 11

Precision : 0.0
Recall : 0.0
Specificity : 93.1034482759

Class 12

Precision : 0.0
Recall : 0.0
Specificity : 90.0

Class 13

Precision : 0.0
Recall : 0.0
Specificity : 96.4285714286

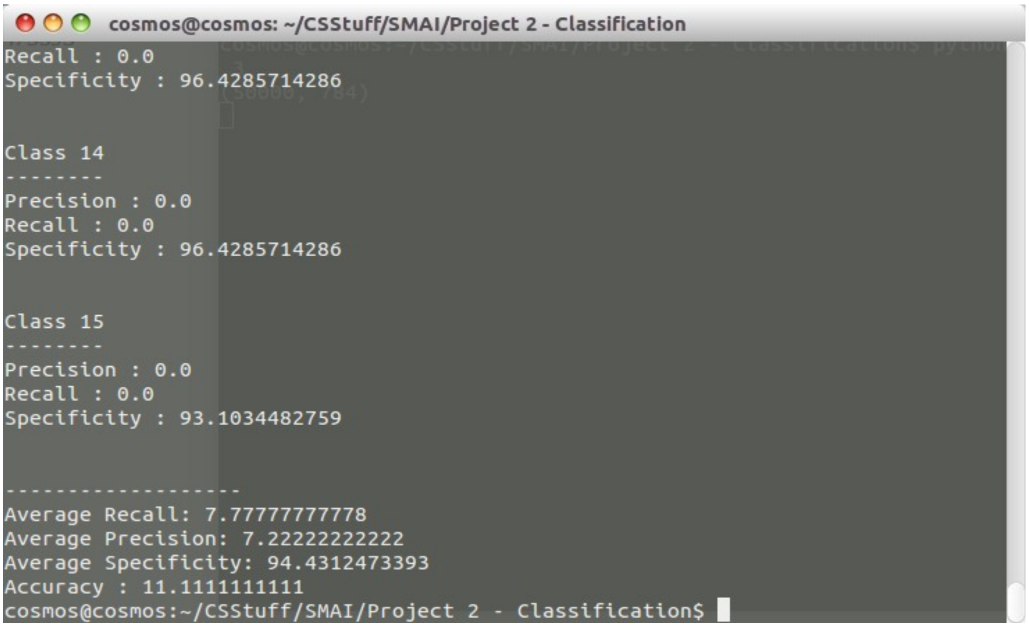
Class 14

Precision : 0.0
Recall : 0.0
Specificity : 96.4285714286

Class 15

Precision : 0.0
Recall : 0.0
Specificity : 93.1034482759

Average Recall: 7.77777777778
Average Precision: 7.22222222222
Average Specificity: 94.4312473393
Accuracy : 11.1111111111



For MNIST Dataset:

1. Souce Code

mnist_loader :

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr  9 21:09:06 2015

@author: pramod
"""

