Name - Anthony Awobasivwe, URN – 6442385, Team Number - Group 2

Project Plan

Section 1 - Group Declaration

URN	Name
6533102	Alex Franke
6478023	Ana Paniagua Cruz
6760445	Khalil El Daou
6435376	Humza Malik-Ramzan
6442385	Anthony Awobasivwe

Section 2 – 13 Labels

1. Neutral
2. Confusion + curiosity
3. realisation + surprise
4. Sadness + grief
5. Approval + pride
6. Admiration
7. Fear + Nervousness + Embarrassment
8. Gratitude + Relief
9. Joy + Amusement
10. Remorse + disappointment
11. Desire + excitement + optimism
12. Annoyance + Anger
13. Disapproval + Disgust
14. Love + caring

To group the labels into 14 categories, we grouped words which had a similar meaning. We did this in a way that aimed to group low frequency words with high frequency words. **Note there was up slight tweak to the original merging plan with disgust now belonging to class 13 and annoyance to class 12. This was because on further observation of multi labels we observed a high number of these contained disgust and anger combinations**

Section 3 – Experiment Plan

Time/resource constraints – We want to avoid implementing computationally expensive models as these may result in out-of-memory errors or increase run times. We will also seek to reduce vocabulary size through a variety of methods such as stop word and punctuation removal.

Scope – Try five different models and experiment with various parameters to achieve the best possible output.

Priorities – Try a wide variety of experiments as a team, performance, good accuracy (F1 score), choosing the best model, good pre-processing “Rubbish in, rubbish out”, and good model features.

Section 4 – Development Environment

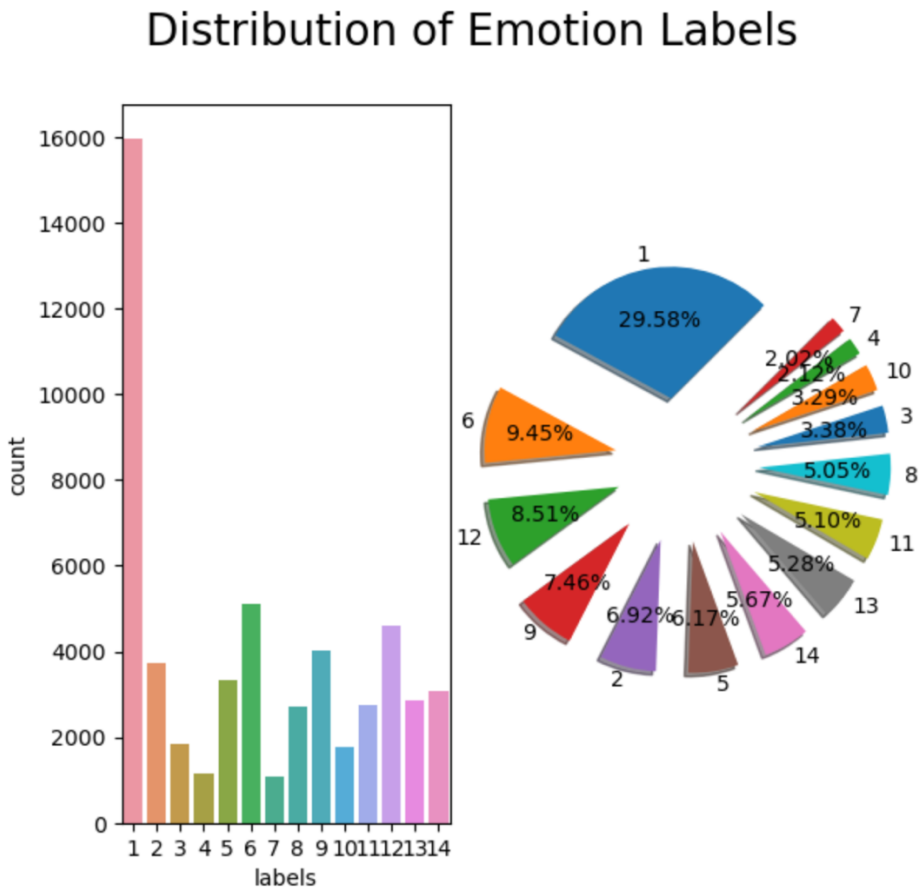
We will use Jupyter Notebook as our main working environment. We will also make use of the following Python libraries: NumPy, Pandas, Matplotlib, Seaborn, Sklearn, NLTK, spaCy, Keras, PyTorch, XGBoost, gensim. Section 5 – Task Distribution

Name	Task
Ana	Pre-Trained Model (BERT)

Alex	Support Vector Machine
Khalil	Extreme Gradient Boosting (XGBoost)
Anthony	Recurrent Neural Network
Humza	Convolutional Neural Network

All team members will independently conduct pre-processing and text representation in accordance with their chosen model. Pre-processing will be in the form of stemming, lemmatization, removal of stop words, etc. For text representation, we will explore basic vectorization approaches and distributed representations (such as word2vec, GloVe, or custom embeddings using Gensim)

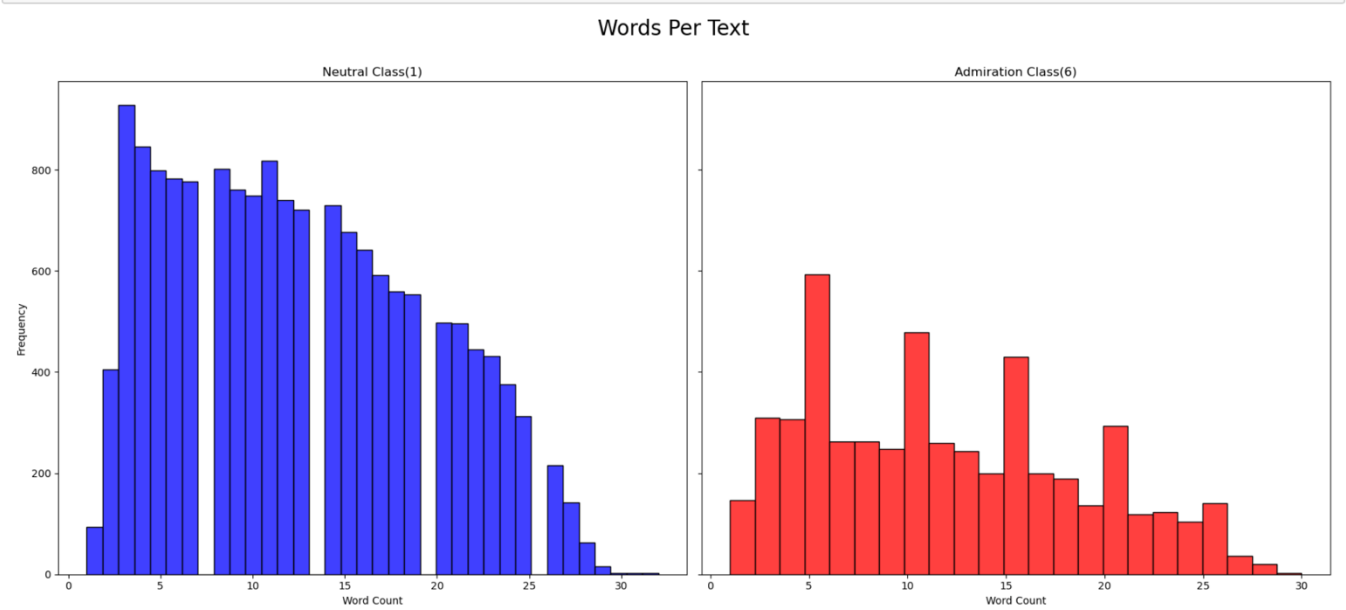
Data Analysis & Visualization



From this bar/pie charts, we see about 30 percent of the data is contained within class 0 .The second highest representation is from class 6 (10 percent)and the lowest representation are classes 4 and 7 with both having about 2 percent of the data respectively .

