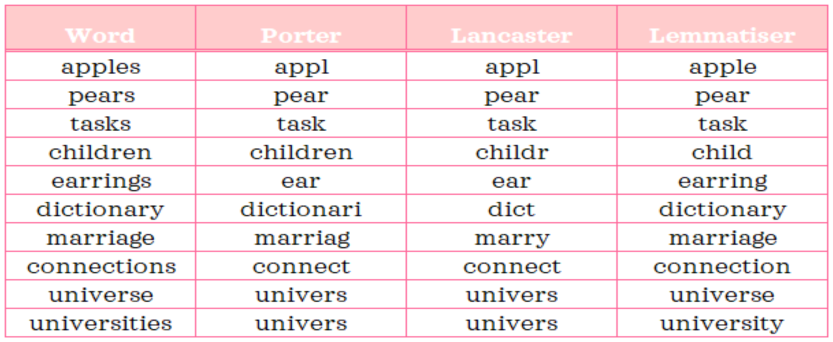
Question 1)

To start off this NLP project, the first goal is to get a sense of our data. To do this I needed to figure out the most common words to effectively get rid of any stop words that might not be helping us in the data. Looking at the most common words in the whole document, "the, and, a, I, is, to, of, was, this, it, in", were in the top 10. These words we can get rid of as they won't really add value to our modelling later. Now we have to build a stop words list, there is one in NLTK library which they caution not to use. Further investing why this might be the case, I looked at what is actually in the stopwords "English" library in the NLTK library. Turns out that some of these words might actually be useful to keep in the case of what we're trying to solve. Words such as "not", "nor", "no", "wouldn't" which are in the stopwords list might be useful. So we can keep those in, and create our own stop words list that we will look for in the stop words list as well as the most common words and see what we can do without from there. Looking at some of the texts we can see there are some punctuation and dashes which can change a meaning of a word. If we are to use the count vectorizer that's imported from NLTK we would lose these punctuation. Punctuations in between numbers such as "/" in "1/10" will mean a lot to us, so when we vectorize the words we can keep those in. In our case I chose to work with work pairs because if the document picks up the word "good" but doesn't pick up the "not" in front of it, it might misclassify. Now that we know what we want to achieve with our bag of words we can go ahead and create a pipeline. I am handpicking some of the stop words in the 'english' stopwords library from NLTK but leaving some of the words that might be useful for our classification as mentioned above and more. We also checked the most common words and added those to the stop word lists that we think will not provide value to our classification. Words such as 'characters', 'take', 'since' and some others. We also pulled the words in bi grams with limited punctuations such as '-' and '/' as described above. We then lemmatized our words which can help us reduce some redundancy in our words. Lemmatisation according to Wikipedia is defined as "is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form." I chose to use lemmatization instead of the porter stemmer because porter stemming sometimes getting rid of the end characters which can change up the meaning of a word. Below is some examples of the difference between porter, lancastor and lemmatiser:



Word	Porter	Lancaster	Lemmatiser
apples	appl	appl	apple
pears	pear	pear	pear
tasks	task	task	task
children	children	childr	child
earrings	ear	ear	earring
dictionary	dictionari	dict	dictionary
marriage	marriag	marry	marriage
connections	connect	connect	connection
universe	univers	univers	universe
universities	univers	univers	university

With n_gram being 1 we have 3620 features, when we change this to bi grams our features scale up to 12181, but when we change the range to 1,3 which I believe will help us during our modeling phase as we have more to compare. Regarding Tf-IDF I chose to keep both as doing some research online it ended being the most reliable.

Below are word clouds that show the most prevalent words, and prevalence is defined in this case by the frequency of the word. The figure on the left show the most common words in the positive reviews, and the figure on the right show the most common words in not positive reviews.

