**Introduction to Pattern Recognition-CSE555**

**PS1**

**1. The images are 28 x 28 pixels in gray-scale. The categories are 0, 1, ... 9. We concatenate the image rows into a 28 x 28 vector and treat this as our feature, and assume the feature vectors in each category in the training data "train-images-idx3-ubyte.gz") have Gaussian distribution. Draw the mean and standard deviation of those features for the 10 categories as 28 x 28 images using the training images ("train-images-idx3-ubyte.gz"). There should be 2 images for each of the 10 digits, one for mean and one for standard deviation. We call those "mean digits" and "standard deviation digits" in CSE455/555.**

**Approach**: Firstly we start by importing the MNIST dataset from tensor flow. Convert the into arrays using the array function from numpy package. Now , we have four arrays – Training image array, Training Labels array, Test images array, Test Labels array. We keep the one hot vector as false so that grouping the training images by their labels become feasible. We further group them according to their labels and find their mean and Standard deviation.

**Code :**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plot

from sklearn.metrics import zero\_one\_loss

from tensorflow.examples.tutorials.mnist import input\_data

from sklearn.naive\_bayes import GaussianNB

mnist\_ip = input\_data.read\_data\_sets('MNIST\_data', one\_hot=False)

training\_images = np.array(mnist\_ip.train.images)

'''

print(training\_images.shape)

'''

training\_labels = np.array(mnist\_ip.train.labels).reshape([-1,1])

test\_images = np.array(mnist\_ip.test.images)

test\_labels = np.array(mnist\_ip.test.labels).reshape([-1,1])

for i in range(training\_images.shape[0]):

plot.imshow(training\_images[i].reshape([28,28]), cmap = plot.get\_cmap('gray'));

training\_images2 = np.concatenate((training\_images, training\_labels), axis=1)

df\_training = pd.DataFrame(training\_images2)

training\_mean = df\_training.groupby(784).mean()

for i in range(training\_mean.shape[0]):

fig = plot.imshow(np.array(training\_mean[i:i]).reshape([28,28]));

plot.show()

plot.draw()

training\_sd = df\_training.groupby(784).std()

for i in range(training\_mean.shape[0]):

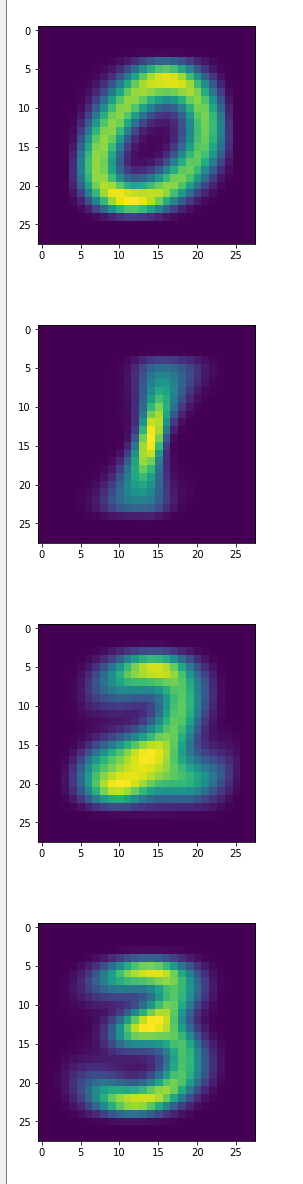
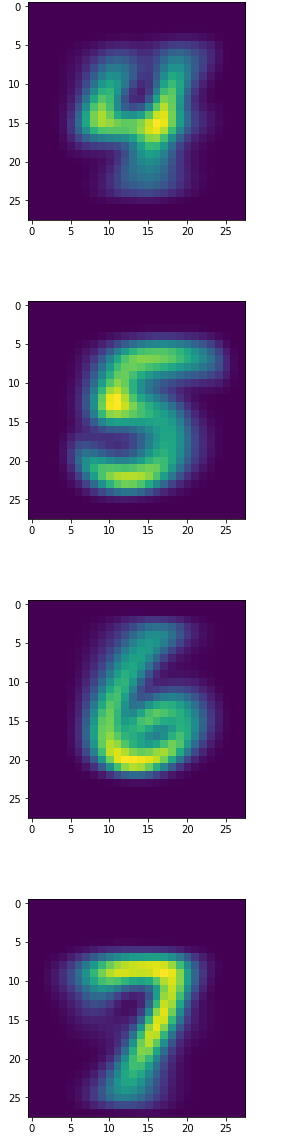
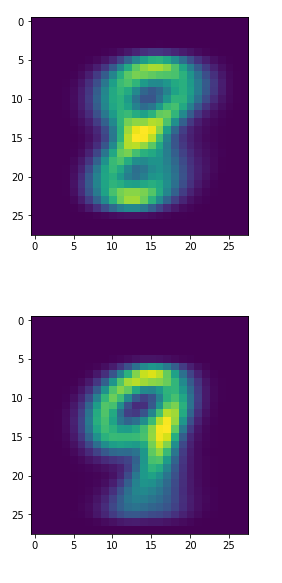
fig = plot.imshow(np.array(training\_sd[i:i]).reshape([28,28]));

plot.show()