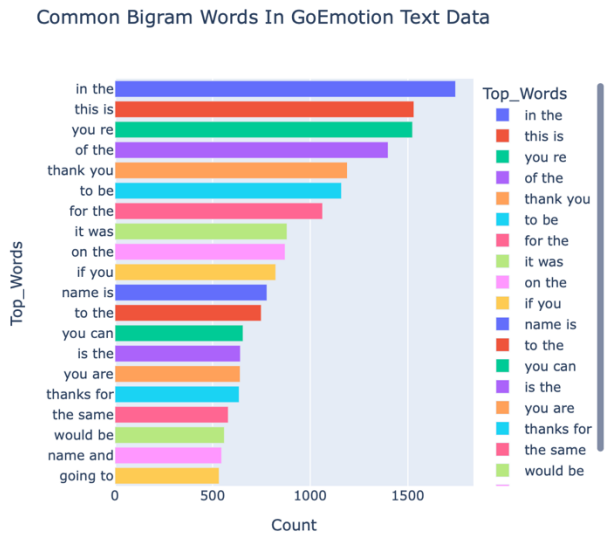
Word Count

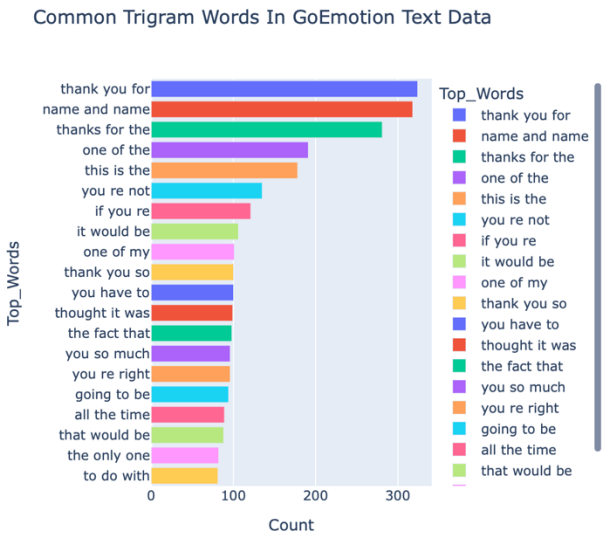
I then checked word count and frequency for most popular classes .



From this we see the neutral class highest frequency is trigrams and the admiration's class highest frequency is 4 grams.

Bigram vs Trigram Analysis





Most Common words in text with bigrams is 'in the' and with trigrams is 'thank you for'. As expected there are a lot of stop words in most common words which will be addressed later in the project.

Experiments Overview

For this project, the following experimentations were chosen were each experiment had a minimum of 2 variations and a maximum of 4 variations

1. Experiments with different NLP algorithms/techniques

2. Experiments with different data pre-processing techniques

3. Experiments with different train/test/validate dataset splits

4. Experiments with different hyperparameter optimization techniques

In terms of additional experiments carried out during the hyperparameter optimization, I also looked into experimenting with different optimizers.

For each experiment the following visualizations/metrics were considered

1. Accuracy Curve - Accuracy vs epoch plot is a visualization method used to show the model accuracy across several epochs. This is an important visualization as it helps to show if the model was overfitting or underfitting and also helps identify the optimal number of epochs needed to train the model.
2. Loss Curve - A loss vs epoch plot is a visualization method used to show how the model's error changes during training across several epochs. This was an important visualization as it helps identify if the model is learning properly and converged to a stable solution as a loss curve with losses decreasing quickly and leveling off after several epochs tends to indicate model convergence.
3. Confusion Matrix – Used for visualization of metrics such as:

a) False Negatives - This metric tells the number of predictions the model predicted as not belonging to a particular class when it belonged to that class.
b) False Positives - This metric tells the number of predictions that model predicted as belonging to a particular class when it belonged to another class.
c) True Positives - This metric tells the number of correct predictions the model made for classifying each emotion.

4. Classification Report – This would be used as it gives a full report of evaluation metrics. This provides metrics such as:
    a. Accuracy - the overall model's ability to make correct classifications.
    b. Recall - models ability to minimize false negatives.
    c. Precision- models ability to minimize false positives.
    d. F1-Score – harmonic mean of precision and recall, useful for imbalanced datasets such as ours.

To manage the project flow amid having a vast number of variations, the experiments followed a built-up pattern where the first experiment made use of base preprocessing/text representation, and then based on what model gave the best performance for that experiment that model will then be used for the next experiment and so on . The chosen text representation model was glove embedding as it works well with sequential data.

Hyperparameters such as activation, dense, and optimizer learning rate were kept the same for all variations in experiments 1,2,3. Optimiser set Adam with a default learning rate (0.001). Each model had two layers one with 64 units and the other with 32 units to reduce model complexities and possibly reduce training times. To further maintain consistency across experiments I will be using the same random state (70) across all experiments.

Experiment 1 - NLP Algorithms and techniques

My first experiment involved training my model on different types of recurrent neural networks. To remove bias in performance across all models used the Adam optimizer with learning rates set to default

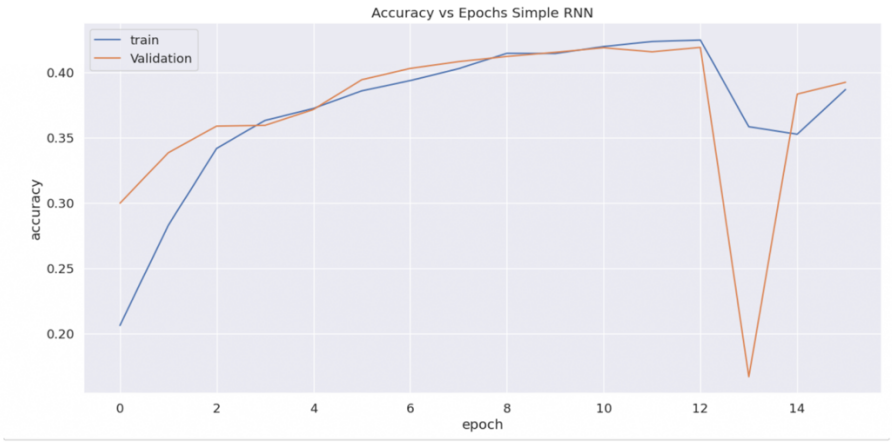
For the focus of the experiment the following variations were considered:

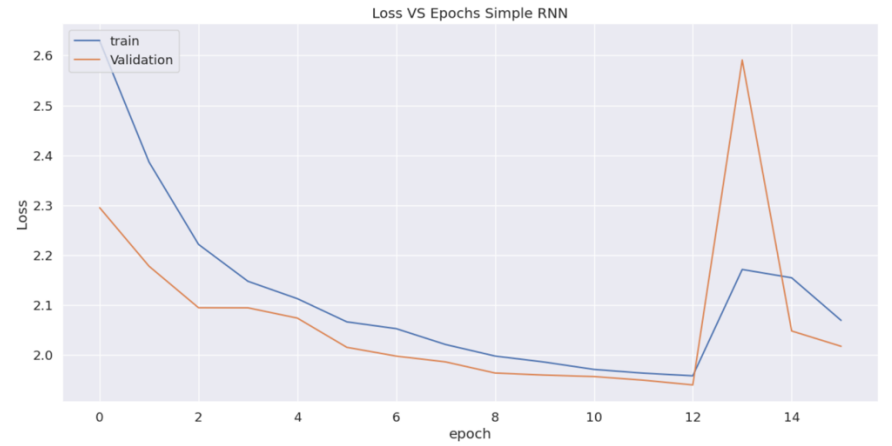
1. Simple RNN
2. Long Short-Term Memory (LSTM)
3. Gated Recurrent Unit (GRU)

Simple RNN

Simple Recurrent Neural Network (RNN) is a neural network that can be used for sequential data such as ours. In RNN each input in the sequence is reliant on the previous inputs with the hidden state that is updated after each time step. The inputs are then multiplied by a weight matrix at each time step and then added to the hidden state from the previous time step, which is also multiplied by a weight matrix. The output is then passed through an activation function in our case softmax, to generate the new hidden state. The new hidden state can then be either used to make predictions or fed back into the network as input for the next time step.

