Assignment 3: Naive Bayes Classification

CS 440 Fall 2015 3 Credits

Name	Email	Contributions
Jaehwan "JJ" Park	park155@illinois.edu	part 2, report
Ruifan Li	li158@illinois.edu	part 1, 2, report

1.1 Digit Clarification

Implementation

The basic idea for this part is to estimate the likelihood of each pixel of the 28x28 pixel image of digit 0~9 by counting 4000 training images. Then with the data base built, the classifier can calculate the posterior of a given image by summing up the natural log of the likelihoods for each pixels for every class and determine which class among 0~9 is the given image most likely be. (Naive Bayes Model) (Part1 > identify_number.java)

Algorithm outlines:

- Training:
    - Read the training file, save it to an ArrayList called ***stored_training_data*** (each line as one string element)
    - Read the training label file, save the data to an integer ArrayList ***stored_training_label***
    - Create a function ***get_next_image*** to read the top 28 lines from the ArrayList ***stored_training_data*** and save the chars from it to a 2D char array, delete the read file from ***stored_training_data***, count the times the function is called and and return the 2D array

- Count the value of each pixel (either 1 for colored, 0 for blank) in the 2D char array for a certain number of training samples (**4000 in our case**), added up and divided by the total number of image of that class, with Laplacian smooth factor:

$$P_{colored\_likelihood}\ (F_{ij} = 1|class) = \frac{Total\ number\ of\ colored\ pixel\ at\ this\ position(i,j) + smooth\ factor}{Total\ number\ of\ image\ of\ this\ calss + smooth\ factor \times 2}$$

$$P_{blank\_likelihood}\ (F_{ij} = 0|class) = \frac{Total\ number\ of\ colored\ pixel\ at\ this\ position(i,j) + smooth\ factor}{Total\ number\ of\ image\ of\ this\ calss + smooth\ factor \times 2}$$

(In our case, we tuned the **smooth factor to be 1** so that the classifier will give the best classification rate)

- Keep track of number of training samples for each class of 0~9, and calculate prior by

$$P_{prior(class)} = \frac{total\ number\ of\ samples\ in\ this\ class}{total\ number\ of\ samples}$$

- Testing:
  - Read the test file, save it to an ArrayList called ***stored_test_data*** (each line as one string element)
  - Read the test label file, save the data to an integer ArrayList ***stored_test_label***
  - Use the ***get_next_image*** function to read an image from the ***stored_test_data***, until there's nothing left in the ArrayList, and calculate the posterior according to the value of each pixel for each class by

$$P_{posterior}(class|F_{ij}) = \ln\ (P_{prior}(class)) + \sum_{i,j} \ln P_{likelihood}(F_{ij}|class)$$

  - Pick up the class with the highest posterior for each image, and save it to an integer ArrayList ***test_result***
  - Compare the elements in ***test_result*** and ***stored_test_label***, and then calculate the classification rate for each class, test sample with highest and lowest posterior in each class and the confusion matrix

- Odd Pair:
    - According to the result from testing, pick out 4 most confused pair from the confusion matrix (4,9) (5,3) (7,9) (8,3)
    - For each digit, find out the likelihood for colored pixel calculate for each pixel, plot each pixel with [+] for likelihood larger than -0.85, [**.**] for likelihood between [-4,-0.85], and [ ] for likelihood less than -4
    - To obtain the odd pair, take the first digit's likelihood of colored pixel divided by that of the second digit, for each of the 28x28 pixel. Then plot each position with [+] for ratio larger than 0.5 (first digit's character), [ ] for likelihood between [-0.5,0.5] (common character), and [-] for likelihood less than –0.5 (second digit's character)

Results

Classification Rate:

Table1. Classification Rate for each digit

Number	Classification Rate
0	**0.8444444444444444**
1	**0.9629629629629629**
2	**0.7766990291262136**
3	**0.8**
4	**0.7663551401869159**
5	**0.6521739130434783**
6	**0.7142857142857143**
7	**0.7264150943396226**
8	**0.6019417475728155**
9	**0.8**

Confusion Matrix:

Table2. Confusion matrix for each digit

	0	1	2	3	4	5	6	7	8	9
0	0.844	0.000	0.011	0.000	0.011	0.056	0.033	0.000	0.044	0.000
1	0.000	0.963	0.000	0.000	0.000	0.019	0.009	0.000	0.009	0.000
2	0.010	0.029	0.777	0.039	0.010	0.000	0.058	0.010	0.049	0.019
3	0.000	0.010	0.000	0.800	0.000	0.030	0.020	0.060	0.020	0.060
4	0.000	0.009	0.009	0.000	0.766	0.000	0.028	0.009	0.019	0.159
5	0.022	0.022	0.011	0.152	0.033	0.652	0.011	0.011	0.022	0.065
6	0.022	0.044	0.088	0.000	0.055	0.066	0.714	0.000	0.011	0.000
7	0.000	0.057	0.028	0.000	0.028	0.000	0.000	0.726	0.019	0.142
8	0.019	0.010	0.029	0.126	0.019	0.058	0.000	0.010	0.602	0.126
9	0.010	0.000	0.010	0.030	0.100	0.020	0.000	0.020	0.010	0.800

(Green for accuracy; Red for odds ratio comparison)

Samples with the highest and lowest posterior probabilities from each digit:

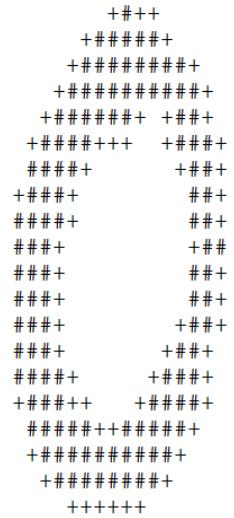

Figure1. Best 0

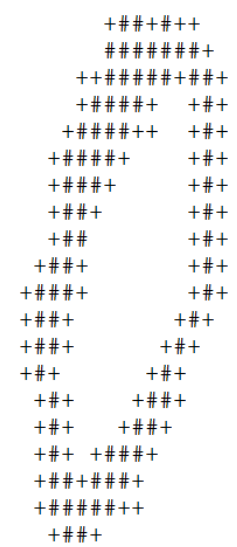

Figure2. Worst 0

Figure3. Best 1

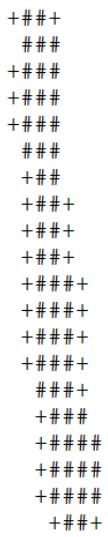

Figure4. Worst 1

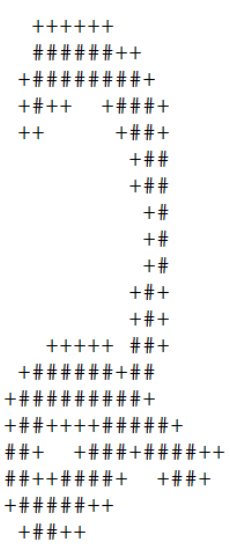

Figure5. Best 2

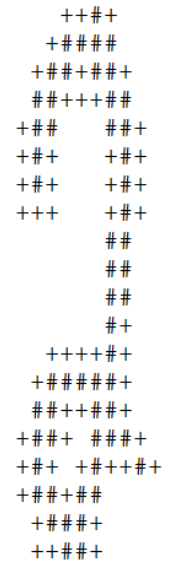

Figure6. Worst 2

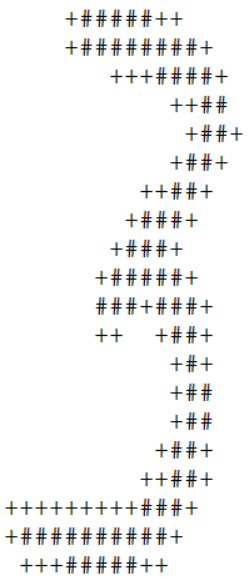

Figure7. Best 3

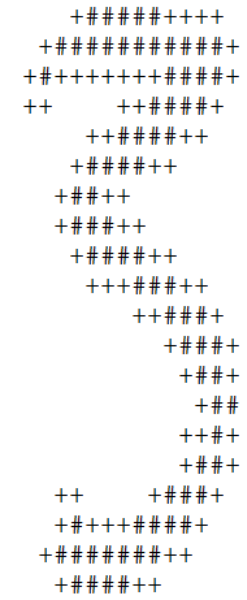

Figure8. Worst

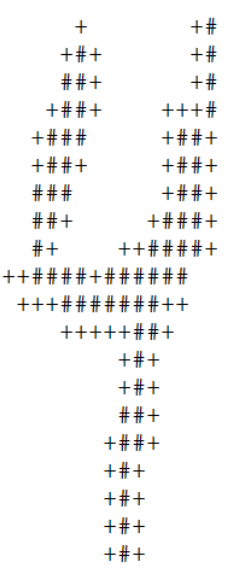

Figure9. Best 4

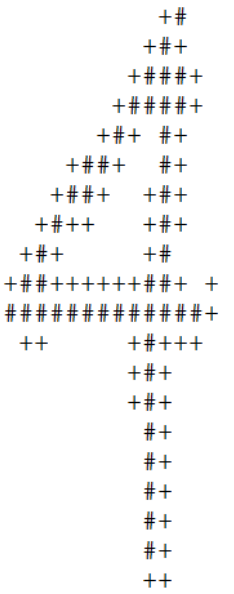

Figure10. Worst 4

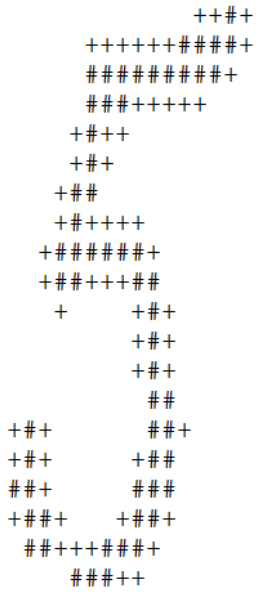

Figure11. Best 5

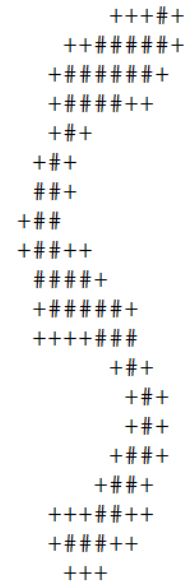

Figure12. Worst 5

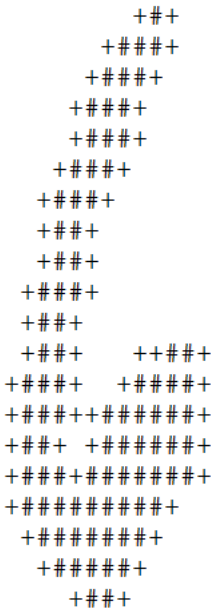

Figure13. Best 6

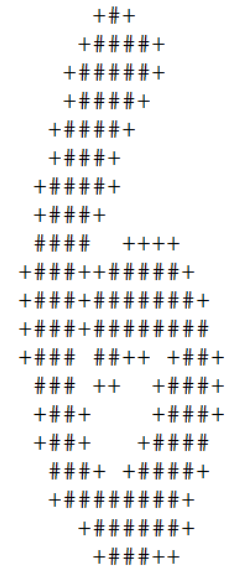

Figure14. Worst 6

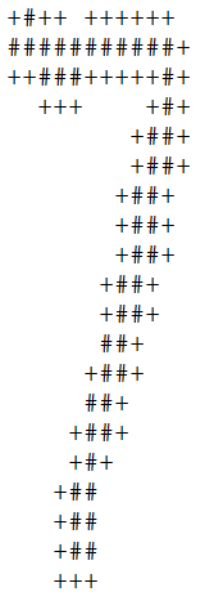

