Naive Bayes and Logistic Regression Topic Classifiers

Damian Franco
Dept. of Computer Science
University of New Mexico
dfranco24@unm.edu

Meiling Traeger
Dept. of Computer Science
University of New Mexico
meilingt@unm.edu

***Abstract*—This project is intended to teach Naive Bayes and Logistic Regression Topic Classifiers. We will be utilizing newsgroup documents to sort them into specific newsgroups. This project's goal is to mainly work with text classification and text clustering. Understanding Naive Bayes and Logistic Regression is very important to understanding the overall goal and accomplishments made in machine learning. Both approaches are proven to be the building blocks of many other approaches and models in machine learning. Improving our knowledge and development skills in both these models is very important to us as a team. Our results show that both of these implementations are very useful and there are some pros and cons to each approach. Text classification, dealing with large data sets and comparing Naive Bayes and Logistic Regression are just some things that you will review in this paper.**

***Index Terms*—Naive Bayes, Logistic Regression, text clustering, text classifying, big data, machine learning**

Link to Jupyter notebook on Google Colab.

I. Introduction

Naive Bayes is a probabilistic machine learning algorithm that is commonly used for classification tasks. It is based on Bayes' theorem, which describes the probability of an event based on prior knowledge of conditions that might be related to the event. Naive Bayes assumes that the presence or absence of a particular feature is independent of the presence or absence of any other feature, hence the name "naive". This assumption simplifies the calculation of probabilities and makes the algorithm computationally efficient. Naive Bayes calculates the probability of each class for a given set of input features, based on the training data. It does this by multiplying the probability of each input feature given the class, assuming that the features are conditionally independent. The class with the highest probability is then selected as the predicted class. Naive Bayes can handle both binary and multi-class classification problems, and is particularly useful when there are many input features or when the training data is limited.

Logistic Regression is a statistical machine learning algorithm used for binary classification tasks, where the goal is to predict whether an input belongs to one of two possible classes. It models the relationship between the input features and the output class using a logistic function, which maps any real-valued input to a probability value between 0 and 1. Logistic Regression assumes that the relationship between the input features and the output class is linear, and estimates the probability of the output class given the input features using a logistic function, also known as a sigmoid function. The logistic function maps the linear combination of the input features to a probability value, which can be interpreted as the likelihood of the input belonging to the positive class. Logistic Regression finds the optimal decision boundary that separates the two classes by minimizing a cost function, such as the negative log-likelihood or the cross-entropy loss, using optimization algorithms such as gradient descent. The decision boundary can be used to classify new input instances based on their feature values. Logistic Regression is a simple and interpretative algorithm that can handle both binary and multi-class classification tasks.

Naive Bayes is considered a computationally efficient algorithm because it requires a relatively small amount of training data and can handle high-dimensional data easily. Naive Bayes has a simple probabilistic model that can be trained quickly and easily, making it usable for real-time applications.

On the other hand, Logistic Regression can be computationally expensive, especially when dealing with a large number of input features. Logistic Regression requires more training data to generalize well and may require more complex optimization methods to achieve good performance. However, Logistic Regression can handle more complex decision boundaries and may perform better when the relationship between the input features and the output class is non-linear.

Text clustering, also known as text clustering analysis or text clustering algorithm, is a technique that involves grouping similar documents or text data into clusters based on their content or topics. The goal of text clustering is to identify patterns and similarities within a large set of documents and to group them into clusters based on their content or topic. Text classification, also known as text categorization, is a technique that involves assigning predefined categories or labels to text data based on their content or topics. The goal of text classification is to classify a given document or text data into one or more predefined categories or labels.

The main difference between text clustering and text classification is that text clustering involves grouping similar documents or text data into clusters based on their content or topics, while text classification involves assigning predefined categories or labels to text data based on their content or

topics. While text clustering can be used to discover topics and patterns within a large set of text data, text classification can be used to categorize individual documents or text data into specific categories or labels.

Our goal with this project is to understand and attempt to implement Naive Bayes and Logistic Regression models. We want to understand the differences between the two models and why those differences occur. Logistic Regression and Naive Bayes are both very important aspects of machine learning and without these two approaches, we would not be nearly as far in innovation within our field as we would normally be. Regardless, we want to attempt to get at least 75% predictions correct with our models and are excited to get started.

