Project 2: Naive Bayes and Logistic Regression

Lasair Servilla (l.servilla@cosmiac.com)

Michael Servilla(chico@unm.edu)

Introduction

We implemented two classification algorithms based on the naive bayes and logistic regression models that were discussed in Tom Mitchell's book Machine Learning (1997, 2017). The dataset used for this project was downloaded from Kaggle's website (https://www.kaggle.com/competitions/cs429529-project-2-topic-categorization/data). This dataset consisted of 12,000 examples (rows) of text from 20 different newsgroups. Each example (row) contained 61,190 columns where column 1 was the index, column 61,190 was the class (newsgroup) values, and columns 2 through 61,189 were a vectorized word count for each example (row).

Methods

The following subsections will cover the methods used to calculate target predictions using both naive bayes and logistic regression. Overview sections will provide a high-level description explaining the general methods used. The algorithm sections will provide pseudo code for how the methods discussed in the overview sections were implemented.

Because of the large sparse matrix file presented for this project, data handling was completed using scipy's sparse matrix library, specifically using the csr_matrix format to compress the matrix into a more efficient format. Prior to compression, the file was split into a 80:20 ratio of training and testing files using sklearns's library train_test_split. These files were then compressed into the training file, *csr_train.csv_lg.ngz*, and the testing file, *csr_test.csv_lg.ngz*. All of the data handling was completed in the make_sparse.py code.

Overview- Naive Bayes

The Naive Bayes classifier uses two files to first train MLE and MAP estimates in main.py, and then calculates predictions in test.py. In main.py a sparse matrix for the training data is read in, and used to determine the following global variables:

- **unique_targets** -> the number of possible targets (classes), for the assignment data set it is the number of newsgroups that a document could be classified as

- **total_docs** -> the number of documents that are available in the training data set (number rows in the training matrix)
- **v_total** -> the number of unique attributes that, for the assignment data set it is the number of unique words (columns in the training matrix - index and target columns)
- **column_count** -> true number of columns in the training matrix

The above variables will be referenced in the algorithms for Naive Bayes. There are two options for running main.py by setting the final global variable (*run_loop*) as either true or false. Running main.py with it set as true will use an array of values to calculate multiple sets of MLE and MAP estimates for an alpha calculated from each value in the array. It will save a .csv file in a folder of the MLE and MAP values for each alpha used. Running main.py with *run_loop* set to false will calculate a single set of MLE and MAP estimates calculating alpha using beta= 1/v_total. This will also save a .csv file of the estimates. The saved .csv files are used by test.py. The process to calculate the MLE and MAP estimates remains the same regardless of *run_loop* state. First each document is added to a subset based on what newsgroup it belongs to. Each subset is then used to calculate the MLE and MAP values for the newsgroup the subset is associated with. The final result is a matrix of MLE and MAP estimates with each row representing an individual newsgroup with the first to *v_total* columns for MAP values and the last column for MLE values. It is this matrix that gets saved to the .csv file. The next step is to run test.py which reads in the matrix of MLE and MAP estimates as well the testing data set. There are three global variables that determine what test.py runs. The first is *run_loop*, if set to true test.py will use the set of estimates created by using the array to test multiple alpha values. If set to false only the alpha with beta=1/v_total will be used. The next variable *plot_results* determines whether or not a graph with the accuracy should be created and then shown. Finally, *true_targets* allows the user to set whether or not the testing data has the true target values in the last column. If the testing data does not have the true value set *true_targets* to false, this prevents accuracy calculations as well as plotting of graphs. Test.py wil calculate the results the same regardless of the global variables states. For each document in the testing data test.py will calculate the $P(Y_i|X)$, the probability that the document is in newsgroup Y_i given that it has the set of words X. From this it assigns the predicted newsgroup as the newgroup with the highest value probability.

Algorithms- Naive Bayes

Main.py

Build MAP and MLE (Final Matrix, beta)

Let *subsets* = a set of subsets created by grouping training data by newsgroup and summing the occurrences of each individual word for that newsgroup

Let *beta* = the value for calculating Beta as $\frac{1}{beta}$

For each subset in subsets:

Yk_docs_count = the number of training documents in the newsgroup associated with the current subset

Yk_words = sum of all words in the newsgroup associated with the current subset

For each word in training data:

X_i = the current words count for the newsgroup associated with the current subset

Append MAP(X_i, Yk_words, v_total, beta) to Final Matrix

Append MLE (Yk_docs_count, total_docs) to the last row of Final Matrix

Return Final Matrix

MAP (X_i, Yk_words, v_total, beta)

---

likelihood = $\frac{X\_i + (Alpha(beta) - 1)}{Yk\_words + ((Alpha(beta) - 1) * v\_total)}$

Return *likelihood*

MLE (Yk_docs_count, total_docs)

---

prior = $\frac{Yk\_docs\_count}{total\_docs}$

Return prior

Alpha (beta)

