Analyzing the Behavior and context of an email to classify a Sender using the Enron E-mail dataset

GROUP 71 TEAM MEMBERS:

Name	Student ID
Sophia Politylo	N10489045
Joshua McAllister	N10712682
Bryce Hart	N10446575
Wei-Yuan Chuang	

Contents

Introduction:

The objective of the proposed project is to successfully train an effective machine learning algorithm that can correctly identify the sender of an email, based on the language and meta data of the email. The motivation behind creating an email sender identifier is to help reduce malicious and fraudulent emails. Allowing a computer to successfully pick up when a sender is acting abnormally, forcing any harmful emails into quarantine or spam for further inspection. Doing this would help stop the spread of viruses like worms and fraudulent messages sent by hackers and impersonators, minimizing harm to the recipient and the original sender.

In an attempt to analyse the varying behaviour of email senders, three varying machine learning models have been created that attempt to build upon the existing knowledge gained from previous work within the field of email classification. The models have been chosen in an attempt to cover the three main types of machine learning models. With three non-deep learning methods, k-nearest neighbours, random forest and logistic regression being used. Along with these 2 deep learning methods, long short-term memory and a feed forward neural network have been created. All 5 models utilise the given email set in a slightly different way with the same emails having the body of the email being processed in slightly different ways. Doing this, the best model and data format for a user base email classification system should be able to be found, based on the weighted f1 scores and accuracies of the given models.

A wide variety of model types were important for this task, given how novel user classification based on email data is within the field of machine learning. Given that they are mostly created to accurately predict spam and malicious emails, which is a much easier task than the one being attempted with these models. Given that the number of classes within a spam and malicious email classier is around 5 depending on the class division or 2 at the minimum. While the model being created using the Enron dataset has a maximum of 150 different classes representing each individual user. Given the nature of emails, this will be a fairly hard task to differentiate between most users, given there are common writing styles depending on the email's subject. Meaning that any given user could switch up their writing stye with every email depending on the recipient. This problem should hopefully be minimised given that the email data set has been pulled from a business, meaning that all emails are official business emails, which should mean that the users will keep to a consistent business tone. So that the models can learn the difference between all the user's formal writing style, without having to be bogged down with making multiple predictors for each user's varying writing style. Saying that, however, it is unlikely that everyone will have a 100% consistent writing style.

Differentiating against the users is the main goal of the created models due to it being a novel concept compared to the other email classification methods, that if successful will help provide a basis in creating models that can protect the user from compromised email accounts through having an understanding of the user's normal email habits and writing style. Splitting the data based on the sending users was also the easiest way, given that there is no spam folder within the dataset, making it hard to split emails based on spam and wanted emails, without a manual read through. Which is out of the time frame of this project.

Previous Work:

A large amount of work has already been done regarding email classification with most of it focusing on email type classification. Many of these models are in everyday use with most of the major email providers like Microsoft and google providing automatic email classifications, based off the sender and body of the email. With Google being able to sort emails into promotions, social and primary categories without the user's input. While Microsoft's outlook produces a similar method, with emails just being placed as primary (focused) or secondary. When it comes to academic work on email classification, a wide variety of reports and models can be dug up, with many trying new methods to classify an email's type and importance.

One notable model dealing with sender classification is a model created by P Oscar Boykin and Vwani Roychowdhury [1]. Where they decided to implement a system of spam classification via turning all users into nodes, with a connection being created between nodes when the users are common correspondences [1]. With this method the pair were able to classify spam emails based on the common connections between the receiving user and the sending user, through finding the common connections between the users [1]. With this method being able to classify 53% of the email set with 100% accuracy, with all other emails being undefined [1]. Overall, this model is quite impressive, with a unique idea. However, this model is clearly limited in its overall usefulness given it was only able to predict correctly around half of the given data set with binary categories.

A more common approach in email classification explored by Twiwo Ayodele, Rinat Khusainov and David Ndzi, is based off word classification in the email [2]. Within this method, the group created an algorithm that classified emails based on the occurrence of each word within an email and would then return an N number of words, with the words being sorted via occurrence [2]. Meaning that for $N = 2$, the top two words within the email would be returned. To then classify the emails into their given activity group [2]. The training set would then be fed into the machine learning model to train the frequency and occurrence of a word for each activity [2]. With a higher weighting for an activity being created when a word was frequently encountered with all emails in an activity set [2]. Meaning that to classify unseen data, the model would utilize the word weighting created off the training set to predict the most likely category a new email would fall into [2]. This approach produced good results with their model producing an accuracy of 98% [2]. Which outperformed a common email classification software at the time LibTextCat by 90% [2].

Steve Martin, Anil Sewani, Blaine Nelson, Karl Chen and Anthony D. Joseph, produced a model very similar to the proposed models with the group attempting to classify the individual user's within the Enron data set [3]. To do this, the team proposed two non-deep learning methods, based on a SVM classification and Naïve Bayes Classification models [3]. With the emails first being preprocessed into a set of classifiers for the two models, through pulling out certain aspect of the email, like presence of html, number of files attached, number of words in the body, number of recipients, average word length, and ratio of attachments to words. Most of the classifiers selected for this task were chosen because this model would ideally be used for worm classification [3]. To evaluate the created models, the team decided to use an accuracy metric, with both models producing a 95% [3].

The last previous model that will be explored is the one created by Ahmad Al Sallab and Mohesen Rashwan, how unlike the other models used a DCCN on the body of the email to reclassify emails into their given folder's using the top 20 largest users in the database [4]. To feed the data into their DCCN they pre-processed the Enron emails via their body content with all the words being transformed into a bin of words with each email having its words stored in the bin by how often each word occurs [4]. The used DCCN model was a feed forward deep neural network, with it scoring an average accuracy of 87% in re-sorting the emails into their given folder [4]. Which hen compared to an SVM and a k-NN is around 15% more accurate over non-deep learning models [4].

Overall, from the previous work created around email classification, the most telling aspect of the email is the content within the email. With models that are based solely on the meta data being inefficient in email classification. This however does not mean that meta data should total be ignored as there is still a place with these models for meta data which the 2nd last model explored successfully pairing the emails meta data and body context to create an effective worm detector model. This does however highlight the limited importance of email senders and file attachments of the email. When looking at the types of machine learning methods used, it is quite varied with deep learning and non-deep learning methods both being used in the classification of emails with both being quite effective when the provided data for classification is formatted correctly for the required task.

Selected approaches:

The Selected approaches for our proposed machine learning models are:

- **k nearest neighbours**
  Utilizing a set of pre-processed classifiers constructed off the Enron data set with the data, with all classifiers for this data set being transformed into a number metric
- **random forest**
  Utilizing a set of pre-processed classifiers constructed off the Enron data set with the data, with all classifiers for this data set being transformed into a number metric
- **logistic regression**
  utilizing a bag of words constructed using the body of the email
- **Long short-term memory**
  Utilizing GloVe for it's embedding method
- **feed forward neural network**
  utilizing a bag of words constructed using the body of the email

These models have been selected for the following task given that they provide a few different baselines to compare the accuracy of user classification with this data set. With data processing and models that have been proven effective within the preexisting work forming a baseline metric for the given dataset, to allow a comparison between the more novel untested methods. A wide variety of data pre-processing was used for all the models as this again provide a great variety of different approaches that would allow for the analysis of the best way to feed email data into a network.

Data:

The Enron Email dataset will be used to train the models, with the users' 'sent' inboxes being used to form the data for the training, validation, and testing sets. These two inboxes were selected over all the others as it provides an already sanitized sample of data, that is already pre-split by the sender id, making data construction easier. Another important reason that the sent inbox was chosen is due to the uncertainty of the senders in the incoming inbox. There is no guarantee of who the sender is, as it could be from someone outside the Enron data set; be it a personal account, customer or from a mailing list. This means that adding the incoming inbox complicates the model without offering the guarantee that each sender has a large enough sample to help form a meaningful model. Since subtracting all the non-Enron emails from the incoming inbox will just create a mixed up sent inbox, there is no reason to choose the incoming inbox, when the sent inbox exists and is already sorted.

For the sent emails, we further shortened the datasets by removing any emails that had more than 5000 characters in them. These emails were found to often be full of long reply chains, that would confuse the classifier, as many of those emails come from different people. To further reduce the news in the dataset, any email with the phrase "-----" in it was removed. These emails were found to be forwarded emails, or contain information from other emails, that would again confuse the classifier too much.

This approach created 150 unique user IDs, with a total unique email count of 65905. To create the 3 different data set's each user will have 80% of their emails within the training data, 10% of their emails in the validation data and 10% of their emails within the testing test. This split was created due the complexity of the task being undertaken, with a large variety of emails being need for each user to train the model accurately. The data was split using stratification to account for some of the class imbalance, however when attempting this, there were some users with too few emails to be split using stratification. So, any user with less than 10 emails was removed from the dataset so every user could have some emails in the train, validation, and testing sets. After doing this, there were 65837 emails in the full dataset, with 137 unique user IDs.

When it comes the data split for each user id, the overall split is not great with one user comprising of 7% of the data set, with only 28 users, 20% of the user Ids making up 70% of all emails within the data set. With the following percentage split per user being displayed in figure 1. This data split is not great given it means any model can get around 70% accuracy via only attempting to classify 20% of the total user base.