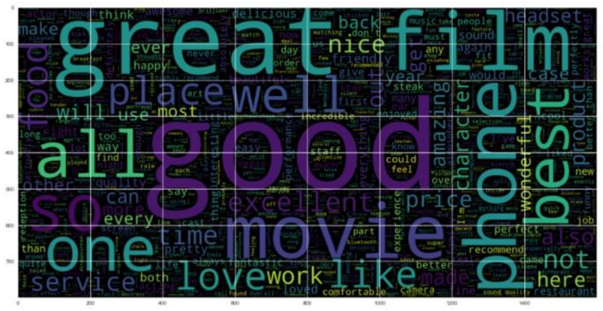


Figure 1 Word cloud of the most common words where the sentiment is positive

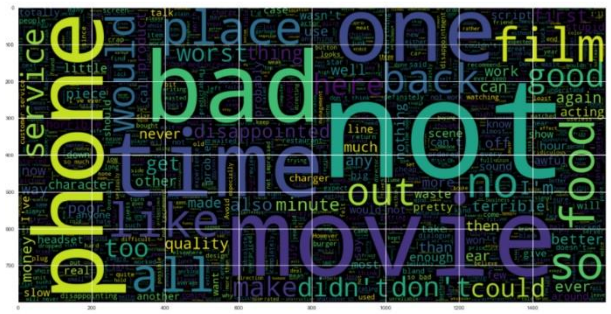
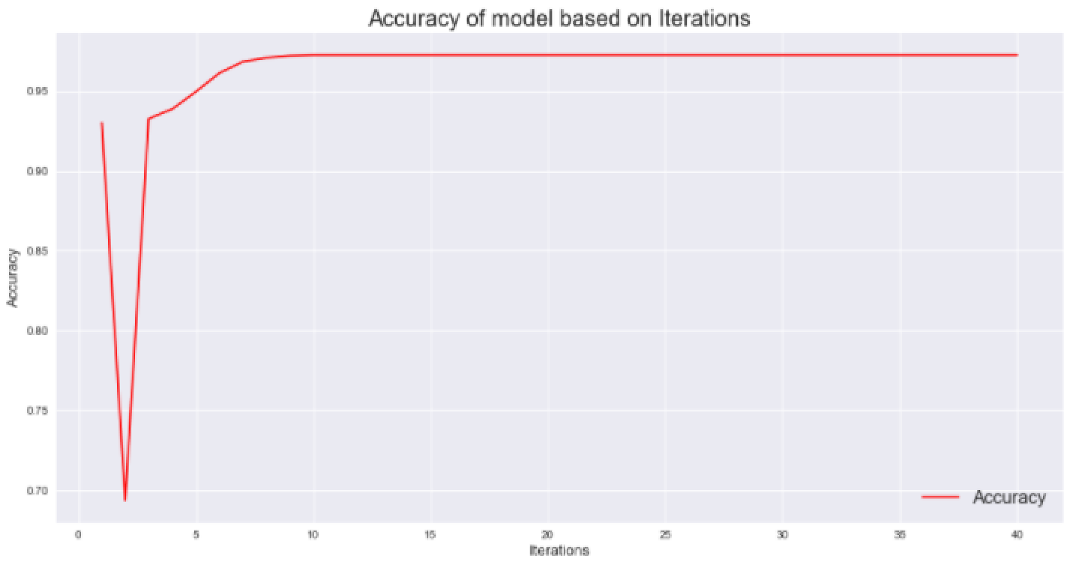


Figure 2 WordCloud of the most common words seen where sentiment is NOT positive

Question 2)

