DENSITY ESTIMATION AND CLASSIFICATION

Francis Mendoza

ASUID: 1213055998

CSE575- Statistical Machine Learning

fmendoz7@asu.edu

1. INTRODUCTION

a. I extracted a subset of the MNIST dataset, particularly dealing with the digits "7" and "8". In this project, I used MLE Density Estimation, Naïve Bayes classification and Logistic Regression to find the average of all pixel values in the image, as well as the standard deviation of all pixel values in the image. The entire file was first developed on the Jupyter Notebook environment and then converted into a .py file using the following command

   i. `jupyter nbconvert --to script Mendoza-DRAFTFORPY_Project1.ipynb`

2. FEATURE EXTRACTION

a. The extracted dataset contains training and testing samples for "7" and "8", stored as a dictionary.
   i. The key is a string literal, denoting training or testing and which digit
   ii. The value is a list of lists, denoting the brightness output of pixels

b. Training Samples:
   i. "7": 6265 rows
   ii. "8": 5851 rows

c. Test Samples:
   i. "7": 1028
   ii. "8": 974

3. NAÏVE BAYES CLASSIFICATION

a. Although professor, in the announcements, mentioned an implementation using multivariate Gaussian distribution utilizing a mean vector and covariance, I utilized the still viable approach of disjointed 1D Gaussians that have the mean of means, mean of standard deviation, standard deviation of means and standard deviation of standard deviation. I have verified that this is a valid approach by going to Mr. Kevin Ding's office hours on Tuesday and Wednesday (2/18-2/19) at 10:00-11:00 hrs MST.
   i. The formula I used for the Seven Naïve Bayes was as follows
      1. $P(Y_7|\mu,\sigma) = P(y_7)P(\mu|y_7)P(\sigma|y_7)$

$$= \frac{samples\ for\ 7}{total} * \left(\frac{1}{\sigma_{\mu,7}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x_{\mu,7}-\mu_{\mu,7}}{\sigma_{\mu,7}}\right)^2}\right) * \left(\frac{1}{\sigma_{\sigma,7}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x_{\sigma,7}-\mu_{\sigma,7}}{\sigma_{\sigma,7}}\right)^2}\right)$$

ii. The formula I used for the Eight Naïve Bayes was as follows
    1. $P(Y_8|\mu,\sigma) = P(y_8)P(\mu|y_8)P(\sigma|y_8)$

$$= \frac{samples\ for\ 8}{total} * \left(\frac{1}{\sigma_{\mu,8}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x_{\mu,8}-\mu_{\mu,8}}{\sigma_{\mu,8}}\right)^2}\right) * \left(\frac{1}{\sigma_{\sigma,8}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x_{\sigma,8}-\mu_{\sigma,8}}{\sigma_{\sigma,8}}\right)^2}\right)$$

iii. To determine the parameters (which were derived from the features), I took mean of the subarrays in the first coordinate spot and the standard deviation of the subarrays in the second coordinate spot to have a general input parameter vector consisting of the mean and standard deviation features. This returned a list of lists from the training set with 12116 rows and two columns.

iv. I then split them according to whether the respective samples would lead to a classification of "7" or "8". Finally, I would then either take the mean or standard deviation of the entire column to get these values (for both "7" and "8" class)
    1. Mean of Means
    2. Mean of Standard Deviation
    3. Standard Deviation of Mean
    4. Standard Deviation of Standard Deviations

v. I then took the higher value from each respective Naïve Bayes formula in order to determine the class

b. The results from the initial feature extraction (calculating the means of both the means and the standard deviations) follows as such:

TRAINING SET

- $Mean\ Of\ Seven\ Gaussian$: 0.015856038875645444
- $STD\ Of\ Seven\ Gaussian$: 0.11294710989818092
- $Mean\ Of\ Eight\ Gaussian$: 1.4463005533833693
- $STD\ Of\ Eight\ Gaussian$: 1.2139208115141336
- $Prior\ Of\ Seven\ Gaussian$: 0.5170848464839881
- $Prior\ Of\ Eight\ Gaussian$: 0.4829151535160119

c. As a result of comparing the values returned from Bayesian formulas (by determining which is higher) for the class being a "7" or an "8", I found that more samples leaned towards "7" than "8". This is also reflected in the priors.

d. The distribution is used to classify a test sample by getting n number of Gaussian distributions (depending on how many features you are measuring, in this case two: one for the mean feature and one for the standard deviation feature)

e. Due to the results of the Naïve Bayes algorithm, I was able to find that the accuracy value is roughly 69%

```
Mean Of Seven Gaussian:  0.015856038875645444
STD Of Seven Gaussian:  0.11294710989818092

Mean Of Eight Gaussian:  1.4463005533833693
STD Of Eight Gaussian:  1.2139208115141336

Prior Of Seven Gaussian:  0.5170848464839881
Prior Of Eight Gaussian:  0.4829151535160119
```

In [25]:
```
#Accuracy function

temp3 = tsY_array.tolist()
tsY_list = temp3[0]

accuracyCounter = 0

for i in range(len(tsY_list)):
    if tsY_list[i] == score[i]:
        accuracyCounter += 1

accuracyValue = accuracyCounter/len(tsY_list)
print("Accuracy Is: ", accuracyValue)
```

```
Accuracy Is:  0.6953046953046953
```

4. LOGISTIC REGRESSION

a. Although the logistic regression code was implemented, due to the complexity and my current lack of understanding on advanced data manipulation into numpy, I was unable to find the decision boundary using gradient descent. Because of this, this code block was commented out to get the rest of the file to compile properly.

b. Essentially, logistic regression works by calculating a probability value between 0 and 1 of how likely a value from belongs to a certain class. This is accomplished by using a logistic function called a sigmoid, which classifies given an input.

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

- e is a constant representing Euler's number, or 2.718281828
- x_0 represents the sigmoid's midpoint on the x axis
- L represents the maximum value
- k represents the slope of steepness of a curve

Within the scope of logistic regression, the logistic function looks like this

$$z = weight^t x$$

$$sigmoid(z) = \frac{1}{1 + e^{-z}}$$

c. The logistic function essentially uses gradient ascent to update the weight values for each iteration (called an epoch), while an optimization function such as gradient descent is used to minimize the loss for each epoch

d. Unfortunately, while I was able to extract the features, have a general method to calculate the accuracy value (nested under Naïve Bayes for now) and have the code for Naïve Bayes, I was not able to complete Logistic Regression with Gradient Ascent to adjust the weights, due to my travels to the 2020 Stanford Blockchain Summit. I did however, list some important functions necessary to get started.

5. RESULTS

a. Although there are slightly more samples in the testing and training data for "7" than "8", using Naïve Bayes yields an accuracy of roughly ~69%.

b. Although I was unable to finish the logistic regression code, if it was completed in time, it essentially would've returned a decision boundary (with conditionals to determine under what class a value from the testing set would fall under), with gradient descent used to minimize loss while gradient ascent is used to adjust the weights for each epoch.