Beta = $\frac{1}{beta}$

alpha = 1 + *Beta*

Return *alpha*

Test.py

Predictions (Testing Data, Final Matrix - last column, Prior, True Target Exists)

Let Final Matrix - last column = the matrix of MAP estimates
Let Prior = the last column of Final Matrix with the MLE estimates
Let True Target Exists = true or false based on if the Testing Data has the true target values in the last column

If *True Target Exists* is true:
Let y = set of true targets from Testing Data
Else:
Let y = Nothing

For document in Testing Data:

For newsgroup (row) in (Final Matrix - last column):

LogMAP = Final Matrix - last column(get row associated with current newsgroup)
LogMAP = log_2(LogMAP)
Dot = document * LogMAP
Probability = log_2(Prior) + Dot

Appended to prediction = newsgroup with the maximum Probability

Return prediction, y

Overview- Logistic Regression

The Logistic Regression classifier (*logistic_regression.py*) uses a compressed sparse matrix training file and a separate compressed sparse matrix testing file to train the weights of the of the logistics regression algorithm. The testing file is loaded and converted to a column vector array (*Y_test*) with the class values and a testing matrix (*X_test*) minus the last column of class values. A value of 1 is then placed in the first column and the columns of the matrix are then normalized to a sum-of-one. The training file is then loaded and converted to a matrix (*X*) minus the last column of class values. A value of 1 is then placed in the first column and the columns of the matrix are then normalized to a sum-of-one. The class values of the training file are used to create a target (class) column vector array (*Y*).

Once the files are loaded and the data is preprocessed, the following values are determined from the current array/matrix sizes (some values are predetermined - *):

- ***iter*** - number of iterations to train the weights of gradient descent/ascent*****
- ***k*** - number of classes in the training set
- ***n*** - number of attributes (features or columns) each example (document or row) has
- ***eta*** - the learning rate or step size*****
- ***lambda_*** - the penalty term used in regularization*****
- ***delta*** - a (*k * m*) matrix where $delta_{j\,i}$ = 1 if *j*th training value, $Y^i = y_j$ and $delta_{j\,i}$ = 0 otherwise
- ***X*** - an (*m * x* (*n+1*)) training set without index or class columns, 1 based index,
- *m* is rows in the training set*****
- ***Y*** - an (*m * 1*) vector(matrix) of true classifications for each example*****
- ***W*** - a (*k * x* (*n+1*)) matrix of weights

Two methods are used for the training as well as testing of the logistic regression algorithm, these include *make_pred* and *likelihood*. The first, *make_pred*, takes the exponent of the dot product of the weight matrix, *W*, and the examples matrix, *X*, transposed). The results of this are then normalized by column to a sum-of-one. The second method, *likelihood*, takes as parameters the weight matrix, *W*, the examples matrix, *X*, and the target vector array, *Y*, to calculate the current row likelihood for each sample. This is accomplished by iterating through each row and subtracting the log of 1 + the exponent of the dot product of *W* and *X* from the dot product of *W* and *X* and then taking the product of this result and the row specific target value from the vector array *Y*. This value is then summed for the conditional likelihood value.

A for loop based on the preselected number of iterations runs the gradient descent/ascent functions that include updating the weight values in *W*. Based on these new weight values, the *likelihood* method is called and if the likelihood has improved, then the new weights are saved as the current weights. This is completed when the number of iterations are met.

Algorithms- Logistic Regression

logistic_regression.py

Data Preprocessing

Load training set:
X <- training set minus target values
Y <- target values from training set.

Normalize training set:
X_sum <- sum of each column of *X*, replace all sums to zero with a one (prevents divide by zero error)
X := *X*/*X_sum* (each value in column is divided by the sum of that column)

Minutia:
num := number of samples (rows) in training set
k := number of unique targets (classes)
n := number of columns (features) in *X*
W <- initialize an *k* * *n* zero matrix
iter := number of chosen iterations
eta := chosen learning rate
lambda := chosen penalty value
delta <- initialize an *k* * *num* zero matrix
    for *i* : *num*
        delta[target value in Y_i][*i*]
    end for

---

make_pred

---

parameters: *W*, *X*
prediction := $exp^{dot(W,\ X\ transposed)}$
prediction(every last column cell) := 1
prediction := *prediction*(every column cell) / sum of column
return *prediction*

---

likelihood

---

parameters: *W*, *X*, *Y*
lh_sum := 0
for *i* : *num*
    a := Y[*i*]
    lh := *a* * (dot(*W*[*a*], *X*[*i*]) - *log*(1 + $exp^{dot(W[a],\ X[\ i\ ])}$)
    lh_sum += *lh*
end for
return *lh_sum*