Accuracy Curve and Loss Curve Simple RNN

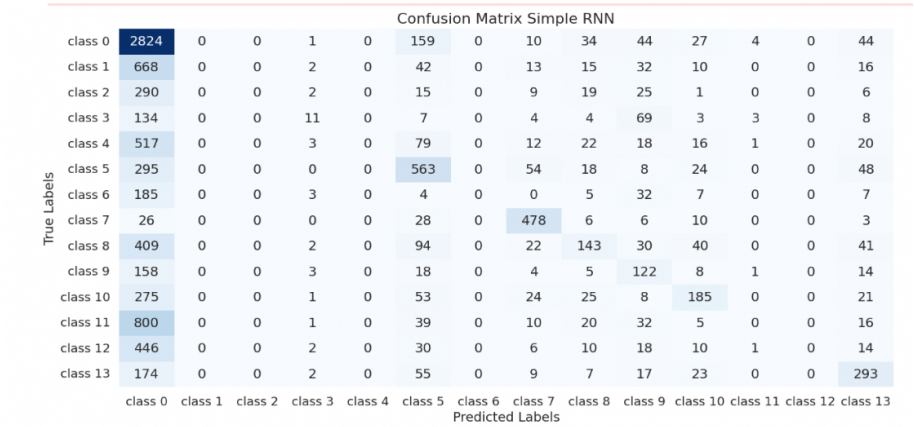




From the accuracy curve, we see a steady increase in accuracy for both training and validation sets as the number of epochs increases At around the 12th epoch both accuracies decrease with the validation accuracy dropping rapidly. This shows overfitting and the ideal number of epochs was around 10-12 epochs. This can also be seen on the loss curve with loss values rapidly increasing after the 12th epoch

Confusion Matrix / Classification Report Simple RNN

```
338/338 [==============================] - 10s 28ms/step
              precision    recall  f1-score   support

           0       0.39      0.90      0.55      3147
           1       0.00      0.00      0.00       798
           2       0.00      0.00      0.00       367
           3       0.33      0.05      0.08       243
           4       0.00      0.00      0.00       688
           5       0.47      0.56      0.51      1010
           6       0.00      0.00      0.00       243
           7       0.73      0.86      0.79       557
           8       0.43      0.18      0.26       781
           9       0.26      0.37      0.31       333
          10       0.50      0.31      0.39       592
          11       0.00      0.00      0.00       923
          12       0.00      0.00      0.00       537
          13       0.53      0.51      0.52       580

    accuracy                           0.43     10799
   macro avg       0.26      0.27      0.24     10799
weighted avg       0.30      0.43      0.33     10799
```



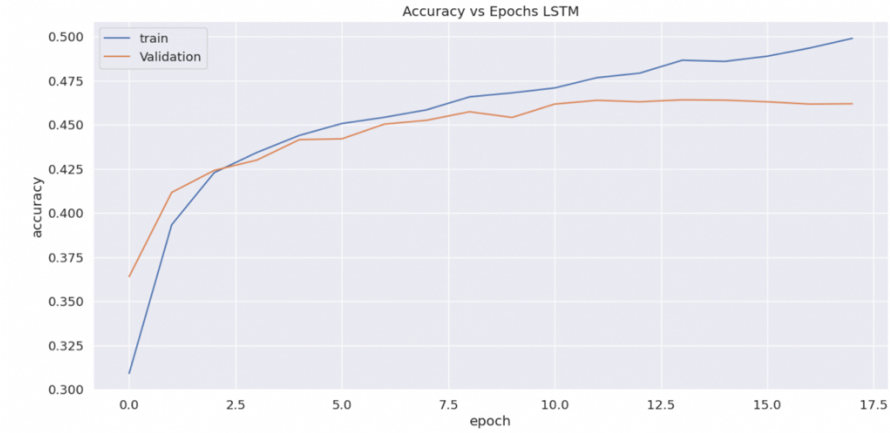
From the classification report, we see the simple rnn model performance wasn't optimal. On initial observation the model accuracy was 43 percent which depending on the use case may be seen as ok accuracy, however on further inspection only four classes had better performance better than the accuracy gotten (class 0, class 5, class 7, class 13). Classes such as class 1 , 2 , 4, 6, 11, 12 performed horribly across all metrics with the model not able to any correct classifications . This is further reflected when looking at the confusion matrix with the model not being to make any correct classifications in some classes leading to lots of false negatives/positives and overall lower precision/recall averages.

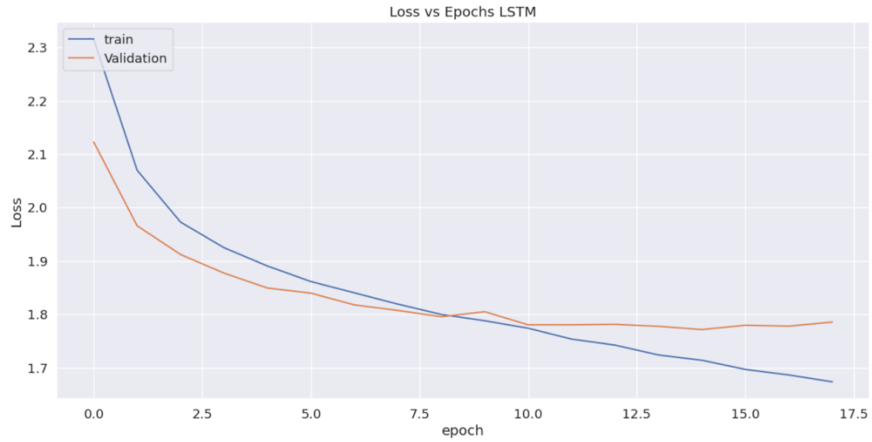
Long Short-Term Memory (LSTM)

LSTM, or Long Short-Term Memory, is a type of recurrent neural network (RNN) architecture that addresses the vanishing gradient problem of our first variation(Simple RNN). LSTMs have additional gates (input, output, forget) compared to simple rnns which regulate the flow of information to and

from the memory cell. These gates are responsible for determining the amount of new and old information that should be allowed into the cell and the amount that should be output from it. The memory cell is also able to store information for longer periods making lstm effective at learning longer-term dependencies.

