RUTGERS

THE STATE UNIVERSITY
OF NEW JERSEY

CS520 Final Project

Digit and Face Classification

Avinash Ramachandran
AR2041
Sreeniketh Aathreya
SPA44
Darshan Senthil
DS1992

1 Description

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

2 Data Processing

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.

3 Naive Bayes

In Bayes the feature we used was pixel occupancy per region. So first, the algorithm finds the maximum pixel occupancy across all features and all pictures for that training set. With that maximum it sets up a matrix for each unique data point (faces/not faces, 1,2,3 etc). Each row corresponds to a feature and each column is a number between 0 and max. Next, the algorithm tallies and plots the feature and pixel data received from the training data in their corresponding matrix. So for example, if the face picture had 5 pixels in region 3. In the face matrix it would add a +1 to row 3 column 5. Then at the end it would divide each cell by the total number of pictures to come up with percents for each feature and pixel combination. Lastly, when you test it the algorithm will take the features from the testing image and look up the associated percent across all matrices. for each matrix it will keep a score by multiplying each feature percent by each other as given by the testing image feature (each matrix score is independent from each other).The highest percent out of all the matrices will be chosen as the guess. 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.

3.1 Features

In this algorithm, the features we used for Naive Bayes digits classifier is a set of pixel features. For each pixel j, it can take either value 0, which means that the pixel is white. Or, it can take a value of 1, which means that the pixel is black/grey. We return a dictionary containing the coordinate of pixel as a key

of either 1 or 0 as values. 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.

3.2 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.

3.3 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.

3.4 Results

The following are the results for both digits and faces for 10%, 20%...100% of the data using Naive Bayes classifier. We have sampled the data 5 times for each percentage set and calculated the mean and standard deviation over the accuracies.

Digit				
Percentage	Training Data	Time(s)	Accuracy	Standard Deviation
10%	500	0.23	76.6	2.073
20%	1000	0.45	78.8	1.923
30%	1500	0.67	79.2	1.923
40%	2000	0.89	78.2	1.095
50%	2500	1.11	78.8	1.303
60%	3000	1.34	78.2	0.836
70%	3500	1.56	79.4	0.547
80%	4000	1.79	79.2	0.447
90%	4500	2.04	78.6	0.547
100%	5000	2.27	79	0.0
Face				
Percentage	Training Data	Time(s)	Accuracy	Standard Deviation
10%	45	0.1	68.2	8.043
20%	90	0.203	73.8	6.379
30%	135	0.303	79.6	2.966
40%	180	0.407	85.6	2.607
50%	225	0.509	83.8	3.117
60%	270	0.61	83.0	1.582
70%	315	0.719	87.6	1.816
80%	360	0.825	85.8	2.774
90%	405	0.928	87.8	1.911
100%	450	1.07	88.6	0.447

4 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.

4.1 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

$$score(y) = \sum_{i=1}^{n} f_i w_y \tag{1}$$

3: $y' =$ maximum(score)
4: **if** $y' \neq y_i$ **then**
5: $w_y = w_y + f$
6: $w_{y'} = w_{y'} - f$

4.2 Features

- This algorithm begins with initializing random weights for each feature, which we defined to be the pixel occupancy per region. We had 100 features, because we split the image up into a 10x10 grid. This, each image has 100 weights. The weight vector is calculated by adding up the multiplication of each weight by its corresponding feature pixel density. For faces, if this resulting number is positive, then the prediction for the image is that it is a face. If it is negative, then the prediction is not a face. The algorithm checks the corresponding training label to validate the correctness of the prediction. If the prediction was correct, the algorithm moves on to a new image and calculates the weight vector again. If the prediction was too high (predicted face and it was not face), then each weight for that image is decremented by its corresponding feature. If the prediction was too low, (predicted not face and it was face), then each weight is incremented by its corresponding feature. Our algorithm continues to loop through the training data and update the weights until it hits a threshold of 85% correct guesses.

- The only difference between training faces and digits is that for each image, the algorithm calculates the weight vector for the digits 1-9, and

chooses the highest one for its prediction. If the prediction was incorrect, the weights for the incorrect predicted digit are decreased by their respective features, and the weights for the correct digit are increased by their respective features. This process takes quite a long time due to the number of calculations that must be made at each iteration.

