CS 440/ECE448

Assignment 3: Pattern Recognition

April, 2018

YUHAO CHEN,
ZIRUI FAN,
JUNHAO PAN

Credit Hour: 3

1 Introduction

1.1 Overview

In this assignment, we have chosen python as our language to implement the functionalities. We majorly use Jupyter notebook as the developing environment.

The assignment includes two parts: Classification with Naive Bayes Model and Classification with Perceptrons. Both are pattern recognition methods which are extensively discussed in class. Whichever methods to use, the common goal is to classify a batch of images and identify into which class the image falls. In this specific problem, we have ten classes, each of which corresponds to a number from 0-9.

In part 1.1, we have implemented Naive Bayes Model in which single pixels are used as features. In part 2.1, we have developed a Perceptron Model in which single pixels are used as features as well.

1.2 Files

./Naive Bayes.ipynb	jupyter nootbook for part 1
./Perceptron.ipynb	jupyter nootbook for part 2
./Bayesian_Model.py	supporting python file for part 1
./Perceptron.py	supporting python file for part 1
./src/*	pictures and misc.

2 Naive Bayes Classifiers on Digit Classification

2.1 Single Pixels as Features

2.1.1 Implementation

Our goal is to use the Naive Bayes Model to classify the data and identify the digit which each image represents.

We have constructed a class **Bayesian_Model** which contains the functions which we need for training, testing and other miscellaneous functions. First, while we parse the given image data for training, we separate the images with the labels and store them in two arrays. The array for training images, $training\_classes$ is a 3-D array. The first dimension has ten 32*32 2-D arrays. Each cell of the 2-D array is a dictionary with holds the look-ups for the appearances of '1's and '0' at the specific location. Similar operations are implemented on testing dataset as well except that the array that holds the images has a dimension of 444*32*32. The indices of the image array and the label array are aligned.

Having parsed all the images, we implemented Laplace Smoothing on the data. Experiments on different smoothing constants suggest that *smaller smoothing constants can usually give more accurate results*. After applying the Laplace Smoothing on the image, we follow the testing schemes and test for each feature to determine the overall possibility of an image belonging to a class. While we update the maximum and minimum possibilities, we also update the largest and smallest posteriors and construct the confusion matrix in the meantime.

In the end, we use the result to calculate odds ratios and plot corresponding heatmaps to better present the outcomes.

2.1.2 Results

Having implemented the model, we have gathered following results:

Classification Rate:

Digit:	Classification Rate:
0	0.972
1	0.933
2	0.854
3	0.909
4	0.881
5	0.931
6	0.977
7	0.979
8	1.0
9	0.928

Confusion Matrix:

	0	1	2	3	4	5	6	7	8	9
0	0.972	0.	0.	0.	0.	0.	0.	0.	0.	0.
1	0.	0.933	0.	0.	0.	0.	0.	0.	0.	0.
2	0.	0.	0.854	0.	0.	0.	0.	0.	0.	0.
3	0.	0.	0.	0.909	0.	0.	0.	0.	0.	0.048
4	0.028	0.	0.	0.	0.881	0.	0.023	0.	0.	0.
5	0.	0.	0.	0.	0.	0.931	0.	0.	0.	0.
6	0.	0.	0.	0.	0.	0.	0.977	0.	0.	0.
7	0.	0.022	0.024	0.030	0.068	0.	0.	0.979	0.	0.024
8	0.	0.022	0.098	0.	0.051	0.	0.	0.021	1.000	0.
9	0.	0.022	0.024	0.061	0.	0.069	0.	0.	0.	0.929

Highest Posterior Probabilities:

-233.997	6798
-297.999	11748
-292.466	9339
-275.720	726
-328.367	10593
-321.741	990
-265.533	11550
-247.611	7656
-340.723	3069
-359.003	8118

Lowest Posterior Probabilities:

-1266.592	7095
-1586.448	11319
-1396.496	10131
-1629.545	10791
-1524.386	8547
-1362.906	13002
-1480.421	14586
-1700.769	9735
-1302.103	825
-1522.523	13068

The feature and odds ratio headmaps of the four most confused pairs of digits are shown below:

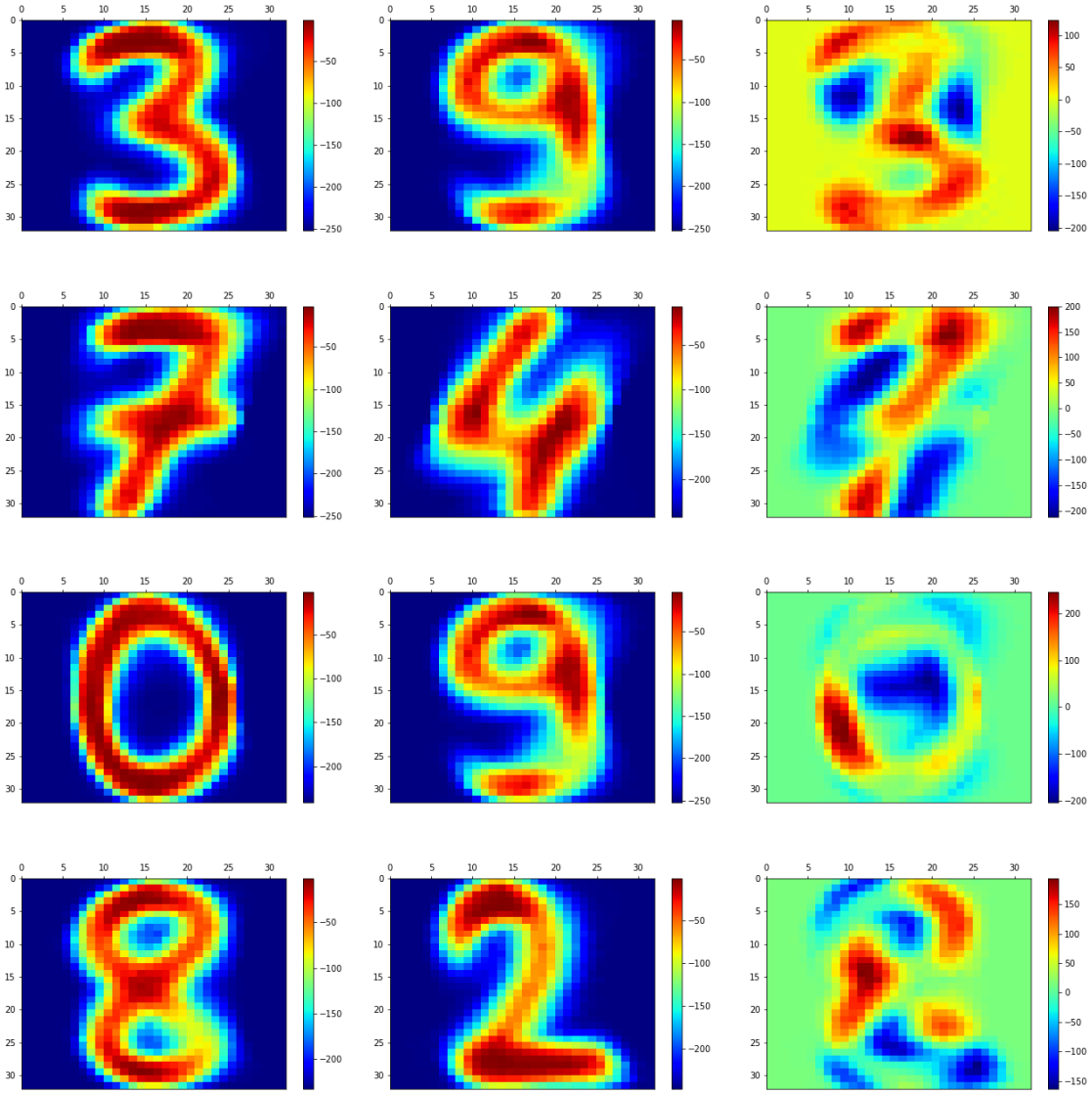

Figure 1: Four most confused pairs

Overall, we have achieved 0.9347 accuracy.

