Anthony Galczak
Tristin Glunt
CS529 - Project 2
Naive Bayes' and Log. Reg.

I. Introduction

The following report is about implementing the classification models known as Naive Bayes and Logistic Regression. Once these machine learning models are implemented, their classification abilities are then tested with the 20-NewsGroup dataset. The dataset consists of 12,000 articles, in which each article has one of twenty topic classifications. The total vocabulary for the dataset is 61,188 words. We will present the implementation of our program, and discuss the certain techniques that we used to better fit the news group data. The paper concludes with talking about specific tuning methods to gain higher accuracy for each algorithm. It is expected that the reader is familiar with both algorithms of Naive Bayes and Logistic Regression, as the methodology of the algorithms will not be explained in detail here.

II. Data Analysis / Setup

We were given a fairly hefty dataset to work with on this project and as such we had to do some pre-processing and think about optimization while working. For the world of machine learning, ~2GB of data isn't huge, but definitely forces you to think a bit before loading in files. A critical part of our work was converting our training and testing data both to non-zero or "sparse matrices". The vast majority of our data was zero's so this gave us the ability to really work with the data as the full data set took approximately 20-30 minutes to load in.

A big hurdle right at the beginning of this project was optimization. Our first "naive implementation" of naive bayes' had a runtime of over a day. Utilizing matrix operations instead of blindly using for loops on every data structure was critical to the success of the project.

Even after loading everything in as non-zero's we still had to wrangle the dimensionality of the features. There are 61188 features which quickly leads to problems in both algorithms, but especially logistic regression. We utilized some feature selection methods that will be talked about in sections III and IV in order to combat this problem.

III. Logistic Regression:

Logistic regression is a simple discriminative machine learning algorithm, as the basic model of the algorithm can be re-thought of as a single perceptron with a sigmoid activation function.

We begin our logistic regression problem with two possibilities, the first is that of lr_tuning. Logistic regression tuning splits the training data into two sets: 80% is training data while 20% is validation data. This is done in order for us to test on the validation data while we tune the hyperparameters lambda, eta, and the number of iterations. After the split is made, we by default use all of the features, but our best performance (.898 on Kaggle) is using the 60,000 highest importances of features to train and classify with. This feature selection is

explained in detail in Question 6. Lr_tuning then proceeds to train and predict on the validation data with subsets of possible hyperparameter values.

Logistic Regression training is the heart of the logistic regression algorithm. After initializing the delta matrix, we then initialize the training data matrix ensuring each column sums up to 1. The weight matrix is then initialized as all 0s with the dimensions (# of classes, # of features + 1). We then begin the tuning of the weight matrix by calculating the Prob(Y | W, X) = exp(W*X.T). The weight matrix is then updated with respect to its gradient, weighted by the regularization term lambda. This update process is then repeated for a specified number of iterations to hopefully find the maximum (in this specific case) of the probability function. Refer to question 3 in which we discuss the "sweet" spots of our hyperparameters.

Logistic regression solve simply skips tuning and directly goes into tuning the weight matrix W. Both approaches conclude with logistic regression predict simply taking the the data to be predicted on and recomputing the P(Y | W,X). We then take the maximum index of each column, that is the maximum probability for some class k of Y. These classes are then our predictions given the data.

IV. Naive Bayes:

Naive Bayes' is a generative classifier that is ultimately considered to be a simplified version of a full Bayesian network. The primary difference between them is that Naive Bayes' assumes conditional independence between the features given a class.

Naive Bayes' has a similar structure to logistic regression in that it has two possibilities, nb_tuning and nb_solve. We will talk about nb_tuning first. The data is again split into two sets: 80% of data is the training set while 20% of the data is for validation. The main point of nb_tuning is to tune the Beta parameter as well as building the confusion matrix for how tightly the predictions are correlated. The beta parameter at 0 makes Naive Bayes' vulnerable to the "black swan paradox"; thus we are tuning this parameter for small positive values.

Naive Bayes' train is the primary method for actually evaluating the Naive Bayes' algorithm. First, we determine how many words are in each classification. This is an important parameter to be used in the denominator of the Naive Bayes' equation. Then we calculate the prior probabilities, which maps to P(Y) in the equation. Lastly, in order to do Naive Bayes', we of course have to calculate P(X|Y), i.e. the likelihoods. If we are in tuning, then at this point we will run analytics (confusion matrix, 2D plots) against the validation data after doing predictions. However, if we are Naive Bayes' solve then we will gather our data from 100% of the training data and then do predictions on the testing data. Afterwards, we then output a file corresponding to our predictions for all of the testing data.