"""
mnist_loader
~~~~~~~~~~~~

A library to load the MNIST image data.  For details of the data
structures that are returned, see the doc strings for ``load_data``
and ``load_data_wrapper``.  In practice, ``load_data_wrapper`` is the
function usually called by our neural network code.
"""

#### Libraries
# Standard library
import cPickle
import gzip

# Third-party libraries
import numpy as np

def load_data():
    """Return the MNIST data as a tuple containing the training data,
    the validation data, and the test data.

    The ``training_data`` is returned as a tuple with two entries.
    The first entry contains the actual training images.  This is a
    numpy ndarray with 50,000 entries.  Each entry is, in turn, a
    numpy ndarray with 784 values, representing the 28 * 28 = 784
    pixels in a single MNIST image.

    The second entry in the ``training_data`` tuple is a numpy ndarray
    containing 50,000 entries.  Those entries are just the digit
    values (0...9) for the corresponding images contained in the first
    entry of the tuple.

    The ``validation_data`` and ``test_data`` are similar, except
    each contains only 10,000 images.

    This is a nice data format, but for use in neural networks it's
    helpful to modify the format of the ``training_data`` a little.
    That's done in the wrapper function ``load_data_wrapper()``, see
    below.
    """
    f = gzip.open('/home/cosmos/CSStuff/SMAI/Project 2 - Classification/input/mnist.pkl.gz', 'rb')
    training_data, validation_data, test_data = cPickle.load(f)
    f.close()
    return (training_data, validation_data, test_data)

def load_data_wrapper():
    """Return a tuple containing ``(training_data, validation_data,
    test_data)``. Based on ``load_data``, but the format is more
    convenient for use in our implementation of neural networks.
```

```
    In particular, ``training_data`` is a list containing 50,000
    2-tuples ``(x, y)``.  ``x`` is a 784-dimensional numpy.ndarray
    containing the input image.  ``y`` is a 10-dimensional
    numpy.ndarray representing the unit vector corresponding to the
    correct digit for ``x``.

    ``validation_data`` and ``test_data`` are lists containing 10,000
    2-tuples ``(x, y)``.  In each case, ``x`` is a 784-dimensional
    numpy.ndarry containing the input image, and ``y`` is the
    corresponding classification, i.e., the digit values (integers)
    corresponding to ``x``.

    Obviously, this means we're using slightly different formats for
    the training data and the validation / test data.  These formats
    turn out to be the most convenient for use in our neural network
    code."""
    tr_d, va_d, te_d = load_data()
    training_inputs = [np.reshape(x, (784, 1)) for x in tr_d[0]]

    training_results = [vectorized_result(y) for y in tr_d[1]]
    training_data = zip(training_inputs, training_results)
    validation_inputs = [np.reshape(x, (784, 1)) for x in va_d[0]]
    validation_data = zip(validation_inputs, va_d[1])
    test_inputs = [np.reshape(x, (784, 1)) for x in te_d[0]]
    test_data = zip(test_inputs, te_d[1])
    return (training_data, test_data)

def vectorized_result(j):
    """Return a 10-dimensional unit vector with a 1.0 in the jth
    position and zeroes elsewhere.  This is used to convert a digit
    (0...9) into a corresponding desired output from the neural
    network."""
    e = np.zeros((10, 1))
    e[j] = 1.0
    return e
```

MNIST Utility Function Source Code:

```
import mnist_loader
trainingData, test_data = mnist_loader.load_data_wrapper()
import copyCode
net = copyCode.Network([784, 30, 10])
net.SGD(trainingData, 30, 10, 3.0,  10, test_data)
```

2. Results

Epoch 0: 9104 / 10000
Epoch 1: 9211 / 10000
Epoch 2: 9262 / 10000
Epoch 3: 9321 / 10000
Epoch 4: 9358 / 10000
Epoch 5: 9370 / 10000
Epoch 6: 9375 / 10000
Epoch 7: 9395 / 10000
Epoch 8: 9390 / 10000
Epoch 9: 9399 / 10000

Confusion Matrix:
[932, 0, 5, 0, 1, 4, 6, 1, 1, 8]
[0, 1108, 0, 1, 2, 3, 1, 5, 1, 4]
[4, 3, 959, 12, 2, 2, 2, 27, 7, 1]
[5, 2, 9, 939, 2, 23, 4, 10, 12, 15]
[2, 0, 7, 1, 927, 3, 7, 9, 4, 27]
[7, 1, 5, 22, 2, 822, 9, 1, 9, 8]
[11, 7, 12, 3, 12, 7, 913, 1, 3, 0]
[1, 1, 12, 7, 2, 1, 0, 956, 7, 15]
[13, 13, 19, 19, 8, 25, 16, 6, 928, 16]
[5, 0, 4, 6, 24, 2, 0, 12, 2, 915]
Class 1

