CS440 - Artificial Intelligence

Assignment 3- Naive Bayes Classification

4 credit hours

By,
Udit Mehrotra (umehrot2)
Sanjana Chandrashekar (chndrsh4)
Suhas Hoskote Muralidhar (shmural2)

REPORT – PART 1

Part 1.1 - Single pixels as features

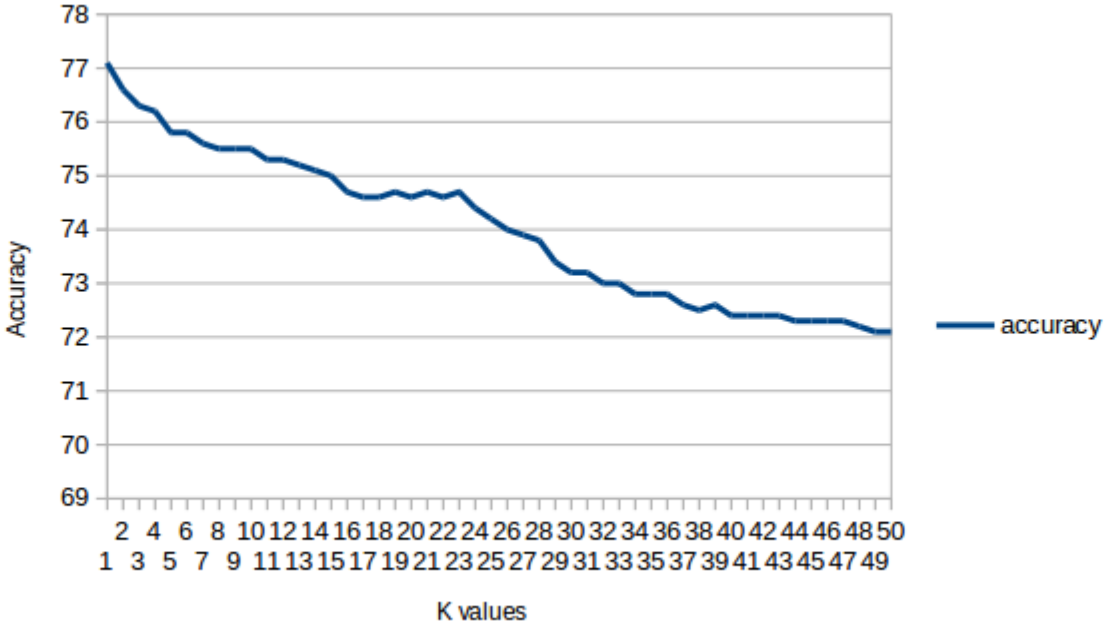
Implementation details:

- The code was implemented in python. Each digit was considered to be an object having the prior probability value, the posterior probabilities for each of the tuples, image positions in the training file and the number of occurrences as properties.
- In the training phase, the training images and training labels are read and the values of posterior probabilities for the digits for each feature and each possible value of the feature was stored as a hash (Python dict). The key of the hash was a tuple of the form (<x-index of feature>, <y-index of feature>, value of feature-{0,1}) The value is the log of the smoothed value of the probability of occurrence of that particular value for that feature. The log of the prior probability is added to this value.
- In the testing phase, the test images are read and for each image the probability of each feature's value is calculated by looking up the corresponding key in the hash table. This value of probability is found for each digit. The digit with the highest value predicted is the predicted label. The values for each of the digit images are predicted.
- The predicted values are compared with the actual values and the confusion matrix is computed.
- The log likelihood maps and odds-ratio maps were constructed using matplotlib in python.

Execution:

1. Navigate to part_1.1 code folder
2. python mp3_map.py <training_images filepath> <training_labels filepath> <test_imges filepath> <test_labels filepath>

Variation of accuracy with the value of smoothing constant k



We varied the values of k from 1 to 50 and found that the accuracy decreases with each value of k. The value of k for smoothing was therefore retained as 1 for all the programs.

Accuracy with MAP- 77.1%
Accuracy with Maximum Likelihood- 77.0%
The maximum likelihood estimate does not vary much from the MAP estimate. This means that the contribution from the prior probability is negligible. This is probably because each of the digits have similar counts in the training set. (The counts range between ~430 to ~560). Hence, when the prior probability for each of the digits is calculated, it is unlikely that there will be a huge variation in the prior probabilities for each of the digits. Therefore, the prior probability does not heavily influence the MAP estimate and similar accuracies are obtained even when it is ignored.

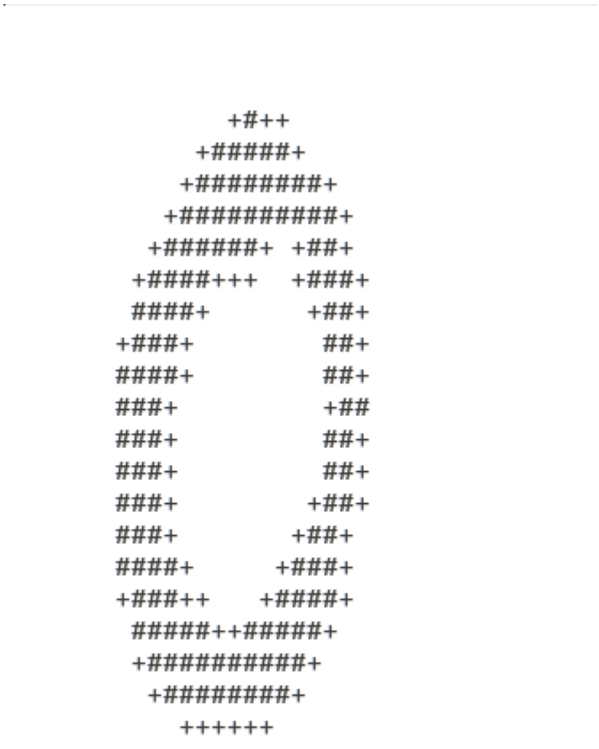
Classification Rates for the digits

Digit	Classification Rate
0	84%
1	96%
2	77%
3	80%
4	78%

5	65%
6	75%
7	73%
8	60%
9	80%

Confusion Matrix

	0	1	2	3	4	5	6	7	8	9
0	0.84	0.0	0.01	0.0	0.01	0.06	0.03	0.0	0.04	0.0
1	0.0	0.96	0.01	0.0	0.0	0.02	0.01	0.0	0.0	0.0
2	0.02	0.03	0.77	0.04	0.01	0.0	0.06	0.01	0.05	0.02
3	0.0	0.02	0.0	0.8	0.0	0.02	0.02	0.06	0.02	0.06
4	0.0	0.01	0.01	0.0	0.78	0.0	0.03	0.01	0.02	0.15
5	0.02	0.02	0.01	0.15	0.03	0.65	0.01	0.01	0.02	0.07
6	0.01	0.07	0.04	0.0	0.04	0.07	0.75	0.0	0.02	0.0
7	0.0	0.07	0.03	0.0	0.03	0.0	0.0	0.73	0.02	0.13
8	0.02	0.01	0.03	0.14	0.02	0.07	0.0	0.01	0.6	0.11
9	0.01	0.01	0.01	0.03	0.09	0.02	0.0	0.02	0.01	0.8

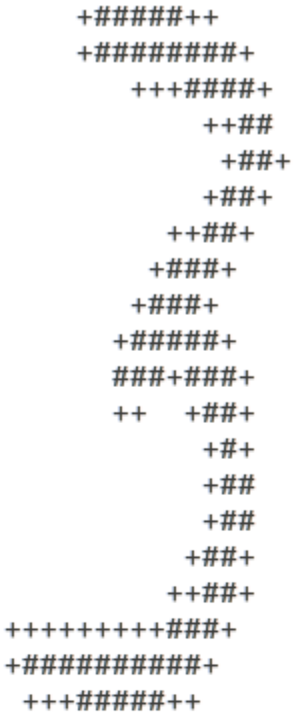
Digit	Prototypical instance
0	

```
       +#++
     +#####+
    +########+
   +##########+
  +######+ +##+
 +####+++  +###+
 ####+      +##+
+###+        ##+
####+        ##+
###+         +##
###+         ##+
###+         ##+
###+        +##+
###+       +##+
####+     +###+
+###++   +####+
 #####++#####+
 +##########+
  +########+
    ++++++
```

1

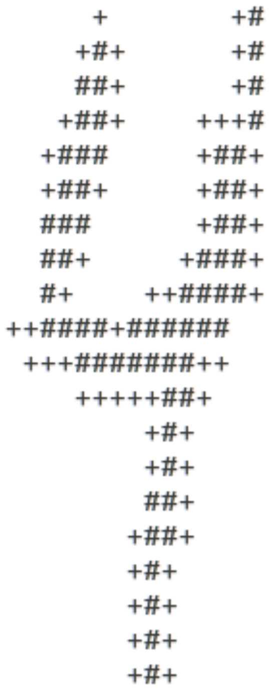
```
    +#+
    +##+
    +##+
    ###+
    ###+
    ###
   +##+
   +##+
   +##+
   ###+
   ###
  +##+
  +##+
  ###+
 +###
 +##+
+###+
+####
+###+
 +#+
```

