Final Project

Face and Digit Classification

Author:
Brian Y. Mahesh A.
brianyi14@gmail.com
ma1700@scarletmail.rutgers.edu

Supervisors:
Abdeslam Boularias
Aravind S.

CS520: Artificial Intelligence
Rutgers University

April 27, 2020

1 Import and Data Processing

For this project, we used NumPy and Pandas modules to import, process, and transform the raw image data into usable information for model training. Below is an example of the the raw digit image data along with the converted image using our convert() method. We chose to equate ‘#’ and ‘+’ pixels with a value of 1 while any empty pixels were assigned a value of 0. We chose to store most of our data in Panda’s Series and DataFrame data structures along with Python’s native list structure.

```
array([['                            ', '0000000000000000000000000000'],
       ['                            ', '0000000000000000000000000000'],
       ['                            ', '0000000000000000000000000000'],
       ['                            ', '0000000000000000000000000000'],
       ['                            ', '0000000000000000000000000000'],
       ['                +++++##+    ', '0000000000000000111111110000'],
       ['        +++++######+###+    ', '0000000011111111111111110000'],
       ['       +##########+++++     ', '0000000111111111111111100000'],
       ['        #######+##          ', '0000000011111111110000000000'],
       ['        +++###  ++          ', '0000000011111100110000000000'],
       ['           +#+              ', '0000000000011100000000000000'],
       ['           +#+              ', '0000000000011100000000000000'],
       ['            +#+             ', '0000000000001110000000000000'],
       ['            +##++           ', '0000000000001111100000000000'],
       ['             +###++         ', '0000000000000111111000000000'],
       ['              ++##++        ', '0000000000000011111100000000'],
       ['                +##+        ', '0000000000000000111100000000'],
       ['                 ###+       ', '0000000000000000011110000000'],
       ['              +++###        ', '0000000000000011111100000000'],
       ['            ++#####+        ', '0000000000001111111100000000'],
       ['          ++######+         ', '0000000000111111111000000000'],
       ['        ++######+           ', '0000000011111111100000000000'],
       ['       +######+             ', '0000000111111110000000000000'],
       ['    ++######+               ', '0000111111111000000000000000'],
       ['    +####++                 ', '0000111111100000000000000000'],
       ['                            ', '0000000000000000000000000000'],
       ['                            ', '0000000000000000000000000000'],
       ['                            ', '0000000000000000000000000000']],
      dtype=object)
```

2 Naive Bayes Model

2.1 Algorithm

For our Naive Bayes model, we predict the classification of an image through the following Naive Bayes assumption: $P(Class|Data) = \frac{P(Data|Class)P(Class)}{P(Data)}$. For each observation, we calculate $P(Class|Data)$ for each possible class, and choose the class with the highest probability as our prediction. Since $P(Data)$ is a constant, we can ignore it and focus on evaluating $P(Data|Class)P(Class)$ when estimating for $P(Class|Data)$. $P(Class)$ is the probability that a certain class appears in a data set and $P(Data|Class)$ is the probability that a certain feature value appears within a certain class. Since we are calculating $P(Data|Class)$ through a discrete methodology, there may be many values for a particular feature that do not appear even once in a class; this causes $P(Data|Class) = 0$ and messes up our estimation of $P(Class|Data)$. As such, we employ a smoothing method over our model parameters such that any $P(Data|Class)$ that equal zero are assigned a very small value of 1×10^{-10}. We use the same training and testing method for both digit and face classification since our method is robust in feature input and adjusts the feature selection algorithm accordingly.

2.2 Method

We use four methods to train and test our Naive Bayes model, partition(), feature_ext(), train_nb(), and test_nb(). Partition() serves to divide a single image of 0s and 1s and divides the image into the respective feature count and returns an array of features for that particular image. Feature_ext() takes these arrays of features returned from partition() and aggregates how often each feature value appears for each feature. Feature_ext() returns a dataframe where the columns represent each unique feature and the rows represent the total count of feature values. At that point, it is a simple matter to calculate P(Data | Class) from the dataframe by dividing each feature value aggregate count by the number of observations in that class. After we have P(Class) and P(Data | Class), we simply take the product for each class and find the highest product and return that as our prediction.

2.3 Digit Classification

We will first look at our model's performance on digit classification for digits of 0 through 9. Our feature selection is obtained by dividing a digit image into n x n dimension partitions such that the sum of colored pixels (1s) within a partition is considered as one feature. Since each digit image is 28 x 28, we considered dividing the image into 7 x 7 partitions for a total of 16 features. Unfortunately, using 100% of the training set, we got an accuracy of 63.8%. As such, we decided to narrow the size of each feature to a 4 x 4 for a total of 49 features. By creating smaller partitions, our features become more precise in capturing the nuances that differ between each digit. This turned out to be the better choice, and increased our model's overall accuracy from 63.8% to 74.7%. The following dataframe shows the accuracy and time it takes to train and test our model for various divisions of our training set. As you can see from the graph below, the accuracy of the model steadily improves the greater the percentage of the training set is used.

Table 2.1: Accuracy

Training	Accuracy (%)	Time (sec)
10%	58.0	2.44
20%	70.0	2.69
30%	60.0	3.41
40%	64.0	3.27
50%	64.0	3.52
60%	62.0	3.69
70%	62.0	3.79
80%	64.0	4.08
90%	64.0	4.22
100%	64.0	4.36

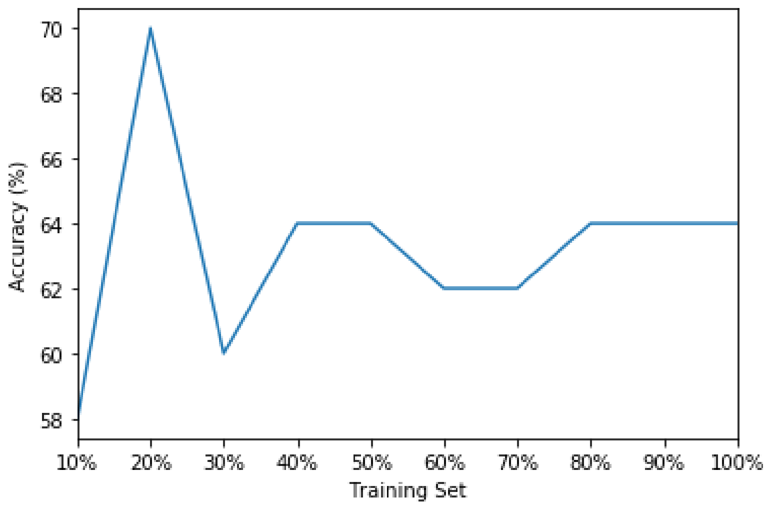

Figure 2.1: Accuracy

2.3.1 Face Classification

3 Digit Classification - KNN

4 Face Classification

5 Old Methods