II. Methods

The first challenge that our team must tackle was the handling of large data sets that does not fit into RAM memory in a single session. We could not simply load the data into a Pandas database, but rather must find a new approach that was more efficient in memory. There were multiple approaches taken but most fell flat when using in practice. The approaches that did not work for us was using the Dask parrallization library, reading and running in chunks of data using Pandas data frames, and other libraries that might of helped us compress the data. We had no luck until we tried out chunking alongside compression of the data in .NPY file. A .NPY file are the file formats that are used to store NumPy arrays that can be directly read into data without the use of a mediator like Pandas. We first chunked the data in over 20 chunks for our training data set and over 10 for our testing data set. We then would read in multiple chunks directly into a NumPy array that represented a "matrix" of our data. This was uses to train, test and validate our model. NumPy arrays were much more efficient in our case than Pandas data frames and operations on them had a stark difference

After loading the data, we must pre=process our data which involved splitting our data into validation, training and testing sets. We dropped unnecessary columns like the class column and id column in the training set. NumPy arrays are very useful, but for this part of our development, we used Pandas data frames to load in chunks and export those chunks as the .NPY files that we discussed earlier.

Our Juypiter notebook consists of three main parts. The first is the data pre-processing code. You can view all the code we used to generate our chunks of data and .NPY files. There is also reading in a data checking in Pandas data frames done within this section to make sure we do not have any missing or wrong data. The next section is our Model 1 or Naive Bayes model section. You will find subsections that show how we implemented MLE, MAP and how we classified and predicted values in each the validation and testing set. Training for our Naive Bayes model is also done here. The last section will be the Logistic Regression method that shows how we implemented our logistic regression model. This is much similar to the Naive Bayes section, but this shows how we implemented Gradient Descent, initialized our weight matrix and how our model is trained. As you can notice, both our model for Naive Bayes and Logistic Regression are in their own Python classes. We did this to save our entire model to Pickle (.pkl) files to read in for later. This allowed us to use our time wisely instead of re-training our model every single time that we want to test some new data in our model. Our code also has our code for plots in their respective sections and includes our proposed method to find the most important feature/word in the first data pre-processing section.

A. Naive Bayes (Model 1)

In model 1 we have a document D containing d words $X_1, \ldots, X_d$. The value of the random variable is X_i which is the word found in position i in the document. We are looking to predict the label Y of the document, which can be one of m categories. The equation we use is:

$$P(Y|X_1...X_d) \propto P(X_1...X_d|Y)P(Y) = P(Y)\prod_{i=1}^{d} P(X_i|Y) \tag{1}$$

Each X_i is sampled from some distribution that depends on its position X_i and the document category Y. Since our data is discrete, we assume $P(X_i|Y)$ is a multinomial distribution over some vocabulary V_i; that is, each X_i can take one of $|V|$ possible values corresponding to the words in the vocabulary. In this model we are assuming that for any pair of document positions i and j, $P(X_i|Y)$ may be completely different from $P(X_j|Y)$.

B. Naive Bayes with MLE & MAP (Model 2)

The second model that we look at is Naive Bayes with MLE & MAP. In this model we make the assumption that $\forall i, j \quad P(X_i|Y) = P(X_j|Y)$. In addition to estimating $P(Y)$, we also estimate the parameters for the single distribution $P(X|Y)$, which is equal to $P(X_i|Y)$ for all X_i.