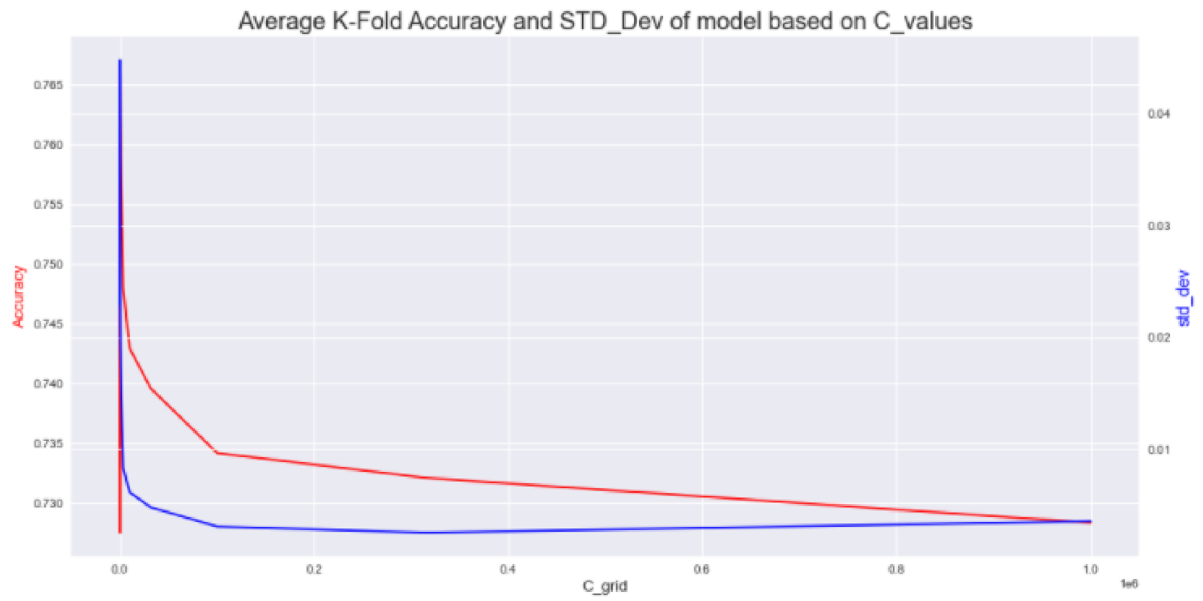
MODEL	TRAINING ACCURACY	LEADERBOARD ACCURACY	LEADERBOARD AUROC
BASELINE LOGREG (N_GRAMS = 1)	.94	0.833	0.912
BASELINE LOGREG (N_GRAMS = 1,2)	.972	0.828	0.91169
BASELINE LOGREG (N_GRAMS = 2,2)	.985	0.657	0.742
LOGREG (MAX_ITER = 10)	.943	0.83	0.912
LOGREG (ITER=10, C= 3.1622)	.991	0.833	0.919

Now that we have our TF-IDF set we can start doing logistic regression. We start with a baseline model and we can check different n_grams and how that performs in the leaderboard. Looks like the unigram did the best in terms of leaderboard error rate but the combination of bigrams and unigrams was a close second. Just the bigrams itself wasn't really that effective and looks like it overfit the training data, so from now when we use parameter tuning we will use on the combination of unigrams and bigrams as this was not only close to the n_grams of 1 accuracy but it also had a great training accuracy. Let's start with the iterations parameter. Looks like iterations in this case doesn't help must as after the 10th iteration the accuracy stays the same.



This shows the accuracy based on the max_iteration parameter in the logistic regression model

Now that we know our max iteration can stay at 10 and anything over will most likely not help we can try another parameter tuning. This time we will try the C parameter. To start it off I gave np.logspace(-9, 6, 31) as a grid to try out the C in the hyper parameter. The ‘C’ refers to how much the classifier should trust the training data. The lower the ‘C’ it gives more weight to the complexity penalty, the higher the ‘C’ the more trust in the training data the model will have.