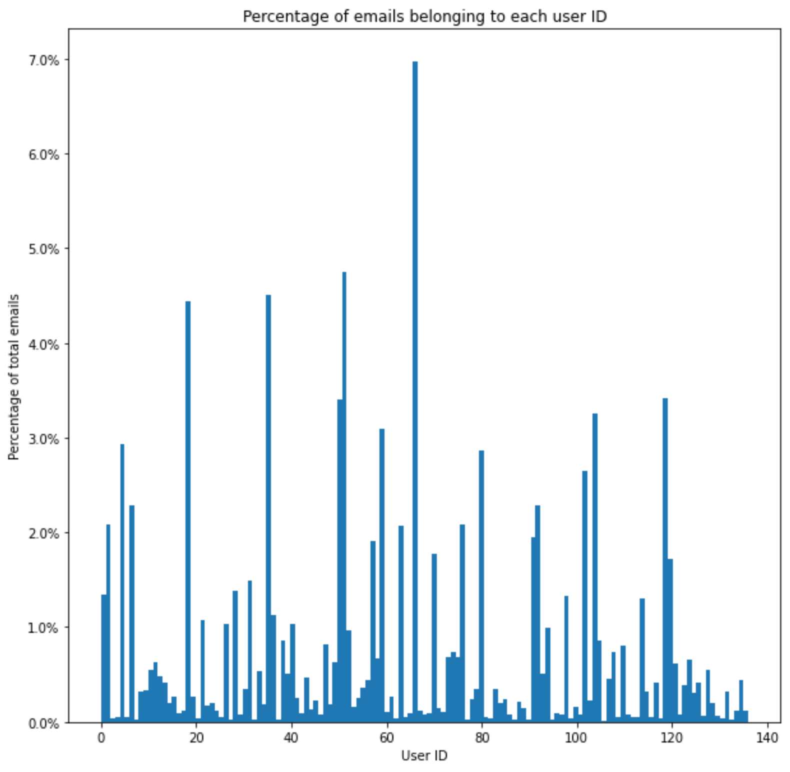


Figure 1 - Class separation within the Eron Dataset when using the user's sent folders

Email Metrics data pre-processing:

The first way that the email data was preprocessed was via manual element classification with the emails being given 9 unique predictors that would form the basis for the non-deep learning models. This data was **praised** through python, loading in each email with the user folder it was found in forming the user id for the email. The loaded in email than had its content read with the program going through each word within the email pulling out the identifiers displayed bellow.

Email length:
Total length of the body of the email. Metric helps the models learn about the user's writing style through finding an average range for length

3 most common words (3 separate categories):
Some words are excluded such as the, a, I, and. This helps the model group users with a small bin of words method, with the model getting a basic over view of a user's preferred words when writing an email

Number of recipients:
This predictor helps the model learn user's based off their mailing habits helping the model learn how many people each user contacts per email

Average Sentence length:
This metric allows the model to get another overview of a user's writing habits allowing it to crudely classify user's based on sentence length.

Email Sign off:
This is the length of the last line of the which hopes to capture each user's unique name or style of from statement. Providing the model with a novel way of classifying users of a fairly unique identifier

Total Amount of punctuation used:
This counts the number of times a user uses '?','.',',','!' In a email allowing the model to identify users off their writing style.

Subject line length:
This is the length of the subject line. Which allows the models to see how a user formats their subject line with weather they use it as a title, ignore it or use it as an email overview

From these identifiers, the following bins were created to show the range and occurrence of each value with the total email data set, seen in figure 2. This shows that some variables have some massive outliers given the total range created for identifiers like number of recipients and last line, with most of that data being centered around the 0 to 100 mark. The range for the top words however is expected given the wide variety of words. However there appears that some common word types have been let through the pre-processing stage given the large peak at around 0 with that bin having a total occurrence of 25000 times in the email data set. Which is far too high for a word that is not used to help construct a sentence.

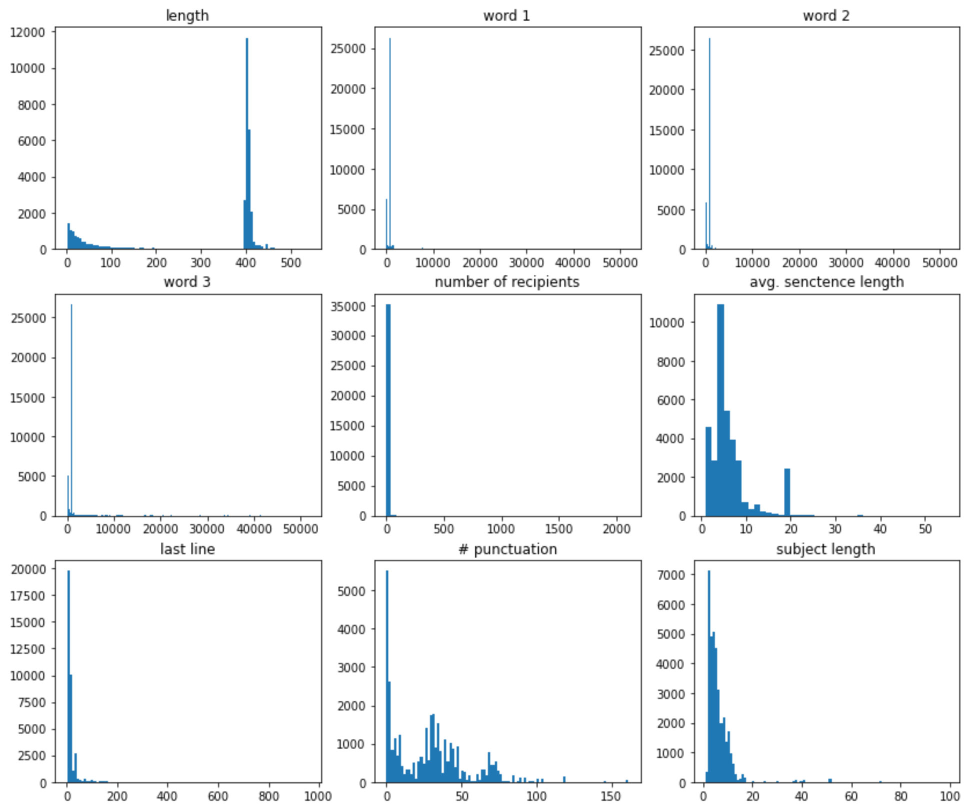


Figure 2 - Bins for the created metrics for the non-deep learning models

Bag of words data pre-processing

The bag of words vocabulary was created off of all the words within the data set, with certain common words that are universally used like “the”, again being ignored, given that they provide no useful data given their frequent occurrence within everyone’s writing. After a vocabulary was created, the emails where then fed through with the vocabulary to form the frequency vector for each email. The bag of word function used within this setup was from the nltk library with it word_tokinizer being used to form the vocabulary list.

N-grams of size 2 were also used with the bag of words data. This is where the word tokenizer finds all of the combinations of 2 words, and assigns them a value in the bag of words. Due to limitations in memory, only N-grams of up to size 2 could be tried.

GloVe Embedding:

In the literature review, we did not see LSTMs or embeddings being used, however, these techniques have proven to be effective methods for classification in other natural language processing (NLP) problems [5].

GloVe embedding is a vectorization embedding those accounts both for a words occurrence and meaning. With GloVe embedding being able to create a relative relationship two words based on their similarity and context [6]. With this embedding model being able to form a relationship around words like steam and ice, via knowing that they are both about water but rarely will appear together tanking about the same water. With one being the gas state and the other being the solid state [6]. This means that this kind of embedding is able to produce an embedding type that is able to place words together that are referring to a same macro concept such as water closer together on the x axis, while also being able to filter them apart from one another when they are referring to opposite positions that that macro object could take [6]. The main purpose of glove embeddings then is in text prediction models with the created vocabulary with the relationship matrix for each word being highly useful [6]. Given that with this a model is able to more accurately predict the following word or subject of the text [5]. As for the case with water, ice and steam, a model using GloVe embeddings would be able to narrow down the context of the text when water is mentioned with it looking for words that commonly occur with water, with the model thus looking at words that have a vectorized value similar to the word water. Now if the next word that comes among is gas, a model using GloVe would then be able to pick out the word steam from its corpus of words, given that the embedding for steam is the gas form of water with its embedding corresponding to that with it having the most similar vector to both water and gas.

Given this this type of embedding should prove helpful when it comes to classification of user’s within the dataset given that via using these embeddings a greater context around a word and it relationship to the other words within the email. This would than mean that a created model would then be able to form predictions around the occurrence of similar and dissimilar words within each user’s email. Which in turn will allow the model to pick up on the user’s writing pattens and habits, based on the occurrence and way words are linked together within their writing style.

Methodology:

Comparison method for models:

The created models will be compared based on two metrics; the accuracy and weighted f1 score. These two values were chosen to be the major comparison values between the different modes due to the data separation. Given that the data set created has a 20% of the user making up 70% of the data, it becomes important to ensure that the models are not disregarding most user IDs. This is where the weighted f1 score comes in, as it produces a score between 0 and 1 that takes into account the precision and recall of a model [7]. This is an important distinction when compared to accuracy, given that an f1 score takes into account the number of false positives, with the following equation being used to calculate f1 [7]:

$$2\frac{p \times r}{p + r}$$

Where:

$$p = \frac{all\, positves\ \cap real\, positives}{all\, postives}$$

$$r = \frac{all\, positves\ \cap real\, positives}{real\, positives}$$

This F1 score is calculated for all of the user IDs, and then a weighted mean is calculated based on the prevalence of each of the user IDs in the true labels. From this equation it can be seen that the weighted f1 score is a useful metric when finding out the validity of a model when there is major class imbalances given that it takes into account all false positives. This means that via using this metric, any bias towards certain classes should be plainly shown given the precision for those values will be dreadful due to all the over abondance of false positives within all positives variables [7].

Accuracy will still also be considered when comparing the modes as it gives a fairly well understood metric that can quickly be read and understood, that can help to quickly rule out a function.

Non-deep learning method k-NN and Random Forest:

These two non-deep learning methods have been chosen as the models for the email metrics data, due to them both presenting varying methods to classify the same data. Even though neither model have been extensively used on the Enron dataset, both models provided very promising solutions to data re-classification that might be able to out preform the more commonly used non-deep learning method of an SVM. This improvement if seen would come from the fact that unlike an SVM both of these methods are not bound by a hard parameter with a hyperplane. This means that these two models have slightly more freedom in figuring out how to classify fringe cases. This is not so much the case with KNN given it still requires some degree of clustering within the data to enable this predictor to work as intended.

The Random Forest method however presents a total unique option in classifying this data given that clusters are not so much needed, with class separation being much more important when creating a model. Given that a random tree is a decision tree the final prediction can be more dynamic and based less of distance and more on the varying relations within the data. This means that a high degree of clustering is not really need for a random tree with the same user being able to have 2 or 3 separate clusters, as long as the user's id has large enough separation between other clusters, so that the created forest doesn't over fit.