<table>
<tr>
<td>2</td>
<td><pre>
  ++++++
  ######++
 +########+
 +#++ +###+
 ++   +##+
         +##
         +##
          +#
          +#
          +#
         +#+
         +#+
   +++++ ##+
 +######+##
+#########+
+##++++#####+
##+ +###+####++
##++####+ +##+
+#####++
 +##++
</pre></td>
</tr>
</table>

3	

```
    +#####++
    +########+
       +++####+
           ++##
            +##+
           +##+
         ++##+
        +###+
       +###+
      +#####+
      ###+###+
      ++  +##+
           +#+
           +##
           +##
          +##+
         ++##+
+++++++++###+
+##########+
 +++#####++
```

4



```
     +       +#
    +#+      +#
    ##+      +#
   +##+    +++#
  +###     +##+
  +##+     +##+
  ###      +##+
  ##+     +####+
  #+    ++####+
++####+######
 +++########++
    +++++##+
        +#+
        +#+
        ##+
       +##+
       +#+
       +#+
       +#+
       +#+
```

5

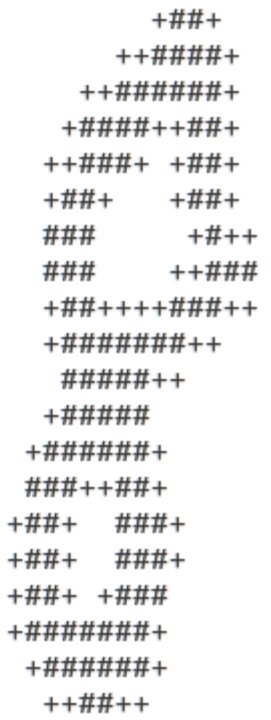
```
            ++#+
     ++++++####+
     #########+
     ###+++++
    +#++
    +#+
   +##
   +#++++
  +######+
  +##+++##
   +    +#+
        +#+
        +#+
         ##
+#+      ##+
+#+     +##
##+     ###
+##+   +##+
 ##+++###+
    ###++
```

6

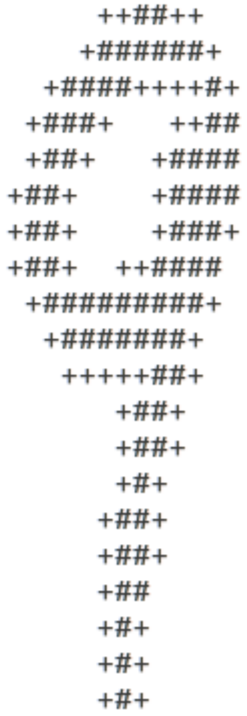
```
        +#+
      +###+
     +###+
    +###+
    +###+
   +###+
  +###+
  +##+
  +##+
 +###+
 +##+
 +##+   ++##+
+###+  +####+
+###++######+
+##+ +######+
+###+#######+
+##########+
 +########+
  +######+
    +##+
```

7

```
+#++ ++++++
###########+
++###+++++#+
  +++    +#+
        +##+
        +##+
       +##+
       +##+
       +##+
      +##+
      +##+
      ##+
     +##+
     ##+
    +##+
    +#+
   +##
   +##
   +##
   +++
```

8	

```
        +##+
      ++####+
    ++######+
   +####++##+
  ++###+ +##+
  +##+   +##+
  ###     +#++
  ###    ++###
  +##++++###++
  +#######++
   #####++
  +#####
 +######+
 ###++##+
+##+  ###+
+##+  ###+
+##+ +###
+#######+
 +######+
  ++##++
```

9	

```
     ++##++
    +######+
  +####++++#+
 +####+  ++##
 +###+  +####
+###+   +####
+###+   +###+
+###+ ++####
 +#########+
  +#######+
   +++++##+
      +##+
      +##+
      +#+
     +##+
     +##+
     +##
     +#+
     +#+
     +#+
```

Interesting miss-predictions

The interesting digits were chosen based on what seems extremely obvious to the human annotator that it is a particular digit and cannot be anything else but strangely has been classified as something else by the classifier. In these selected cases, it seems hard to predict why the classifier could have mistaken the actual digit to be something else.

Image	Actual Digit	Predicted Digit

++
       +##
       +#+
        #+
  ++   ##
  +#   ##+
  +#   +##+
  +#   +##+
  ##   +##+
 +##   ##+
 +##   +#+
 +#+   ##+
 ##   +## ++
+## +++#####
+##+######+++
+#####++#+
++#++ +#+
       +#+
       +#+
       +#+
       +#+
	4	6

+####+++
   +#++#####+
   +#+ ++####+
      ++####+
     +###++
    +##++
    ##+
    +##+
     +##+
      +##+
       +##+
        +#+
        +#+
        ##+
       +#+
      +##+
     +##+
+#+++##+
+#####+
 +###+
	3	7

<table>
<tr>
<td><pre>
    +++#+
  +#####+
 +##+++
 ##+
+##
+#+
+#+
##+
+#+
+#+
 ##
 ##+   +++
 +##   +#####+
  +#+   +###++##+
   ##+   ###   +##
   +##+ +##+   +#+
    ++##+ ##+   +#+
      ++#####+ +##
        +##########
           +++++++
</pre></td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td><pre>
   +++   ##
  +##+   +#+
  ##+   +##
  ##+   +##+
  ##+   +##
  ##   +##
  ##   +##+
  ##   +##++
  ##+ +####+
  +##++####+
  +#######+
   ++###+
    +##+
   +##+
   +##+
  +##+
  ##+
 +###++
+####+
 ##+
</pre></td>
<td>4</td>
<td>8</td>
</tr>
</table>

<table>
<tr>
<td><pre>
  +++#####++
 +##########++
+###+   ++####+
+###+   +++##
+###+   +
 ###+
 ###+
 ###
 ##+
 ##+
 ##+ +##+
+###+####+
+#########
+#####++##+
 ###++ ##+
  ++ ++ +##
     +##+##
      +####+
       +###+
        +##+
</pre></td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td><pre>
     ###+++   ++++
     ##############+
     +#############+
      ++++#########+
            +#####
           ++###++
           ####+
         +####+
    +++++####+
   +########+
   +###########+
   +###########+
   +####++++++++
  ++###+
  +###+
 +###+
+###+
####+
####+
+#+
</pre></td>
<td>7</td>
<td>8</td>
</tr>
</table>

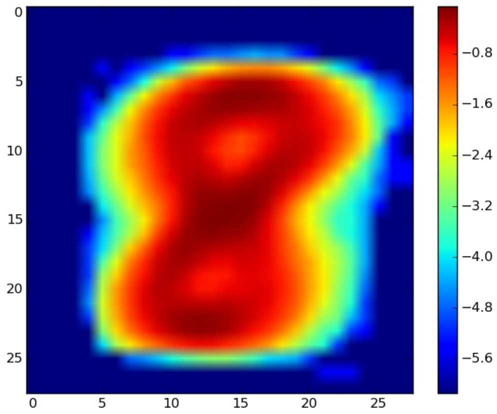
<table>
<tr>
<td><pre>
           ++++
         ++####+
         +#####+
       +#######+
       +##+####
      +##+ ###+
     +##+ ###
    +##+   ###
   +##+   ###
   +##+   ###
  +##+   ###
  ##+   ###
 +##+   ##+
 ##+   +##
+##+   +##+
+##+   +##+
+##   +##+
+##+ +###+
 +######
  +##++
</pre></td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td><pre>
     +++++++++++
   ++###########++
 ++#########+#####+
+####++   +++###+
+####+   +###
##++++   +##
+##+   +###
+###+   ++###+
 +####++++++++####+
  ++#############+
    ++++######+++
        ++++++
</pre></td>
<td>0</td>
<td>4</td>
</tr>
</table>

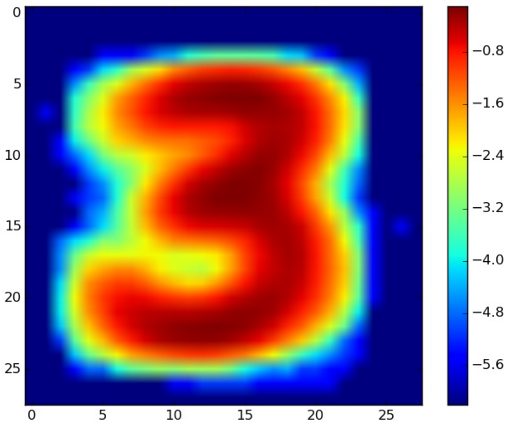
Max Likelihood ratios and odds Ratios

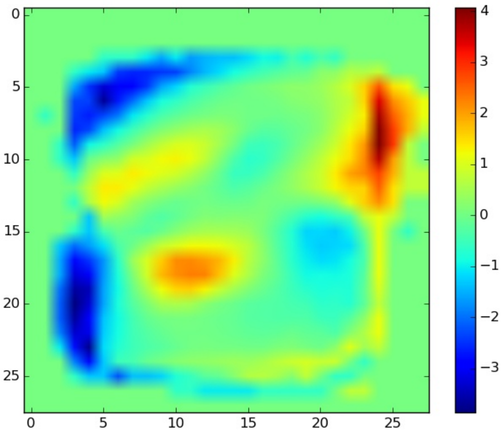
From the confusion Matrix, the pairs of digits with the highest values are