V. Conclusion:

After predicting the classifications of the NewsGroup dataset using the testing data with both Logistic Regression and Naive Bayes, we ended up with a higher accuracy of 89% using Naive Bayes with a beta value of 1/100 with using static feature selection of 60000 features using the method from Question 6. However, as time has gone on, we continually find new hyperparameter settings that show accuracy improvement using Logistic Regression up to 84%. Because of this, we believe with continued effort of tuning the hyperparameters one may be able to

find an optimal setting that has a higher accuracy than 89% on the testing data in Kaggle. This, with the possibility of an intelligent dimensionality reduction technique may show that Log. Reg. is capable of achieving higher accuracy. We were surprised that Naive Bayes performed so well due to the assumption that the features are conditionally independent given the class. This assumption is certainly not true given contextual information of an article, where word placement in a sentence heavily impacts the meaning of the sentence. Despite this, Naive Bayes performed extremely well over Logistic Regression.

VI. Future Enhancements

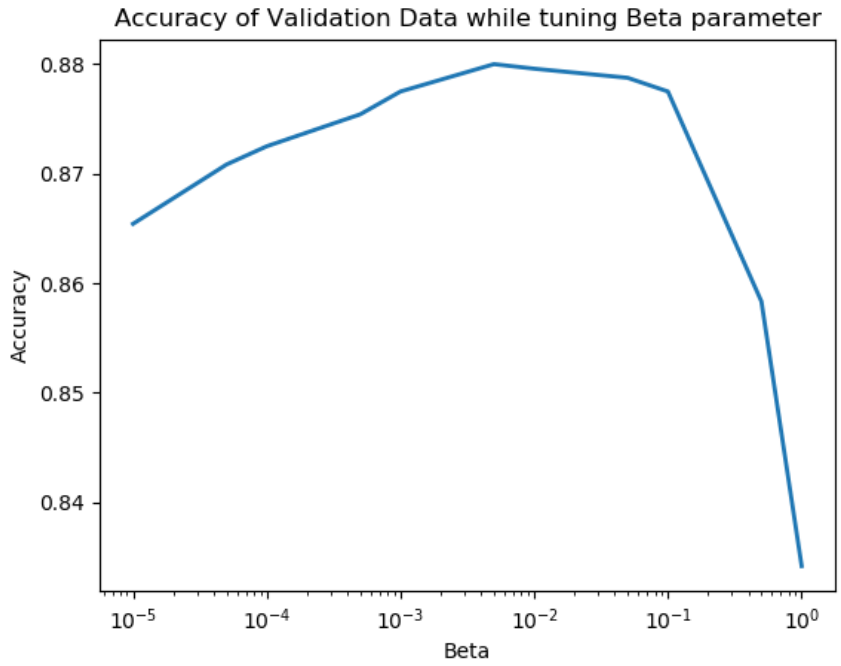
1.) Iteratively test continuous beta values(1/10 - 1/1000) for Naive Bayes' until we get the optimal Beta for our validation set. We did this discretely and got Beta = 1/100.
2.) Re-write determine_likelihoods() in Naive_Bayes.py with matrix operations.
3.) Convert our implementations of Naive Bayes' and Logistic Regression to SKLearn libraries.

Answers to Questions:

Question 1:

Given 1000 documents, 1000 words per document and a 50000 word vocabulary for a sample problem. This means that our given probability matrix would need 1000^{50000} - 1 parameters. I made my best attempt to calculate this number, but both Google and Python gave me "Infinity". The short version is that this amount of parameters is intractable and not anywhere near feasible to use in computation.

Question 2:



As the plot above shows, we got the best accuracy for using Beta variable between 1/100 to 1/10. There's a significant drop off in accuracy for very large (1) and very small (1e-05) values for the Beta variable.

My belief is that very small values for Beta don't give enough scaling for the very large dimensionality of the features. All the very small values of Beta do is prevent the "black swan paradox" here. The very large values of Beta add 1*61188 to the denominator which is an enormous scaling factor which ends up outscaling the actual values of our queries.