We implement model 2 by first implementing the Naive Bayes classifier from model 1. We then estimate $P(Y)$ using the MLE and estimate $P(X|Y)$ using MAP estimate with the prior distribution Dirichelet$(1+\beta, \ldots, 1+\beta)$, where $\beta = 1/|V|$, V = vocabulary.

$$\textbf{MLE for P(Y)} : P(Y_k) = \frac{\text{num of docs labeled } Y_k}{\text{total num of docs}} \tag{2}$$

$$\textbf{MAP for } P(X|Y) : P(X_i|Y_k) = \frac{\text{count of } x_i \text{ in } Y_k + (\alpha - 1)}{\text{total words in } Y_k + (\alpha - 1) \times \text{length of vocab list}} \tag{3}$$

where $\alpha = 1 + \beta, \beta = 1/|V|$. Then we classify using

$$\textbf{Classify} : Y^{\text{new}} = \text{argmax}\left[\log_2(P(Y_k)) + \sum_i (\text{num of } X_i^{\text{new}}) \log_2(P(X_i|Y_k))\right] \tag{4}$$

In the initial implementation, a prior Dirichlet distribution was used to estimate $P(X|Y)$, with the parameter b set to $1/|V|$. However, choosing an appropriate prior distribution can

be a challenging task in Bayesian learning. It typically involves either using domain knowledge to make an informed choice or analyze the performance of various values on the same validation set. We aim to analyze how the performance on the testing set is affected by changes the changes in β.

Within our code, we used dictionary objects to correlate the measured likelihood/probability values to the word. The model will then take the trained likelihood and prior numerical values and attempt to correctly classify the given instance based off of those two values. We set our α and β values to various different values that we will discuss in our results and discussion sections. When classifying a new instance, we used the log likelihood of both MAP and MLE results to more confidently predict a value.

C. Logistic Regression (Model 3)

For model 3 we use logistic regression to maximize the conditional data likelihood using:

$$P(D_y \mid D_x, w) = \sum_{j=1}^{N} \tag{5}$$

In $P(y^i|x^i, w)$

$= \sum_{j=1} y^i(w_0 + \sum_{i}^{n} w_i x_i) - \ln(1 + exp(w_o + \sum_{i}^{N} w_i x_i))$

This function can be optimized using the gradient ascent/descent algorithm.

Multinomial Logistic Regression and Gradient Descent Implementation

- m, the number of examples
- k, the number of classes
- n, the number of attributes each example has
- η, a learning rate
- λ, a penalty term
- Δ, a k x m matrix where $\Delta_j i = \delta(Y^i = y_j)$
- X, an m x (n+1) matrix of examples, where $\forall i, X_i 0 = 1$ and $X_i 1$ through $X_i n$ are the attributes for example i
- Y, an m x 1 vector of true classifications for each example
- W, a k x (n +1) matrix of weights
- $P(Y|W, X)$ $exp(WX^T)$, a k x m matrix of probability values. Fill the last row with all 1's, and then normalize each column to sum to one by dividing each value in the column by the sum of the column

The update step for the logistic regression is:

$$W^t + 1 = W^t + \eta((\Delta - P(Y|W, X))X - \lambda W^t) \tag{6}$$

Our implementation in our code is an (almost) direct implementation of the psuedocode above and our class consists of with multiple methods to train and predict our data. We treated this as an object class, which is similar to how libraries set up their models. We have an init method sets the learning rate and the number of iterations for training the model. The fit method takes in the input data and the corresponding labels and trains the model using gradient descent. The predict method takes in a new set of input data and returns the predicted labels using the learned weights and biases. We use the loss function to measure the difference between the predicted probability distribution and the actual labels and adjust the model's weights and biases to minimize this difference during training. We also use one hot encoding is utilized to represent categorical variables as binary features that the model can use to predict the probability of each class. We used this as the normalization step to between our iterations of training. Training consisted of using all these functions to efficiently update our weights for each iteration. We used techniques to save weights to continue to train within multiple sessions just as we did with the Naive Bayes model.

III. Results

We were able to test four to five session for each model with a validation and testing set. You will see scores that we achieved from our predictions based on our accuracy within plots that show our hyper-parameters for our trained model.

A. Naive Bayes with MLE & MAP (Model 2)

For Naive Bayes testing, we tested eight total runs. We first tested with a very "naive" approach by setting our α term to 1. We learned that the α term represents the smoothing parameter used to avoid zero probabilities when estimating the likelihood probabilities using MLE and MAP from the training data, particularly when a particular feature does not appear in the training set for a given class. Since it is the smoothing parameter and can prevent over-fitting and improve the model's generalization performance when the training data is sparse when the α term is large, we estimated that this would be a very good place to start. We noticed this was classifying on our validation set with about 64% accuracy.

