Name : Abhik Dey

ASU ID : 1216907406

Project : Density Estimation and Classification

1. Introduction

In this project we are given MNIST dataset which contains 70000 images of handwritten digits, divided into 60,000 training images and 10,000 testing images. We have to use the digits 7 and 8 in the images to perform the parameter estimation and classification. Given dataset has the below details:

a. Number of samples in a training set: “7”:6265; “8”:5851
b. Number of samples in testing set: “7”:1028, “8”:974

Our goal is:

a. Extracting features and parameter estimation for the normal distribution of each digit using training data.
b. Using estimated distributions to do Naive Bayes Classification on testing data.
c. Using training data to train the model for Logistic Regression using gradient ascent.
d. Calculate the classification accuracy for both “7” and “8”

We will be using the algorithms for MLE Density estimation, Naive Bayes and Logistic Regression.

Language : Python
Constraint : Use of sklearn package is not allowed
Data Resource : mnist_data.mat files is provided.

2. Approach

The data is extracted from “mnist_data.mat” file. The file is the form of dictionary having the training data (trX), testing data(tsX), training Label (trY) and testing Label (tsY). In the training and testing data, each row represents a digit where as in the Label data, 0 and 1 represents 7 and 8 respectively. We will be describing each step below of how we are training the model with the training dataset and thereafter testing it with the testing data set.

2.1 Extraction of data from the data file (mnist.mat)

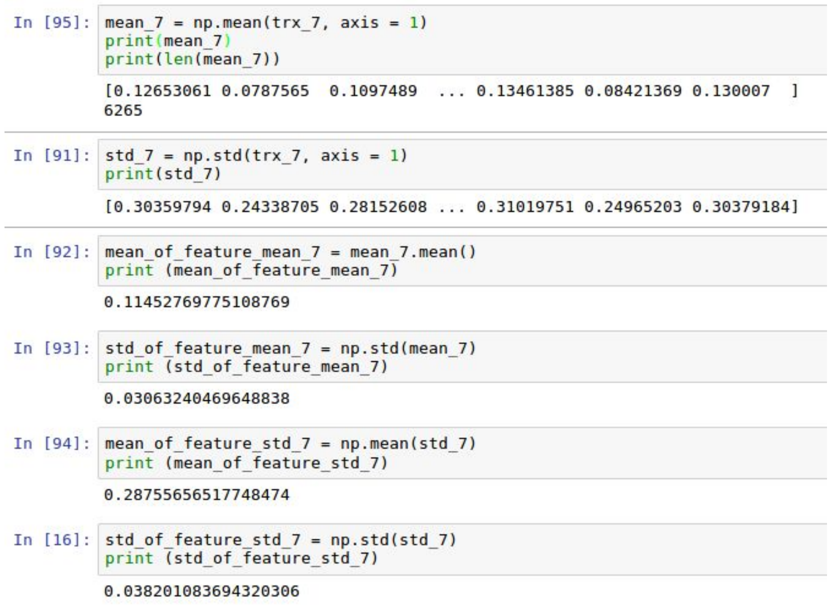
We are taking the whole content of the .mat file in a dictionary called “dataset” where we are separating the training data(trX) and testing data(tsX). The length of the training data is 12116 and each row in the training data represents a digit. Total 784 elements are present in each row of the training data.

2.2 Determine the mean and standard deviations of 7 and 8

2.2.1 Training Data

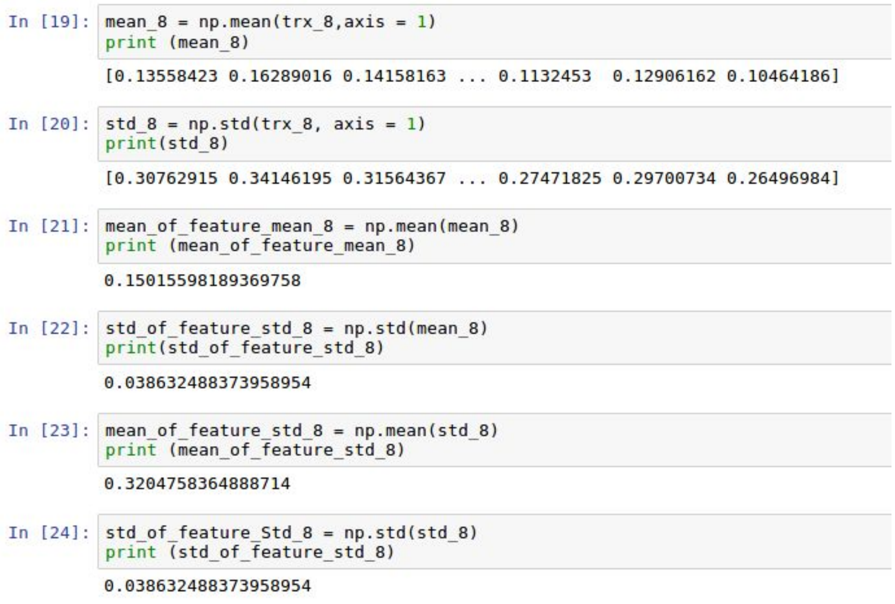
Now we will determine the mean and standard deviations of 7 and 8. We are using the numpy library mean, std and var functions to determine the mean, standard deviations respectively. Let us consider the data for image 7. The same steps will be performed for image 8 The data trx_7 is an array of dimensions 6265X784. When we take mean of trx_7, we will get another array of dimension (6265,) . Similar is the case for standard deviation. We take standard deviation of trx_7 and we get another array. Hence, we have two arrays mean_7 and std_7. From here we can say that the mean and standard deviations are the two features of class 7. Similar will be the case for image 8. Now we will have to determine the mean and standard deviation of feature mean and standard deviation for class 7. We will get 4 values , each for mean and standard deviation and then we plot a graph to see the normal distribution.

