# CS520 Final Project

Digit and Face Classification

**Deepak Vagish K**
dk1156
**Suraj K A**
skk140
**Shirisha Sudhakar Rao**
ss4313

December 9, 2022

## 1 Description

In this project, we implemented classifiers for detecting faces and classifying digits. The three classifiers were Naive Bayes, Perceptron and Neural Network. We referred the Berkeley's CS188 course for a description of the classifiers

## 2 Data Processing and Command

After reading and processing the data for classification, we have facial data containing 451 and 150 train and test images respectively. On the other hand we have 5000 training and 1000 test images for digits. The training data points were picked randomly in groups of 10%, 20%, 30%,...100% to train and we computed the accuracy in each case and tabulated.
The following commands are used to run the respective models. To run on the face dataset replace the term digit with face

```
python naiveBayes.py --data_type digit
python knn.py --data_type digit
python perceptron.py --data_type digit
```

## 3 Perceptron

Perceptron is a binary classifier where it stores a weight vector for each class. Multiplying the weight vector with the feature vector will return a set of scores, where the classification is based on the maximum score.

### 3.1 Algorithm

**Algorithm 1** Perceptron

```
Require: $f$ = training features of each image and $y$ = training labels
Ensure: weight vectors for each class
1: procedure PERCEPTRON
2:     For each training dataset $i$ having feature vector $f_i$, get the score for
       each label using the formula
           $$score(y) = \sum_{i=1}^{n} f_i w_y \quad (1)$$
3:     $y' = \text{maximum(score)}$
4:     if $y' \neq y_i$ then
5:         $w_y = w_y + f$
6:         $w_{y'} = w_{y'} - f$
```

### 3.2 Results

| Digit | | | | |
|---|---|---|---|---|
| Percentage | Training Data | Time(s) | Accuracy | Standard Deviation |
| 10% | 500 | 0.70 | 70.5 | 8.54 |
| 20% | 1000 | 1.24 | 73.7 | 5.92 |
| 30% | 1500 | 1.75 | 76.8 | 3.88 |
| 40% | 2000 | 2.19 | 78.0 | 2.6 |
| 50% | 2500 | 2.28 | 74.6 | 2.2 |
| 60% | 3000 | 2.35 | 79.4 | 1.86 |
| 70% | 3500 | 2.80 | 81.0 | 1.7 |
| 80% | 4000 | 3.13 | 81.4 | 1.47 |
| 90% | 4500 | 3.52 | 77.1 | 1.16 |
| 100% | 5000 | 3.76 | 79.8 | 1.1 |

| Face | | | | |
|---|---|---|---|---|
| Percentage | Training Data | Time(s) | Accuracy | Standard Deviation |
| 10% | 45 | 0.12 | 70.4 | 9.13 |
| 20% | 90 | 0.15 | 77.6 | 7.12 |
| 30% | 135 | 0.15 | 79.6 | 5.35 |
| 40% | 180 | 0.16 | 81.87 | 4.91 |
| 50% | 225 | 0.19 | 81.46 | 4.5 |
| 60% | 270 | 0.23 | 79.2 | 3.27 |
| 70% | 315 | 0.27 | 82.53 | 2.76 |
| 80% | 360 | 0.3 | 82.93 | 2.69 |
| 90% | 405 | 0.34 | 84.53 | 2.59 |
| 100% | 451 | 0.37 | 85.6 | 1.19 |

## 4 Naive Bayes

A Naive Bayes classifier uses a probabilistic approach for classification namely Bayes Theorem. Bayes theorem is given by the following formula:

$$P(C_i|D) = \frac{P(C_i)P(D|C_i)}{P(D)}$$

We use this to compute probabilities for each of the features of the given image. These probabilities are stored in a 2x2 matrix. This matrix is then used to predict the correct label for a given test image.

### 4.1 Features

Since the image consists of only black and white pixels, we re-constructed the image as a two dimensional array of 0's and 1's with 0 representing white pixels and 1 representing black pixels. This array is then divided into multiple smaller arrays. Each of these smaller array is considered a feature. For digits, we have broken down an image of size 28x28 into 49 features of size 4x4. Similarly, for faces, we have broken down an image of size 70x60 into 100 features of size 7x6 each.

## 4.2 Training

Training phase involves computing percentage of pixels present in each image and then consolidating this over the entire set of images. We first computed percentage of pixels in each feature of a given image and associated them with their respective label by storing them in a dictionary. Probabilities of a particular feature having a particular percentage are then computed over the set of images associated with a specific label. Counts of images associated with a label are also computed and stored in a dictionary.