We then wanted to choose a value a bit larger to see if we were over-fitting in our model. We chose the α value 2, and found that it was over-fitting our data since we were predicting much worse than the value at 1. Were getting about 55% accuracy with these hyper-parameters. Knowing we were over-fitting, we decided to test the next three testing runs was 0.00001, 0.0001, 0.001, 0.01, 0.1 and 1. We found that our best value for predicting was actually the 0.001 value. This was a not surprising to us because we have heard that with sparse data that larger and too small values for α. We will expand more about this in the questions and answers section. We maxed out our prediction accuracy on our validation set with 82% accuracy and with our validation set and at most 78% accuracy with our testing data set. We think if we tuned our hyper-parameters a bit more than what we did, we would definitely have seen improved results, but due to time constraints we were not able to.

We then tested our model with a very, very small α term which was $\frac{1}{|V|}$ where V is the size of our vocabulary which is 61188. This effectively made our α very small which is would most likely be a bad model to classify. We found that we only classified 9% which is way lower than we expected. We will

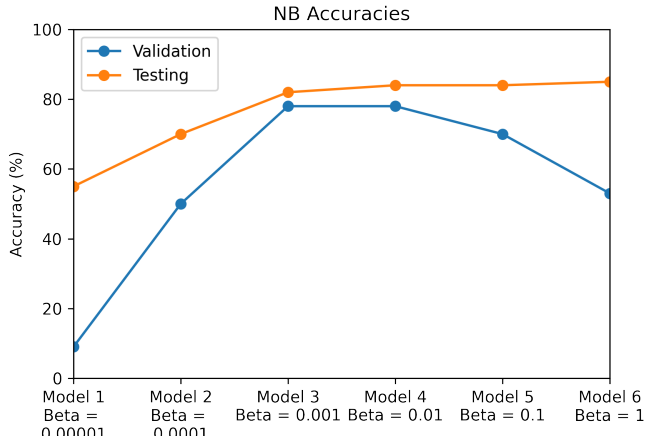

Fig. 1. The figure above shows the Naive Bayes models and the accuracy recorded with both the validation and testing set. We see a drop in accuracy when predicting on the testing set for both lower and higher β values which can indicate over-fitting. The validation data set predicts consistently better than the testing data set.

expand more about this in our section where we answer the questions that were asked in the document.

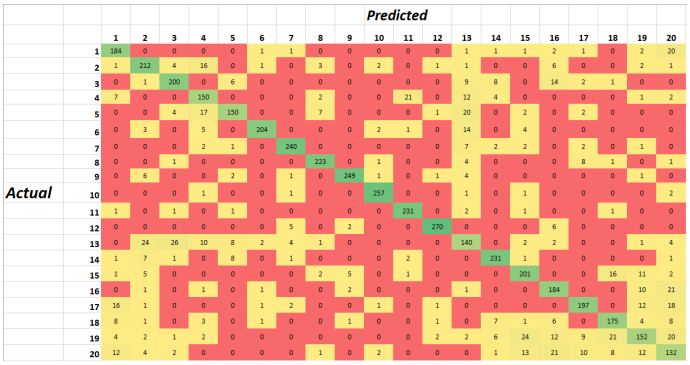

Fig. 2. The figure above shows the Naive Bayes confusion matrix and the number of true and false predictions recorded with both the validation set. The numbers correlate to the ID of each article found in Table I. Sorry about the scaling, this is the only way I could fit this in the paper. You can also find this plot in our zip file with our code. The color scale shows green for anything that is very greatly predicted, and yellow for anything from 1 to 30 misclassificaions. We see an increase in false predictions when articles are very similar to one another, specifically the articles about computers and electronics get confused. The algorithm seems to get confused with the right article to classify when classifying computer articles due to similar words used in each article.

Overall, this model predicted very good and we were excited to see our outcome with it. The predictions made were very promising and with some more tuning, we believe this model could be much better than our results. We learned that this model is a simple and fast that can likely be used for a wide range of classification problems, especially text classification. We believe that with a deeper analysis than we did, that one could examine the contribution of each feature to the prediction and identify potential sources of errors or biases in the model. Very useful technique and we are glad that we learned it and got to implement a good classification model.