The KNN model was compiled using a weighted distance, with the value for number of neighbors being fine tuned with the validation data, with the output accuracy and k1 score being viewable in appendix _. From this, a few different candidates were found for number of neighbors with there hardly being any improvement in the validation or training data's f1 score or accuracy with the best for both in the validation data getting around 0.781 and 78.3% respectively only 0.02 and 0.3% higher than other values for number of neighbors. Two n_neighbours values were found to produce this accuracy with 10 and 30 both giving the same score. The final model however was compiled using number of neighbors equals 10. This was chosen over 30 as 30 would place to much significance on the larger user sets given the 3 times larger pool of points a model with 30 neighbors. The final accuracy found using this model was 99% for the training data and 79.3% on the test set. With the f1 scores for the training data being 0.99 and for the testing set it was 0.792. Overall, these scores are fairly good given the data set with the f1 score and accuracy being fairly consistent meaning that there is not too much overfitting to a single case going on, with the model at least attempting to identify all users. The accuracy of 79.3% is also very surprising when considering the limited and basic metrics that is used via the model and the complexity of classifying user's based on their emails. This would most likely come down to the metrics that classify the user based on their writing pattens as all the metrics regarding the meta data appeared to be fairly useless in separating the data when all metrics were compared in a bin graph.

To help collaborate the f1 and accuracy score for this model, the individual accuracy for each user id was plotted for the testing and training data. With an individual accuracy being found via summing up the total amount of predicted values for each user id being dived by the total amount of real emails with that user id with the following graph's being shown in figure 3. From these graphs it does appear that the KNN model has in fact been trying to predict all classes with almost all classes having a 100% accuracy in the training data with only 2 classes having an accuracy below 60% with the rest of the classes being above 80%. When comparing the testing data, the created class accuracy are not as smooth, with around 50 user id's being over predicted with an over 100% (red coloured bars) accuracy and with 5 classes having an accuracy of over 200%. This is not great but via chopping off all the accuracies at 100% the third graph is created which shows that the accuracies are not so bad for each class with around 86% of the total class accuracies being above 50%.

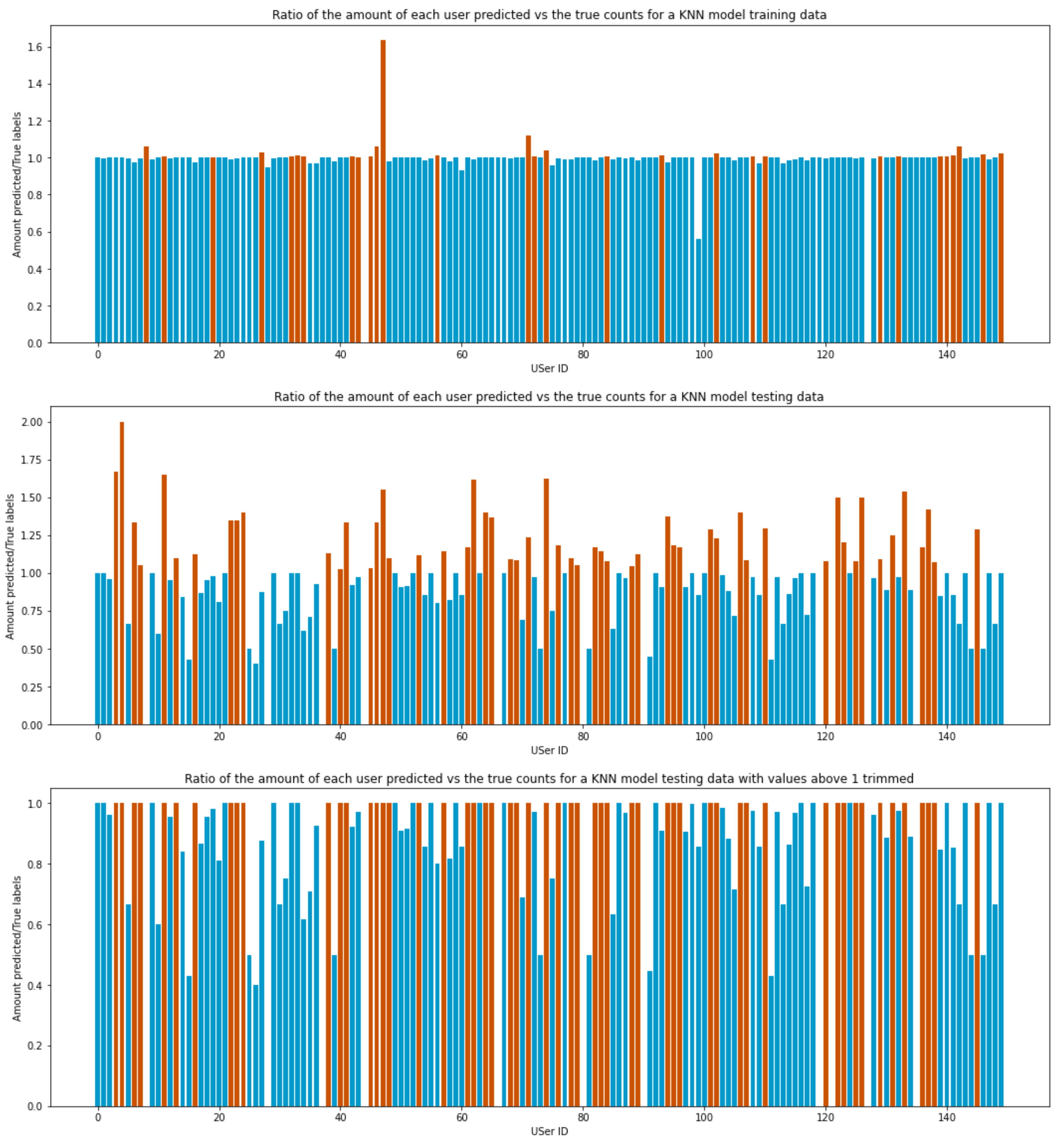


Figure 3 - KNN ratio of predicted and real values for testing and training data for each class

To try and find out where the model was going wrong with the data classification on the testing set, the results for each class accuracy was ploted againest the total precentage of the dataset they make up, showin in figure 4. With the percantage of the data set being shown in green. From this dual analises, the first assumtion of the model over fitting to the larger classes have appered to be wrong, with the 3 largest classes having an accracy below or at 100%. The other larger classes that make up around 1% of the database each are not so good with that being the second most likely class to be over predicted. These over predictions for the 1% of the data set classes however are not out of the ordinary for a model with it being wrongly predicted about 10% of the time at max. The truly bad predicted classes are coming from the smallest data sets that make up less than 1% of the data set. This would mean that the smaller cases must be being lumped together in some cases, given their size of samples are too small to form meaingful clusters, causing the small classes to merge together and be predicted in one of the other classes. Meaning that for an accurate model a class need at least around 43 emails.

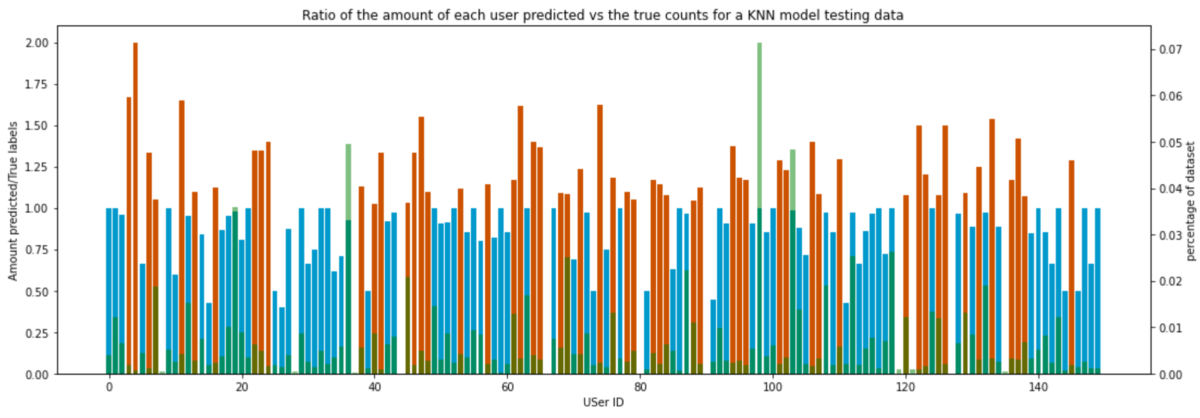


Figure 4 - ratio of predicted vs real labels for testing data classes plotted against the percentage each user makes up of the dataset

When creating the random forest model both the max depth and number of estimates were fine-tuned with a depth in a range of 10 to 60 going up by 10 was tested, while 10,20,30,40,50 and 100 were tested as values for the number of estimators with the output for each test using the validation data being in appendix _. From this, the best values for max depth were found to be 30, while number of estimators were found to be 100, with an accuracy of 80.5% and a f1 score of 0.805. The accuracy and f1 for the training set was 99% and 0.99. This model again had a fairly consistent accuracy with it sitting around 99% and 80% for the training and validation set regardless of the increasing depth and number of estimators. This model however was able to produce some models with a validation accuracy at around 65% for the validation data when the depth was set to 10, with a depth of 20 fixing the accuracy placing it at around 79%. This would appear to show that the most important aspect of the created model is the depth with the number of estimators having very little effect on the overall accuracy once the depth (Just double checking that that depth is supposed to be depth and not num estimators) has surpassed 30.

when running the chosen model on the test set, an accuracy of 82.5% and f1 score of 0.824, were generated. Again, to test each individual class accuracy, a bar graph was plotted shown in figure 5. This graph has been set out the same as the last one with the percentage of the data set each user making up being overlayed on the second graph. Again, a very similar trend emerges as seen in the KNN model with the training data being almost perfect, while a very similar trend is seen within the testing data with now 55 classes getting accuracy above 100%, 5 more than KNN, with 80% of the classes getting above 50% again. When it comes to individual class accuracy and total amount of the data set each class makes up, the same trend in KNN was seen with larger classes again scoring the closet to 100% accuracy, while the smallest classes are over(Is it supposed to be under predicted?) predicted.