3 Discriminative Machine Learning Methods

3.1 Digit Classification with Perceptrons

3.1.1 Implementation

The goal of this section is to develop perceptron model which learns from training dataset and try to identify images correctly based on the training results.

Similar to part 1, we have also constructed a class *Perceptron* to hold the functions we need. Nevertheless, in this implementation, we simply append each image to the image array and align with the labels in the label array. Thus, the image array is a 2-D array of dimensions 2436*1024, meaning it has 2436 rows of 1024 pixels (features). In addition to the image arrays and label arrays, we also have a weight array of dimension 10*1024, which holds the weights for the 10 classes.

When we start training the model, we append the bias of either 1 or 0 to the end of each row depending on whether bias is enabled. During the training, we loop through all ten labels, and in the inner loops, we train each of the weight class (rows of the weight array) according to the labels according to the rules which we have discussed in classes. Specifically, when the outer loop is at label 5, the inner loop is only updating the weights that correspond to label 5 during each pass of epoch.

3.1.2 Comments on Parameters

We have played with several different combinations of the parameters, including learning rate decay function, bias, initialization of weights, ordering of training examples, and number of epochs.

First of all, we think that the learning rate decay function is related to number of epochs. If we have a larger number of epochs and want the learning rate to drop to the same level, we need to increase the decay rate so that it drops faster. The number of epoch is very closely related to the accuracy. Generally speaking, the more passes through the training data, the more accurate the model becomes. However, the trade-off is time and resource. To find a perfect balance, we tried difference initial learning rate, learning rate decay function, and numbers of epochs. Finally, we think that initial rate at 0.025, decay 10% at each epoch with 20 epochs is a quite balanced combination.

Randomizing initial weights gives mixed results. Therefore, we decided to stick with zeroing the weights at the beginning. It may because of the impacts of other parameter settings. The change of ordering of training example does not give substantially changed results either.

3.1.3 Results

Having implemented the model, we have gathered following results with learning rate = 0.025 and epoch = 20