Question 3:

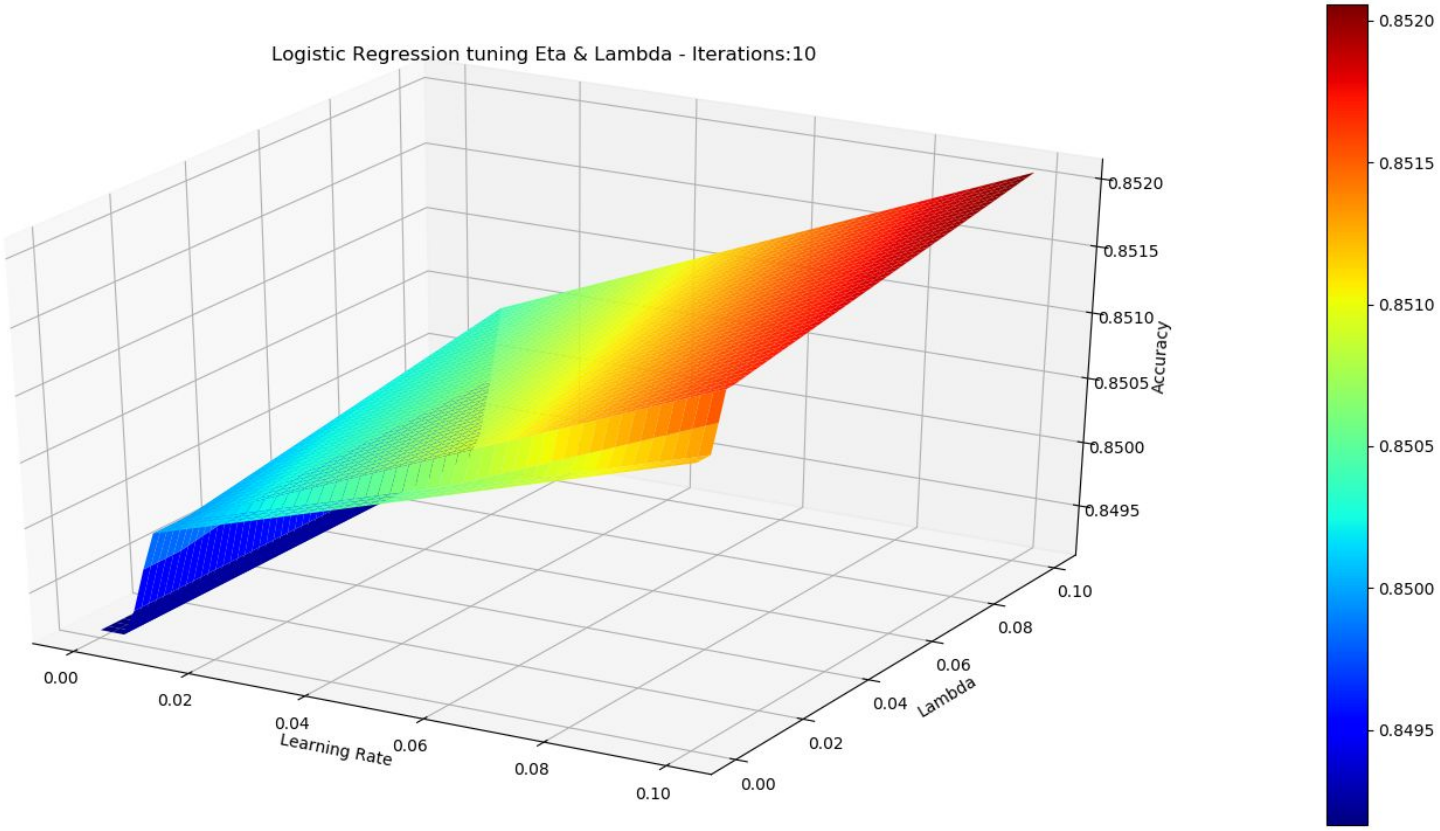
Note: Reference the Additional Figures doc if you'd like to see additional 3D plots. There is more information regarding the specific process for generating each figure.