B. Logistic Regression (Model 3)

Our testing for Logistic Regression on both validation and testing data sets was conducted with four models of training. We were only able to train four models with values of η, our learning rate, set to 0.001 to 0.01 and λ, our penalty term with values of 0.001 to 0.01 as well. We set our η and λ equal to one another for each run. We also implemented stopping criteria within our Logistic Regression model and ours was based on the number of iterations the model has done. We set our stopping criteria to 8,000 on every single test run that we had, because if we had more then we would run out of RAM memory and will not be able to run our model training.

For our tests we trained our model with our η and λ values equal to 0.001, 0.004, 0.007, 0.01 with both of them being the same. We found that the testing results. We found that with a smaller learning rates that our performance was worse than when η was equal to 0.001 was classifying 10% worse then when it is equal to 0.1. We believe that is due to the fact that with smaller learning rates and penalty rates can lead to smaller changes in the weight matrix which would lead to worse classification predictions. Overall our validation data testing was about 5% better than our testing data which is an indication of over-fitting to us or some other error.

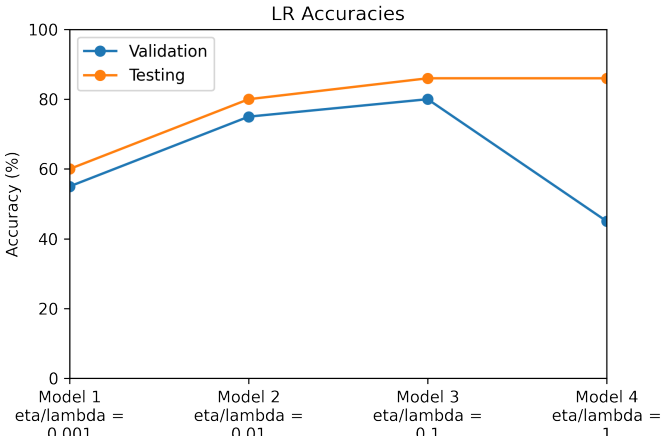

Fig. 3. The figure above shows the Logistic Regression models and the accuracy recorded with both the validation and testing set. We see a drop in accuracy when predicting on the testing set for only higher η and λ values which can indicate over-fitting for higher values.

We would like to note that although that we did get good model runs, that we get to run into an over-fitting model when both the penalty term λ and η were large (0.01 or larger specifically). Encountering a model that is over-fitting is very informative. We implemented a loss function to figure out exactly where we were over-fitting and could not determine where over-fitting exactly occurs without running the model at least five to seven more times. Due to time constraints, we were not able to fully do runs over 0.01 besides our initial run with a learning rate that was over 0.01. Since we did find that our accuracy was better for our validation set compared to our testing set then we assumed it was over-fitting. We wanted to

use our loss function to plot, but did not have the time to use and plot our loss.

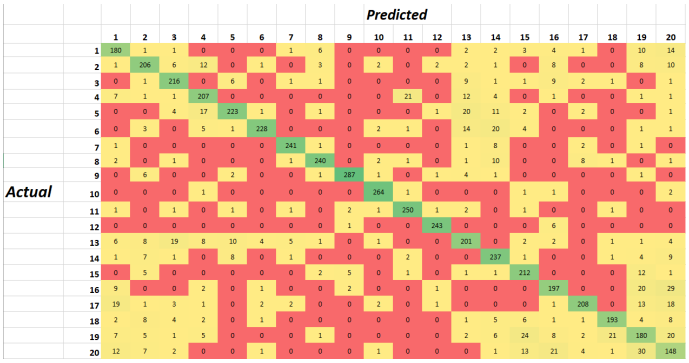

Fig. 4. The figure above shows the Logistic Regression confusion matrix and the number of true and false predictions recorded with both the validation set. The numbers correlate to the ID of each article found in Table I. We see an increase in false predictions when articles are very similar to one another. There is much more misclassificaion distribution than Naive Bayes, but overall less misclassification numbers compared to Naive Bayes so this algorithm tends to misclassify just a bit less that Naive Bayes. The algorithm seems to get confused with the right article to classify, much similar to Naive Bayes.

