# CS440/ECE448 Fall 2016

# Artificial intelligence Assignment 3: Naive Bayes Classification

**Work Distribution:**
Part 1: Yifei Li
Part 2: Bangqi Wang
13th Nov, 2016

## Part 1: Digit classification

The goal of this assignment is to implement Naive Bayes classifiers and to apply it to the task of classifying digit and face data.

## General Implementation

The basic idea of applying image classification is to first calculate likelihood and prior probability on the training set and then calculate posteriori probability according to the previous step. There are 28*28 pixels in each training image and ‘+’, ‘#’ indicates it is foreground and ‘ ’ denotes background. For this part, we don’t distinguish between two foreground value but will differentiate them later.

In step 1, there are two probabilities that needs to be calculated, Prior and likelihood. The likelihood is defined as **$P(F_{ij}$ | class)** for every pixel location (i, j) and for every digit class. More specifically:

**$P(F_{ij} = f$ | class) = (# of times pixel (i, j) has value f in training examples from this class) / (Total # of training examples from this class)**

Note that in order to dealing with feature that were never seen or seen too few times, we experiment with few laplace smoothing constant for best accuracy. Finally, the constant k works best when it is equal to 1.

The prior probability is simply obtained by count frequencies of each digit in the example set.

In step 2, now we should use the likelihood and prior we have to predict the class of testing image. This is achieved by performing **maximum a posteriori (MAP)** classification of test images. We calculate:

**$P(\text{class}) \cdot P(f_{1,1} \mid \text{class}) \cdot P(f_{1,2} \mid \text{class}) \cdot \ldots \cdot P(f_{28,28} \mid \text{class})$**

In order to avoid underflow, the log version should be used:

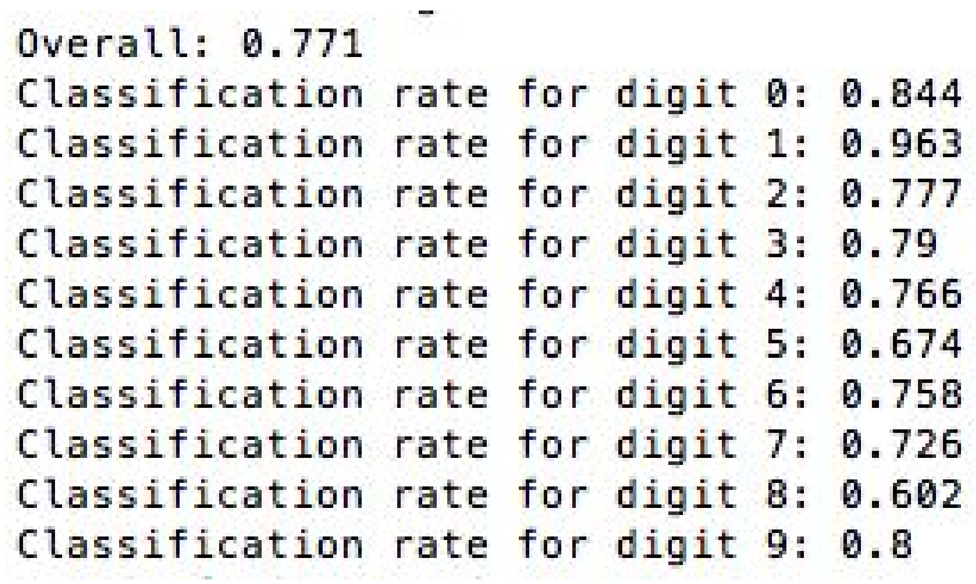
$$\log P(\text{class}) + \log P(f_{1,1} \mid \text{class}) + \log P(f_{1,2} \mid \text{class}) + \ldots + \log P(f_{28,28} \mid \text{class})$$

(Note we need to calculate the posteriori probability for class 0-9)

After calculating probability for all classes, we pick the class with largest probability and use it as our prediction.