---

gradient ascent

---

lh_sum := likelihood(*W*, *X*, *Y*) initialize likelihood value
for *i* : *iter*
    W := *W* + *eta* * dot((*delta* - make_pred(*W*, *X*)), *X*) - (*lambda_* * *W*))
    lh_new := likelihood(*W*, *X*, *Y*)
    if *lh_sum* < *lh_new*
        lh_sum := *lh_new*

end if
end for

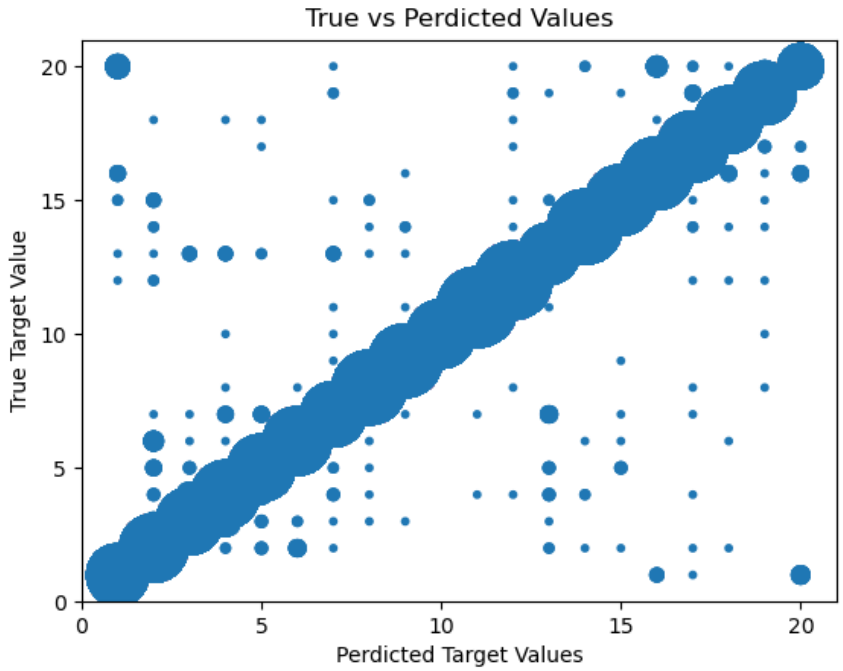
Results

General Performance- NB vs LR

Naive Bayes' accuracy was on average higher than the accuracy of the Logistic Regression method. With the range in accuracy of Naive Bayes sticking in the 80's and the range in accuracy for Logistic Regression staying in the 70's. Naive Bayes also worked much more efficiently than Logistic Regression. This was primarily due to Logistic Regression's need to navigate the gradient in order to train the weights and reach the highest accuracy. This resulted in Logistic Regression taking hours to complete while Naive Bayes would complete in minutes. However Logistic Regression's true runtime could be changed by setting the algorithm to end after a given number of iterations or after some other threshold has been reached. In Logistic Regression by not running the algorithm past the sweet spot where accuracy is minimally changed with each further iteration, the runtime can be greatly reduced.

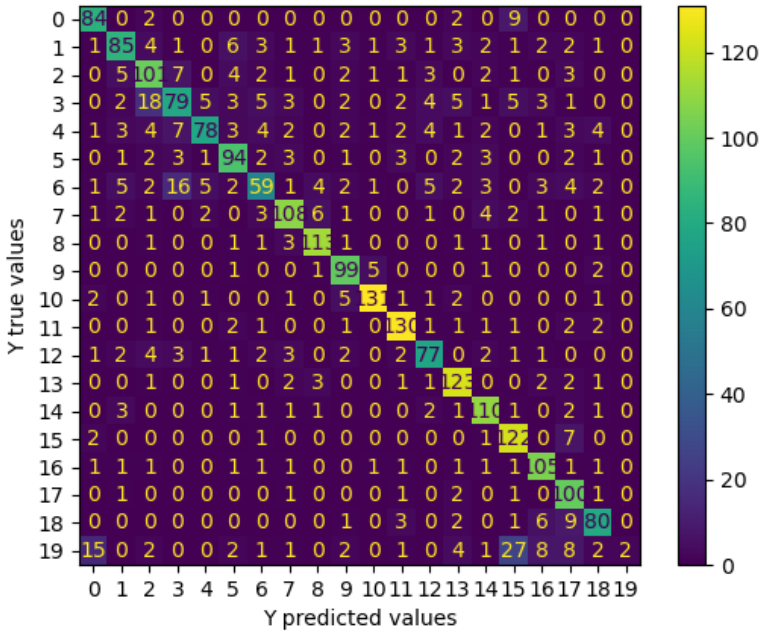
Accuracy- Naive Bayes

Using the validation data set created from a subset of the training data that was not used Naive Bayes produced an accuracy in the mid 80 percent range. The accuracy varied based on the Beta value that was used. For the initial test Beta was set to be 1/|*v_total*| which was approximately 0.000016. This value puts it on the lower end of the range of Beta values tested, 0.00001 to 1. This range resulted in a sweet spot for accuracy right before 1, drastically dropping off after reaching 1. This maximum recorded accuracy was 87 percent and the minimum record accuracy was 84.25 percent. The graph below shows the predicted values vs the true values with a 0.8633 accuracy using $\beta = 1/v\_total$. The larger the bubble the more occurrences at that point. From the graph you can see that Naive Bayes frequently misclassified documents in newsgroup 20 as being in newsgroup 1.