Overall, Logistic Regression results were very impressive and much better than we expected. Although they were not much higher than the Naive Bayes models, we would like to note that they were higher and with more training, I believe our model would be much better. Overall, Logistic Regression is a simple but effective approach for text classification that can achieve good results with proper tuning and feature engineering that we were able to do a bit here.

IV. Discussion and Conclusions

Both Naive Bayes and Logistic Regression approaches have their advantages and disadvantages. When asking us about which approach that we prefer then we cannot decide. We feel like with Logistic Regression, we had more influence on how well our model will train. This is of course a biased (no pun intended) view because we had more control of various hyper-parameters with Logistic Regression then we did with Naive Bayes. Instead of just changing a single (or in our case two, α & β), we were able to do much more with training Logistic Regression.

When comparing prediction results, we found that Logistic Regression performed a bit more accurate then Naive Bayes. We believe that this is due to the fact that Naive Bayes tends to over-fit if the α term is just a bit too large or too small. We have to fit it perfectly to have a great trained model, but will overall perform worse than Logistic Regression in our case. Logistic Regression does over-fit and arguably over-fits worse than Naive Bayes. In our case, we found that the Logistic Regression predictions to be a bit better than Naive Bayes.

Overall, our models were predicting not at the highest degree that it could. We know that the max we can get from these models should be around 88%, when we were predicting around 78%-80%. We believe we know why this is occurring and it has something to do with how we split our testing data set. In order to train a model and with a large training data set, we did make some sacrifices. We only used about 80% of our training data set then samples about 20% of that data set for validation and the other for testing. Which means that we training with very little data, if we training with a bit more data, then we believe that could maybe improve our performance, alongside adjusting our hyper-parameters effectively. Unfortunately, like we said before multiple times, we unfortunately ran out of time for this project and had to move on to accommodate for other further machine learning projects and other course projects.

There are advantages and disadvantages for Naive Bayes. Naive Bayes is simple and efficient. Let's first talk about some advantages we found. The model was able that can handle large data sets and high-dimensional data such as text with some ease. When a model is trained great, then it will always produce good predictions. Of course, there are also some disadvantages. We know that Naive Bayes assumes that all the features are independent of each other, which may not be true in many real-world applications. This can lead to under performing, especially when the features are correlated. Naive Bayes is very sensitive and hyper-parameter and feature selection can be make or break a model. Bias can occur within training that can be out of our control. The biggest advantage that Naive Bayes has is its time it takes to classify and train data. It is much faster than Logistic Regression which is a big plus for us. We know Naive Bayes is a useful and important but it is also important to be aware of its assumptions and limitations and to choose the appropriate hyper-parameters and train the model multiple times to make sure the model performs correctly.

Logistic Regression is a very great approach to text classification, but there is always cons when discussing the pros. The advantages that Logistic has are that it is a very simple algorithm that is easy to implement if you understand it. The multiple hyper-parameters allow more control over properly fitting a model. This also leads to a disadvantage that there is more of a chance to over-fit if there is no understanding of how the hyper-parameters and the algorithm works, especially normalization and regularization. This also leads to a disadvantage to novice users which is a that there may be a learning curve. This also leads us to believe that this model can be sensitive to outliers and influential observations and tends to over-fit in those cases. The biggest disadvantages in our mind is the training and classifying time. This is a very, very long model to train. Each model we trained took about 2-3 hours with a data set that did not consist of all the data. Our chunks still took us a great amount of time to train. To classify, it took us about 1.5 to 2 hours to classify our training set. Although it is a sometimes complicated and takes a long time, this algorithm predicted better than Naive Bayes and taught us more about machine learning than any other algorithm we have looked at. This approach is a very important topic to discuss and learn in machine learning and we are glad we got to gain some knowledge on it.

We are very happy with the outcome of our models and we learned a great amount about a many various topics. Naive Bayes and Logistic Regression are two very great approaches that have their advantages and disadvantages. Dealing with large data sets is much harder than it seems. Hyper-parameter tuning is the most important step (in our opinion) to generating a great model that is accurate. Overall, this was a very informative and enjoyable experience and we are happy by participating in it.