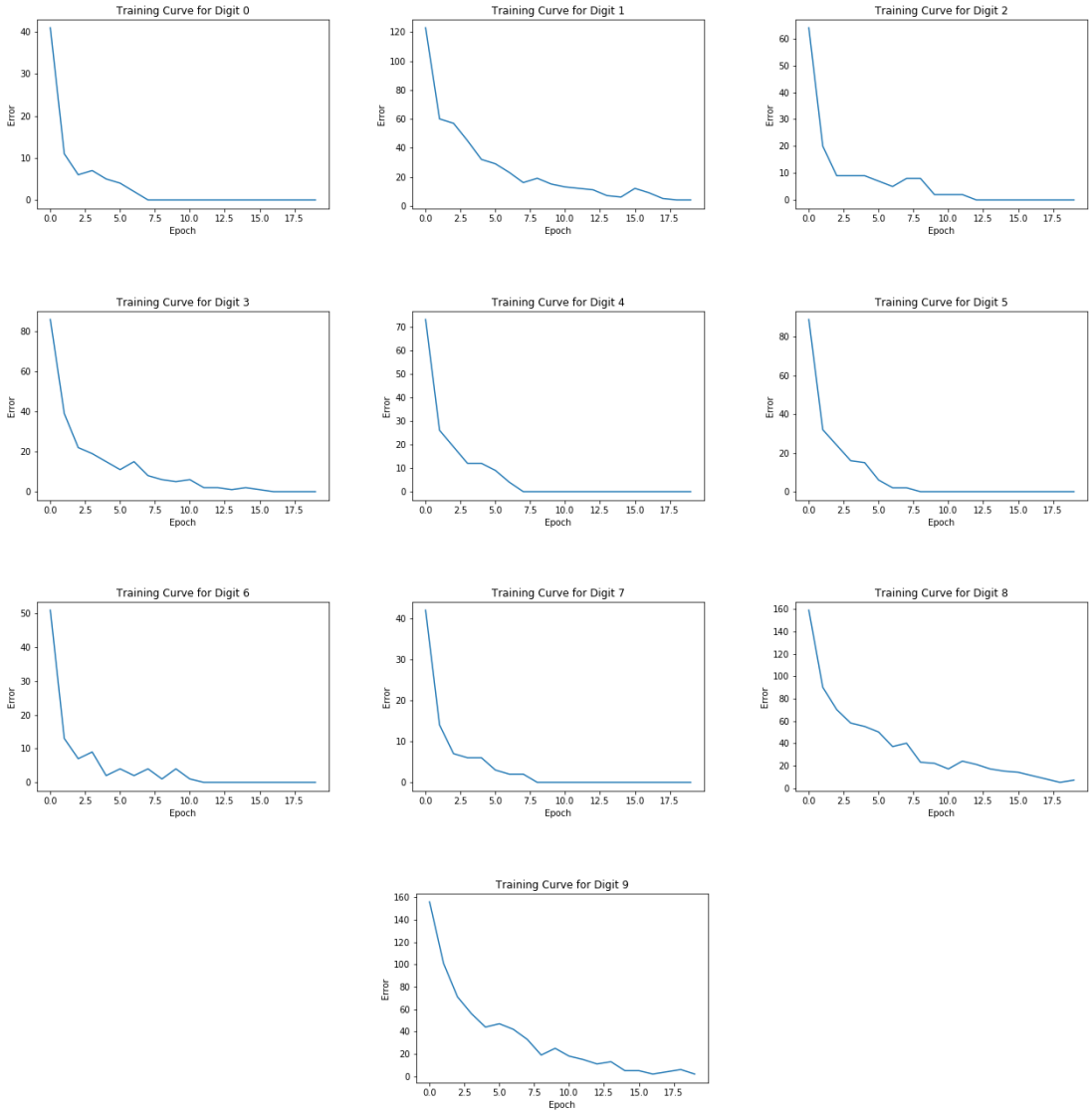

Figure 2: Learning Curves for Each Digit

Classification Rate:

Digit:	Classification Rate:
0	1.0
1	0.978
2	0.878
3	0.969
4	0.915
5	1.0
6	0.953
7	0.979
8	0.950
9	0.905

Confusion Matrix:

	0	1	2	3	4	5	6	7	8	9
0	1.000	0.	0.	0.	0.	0.	0.	0.	0.	0.
1	0.	0.978	0.	0.	0.017	0.	0.	0.	0.	0.048
2	0.	0.	0.878	0.	0.	0.	0.	0.	0.	0.024
3	0.	0.	0.	0.969	0.017	0.	0.	0.	0.	0.
4	0.	0.	0.	0.	0.915	0.	0.023	0.021	0.	0.
5	0.	0.	0.	0.	0.	1.0	0.	0.	0.025	0.
6	0.	0.	0.	0.	0.	0.	0.953	0.	0.	0.
7	0.	0.022	0.	0.	0.	0.	0.	0.979	0.	0.
8	0.	0.	0.098	0.	0.034	0.	0.023	0.	0.950	0.024
9	0.	0.	0.024	0.030	0.017	0.	0.	0.	0.025	0.905

Overall, we have achieved 0.9527 accuracy.

4 Credits

We worked together wonderfully in coding the solutions. Everyone contributed in constructing the algorithms. Junhao Pan typed the report in LaTeX, and the others reviewed it.