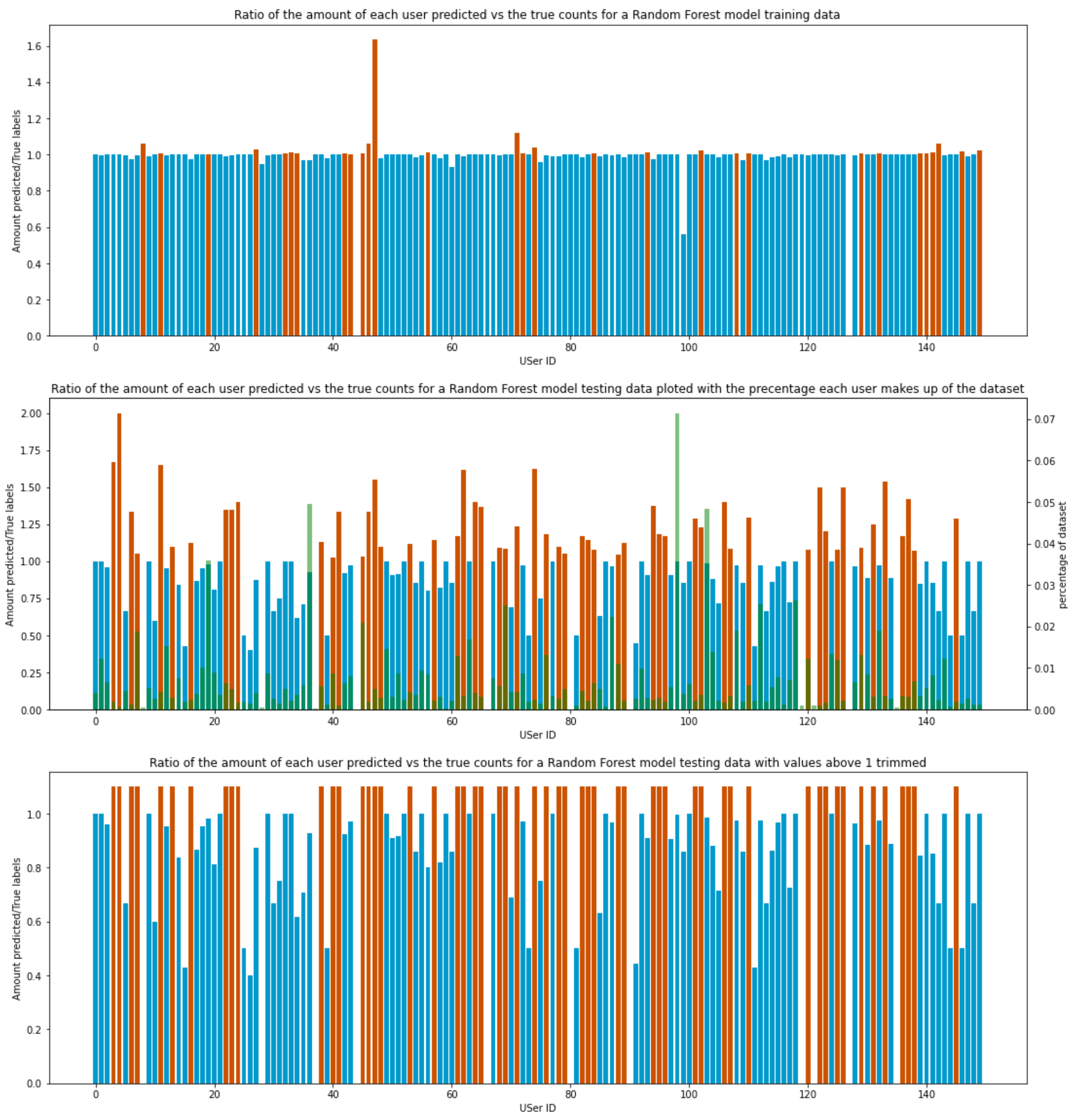


Figure 5 - ratio of predicted and real values for all classes in the testing and training dataset

Overall, from these two models a good starting point have been found for what can be achieved on the Enron email dataset, with KNN and a random forest not producing greatly different results from each other with both getting within 3% of each other on the testing data.

Logistic regression:

The logistic regression will be performed on the bag of words pre-processed data model. This model has been selected as one of the candidates given its wide use within the machine learning field and the already proven track record when it comes to email classification, with it able to score 94% on email spam classification [8]. This model is also perfect to compare against the K-NN and random tree given it utilizes the email data in a totally different way. With all variables within the input data being feed into a log equation with weighing being assigned to each variable, to form the prediction a probability scale of 0 to 1 for a given email being sent by a certain user [9]. This means that with the bag of words model, this model should be able to learn effectively with it creating a probability log scale for each user within the data set based off how often and what words each user is most likely to use when compared to one another.

When training this model on the bag of words, the model produced an f1 score of 0.825. This is a good sign of the validity of this model given it has scored higher than the previously trained models so far. Further showing that when it comes to user classification the frequency and words used is incredibly important in correct classification.

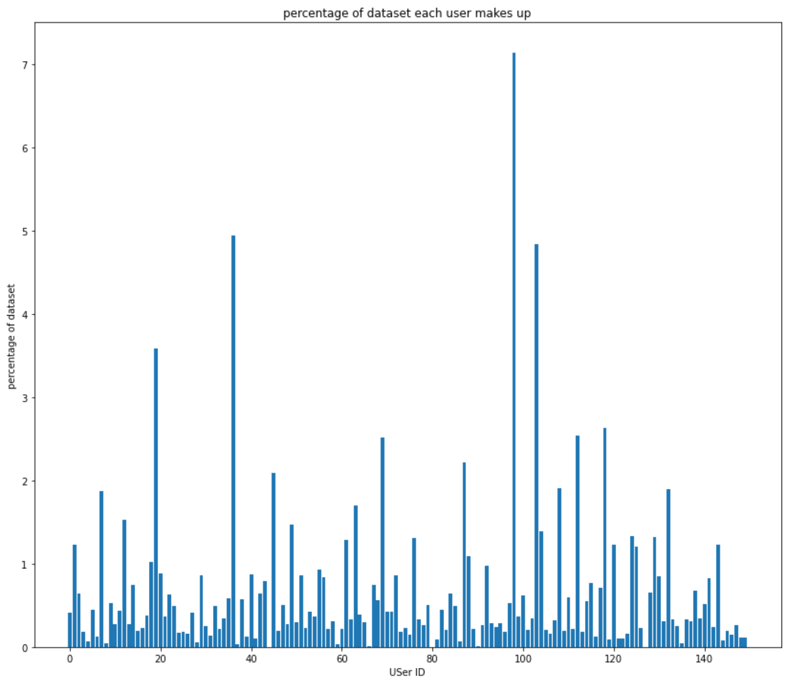
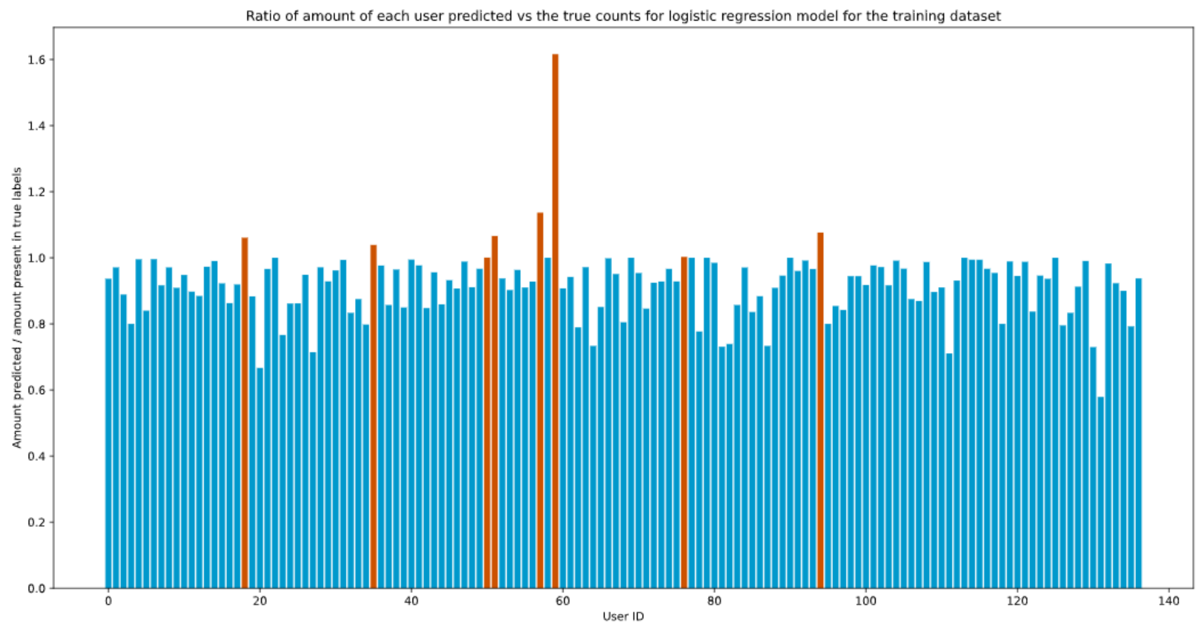


Figure 6 - class separation within the Enron dataset

We can now look at how this model was able to predict each of the classes. The plots below in figure 7 show the predictive performance of the logistic regression model for the training set and the testing set. The first plot, as has been seen before, shows the ratio of the predicted counts and the true counts for each of the user IDs for the training set. In this plot, you can see that the model was able to adequately predict most classes quite well in the training dataset. The predictive performance in the training set wasn't as good in the training dataset as the previous models (like KNN and random forest). However this might partially be explained by the fact that the model failed to converge. The training for the model was halted at 500 iterations because it was taking too much time to learn, however even with that stopped training, it performs almost as well as the other models on the training set. The only major exception seems to be ID 59, that is predicted about 1.6 times as often as it actually appears. User ID 131 is also severely underpredicted, being predicted only about 0.6 times as often as it appears in the true values.