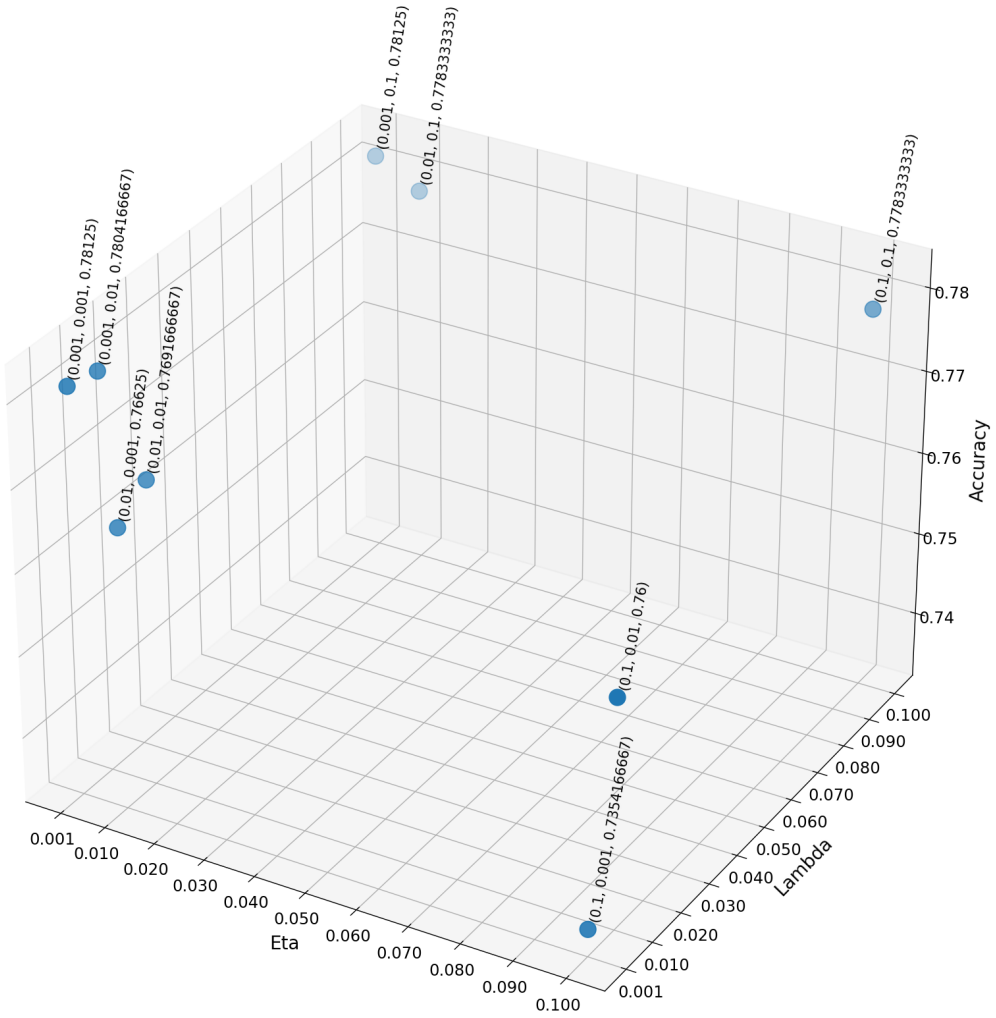
Accuracy- Logistic Regression

Overall accuracy for the logistic regression algorithm loomed in the high 70 percent range with a peak of 79 percent based on the parameters of eta = 0.001, lambda_ 0.001, and at 100 iterations. The range for accuracy was 74 percent for a minimum to 79 percent for a maximum.