8,3
7,9
5,3
4,9

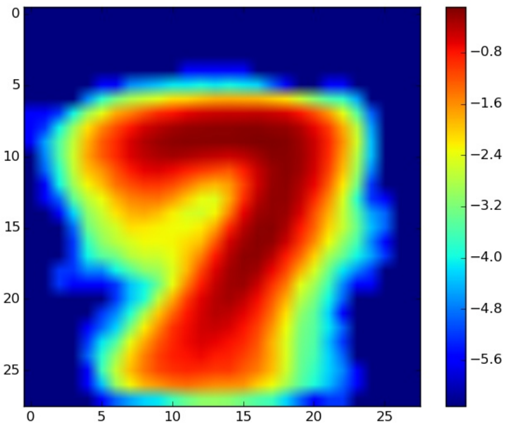
1. 8 and 3

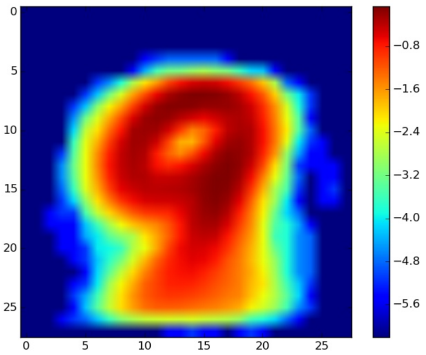

0
5
10
15
20
25
0
5
10
15
20
25
−0.8
−1.6
−2.4
−3.2
−4.0
−4.8
−5.6

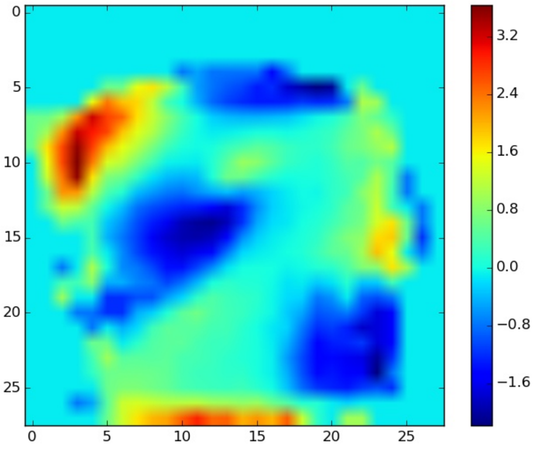

0
5
10
15
20
25
0
5
10
15
20
25
−0.8
−1.6
−2.4
−3.2
−4.0
−4.8
−5.6


0
5
10
15
20
25
0
5
10
15
20
25
4
3
2
1
0
−1
−2
−3

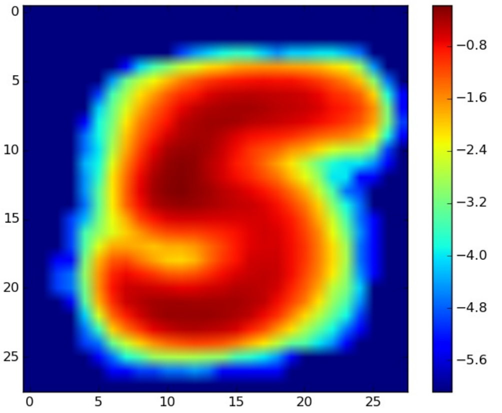
2. 7 and 9

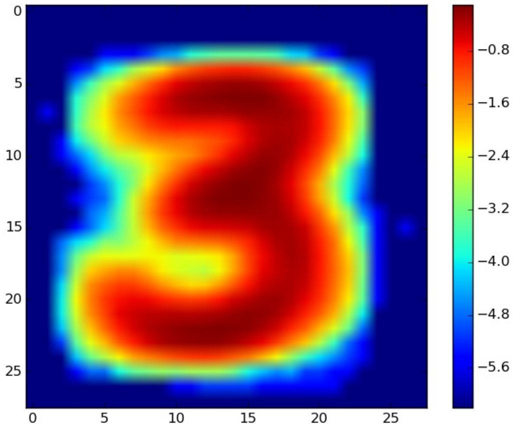

0
5
10
15
20
25
0
5
10
15
20
25
−0.8
−1.6
−2.4
−3.2
−4.0
−4.8
−5.6

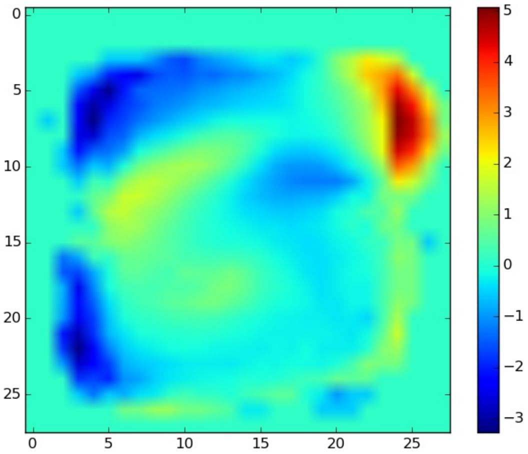

0
5
10
15
20
25
0
5
10
15
20
25
−0.8
−1.6
−2.4
−3.2
−4.0
−4.8
−5.6


0
5
10
15
20
25
0
5
10
15
20
25
3.2
2.4
1.6
0.8
0.0
−0.8
−1.6

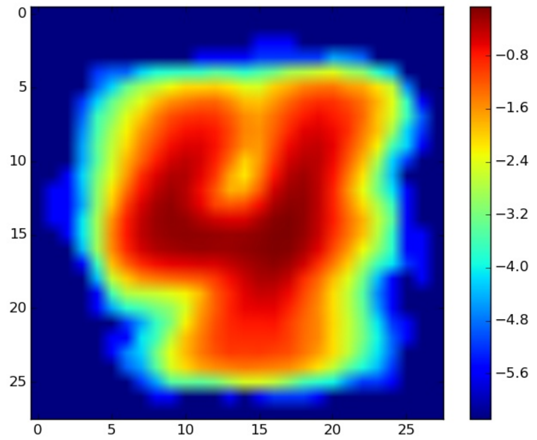
3. 5 and 3

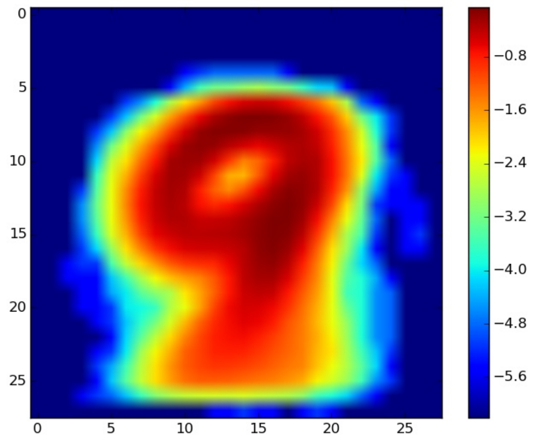

0
5
10
15
20
25
0
5
10
15
20
25
−0.8
−1.6
−2.4
−3.2
−4.0
−4.8
−5.6

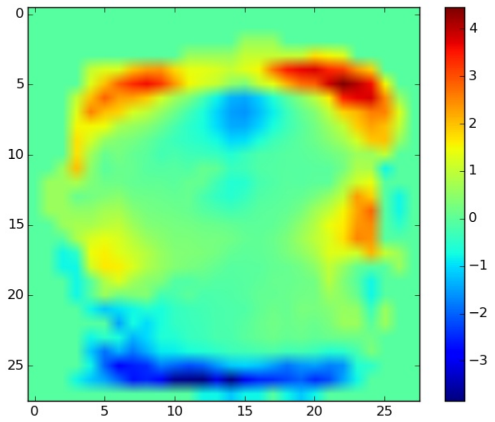

0
5
10
15
20
25
0
5
10
15
20
25
−0.8
−1.6
−2.4
−3.2
−4.0
−4.8
−5.6


0
5
10
15
20
25
0
5
10
15
20
25
5
4
3
2
1
0
−1
−2
−3

4. 4 and 9


0
5
10
15
20
25
0
5
10
15
20
25
−0.8
−1.6
−2.4
−3.2
−4.0
−4.8
−5.6


0
5
10
15
20
25
0
5
10
15
20
25
−0.8
−1.6
−2.4
−3.2
−4.0
−4.8
−5.6


0
5
10
15
20
25
0
5
10
15
20
25
4
3
2
1
0
−1
−2
−3

Part 1.2 - Pixel groups as features

Implementation details:

1. Depending on the pixel group window size, we consider the value at corresponding pixel inside the boundary.
2. For each of the value i.e. 0 for “ “ and 1 for # or +, we calculate the binary to decimal value and use the decimal value as a distinct feature value for our model.
3. Ex: consider 2 * 2 Disjoint grouping, the pixel values at position 0, 1, 28 and 29 are considered and assume their values are 0, 1, 0, 1, then the binary value of 0101 is converted to decimal, which is 5 and for this value, the probability for each of the class is calculated and the model is built.
4. The same procedure is followed during testing, where the decimal value calculated is used to look up the probabilities from the model. Smoothing is applied to avoid zero probability problem.
5. The above steps are same for both disjoint and overlapped grouping, however, in each case, care is taken to not move the boundary outside 28 * 28 original window.