Figure15. Best 7

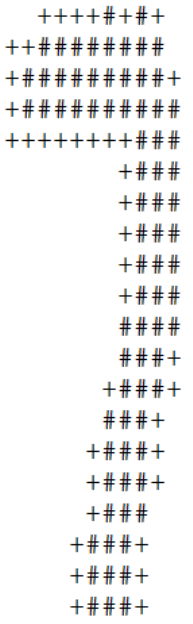

Figure16. Worst 7

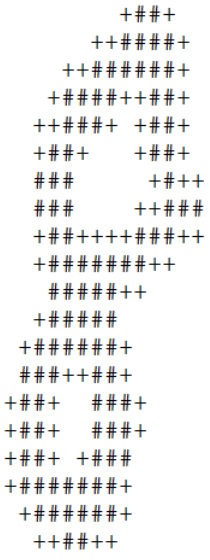

Figure17. Best 8

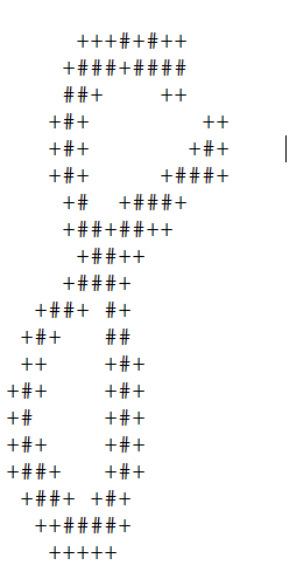

Figure18. Worst 8

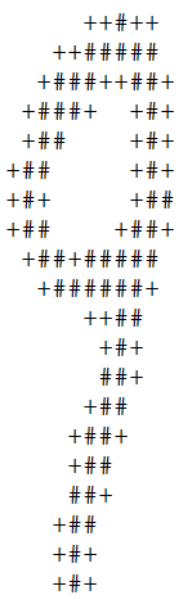

Figure19. Best 9

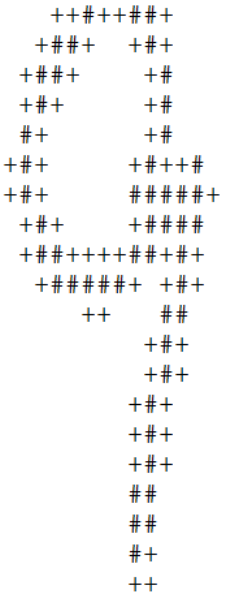

Figure20. Worst 9

Odd Pairs:

The pair chosen here, as indicated in the confusion matrix is (4,9) (5,3) (7,9) (8,3).

For all the feature likelihood figures below:

[+] high probability [.] medium probability [ ] low probability

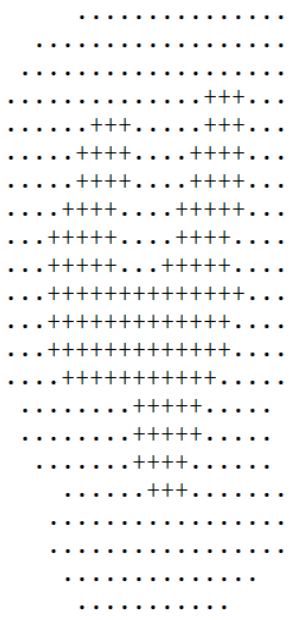

Figure 21. 4's Feature Likelihood

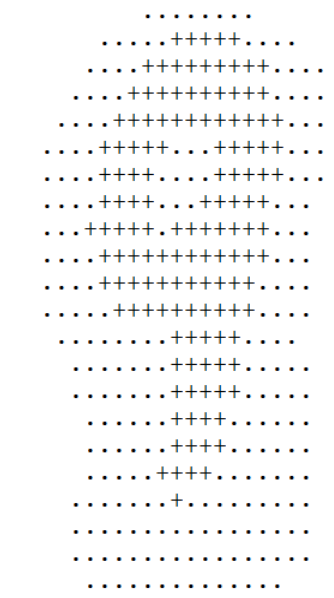

Figure 22. 9's Feature Likelihood

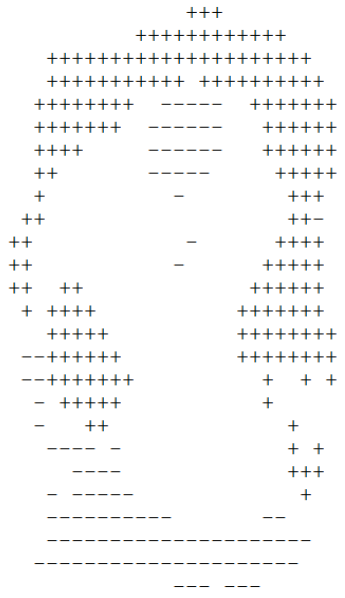

Figure 23. 4&9 Odd Pairs

([+] for 4's characteristics [ ] for common value [-] for 9' characteristics)

