CS 440 Assignment2

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 use a fixed number of 100 samples for testing. After we finish implementing these two classifiers, we will compare their performances and discuss the results.

2 Implementation

2.1 Scheme 1:

During the first try, 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 records 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. For example, '-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).

Though the method above indeed improved the overall accuracy for both digit recognition and face recognition, the above feature extraction method was too time consuming. Running 5000 training data with the Perceptron model and doing 3 iteration would take more than 5 minutes. Therefore, we switched to another scheme.

2.2 Scheme 2:

During the second try, we focused on reducing the time complexity of the feature extraction process. We assume the most time consuming part in scheme 1 is the row/column feature difference comparison part since we have to run four separate loops to detect: white color difference in row, white color difference in column, black color difference in row, black color difference in column.

1. To reduce the loop number, we noticed that different digits have significant difference in length-width ratio. As a result, we first trimmed the peripheral white spaces, then calculate the width and height of each digit. We define width as the the difference between the last column that contain at least one non-white pixel and the first column that contain at least one non-white pixel, similar to the height. After that, we calculate the $ratio = height \div width$, define $ratio < 1.2$ as 'fat' feature, $1.2 <= ratio < 2$ as 'mid' feature and $ratio >= 2$ as 'fat' feature.

2. We notice the ratio can't explain everything; for example, if the digit '1' incline a little bit, the ratio would change drastically. Also, the ratios of '3' and '4' are indistinguishable. However, if we can find a way to grasp the outline of the digit, then we can enhance our accuracy. Fortunately, the 'grey' pixels nicely capture such features. Thus, we calculated the number of grey pixel in each row and each column in the format: features[(row_num, 'row', grey_num)] = 1 and features[(column_num, 'column', grey_num)] = 1. Getting the black pixel number, as experiment showed, won't have noticeable improvement to the accuracy but will slow down the training time.

3. Since there is no 'grey' pixel in face image, so for the faces feature extraction part, we first trim the image, then calculate the black pixel number of each row and each column and get decent results.

3 Testing

1. First, we want to roughly compare the performance of scheme 1 and scheme 2, and we recorded the following table:

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 applying scheme 1

Digit recognition	Naive bayes				Perceptron (3 iterations)		
Data set	Run time	Validation Accuracy	Testing Accuracy		Run time	Validation Accuracy	Testing Accuracy
500	55.81	0.85	0.77		16.57	0.75	0.72
1000	47.57	0.86	0.83		33.17	0.8	0.84
1500	48.58	0.87	0.82		50.33	0.82	0.78
2000	44.97	0.87	0.8		67.61	0.76	0.79
2500	42.9+	0.87	0.82		91.37	0.87	0.85
3000	48.4	0.85	0.82		146.52	0.9	0.85
3500	52.43	0.84	0.81		145.43	0.9	0.82
4000	45.85	0.85	0.79		162.37	0.9	0.84
4500	49.9	0.85	0.79		183.96	0.87	0.87
5000	60.84	0.87	0.79		208.97	0.86	0.85

Figure 2: Results from the Digit recognition applying scheme 2

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 3: Results from the Face recognition applying scheme 1

Face Detection	Naive bayes				Perceptron (3 iterations)		
Data set	Run time	Validation Accuracy	Testing Accuracy		Run time	Validation Accuracy	Testing Accuracy
45	40.92	0.8	0.75		1.95	0.66	0.53
90	39.85	0.98	0.82		3.88	0.97	0.83
135	42.13	1	0.86		5.57	0.9	0.75
180	41.69	1	0.89		7.56	0.94	0.78
225	39.56	0.99	0.89		9.35	0.97	0.88
270	44.42	0.98	0.88		11.15	0.99	0.89
315	42.89	0.98	0.89		12.69	0.99	0.9
360	44.22	0.99	0.89		14.59	1	0.9
405	43.87	0.98	0.9		16.16	0.97	0.89
451	43.75	0.98	0.91		18.01	0.96	0.9

Figure 4: Results from the Face detection applying scheme 2

2. 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 87%, and the correctness of testing data points reaches 79%, as we finish training 100% of the data point set applying scheme 2.