## Classification results

Our model achieves 77.1% overall accuracy on all digit classes. The classification result for each digit is reported as follows:



```
Overall: 0.771
Classification rate for digit 0: 0.844
Classification rate for digit 1: 0.963
Classification rate for digit 2: 0.777
Classification rate for digit 3: 0.79
Classification rate for digit 4: 0.766
Classification rate for digit 5: 0.674
Classification rate for digit 6: 0.758
Classification rate for digit 7: 0.726
Classification rate for digit 8: 0.602
Classification rate for digit 9: 0.8
```

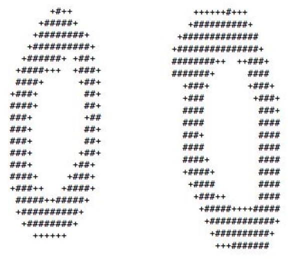
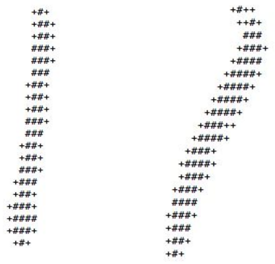
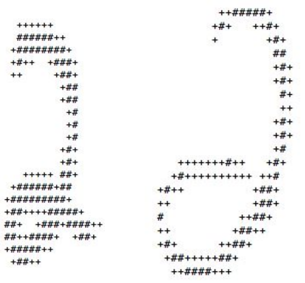
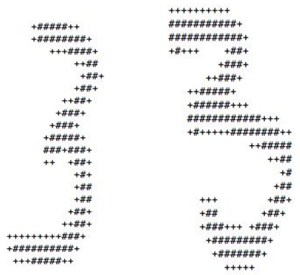
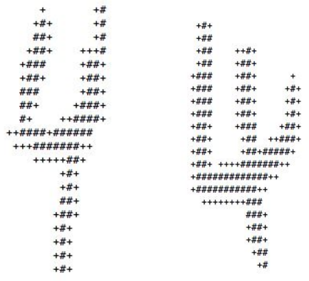
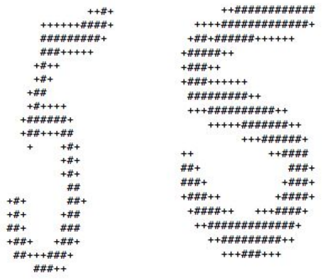
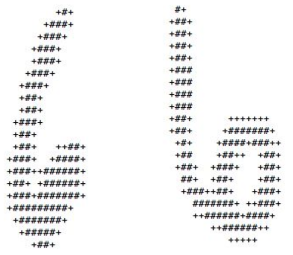
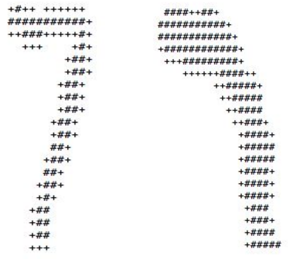
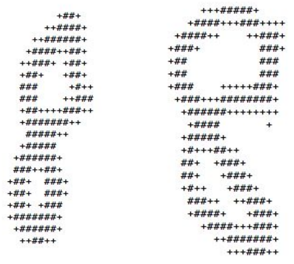
Also, we calculate the confusion matrix and most/least prototypical instance for better understanding of the learning result. (the original confusion matrix from program output is messy...so we manually input the matrix here)

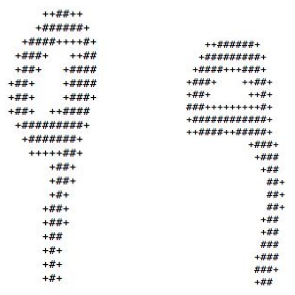
**Confusion Matrix**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.844 | 0.0 | 0.011 | 0.0 | 0.011 | 0.056 | 0.033 | 0.0 | 0.044 | 0.0 |
| 1 | 0.0 | 0.963 | 0.009 | 0.0 | 0.0 | 0.019 | 0.009 | 0.0 | 0.0 | 0.0 |
| 2 | 0.01 | 0.029 | 0.777 | 0.039 | 0.01 | 0.0 | 0.058 | 0.01 | 0.049 | 0.019 |
| 3 | 0.0 | 0.02 | 0.0 | 0.079 | 0.0 | 0.03 | 0.02 | 0.06 | 0.02 | 0.06 |
| 4 | 0.0 | 0.009 | 0.0 | 0.0 | 0.766 | 0.0 | 0.028 | 0.009 | 0.019 | 0.168 |
| 5 | 0.022 | 0.022 | 0.011 | 0.13 | 0.033 | 0.674 | 0.011 | 0.011 | 0.022 | 0.065 |
| 6 | 0.011 | 0.066 | 0.044 | 0.0 | 0.044 | 0.055 | 0.758 | 0.0 | 0.022 | 0.0 |
| 7 | 0.0 | 0.057 | 0.028 | 0.0 | 0.028 | 0.0 | 0.0 | 0.726 | 0.028 | 0.132 |
| 8 | 0.019 | 0.01 | 0.029 | 0.136 | 0.019 | 0.058 | 0.0 | 0.01 | 0.602 | 0.117 |
| 9 | 0.01 | 0.01 | 0.01 | 0.03 | 0.09 | 0.02 | 0.0 | 0.02 | 0.01 | 0.8 |