2Average accuracy and std deviation across k-folds

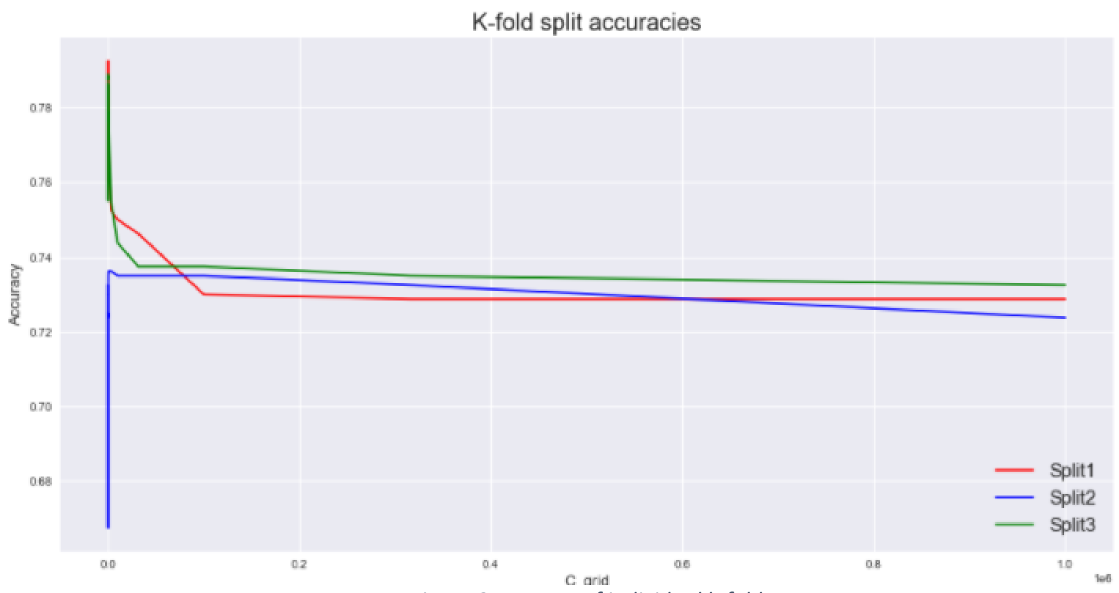
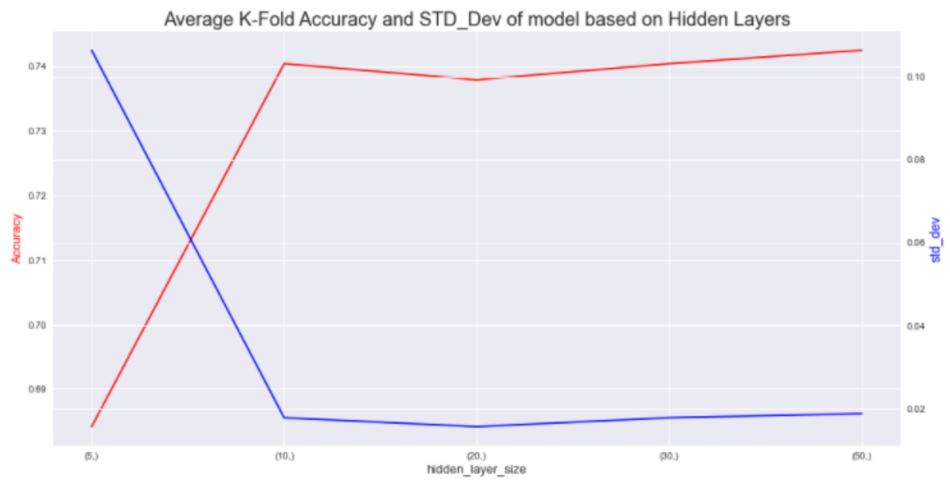