## 4.3 Testing

Testing phase involve calculating the chance for an image to be associated to a particular label. The label with the highest chance is picked and the image is predicted to be associated with that label. The test image is first divided into features and the percentage of pixels present is computed. These percentages are then used to compute the likelihood of the image being related to a particular label. We use conditional probability to do that:

$$P(y = label|X) = \frac{P(X|y = label)P(y = label)}{P(X)}$$

X is the test image and y is a possible label of X. In the above formula, the following can be computed as:

$$P(y = label) = \frac{\text{number of training data with label y=label}}{\text{total training data}}$$

$$P(X|y = label) = P(f_1(X)|y = label) * P(f_2(X)|y = label) * ... * P(f_n(X)|y = label)$$

where f represents corresponding feature of the image. The above conditional probability is calculated for all the set of labels possible for a particular test class. The label with highest probability is then assigned to the test image.

### 4.4 Results

| Digit | | | | |
|---|---|---|---|---|
| Percentage | Training Data | Time(s) | Accuracy | Standard Deviation |
| 10% | 500 | 0.70 | 79.8 | 1.5 |
| 20% | 1000 | 1.41 | 79.6 | 1.2 |
| 30% | 1500 | 2.07 | 79.7 | 1.3 |
| 40% | 2000 | 2.79 | 78.2 | 0.5 |
| 50% | 2500 | 3.40 | 79.6 | 0.2 |
| 60% | 3000 | 4.21 | 79.5 | 0.1 |
| 70% | 3500 | 4.80 | 79.3 | 0.1 |
| 80% | 4000 | 5.43 | 79.1 | 0.3 |
| 90% | 4500 | 6.13 | 79.5 | 0.04 |
| 100% | 5000 | 6.94 | 79 | 0.0 |

| Face | | | | |
|---|---|---|---|---|
| Percentage | Training Data | Time(s) | Accuracy | Standard Deviation |
| 10% | 45 | 0.24 | 81.8 | 3.6 |
| 20% | 90 | 0.47 | 75.4 | 3.0 |
| 30% | 135 | 0.75 | 77.2 | 3.7 |
| 40% | 180 | 0.94 | 80.1 | 1.8 |
| 50% | 225 | 1.24 | 79.0 | 0.9 |
| 60% | 270 | 1.43 | 79.4 | 1.7 |
| 70% | 315 | 1.77 | 80.6 | 1.5 |
| 80% | 360 | 2.01 | 80.5 | 0.4 |
| 90% | 405 | 2.11 | 81.1 | 0.4 |
| 100% | 451 | 2.49 | 82.6 | 0.0 |

## 5 KNN

The k-nearest neighbors (KNN) algorithm is a data classification method for estimating the likelihood that a data point will become a member of one group or another based on what group the data points nearest to it belong to.

### 5.1 Features

Since the image consists of only black and white pixels, we re-constructed the image as a two dimensional array of 0's and 1's with 0 representing white pixels and 1 representing black pixels. This array is then flattened out to provide as input to the KNN model. Therefore, the input given to the KNN model during fitting is of the below dimensions :

1. (5000, 784) for digit dataset
2. (750, 4200) for digit dataset

## 5.2 Training

Training phase of KNN involves storing all the data points. It also arranges the data points in order to find the closest neighbors efficiently during the inference phase.

## 5.3 Testing

Testing phase involve calculating the distance from the test point to the points from the training phase to classify each point in the test dataset. It involved taking the k closest values and finding the most frequent label to assign to the test data point.

After trial and error, these were the best values of k :

1. k = 3 for digit dataset
2. k = 11 for digit dataset

After trial and error, these were the best distance metrics :

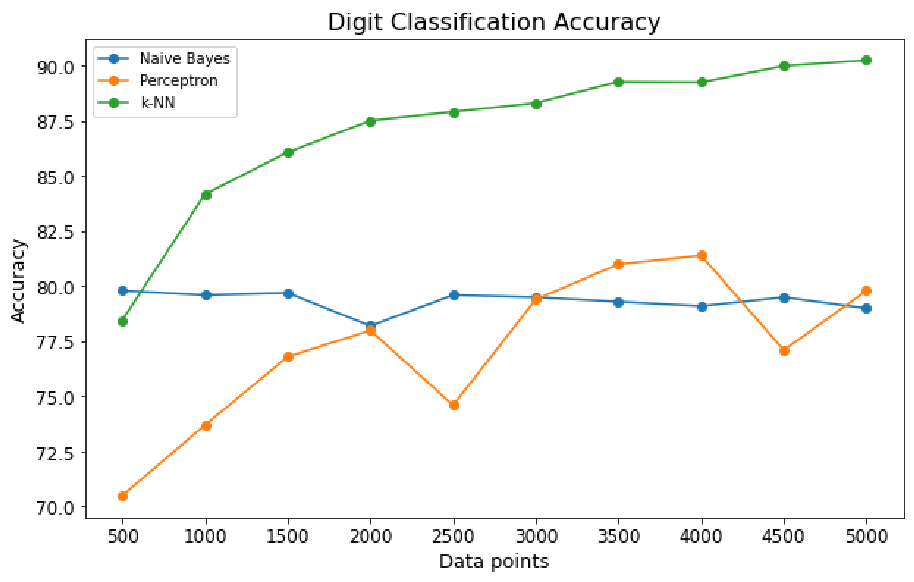
1. Euclidian distance for digit dataset
2. Braycurtis dissimilarity for digit dataset