For class 7:



```
In [95]: mean_7 = np.mean(trx_7, axis = 1)
         print(mean_7)
         print(len(mean_7))

         [0.12653061 0.0787565  0.1097489  ... 0.13461385 0.08421369 0.130007  ]
         6265

In [91]: std_7 = np.std(trx_7, axis = 1)
         print(std_7)

         [0.30359794 0.24338705 0.28152608 ... 0.31019751 0.24965203 0.30379184]

In [92]: mean_of_feature_mean_7 = mean_7.mean()
         print (mean_of_feature_mean_7)

         0.11452769775108769

In [93]: std_of_feature_mean_7 = np.std(mean_7)
         print (std_of_feature_mean_7)

         0.03063240469648838

In [94]: mean_of_feature_std_7 = np.mean(std_7)
         print (mean_of_feature_std_7)

         0.28755656517748474

In [16]: std_of_feature_std_7 = np.std(std_7)
         print (std_of_feature_std_7)

         0.038201083694320306
```

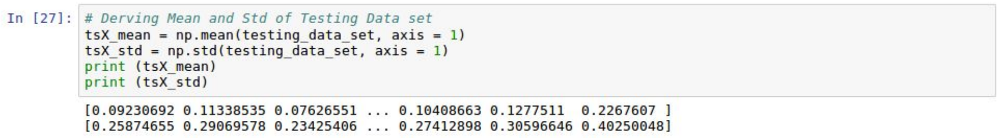
For class 8:



```
In [19]: mean_8 = np.mean(trx_8,axis = 1)
         print (mean_8)

         [0.13558423 0.16289016 0.14158163 ... 0.1132453  0.12906162 0.10464186]

In [20]: std_8 = np.std(trx_8, axis = 1)
         print(std_8)

         [0.30762915 0.34146195 0.31564367 ... 0.27471825 0.29700734 0.26496984]

In [21]: mean_of_feature_mean_8 = np.mean(mean_8)
         print (mean_of_feature_mean_8)

         0.15015598189369758

In [22]: std_of_feature_std_8 = np.std(mean_8)
         print(std_of_feature_std_8)

         0.038632488373958954

In [23]: mean_of_feature_std_8 = np.mean(std_8)
         print (mean_of_feature_std_8)

         0.3204758364888714

In [24]: std_of_feature_Std_8 = np.std(std_8)
         print (std_of_feature_std_8)

         0.038632488373958954
```

2.2.2 Testing Data

For testing data, we derive just the mean, standard deviation of the data set.



```
In [27]: # Derving Mean and Std of Testing Data set
         tsX_mean = np.mean(testing_data_set, axis = 1)
         tsX_std = np.std(testing_data_set, axis = 1)
         print (tsX_mean)
         print (tsX_std)

         [0.09230692 0.11338535 0.07626551 ... 0.10408663 0.1277511  0.2267607 ]
         [0.25874655 0.29069578 0.23425406 ... 0.27412898 0.30596646 0.40250048]
```

2.3 Estimating Naive Bayes Classifier

Naive Bayes Classifier is a probabilistic classifier based on Bayes theorem where the features of the class are independent of each other. According to Bayes theorem, conditional probability is defined as :

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

For multiple features, the above equation can be written as

$p(C_k|x) = (\prod_{i=1}^{n} p(x_i|C) \times p(C_k))/p(x)$

This can be also written as:

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

Therefore according to Naive Bayes Classifier, we can write the above equation as:

Posterior ∝ prior x likelihood

In the project we will calculate the prior and likelihood to determine the posterior value using the testing data set. For this the below steps were followed.