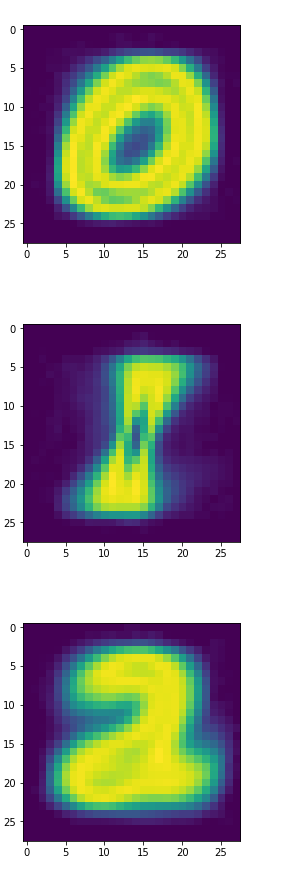
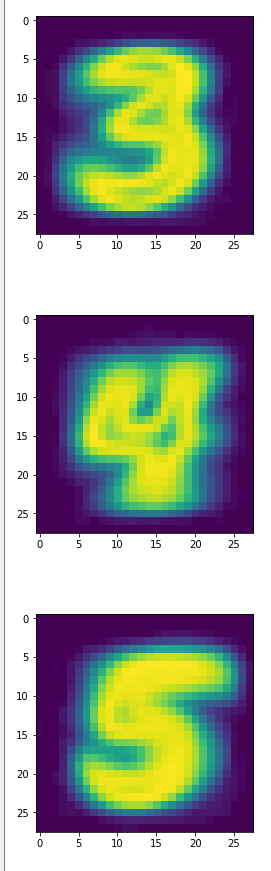
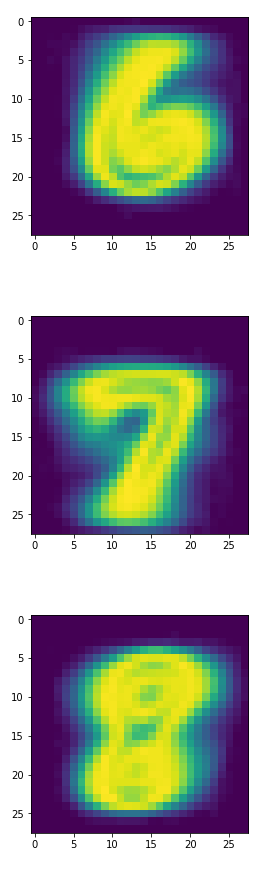
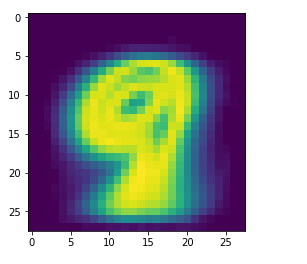
plot.draw()

**Output :**

**Mean:**

**Standard Deviation:**

**2. Classify the images in the testing data set ("t10k-images-idx3-ubyte.gz") using 0-1 loss function and Bayesian decision rule and report the performance. Why it doesn't perform as good as many other methods on LeCuns web page? Before coding the discriminant functions, review Section 2.6.**

**Approach**: Import GaussianNB from sklearn.naive\_bayes. Train the model by feeding the training input and training labels. Use the predict () function on the test images of MNIST dataset to predict a list of labels.

Find the probability of error by comparing the predicted array with the actual test labels. Traverse through each element. for every element that is different, we increment counter. Find the error and accuracy by dividing it with the total number of test labels.

**Code :**

#Question 2

#naive-bayes

model = GaussianNB()

model.fit(training\_images, training\_labels.ravel())

predicted= model.predict(test\_images)

'''

print(predicted.shape)

error = zero\_one\_loss(test\_labels,predicted)

print("error: %f" %(error) )

accuracy = 1 - error;

print("Probability of accuracy:%f" %(accuracy))

'''

counter = 0;

for i in range(0,10000):

if predicted[i] not in test\_labels[i]:

counter = counter + 1

P\_error = (counter/10000)

print("Probablity of Error: %f"%(P\_error))

P\_accuracy = 1 - P\_error

print("Accuracy rate : %f" %(P\_accuracy))

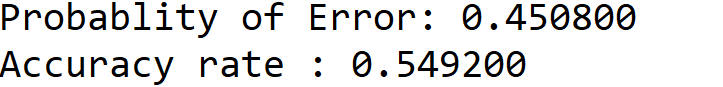
**Output :**

**Probablity of Error: 0.450800**

**Accuracy rate : 0.549200**

**Why it doesn't perform as good as many other methods on LeCuns web page?**

In this case of Gaussian Naïve Bayes, the probability of error and accuracy are as follows:



Error = 45 %

Accuracy = 54.9%

We can notice that the probability of error is much higher than with other classification techniques in LeCuns Web page.

For example: these classification techniques have a much lower rate.

|  |  |
| --- | --- |
| linear classifier (1-layer NN) | 12 |
| linear classifier (1-layer NN) | 8.4 |
| pairwise linear classifier | 7.6 |
| K-nearest-neighbors, Euclidean (L2) | 5 |
| K-nearest-neighbors, Euclidean (L2) | 3.09 |

This classification technique based on [Bayes’ Theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem) with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. In real life, it is almost impossible that we get a set of predictors which are completely independent.

* [Gaussian](http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB) NB tends to push probabilities to 0 or 1 . This is mainly because it makes the assumption that features are conditionally independent given the class, which is not the case in this dataset which contains 2 redundant features.
* Hence naive Bayes is also known as a bad estimator, so the probability outputs from predict\_proba are not to be taken too seriously.

**Source** : analyticsvidhya.com, scikit-learn.org, machinelearningmastery.com, LeCuns webpage.