We did a lot of tuning of hyperparameters for Logistic Regression. The plot below used eta and lambda values ranging from [.0001, .001, .0025, .005, .0075, .01, .1]. I am confident that the most ideal values for both parameters are found between .001 and .01, but .0001 and .1 are added as boundary conditions for this

problem. Additionally, we found that running past 100 iterations wasn't always productive. This could be argued to be "convergent behavior". 1000 iterations or more was definitely leading to decreased accuracy as well. The two conclusions that we can be sure of is that the ideal range of values are .001-.01 and 100 iterations is a sweet spot for accuracy.

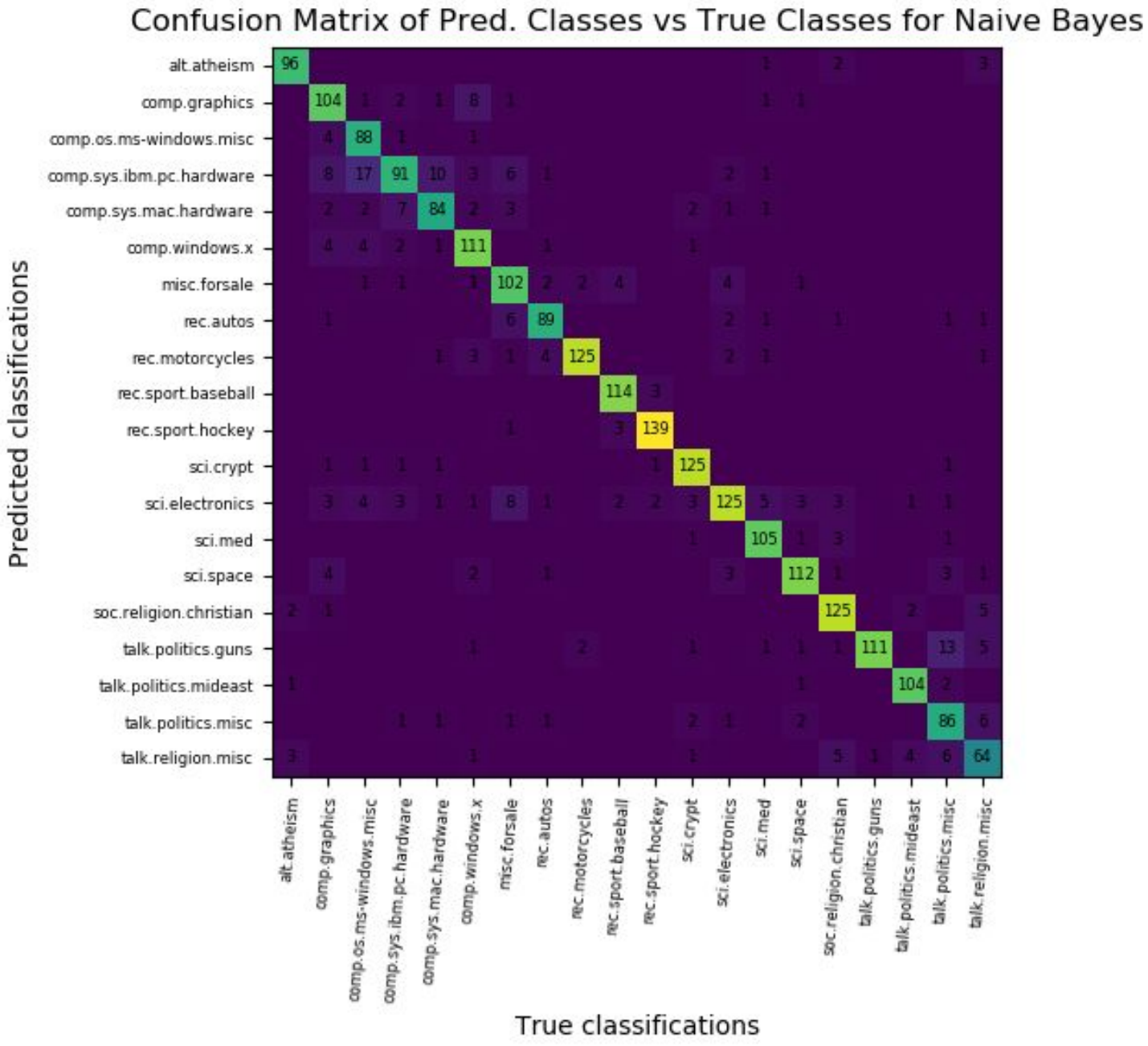
We ran hundreds of tuning runs for Logistic Regression and the following conclusions are biased opinions about the hyperparameters eta, lambda and the number of iterations. Since we are using a pretty small amount of data (n=12000) and 20% of that is validation; the results are heavily dependent on the random seed that we use for validation data. The "dumb" feature selection runs were done using a static validation dataset which made our analysis a bit easier, but still didn't make the conclusions cut and dry. I believe that as the number of iterations go up that you should decrease eta (learning rate) and increase lambda (penalty term). This is a big generalization, but it is based upon the fact that as your iterations increases your weight matrix will grow which will lead to overfitting. Thus, you want your lambda to reduce this "blow-up" that leads to overfitting. Additionally, you want to take longer for the learning rate to converge with a high number of iterations.