The second plot in 7, is the same as the first plot, but for the testing set. You can see that overall the model performed much worse in the testing set. There were 14 IDs that were not even predicted once, even though they appeared in the testing set. Also, the number of overpredicted IDs has increased from the number in the training set, probably accounting for the fact that multiple classes were not predicted at all. The most overpredicted ID in the training set, 59, is still the most overpredicted in the testing set. For the testing set, 53 IDs were severely underpredicted (the ratio of predicted to actual was less than 0.75), but only 6 IDs were severely overpredicted (the ratio of predicted to actual was more than 1.25). This model has a large number of IDs that are only slightly overpredicted.



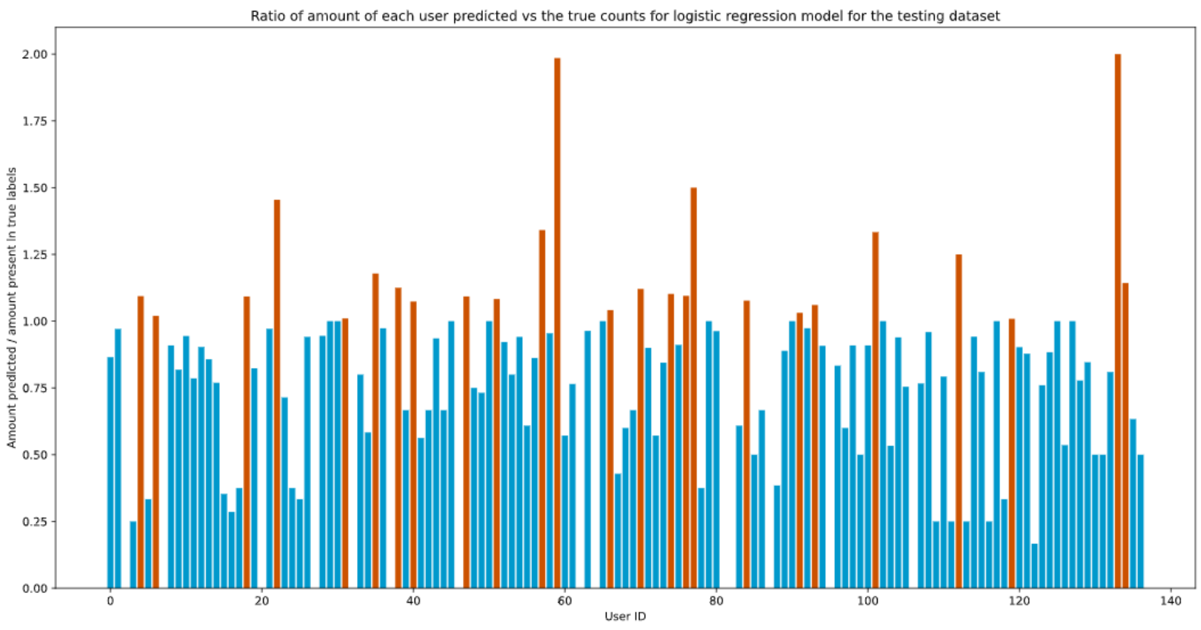


Figure 7 - Predictive performance for each user ID for the logistic regression model.

Long short-term memory Neural Network:

A Long short-term memory neural network using Global vector for word representation (GloVe) embeddings to predict a given user for an email. These two methods were paired together due to the nature of sentence prediction.

Glove is an embedding function that attempts to capture both the number of times a word occurrence along with the context of a given word. Along with this, GloVe can extrapolate the co-occurrence of similar words via using it corpus to form relationship between synonyms for a given word [5]. An example of this is with man and woman, which in the vector spaced used are place close together on the x axis while on opposite ends of y axis [5].Given that they both are referring to people, but both classify a different type of person [5]. Extrapolating this Glove would than use it's pre designed corpus to decided that king and queen have a similar weighting when compared to each other [5].

A LSTM model is a great model to select for word embedding and relationship tasks given that it is able to recall previously learnt data from the context of the sentence to form its predictors [10]. This means that unlike a normal NN this model is able to feedback learnt information to the early neurons allowing it to form base assumptions and predictors based off the context of the sentence and words used [10]. This means that a LSTM pair with GloVe as the word imbedding method, will allow this model to learn the context of the given world to help form it's predictors for the user that wrote the email [5].

For the LSTM model, the general layout of the model was an embedding layer, a SpatialDropout1D, a variable number of LSTM layers of size 300, a variable number of dense layers with a dropout following each dense layer (the dense layers used a relu activation), then finally a dense layer with size given by the number of classes to predict and a softmax actiation. The model used batch sizes that were decided as a hyperparameter to be tuned, with 100 epochs. However, early stopping was used with a patience of 3, so the models never actually made it to the full 100 epochs. Sparse categorical crossentropy was used with the adam optimizer. For the LSTM layers, the parameters that allowed for effective GPU training were used. These were an activation of "tanh", a sigmoid recurrent activation, a recurrent dropout of 0, no unroll, and using bias.

A number of the hyperparameters could be tuned, these were:

1. The size of the initial spatial dropout, this could be any float from 0.1 to 0.8.
2. The number of bidirectional LSTM layers, each layer had 300 units. But the model had anywhere between 1 and 3 LSTM layers (integers).
3. The dropout of the LSTM layer. A float between 0.1 and 0.8.
4. The number of dense layer blocks (dense layers and dropouts). Integer between 1 and 4.
5. The size of the initial dense layer. Integer between 16 and 2048.
6. The size of the dropout between each dense layer. Float between 0.1 and 0.8.
7. The multiplier of the subsequent dense layers. So the initial dense layer was decided by a hyperparameter, then the subsequent layers were some number multiplied by the previous layer. So if the multiplier was 0.5 and the first dense layer had a size of 1000, then the second layer would be 500, then the third would be 250, and so on. This was a float between 0.5 and 1.25.
8. Lastly was the batch size. This was in integer between 256 and 1024.

For each iteration of the hyperparameter tuning, the model was fit with the given hyperparameters, and the weighted F1 score on the validation set was returned. The optuna algorithm would then try and select the hyperparameters than maximized the F1 score. The hyperparameter tuning was run for 37 iterations until it was decided to stop the tuning because the F1 scores of each model were usually around 0.77, which was lower than the rest of the models and it didn’t seem like it would ever get much higher.

The best hyperparameters were: Batch size of 339. Dense layer multiplier of 0.72. Dense layer size of 1717. Dense layer and LSTM layer dropout of 0.23. 1 dense layer, 2 LSTM layers, a spatial dropout of 0.17.

Using LSTM and Glove for embedding, this model was able to get a weighted f1 score of 0.78. Which is around about the same as all the other models tested so far. This model however did not provide the exact same accuracy however. This could be caused due to the fact the GloVe embedding are providing too much information to the LSTM model making it hard to decide what the important information is. Since the embeddings also contain data on how related words are to each, this metric could be making it harder for the model to fit email to a specific user, given the embedding is more complex than needed for differentiating users with GloVe embedding working better as a future text prediction than predicting the text writer.

We can also look at the predictions of specific classes for the LSTM model in figure 8. This model followed the same trend as the previous models with very good training set performance but much worse performance on the testing set. For this model, fewer values were predicted to be 0 than the logistic regression, at only 8 IDs that were never predicted. However, this model has more values that were overpredicted at 42. The most overpredicted ID in the training set was underpredicted in the testing set, however most of the other IDs that were overpredicted in the training set were also overpredicted in the testing set. For the testing set, 46 IDs were severely underpredicted (the ratio of predicted to actual was less than 0.75), which is less than the logistic regression model, but 11 IDs were severely overpredicted (the ratio of predicted to actual was more than 1.25), which is more than the logistic regression model.

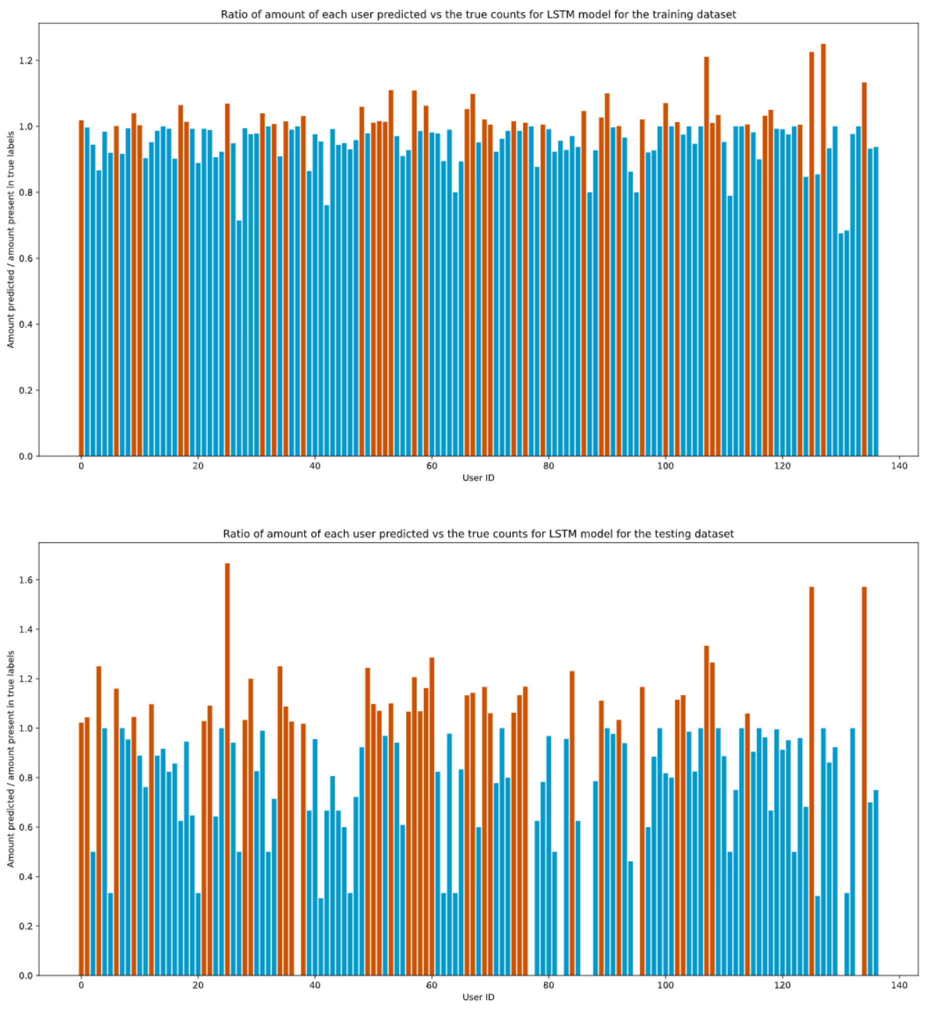


Figure 8 - LSTM predictive performance for each user IDs.

