UNM CS429: Naive Bayes and Logistic Regression

Vasileios Grigorios Kourakos and Chai Capili

March 25, 2023

1 Introduction

Github Repository

This project Implements the following models in Java:

- Naive Bayes
- Logistic Regression
- Mutual Information

Our Java implementation uses the following libraries for matrix linear algebra and operations:

- matrix-toolkits-java
- ojAlgo

1.1 The Problem

The implementation of this project aims to classify documents using Naive Bayes, Logistic Regression, and tuning to get optimal accuracy in document classification. Document classification aims to train a model based on a set of documents, the words in those documents, and the category that each document belongs to. Once the model has been trained, we pass in new documents to the model and ask which category that the new document would most likely belong to.

1.2 Implementation

Naive Bayes, Logistic Regression, Mutual Information, and all of the other methods required to successfully create this project were coded in Java with the help a few libraries.

The jar file to run this is contained in our repository Upon execution the program will take the following options for the user:

1. Naive Bayes
   (a) Beta value
2. Logistic Regression
   (a) Lambda Value (penalty term)
   (b) Eta value (learning rate)
   (c) Number of Iterations
3. Exit Option
4. Note on Files

   There is no option to input file paths while the program is running. To run the jar, place the files in the data set below in the same folder/directory that the jar is in. The file names need to be exactly the same as the ones in our Data Set (Section 1.3).

Implementation of Naive Bayes is standard, we used Java's hashmaps to navigate the data sets and implemented the formulas provided in the project pdf. The way we identified important features will be discussed later in the report. I don't believe there will be much deviation in accuracies for NB between different teams as the only difference is the beta value, and the beta values we all used to test are the same (1/V, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1).

As for Logistic Regression, there are a lot more parameters to consider. We used libraries that allowed for sparse matrix structures and linear algebra as suggested in class by the professor. This reduced memory usage, increased performance and let us go through ≈2-3 iterations of gradient descent per second.

The matrices in the code are defined the same as the matrices described in the project pdf, and the weight update method performs the matrix operation equation given to us. There is a slight deviation in calculating the probabilities matrix, as before raising e to every element, we had to scale down the matrix to prevent overflow. Letting the overflow through produced Infinity and NaN values and the code was not working at all. Even after scaling though, we should still get the relative probabilities for each class, and the prediction will be the same, and the model's accuracy is 88% with the correct hyperparameters so the results are good. After the final iteration, the model uses the final weights to predict the class of each document in the testing file and prints them out in console. We also print the confusion matrices for both LR and NB.

We tried a few different scaling methods for the input data, such as Tf-Idf, Min-Max, Standard Score. We scrapped standard score because that would mean having to switch to a dense matrix instead of sparse in the code, which would increase memory usage to about 8GB instead of ≈1GB now and make the algorithm much slower. That is because for standardization, we would have the apply it to the zero values as well making them non-zero, and the matrix would end up not being sparse. Tf-Idf by itself gave around a ≈70% accuracy. Min-Max performed really badly, with a ≈25% accuracy. We realized that columns that had really high values were most likely common English words like "the" "of" etc. We came up with another way to manipulate the data which was to get the sum of each column, so the total count of a specific word in all documents, then divide every element of that column by the sum. That made all elements of the matrix be between [0,1] and the sum of each column being equal to 1. This meant that common words would not dominate the weights. This type of normalization by itself gave us a higher accuracy, ≈79%. We tried a few combinations and Tf-Idf followed by this normalization technique ended up giving us the 88% accuracies which ended up being our final model.

1.3 Data set

Below is the data set which was used to train the models. Of the 12,000 documents there was a split of 10,000 training with 2,000 testing documents. There are 20 different news groups of which we aim to classify.

- vocabulary.txt

  is a list of the words that may appear in documents. The line number is word's d in other files. That is, the first word ('archive') has wordId 1, the second ('name') has wordId 2, etc.

- newsgrouplabels.txt

  is a list of newsgroups from which a document may have come. Again, the line number corresponds to the label's id, which is used in the .label files. The first line ('alt.atheism') has id 1, etc.

- training.csv

  Specifies the counts for each of the words used in each of the documents. Each line contains 61190 elements. The first element is the document id, the elements 2 to 61189 are word counts for a vectorized representation of words (refer to the vocabulary for a mapping). The last element is the class of the document (20 different classes). All word/document pairs that do not appear in the file have count 0.