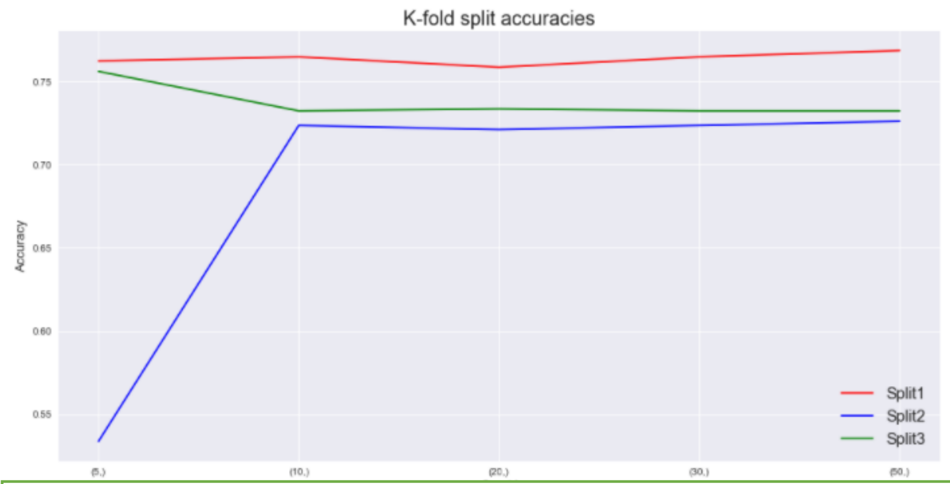
Figure 2Accuracy of individual k-folds

The above diagrams show the average K-fold accuracy and standard deviation based on the C parameter in logistic regression. The second diagram shows the individual accuracies of the 3 K-fold cross validation splits. We can see the uncertainty of the very low C param. The standard deviation is high because the three folds disagreed on the model with that specific C as a parameter. Looking at the table above our models over fit on the training data that we had. This could be because of the way we vectorized our words. Out of all the logistic models that we tried our last model with the hyperparameter max_iter=10 and c= 3.1622 performed the best. This higher C value means that the model put its trust into the training data which explains why that model had the highest training accuracy out of the other models.

Question 3)

MODEL	TRAINING ACCURACY	LEADERBOARD ACCURACY	LEADERBOARD AUROC
BASELINE MLP (N_GRAMS = 1)	.999	0.793	0.878
BASELINE MLP (N_GRAMS = 1,2)	1.0	0.825	0.911
BASELINE MLP (N_GRAMS = 2,2)	.99	0.653	0.746
MLP (HIDDEN_LAYER = (50,))	1.0	0.828	0.911
MLP (HIDDEN_LAYER = (50,), ALPHA=1.0)	1.0	0.832	0.920

