CS 440 Assignment2

Anhelina Mahdzyar (am1904), Boyang Fu (bf249), Yao Shi (ys517)

April 29, 2018

1 Overview

Acknowledgement: This project is based on the one created by Dan Klein and John DeNero that was given as part of the programming assignments of Berkeley's CS188 course.

In this project, we designed two classifiers: a Naive Bayes classifier and a Perceptron classifier. Using these classifiers, we will mainly perform two tasks: optical character recognition(OCR) and face detection. There are two data sets: a set of scanned handwritten digit images and a set of face images in which edges have already been detected. We will design and extract features from the given image files using for both classifiers. We will start with 10% of the data points for training, and increase the training set by 10% each time until we can use 100% of the data points for training and testing. After we finish implementing these two classifiers, we will compare their performances and discuss the results.

2 Implementation

Feature enhance for digit: We extract the features in three ways and combine them together: Use all the basic features extraction, that is, denote black and white as features for each pixel. For example, feature (x, y) = 1 means pixel (x, y) is non-white feature, while pixel (x, y) = 0 is white feature.

1. Since the grey pixels ('+') are more likely to capture the outline of the digits, we calculate the total gray pixels for each row and the total pixel for each row. For example, suppose a variable grey store the number of grey pixels. Then (DIGIT_DATUM_WIDTH , y, 2, grey) = 1 means row y has 'grey' number of grey pixels. Similarly, (x, DIGIT_DATUM_HEIGHT, 2, grey) = 1 means column x has 'grey' number of grey pixels.

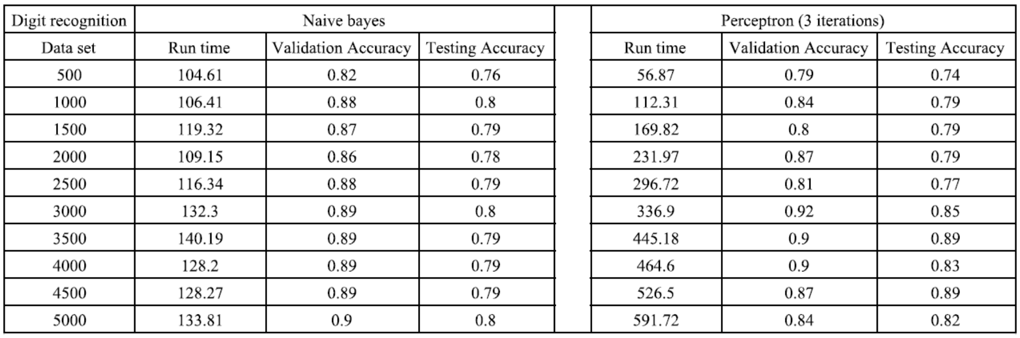
2. Since a digit may end up taking only a small area of the whole image, all the white-space peripheral to the digit can be considered as useless features. Therefore, we trimmed the peripheral of the digit first. After that, we want to find a way to calculate the black pixel and white pixel variation of the image. For example, suppose row 5 has 10 black pixels and row 6 has 12 black pixels, we define the feature to be ('black', 'row', 5, 6, 1) = 1.

3. Basically, the feature format is ('color', 'row(column)', i, i + 1, num), where color can be 'black' or 'white'; 'row' means the feature record the row feature difference between i and (i + 1) row, where 'column' means the feature records the column feature difference between column i and column (i + 1). Num can only take 5 values: -2, -1, 0, 1, 2. '-2' means the ith row(column) has more than 3 additional black(white) pixels than i+1th row(column). '-1' means the ith row(column) has 1-3 additional black(white) pixels than i+1th row(column). '0' means the ith row(column) has the same black(white) column as the (i+1) th row(column). Similar rules apply for the positive value of Num, where i th row(column) has less black(white) pixels than the (i + 1) th row(column).

3 Testing

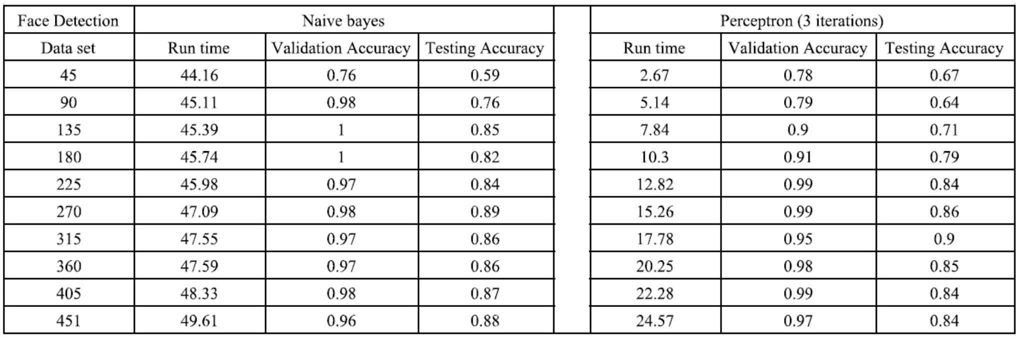
For the Digit recognition using the Naive Bayes algorithm, the accuracy and run time both generally increase as we increase the training data points. The correctness of validating data points reaches 90%, and the correctness of testing data points reaches 80%, as we finish training 100% of the data point set.