- testing.csv

  The same as training.csv except that it does not contain the last element.

2 Guided Questions

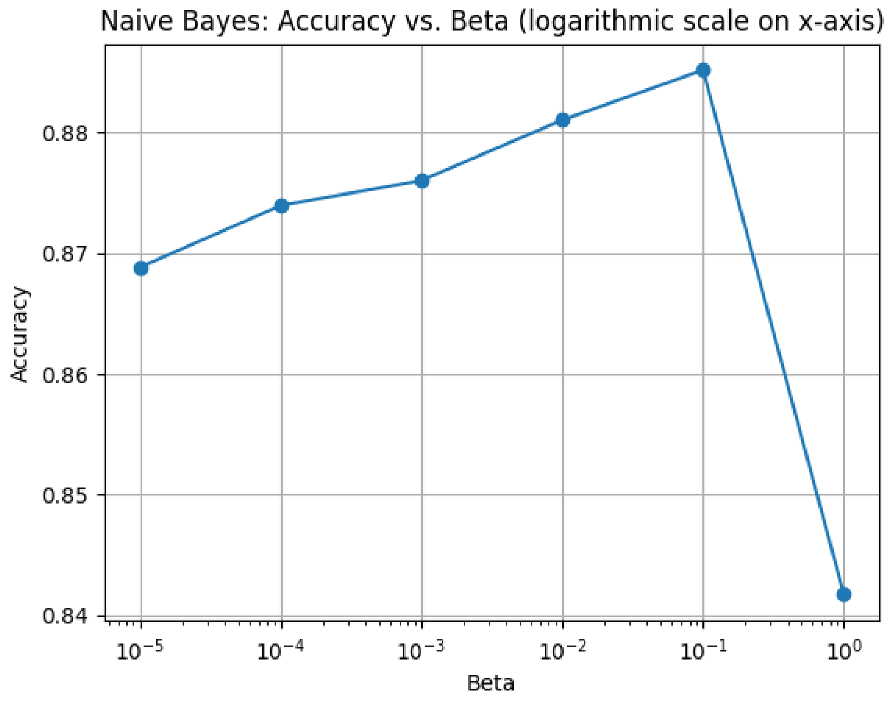
2.1 Difficulty estimating parameters

In your answer sheet, explain in a sentence or two why it would be difficult to accurately estimate the parameters of this model on a reasonable set of documents (e.g. 1000 documents, each 1000 words long, where each word comes from a 50,000 word vocabulary)

The model proposed above talks about specific positions in a document having the value of a specific word. The problem with that is that there are too many dimensions in that problem with only 1000 training documents. With every position in each document having its own probability distribution, and with the assumption we're making that P(Xi—Y) maybe be completely different from P(Xj—Y) as the model states, we will not have nearly enough data for each distribution to accurately estimate the model parameters. This is why for our Naive Bayes model, we don't care where a word is in a document but that it exists somewhere in the document and how many times.

2.2 Naive Bayes and the effect of β

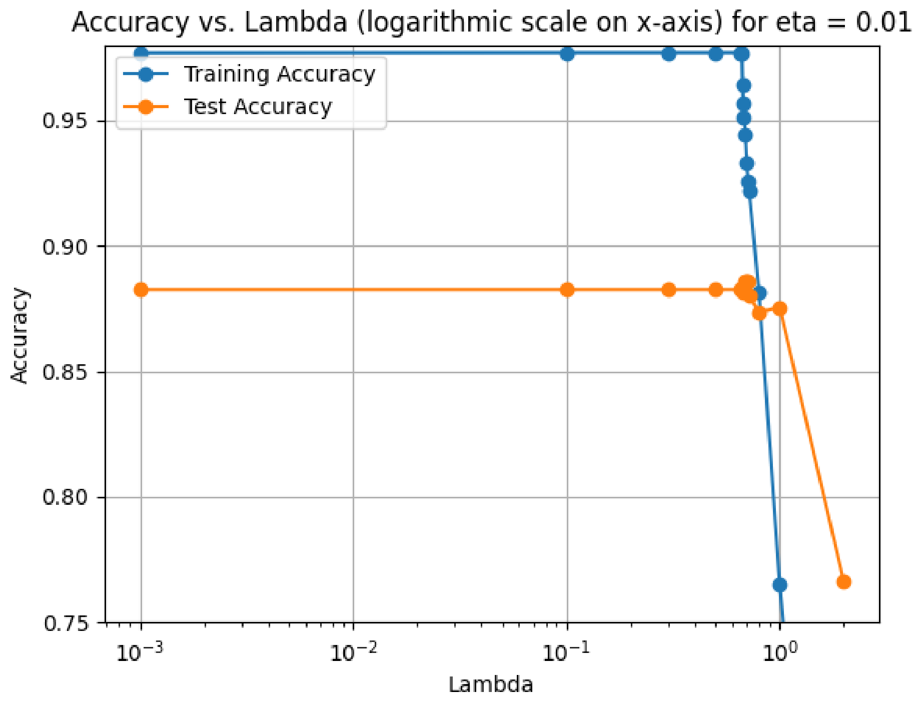
Re-train your Naive Bayes classifier for values of b between .00001 and 1 and report the accuracy over the test set for each value of b. Create a plot with values of b on the x-axis and accuracy on the y-axis. Use a logarithmic scale for the x-axis (in Matlab, the semilogx command). Explain in a few sentences why accuracy drops for both small and large values of b.