Row represents Truth, Column represents prediction

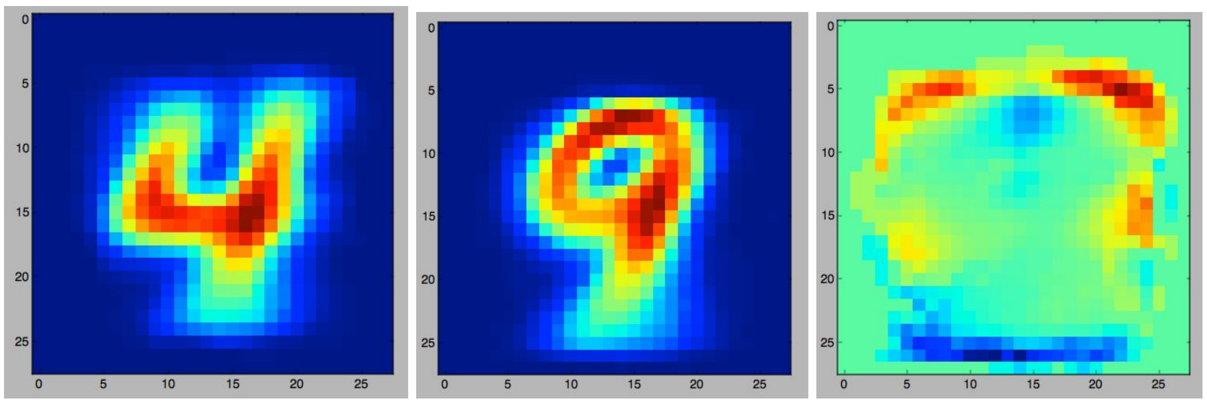
**Most/Least prototypical instance**

| | | |
|---|---|---|
|  Most (image: 723)<br>Least (image: 610) |  Most (image: 633)<br>Least (image: 993) |  Most (image: 795)<br>Least (image: 50) |
|  Most (image: 205)<br>Least (image: 291) |  Most (image: 111)<br>Least (image: 253) |  Most (image: 471)<br>Least (image: 70) |
|  Most (image: 632)<br>Least (image: 362) |  Most (image: 784)<br>Least (image: 671) |  Most (image: 560)<br>Least (image: 758) |

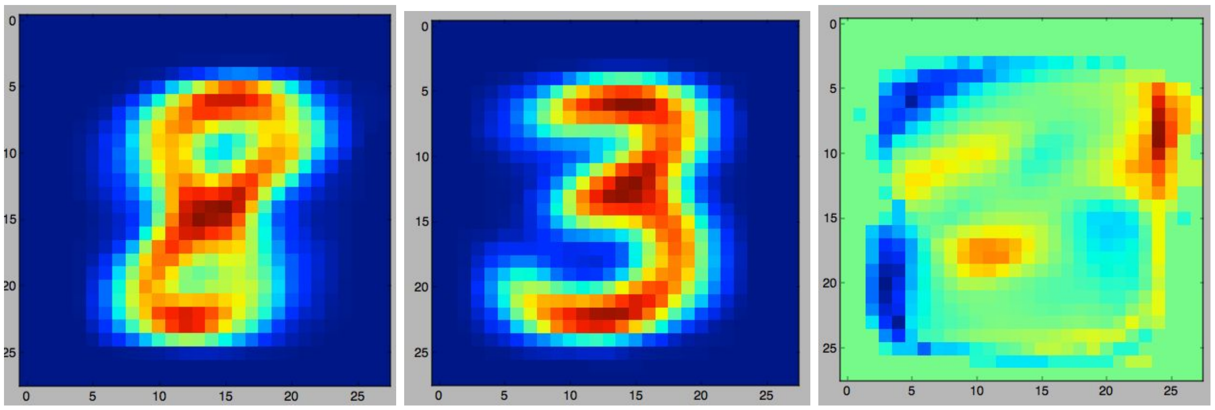
| | |
|---|---|
|  Most (image: 745)<br>Least (image: 492) | **The images use 0-based index.**<br>**Left is Most prototypical, Right is Least prototypical.** |

For the most confusing pairs in the confusion matrix, I have generated odd ratios and displayed results here. Note that the deeper the color the lower probability is. Light color denotes higher odd ratios.