Feed forward Neural network:

A feed forward neural network has been created as the last model to compare results from with the data being fed into it being the bag of words data model, and the N-gram bag of words of size 2. Bags of words were used for this model since a feed forward network is unable to base predictors off what have come previously in the word. Meaning that the GloVe embedding would be no use since this model is unable to predict the sentence structure off previous known word. Making bag of words the better model to use for this network type since it allows for the model to learn the sentences structure without having to worry about the words context. Using a bag of words method also provides a deep learning method comparison to the non-deep learning bag of words model created with the linier regression model.

A feed forward network unlike a bidirectional LSTM, travels in only one direction with the parsed data only being fed forwards to neurons in the next layer with no feedback to lower layer's to help provide context to the whole data [11]. A feed forward network however should be more than sufficient for this task with it being a commonly used deep learning method when forming email predictors, given their reliability and simple implementation [11]. Given their reliability for the given task of email classification this model helps to form a good base line for the other models allowing for a proven method to be compared against the less tested models.

To train the feed forward networks, optuna was used to find the hyperparameters that maximise the weighted F1-score on the validation set. The network structure was as follows:

1. An initial dense layer for the input, with the input dimensions being the number of different words in the bag of words. This used a relu activation. Following this was a dropout layer and a batch normalization.
2. After this were a variable number of units that were described above (dense layer with relu activation followed by dropout and then a batch normalization). For these layers, the input dimension would just be the size of the layer before it.
3. After these dense layers was a final dense layer with size given by the number of unique user IDs (137), this one had a softmax activation.

The model used sparse categorical crossentropy and the adam optimizer. The number of epochs was 300, but it also used early stopping with a patience of 3, so that number of epochs was never reached. The batch size was determined by a hyperparameter.

The hyperparameters that could be tuned by optuna were:

1. The size of the initial dense layer. This one varied for the bag of words and the N-gram bag of words. For the normal bag of words, this was an integer between 16 and 1500. For the N-gram bag of words, using a size that big was too large to fit in memory. So the N-gram bag of words could have integer values between 16 and 200.
2. The size of the initial dropout. Float between 0.1 and 0.8.
3. The number of dense layers after the initial dense layer. Integer between 1 and 4.
4. The size of those dense layers (that were not the original dense layer). Integer between 16 and 1500 (for both the N-gram model and the normal bag of words model).
5. The size of the dropout (for layers that weren't the initial one). Float between 0.1 and 0.8.
6. The batch size. For the N-gram one, this was between 50 and 100, for the normal bag of words model, this was between 256 and 1024. Note that it was so small in the N-gram model largely because of VRAM limitations.

Each iteration of training worked in the same way is for the LSTM. The model was fit with the given hyperparameters, and then the weighted F1 score was returned for the models performance on the validation set. This was then used to inform the choice of hyperparameters to test out for following generations. The model parameters with the best F1 score were recorded. This model was trained for 100 iterations.

Model performance on bag of words data (no N-grams)

The best model for the bag of words got a weighted F1 score of 0.863 and it had the parameters of: 1003 batch size, 976 size for dense layers that were not the initial layer, 0.175 dropout, 0.713 dropout for the first dropout, 2 layers of dense layers (not including first layer), and 1499 size for first dense layer.

With it having by far the highest f1 score out of all the models with the increase being around 0.03 from the second-best model. With the rest of the models having a variation of around 0.2 from each other at max.

We can now look at the predictive performance on each of the IDs in figure 9. This model performed very well on the training set, but on the testing set, there are some very clear overpredictions happening. For user 64, they are predicted over 6 times as many times as they appear in the testing set. While it seems very extreme on that plot, in reality, user 64 only appear 3 times in the testing set and was predicted 13 times, which isn't too many extra predictions. This may be why even though these plots seem like some of the worst that we have seen, the model still had a better F1 than the other models. Though there were still 4 IDs that were predicted over twice as many times as they appears. Each of these had 22, 6, 3, and 4 instances of appearing respectively, so it isn't too impactful. Looking at the third plot in this figure, it clips the ratios at 1, so we can get a better view of the values that were underpredicted. This model failed to predict any instance of a user 10 times, which is roughly comparable with all the other models.

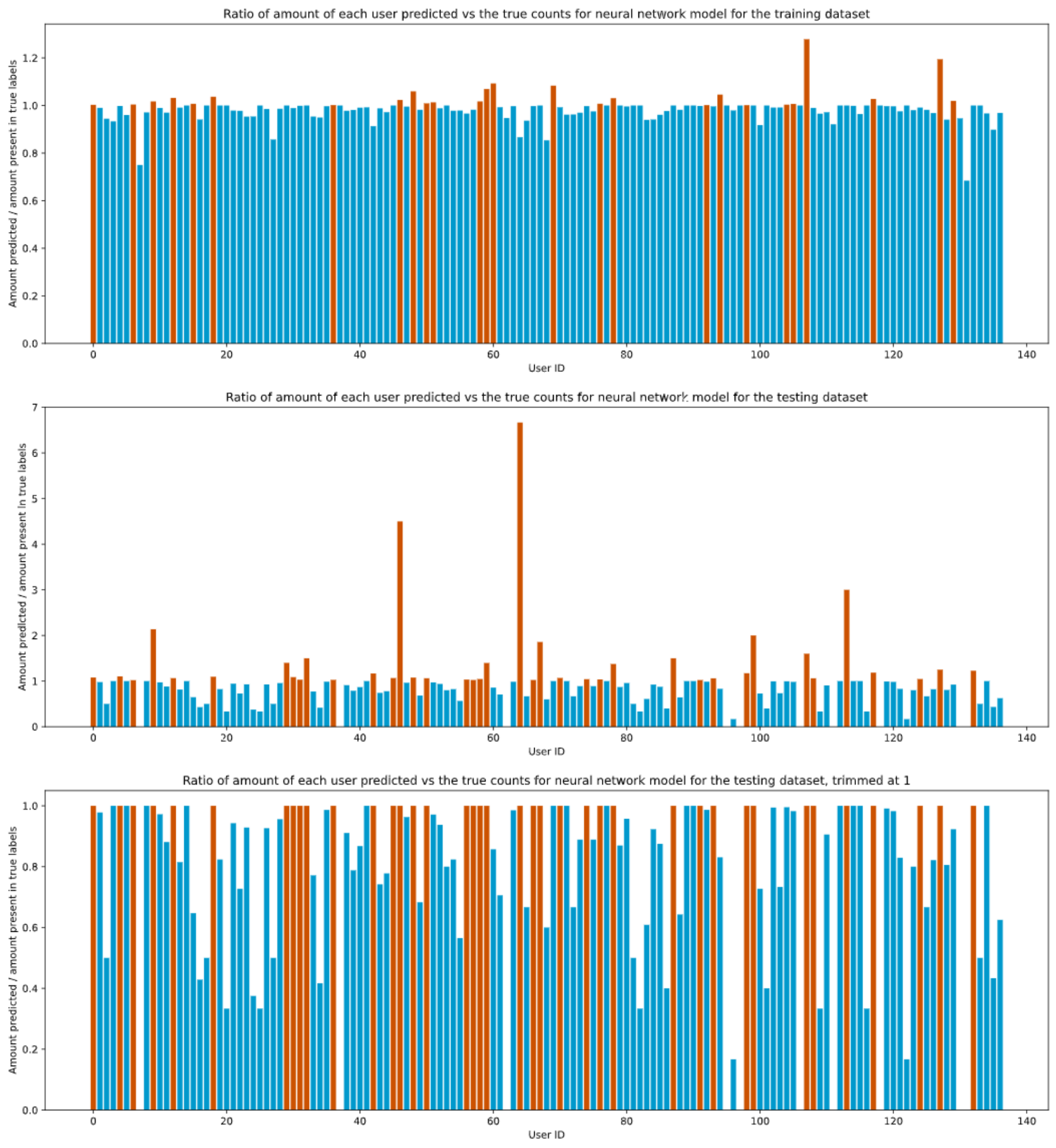


Figure 9- Feed Forward neural network class prediction

Model performance on N-gram bag of words

The best N-gram model got an accuracy of 0.8729, with a batch size of 70, dense size (not first layer) of 818, dropout of 0.677, initial dropout of 0.582, 1 dense layer (that wasn't initial), and an initial dense layer size of 168. This model got the best performance of all the models, beating out the second best model, which was also a neural network, by about 0.01. Even with the constraint of needing a smaller model size than the other neural network, this model still performed the best. It seems that N-grams are very valuable for predicting the email sender.

Finally, we can look at the predictive performance for the best model on each of the user IDs in figure 10. This model performed alright on the training set for the most part, but it has some very large outliers that are at 2 or greater than 2. Users 87, 130, and 135 have all been greatly overpredicted on the testing set. On the training set, they have also been greatly overpredicted, with user 130 being predicted over 20 times as much as it actually appears in the testing set. These 3 IDs are the 3 most overpredicted in both the training and testing sets. Though in the testing set, these IDs only occur 2, 4, and 30 times respectively, which isn't too much out of a set of 6580.

When zooming in to the third plot, the plot that is trimmed at 1, we can see that this model predicts the values in a similar proportion to how often hey appear most of the time. There are 10 levels which are never predicted, but outside of those, the model performs quite well.