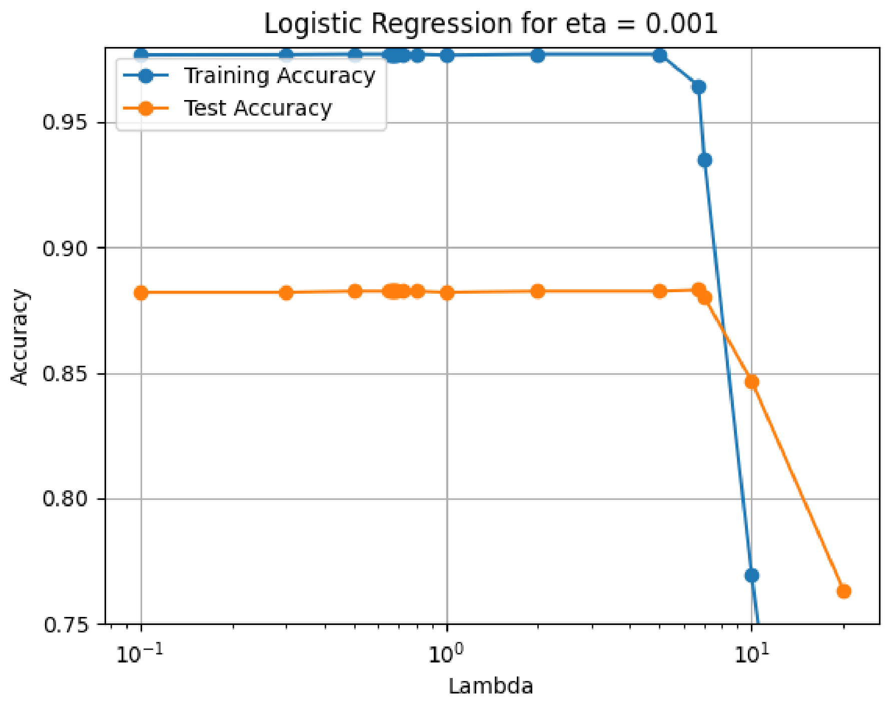
The beta value signifies the strength of the prior, which is how confident we are that the training data we have is correct and unbiased. When it's too low, we don't use the training set as much as we should, and when it's too high, we use the training set data too much. Finding the sweet spot helps us increase the accuracy of the model, which in this case is beta = 0.1 .

2.3 Logistic Regression: η (learning rate), λ (penalty term), iterations

Re-train your Logistic Regression classifier for values of η starting from 0.01 to 0.001, $\lambda = 0.01$ to 0.001 and vary your stopping criterium from number of total iterations (e.g. 10,000) or $= 0.00001$ and report the accuracy over the test set for each value (this is more efficient if you plot your parameter values vs accuracy) . Explain in a few sentences your observations with respect to accuracies and sweet spots



The above plot shows our training and validation accuracy for eta = 0.01 and various lambda values. Having too low lambda means that the model overfits the training data, and while we have very high training accuracy, the validation accuracy is not that great. At some point if the lambda is high enough, the model starts fitting the training data less, but there is a point where the validation accuracy can increase by a little bit. We found that sweet spot to be at lambda = 0.69-0.71 where our validation accuracy was 0.886 and the training accuracy 0.933, instead of 0.882 and 0.9769 for lower lambdas. Using a higher lambda than that leads to the model underfitting the training data and producing a model too simplistic which performs worse for the validation set as well.