Lastly, logistic regression isn't super speedy with lots of iterations so if one decides to use a relatively low number of iterations (i.e., 10, like in the figure below) then you will want to use a rather high learning rate so that you can converge faster. It appears from the figure below that when using low iterations the lambda hyperparameter doesn't effect the accuracy very much. This makes sense since you aren't running enough iterations for the weight matrix to grow and overfit.

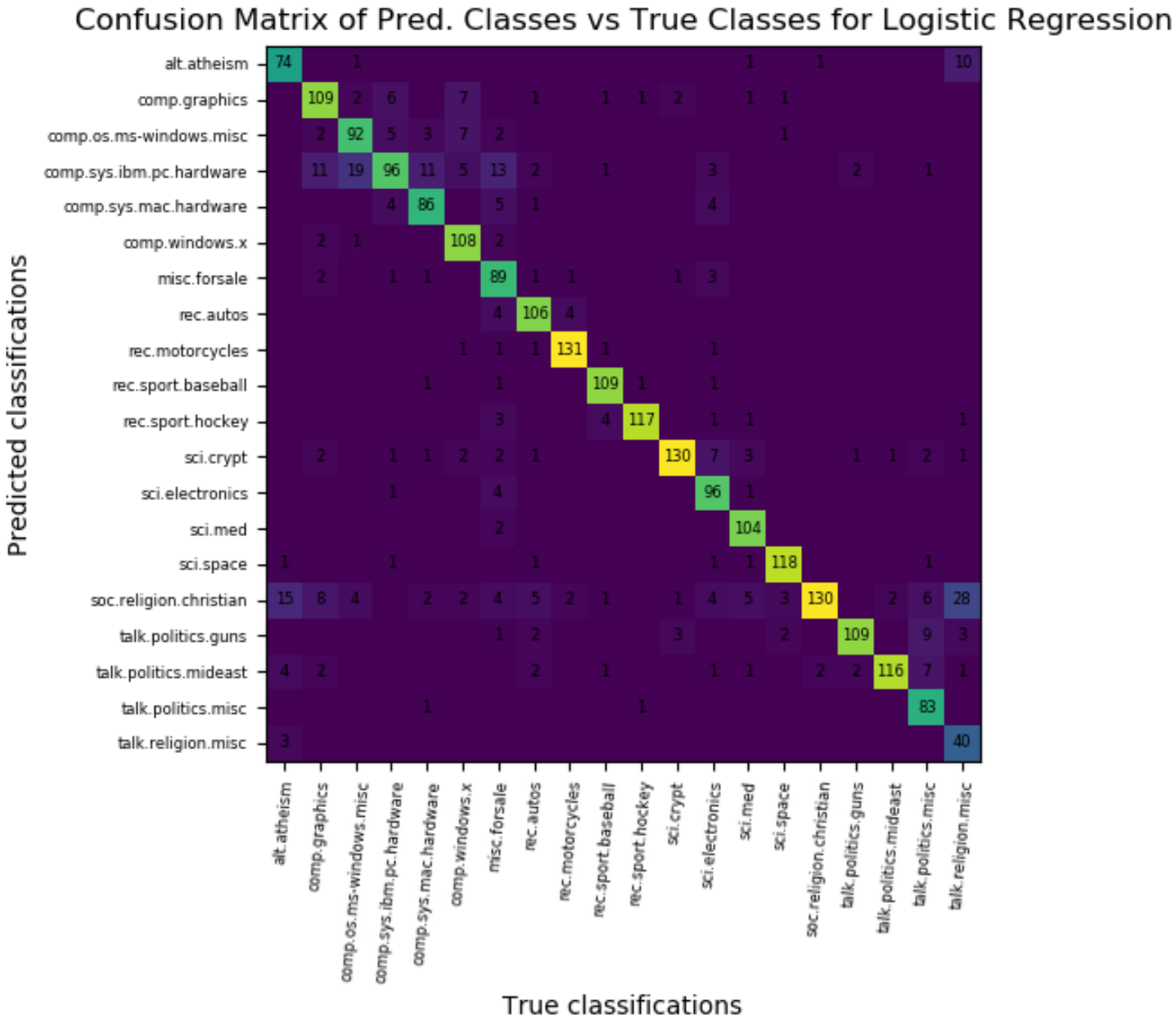


Question 4:

For Naive Bayes' our best testing accuracy on Kaggle was 89.873% with feature selection of 60000 features (non mutual information) and Beta set to .01.



For Logistic Regression our best testing accuracy on Kaggle was 84.35% with feature selection of 60000 features (non mutual information) and eta, lambda both set to .001.



For the mutual information feature selection; the best we obtained was using Naive Bayes', Beta = 1/100 and 61000 features. This got 89.784% accuracy on the testing data. This is very similar to our other runs without mutual information at 60000 features.

Question 5:

Naive Bayes: For the most part, there are no specific topics being confused with another. Instead, the sub-topics such as "comp" gets confused with other "comp" specific topics. This is most likely due to the general vocabulary

used in these sub-topic articles. Additionally, some topics like Guns and Politics are being slightly confused with one another and forsale and electronics. This confusion makes sense as these topics sometimes mention very similar words when discussing their respective topic.

Logistic Regression: There is a fairly strong diagonal for logistic regression, but there are a few classes that are often misclassified. Specifically, out of 84 samples of talk.religion.misc, 44 of them are misclassified. This makes intuitive sense as they are most commonly misclassified as soc.religion.christian and alt.atheism. Of course, these topics are highly correlated in the words that are used in those documents.
Additionally, many examples as classified as comp.sys.ibm.pc.hardware, but out of the 154 examples that are predicted to be comp.sys.ibm.pc.hardware only 96 of them actually are comp.sys.ibm.pc.hardware. This makes sense as the newsgroups that are misclassified as comp.sys.ibm.pc.hardware are: comp.graphics, comp.os.ms.windows.misc, comp.sys.mac.hardware. These are all general computer topics that could be easily mixed up.