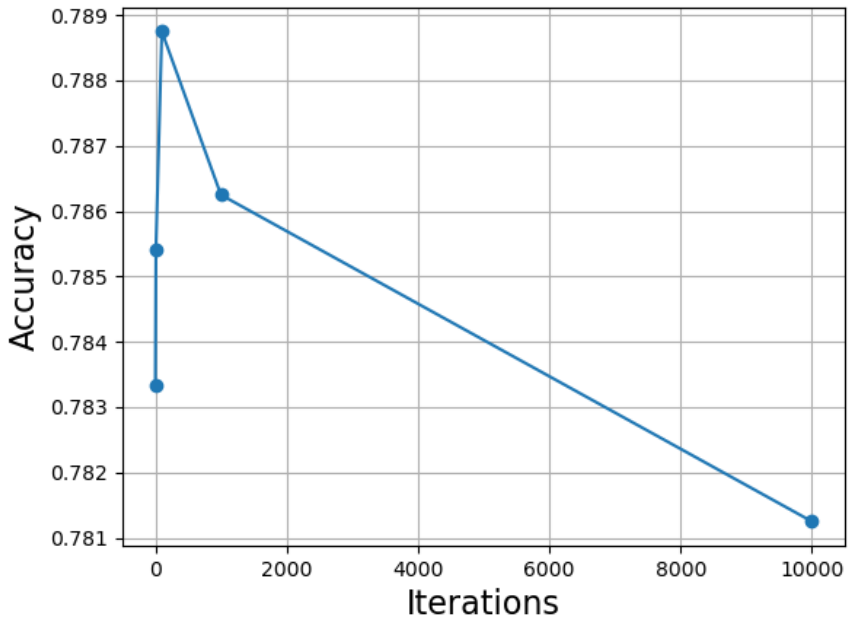


Confusion matrix for the logistic regression classifier.

Note: the newsgroup numbering is reduced by a value of 1 due to zero based array indexing.



Accuracy plotted against the learning rate, eta (η), and the penalty lambda (λ).



Accuracy plotted against number of training iterations.

Conclusion

In all the implementation of Naive Bayes worked successfully and without issue. We were able to train the model in a reasonable amount of time and with a high accuracy. The Naive Bayes was able to be improved with the change in Beta values, bringing the accuracy up to 87%. A large part of what made Naive Bayes easy to work with was that there were simply far fewer factors than Logistic Regression and given the shorter runtime it was also easier to test different Beta values to improve results.

The implementation for Logistic Regression was less successful and far more problematic compared to Naive Bayes. Initial construction of the code used a scaled down data set to check for correctness and debugging purposes. This scaled down version ran very well on the initial Logistic Regression algorithm, however overflow issues were encountered as soon as the data was scaled up. The overflow was due to the e^x expression. To solve this the data was normalized. However we were never able to get the accuracy for Logistic Regression to get above 78% accuracy. Another contributing factor that made Logistic Regression difficult to work with was the amount of time it took to run, because there simply was not enough time to do as much testing as we would have wanted.

References

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill Education.

Mitchell, T. (2017). *Machine Learning*. http://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf.

Kaggle. https://www.kaggle.com/competitions/cs429529-project-2-topic-categorization.

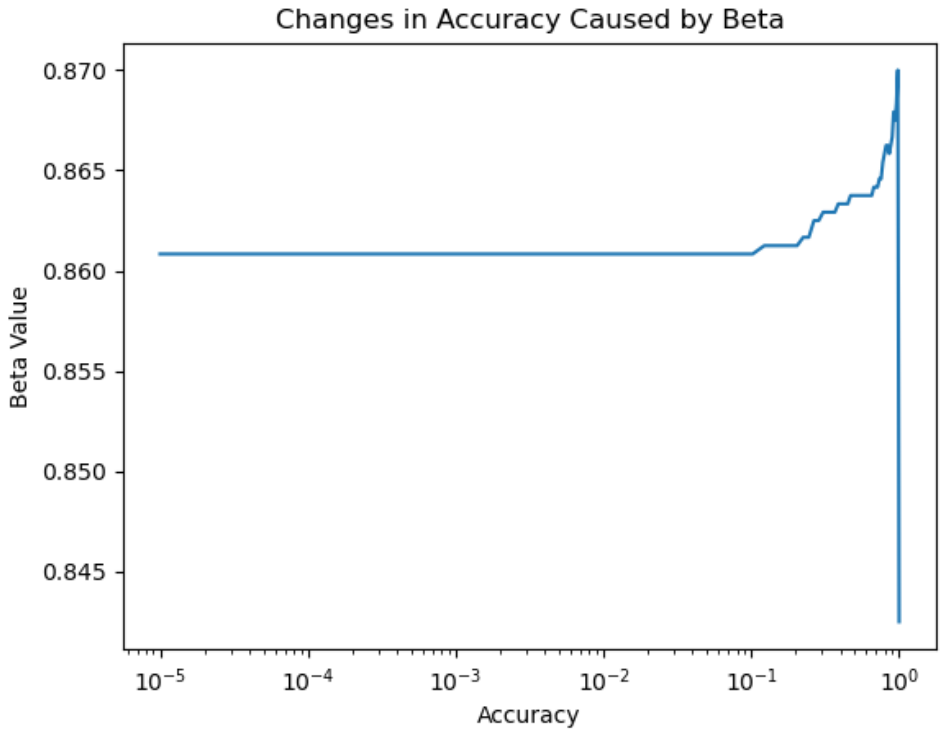
Question Answers

Question 1: 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). [5 points]