The above plot shows our training and validation accuracy for eta = 0.001 and various lambda values. Same as above, too low lambda leads to overfitting and too high leads to underfitting. For 0.001 we found the sweet spot to be around lambda = 6.7 where the validation accuracy slightly increased to 0.883 instead of 0.882. Our best validation accuracy was with eta 0.01.

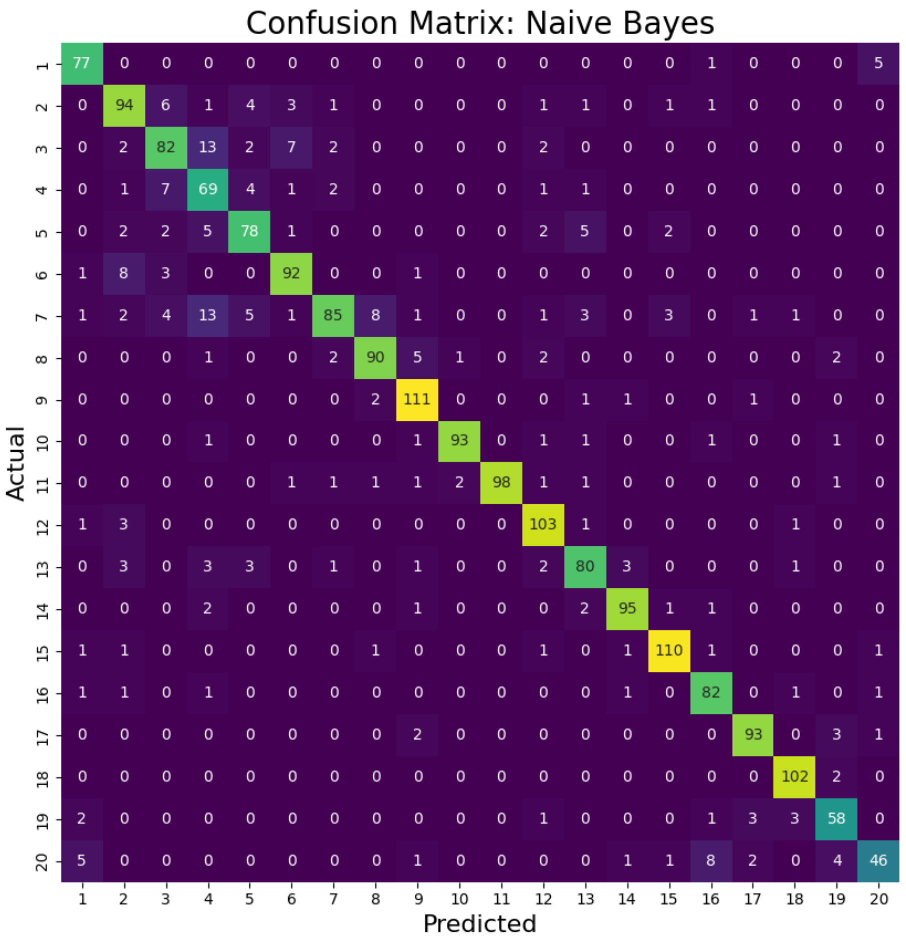
We suspect that the penalty term didn't have much of an impact in increasing the accuracy because of the way we manipulated the training data (which is discussed at the beginning of this document), making the x values small and keeping the weights pretty low regardless.

2.4 Results

In your answer sheet, report your overall testing accuracy (Number of correctly classified documents in the test set over the total number of test documents), and print out the confusion matrix (the matrix C, where c_{ij} is the number of times a document with ground truth category j was classified as category i).