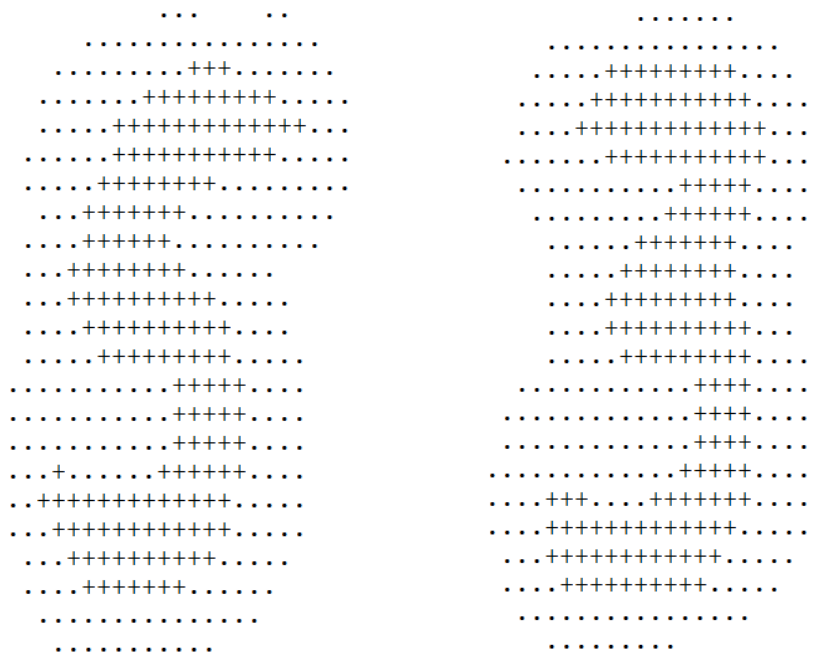

Figure 24. 5's Feature Likelihood

Figure 25. 3's Feature Likelihood

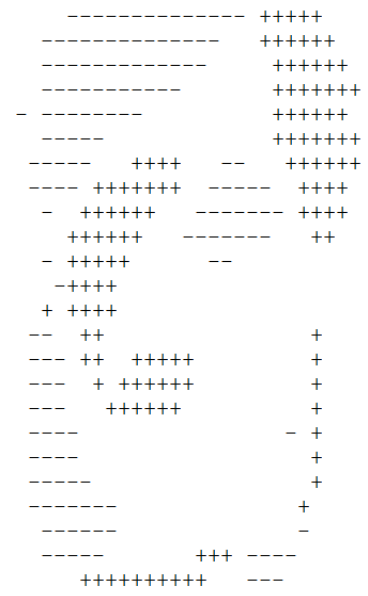

Figure 26. 5&3 Odd Pairs

([+] for 5's characteristics [ ] for common value [-] for 3' characteristics)