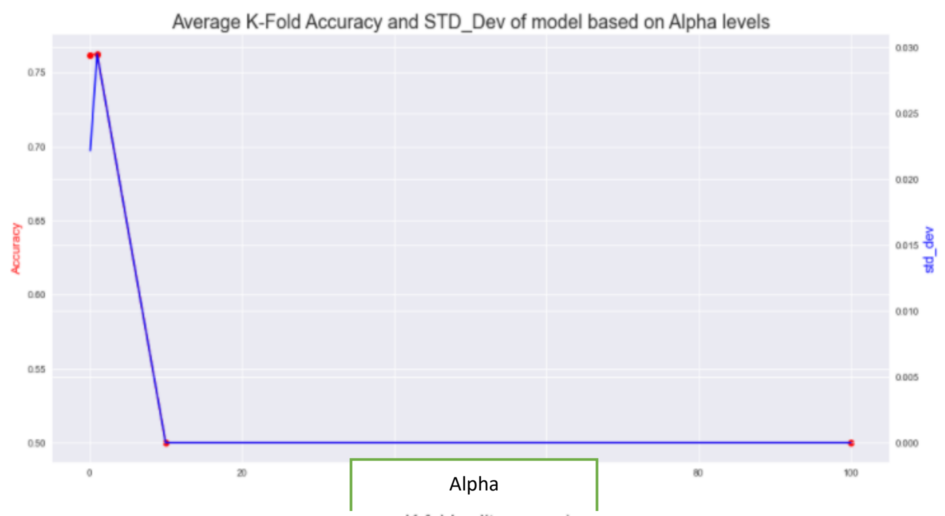


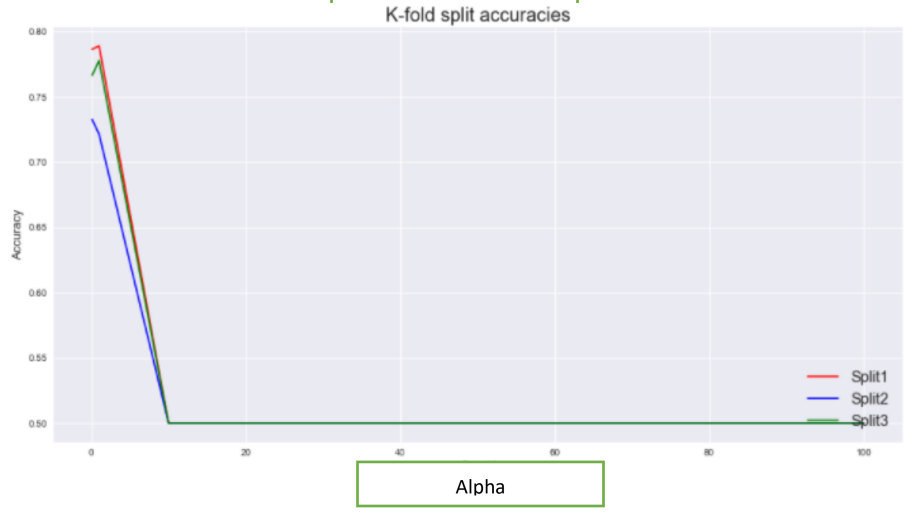


Hidden_layer

Figure 3 K-fold split accuracies of hidden ***layer*** *grid search*

Next we want to try a neural network to see if we can improve our accuracy on the testing set that's on the leaderboard. To do this we use a simple Multilayer perceptron (MLP). Above table shows a breakdown of the MLP models starting with baseline. I wanted to see if the ngrams would change the baseline of the MLP modeling. Turns out the combination of both the unigram and bigrams gave the best leaderboard accuracy and auroc score. For the rest of the MLP modeling we will use the combination of both unigram and bigrams. To start off our parameter tuning I wanted to see how changing up the hidden layers would affect our accuracies on both the training dataset and the leaderboard. Hidden layers can better detect patterns in the data that we have, but too big of a hidden layer can lead to overfit of the training data. The figures here only show some of the parameters that were tried. Initially I tried layers starting at 50, 70, 100, 150. This gave us 50 as being the best parameter so I lowered the parameters to start at 10 and found that 50 was still the most optimal hidden layer. These figures show that as the hidden layer increase the accuracy increase but after a certain point though (not shown) it does tend to overfit the model and lowers our cross-validation STD deviation. In the case above though after the 20 hidden layer mark the k-fold splits tended to agree with each other thus giving a lower standard deviation which is good.