Accuracy Curve and Loss Curve LSTM

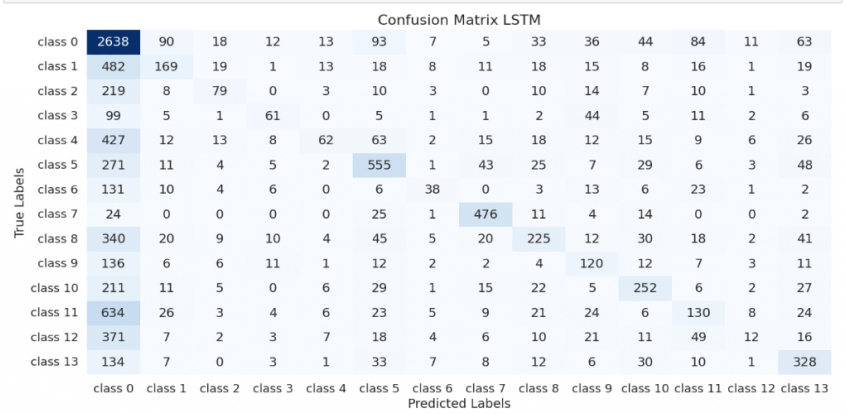




From the accuracy curve, we see a steady increase in accuracy for both training and validation sets as the number of epochs increases similarly to the simple rnn model at around the 12th epoch both accuracies start to decrease. However, the drop-off in accuracies between training and validation sets isn't as big compared with the simple rnn model which shows this model is less prone to overfitting. This is also seen with the loss curve which shows model convergence around the 12th epoch.

Confusion Matrix / Classification Report *LSTM*

```
338/338 [==============================] - 3s 6ms/step
              precision    recall  f1-score   support

           0       0.43      0.84      0.57      3147
           1       0.44      0.21      0.29       798
           2       0.48      0.22      0.30       367
           3       0.49      0.25      0.33       243
           4       0.53      0.09      0.15       688
           5       0.59      0.55      0.57      1010
           6       0.45      0.16      0.23       243
           7       0.78      0.85      0.82       557
           8       0.54      0.29      0.38       781
           9       0.36      0.36      0.36       333
          10       0.54      0.43      0.48       592
          11       0.34      0.14      0.20       923
          12       0.23      0.02      0.04       537
          13       0.53      0.57      0.55       580

    accuracy                           0.48     10799
   macro avg       0.48      0.35      0.38     10799
weighted avg       0.47      0.48      0.43     10799
```

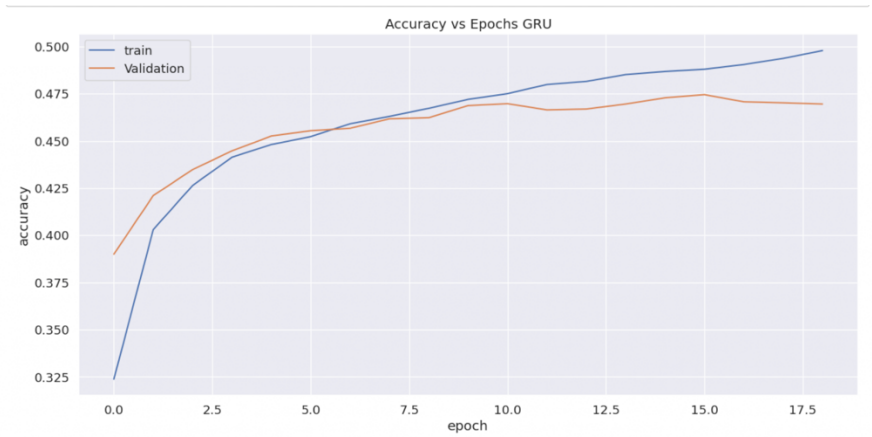


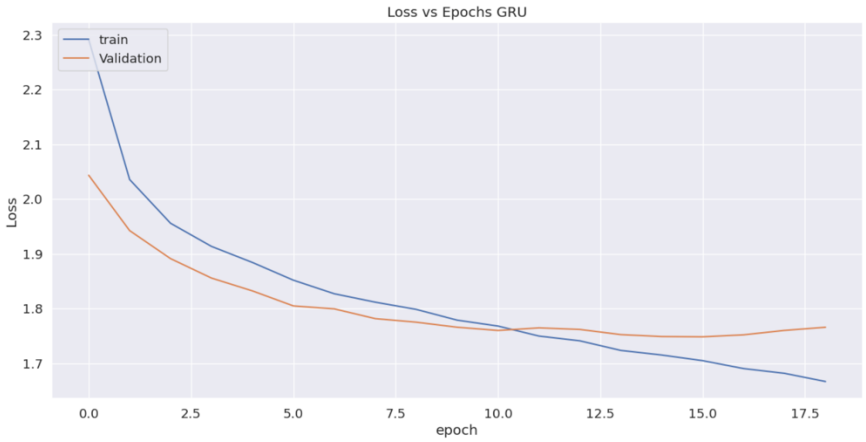
From the classification report, we see the lstm model performed slightly better in terms of overall accuracy compared with the previous model(simple rnn). On further inspection, the model did substantially better than the simple rnn model in terms of spreading out classifications with each class having some correct predictions made. Classes 1 and 7 remained the best performing classes, the performances of the classes that gave zero performance in variation 1 all increased however those said classes remained among the lowest performers. this can be further observed in the confusion matrix with every class having some correct classifications made compared to the simple rnn model where the model wasn't able to make any predictions in several classes. This is further reflected by this model giving better higher macro and weighted averages across metrics(recall, precision, recall).

Gated Recurrent Unit (GRU)

Gated Recurrent Units (GRU) are similar to LSTMS as they maintain an internal state representative of the sequential context of the data. However, unlike lstms , Gru's only has two gates (reset and update) that regulate the flow of information to and from the hidden state. The reset gate determines how much of the prior hidden state to discard, while the update gate determines how much of the new input to pass into the hidden state which allows the GRU to update its internal state selectively based on the input and previous state
Apart from having similar performance to lstm's, gru's provide a computationally efficient alternative due to it having fewer parameters and gates .

Accuracy Curve and Loss Curve

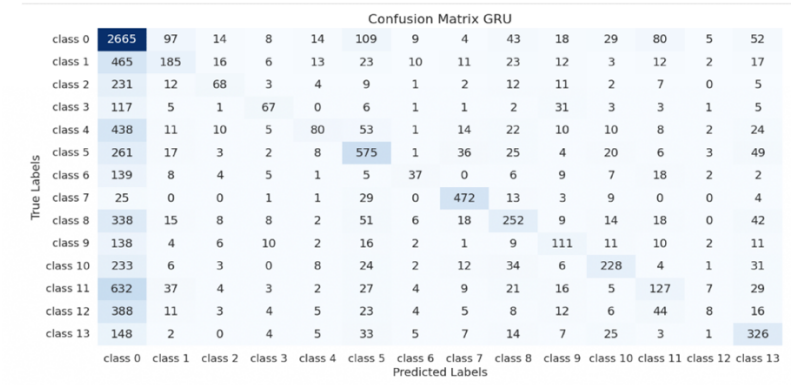




From the accuracy curve, similar to the lstm model, we see a steady increase in accuracy for both training and validation sets as the number of epochs increases. At around the 11th epoch, both accuracies start to decrease. This is also seen with the loss curve which shows model convergence around the 12th epoch. The accuracy and loss curves for both gru and lstm models look identical with similar accuracy peaks and convergence points.