Disjoint patches

Window Size	Accuracy	Running time for training feature sets in milli seconds	Running time for testing feature sets in milli seconds	Total Time taken in milli seconds
2 * 2	84.3%	1198	1259	5054
2 * 4	83%	857	902	4330
4 * 2	83.6%	926	778	4221
4 * 4	74%	701	901	4028

Overlapping patches

Window Size	Accuracy	Running time for training feature sets in milli seconds	Running time for testing feature sets in milli seconds	Total Time taken in milli seconds of execution
2 * 2	85.3%	3025	4184	9663
2 * 4	85.3%	3683	4997	11139
4 * 2	85.6%	4035	5158	11649
4 * 4	80.3%	5163	8438	16491
2 * 3	86.1%	3261	5056	10738

3 * 2	86.6%	3519	6284	12473
3 * 3	83.8%	4176	6215	13127

Steps to Execute:

1. Navigate to Part_1.2 code folder
2. javac *.java
3. java PixelGrouping <Type> <feature row size> <feature col size>
    a. Type = 0 for Disjoint, 1 for overlapping
    b. row size and col size of feature set. Ex: 2 * 2

Observation of different combination of pixel groups for accuracy

1. For 1 * 1 pixel group we find that the overall accuracy for digit classification is 77.1%. However, when we consider any other pixel grouping feature, either disjoint or overlapping, the overall accuracy is always more than 1 * 1 except for 4 * 4 disjoint pixel grouping.
2. The above observation can be justified, as the pixel grouping feature is a relaxed estimate of Naive Bayes Classification to design more accurate model of dependencies between random variables. Viewing pixels as groups gives the classifier more granular information about the image when compared to viewing individual pixels.
3. Among both Disjoint and Overlapping pixel grouping feature, we find that 3 * 2 overlapping pixel grouping gave the highest accuracy of 86.6%. The reason for this increased accuracy is due to the rectangle window combination as opposed to square window size. This enables the classifier to build a model which considers all the corner cases and hence evaluates multiple pixel grouping combination. For example, most of the training/ test samples consists blank in first few rows and columns. Thus, when we consider overlapping 3 * 2 pixel grouping, we are leveraging rectangular window with multiple feature combination. This feature considers blank and adjacent foreground pixels which helps to better classify certain digits.
4. Also, the overall accuracy for each of the Overlapping pixel grouping is always greater than that of its Disjoint counterpart. This is expected, as overlapping pixel grouping considers more features while building the model and hence it evaluates most of the pixel combination and hence builds a more accurate model i.e. number of pixel combination explored by Overlapping pixel grouping is always more than that of Disjoint. Thus, the model generated by Overlapped pixel grouping provides better accuracy for unseen test samples, when compared to that of Disjoint grouping.

Discussion of running time for training and testing for the different feature sets

1. For Disjoint pixel grouping, we observe that the total time for training and testing as well as total program execution time decreases as window size increases. i.e. time taken for

2 * 2 pixel grouping is more compared to that of 4 * 4. This is expected, because as the window size increases, due to disjoint grouping, the number of different pixel combination decreases due to boundary constraints.

2. In case of Disjoint grouping, 2 * 2 window size resulted in highest training and testing time of 1198 and 1259 milli seconds respectively.
3. For Overlapping pixel grouping, the total time for training, testing and overall execution time increases as window size increases. This is due to the fact that increase in window size results in exploration of huge number of pixel grouping combination for Overlapping case. This results in increased execution time for building the model (training) and predicting the labels (testing).
4. In case of Overlapped grouping, 4 * 4 window size resulted in highest training and testing time of 5163 and 8348 milli seconds respectively.
5. To Summarize: Running time decreases linearly with increase in feature set size for Disjoint and running time increases linearly with increase in feature set size for Overlapped grouping.

Extra Credit:

Experimenting with Ternary features to improve the accuracy of the Naive Bayes model

Implementation details: In this case, we considered 3 values i.e. if pixel value is “ ”, attribute as 0, if pixel value is #, attribute value is 1 and if pixel value is +, attribute value is 2.
Accuracy of the classification = 77.5%.
Confusion Matrix

	0	1	2	3	4	5	6	7	8	9
0	0.84	0.0	0.01	0.0	0.0	0.07	0.04	0.0	0.03	0.0
1	0.0	0.95	0.0	0.0	0.0	0.02	0.01	0.0	0.02	0.0
2	0.01	0.03	0.76	0.06	0.01	0.0	0.06	0.02	0.05	0.01
3	0.0	0.02	0.0	0.8	0.0	0.03	0.02	0.05	0.03	0.05
4	0.0	0.01	0.0	0.0	0.79	0.0	0.02	0.01	0.02	0.15
5	0.02	0.01	0.01	0.15	0.03	0.66	0.01	0.01	0.02	0.07
6	0.0	0.04	0.04	0.0	0.07	0.07	0.76	0.0	0.02	0.0
7	0.0	0.07	0.03	0.0	0.03	0.0	0.0	0.75	0.02	0.11
8	0.02	0.02	0.03	0.12	0.02	0.08	0.0	0.01	0.62	0.09
9	0.01	0.01	0.0	0.03	0.1	0.02	0.0	0.02	0.01	0.8

Thus, by considering more features, the overall accuracy of the classifier increased by 0.4%. This result can be explained as the new classifier considers all of the 3 possible values distinctly and hence builds a model, which can predict well on the test data set for unknown labels.

Steps to execute

a. Navigate to Part_1.2 code folder
b. javac TerneryBonus.java
c. java TerneryBonus

Naive Bayes Classifier for face data

Implementation details: The code for digit classification was modified to take into account the changes in dimensions of the images and the values of the features. The number of objects was reduced from 10 to 2 since in this case there are only 2 possible classes into which the image can be classified.

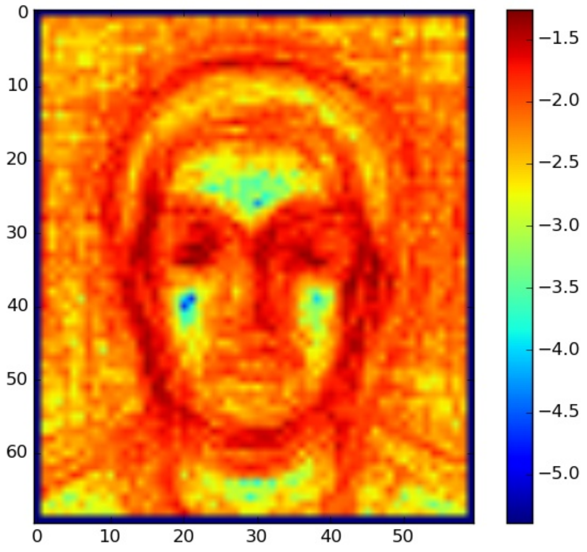
Accuracy: 90.67%

Classification Rate:
0 - Non-Face = 89%
1 - Is Face = 93%

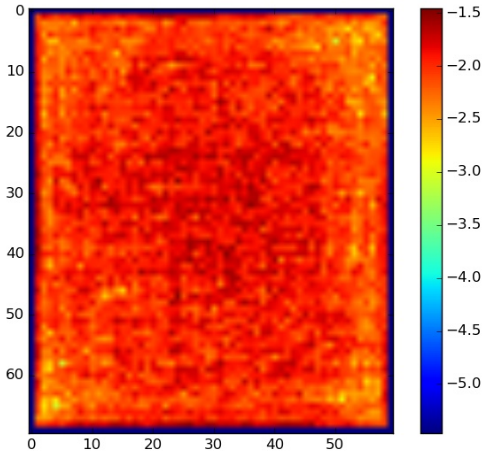
Confusion Matrix

	0	1
0	0.89	0.12
1	0.07	0.93

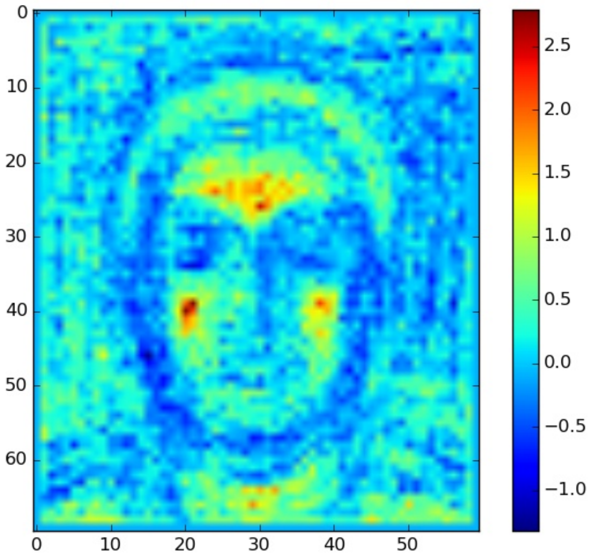
Log Likelihood ratio map for “is face”



