CS440 MP3 Report

Andong Jing ajing2 - 3 credit In charge of Part 1.1/1.2/1.3

Siping Meng smeng10 - 3 credit In charge of Part 2.1

Siyu Tao siyutao2 - 3 credit In charge of Part 2.2

# Naive Bayes Classifiers on Digit classification

## 1.1 Single pixels as features

* Briefly discuss your implementation, especially the choice of the smoothing constant.

I build a dictionary to store the train data, the key of the dictionary is the classes (digits 0-10, in this problem), and the value of the dictionary is the time of occurrence of each digit for each class. Then I use the train dictionary to predict the digit shown in the test data. For the smoothing constant, I wrote a loop to try each of numbers from 0.1 to 10 with step of 0.1. And I found that when smoothing constant is 0.1, the over-all accuracy is greatest, which is 0.9369369369369369.

* Report classification accuracy for each digit (note: this can be just the diagonal elements on the confusion matrix).

The classification accuracy of 0 is: 0.97222222

The classification accuracy of 1 is: 0.93333333

The classification accuracy of 2 is: 0.85365854

The classification accuracy of 3 is: 0.90909091

The classification accuracy of 4 is: 0.88135593

The classification accuracy of 5 is: 0.93103448

The classification accuracy of 6 is: 0.97674419

The classification accuracy of 7 is: 1

The classification accuracy of 8 is: 1

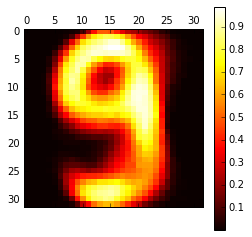
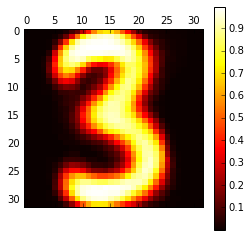
The classification accuracy of 9 is: 0.92857143

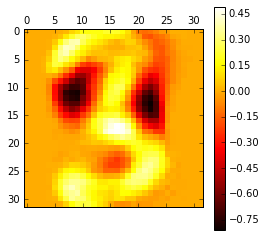
* Show the confusion matrix.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| True\guess | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 0.972 | 0 | 0 | 0 | 0.028 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0.933 | 0 | 0 | 0 | 0 | 0 | 0.022 | 0.022 | 0.022 |
| 2 | 0 | 0 | 0.854 | 0 | 0 | 0 | 0 | 0 | 0.122 | 0.024 |
| 3 | 0 | 0 | 0 | 0.909 | 0 | 0 | 0 | 0.030 | 0 | 0.061 |
| 4 | 0 | 0 | 0 | 0 | 0.881 | 0 | 0 | 0.068 | 0.051 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0.931 | 0 | 0 | 0 | 0.069 |
| 6 | 0 | 0 | 0 | 0 | 0.023 | 0 | 0.977 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.024 | 0 | 0.929 |

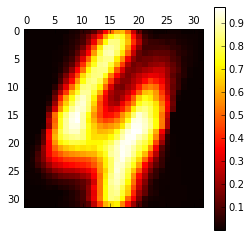
* For each digit, show the test tokens from that class that have the highest and lowest posterior probabilities according to your classifier.
* Take four pairs of digit types that have the highest confusion rates, and for each pair, display feature likelihoods and odds ratio.

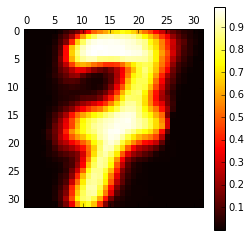
3 & 9:

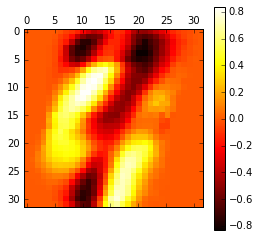




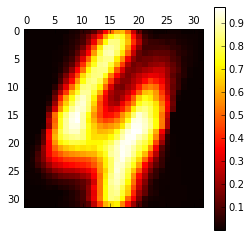
4 & 7:

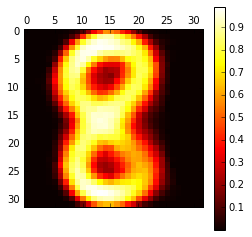


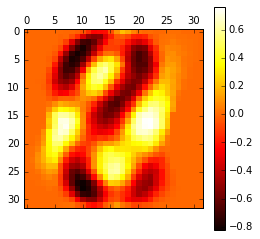




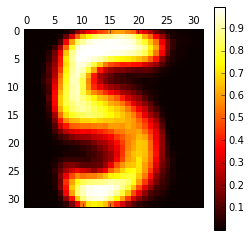
4 & 8:

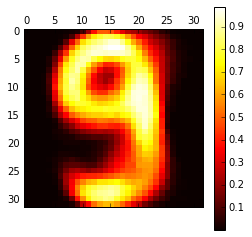


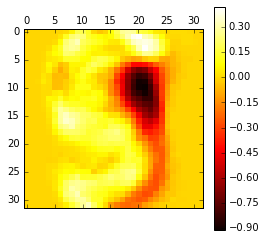




5 & 9:



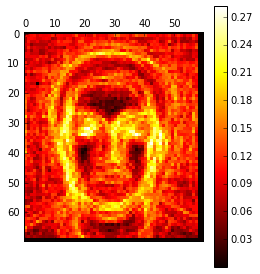




## 1.2 Pixel groups as features

## 1.3 Face classification (Extra Credit)

I applied the Naïve Bayes classifier with single pixels as features (same as part 1.1). I took 0.1 as the smoothing constant, and here is the likelihood maps for the face (when class=1):



I wrote a loop to find the optimized smoothing constant and found that when smoothing constant is 1.1, the accuracy is 90%. The confusion matrix is following:

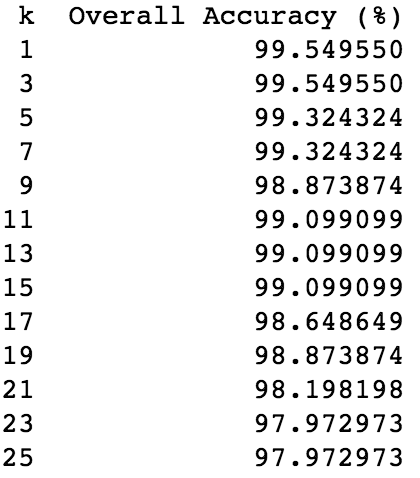
|  |  |  |
| --- | --- | --- |
| Real/guess | Non-face | face |
| Non-face | 0.8831 | 0.1169 |
| face | 0.0685 | 0.9315 |

# 2. Alternative models for Digit classification

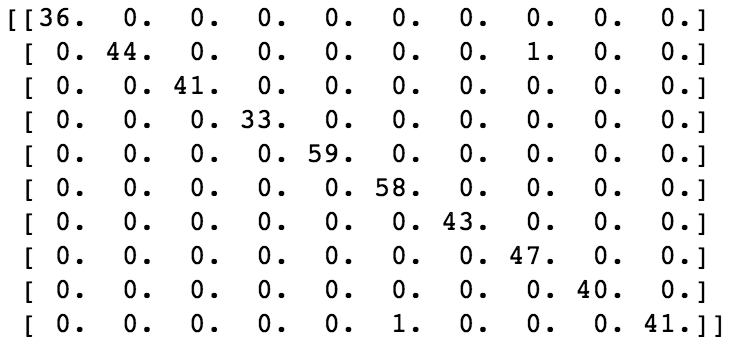
## 2.1 Digit classification with perceptrons

## 2.2 Digit classification with nearest neighbor

The team’s choice of similarity function is by evaluating the Euclidean distance between any two vectors. It yields the same output as the Manhattan distance, but is more efficient and thus quick to compute. Based on the team’s research, Euclidean distance is also the mostly widely used similarity function in the implementation of nearest neighbor classification. The overall accuracy decreases as n increases, as shown by the following figure. Based on the team’s speculation, a large k introduces bias, which may explain the relatively poor performance of large k’s. However, small k’s are prone to outliers in the training datasets. Luckily, the good performance of the model when k is small indicates the overall quality of the dataset is very good.



Since both k=1 and k=3 yields the highest accuracies and a model with k=1 is more efficient. The team constructs the confusion matrix with k=1. It is shown below.



To account for the fluctuation in the running time of individual queries, the team takes the total running time of all 444 pieces of data in the test sets and divide it by 444. The running time for a single query is about 196 μs. It is very efficient, since we utilized Euclidean distance as our similarity function. To further improve its performance, we may consider eliminating choices of potential labels if the distance between the query in the test dataset and several points in the training set far exceeds the distance with the current nearest neighbor. This process of elimination saves running costs from evaluating labels of tiny possibilities.

Finally, compare your nearest-neighbor accuracy to the accuracies you got with Naive Bayes and Perceptron.

The nearest-neighbor accuracy is about 100%, while the Naïve Bayes is only about 93% (see above). It is obvious that nearest-neighbor algorithm has much better accuracy than Naïve Bayes algorithm.

## 2.3 Extra Credit

### 2.3.1 Visualization

### 2.3.2 Differentiable perceptron

### 2.3.3 Other learning algorithms

The team referenced the svm classifier (by scikit-learn) tutorial which could be found at the following URL.

[http://scikit-learn.org/stable/auto\_examples/classification/plot\_digits\_classification.html](# http://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html)

The team used it to classify the digits supplied by this assignment. The svm model with default parameters yields an accuracy of less than 97% accuracy on the test dataset. However, after experimenting with different gamma values, the team found that a gamma=0.01 yields only 3 errors out of the 444 data, which is an accuracy of 99.32%. The confusion matrix is shown in the following figure.