4.3 Results

Digit				
Percentage	Training Data	Time(s)	Accuracy	Standard Deviation
10%	500	4.001	75.0	2.549
20%	1000	8.004	76.4	4.560
30%	1500	11.9	77.2	2.774
40%	2000	15.8	77.6	2.880
50%	2500	19.9	79.2	3.11
60%	3000	21.26	80.6	5.272
70%	3500	23.9	79.0	4.301
80%	4000	27.9	78.3	2.302
90%	4500	31.8	79.2	4.207
100%	5000	35.8	83.2	3.420
Face				
Percentage	Training Data	Time(s)	Accuracy	Standard Deviation
10%	45	0.463	72.4	4.669
20%	90	0.908	70.4	9.964
30%	135	1.369	76.0	8.660
40%	180	1.804	81.2	3.033
50%	225	2.26	82.2	2.167
60%	270	2.671	82.2	4.969
70%	315	3.087	82.8	2.588
80%	360	3.33	85	2.00
90%	405	3.95	82.3	2.507
100%	450	4.46	83.0	2.549

5 K Nearest Neighbor

5.1 Features

This algorithm works by taking the euclidean distance of the features in the testing image against all of the features of the images in the training data. So for example image 1 of the testing data would be compared against all 5000 images in the training data by finding the euclidean distance for each paring. Then all of that data is saved into an array and sorted from lowest to highest. After that the first 3 cells are chosen to vote. the most frequently seen within those 3 cells is chosen for the guess. This process repeats for each image in the training set.

5.2 Results

Digit				
Percentage	Training Data	Time(s)	Accuracy	Standard Deviation
10%	500	0.093	62.4	2.880
20%	1000	0.193	66.40	3.507
30%	1500	0.286	67.8	4.658
40%	2000	0.373	70.8	1.303
50%	2500	0.467	70.4	1.816
60%	3000	0.557	71.0	4.847
70%	3500	0.673	72.2	0.836
80%	4000	0.782	72.2	1.303
90%	4500	0.936	72.6	1.673
100%	5000	1.005	74.4	1.673
Face				
Percentage	Training Data	Time(s)	Accuracy	Standard Deviation
10%	45	0.029	69.2	5.630
20%	90	0.057	76.8	3.962
30%	135	0.092	76.4	2.701
40%	180	0.114	77.6	4.159
50%	225	0.143	78.8	2.167
60%	270	0.176	79.4	3.507
70%	315	0.216	80.2	2.280
80%	360	0.235	79.6	1.516
90%	405	0.271	82.3	1.207
100%	450	0.29	81.8	0.447

6 Graph

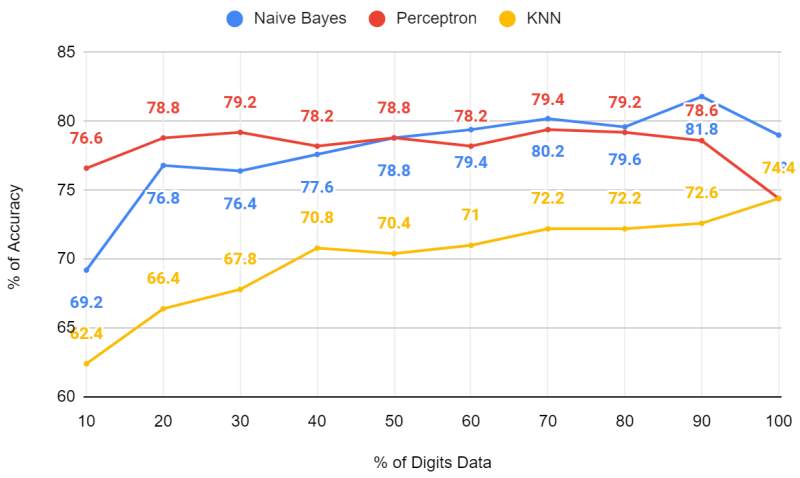

Figure 1: Digit Classification

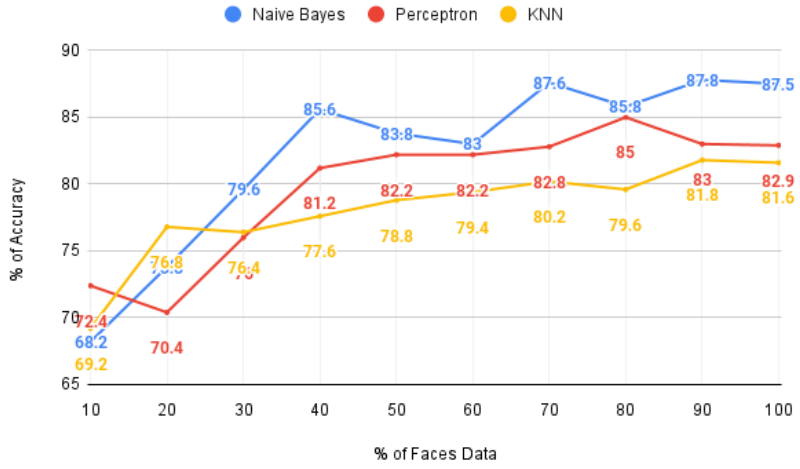

Figure 2: Face Classification

7 Conclusion

From the tables, it is evident that a model that is well trained doesn't necessarily have to be trained using more datasets. Owing to the issue of overfitting, a model can perform better even if it is trained using lesser data points. It is more dependent on factors such as the distribution of the data, balanced data points, etc. The time taken for the model to get trained is directly proportional to the number of data points.