Log likelihood ratio map for “not face”



Odds Ratio map



Steps to Execute:

1. Navigate to part_1.1 code folder
2. python mp3_facedata.py <training_images filepath> <training_labels filepath> <test_imges filepath> <test_labels filepath>

Using Pixel groups as features for face data classification

Disjoint

Pixel group	Accuracy
2 * 2	98%
2 * 4	98.67%
4 * 2	96.67%
4 * 4	96.67%

Overlapping

Pixel group	Accuracy
2 * 2	98%
2 * 4	98.67%
4 * 2	97.33%
4 * 4	98%
2 * 3	98%
3 * 2	98%
3 * 3	98.67%

The best accuracy was observed for Overlapping pixel grouping of size 2 * 4 = 98.67%

Steps to Execute for pixel grouping features for face data classification

1. Navigate to part_1.2 code folder
2. javac *.java
3. java FaceData <Type> <feature row size> <feature col size>
    a. Type = 0 for Disjoint, 1 for overlapping
    b. row size and col size of feature set. Ex: 2 * 2

REPORT – PART 2

Naïve Bayes Implementation

The code was implemented in Java, and it consisted of following steps:

1. Find all the unique words from the training data, and store it in a list.
2. Load the complete training and test data by reading the files, and storing it in appropriate data structures. Also loaded the training and test class labels.
3. Build a classifier model from the training data. The classifier stores the frequency count of each word for each of the classes.
4. Also compute the total frequency of all the words, for each of the classes.
5. Finally use the classifier to predict the labels for test data. Following approach was used to predict the labels:

   For each tuple in test data we compute the following probability for each class:

   $$P(tuple \mid class) = P(w_1, \ldots, w_n \mid class) = \prod_{i=1}^{n} P(w_i \mid class)$$

   Where:

   $$P(w_i \mid class) = \frac{Total\ count\ of\ word\ w_i\ in\ class\ 'class'}{Total\ count\ of\ all\ the\ words\ in\ class\ 'class'}$$

   For each tuple, the predicted class is the class, which gives maximum value for the probability P(tuple | class).

   Also, applied Laplace smoothing by using the following formula while calculating word probabilities for each class:

   $$P(w_i \mid class) = \frac{Total\ count\ of\ word\ w_i\ in\ class\ 'class' + 1}{Total\ count\ of\ all\ the\ words\ in\ class\ 'class' + Number\ of unique\ words}$$

SPAM DETECTION

How to run:
Code inside folder Part2/NaiveBayes
javac TextDocumentClassification.java

```
java TextDocumentClassification training_file test_file
```

Here training_file and test_file are the paths of training and test files of the spam data set. For spam dataset:

```
java TextDocumentClassification spam_detection/train_email.txt
spam_detection/test_email.txt
```

Accuracy: 0.9769

Classification Accuracy:
Classification Accuracy for class 0: 0.9692
Classification Accuracy for class 1: 0.9846

Confusion Matrix:

```
          0               1
--------------------------------------
0|      0.97            0.03
1|      0.02            0.98
```

Top 20 words for Class 0:
language: 0.0104
university: 0.0084
s: 0.0061
linguistic: 0.0044
de: 0.0041
information: 0.0041
conference: 0.0035
workshop: 0.0033
email: 0.0030
paper: 0.0030
e: 0.0029
english: 0.0029
one: 0.0026
please: 0.0026
include: 0.0026
edu: 0.0025
http: 0.0024
research: 0.0024
abstract: 0.0023

address: 0.0023

Top 20 words for Class 1:
email: 0.0108
s: 0.0094
order: 0.0090
report: 0.0082
our: 0.0075
address: 0.0074
mail: 0.0072
program: 0.0065
send: 0.0062
free: 0.0058
money: 0.0056
list: 0.0056
receive: 0.0052
name: 0.0049
business: 0.0048
one: 0.0043
d: 0.0042
work: 0.0041
com: 0.0041
nt: 0.0041

Highest Confusion Pair:
Class 0 & Class 1: 0.03

Highest Log odd Ratio words: Class 0 & 1
linguistic: 2.7515
workshop: 2.6296
abstract: 2.4769
theory: 2.3795
syntax: 2.2679
grammar: 2.1926
chair: 2.1213
discourse: 2.1134
translation: 2.0932
movement: 2.0932
computational: 2.0720
committee: 2.0677
semantic: 2.0633
benjamin: 2.0588

```
verb: 2.0263
context: 2.0065
programme: 2.0065
phonology: 1.9859
semantics: 1.9805
nl: 1.9697
```

EIGHT NEWSGROUPS

How to run:

Code inside folder Part2/NaiveBayes

```
javac TextDocumentClassification.java
java TextDocumentClassification training_file test_file
```

Here training_file and test_file are the paths of training and test files of the 8 newsgroups data set. For 8 newsgroups data set:

```
java TextDocumentClassification 8category/8category.training.txt
8category/8category.testing.txt
```

Accuracy: 0.9202

Classification Accuracies:

```
Classification Accuracy for class 0: 0.9706
Classification Accuracy for class 1: 0.8485
Classification Accuracy for class 2: 0.9722
Classification Accuracy for class 3: 0.8929
Classification Accuracy for class 4: 0.9787
Classification Accuracy for class 5: 0.4000
Classification Accuracy for class 6: 1.0000
Classification Accuracy for class 7: 0.8621
```

Confusion Matrix:

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0.97	0	0	0	0.03	0	0	0
1	0	0.85	0	0.12	0.03	0	0	0
2	0	0	0.97	0	0	0	0	0.03
3	0	0	0	0.89	0.04	0	0	0.07
4	0.02	0	0	0	0.98	0	0	0
5	0	0.4	0	0	0.1	0.4	0	0.1

6	0	0	0	0	0	0	1	0
7	0.03	0.03	0	0.07	0	0	0	0.86

Top 20 words for Class 0:
space: 0.0075
nt: 0.0043
would: 0.0041
one: 0.0028
launch: 0.0026
nasa: 0.0025
earth: 0.0024
subject: 0.0024
like: 0.0022
us: 0.0020
system: 0.0020
also: 0.0020
writes: 0.0020
could: 0.0019
time: 0.0018
first: 0.0018
data: 0.0018
orbit: 0.0018
edu: 0.0018
mission: 0.0018

Top 20 words for Class 1:
drive: 0.0058
scsi: 0.0049
nt: 0.0046
ide: 0.0036
one: 0.0031
card: 0.0030
drives: 0.0027
controller: 0.0027
disk: 0.0025
system: 0.0025
subject: 0.0024
use: 0.0024
would: 0.0024
edu: 0.0023
hard: 0.0023
bus: 0.0022
get: 0.0021

m: 0.0021
data: 0.0019
also: 0.0018

Top 20 words for Class 2:
nt: 0.0088
would: 0.0043
year: 0.0040
edu: 0.0039
writes: 0.0033
game: 0.0030
one: 0.0030
good: 0.0028
team: 0.0028
subject: 0.0027
last: 0.0027
article: 0.0027
think: 0.0027
players: 0.0026
like: 0.0025
baseball: 0.0024
games: 0.0023
better: 0.0023
well: 0.0021
time: 0.0021

Top 20 words for Class 3:
x: 0.0290
window: 0.0042
use: 0.0037
nt: 0.0035
subject: 0.0034
file: 0.0032
server: 0.0029
also: 0.0026
available: 0.0025
get: 0.0025
edu: 0.0023
motif: 0.0023
version: 0.0022
system: 0.0022
sun: 0.0021
program: 0.0021
c: 0.0021

```
one: 0.0020
m: 0.0020
windows: 0.0020
```

Top 20 words for Class 4:

```
nt: 0.0086
would: 0.0058
people: 0.0051
q: 0.0042
one: 0.0037
mr: 0.0037
think: 0.0035
president: 0.0034
writes: 0.0034
article: 0.0031
government: 0.0030
stephanopoulos: 0.0028
know: 0.0027
us: 0.0026
edu: 0.0025
like: 0.0025
subject: 0.0023
going: 0.0023
get: 0.0020
right: 0.0020
```

Top 20 words for Class 5:

```
new: 0.0030
edu: 0.0026
dos: 0.0022
sale: 0.0020
appears: 0.0020
art: 0.0019
subject: 0.0018
wolverine: 0.0018
shipping: 0.0016
cover: 0.0016
price: 0.0016
one: 0.0015
list: 0.0015
comics: 0.0015
drive: 0.0015
nt: 0.0015
hulk: 0.0014
```

good: 0.0014
vs: 0.0014
system: 0.0013

Top 20 words for Class 6:
nt: 0.0066
game: 0.0051
team: 0.0050
hockey: 0.0044
would: 0.0032
play: 0.0029
subject: 0.0027
period: 0.0027
season: 0.0026
nhl: 0.0026
games: 0.0025
one: 0.0025
first: 0.0023
year: 0.0022
think: 0.0022
players: 0.0021
la: 0.0021
get: 0.0021
edu: 0.0020
like: 0.0020