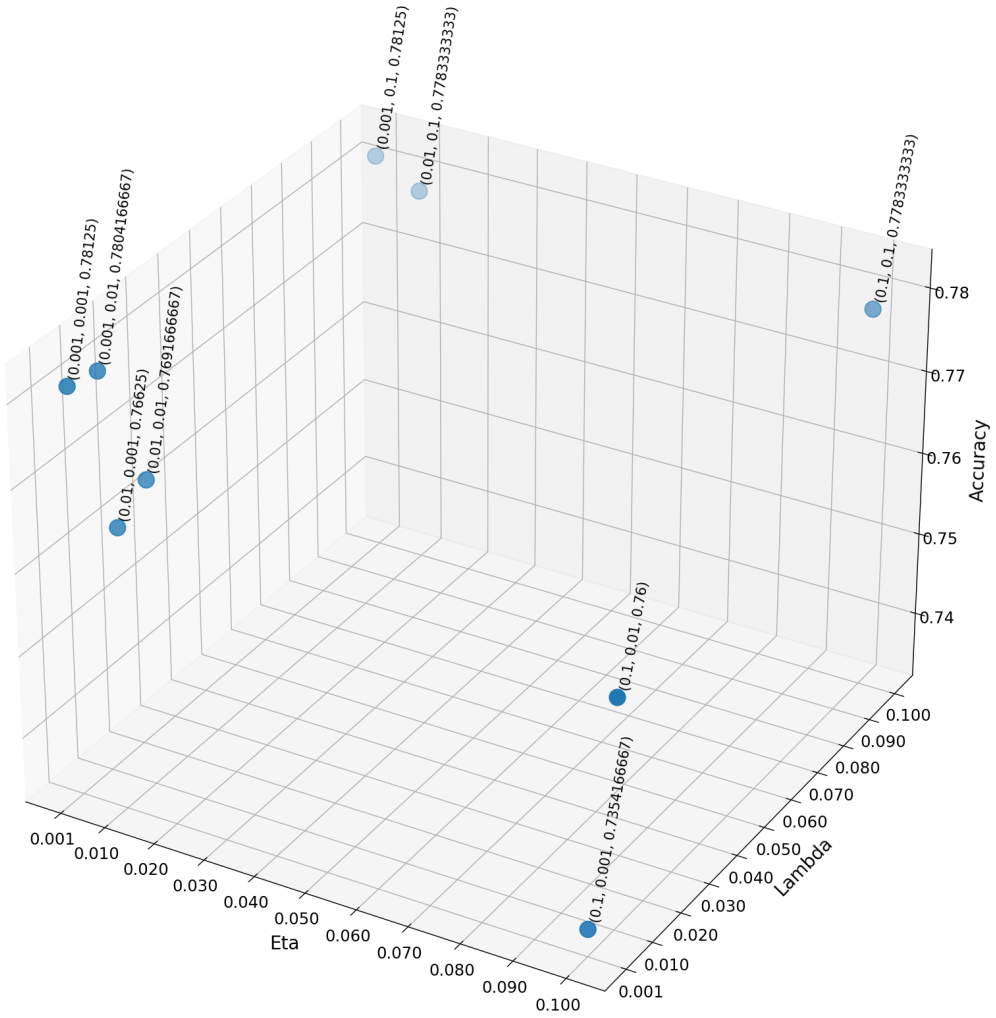
Because the probability of $(X_i|Y)$ does not equal $(X_j|Y)$, and as the document size grows, it would be more unlikely that the parameters for this model would be accurate.

Question 2: Re-train your Naive Bayes classifier for values of β between .00001 and 1 and report the accuracy over the test set for each value of β. Create a plot with values of β 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 β [5 points]

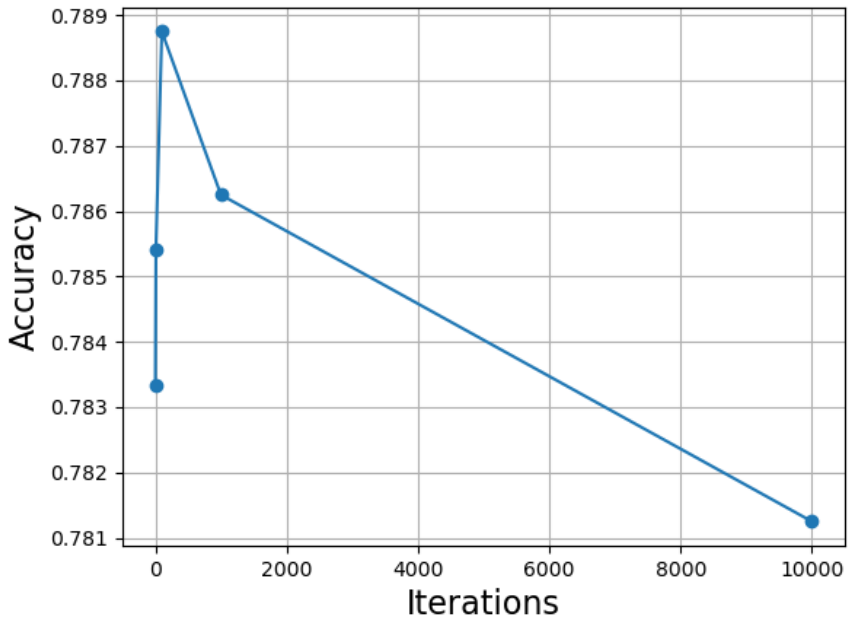
In this Naive Bayes implementation the strength of the prior directly depends on what value of β is used. This prior allows the algorithm to overcome the black swan paradox where if a word isn't seen for a particular newsgroup it assumes the probability is 0. By adding an assumed prior it moves the probability up from 0 to some value based on said prior. That way if a document in the testing data belonging to a particular news group has words in it that were not in any of the documents from the training data, it still gets a high enough probability with the prior that the document is classified correctly. In the graph below, for the first section as β is increasing, the strength of the prior is increasing too, which in turn is allowing the algorithm to better combat the black swan paradox. However the accuracy decreases after reaching β=1 because at that point the prior is creating a bias that is inaccurately classifying documents.