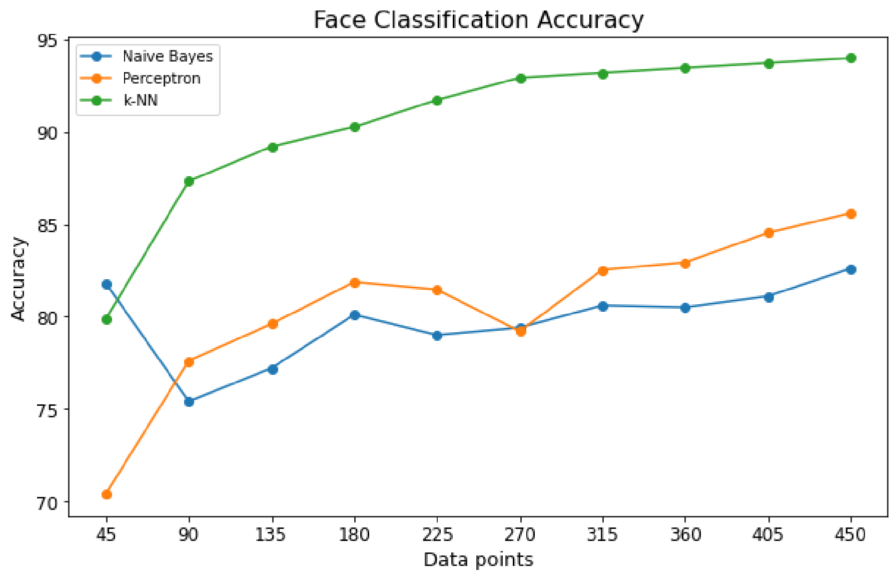
## 5.4 Results

| Digit | | | | |
|---|---|---|---|---|
| Percentage | Training Data | Time(s) | Accuracy | Standard Deviation |
| 10% | 500 | 0.00035 | 78.46 | 1.20 |
| 20% | 1000 | 0.00077 | 84.18 | 0.98 |
| 30% | 1500 | 0.00098 | 86.08 | 0.86 |
| 40% | 2000 | 0.00112 | 87.52 | 0.93 |
| 50% | 2500 | 0.00129 | 87.92 | 0.61 |
| 60% | 3000 | 0.00150 | 88.32 | 0.35 |
| 70% | 3500 | 0.00170 | 89.28 | 0.48 |
| 80% | 4000 | 0.00229 | 89.26 | 0.42 |
| 90% | 4500 | 0.00281 | 90.00 | 0.55 |
| 100% | 5000 | 0.00267 | 90.26 | 0.18 |

| Face | | | | |
|---|---|---|---|---|
| Percentage | Training Data | Time(s) | Accuracy | Standard Deviation |
| 10% | 75 | 0.00025 | 79.87 | 5.58 |
| 20% | 150 | 0.00024 | 87.33 | 3.88 |
| 30% | 225 | 0.00049 | 89.2 | 2.37 |
| 40% | 300 | 0.00078 | 90.27 | 1.80 |
| 50% | 375 | 0.00102 | 91.73 | 0.76 |
| 60% | 450 | 0.00120 | 92.93 | 1.86 |
| 70% | 525 | 0.00132 | 93.2 | 0.87 |
| 80% | 600 | 0.00137 | 93.47 | 1.28 |
| 90% | 675 | 0.00135 | 93.73 | 0.36 |
| 100% | 750 | 0.00156 | 94.00 | 0.00 |

## 6 Graphs





## 7 Inference

For the above models implemented above the time taken for the model to get trained is proportional to the number of data points. The standard deviation as shown is inversely proportional to the number of data points. From the above tables we can infer that a well trained model doesn't necessarily need to have been trained on more datasets. A model can potentially perform better than another even if the former is trained on lesser data points possibly due to overfitting. It is more dependent on factors such as the distribution of the data, balanced data points, etc. Therefore we can conclude that quality of data matters over the quantity of data.