Top 20 words for Class 7:
image: 0.0074
jpeg: 0.0039
edu: 0.0036
nt: 0.0035
file: 0.0034
images: 0.0032
data: 0.0031
also: 0.0030
graphics: 0.0029
software: 0.0026
available: 0.0025
use: 0.0024
one: 0.0021
program: 0.0021
files: 0.0020
format: 0.0020

get: 0.0019
version: 0.0018
system: 0.0018
would: 0.0018

Highest Confusion Pairs:
Class 5 & Class 1: 0.40
Class 1 & Class 3: 0.12
Class 5 & Class 7: 0.10
Class 5 & Class 4: 0.10

Highest Log odd Ratio words: Class 5 & 1
wolverine: 2.1775
comics: 2.1083
hulk: 2.0881
annual: 1.9912
spiderman: 1.9534
ghost: 1.9303
liefeld: 1.9243
sabretooth: 1.9121
rider: 1.9121
condition: 1.8995
app: 1.8523
bagged: 1.8451
hobgoblin: 1.8304
xforce: 1.8228
guide: 1.7993
punisher: 1.7745
sale: 1.7542
edition: 1.7482
marvel: 1.7297
ufl: 1.7004

Highest Log odd Ratio words: Class 1 & 3
ide: 2.6506
controller: 2.5252
bios: 2.3064
scsi: 2.3064
drives: 2.0537
master: 2.0442
slave: 2.0327
adaptec: 2.0267
rom: 2.0267

dma: 1.9830
vlb: 1.9763
bus: 1.9651
hd: 1.9038
diamond: 1.8794
irq: 1.8447
maxtor: 1.8355
compos: 1.8262
cable: 1.8262
connector: 1.7969
chip: 1.7969

Highest Log odd Ratio words: Class 5 & 7
wolverine: 2.3304
comics: 2.2612
hulk: 2.2410
spiderman: 2.1063
liefeld: 2.0771
sabretooth: 2.0649
app: 2.0051
bagged: 1.9980
hobgoblin: 1.9832
xforce: 1.9757
punisher: 1.9274
cable: 1.9010
marvel: 1.8826
panther: 1.8326
mutants: 1.8109
comic: 1.7880
pom: 1.7880
mint: 1.7761
bwsmith: 1.7639
rider: 1.7639

Highest Log odd Ratio words: Class 5 & 4
dos: 2.5566
wolverine: 2.4604
shipping: 2.4217
comics: 2.3912
spiderman: 2.2363
liefeld: 2.2071
sabretooth: 2.1949
rider: 2.1949
bagged: 2.1280

```
cd: 2.1132
hobgoblin: 2.1132
xforce: 2.1057
hulk: 2.0700
disks: 2.0658
punisher: 2.0574
unix: 2.0400
marvel: 2.0126
ufl: 1.9833
panther: 1.9626
mutants: 1.9409
```

BONUS POINTS

I. ADVANCED TECHNIQUES:

Implemented Stop words Removal, and Stemming of words to improve the accuracy of the results obtained.

Stop words: In computing, stop words are words which are filtered out before or after processing of natural language data (text).
Stemming: It is the term used in linguistic morphology and information retrieval to describe the process for reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form.

Implementation:

1. For stop words removal we downloaded a stop words list from the Internet. We filter out any of the words present in the stop words list.
2. For stemming we are using the Lucene Snowball Porter Stemmer library, which returns the stemmed form, given a word.

How to Run:

```
javac -cp lucene-analyzers-common-5.0.0.jar StopWordsStemming.java

java -cp lucene-analyzers-common-5.0.0.jar: StopWordsStemming
"spam_detection/train_email.txt" "spam_detection/test_email.txt"
```

The code depends on **lucene-analyzers-common-5.0.0.jar** file (lucene library used for word stemming), and the **stop-word-list.txt**, which is a list of stop words.

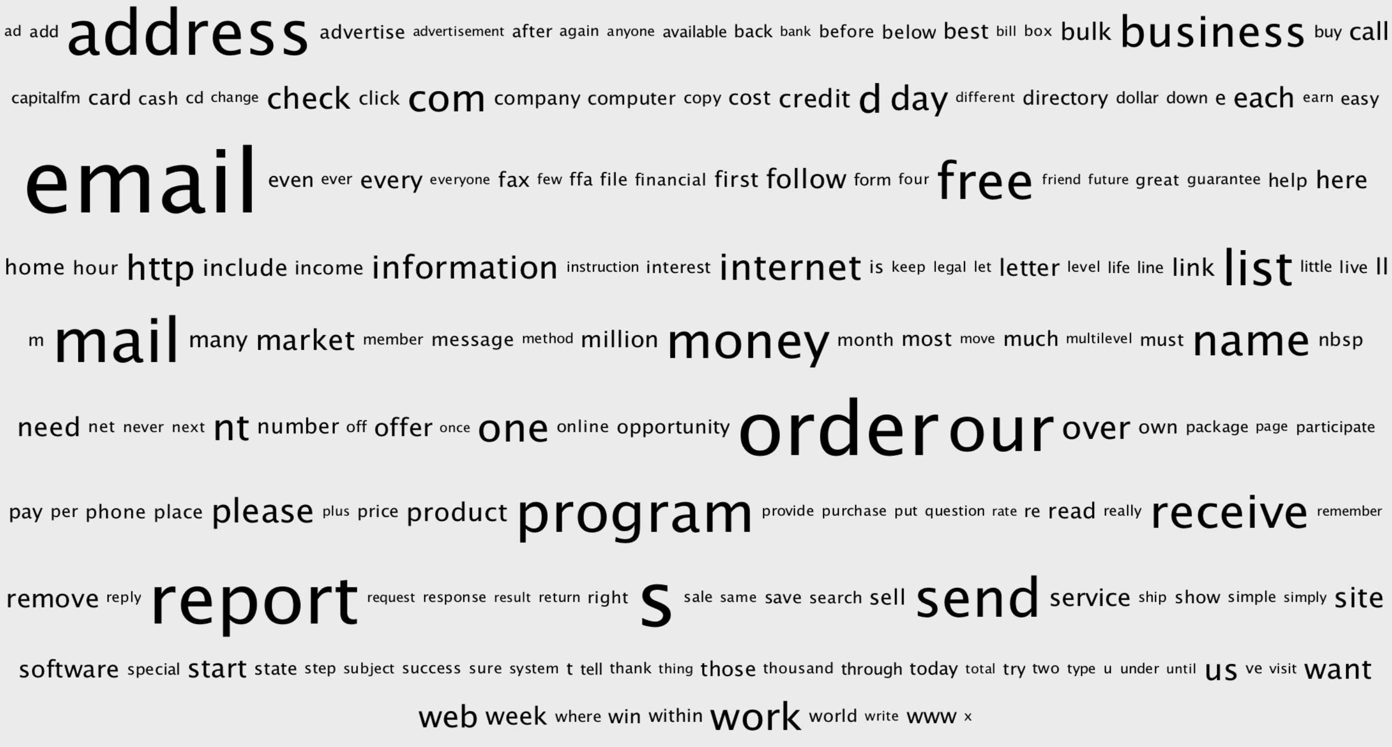
Results:

1. Spam Detection dataset: The accuracy increases slightly from 0.9769 to **0.9808**.
2. Eight Newsgroup dataset: The accuracy increases slightly from 0.9202 to **0.9240**.

II. WORD CLOUD

Implemented the word cloud using the Open Cloud library in Java http://opencloud.mcavallo.org/. It is displayed using the java swing framework, where the size and boldness of each word denote its importance. The program takes a file (training or test file from any of the data sets) and class number as inputs. It will display word cloud of **top 200 words** frequent in that particular class in that file.

For example: Following is the word cloud obtained for spam detection training data set for class 1 i.e SPAM.



From the above word cloud we can observe that highly used words are email, address, free, order, send, receive etc. It makes very much sense for these words to be frequent in Spam messages because in a lot of spam messages they might ask for ‘address’ information, or they might messages offering you ‘free’ money etc., or might ask you to ‘order’ or ‘send/receive’ something.

How to Run:

```
javac -cp opencloud.jar WordCloud.java
java -cp opencloud.jar: WordCloud file_path class_label
```

For example:

```
java -cp opencloud.jar: WordCloud "spam_detection/train_email.txt" 1
```

This would give the word cloud for spam detection training file, for class label 1 that is ‘Spam’ class.

III. 20 NEWSGROUP DATASET:

Implemented a program which converts the 20 newsgroup dataset ‘20news-bydate-matlab’, and converts the data from train.data, train.label, test.data and test.label into the same format as is expected by our code which we wrote for spam detection and 8 category newsgroup dataset. It outputs two file training.txt and test.txt, which can then be run using our naïve bayes classification code.