Question 6:

Mutual information quantifies the amount of information obtained by using one variable through observing another variable. In our code, we're currently using sklearns mutual_info_classif function which returns us a list of positive values. In this list, the higher the value the more the classification depends on it, and if the value is 0, the classification is independent of that feature. We take the top 100 maximum values from this list, giving us the 100 most important words found int "top_100_vocabulary.txt". These 100 words are the highest ranking words despite any specific classification.

Question 7:

For sake of space, please refer to "top_100_vocabulary.txt" in our submission. These are the words after running Naive Bayes' with beta set to 1 / 61188. These words were the top 100 most "important" features using the metric from question 6. Although some of these words seem very unimportant and common, they had the best accuracy of 32.6% on a validation set. If you look closely, these 100 words do contain the possible buzzwords for each topic, and also possibly some stop words that could be removed and replaced using a counting metric.

Question 8:

Since the dataset in the NewsGroup problem was independently sampled we don't really see any immediate forms of bias. However, all algorithms have bias and our biggest assumptions are the assumptions of Naive Bayes and that each of our words that we're training on will continue to show up in their respective topics in future articles. Imagine a scenario where all of the articles for politics were coming from CNN's Anderson Cooper. This would clearly not be an independently sampled dataset, and it is very likely that the algorithm may learn how to classify Anderson Cooper's writings on politics and not the topic politics itself. This would be a heavy form of bias, as we're assuming that all future political articles will be from Anderson Cooper. A way to spot this would be to see the most common/useful words for a given topic, and if they have absolutely nothing to do with that topic and they're not commonly used word, then there's a high chance some bias may be induced. Luckily, our dataset does not reflect this issue.