Confusion Matrix / Classification Report

```
338/338 [==============================] - 2s 5ms/step
              precision    recall  f1-score   support

           0       0.43      0.85      0.57      3147
           1       0.45      0.23      0.31       798
           2       0.49      0.19      0.27       367
           3       0.53      0.28      0.36       243
           4       0.55      0.12      0.19       688
           5       0.58      0.57      0.58      1010
           6       0.45      0.15      0.23       243
           7       0.80      0.85      0.82       557
           8       0.52      0.32      0.40       781
           9       0.43      0.33      0.38       333
          10       0.61      0.39      0.47       592
          11       0.37      0.14      0.20       923
          12       0.24      0.01      0.03       537
          13       0.53      0.56      0.55       580

    accuracy                           0.48     10799
   macro avg       0.50      0.36      0.38     10799
weighted avg       0.48      0.48      0.43     10799
```



From the classification report, we see the gru model performed eerily similar to the lstm model in terms of overall accuracy. On further inspection, the gru model also had similar performance across different metrics such as precision, recall, and f1 score with just slightly better performance compared with lstm in terms of f1 score of class 4 and 1, two classes notorious for their bad performance. Like the lstm the gru model's best-performing classes were classes 0 and 7 which can be seen in the confusion matrix with these classes having a low number of false negatives/positives. We also see from the confusion matrix the model was able to make classifications across all class labels with each class having some correct predictions made.

Conclusion From Experiment 1

The lstm and gru models performed a lot better than the simple rnn. This was expected as stated previously that both lstm and gru are computationally more effective than the standard RNNs because they explicitly attempt to address the vanishing and exploding gradient problems. The lstm and gru performance was very similar which was also expected as stated earlier hence I tried one more variation from my notebook having both lstm and gru layers to see if performance was increased . The performance of the lstm model with gru layers was quite similar to both lstm and gru models with

slightly less overall performance. Taking all this into consideration, I decided on using the gru model for all future experiments as it had slightly better performance than lstm.

Experiment 2 -Train Test Validation Splits Variation

Building on my first experiment , my second experiment involved experimenting with different data splits to see the effect of each on the performance of the model which performed best in the experiment 1 (GRU) and to see if any of these splits gave me better performance than my original split used in experiment 1 (70/15/15) . For the purpose of this experiment the following data splits were used where XX/XX/XX means train/test/validation.

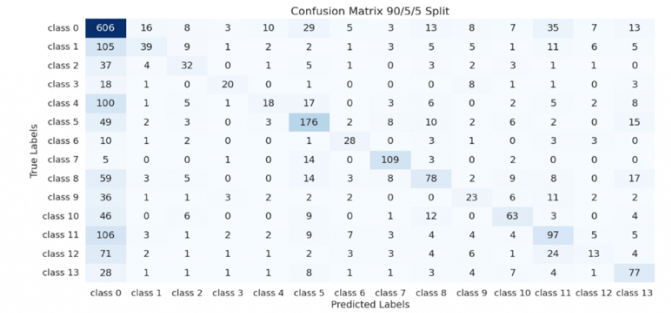
1. 90/5/5 Split
2. 10/45/45 Split
3. 25/25/50 Split
4. 25/50/25 Split

90/5/5 Split - Variation 1

For the First split, I increased the amount of data by 20 percent from my base 70/15/15 split used in my first experiment. My goal was to see the effects using a very large sample size of training data will have on my model performance.

Confusion Matrix / Classification Report

```
85/85 [==============================] - 2s 14ms/step
              precision    recall  f1-score   support

           0       0.47      0.79      0.59       763
           1       0.53      0.20      0.29       195
           2       0.43      0.36      0.39        90
           3       0.61      0.38      0.47        53
           4       0.45      0.11      0.17       168
           5       0.61      0.63      0.62       278
           6       0.53      0.54      0.53        52
           7       0.77      0.81      0.79       134
           8       0.54      0.38      0.45       206
           9       0.35      0.25      0.29        91
          10       0.56      0.44      0.49       144
          11       0.47      0.38      0.42       252
          12       0.33      0.10      0.15       136
          13       0.50      0.56      0.53       138

    accuracy                           0.51      2700
   macro avg       0.51      0.42      0.44      2700
weighted avg       0.51      0.51      0.48      2700
```



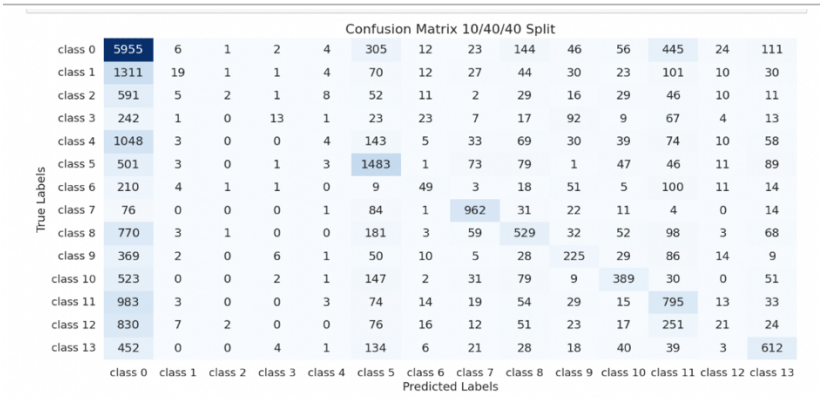
From the classification report, we see the 90/5/5 split performed better than the base 70/15/15 used with an overall accuracy of 51 percent. More importantly, metrics such as precision, recall, and f1 score were all higher than previous. In terms of precision, most classes had average to good performance with class 3 and class 7 performing the best while the only models having below 30 percent performance were class 9 and class 12. On further inspection the low precision performance of class 3 could be because of its low representation in the dataset however for class 12 I am not entirely sure about its bad performance as this label was among the top 3 in terms of representation. Possibly the poor performance may have been from initial data merging. In terms of recall as expected class 0 (neutral) remained among the highest performing classes due to its high representation however class 7 was the best performing which is a huge contrast to my earlier theory as class 0 has the highest representation while class 7 had the lowest representation. Again I will leave the high recall performance in class 7 possibly due to proper merging in an earlier stage of the project. This high performance is further reflected in the f1 score with classes 0 and 7 being 2 out of the top 3 in terms of f1 score performance. For recall the lowest-performing models were class 12 and class 4. In particular, there is a concern for these two labels as they were also lower performing in terms of precision, this can further be reflected by these two having the lowest f1 scores which show our model struggled for making predictions for these class labels which can further be seen looking at the confusion matrix . The better performance apart from neural networks, in general, performing better on more training data, could be because a larger dataset would mean there potentially would be less out of vocabulary words leading to better model performance

10/45/45 Split - Variation 2

My second variation was the opposite of the first variation where in this case I wanted to see the effects using a very small sample size of training data will have on my model performance. From general knowledge of neural networks, I deduce this model may perform poorly due to neural networks performing better on larger training data. However, I was still interested to see if this holds for this particular dataset

Confusion Matrix / Classification Report

	precision	recall	f1-score	support
0	0.43	0.83	0.57	7134
1	0.34	0.01	0.02	1683
2	0.25	0.00	0.00	813
3	0.42	0.03	0.05	512
4	0.13	0.00	0.01	1516
5	0.52	0.63	0.57	2338
6	0.30	0.10	0.15	476
7	0.75	0.80	0.77	1206
8	0.44	0.29	0.35	1799
9	0.36	0.27	0.31	834
10	0.51	0.31	0.38	1264
11	0.36	0.39	0.38	2035
12	0.16	0.02	0.03	1330
13	0.54	0.45	0.49	1358
accuracy			0.46	24298
macro avg	0.39	0.30	0.29	24298
weighted avg	0.41	0.46	0.38	24298



From the classification report, we see the 10/45/45 split performed badly compared to both the base 70/15/15

and variation (90/5/5) was used with an overall accuracy of 46 percent. All Metrics were also lower across the board. In terms of precision, most classes had poor performance, however just like variation 1 class 7 performed best having almost identical performance compared to when trained on variation 1 which had a lot more training data. The worst-performing models in terms of precision were classes 4 and 12. Notably, these two labels gave the lowest recall from the 1st variation. In terms of recall class 0 and class 7 just like in variation 1 gave very high recall performance with scores in the 80s. This variation, however, did quite worse in general in terms of recall as they are now several labels (class 1,2,3,4.12) with 0 or almost 0 representation. This can be further seen from the confusion matrix with said labels having a substantial amount of false negatives.