How to Run:

```
javac NewsData.java
java NewsData "20news-bydate/matlab/"
```

It will produce two files **20news-train.txt** and **20news-test.txt** files, which are in the format, which can be used by our naïve bayes implementation. To get the results:

```
javac TextDocumentClassification.java
java TextDocumentClassification "20news-train.txt" "20news-test.txt"
```

RESULTS:

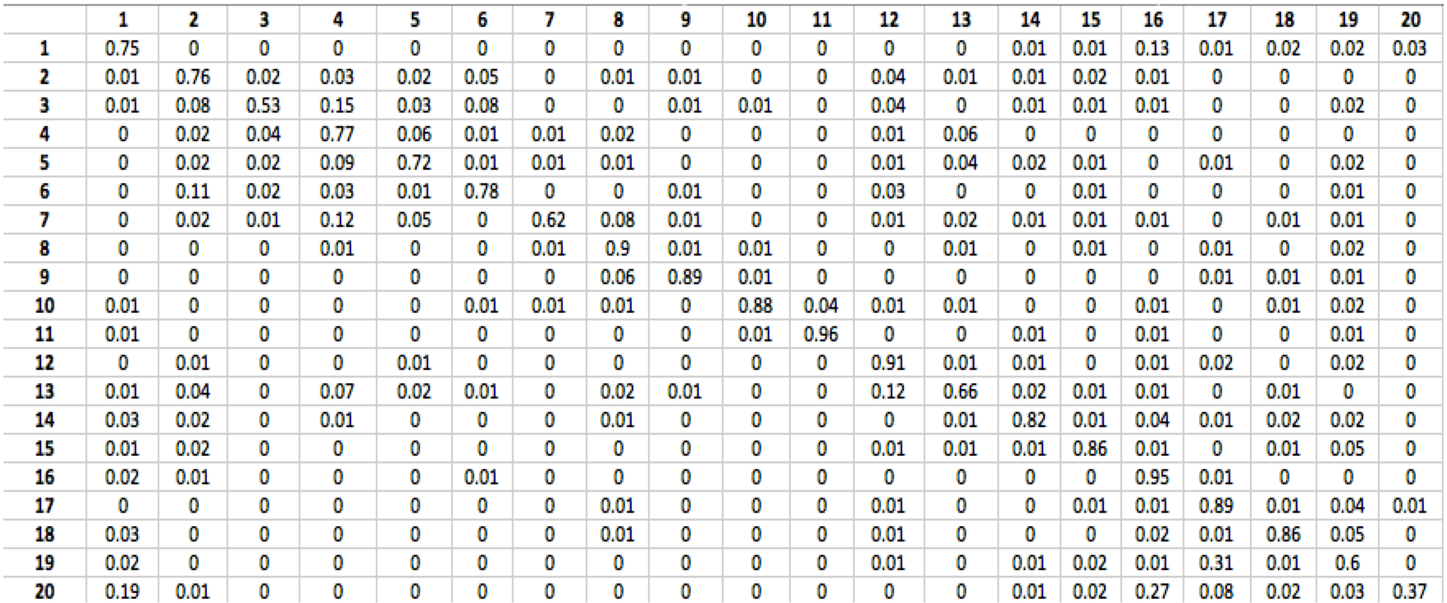
Accuracy: 0.7847

Classification Accuracy:

```
Classification Accuracy for class 1: 0.7547
Classification Accuracy for class 2: 0.7609
Classification Accuracy for class 3: 0.5345
Classification Accuracy for class 4: 0.7730
Classification Accuracy for class 5: 0.7232
Classification Accuracy for class 6: 0.7821
Classification Accuracy for class 7: 0.6152
Classification Accuracy for class 8: 0.9013
Classification Accuracy for class 9: 0.8892
Classification Accuracy for class 10: 0.8766
Classification Accuracy for class 11: 0.9599
Classification Accuracy for class 12: 0.9139
Classification Accuracy for class 13: 0.6616
Classification Accuracy for class 14: 0.8244
Classification Accuracy for class 15: 0.8571
Classification Accuracy for class 16: 0.9472
Classification Accuracy for class 17: 0.8901
```

Classification Accuracy for class 18: 0.8644
Classification Accuracy for class 19: 0.5968
Classification Accuracy for class 20: 0.3665

Confusion Matrix:



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0.75	0	0	0	0	0	0	0	0	0	0	0	0	0.01	0.01	0.13	0.01	0.02	0.02	0.03
2	0.01	0.76	0.02	0.03	0.02	0.05	0	0.01	0.01	0	0	0.04	0.01	0.01	0.02	0.01	0	0	0	0
3	0.01	0.08	0.53	0.15	0.03	0.08	0	0	0.01	0.01	0	0.04	0	0.01	0.01	0.01	0	0	0.02	0
4	0	0.02	0.04	0.77	0.06	0.01	0.01	0.02	0	0	0	0.01	0.06	0	0	0	0	0	0	0
5	0	0.02	0.02	0.09	0.72	0.01	0.01	0.01	0	0	0	0.01	0.04	0.02	0.01	0	0.01	0	0.02	0
6	0	0.11	0.02	0.03	0.01	0.78	0	0	0.01	0	0	0.03	0	0	0.01	0	0	0	0.01	0
7	0	0.02	0.01	0.12	0.05	0	0.62	0.08	0.01	0	0	0.01	0.02	0.01	0.01	0.01	0	0.01	0.01	0
8	0	0	0	0.01	0	0	0.01	0.9	0.01	0.01	0	0	0.01	0	0.01	0	0.01	0	0.02	0
9	0	0	0	0	0	0	0	0.06	0.89	0.01	0	0	0	0	0	0	0.01	0.01	0.01	0
10	0.01	0	0	0	0	0.01	0.01	0.01	0	0.88	0.04	0.01	0.01	0	0	0.01	0	0.01	0.02	0
11	0.01	0	0	0	0	0	0	0	0	0.01	0.96	0	0	0.01	0	0.01	0	0	0.01	0
12	0	0.01	0	0	0.01	0	0	0	0	0	0	0.91	0.01	0.01	0	0.01	0.02	0	0.02	0
13	0.01	0.04	0	0.07	0.02	0.01	0	0.02	0.01	0	0	0.12	0.66	0.02	0.01	0.01	0	0.01	0	0
14	0.03	0.02	0	0.01	0	0	0	0.01	0	0	0	0	0.01	0.82	0.01	0.04	0.01	0.02	0.02	0
15	0.01	0.02	0	0	0	0	0	0	0	0	0	0.01	0.01	0.01	0.86	0.01	0	0.01	0.05	0
16	0.02	0.01	0	0	0	0.01	0	0	0	0	0	0	0	0	0	0.95	0.01	0	0	0
17	0	0	0	0	0	0	0	0.01	0	0	0	0.01	0	0	0.01	0.01	0.89	0.01	0.04	0.01
18	0.03	0	0	0	0	0	0	0.01	0	0	0	0.01	0	0	0	0.02	0.01	0.86	0.05	0
19	0.02	0	0	0	0	0	0	0	0	0	0	0.01	0	0.01	0.02	0.01	0.31	0.01	0.6	0
20	0.19	0.01	0	0	0	0	0	0	0	0	0	0	0	0.01	0.02	0.27	0.08	0.02	0.03	0.37

Highest Confusion Pairs:
Class 19 & Class 17: 0.31
Class 20 & Class 16: 0.27
Class 20 & Class 1: 0.19
Class 3 & Class 4: 0.15

Highest Log odd Ratio words: Class 19 & 17
stephanopoulos: 2.5197
package: 2.1449
br: 2.0598
optilink: 2.0598
isc: 2.0598
russian: 1.9396
cramer: 1.9049
kaldis: 1.8439
vat: 1.8318
thor: 1.8001
steveh: 1.8001
deficit: 1.7799
rutgers: 1.7659
myers: 1.7551

gay: 1.7402
naval: 1.7209
clayton: 1.7170
rockefeller: 1.7130
stimulus: 1.7130
hendricks: 1.6966

Highest Log odd Ratio words: Class 20 & 16
sandvik: 2.3241
ra: 2.2062
tyre: 1.8847
weiss: 1.8408
brian: 1.8219
decenso: 1.7369
malcolm: 1.7250
tony: 1.7128
kendig: 1.7128
convenient: 1.6872
dwyer: 1.6872
batf: 1.6739
hanging: 1.6739
buffalo: 1.6739
royalroads: 1.6458
gb: 1.6458
alink: 1.6458
ksand: 1.6458
mcconkie: 1.6001
muhammad: 1.6001

Highest Log odd Ratio words: Class 20 & 1
hudson: 1.8092
lds: 1.7678
weiss: 1.7409
magi: 1.6486
decenso: 1.6370
malcolm: 1.6251
beast: 1.6003
batf: 1.5740
je: 1.5602
ye: 1.5602
royalroads: 1.5459
lee: 1.5459
fbi: 1.5237
mcconkie: 1.5002

```
iniquity: 1.4838
db: 1.4490
utarlg: 1.4490
uta: 1.4490
psyrobtw: 1.4305
caligiuri: 1.4112
```

Highest Log odd Ratio words: Class 3 & 4

```
ei: 2.1706
di: 1.9929
risc: 1.8995
um: 1.8726
lq: 1.8569
tt: 1.7726
allocation: 1.7726
paradox: 1.7568
dy: 1.7487
el: 1.7404
wg: 1.7320
lw: 1.7057
qu: 1.7057
rlk: 1.7057
bmp: 1.6965
cx: 1.6872
mk: 1.6477
apps: 1.6372
yd: 1.6155
umu: 1.6042
```