2.4.1 Naive Bayes

Accuracy: 88.514 for $\beta = 0.1$

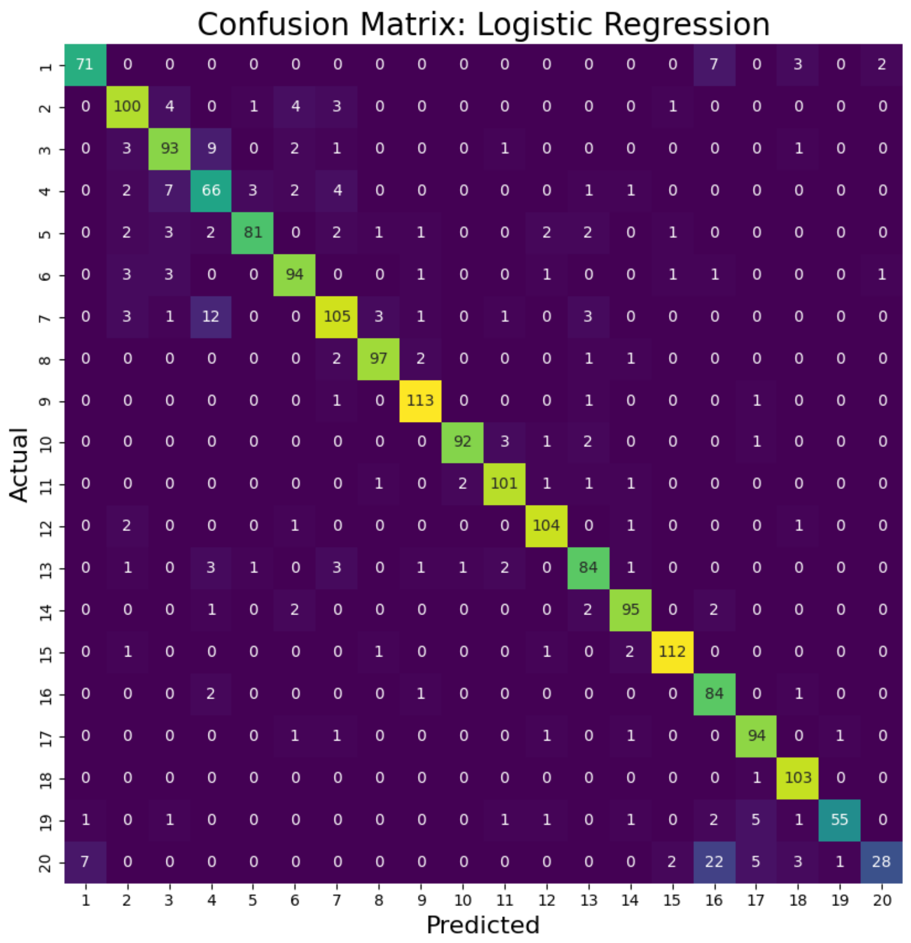


Confusion Matrix: Naive Bayes

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	77	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	5
2	0	94	6	1	4	3	1	0	0	0	0	1	1	0	1	1	0	0	0	0
3	0	2	82	13	2	7	2	0	0	0	0	2	0	0	0	0	0	0	0	0
4	0	1	7	69	4	1	2	0	0	0	0	1	1	0	0	0	0	0	0	0
5	0	2	2	5	78	1	0	0	0	0	0	2	5	0	2	0	0	0	0	0
6	1	8	3	0	0	92	0	0	1	0	0	0	0	0	0	0	0	0	0	0
7	1	2	4	13	5	1	85	8	1	0	0	1	3	0	3	0	1	1	0	0
8	0	0	0	1	0	0	2	90	5	1	0	2	0	0	0	0	0	0	2	0
9	0	0	0	0	0	0	0	2	111	0	0	0	1	1	0	0	1	0	0	0
10	0	0	0	1	0	0	0	0	1	93	0	1	1	0	0	1	0	0	1	0
11	0	0	0	0	0	1	1	1	1	2	98	1	1	0	0	0	0	0	1	0
12	1	3	0	0	0	0	0	0	0	0	0	103	1	0	0	0	0	1	0	0
13	0	3	0	3	3	0	1	0	1	0	0	2	80	3	0	0	0	1	0	0
14	0	0	0	2	0	0	0	0	1	0	0	0	2	95	1	1	0	0	0	0
15	1	1	0	0	0	0	0	1	0	0	0	1	0	1	110	1	0	0	0	1
16	1	1	0	1	0	0	0	0	0	0	0	0	0	1	0	82	0	1	0	1
17	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	93	0	3	1
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	102	2	0
19	2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	3	3	58	0
20	5	0	0	0	0	0	0	0	1	0	0	0	0	1	1	8	2	0	4	46