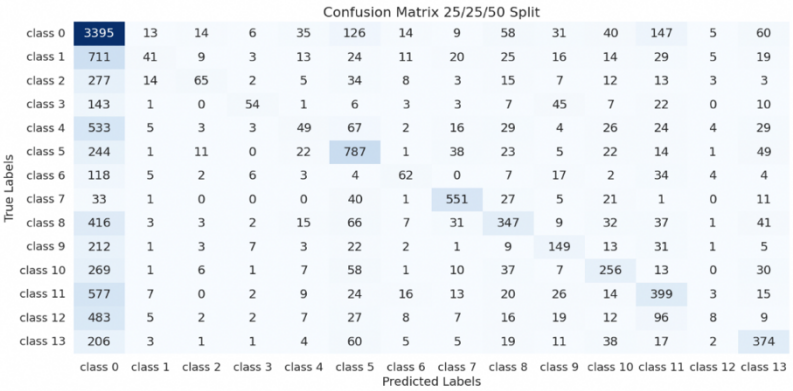
The worse performance in variation 2 matched my initial deduction of the neural net model performing worse due to a smaller training and more out-of-vocabulary words leading to worse model performance. Also, the model's ability to perform well in classes 0 and 7 regardless of training size could be due to these classes not having a lot of oov words even on smaller sample sizes

25/25/50 Split and 25/50/25 Split – Variation 3 and 4

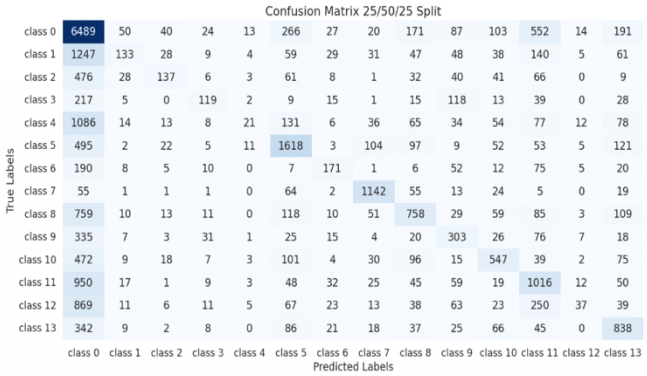
My third and fourth variation took a slightly different approach from the previous variation. The main goal of this experiment variation was to see if having different weights of validation and test sets would have any effect on the model performance, hence this variation gave double the weights to the validation set while reducing the weight of the training and test sets and vice versa for the fourth variation .

Confusion Matrix / Classification Report

```
418/418 [==============================] - 3s 5ms/step
              precision    recall  f1-score   support

           0       0.45      0.86      0.59      3953
           1       0.41      0.04      0.08       940
           2       0.55      0.14      0.22       461
           3       0.61      0.18      0.28       302
           4       0.28      0.06      0.10       794
           5       0.59      0.65      0.61      1218
           6       0.44      0.23      0.30       268
           7       0.78      0.80      0.79       691
           8       0.54      0.34      0.42      1010
           9       0.42      0.32      0.37       459
          10       0.50      0.37      0.42       696
          11       0.45      0.35      0.40      1125
          12       0.22      0.01      0.02       701
          13       0.57      0.50      0.53       746

    accuracy                           0.49     13364
   macro avg       0.49      0.35      0.37     13364
weighted avg       0.48      0.49      0.43     13364
```




```
848/848 [==============================] - 5s 6ms/step
              precision    recall  f1-score   support

           0       0.46      0.81      0.59      8047
           1       0.44      0.07      0.12      1879
           2       0.47      0.15      0.23       908
           3       0.46      0.20      0.28       581
           4       0.32      0.01      0.02      1635
           5       0.61      0.62      0.62      2597
           6       0.47      0.30      0.37       562
           7       0.77      0.83      0.80      1382
           8       0.51      0.38      0.43      2015
           9       0.34      0.35      0.34       871
          10       0.51      0.39      0.44      1418
          11       0.40      0.44      0.42      2286
          12       0.36      0.03      0.05      1455
          13       0.51      0.56      0.53      1497

    accuracy                           0.49     27133
   macro avg       0.47      0.37      0.37     27133
weighted avg       0.48      0.49      0.44     27133
```



From the classification reports, we see very similar performance for both variation 3 and variation 4. The performance of these two variations is worse than variation 1 but better than variation 2. The overall accuracy for both variations was identical(49 percent). In terms of precision classes, 4 and 12 were among the lower-performing classes, however, the precision of both classes increased in variation 4 compared to variation 3. The highest-performing models in terms of precision were as suspected class 7 for both variations as this class has performed well in all previous experiments. In terms of recall classes 1, 4, and 12 were among the lower-performing classes with almost zero representation across these classes with precision barely increasing across the two variations. The highest performing models in terms of precision were again class 0 and 7 for both variations. This can be further seen in the

confusion matrix with classes 1,2,12 having a high amount of false negatives and positives and barely any true positive representation compared to classes 0 and 7.

The performance on both variations 3 and 4 was very similar which was contrary to my original deduction. As I thought variation 3 would perform better as generally bigger weights for validation sets could lead to overfitting as the model would start memorizing the validation patterns instead of general patterns but this clearly doesn't hold for this experiment

***Conclusions From Experiment* 2**

The best-performing model from this experiment was using the 90/5/5 split. The performance obtained on this model was better than when using the base split from experiment 1. Although this variation gave the best results there might be slight issues with the reliability of the performance metrics given the test split was only 5 percent of the data however for the scope of this project I will be moving forward with the 90/5/5 split for all future experiments.

Experiment 3 - Data Preprocessing Techniques

Building on my second experiment , my third experiment involved trying out different preprocessing techniques to see its effect on model performance .The preprocessing techniques considered were :

1. Stop Words Vs No Stop Words – Stop word removal in natural language processing has always been a big topic of debate. For example, the text "The movie was not good at all" when stop words are removed translates to "movie good". From this we can see in this specific case it will lead to misclassification.
2. Lemmatization Vs Stemming - Stemming is a process that removes the last few characters from a word while lemmatization considers the context and converts the word to its meaningful base form.