For the Digit recognition using the Perceptron algorithm, the accuracy and run time both generally increase as we increase the training data points. The correctness of validating data points reaches 84%, and the correctness of testing data points reaches 82%, as we finish training 100% of the data point set.



Digit recognition	Naive bayes				Perceptron (3 iterations)		
Data set	Run time	Validation Accuracy	Testing Accuracy		Run time	Validation Accuracy	Testing Accuracy
500	104.61	0.82	0.76		56.87	0.79	0.74
1000	106.41	0.88	0.8		112.31	0.84	0.79
1500	119.32	0.87	0.79		169.82	0.8	0.79
2000	109.15	0.86	0.78		231.97	0.87	0.79
2500	116.34	0.88	0.79		296.72	0.81	0.77
3000	132.3	0.89	0.8		336.9	0.92	0.85
3500	140.19	0.89	0.79		445.18	0.9	0.89
4000	128.2	0.89	0.79		464.6	0.9	0.83
4500	128.27	0.89	0.79		526.5	0.87	0.89
5000	133.81	0.9	0.8		591.72	0.84	0.82

Figure 1: Results from the Digit recognition



Face Detection	Naive bayes				Perceptron (3 iterations)		
Data set	Run time	Validation Accuracy	Testing Accuracy		Run time	Validation Accuracy	Testing Accuracy
45	44.16	0.76	0.59		2.67	0.78	0.67
90	45.11	0.98	0.76		5.14	0.79	0.64
135	45.39	1	0.85		7.84	0.9	0.71
180	45.74	1	0.82		10.3	0.91	0.79
225	45.98	0.97	0.84		12.82	0.99	0.84
270	47.09	0.98	0.89		15.26	0.99	0.86
315	47.55	0.97	0.86		17.78	0.95	0.9
360	47.59	0.97	0.86		20.25	0.98	0.85
405	48.33	0.98	0.87		22.28	0.99	0.84
451	49.61	0.96	0.88		24.57	0.97	0.84

Figure 2: Results from the Face detection

For the Face recognition using the Naive Bayes algorithm, the accuracy and run time both generally increase as we increase the training data points. The correctness of validating data points reaches 96%, and the correctness of testing data points reaches 88%, as we finish training 100% of the data point set.

For the Face recognition using the Perceptron algorithm, the accuracy and run time both generally increase as we increase the training data points. The correctness of validating data points reaches 97%, and the correctness of testing data points reaches 84%, as we finish training 100% of the data point set.

If the size of our training data set is small, the Perceptron algorithm is faster than the Naive Bayes algorithm, however, the Perceptron algorithm will be much slower if the training data size becomes relatively big, especially if we increase the number of iterations. We believe the accuracy of the Perceptron algorithm is strongly dependent on the number of iterations we set in some range for training the same number of data points. For example, using perceptron to train 500 data points using 3 iteration gives 75% validation accuracy and 75% testing accuracy; with 5 iterations it gives us 82% validation accuracy and 75% testing accuracy; with 10 iterations it gives us 83% validation accuracy and 80% testing accuracy; with 20 iterations, however, the accuracy of both validation and testing don't increase anymore. In addition, the Perceptron algorithm gives us a clear increase of accuracy when we increase the training data points. Comparing with the Perceptron algorithm, the accuracy of the Naive Bayes algorithm barely increases when we increase the sample size.

4 Conclusion

Implementing the algorithms is not the hardest part of this project, however, finding good features to extract is very critical in this project. In our case, having more features does not always indicate better outcomes, and does improve the accuracy of the classifiers to some degree. Comparatively, our implementations do not give us good results when our training set is small. When we gradually increase the size of the data set, the recognition/detection accuracy can reach a good level (around 80%-90%) in general. We surprisingly found out that the Perceptron algorithm gives us decent results for both the digit recognition and face detection, and the Perceptron does a better job than the Naive Bayes in general. During our testing process, we only used 3 iterations for the sake of time; if we process training with more iterations or with more training data set, we will result in better accuracy. In addition, face detection generally produces better results, and it may be because the face detection only allows two outcomes, either “True” or “False”, which means that it has less of a chance to “make mistakes” compared with digit recognition.