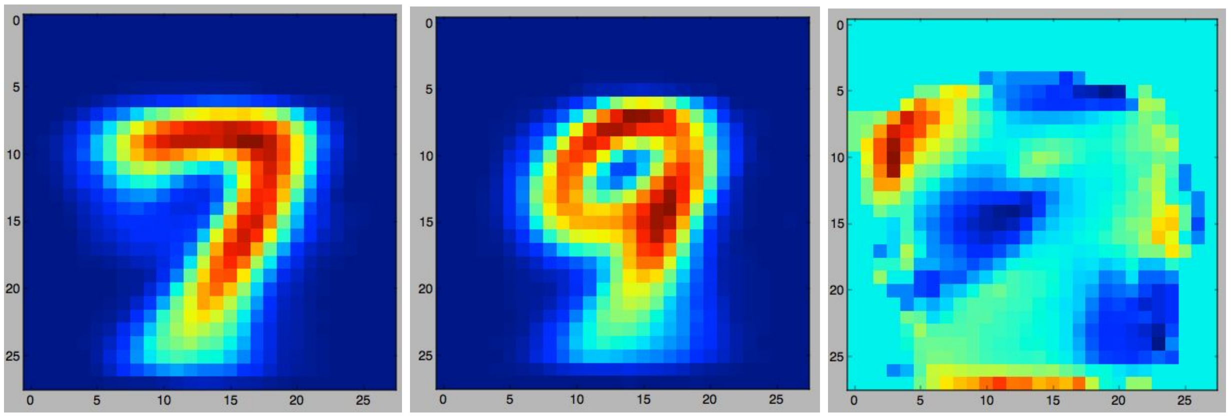
**odd ratio**: 0.168 (4,9)



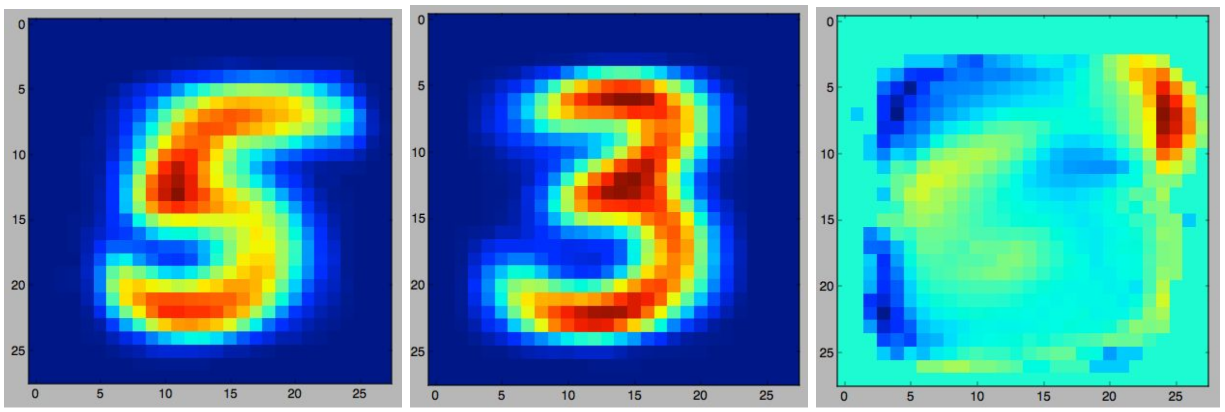
**odd ratio**: 0.136 (8,3)



**odd ratio**: 0.132 (7,9)



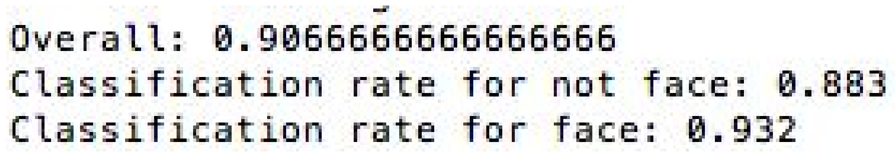
**odd ratio**: 0.13 (5,3)



## Extra credit

In this part, we will perform classification on ternary features, which means treating '+' and '#' differently. Since we favor '#' more than '+', we will assign more weight when counting foreground. In this MP, we assign 1 to '#' and 0.5 to '+'. The accuracy is thus increased by 0.1%.

For facedata, we did some modification on the original code and achieve the following results. The output is satisfying:



```
Overall: 0.906666666666666
Classification rate for not face: 0.883
Classification rate for face: 0.932
```