Hence for this experiment I'll be looking to see how applying stop words or not affects model performance while also considering the effect keeping stop words will have on model training time . I will be considering four variations :

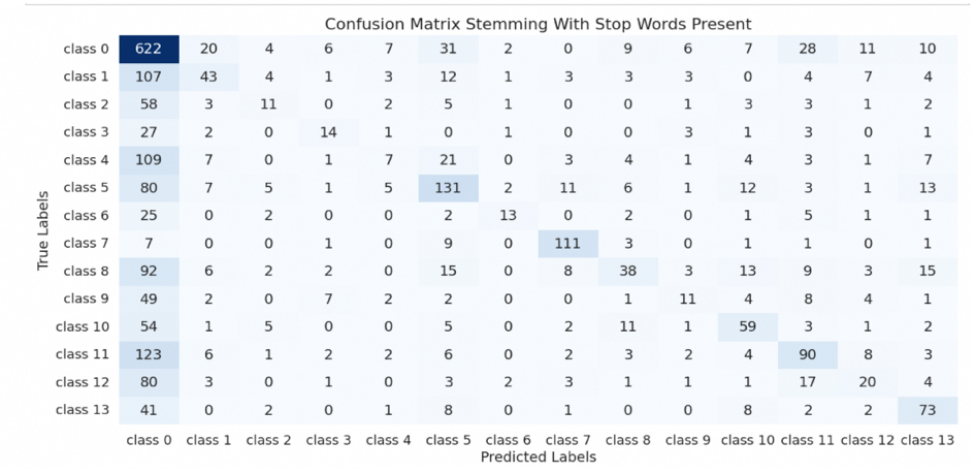
1. Stop words present with lemmatization
2. Stop words present with stemming
3. Stop words absent with lemmatization
4. Stop words absent with stemming

Stop words absent with lemmatization

This was my base model so results and confusion matrix are already present for this from experiment 2 variation 1

Confusion Matrix / Classification Report Stop words present with stemming

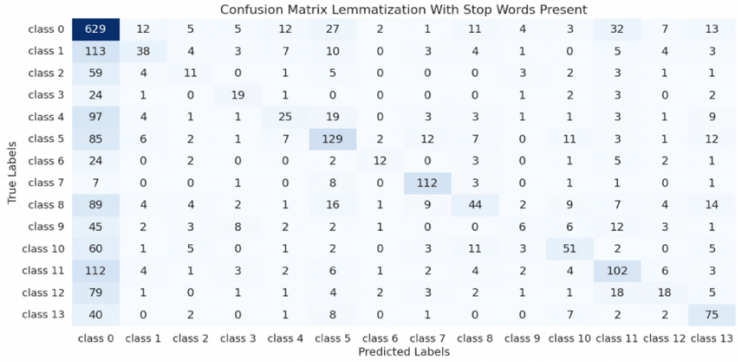
```
85/85 [==============================] - 2s 11ms/step
              precision    recall  f1-score   support

           0       0.42      0.82      0.56       763
           1       0.43      0.22      0.29       195
           2       0.31      0.12      0.17        90
           3       0.39      0.26      0.31        53
           4       0.23      0.04      0.07       168
           5       0.52      0.47      0.50       278
           6       0.59      0.25      0.35        52
           7       0.77      0.83      0.80       134
           8       0.47      0.18      0.26       206
           9       0.33      0.12      0.18        91
          10       0.50      0.41      0.45       144
          11       0.50      0.36      0.42       252
          12       0.33      0.15      0.20       136
          13       0.53      0.53      0.53       138

    accuracy                           0.46      2700
   macro avg       0.45      0.34      0.36      2700
weighted avg       0.45      0.46      0.42      2700
```



From the classification report, we see this model didn't perform as well as variation 1 . Metrics are down across every category. However, just like with previous experiments class 0 and class 7 performance aren't overly affected. The lowest-performing class in terms of recall was class 4. Interestingly with this model although overall results are worse, the model did a better job at averaging out classifications as there is only 1 class (class 4) with close to 0 performance. This can be further analysed in the confusion matrix with class 4 having the lowest number of true positive cases.

Confusion Matrix / Classification Report Stop words present with lemmatization

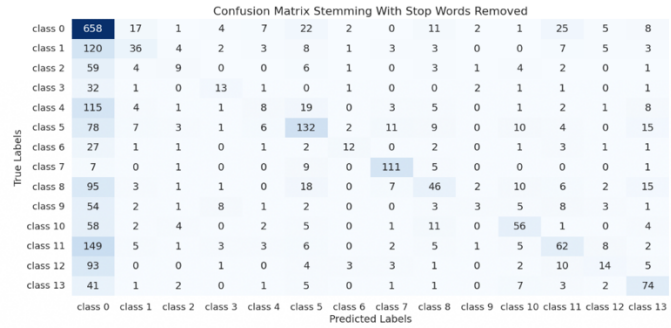
```
85/85 [==============================] - 2s 13ms/step
              precision    recall  f1-score   support

           0       0.43      0.82      0.57       763
           1       0.49      0.19      0.28       195
           2       0.28      0.12      0.17        90
           3       0.43      0.36      0.39        53
           4       0.41      0.15      0.22       168
           5       0.54      0.46      0.50       278
           6       0.57      0.23      0.33        52
           7       0.75      0.84      0.79       134
           8       0.48      0.21      0.30       206
           9       0.25      0.07      0.10        91
          10       0.52      0.35      0.42       144
          11       0.52      0.40      0.45       252
          12       0.37      0.13      0.19       136
          13       0.52      0.54      0.53       138

    accuracy                           0.47      2700
   macro avg       0.47      0.35      0.37      2700
weighted avg       0.47      0.47      0.43      2700
```



From the classification report, we see the performance of this model was similar to performance in variation 2. The recall of class 4 increased however there was a trade-off and a decrease in recall of class 9. In terms of f1 score class 0 and class 7 performed best with class 2and 9 having the lowest performance.

Confusion Matrix / Classification Report Stop words removed with stemming

```
85/85 [==============================] - 2s 17ms/step
              precision    recall  f1-score   support

           0       0.41      0.86      0.56       763
           1       0.43      0.18      0.26       195
           2       0.31      0.10      0.15        90
           3       0.38      0.25      0.30        53
           4       0.24      0.05      0.08       168
           5       0.55      0.47      0.51       278
           6       0.55      0.23      0.32        52
           7       0.78      0.83      0.80       134
           8       0.44      0.22      0.30       206
           9       0.27      0.03      0.06        91
          10       0.54      0.39      0.45       144
          11       0.46      0.25      0.32       252
          12       0.34      0.10      0.16       136
          13       0.53      0.54      0.53       138

    accuracy                           0.46      2700
   macro avg       0.45      0.32      0.34      2700
weighted avg       0.45      0.46      0.41      2700
```



From the classification report, we see this model performed very similar to variation 2 .Although it did have worse performance than variations 1and 3.

***Conclusions From Experiment* 3**

Using stemming led to worse model performance which is to be expected as context Is very important for this dataset. In terms of stop words the performance decreased when using lemmatization, for stemming the results stayed the same. Having stop words also led to increased training times. Taking this into account I will be using variation 1(lemma and no stop words) for my final experiments as this gave the best results.

Experiment Variation 4 - Hyperparameter Optimisation

Building on my previous experiments, my final experiment involved trying to further improve the performance of my model from previous experiments through hyper tuning.

In terms of hyper tuning the gru model, many different parameters could be considered for tuning such as optimizer, learning rates, dropout, activation, units, and loss. However, tuning all these parameters would be impractical as it will take an extensive amount of time to train so for the focus of this experiment I decided to tune the learning rates across different first order optimizers to see its effect on model performance. The optimisers used were :

1. Adaptive moment estimation(Adam)

2. Adaptive Gradient Algorithm (Adagrad)
3. Root mean square propagation (RMSprop)
4. Stochastic gradient descent (SGD)

For the purpose of optimization I considered two approaches:

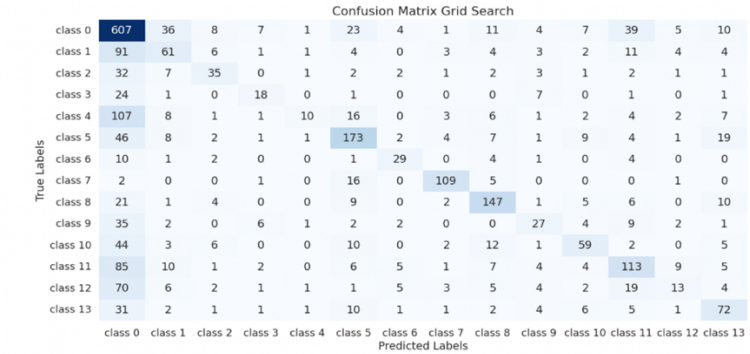
1. Grid Search
2. Random Search

Grid Search

Grid search is a hyper-tuning method used to find the best hyperparameters for a model. In a grid search, the model is trained on all hyperparameter combinations which can be an exhaustive process. For this project, I will be using a GridSearchCV with cross-validation set to 3. As mentioned earlier the hyperparameters to be trained will be optimizer and learning rate.