V. Questions and Answers

1) In your answer sheet, explain in a sentence or two why it explain 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) because there would be a lot of parameters required. In total we would have $50{,}000^{1000}$ *which is very large.:* **This model assumes that each position in a document has its own probability that we would estimate. If we have a large number of 50,000 words then that would mean that we need to calculate the probability using only a small number of 1,000 documents which is not enough data to get accurate probability estimates. We would need much more documents and instances to more accurately predict probability.**

2) Naive Bayes classifier for values of β *between .00001 and 1. plot of beta values on x-axis and accuracy on y-axis. USE logarithmic. Why does accuracy drop for both small and large values of* β*?:* **As you seen in Fig 1, we can see our results for the values for** β **and the accuracy and you can see that that accuracy does drop for both small and large values of** β**. We were taught in class between variance and bias and we believe this is playing a big part in this distribution. We believe that When the** α **value is too low, the likelihood estimates can be over-fit to the training data, resulting in high variance and low bias. When the** α **value is too high, the likelihood estimates can be over-smoothed, resulting in high bias and low variance. This is why we believe that we are getting inaccuracies within the two areas of our predictions.**

3) Logistic Regression classifier for values of η *starting from 0.01 to 0.001,* $\lambda = 0.001$ *to 0.001. report the accuracy over the test for each value. PLOT HERE. explain observations w.r.t accuracy's and sweet spots :* **Our "sweet spots" here were the lower and middle values to** η **and** λ**, but lower accuracy as both of our terms increase. We spoke about this within our paper, but we are most likely experiencing over-fitting here. We will describe this in terms of bias and variance as we did in the last question. When the learning rate** η **is too high, the model may converge too quickly and overshoot the optimal parameter values, which we think can result in the model fitting too closely to the training data and not generalizing well to new data. This will lead to high variance and low bias. For when the penalty term** λ **is too high, the model may over-regularize and under-fit the data which means that the model is too simple and may not capture the important patterns in the data. This would lead to high bias and low variance. This is why we have our "sweet spots" and a bit more errors when estimating our testing set for this model.**

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).: **Our accuracy for both Naive Bayes and Logistic Regression was shown earlier in the paper in Fig. 1 for our Naive Bayes accuracy and Fig. 3 for our Logistic Regression accuracy. You can view and read a little bit about what we thought of the accuracy in those sections. The confusion matrices are also shown alongside the plots in their respective sections above. We also discuss them within the plot description and the section. The Naive Bayes confusion matrix is shown in Fig. 2 and the Logistic Regression confusion matrix is shown in Fig. 4.**

5) Are there any newsgroups that the algorithm(s) confuse more often than others? Why do you think this is?: **Within the confusion matrices for both Naive Bayes and Logistic Regression, the newsgroups that the algorithm confuses on both of them are the ones with very similar topics. These articles that talk about the same topic such as computers or politics will get confused and classify as the wrong. There is a specific group that gets confused the most which is the computer group and I believe its because the words used in each newsgroup text is very precise and will only be used in articles that are computer related. The words and verbiage only occur in those newsgroups article and the algorithm confuses and misclassifies them thinking they are classifying correctly. This is why we think there is more concentrated area of misclassifcation within the area of computers and why misclassificaiton occurs in general.**

6) Propose a method for ranking the words in the data set 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 data set. Finally, in your method this should be an overall ranking for the words, not a per-category ranking:* **To measure the importance of the words that the classifier relies on, we can use entropy and mutual information. Specifically we are going to use mutual information. Mutual information is a measure of the amount of information that two variables share with each other. In the context of feature selection for a Naive Bayes classifier, mutual information is used to measure how much information a feature provides about the target variable. The equation is shown below:**

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \tag{7}$$

To calculate mutual information, we compare the joint distribution of the two variables with the product of their marginal distributions. If the joint distribution is different from what we would expect based on the marginal distributions, then the variables are said to be dependent and have a high mutual information. On the other hand, if the joint distribution is similar to what we would expect based on the marginal distributions, then the variables are said to be independent and have a low mutual information.