Question 3: Re-train your Logistic Regression classifier for values of η starting from 0.01 to 0.001, λ = 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 [5 points]



Accuracy plotted against the learning rate, eta (η), and the penalty lambda (λ).



Accuracy plotted against number of training iterations.

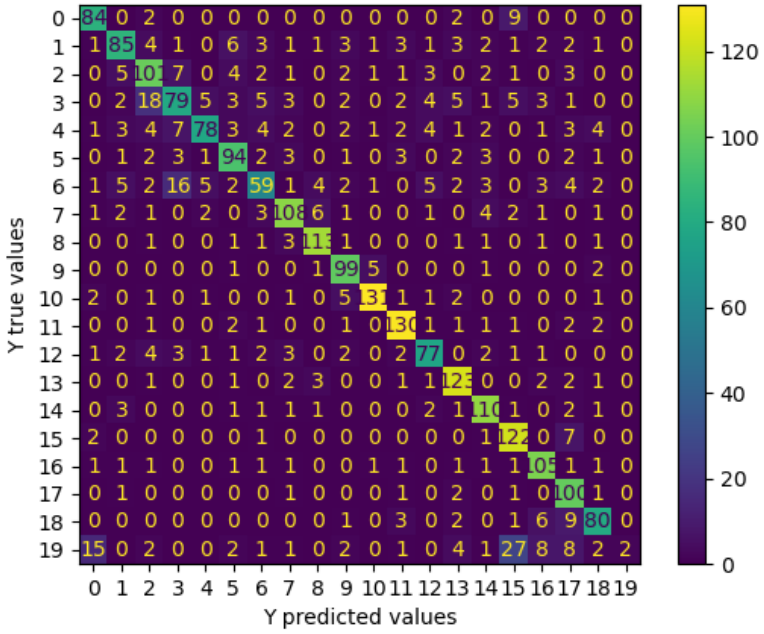
Iterations	eta	lambda	accuracy
10000	0.1	0.1	0.7783333333
10000	0.1	0.01	0.76
10000	0.1	0.001	0.7354166667
10000	0.01	0.1	0.7783333333
10000	0.001	0.1	0.78125
10000	0.01	0.01	0.7691666667
10000	0.001	0.01	0.7804166667
10000	0.01	0.001	0.76625
10000	0.001	0.001	0.78125

Table of parameter changes and associated accuracies.

Overall we saw only a modest difference of accuracy during training/testing when varying either the learning rate, eta (η), or the penalty lambda (λ). The accuracy range for these different parameters was from a minimum of 0.74 (eta = 0.1, lambda = 0.001, iterations = 10000) to a maximum of 0.78 (eta = 0.001, lambda = 0.01, iterations = 10000). For this testing, the iterations

remained constant at 10,000. See the above table for specific testing values and results. We also found no significant change in accuracy with changes in the number of iterations. This also was noted with only one iteration, which led us to believe that there is an error in our code, or the calculations. However, a search for this error was unrevealing. We also attempted to use a threshold difference of likelihood estimates of 0.00001 to stop iterations. Unfortunately this threshold cutoff resulted in iterations nearing one million at which point our computer crashed with a frozen screen and had to be restarted.

Question 4: 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 cij is the number of times a document with ground truth category j was classified as category i). [5 points]



Confusion matrix for the logistic regression classifier.

Note: the newsgroup numbering is reduced by a value of 1 due to zero based array indexing.