Precision : 95.1020408163
Recall : 97.2860125261
Specificity : 99.473453269

Class 2

Precision : 97.6211453744
Recall : 98.4888888889
Specificity : 99.6972754793

Class 3

Precision : 92.9263565891
Recall : 94.1118743867
Specificity : 99.1990344525

Class 4

Precision : 92.9702970297
Recall : 91.9686581783
Specificity : 99.2225142357

Class 5

Precision : 94.399185336
Recall : 93.9209726444
Specificity : 99.3974583699

Class 6

Precision : 92.1524663677
Recall : 92.776523702
Specificity : 99.2430795848

Class 7

Precision : 95.3027139875
Recall : 94.2208462332
Specificity : 99.5072273325

Class 8

Precision : 92.9961089494

Recall : 95.4091816367
Specificity : 99.2101799035

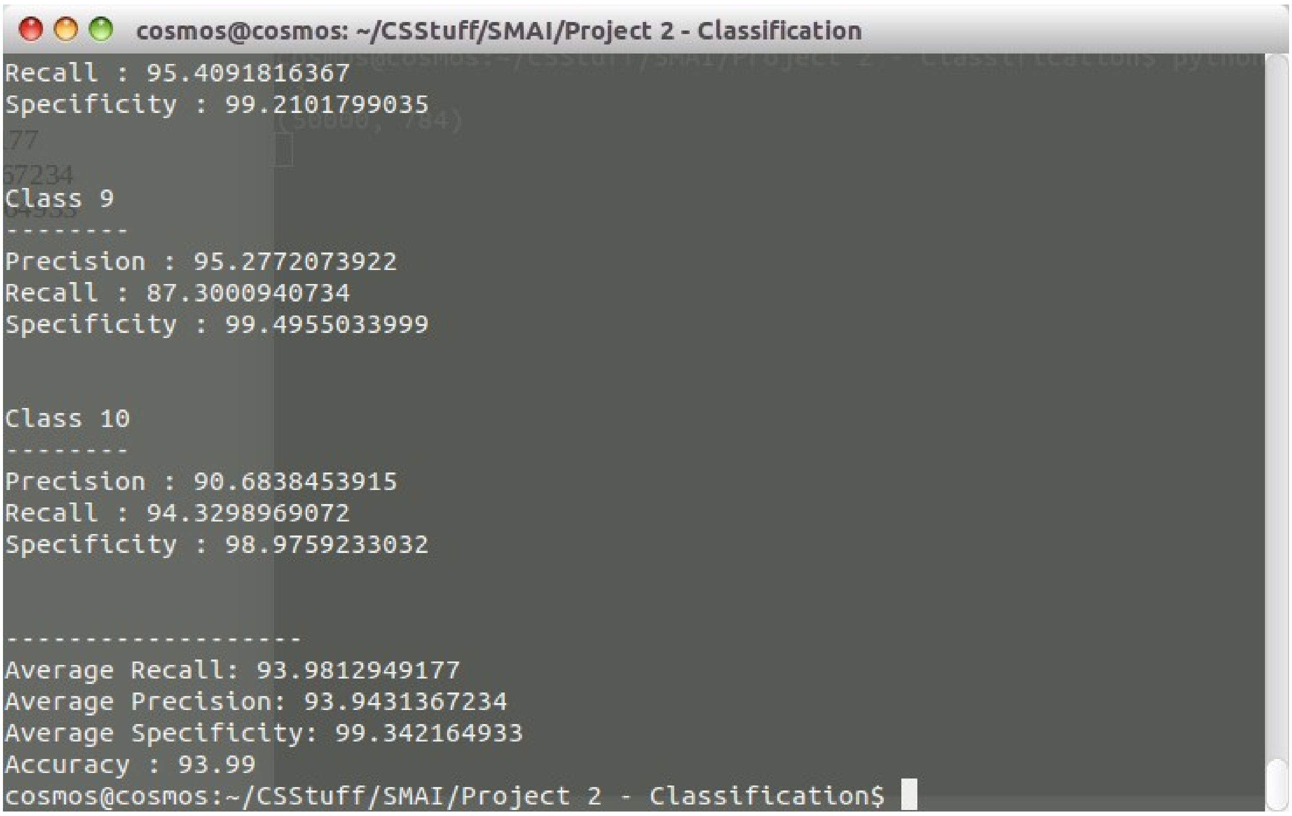
Class 9

Precision : 95.2772073922
Recall : 87.3000940734
Specificity : 99.4955033999

Class 10

Precision : 90.6838453915
Recall : 94.3298969072
Specificity : 98.9759233032

Average Recall: 93.9812949177
Average Precision: 93.9431367234
Average Specificity: 99.342164933
Accuracy : 93.99



Comparisons Between Different Methods:

Dataset / Methods	K- Nearest Neighbours	SVM	Multilayer Feedforwards Neural Network
Yale Dataset	86.1221321373 %	90.0948621554 %	11.11 %
MNIST Dataset	82.15 %	94.35 %	93.99%