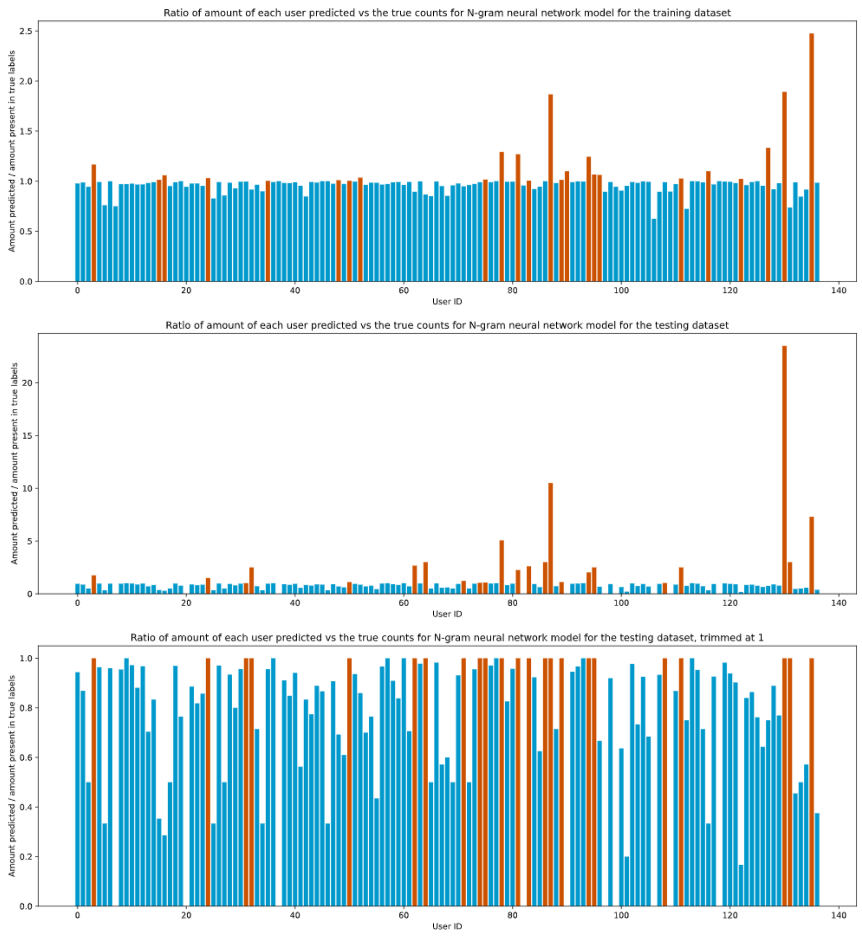


Figure 10 -feedforward network class prediction using n-gram bag of words model

Evaluation:

From these models a few key finding have been found when it comes to predicting the sender of an email. The first of these is regarding the type of data needed to successfully build a model that can classify the sender. With the majority of the predictor information coming from the body of the email. Meaning that the created model for sender classification is heavily based on text prediction, with sentence structures and word count being incredibly important for the models, while the words meaning is not very important, with the overall predictor mainly being built around predicting what words a user is likely to. This is backed up with all the models created given that the 3 models utilizing only the body scored just as well or better than the model that utilized the body and meta data of the email.

Another important observation made from the models is in regards to the size of the sample data needed to enable the model to form meaningful predictors for each user. Given that within the data set the best predictors were for the classes that had a large pool of points to pull from, with the model not over predicting them over other emails to achieve this result. With instead the classes that were being over predicted belonged to the smaller classes, with a majority of the smaller class cases being either over predicted or under predicted. With the minimum size needed for meaningful predictors to be made with these models being around 37 unique emails.

The last major observation from these models is that deep learning and non-deep learning methods do not really provide a remarkable improvement form each other with all models when fined tuned to their optimal values being only 0.05 off when it comes to their f1 score. With the data processing method being used being much more important than the chosen model, given that GloVe embedding preformed dreadfully when compared to the bag of words and self-selected metric data for the emails.

Overall though from these models the best data preparation and machine learning method for these models were a bag of words using an N-gram of 2 data embedding method along with the fine-tuned feed forward network providing the best results producing an improvement of 0.08 in f1 score when compared to the worse model (LSTM) and an improvement of 0.01 when compared to the second best model which just used the bag of words without N-grams.

When comparing these accuracies to other approaches, there is not much deviation within the finding of this report and other existing literature. With most models that either utilizes a bag of words or a compiled array of metrics outperforming any of the more novel ideas so far. With it appearing that when it comes to classifying emails the body of what is written is always the most important aspect. There are however a few fridge cases within the literature where meta data can be useful, but these approaches are used for the classification of worms and other computer viruses, with human classification being much more based around text comprehension.

Conclusion:

From using the Enron data set to train 5 varying models on a clear best model and data processing method has been found. With A Feed Forward Neural Network providing the best results with an f1 score of 0.87 which when considering the classification complexity and data separation is remarkably good. With it only being around 0.10 worse than the models created on the Enron dataset to classify spam and non-spam emails. Apart from this another insight gained is on how to best choose the most important aspect of an email for a given classification task. With meta data having very little diversity when it comes to normal user's, with the user's text in the body providing much more information that can be easily separated via the machine learning algorithms. There is however a limit on the usefulness of reading the email's text for sender classification however with data embedding like GloVe providing too much useless information for a model to form as meaningful predictions given that this embedding focuses too much on the macro view of the dataset instead of the micro overview of the context of each induvial user and their writing style.

With this in mind, there are still improvements to these models that can be made with the next possible study area looking into other types of text embeddings that look more towards the micro context of word use, that would enable a model to learn the meaning behind each word for each individual user. Looking into increasing the n-grams used would potentially improve the accuracy of the models, however requiring more memory. With this model possibly being able to produce better predictions given it looks at the smaller context of the word instead of it overall relationship in the corpus.

Given the current accuracy of the best model created, it is not recommended that this model is put into real life use given the accuracy is not quite there for everyday use in an email server with the overall accuracy being too small for reliable predictions on a large scale. While also being too sensitive to the amount of data needed, given that the model needs a minimum number of emails from each user to allow it to correctly predict a user around 80% of the time, with a data set too small causing over predictions and thus letting malicious emails out or the accuracy becomes too low offing most emails into quarantine. This is not taking into the assumption either that the total amount of emails for each person will go up with the number of user's that need to be predicted given that as more users are added the smaller and harder it becomes to separate each user form one other. This means that overall, the created models are great test and theory for basing a future model off, providing a starting position for any future work in email classification based off the sender of the email.

Bibliography

[1] O. Boykin and V. Roychowdhury, "Personal Email Networks: An Effective Anti-Spam Tool," Department of Electrical Engineering, University of California, Los Angeles, 2004.

[2] R. K. D. N. Twiwo Ayodele, "Email Classification and Summarizaion: A Machine Learning Approach," University of Portmouth, Portsmouth.

[3] A. S. B. N. K. C. a. A. D. Steve Martin, "Analuzing Behaviorial Features for Email Classification," University of California, Berkeley.

[4] A. AL SALLAB and M. RASHWAN, "E-MAIL CLASSIFICATION USING DEEP NETWORKS," Journal of Theoretical and Applied Information Technology, 2012.

[5] K. Vaidya, "Sentiment Analysis using LSTM and GloVe Embeddings," towards data science, 20 November 2020. [Online]. Available: https://towardsdatascience.com/sentiment-analysis-using-lstm-and-glove-embeddings-99223a87fe8e. [Accessed 24 May 2021].

[6] J. Pennington, R. Socher and C. Manning, "GloVe: Global Vectors for Word Representation," Computer Science Department, Stanford University, Stanford, CA.

[7] T. Wood, "What is the F-score?," DeepAI, [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/f-score. [Accessed 22 May 2021].

[8] S. Helmini, "Empirical Analysis on Email Classification Using the Enron Dataset," Towards Data Science, 11 November 2018. [Online]. Available: https://towardsdatascience.com/empirical-analysis-on-email-classification-using-the-enron-dataset-19054d558697. [Accessed 24 April 2021].

[9] R. N. Sucky, "Multiclass Classification Using Logistic Regression from Scratch in Python: Step by Step Guide," Towards Data Science, 6 September 2020. [Online]. Available: https://towardsdatascience.com/multiclass-classification-algorithm-from-scratch-with-a-project-in-python-step-by-step-guide-485a83c79992. [Accessed 24 May 2021].

[10] "Understanding LSTM Networks," Colah's Blog, 27 August 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed 23 May 2021].

[11] M. H. SAZLI, "A brief review of feed-forward neural networks," Communications Faculty Of Science University of Ankara, Ankara, 2006.

Appendix:

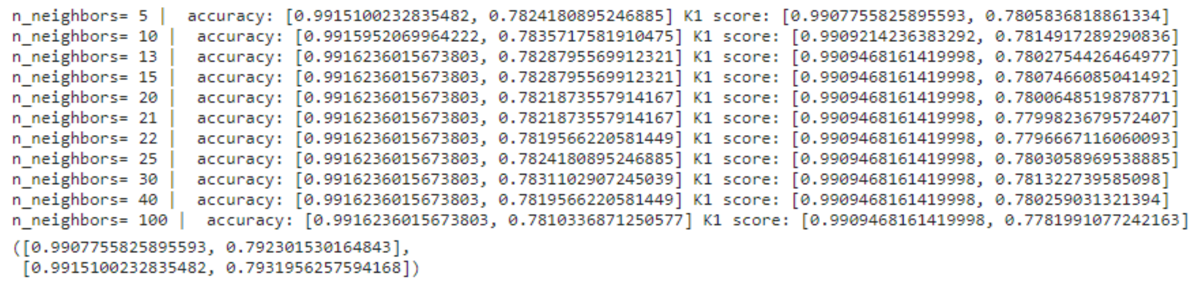
Team member contribution breakdown:

Sophia Politylo	- KNN model - Random forest mode - Data analysis - Introduction - Data preprocessing and splitting - Previous work - Methodology - Evaluation - Conclusion **Contributed Percentage:**
Joshua McAllister	
Bryce Hart	- LSTM model - Logistic regression model - Feedforward neural networks - Data preprocessing and splitting - Document proof reading and editing **Contributed Percentage: 40%**
Wei-Yuan Chuang	

Group members' signatures signifying that this is indeed correct.