Top 20 words for Class 1

```
the: 0.0353
to: 0.0206
of: 0.0201
is: 0.0171
that: 0.0155
and: 0.0135
in: 0.0133
it: 0.0113
you: 0.0096
not: 0.0078
be: 0.0064
are: 0.0064
this: 0.0060
for: 0.0058
```

have: 0.0057
as: 0.0050
but: 0.0046
or: 0.0044
if: 0.0042
on: 0.0039

Top 20 words for Class 2
the: 0.0273
to: 0.0160
and: 0.0143
of: 0.0134
is: 0.0106
in: 0.0100
for: 0.0091
it: 0.0089
you: 0.0064
that: 0.0061
on: 0.0053
this: 0.0051
or: 0.0045
be: 0.0044
with: 0.0042
have: 0.0040
edu: 0.0039
can: 0.0039
are: 0.0037
if: 0.0036

Top 20 words for Class 3
the: 0.0258
to: 0.0146
and: 0.0103
is: 0.0099
it: 0.0098
of: 0.0089
in: 0.0084
for: 0.0070
that: 0.0066
windows: 0.0065
you: 0.0061
have: 0.0055
with: 0.0047
on: 0.0045

this: 0.0044
edu: 0.0043
be: 0.0038
or: 0.0038
but: 0.0037
if: 0.0036

Top 20 words for Class 4
the: 0.0316
to: 0.0151
and: 0.0125
is: 0.0102
of: 0.0097
it: 0.0096
in: 0.0081
for: 0.0073
that: 0.0069
with: 0.0064
on: 0.0059
you: 0.0056
have: 0.0053
this: 0.0052
scsi: 0.0044
drive: 0.0044
or: 0.0041
my: 0.0040
can: 0.0038
be: 0.0037

Top 20 words for Class 5
the: 0.0334
to: 0.0150
and: 0.0119
of: 0.0101
is: 0.0099
in: 0.0089
it: 0.0085
that: 0.0073
for: 0.0072
with: 0.0060
on: 0.0053
have: 0.0052
you: 0.0052
this: 0.0048

be: 0.0040
if: 0.0038
edu: 0.0038
but: 0.0037
not: 0.0036
or: 0.0036

Top 20 words for Class 6
the: 0.0372
to: 0.0194
and: 0.0133
of: 0.0131
is: 0.0128
in: 0.0110
for: 0.0078
it: 0.0074
that: 0.0065
this: 0.0063
on: 0.0063
you: 0.0060
be: 0.0052
are: 0.0048
if: 0.0047
with: 0.0046
not: 0.0043
or: 0.0043
can: 0.0042
an: 0.0041

Top 20 words for Class 7
the: 0.0150
for: 0.0112
and: 0.0107
to: 0.0098
of: 0.0078
in: 0.0067
it: 0.0052
is: 0.0052
you: 0.0051
or: 0.0044
with: 0.0042
have: 0.0037
edu: 0.0033
are: 0.0032

if: 0.0030
this: 0.0029
all: 0.0028
new: 0.0027
that: 0.0025
me: 0.0025

Top 20 words for Class 8
the: 0.0358
to: 0.0156
and: 0.0136
in: 0.0124
of: 0.0124
it: 0.0097
is: 0.0096
that: 0.0086
you: 0.0081
for: 0.0068
on: 0.0063
have: 0.0055
car: 0.0050
are: 0.0049
this: 0.0044
with: 0.0043
my: 0.0042
be: 0.0042
not: 0.0041
they: 0.0041

Top 20 words for Class 9
the: 0.0326
to: 0.0160
and: 0.0126
of: 0.0119
in: 0.0114
it: 0.0093
you: 0.0080
is: 0.0079
that: 0.0075
for: 0.0066
on: 0.0065
my: 0.0051
com: 0.0042
with: 0.0041

this: 0.0039
have: 0.0038
was: 0.0038
or: 0.0034
bike: 0.0033
writes: 0.0032

Top 20 words for Class 10
the: 0.0340
to: 0.0143
in: 0.0133
and: 0.0128
of: 0.0114
that: 0.0087
is: 0.0086
he: 0.0071
for: 0.0059
it: 0.0057
edu: 0.0052
have: 0.0047
you: 0.0046
this: 0.0045
be: 0.0045
but: 0.0044
on: 0.0043
was: 0.0043
they: 0.0042
with: 0.0040

Top 20 words for Class 11
the: 0.0408
to: 0.0150
in: 0.0137
and: 0.0124
of: 0.0119
that: 0.0076
is: 0.0072
for: 0.0068
it: 0.0062
on: 0.0055
he: 0.0053
was: 0.0053
you: 0.0050
have: 0.0043

be: 0.0041
but: 0.0039
they: 0.0037
with: 0.0036
this: 0.0036
at: 0.0036

Top 20 words for Class 12
the: 0.0442
to: 0.0236
of: 0.0192
and: 0.0170
is: 0.0137
in: 0.0119
that: 0.0105
it: 0.0094
be: 0.0080
for: 0.0077
this: 0.0069
you: 0.0056
on: 0.0055
are: 0.0054
with: 0.0048
as: 0.0047
not: 0.0047
or: 0.0046
key: 0.0046
they: 0.0042

Top 20 words for Class 13
the: 0.0348
to: 0.0174
of: 0.0127
and: 0.0115
is: 0.0109
in: 0.0102
it: 0.0080
you: 0.0076
that: 0.0074
for: 0.0074
on: 0.0052
are: 0.0051
be: 0.0049
this: 0.0048

have: 0.0046
or: 0.0046
with: 0.0045
if: 0.0041
can: 0.0033
as: 0.0032

Top 20 words for Class 14
the: 0.0325
of: 0.0205
to: 0.0186
and: 0.0160
in: 0.0141
is: 0.0131
that: 0.0103
it: 0.0099
for: 0.0068
this: 0.0057
you: 0.0056
are: 0.0054
be: 0.0052
not: 0.0051
with: 0.0048
have: 0.0045
or: 0.0042
edu: 0.0041
as: 0.0041
on: 0.0040

Top 20 words for Class 15
the: 0.0393
of: 0.0186
to: 0.0182
and: 0.0156
in: 0.0130
is: 0.0099
that: 0.0080
for: 0.0078
it: 0.0074
on: 0.0063
space: 0.0057
be: 0.0054
you: 0.0052
this: 0.0050

are: 0.0044
as: 0.0039
have: 0.0037
with: 0.0036
was: 0.0036
at: 0.0036

Top 20 words for Class 16
the: 0.0435
of: 0.0239
to: 0.0236
that: 0.0172
and: 0.0171
is: 0.0166
in: 0.0147
it: 0.0102
not: 0.0083
you: 0.0078
this: 0.0070
be: 0.0070
are: 0.0067
for: 0.0066
as: 0.0063
we: 0.0057
god: 0.0057
have: 0.0057
but: 0.0052
he: 0.0049

Top 20 words for Class 17
the: 0.0432
to: 0.0213
of: 0.0206
and: 0.0155
in: 0.0144
that: 0.0121
is: 0.0102
you: 0.0086
it: 0.0082
for: 0.0067
this: 0.0056
have: 0.0054
are: 0.0053
be: 0.0053

they: 0.0052
not: 0.0049
on: 0.0048
as: 0.0047
or: 0.0045
was: 0.0041

Top 20 words for Class 18
the: 0.0510
of: 0.0252
to: 0.0220
and: 0.0212
in: 0.0180
that: 0.0125
is: 0.0092
it: 0.0086
you: 0.0082
they: 0.0071
was: 0.0067
for: 0.0063
not: 0.0061
are: 0.0057
on: 0.0056
were: 0.0049
by: 0.0048
as: 0.0048
from: 0.0044
with: 0.0044

Top 20 words for Class 19
the: 0.0438
to: 0.0245
of: 0.0200
and: 0.0162
that: 0.0156
in: 0.0146
is: 0.0113
it: 0.0102
you: 0.0085
for: 0.0070
not: 0.0059
this: 0.0059
are: 0.0058
have: 0.0057

on: 0.0056
be: 0.0056
we: 0.0053
as: 0.0050
they: 0.0045
with: 0.0044

Top 20 words for Class 20
the: 0.0380
of: 0.0206
to: 0.0188
and: 0.0159
that: 0.0137
in: 0.0126
is: 0.0122
you: 0.0091
it: 0.0081
not: 0.0066
for: 0.0053
are: 0.0053
this: 0.0052
be: 0.0050
as: 0.0049
have: 0.0046
with: 0.0041
on: 0.0037
they: 0.0037
was: 0.0037

Statement of Individual Contribution

Udit Mehrotra (umehrot2)

- Implemented the complete code for Part 2 of the assignment.
- Implemented the entire bonus / extra credit portion of part 2 – lemmatization, 20 news groups dataset and Word cloud.

Sanjana Chandrashekar (chndrsh4)

- Implemented Naïve Bayes Classifier along with various smoothing values, odds ratio for Part 1.1 and 1.2.
- For bonus point implemented face data classification using Naïve bayes.
- Report for Part 1

Suhas Hoskote Muralidhar (shmural2)

- Implemented Naïve Bayes Classifier along with laplace smoothing for Part 1.1 and 1.2.
- For bonus point implemented Ternary feature and face data classification
- Report for Part 1