2.3.1 Determining prior

Prior is determined by dividing the number of samples of a class to the total number of the samples. Therefore prior for class y is given by:

prior(y) = $Number\ of\ samples\ of\ class\ y \div Total\ number\ of\ samples\ in\ the\ set$

here, for y = 7 we have,

prior(7) = 6265/12116

and, for y = 8

prior(8) = 5851/12116

```
p_prior_7 = len(trx_7)/len(training_data_set)
```

```
p_prior_8 = len(trx_8)/len(training_data_set)
```

2.3.2 Determining Likelihood:

To determine likelihood, we need to find out the value for P(features|class), here its $P(mean_7,std_7|7)$ and $P(mean_8,std_8|8)$. To determine $P(mean_7,std_7|7)$, we determine the probability distribution function for $P(mean_7|7)$ and $P(std_7|7)$ and multiply both. Here we are taking the value of X as the mean/std of the testing dataset.

```
In [28]: def p_x_given_y(X, mean, var):
             p = (1/(np.sqrt(2*np.pi*var)))*np.exp(-(X-mean)**2/(2*var))
             return p
```

```
p_likelihood_7 = p_x_given_y(tsX_mean,mean_of_feature_mean_7,mean_7.var())*\
                 p_x_given_y(tsX_std, mean_of_feature_std_7,std_7.var())
```

```
p_likelihood_8 = p_x_given_y(tsX_mean,mean_of_feature_mean_8,mean_8.var())*\
                 p_x_given_y(tsX_std, mean_of_feature_std_8,std_8.var())
```

2.3.3 Determining the posterior

Posterior is the product of prior and likelihood.

```
p_posterior_7 = p_prior_7*p_likelihood_7
```

```
p_posterior_8 = p_prior_8*p_likelihood_8
```

2.3.4 Compare between Posterior of class 7 and 8

We compare between the posterior values of class 7 and 8. In our case we compare with 8 and 7. Based on comparison, output will be an array comprising of True and False. We will convert True and False to integers 0 and 1. 0 will be for false, 1 will be for True. Next we'll compare this output array with tsY dataset and find the number of nonzero values present.

2.3.5 Calculate Accuracy

Based on the output above, we will get the count of the number of non-zero values present in the dataset. We will then calculate the % over the whole dataset with this value. In my case the classification accuracy is 69.5%

2.3.6 Conclusion

From the above, we can conclude that by training model with 12116 training data, it can classify 69.5% of the digits in the testing dataset as 7 or 8 using Naive Bayes classifier.

2.4 Estimating Logistic Regression

In Logistic Regression, for a given set of inputs, we try them to assign them to 0 or 1. LR predicts the probability to assign the inputs in one of the two categories. We will be using the sigmoid function to determine the probability.

2.4.1 Sigmoid function

Sigmoid function is defined as

$g(z) = 1/(1+e^{-z})$

```
def sigmoid_func(self, z):
    return 1 / (1 + np.exp(-z))
```

2.4.2 Loss function

To determine the weights/parameters for the function, Loss function is calculated by

(-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

```
def loss(self, h, y):
    return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()
```

2.4.3 Gradient Ascent

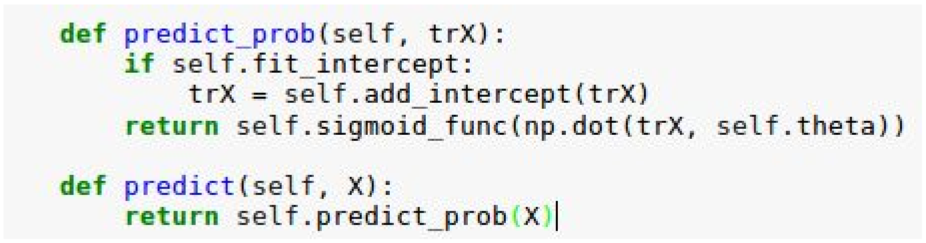
To determine the gradient ascent, we calculate as below:
gradient = np.dot(X.T, (h - y)) / y.shape[0]
theta = theta - learningRate * gradient

Where learningRate = 0.01

```
def sigmoid_func(self, z):
    return 1 / (1 + np.exp(-z))

def loss(self, h, y):
    return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()
```

2.4.4 Predictions

To find the probability that the input belongs to a particular class, we use the sigmoid function to calculate.



```
    def predict_prob(self, trX):
        if self.fit_intercept:
            trX = self.add_intercept(trX)
        return self.sigmoid_func(np.dot(trX, self.theta))

    def predict(self, X):
        return self.predict_prob(X)
```

2.4.5 Calculate Accuracy

Based on the output above, we will get the count of the number of non-zero values present in the dataset. We will then calculate the % over the whole dataset with this value. In my case the classification accuracy is 68.7%

2.4.6 Conclusion

From the above, we can conclude that by training model with 12116 training data, it can classify 68.7% of the digits in the testing dataset as 7 or 8 using Logistic Regression.