2.4.2 Logistic Regression

Accuracy for Logistic Regression: 0.886



Confusion Matrix: Logistic Regression

Actual \ Predicted	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	71	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	3	0	2
2	0	100	4	0	1	4	3	0	0	0	0	0	0	0	1	0	0	0	0	0
3	0	3	93	9	0	2	1	0	0	0	1	0	0	0	0	0	0	1	0	0
4	0	2	7	66	3	2	4	0	0	0	0	0	1	1	0	0	0	0	0	0
5	0	2	3	2	81	0	2	1	1	0	0	2	2	0	1	0	0	0	0	0
6	0	3	3	0	0	94	0	0	1	0	0	1	0	0	1	1	0	0	0	1
7	0	3	1	12	0	0	105	3	1	0	1	0	3	0	0	0	0	0	0	0
8	0	0	0	0	0	0	2	97	2	0	0	0	1	1	0	0	0	0	0	0
9	0	0	0	0	0	0	1	0	113	0	0	0	1	0	0	0	1	0	0	0
10	0	0	0	0	0	0	0	0	0	92	3	1	2	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	1	0	2	101	1	1	1	0	0	0	0	0	0
12	0	2	0	0	0	1	0	0	0	0	0	104	0	1	0	0	0	1	0	0
13	0	1	0	3	1	0	3	0	1	1	2	0	84	1	0	0	0	0	0	0
14	0	0	0	1	0	2	0	0	0	0	0	0	2	95	0	2	0	0	0	0
15	0	1	0	0	0	0	0	1	0	0	0	1	0	2	112	0	0	0	0	0
16	0	0	0	2	0	0	0	0	1	0	0	0	0	0	0	84	0	1	0	0
17	0	0	0	0	0	1	1	0	0	0	0	1	0	1	0	0	94	0	1	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	103	0	0
19	1	0	1	0	0	0	0	0	0	0	1	1	0	1	0	2	5	1	55	0
20	7	0	0	0	0	0	0	0	0	0	0	0	0	0	2	22	5	3	1	28

2.5 Which instances of Y (newsgroups) confuse the most?

Are there any newsgroups that the algorithm(s) confuse more often than others? Why do you think this is?

As expected, similar newsgroups often get confused. For example, our logistic regression confusion matrix actually shows more documents incorrectly classified for newsgroup 20 than correctly classified! Newsgroup 20, which is "talk.religion.misc" mostly gets confused with 16, "soc.religion.christian" and group 1, "alt.atheism". Group 1 also gets confused with group 16 but less often. Other similar newsgroups also get confused sometimes. Surprisingly, the 3 "politics" newgroups did not have many misclassification errors. The Naive Bayes model also misclassified group 20, but not as much. Another 2 groups that were confused with each other in both models were 3 and 4, ”comp.os.ms-windows.misc” and ”comp.sys.ibm.pc.hardware”. That is because in these similar newsgroups, the same words are used frequently and the important words that the model learns from might be used more in the incorrect newsgroup. That and the newsgroup not having important unique words with high weights can lead to these mistakes.

2.6 Finding the most ”important words” in our vocabulary

Propose a method for ranking the words in the dataset based on how much the classifier ‘relies on’ them when performing its classification (hint: information theory will help). Your metric should use only the classifier’s estimates of P (Y) and P (X—Y). It should give high scores to those words that appear frequently in one or a few of the newsgroups but not in other ones. Words that are used frequently in general English (‘the’, ‘of’, etc.) should have lower scores, as well as words that only appear extremely rarely throughout the whole dataset. Finally, in your method this should be an overall ranking for the words, not a per-category ranking

2.6.1 Methodology

Our team used Mutual Information to determine the 100 most important words in the data set. When first approaching the problem, we implemented normal formula for Mutual Information defined by:

$$\mathrm{MI}(X;Y) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(x,y)}{p(x)p(y)}$$

Mutual Information is a metric from Information theory which associates the degree of dependence between two variables. In the case of our Document Classification, we can use the concept of mutual information to find the most ”important words”, which are words that appear frequently in one or a few of the newsgroups. Words with low

Mutual Information scores are frequently used words across all documents and words that are rare through the whole data set. Note, this is not a per category ranking, instead, it is the ranking of the words across the whole data set.