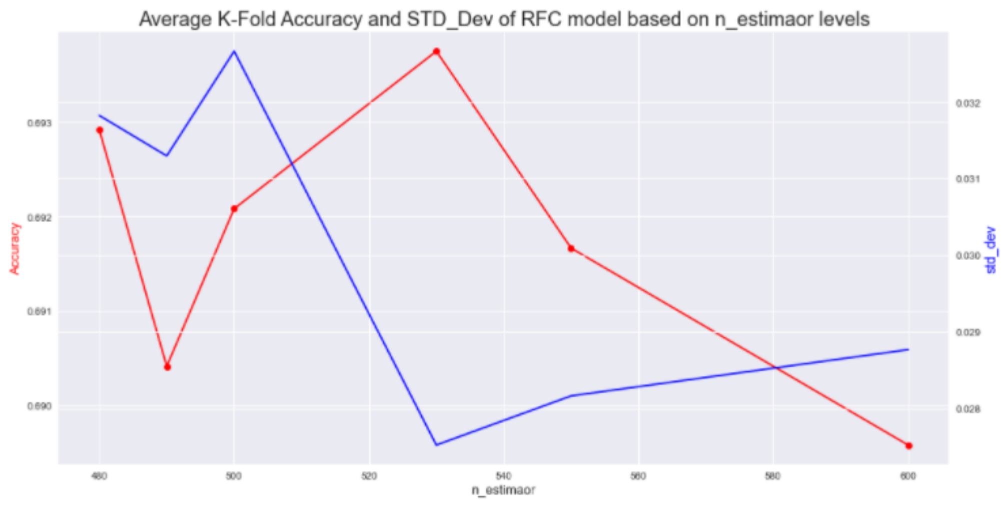


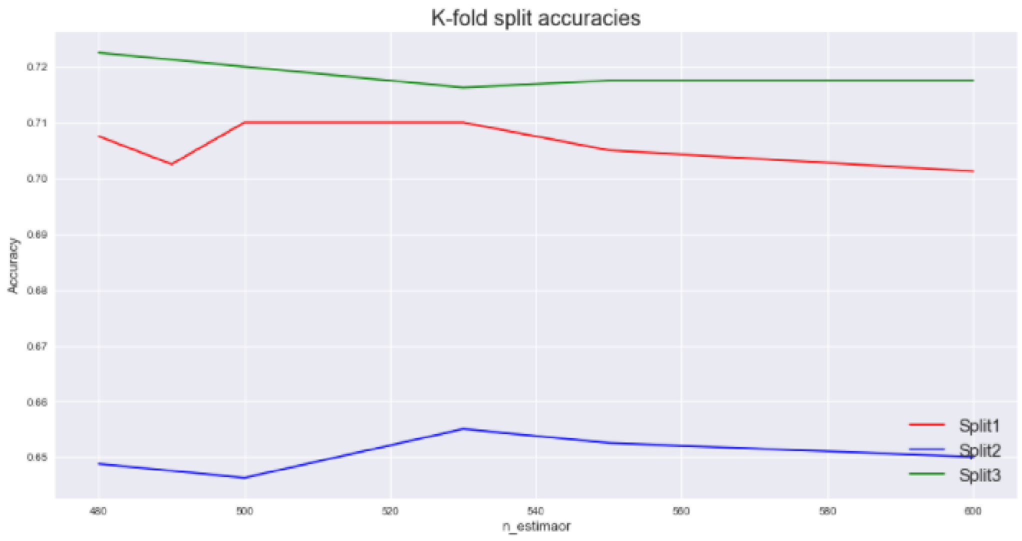


I then moved onto parameter tuning of the alpha parameter. Alpha is a parameter for regularization penalty. The lower the alpha, the larger the weights which can give to complicated decision boundaries. This means that the boundaries can be complex curvatures to even circular shapes. The higher the alpha, it gives smaller weights resulting in decision boundaries that have less curvatures. Low alphas are good or fixing high variance, meaning the model is overfitting, and high alphas are good for fixing high bias or underfitting. From the figures we are able to see that the higher the alpha the less accurate our model was. The sweet spot for the alpha was at 1.0. The average and std deviation seemed to follow the same trend. Where after alpha of 10 the accuracy basically went down to 0% and all three k-fold cross validation agreed with it.

Question 4) Random Forest Classifier (RFC)

MODEL	TRAINING ACCURACY	LEADERBOARD ACCURACY	LEADERBOARD AUROC
BASELINE RFC (N_GRAMS = 1)	1.0	0.798	0.879
BASELINE RFC (N_GRAMS = 1,2)	1.0	0.788	0.883
BASELINE RFC (N_GRAMS = 2,2)	.99	0.64	0.738
RFC {'CRITERION': 'ENTROPY', 'MAX_FEATURES': 12152, 'N_ESTIMATORS': 530}	.0.693	0.787	0.88
RFC (CRITERION= 'GINI', MAX_DEPTH= 12152, N_ESTIMATORS= 800)	1.0	0.795	0.88