Confusion Matrix / Classification Report

```
85/85 [==============================] - 1s 17ms/step
              precision    recall  f1-score   support

           0       0.50      0.80      0.62       763
           1       0.42      0.31      0.36       195
           2       0.51      0.39      0.44        90
           3       0.46      0.34      0.39        53
           4       0.59      0.06      0.11       168
           5       0.63      0.62      0.63       278
           6       0.58      0.56      0.57        52
           7       0.84      0.81      0.83       134
           8       0.69      0.71      0.70       206
           9       0.44      0.30      0.36        91
          10       0.58      0.41      0.48       144
          11       0.52      0.45      0.48       252
          12       0.33      0.10      0.15       136
          13       0.52      0.52      0.52       138

    accuracy                           0.55      2700
   macro avg       0.54      0.46      0.47      2700
weighted avg       0.54      0.55      0.52      2700
```



From the classification report, we see the performance using grid search was better than all previous models with an overall accuracy of 55 percent. Apart from accuracy, all other metrics were higher than in previous experiments. The best-performing class in terms of f1 score was class 0 and class 7. The worst performing classes were class. 4 and 12. From the confusion matrix, we also see this model was

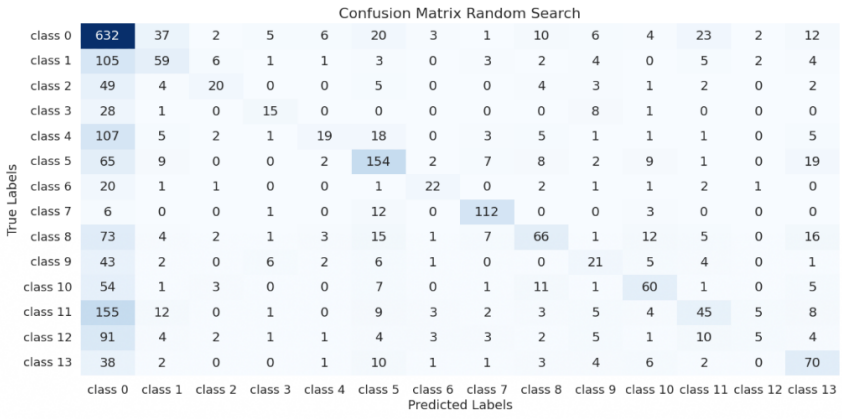
able to minimize false-negative and positives better than previous ones. However, just like with other experiments performance was still abysmally low in classes 4 and 12 which makes builds on my previous beliefs that this low performance is either due to bad merging at the initial stage or underrepresentation in the case of class 4

Random Search

Random search is another hyper-tuning method used to find the best hyperparameters for a model. Unlike grid search , random search only selects and test random sets of hyperparameters . Using this technique can be less computationally expensive however can lead to worse results than from a grid search

Confusion Matrix / Classification Report

```
85/85 [==============================] - 2s 13ms/step
              precision    recall  f1-score   support

           0       0.43      0.83      0.57       763
           1       0.42      0.30      0.35       195
           2       0.53      0.22      0.31        90
           3       0.47      0.28      0.35        53
           4       0.54      0.11      0.19       168
           5       0.58      0.55      0.57       278
           6       0.61      0.42      0.50        52
           7       0.80      0.84      0.82       134
           8       0.57      0.32      0.41       206
           9       0.34      0.23      0.27        91
          10       0.56      0.42      0.48       144
          11       0.45      0.18      0.25       252
          12       0.33      0.04      0.07       136
          13       0.48      0.51      0.49       138

    accuracy                           0.48      2700
   macro avg       0.51      0.38      0.40      2700
weighted avg       0.49      0.48      0.44      2700
```



From the classification report, we see the performance using random search wasn't as good as using grid search with an overall accuracy of 48 percent. It failed to outperform grid search in anyother

metrics. Although as seen from the confusion matrix the performance on class 0 and 7 was still maintained.

Conclusions From Experiment 4 (Grid Search Best Result)

The grid model performed better than all previous models in previous experiments including random search although it did take a longer time to train (36 minutes) best optimizer was the RMSprop with a learning rate of 0.05. RMSprop is an optimizer better suited to handle sparse gradients as it computes the running average of squared gradients for each of the parameters. The learning rate is then divided by the average which effectively scales updates for each of the said parameters based on gradient magnitude. As the dataset is quite imbalanced I expect there were some sparse gradients. In terms of the highest learning rate in the param list being best this could be to higher learning rates leading to faster model convergence or acting as a form of regularisation, preventing the model from overfitting to training data.

The random search also took a while to run(23 minutes) with its best optimizer adagrad . Considering the difference in training time was only 13 minutes I would say grid search is the way to go for this specific use case. It's a point to note that my grid search chose the optimal learning rate as the highest currently specified, in the case I was repeating this experiment in the future I would specify higher learning rate values in the grid search to see if the model performance is improved.

Final Evaluation

I feel although my best model (experiment 4variation 1) did ok (55 percent accuracy), the model still can't be used in a real-world setting to produce sensible predictions across all classes. The f1 score performance in classes such as 4 and 12 was downright awful (10 percent) which is almost equal to a random guess. The model did perform very well for some classes such as 0,7,8 so I think in this sense the model can still be of some use. Given this project again in the future, as none of my models had a particularly standout performance, to attempt to improve this I would do some more research when merging labels as I feel this particularly affected the performance of class 4/12 as no experiment variation was able to classify them properly. I would also look into oversampling and under sampling techniques such as smote for example class 0 (neutral) makes up about 30 percent of the data making the dataset very imbalanced and causing result metrics to be skewed which makes it hard to know what exactly a good accuracy result is for this dataset. Using techniques such as smote may sacrifice the quality of the data as it uses artificial data, however, I feel it will improve the model performance overall. Also if given additional time I would have trained my own corpus to perform bigram vectorisation on my glove model as currently the glove model only supports unigrams .For this reason, I feel for this dataset more attention needs to be paid to the f1 score, however, if considering overall accuracy I would say anything above 60 percent is good enough, unfortunately, none of my models were able to reach that target.

References

Websites

https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

https://www.modelop.com/blog/when-not-to-lemmatize-or-remove-stop-words-in-text-preprocessing/

https://medium.com/@limavallantin/why-is-removing-stop-words-not-always-a-good-idea-c8d35bd77214

https://www.analyticsvidhya.com/blog/2021/06/lstm-for-text-classification/

https://blog.marketmuse.com/glossary/gated-recurrent-unit-gru-definition/#:~:text=The%20Gated%20Recurrent%20Unit%20(GRU,using%20datasets%20with%20longer%20sequences.

https://machinelearningmastery.com/understanding-simple-recurrent-neural-networks-in-keras/

https://www.kdnuggets.com/2022/10/hyperparameter-tuning-grid-search-random-search-python.html#:~:text=While%20grid%20search%20looks%20at,of%20conducting%20an%20exhaustive%20search.

https://www.geeksforgeeks.org/pre-trained-word-embedding-using-glove-in-nlp-models/

https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Papers

https://arxiv.org/abs/2005.00547

https://aclanthology.org/2020.acl-main.372/