Question 5: Are there any newsgroups that the algorithm(s) confuse more often than others? Why do you think this is? [5 points]

There are approximately 4 newsgroups that the confusion matrix suggests significant misclassifying. The most significant appears to be the prediction of the true value of newsgroup

19 being misclassified as newsgroup 15. This also occurs for the prediction of the true value of newsgroup 19 being misclassified as newsgroup 0. Two other similar misclassifications occur when there are predictions of the true value of 6 being misclassified as 3, and 3 being misclassified as 2. We reran a new train/test split and repeated a new confusion matrix with similar results for the misclassification of newsgroup 19 for 15 and 0, however the other misclassifications have fallen out. This suggests to us that there are possible words in newsgroup 19 that are very similar to newsgroups 0 and 15, or possibly newsgroup 19 has a high frequency of stop words that can mislead the algorithm.

Question 6: 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 give high scores to those words that appear frequently in one or a few of the newsgroups but not in all of them. Words that are used frequently in general English (the, of, in, at, etc) should have low scores. Words that appear only extremely rarely in the datasets should also have low scores.[5 points]

One method would be to use a known algorithm such as Entropy to calculate the information gain for each word. Those words with a higher information gain are 'relied' on more by the classifier to make the classification. However this method when implemented takes a long time to run with the size of the data set for this assignment.
Another method that is implementable in the available time is to first normalize the training data by summing each column and then dividing all word count values by the sum. This will result in a data set where each column sums to 1 and the individual values provide what percentage of the word appearance occurs in each news group. After this is done, split the normalized data into subsets based on what newsgroup each document belongs to. This will create subsets such that each document in the subset belongs to the same newsgroup. The next step would then be to sum all the columns of these subsets. Finally this leaves you with the percentages of each word's occurrence per newsgroup. If a word is used often in general English and as such occurs frequently across all newsgroups the maximum percentage will be much lower. Meanwhile words that are only used in one or a few newsgroups will have a much higher max percentage. This means that the higher the percentage the more the classifier 'relies on' the word for classification. Due to words that are used only extremely rarely having a higher max percentage where they appear as well, there would need to be a minimum usage threshold in order to rule those words out.

Question 7: Implement your method, and print out the 100 words with the highest measure. [5 points]

The list of words proved below was calculated using the second method talked about in question 6. This was due to time constraints regarding the running of the first method.

Minimum Usage: 175
['nhl', 'arabs', 'stephanopoulos', 'armenian', 'armenia', 'cramer', 'armenians', 'turkish', 'turks', 'israeli', 'xterm', 'arab', 'bike', 'firearms', 'jpeg', 'hockey', 'escrow', 'clipper', 'orbit', 'turkey', 'launch',

'nsa', 'encryption', 'det', 'patients', 'lib', 'genocide', 'privacy', 'israel', 'shuttle', 'dod', 'widget', 'msg', 'ide', 'des', 'motif', 'bios', 'pgp', 'ride', 'moon', 'um', 'satellite', 'disease', 'pit', 'gun', 'cars', 'catholic', 'sin', 'henry', 'jews', 'mission', 'atheism', 'controller', 'secure', 'muslims', 'scsi', 'shipping', 'muslim', 'keys', 'church', 'doctor', 'guns', 'rutgers', 'soviet', 'baseball', 'key', 'christ', 'image', 'weapon', 'window', 'batf', 'cup', 'algorithm', 'weapons', 'scripture', 'spirit', 'space', 'images', 'entry', 'sale', 'mary', 'bmw', 'gif', 'enforcement', 'ball', 'car', 'isa', 'jewish', 'medical', 'graphics', 'islam', 'treatment', 'compound', 'medicine', 'holy', 'dx', 'processing', 'fbi', 'server', 'greek']

Minimum Usage: 125
['athos', 'argic', 'nhl', 'arabs', 'pitching', 'wiretap', 'istanbul', 'candida', 'stephanopoulos', 'nyr', 'wolverine', 'ripem', 'pitcher', 'leafs', 'serdar', 'armenian', 'armenia', 'cramer', 'armenians', 'azerbaijan', 'ei', 'spacecraft', 'crypto', 'ottoman', 'villages', 'lunar', 'turkish', 'turks', 'palestinians', 'israeli', 'penguins', 'xterm', 'rsa', 'arab', 'hitter', 'cryptography', 'infection', 'bike', 'firearms', 'bikes', 'palestinian', 'jpeg', 'flyers', 'puck', 'hockey', 'palestine', 'encrypted', 'escrow', 'clipper', 'playoffs', 'orbit', 'turkey', 'launch', 'nsa', 'cubs', 'gopher', 'encryption', 'det', 'optilink', 'patients', 'occupied', 'lib', 'genocide', 'xview', 'clayton', 'greeks', 'privacy', 'israel', 'stl', 'di', 'shuttle', 'diet', 'dod', 'riding', 'bos', 'blues', 'widget', 'braves', 'tor', 'cdt', 'motorcycle', 'espn', 'greece', 'msg', 'soldiers', 'ide', 'kuwait', 'irq', 'nist', 'des', 'resurrection', 'solar', 'helmet', 'pens', 'motif', 'cancer', 'widgets', 'bios', 'pgp', 'ride']