The model I wanted to explore was the random forest model. This model is basically a bunch of decision tree classifiers getting a say in the answer. It is called an ensemble method. Each individual tree in the random forest makes a prediction and the class that has the most votes becomes the model's prediction. Random forest seeks to fit the problem of overfitting that decision trees have a tendency of doing. Random forest fixes this by feature randomizing when it builds each of the individual trees and tries to make them uncorrelated. This helps the overfitting question because a "committee" of predictions is more accurate than an individual tree. For this case, the hyperparameters that I tuned were the criterion, max features, and the n_estimators. Max features refers to how many features are used to create a tree. In our case, features refer to the amount of words, the parameters that I tried were square root of the length of features, log of length of features, and the actual length of feature. Turned out our best accuracy was achieved by using the length of features. The n_estimators parameters refer to the amount of decision trees were used to create the "committee". There was a sweet spot of how many estimators were good for the model. It looks like from the graph and from our testing that the sweet spot was around 530 estimators. I tried a lot of different estimators which aren't shown in the figure above. The way I went about it was give it five different estimators to try and change up those five numbers based on what my grid search cv game back as best parameters. So if I gave it [10,20,30,40,50] and the gridsearch cv came back as 50 to be the best, the next set I'd try is [45,50,60,70,80] and so on. RFC {'criterion': 'entropy', 'max_features': 12152, 'n_estimators':

530} seemed to be our best model because it did not overfit the training data. Though this wasn't the highest accuracy achieved in the leaderboard it didn't over fit.

Question 5 and 6) Summary and discussion

MODEL	TRAINING ACCURACY	LEADERBOARD ACCURACY	LEADERBOARD AUROC
BASELINE LOGREG (N_GRAMS = 1)	0.94	0.833	0.912
LOGREG (ITER=10, C= 3.1622)	0.991	0.833	0.919
MLP (HIDDEN_LAYER = (50,), ALPHA=1.0)	1	0.832	0.92
LOGREG (MAX_ITER = 10)	0.943	0.83	0.912
BASELINE LOGREG (N_GRAMS = 1,2)	0.972	0.828	0.91169
MLP (HIDDEN_LAYER = (50,))	1	0.828	0.911
BASELINE MLP (N_GRAMS = 1,2)	1	0.825	0.911
BASELINE RFC (N_GRAMS = 1)	**1**	**0.798**	**0.879**
RFC (CRITERION= 'GINI', MAX_DEPTH= 12152, N_ESTIMATORS= 800)	1	0.795	0.88
BASELINE MLP (N_GRAMS = 1)	**0.999**	**0.793**	**0.878**
BASELINE RFC (N_GRAMS = 1,2)	1	0.788	0.883
RFC {'criterion': 'entropy', 'max_features': 12152, 'n_estimators': 530}	.0.693	0.787	0.88
BASELINE LOGREG (N_GRAMS = 2,2)	0.985	0.657	0.742
BASELINE MLP (N_GRAMS = 2,2)	0.99	0.653	0.746
BASELINE RFC (N_GRAMS = 2,2)	0.99	0.64	0.738

The table shows all of the models that were tried on the training data set and the leaderboard testing set sorted descending by the leaderboard accuracy. As we can see the baseline logistic regression with unigrams actually achieved the highest accuracy score on the leaderboard. Though it didn't have the highest AUROC. From the models if we were to deploy I would say the Logistic regression with max iter of 10, and C of 3.1622 would be the model I would go with. Not only did it achieve a pretty high leaderboard accuracy score but it also had a higher AUROC. I would choose the logistic regression model because it is easier to explain and doesn't have a black box so we can clearly see what the coefficients are how much variance we have, and etc. The logistic regression is also very flexible, and is less prone to over fitting. All of the MLP models that we tried seemed to overfit the training data because they had really high accuracies in the training set but lower on the testing set. The MLP is also harder to decipher on what the model is actually doing. Surprisingly random forest did not get into the top 5 of the models we tried. This is interesting as random forest should've been able to circumvent overfitting to the training set and learn pretty accurately given the amount of "data" that we had. This tells me that, maybe next time I may need to vectorize the words differently as that can be a reason why the accuracy wasn't able to achieve high.