Optimizing Mutual Information for our Data set:

The formula above assumes that all of the X values will be taken, however, this is extremely costly to performance in the case of our implementation and data set. Instead we can use 0 and 1 to denote different cases using the following:

$$MI(x, y_i) = \sum_{i,j \in 0,1} p(x = i, y_i = j) * log(p(x = i, y_i = j)/p(x = i) * p(y_i = j))$$

For each word calculate the following:

p(x,y)log(p(x,y)/(p(x)p(y)))

For the combinations of x,y (0,0) (0,1) (1,0) (1,1)

- (0,0): x word not occuring in classes that are not y
- (0,1): x word not occuring in class y
- (1,0): x word occuring in classes that are not y
- (1,1): x word occuring in class y

Sum those 4 metrics, multiply by the probability of class y. Finally, sum the above metric from all classes to get the average MI of the word. This greatly increases performance from the first formula of Mutual Information as we will have to perform far less calculations per word.

2.6.2 Implementation

The code for our implementation lives in our Java Project, within the Naive Bayes class, found in the method *public void mutualInformation()*. In the larger scale of the Naive Bayes class, this method is called after Naive Bayes creates the data set and calculates the probabilities. When running this method, the user will know it has been called when the line 'Calculating mutual information of words...' is shown. After running, the code returns the 100 best words in our data set.

The most important words are words with the highest MI score, with 0.0 meaning no importance and 1.0 being max importance. The words are ranked as follows:

2.6.3 Ranking the 100 most important words

Using Naive Bayes with $\beta = 1 / \mid V \mid$ and our method print out the 100 words with the highest measure:

1. windows
2. dod
3. god
4. bike
5. he
6. team
7. game
8. government
9. hockey
10. rutgers
11. clipper
12. car
13. encryption
14. jesus
15. christ
16. people
17. his
18. christians
19. athos
20. who
21. writes
22. christian
23. bible
24. israel
25. baseball
26. sale
27. we
28. games
29. space
30. article
31. season
32. chip
33. cars
34. were
35. dos
36. key
37. graphics
38. players
39. that
40. thanks
41. jews
42. they
43. mac
44. christianity
45. church
46. israeli
47. fbi
48. card
49. was
50. nhl
51. win

52. pc
53. their
54. faith
55. him
56. ride
57. religion
58. league
59. arab
60. gun
61. shipping
62. apple
63. teams
64. playoffs
65. window
66. file
67. law
68. drive
69. believe
70. by
71. turkish
72. sin
73. as
74. player
75. program
76. keys
77. nsa
78. play
79. clh
80. jewish
81. security
82. being
83. escrow
84. not
85. our
86. year
87. orbit
88. armenia
89. against
90. motif
91. controller
92. riding
93. religious
94. turks
95. com
96. mb
97. nasa
98. of
99. say
100. scsi

2.7 Difference between Naive Bayes and Logistic Regression

Both Logistic Regression and Naive Bayes were able to score in the high 80 percent range on the validation set. In the beginning of this process, Naive Bayes was scoring much higher than Logistic Regression (88 percent vs high 70 percent), however, after tuning our parameters and algorithms, Logistic Regression and Naive Bayes can score in the high 80 percentages. Our highest entry on our Kaggle competition was through Naive Bayes with a Beta Value of (value). Logistic Regression was not able to beat Naive Bayes in this case, however, both peak accuracies are within a percent of each other.

2.8 Data set Bias

If the points in the training data set were not sampled independently at random from the same distribution of data we plan to classify in the future, we might call that training set biased. Data set bias is a problem because the performance of a classifier on a biased data set will not accurately reflect its future performance in the real world. Look again at the words your classifier is ‘relying on’. Do you see any signs of data set bias?

We can see that some of the words the classifier most ”relies on” have some bias. Words like ”playoffs” may appear a lot in sports articles during playoff seasons, but maybe not as much the rest of the time. The word ”scsi” which stands for ”Small Computer System Interface” while still being used in some applications, has mostly been replaced by USB. ”mb” which I assume stands for megabyte, might not be as relevant nowadays and in the future, being replaced by ”gb”/”tb”. Words like ”armenia”, ”turkish”, ”israel”, ”arab” etc. might have been in a lot of news articles when there was something important happening in that part of the world. So while these words would still classify a document in the group, they may not be used as much in future data that we want to classify. So the model giving these topical words a high weight may reduce the model’s accuracy in the future.