3. 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 86%, and the correctness of testing data points reaches 85%, as we finish training 100% of the data point set applying scheme 2.

4. 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 98%, and the correctness of testing data points reaches 91%, as we finish training 100% of the data point set applying scheme 2.

5. 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 96%, and the correctness of testing data points reaches 90%, as we finish training 100% of the data point set applying scheme 2.

6. Since Scheme 2 have more advantage in both speed and accuracy, we decided to use scheme 2 for further analysis. For the next part, we use only 10% of the data points that are reserved for training, then 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and finally 100%. For each percentage group, we randomly chose the training data every time and do the testing. We operated such procedure for 5 times, took the mean value and standard deviation. The table is as follow:

Naive(digit)	run time mean	validation mean	testing mean	testing std
500	46.042	0.838	0.796	0.01341640786
1000	47.292	0.858	0.788	0.01483239697
1500	52.084	0.864	0.82	0.007071067812
2000	50.814	0.862	0.796	0.01140175425
2500	49.02	0.854	0.8	0.0158113883
3000	50.816	0.856	0.816	0.01673320053
3500	52.518	0.85	0.812	0.01788854382
4000	50.6	0.84	0.794	0.01140175425
4500	53.754	0.856	0.804	0.0219089023
5000	60.84	0.87	0.79	0

Figure 5: Final results from the Digit detection (Naive Bayes)

Perceptron (digit)	run time mean	validation mean	testing mean	testing std
500	18.098	0.756	0.746	0.05458937626
1000	34.544	0.846	0.822	0.03633180425
1500	54.946	0.846	0.826	0.03130495168
2000	74.748	0.828	0.806	0.04037325848
2500	89.734	0.846	0.812	0.031144823
3000	104.688	0.862	0.856	0.01949358869
3500	124.642	0.854	0.83	0.02236067977
4000	138.544	0.888	0.838	0.02683281573
4500	161.654	0.868	0.834	0.02880972058
5000	208.97	0.86	0.85	0

Figure 6: Final results from the Digit detection (Perceptron)

Naive(face)	run time mean	validation mean	testing mean	testing std
45	38.034	0.772	0.738	0.106
90	43.202	0.846	0.83	0.0126
135	64.412	0.904	0.876	0.036
180	59.432	0.916	0.878	0.0248
225	70.63	0.928	0.898	0.00979
270	75.614	0.956	0.904	0.0162
315	80.56	0.962	0.894	0.0049
360	72.372	0.97	0.9	0.0155
405	76.6866	0.986	0.896	0.008
451	72.60666667	0.98	0.91	0

Figure 7: Final results from the Face detection (Naive Bayes)

Perceptron (face)	run time mean	validation mean	testing mean	testing std
45	2.958	0.666	0.666	0.08443932733
90	5.008	0.734	0.752	0.08288546314
135	6.13	0.83	0.81	0.06363961031
180	8.026	0.856	0.824	0.01140175425
225	10.32	0.89	0.82	0.06
270	13.504	0.924	0.856	0.0102
315	13.448	0.926	0.854	0.0378
360	17.28	0.96	0.876	0.0272
405	17.28	0.96	0.876	0.0273
451	17.28	0.96	0.9	0.0126

Figure 8: Final results from the Face detection (Perceptron)

7. **Observation:** 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, for a random iteration, using Perceptron to train 500 data points using 3 iteration gives 75% validation accuracy and 72% testing accuracy; with 5 iterations it gives us 85% validation accuracy and 77% testing accuracy; with 10 iterations, however, the accuracy of both validation and testing does not 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. The reason is perhaps that when the training sample number is larger than a certain amount, then the feature probability would become relatively stable.

4 Conclusion

Implementing the algorithms is not the hardest part of this project, however, finding good features to extract is very critical. 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 sets, we would result in better accuracy. In addition, face detection generally produces better results, and it may be because face detection only allows two outcomes, either “True” or “False”, which means that it has less of a chance to “make mistakes” compared to digit recognition.