Sophia Politylo	**Joshua McAllister**	**Bryce Hart**	**Wei-Yuan Chuang**
[illegible]		[illegible]	

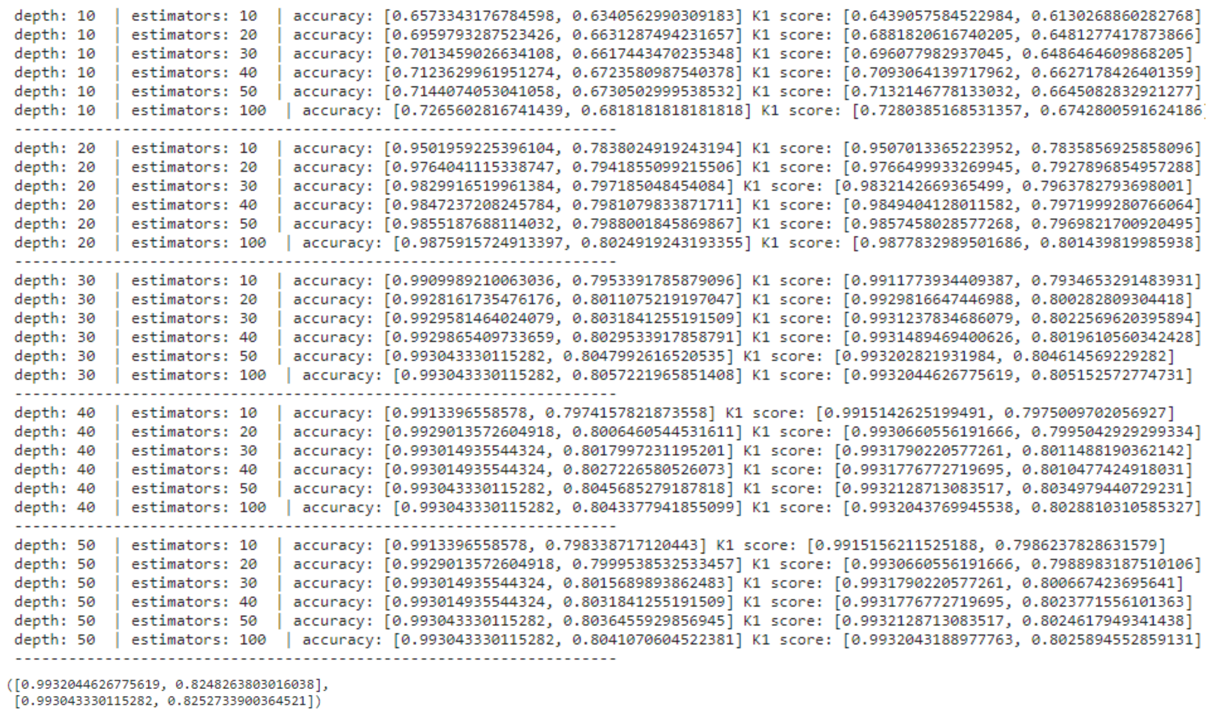
KKN results for fine tunning



```
n_neighbors= 5 |  accuracy: [0.9915100232835482, 0.7824180895246885] K1 score: [0.9907755825895593, 0.7805836818861334]
n_neighbors= 10 |  accuracy: [0.9915952069964222, 0.7835717581910475] K1 score: [0.9909214236383292, 0.7814917289290836]
n_neighbors= 13 |  accuracy: [0.9916236015673803, 0.7828795569912321] K1 score: [0.9909468161419998, 0.7802754426464977]
n_neighbors= 15 |  accuracy: [0.9916236015673803, 0.7828795569912321] K1 score: [0.9909468161419998, 0.7807466085041492]
n_neighbors= 20 |  accuracy: [0.9916236015673803, 0.7821873557914167] K1 score: [0.9909468161419998, 0.7800648519878771]
n_neighbors= 21 |  accuracy: [0.9916236015673803, 0.7821873557914167] K1 score: [0.9909468161419998, 0.7799823679572407]
n_neighbors= 22 |  accuracy: [0.9916236015673803, 0.7819566220581449] K1 score: [0.9909468161419998, 0.7796667116060093]
n_neighbors= 25 |  accuracy: [0.9916236015673803, 0.7824180895246885] K1 score: [0.9909468161419998, 0.7803058969538885]
n_neighbors= 30 |  accuracy: [0.9916236015673803, 0.7831102907245039] K1 score: [0.9909468161419998, 0.781322739585098]
n_neighbors= 40 |  accuracy: [0.9916236015673803, 0.7819566220581449] K1 score: [0.9909468161419998, 0.780259031321394]
n_neighbors= 100 |  accuracy: [0.9916236015673803, 0.7810336871250577] K1 score: [0.9909468161419998, 0.7781991077242163]

([0.9907755825895593, 0.792301530164843],
 [0.9915100232835482, 0.7931956257594168])
```

Random tree results for fine tunning



```
depth: 10  | estimators: 10  | accuracy: [0.6573343176784598, 0.6340562990309183] K1 score: [0.6439057584522984, 0.6130268860282768]
depth: 10  | estimators: 20  | accuracy: [0.6959793287523426, 0.6631287494231657] K1 score: [0.6881820616740205, 0.6481277417873866]
depth: 10  | estimators: 30  | accuracy: [0.7013459026634108, 0.6617443470235348] K1 score: [0.696077982937045, 0.6486464609868205]
depth: 10  | estimators: 40  | accuracy: [0.7123629961951274, 0.6723580987540378] K1 score: [0.7093064139717962, 0.6627178426401359]
depth: 10  | estimators: 50  | accuracy: [0.7144074053041058, 0.6730502999538532] K1 score: [0.7132146778133032, 0.6645082832921277]
depth: 10  | estimators: 100  | accuracy: [0.7265602816741439, 0.6818181818181818] K1 score: [0.7280385168531357, 0.6742800591624186
-----------------------------------------------------------------
depth: 20  | estimators: 10  | accuracy: [0.9501959225396104, 0.7838024919243194] K1 score: [0.9507013365223952, 0.7835856925858096]
depth: 20  | estimators: 20  | accuracy: [0.9764041115338747, 0.7941855099215506] K1 score: [0.9766499933269945, 0.7927896854957288]
depth: 20  | estimators: 30  | accuracy: [0.9829916519961384, 0.797185048454084] K1 score: [0.9832142669365499, 0.7963782793698001]
depth: 20  | estimators: 40  | accuracy: [0.9847237208245784, 0.7981079833871711] K1 score: [0.9849404128011582, 0.7971999280766064]
depth: 20  | estimators: 50  | accuracy: [0.9855187688114032, 0.7988001845869867] K1 score: [0.9857458028577268, 0.7969821700920495]
depth: 20  | estimators: 100  | accuracy: [0.9875915724913397, 0.8024919243193355] K1 score: [0.9877832989501686, 0.801439819985938]
-----------------------------------------------------------------
depth: 30  | estimators: 10  | accuracy: [0.9909989210063036, 0.7953391785879096] K1 score: [0.9911773934409387, 0.7934653291483931]
depth: 30  | estimators: 20  | accuracy: [0.9928161735476176, 0.8011075219197047] K1 score: [0.9929816647446988, 0.800282809304418]
depth: 30  | estimators: 30  | accuracy: [0.9929581464024079, 0.8031841255191509] K1 score: [0.9931237834686079, 0.8022569620395894]
depth: 30  | estimators: 40  | accuracy: [0.9929865409733659, 0.8029533917858791] K1 score: [0.9931489469400626, 0.8019610560342428]
depth: 30  | estimators: 50  | accuracy: [0.993043330115282, 0.8047992616520535] K1 score: [0.993202821931984, 0.804614569229282]
depth: 30  | estimators: 100  | accuracy: [0.993043330115282, 0.8057221965851408] K1 score: [0.9932044626775619, 0.805152572774731]
-----------------------------------------------------------------
depth: 40  | estimators: 10  | accuracy: [0.9913396558578, 0.7974157821873558] K1 score: [0.9915142625199491, 0.797500970205692 7]
depth: 40  | estimators: 20  | accuracy: [0.9929013572604918, 0.8006460544531611] K1 score: [0.9930660556191666, 0.7995042929299334]
depth: 40  | estimators: 30  | accuracy: [0.993014935544324, 0.8017997231195201] K1 score: [0.9931790220577261, 0.8011488190362142]
depth: 40  | estimators: 40  | accuracy: [0.993014935544324, 0.8027226580526073] K1 score: [0.9931776772719695, 0.8010477424918031]
depth: 40  | estimators: 50  | accuracy: [0.993043330115282, 0.8045685279187818] K1 score: [0.9932128713083517, 0.8034979440729231]
depth: 40  | estimators: 100  | accuracy: [0.993043330115282, 0.8043377941855099] K1 score: [0.9932043769945538, 0.8028810310585327]
-----------------------------------------------------------------
depth: 50  | estimators: 10  | accuracy: [0.9913396558578, 0.798338717120443] K1 score: [0.9915156211525188, 0.7986237828631579]
depth: 50  | estimators: 20  | accuracy: [0.9929013572604918, 0.7999538532533457] K1 score: [0.9930660556191666, 0.7988983187510106]
depth: 50  | estimators: 30  | accuracy: [0.993014935544324, 0.8015689893862483] K1 score: [0.9931790220577261, 0.800667423695641]
depth: 50  | estimators: 40  | accuracy: [0.993014935544324, 0.8031841255191509] K1 score: [0.9931776772719695, 0.8023771556101363]
depth: 50  | estimators: 50  | accuracy: [0.993043330115282, 0.8036455929856945] K1 score: [0.9932128713083517, 0.8024617949341438]
depth: 50  | estimators: 100  | accuracy: [0.993043330115282, 0.8041070604522381] K1 score: [0.9932043188977763, 0.8025894552859131]
-----------------------------------------------------------------

([0.9932044626775619, 0.8248263803016038],
 [0.993043330115282, 0.8252733900364521])
```