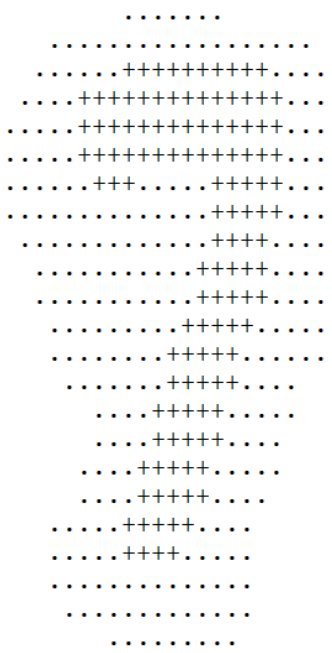

Figure 27. 7's Feature Likelihood

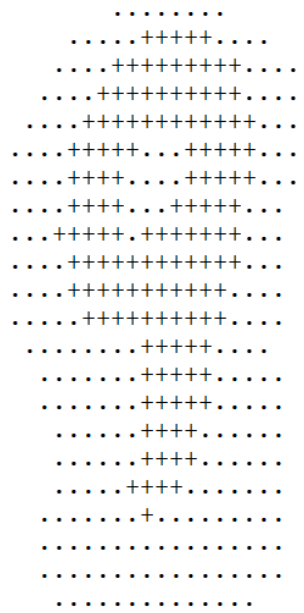

Figure 28. 9's Feature Likelihood

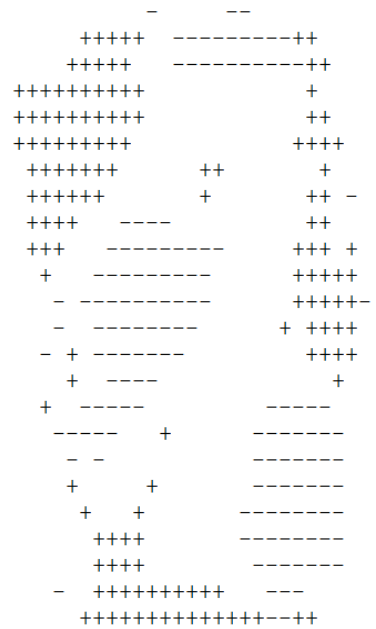

Figure 29. 7&9 Odd Pairs

([+] for 7's characteristics [ ] for common value [-] for 9' characteristics)

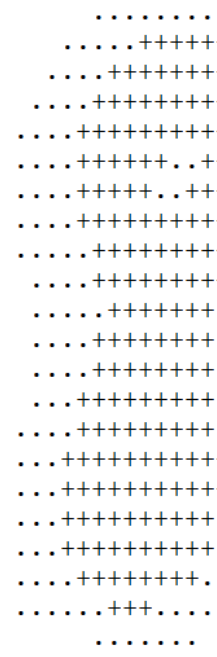

Figure 30. 8's Feature Likelihood

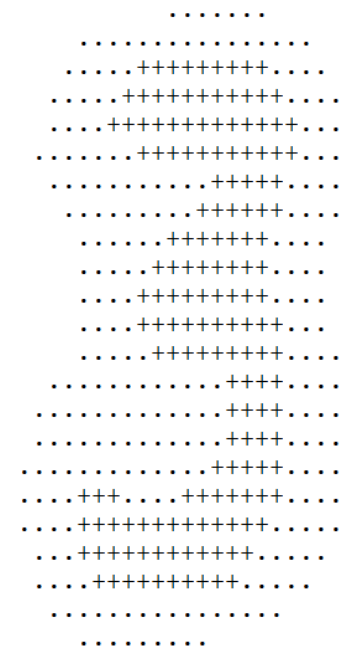

Figure 31. 3's Feature Likelihood

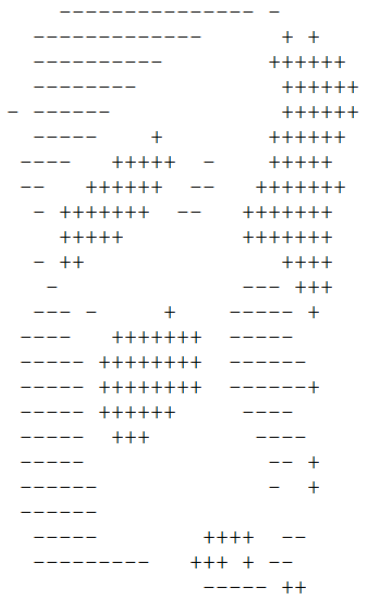

Figure 32. 8&3 Odd Pairs

([+] for 8's characteristics [ ] for common value [-] for 3' characteristics)

1.1.2 Part 1 Extra Credit (Face Data)

Implementation

The algorithm for this part is very similar to that of 1.1. However, the size of each sample from training or test file is changed to be 70x60, and the number of class parameters becomes 2 (face or not). (Part1 > identify_faces.java)

Results:

Accuracy:

Total Classification Percentage = 0.88

Classification Rate:

The Classification Ratio for 0 is 0.8311688311688312

The Classification Ratio for 1 is 0.9315068493150684

Confusion Matrix:

Table3. Confusion matrix for face classifier

	0 (non-face)	1 (face)
0 (non-face)	0.831	0.169
1 (face)	0.932	0.068

2.1 Text Document Clarification

Implementation

The basic idea for this part is to count the frequency of all the words in spam (1) and non-spam (0) emails, or in good (1) and bad film (-1) reviews. Then build naïve Bayes models to classify a give message by estimating the posterior with the words shown in the message, with a) multinomial Bayes model and b) Bernoulli Bayes model.
(Part2 > spam_email_filter; Part2 > movie _review_classifier)

Algorithm Outline:

- Training
    - For each line in the training file, split " ", then read the first char (first 2 char in movie review case) as the class identifier, split ":" read the following two strings and transfer and save it to two HashMap<String, Double> spam_train or non_spam_train according to the class identifier as word and their correspond occurrence. If the word already exists in the HashMap, then add-up old value and new reading as a new value to save
    - Count the number of unique words in each class by summing the number of elements in each HashMap with value of 1.0
    - Keep track of the total number of messages in each class
    - Calculate the prior for each class as

$$P_{prior}(class) = \frac{total\ number\ of\ message\ in\ this\ calss}{total\ number\ of\ training\ message}$$

    - Calculate the likelihood for each word in each class by

$$P_{likelihood}(word|class) = \frac{number\ of\ occurrence\ of\ this\ word\ in\ this\ class + 1}{Total\ number\ of\ words\ in\ this\ class + number\ of\ unique\ words}$$

(Multinomial Case)

$$P_{likelihood}(word|class) = \frac{1(if\ this\ word\ shows\ up\ in\ the\ message) + 1}{Total\ number\ of\ words\ in\ this\ class + number\ of\ unique\ words}$$

(Bernoulli Case)

- Testing:
    - Read the test file and save each line as a String in a queue ***test_reading***
    - Create a function to read out one line at a time in the queue. split " ", then read the first char (first 2 char in movie review case) as the class identifier to be added to the ArrayList for test labels, split ":" read the following two strings and transfer and save it to a HashMap<String, Double>***test_read*** as word and their correspond occurrence
    - Iterate the HashMap ***test_read,*** calculate the posterior for each class as

$$P_{posterior}(class) = \ln\ (P_{prior}(class)) + \sum_{word} \ln(P_{likelihood}(word|class))$$

If the word does not show up in training email, then

$$P_{likelihood}(word|class) = \frac{1}{Total\ number\ of\ words\ in\ this\ class + number\ of\ unique\ words}$$

    - For each message in the test file, find the class with highest posterior, and add the result to an ArrayList, compared with the test labels. Build confusion matrix according to the results
    - Use a comparator to find the top 20 words with highest likelihood in each class

Results

1. Spam Detection (email)

a) Multinomial

Classification Rate:

Table4. Classification Rate for spam email filter

Class	Rate
0 (non-spam)	0.984375
1 (spam)	0.9696969696969697

Confusion Matrix:

Table5. Confusion matrix for spam email filter

Class	0 (Non-spam)	1 (Spam)
0 (Non-spam)	0.984	0.016
1 (Spam)	0.030	0.970

Table6. Top 20 words in each class

Top 20 words in Spam Email:	Top 20 words in non-Spam Email:
email | 1380.0	**language | 1130.0**
s | 1207.0	**university | 906.0**
order | 1159.0	**s | 661.0**
report | 1053.0	**linguistic | 477.0**
our | 965.0	**de | 445.0**
address | 954.0	**information | 444.0**
mail | 923.0	**conference | 378.0**
program | 828.0	**workshop | 360.0**
send | 800.0	**email | 321.0**
free | 744.0	**paper | 320.0**
money | 722.0	**e | 314.0**
list | 713.0	**english | 312.0**
receive | 662.0	**one | 280.0**
name | 627.0	**please | 278.0**
business | 608.0	**include | 277.0**
one | 553.0	**edu | 271.0**
d | 541.0	**http | 264.0**
work | 528.0	**research | 259.0**
com | 524.0	**abstract | 253.0**
nt | 520.0	**address | 252.0**

b) Bernoulli

Classification Rate:

Table7. Classification Rate for spam email filter

Class	Rate
0 (non-spam)	0.5179282868525896
1 (spam)	1

Confusion Matrix:

Table8. Confusion matrix for spam email filter

Class	0	1
0	0.518	0.482
1	0	1

Table9. Top 20 words in each class

Top 20 words in Spam Email:	Top 20 words in non-Spam Email:
email | 1380.0	**language | 1130.0**
s | 1207.0	**university | 906.0**
order | 1159.0	**s | 661.0**
report | 1053.0	**linguistic | 477.0**
our | 965.0	**de | 445.0**
address | 954.0	**information | 444.0**
mail | 923.0	**conference | 378.0**
program | 828.0	**workshop | 360.0**
send | 800.0	**email | 321.0**
free | 744.0	**paper | 320.0**
money | 722.0	**e | 314.0**
list | 713.0	**english | 312.0**

receive | 662.0	**one | 280.0**
name | 627.0	**please | 278.0**
business | 608.0	**include | 277.0**
one | 553.0	**edu | 271.0**
d | 541.0	**http | 264.0**
work | 528.0	**research | 259.0**
com | 524.0	**abstract | 253.0**
nt | 520.0	**address | 252.0**

2. Movie Reviews (sentiment)

a) Multinomial

Classification Rate:

Table10. Classification Rate for Review Classifier

Class	Rate
+1 (Good Review)	0.7529411764705882
-1 (Bad Review)	0.763265306122449

Confusion Matrix:

Table11. Confusion Matrix for Review Classifier

Class	+1 (Good Review)	-1 (Bad Review)
+1 (Good Review)	0.763	0.237
-1 (Bad Review)	0.247	0.753

Table12. Top 20 words for each class

Top 20 words in Good Film Review	Top 20 words in Bad Film Review:
film | 285.0	**movie | 290.0**
movie | 187.0	**film | 227.0**
-- | 136.0	**like | 163.0**
one | 111.0	**one | 143.0**
like | 99.0	**-- | 114.0**
story | 94.0	**bad | 87.0**
good | 84.0	**story | 85.0**
comedy | 83.0	**much | 83.0**
way | 80.0	**time | 75.0**
even | 76.0	**even | 70.0**
time | 73.0	**good | 64.0**
best | 72.0	**characters | 64.0**
much | 66.0	**little | 62.0**
performances | 62.0	**would | 58.0**
make | 60.0	**comedy | 57.0**
funny | 60.0	**never | 53.0**
us | 58.0	**nothing | 52.0**
life | 58.0	**plot | 51.0**
makes | 58.0	**makes | 51.0**
characters | 56.0	**make | 50.0**

b) Bernoulli

Classification rate:

Table13. Classification Rate for Review Classifier

Class	Rate
+1 (Good Review)	0.6738544474393531
-1 (Bad Review)	0.6025437201907791

Confusion Matrix:

Table14. Confusion Matrix for Review Classifier

Class	+1 (Good Review)	-1 (Bad Review)
+1 (Good Review)	0.603	0.397
-1 (Bad Review)	0.326	0.674

Table15. Top 20 words for each class

Top 20 words in Good Film Review	Top 20 words in Bad Film Review:
film | 285.0	**movie | 290.0**
movie | 187.0	**film | 227.0**
-- | 136.0	**like | 163.0**
one | 111.0	**one | 143.0**
like | 99.0	**-- | 114.0**
story | 94.0	**bad | 87.0**
good | 84.0	**story | 85.0**
comedy | 83.0	**much | 83.0**
way | 80.0	**time | 75.0**
even | 76.0	**even | 70.0**
time | 73.0	**good | 64.0**
best | 72.0	**characters | 64.0**
much | 66.0	**little | 62.0**
performances | 62.0	**would | 58.0**
make | 60.0	**comedy | 57.0**
funny | 60.0	**never | 53.0**
us | 58.0	**nothing | 52.0**
life | 58.0	**plot | 51.0**
makes | 58.0	**makes | 51.0**
characters | 56.0	**make | 50.0**