In the context of feature importance, we will calculate the mutual information between each feature and our target variable. If a feature has a high mutual information score, it means that the feature is strongly related to the target variable and provides useful information for the Naive Bayes classifier to make predictions. We can apply this approach and then we can view the top 100 features with the highest mutual information scores as the most important features for our Naive Bayes model. We will implement this, remove any common words that will give us large mutual information values like 'the' and then compute mutual information on all features/words with the actual classification within the training data set to find our most important features.

7) Implement your method, set β back to $\frac{1}{|V|}$, and print out the 100 words with the highest measure.: **After implementing our method above for measuring the entropy between each word and measuring importance, we can see the top 100 words that we measured. This list is shown below and is very interesting. There are many words in here like 'fprintf' or 'god' that are interesting because this word seems like a very inclusive word that would only show up in specific articles of a certain topic. Most of these words are very specific and are uncommon which makes some sense why they are more specified words that belong to certain documents and therefore has higher entropy. I also find it interesting that the words that have high entropy and mutual information are unique sports teams and terms that are only used in certain settings. There are a few words in here like the m's or t's that I believe are very obscure words that will most likely not be able to help classify so it is also interesting to point out the obscurities like those. We did remove some words that were very particular for the ease of not viewing a bunch of letters that are used to classify certain documents, but kept some in to make this point. The list of the words are:**

['acsddc', 'adg', 'adsp', 'albicans', 'alomar', 'amdahl', 'atheism', 'athos', 'autopsies', 'azerbaijanians', 'blackhawks', 'bontchev', 'bruins', 'canadiens', 'cardinals', 'christianity', 'countersteering', 'cousineau', 'cryptanalysts', 'cryptographic', 'cryptography', 'deciphering', 'denning', 'dineen', 'escherichia', 'excalibur', 'firearm', 'fprintf', 'gfci', 'goalie', 'goaltender', 'god', 'gregmeister', 'homicides', 'imakefile', 'infante', 'inning', 'ioccc', 'jjrj', 'karabakh', 'konfessionslosesn', 'kovalev', 'leafs', 'lemieux', 'libxmu', 'maciisi', 'memmedov', 'mitteilungsblatt', 'mmmmmmmmmm', 'moncton', 'motorcyclists', 'mutants', 'mydisplay', 'nba', 'nctams', 'nhl', 'nordiques', 'nsmca', 'obfuscated', 'obp', 'oilers', 'oname', 'orbiter', 'pitcher', 'plaintext', 'potvin', 'powerbook', 'powerusersgroupchairman', 'prometheus', 'punisher', 'ranck', 'rayshade', 'rbi', 'recchi', 'ripem', 'rlk', 'rsa', 'sandberg', 'sdpa', 'serdar', 'soderstrom', 'sportstalk', 'srcsignature', 'ssto', 'stderr', 'stepanakert', 'stephanopoulos', 'suprafaxmodem', 'tettleton', 'ttttttttttttt', 'turgeon', 'uccxkvb', 'unverzagt', 'whalers', 'wip', 'xdpyinfo', 'xfor', 'xfree', 'zeitlin', 'zrepachol']

8) 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?: **Data set bias being introduced in this data set could exist in this data set. If we look into the list of words we can see many things that can "date" this data set in the time period in which is was taken. This can possibly introduce a inherit bias for the data set when trying to predict new data. If we predict new newsgroup articles that were posted in 2023, then we might have some issues getting a good result with the models that we trained with the data that was recorded in that time period. This is the very basis of the data set bias and I believe that this data set does suffer from this problem when reflecting its future performance in the real world.**

REFERENCES

[1] T. M. Mitchell et al., Machine learning. wcb, 1997.

[2] Goodfellow, Ian, et al. Deep Learning. The MIT Press, 2017.

[3] Duch W, Adamczak R, Grabczewski K (1996) Extraction of logical rules from training data using backpropagation networks, in: Proc. of the The 1st Online Workshop on Soft Computing, 19-30.Aug.1996, pp. 25-30, available on-line at: http://www.bioele.nuee.nagoya-u.ac.jp/wsc1/