cs579 Project 2: Final Report

Ameer ALI – A20424662

Mouhammad BAZZI – A20522180

Department of Computer Science

Illinois Institute of Technology

Problem Statement

The problem that is to be solved in this project is fake news classification. The reason why this is important is that since the advent of social media, it has become more and more popular to use social media as a source of news and information. The caveat with this is that since it is much cheaper, faster, and easier to spread news through social media it makes it such that fake news is easily spread for the purpose of financial/political gain. The spread of fake news can have detrimental effects on individuals and society as it breaks the authenticity balance of the news ecosystem, intentionally persuades people to accept false beliefs, and changes the way people interpret real news/information. For these reasons and many more, it is very important to detect and stop fake news.

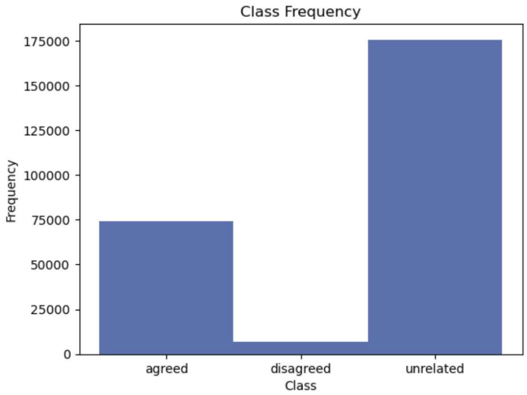
Proposed Solution

The solution that we have proposed is to use multiple networks to help combat fake news. The four networks that are used throughout this project are a simple neural network, a bidirectional LSTM i.e. a recurrent neural network, a more complex neural network, and a transformer network.

To do so we decided to use the data set provided and to fit some supervised learning models in order to then create an ensemble classifier.

The data set is composed of 3 main components: a fake news title, a title of a new news title, and a label that can take 3 different values: ‘agreed’, ‘disagreed’, and ‘unrelated’. To label this data we asked participants to tell us if they think that the new news title talks about the same fake news as the one that we already have (in that case they label ‘agreed’). If the new newspaper title refutes the fake news then they label ‘disagreed’ and if both titles are unrelated they label the combination of the two as ‘unrelated’. We have two files, one with 256 442 labeled data and another one with 64 111 unlabeled data. So, our main objective was to train some models using the labeled data in order to then test it on the unlabeled data.

By the way, the dataset is very **unbalanced** as we can see on this plot:



Plot 1: Distribution of the different classes

Therefore, in our analysis, we will not use only the accuracy, because in that case it's not very meaningful, but we will look at the F1-score, and in particular the weighted average F1-Score to compare the models and evaluate the performance of the different models.

Then to train the models we had to **pre-process** them using the bag of words approach that we will discuss more in detail in the next part of this report. But before that, we also checked that all the data is in a good format, they were no missing values, etc.

Then we decided to fit 4 different models:

1. **A convolutional neural network** (CNN): This model is used to aim to capture efficiently the local patterns and the spatial hierarchies in the data.
2. **A bidirectional LSTM model** (RNN): This model should help us to capture both forward and backward context and should handle long-dependencies in the text.
3. **A complex multilayer perceptron model** (MLP): This model with its simple architecture, should be faster in training and therefore we could probably add more parameters to it and hope to better fit the data, even if it should struggle to capture all the context in the sequential data.
4. **A transformer model**: This model should be highly efficient at capturing long-range dependencies.

And then instead of comparing each model against each other, we decided to combine them and make them vote (using the soft voting approach), and this in order to create **an ensemble classifier**. And this ensemble classifier, by combining different models, should leverage their individual strengths, leading to more robust and accurate predictions. And should also reduce the likelihood of overfitting (because it's way harder to overfit 4 different models). And therefore, should help to mitigate the weaknesses of each individual model.

Implementation Details

The implementation of all the code was not so difficult.

First, for the **data preprocessing**, they were no missing data and no problem in correctly importing the data in the right type, etc. We also decided to hone-hot encode the labels in order to have something more suitable to feed our networks. And we decided to embed the news titles by applying a **bag-of-word** process on them using all variable vocabulary sizes. We decided to keep all the information in the text and not remove the stop words, etc., because we believe that in fake news detection, **without removing stop words and performing text normalization**, the models retain more information from the original news titles, potentially capturing subtle patterns and complexities that can help in fake news detection. And finally, to weigh the importance of words in the news titles, we employed the Term Frequency-Inverse Document Frequency (**tf-idf**) technique, which highlights both frequently occurring and unique terms.

Then the main difficulty in the implementation of our models was **tuning the hyperparameters** and the **training time**. And especially for the **Bidirectional LSTM** model we used only one Bidirectional LSTM unit in our model and each epoch took approximately 5 minutes to train. The other models were relatively fast to train and therefore easier to tune their hyperparameters: for the **CNN model** 1D convolutional layers with their 1D MaxPooling layers and then a dense layer, for the **MLP model** we used some Dense, Dropout, and Activation layers and for the **Transformer model** we used 2 blocks of Transformers with a number of heads equals to 4, associated to a dense layer.

We used **cross-entropy loss** and **Adam Optimizer** with a **learning rate of 0.001** and **accuracy** as a metric for all the models.

Also **to take advantage of the validation set** (because we divided our training set into training, validation, and testing set with a ratio of 60%, 20%, and 20%), we decided to merge our training and validation data (in order to get a super training set that has 80% of data) and train it using the best set of hyperparameters found during the previous training with the training set (of 60%) and the validation set (20%). And then evaluate our final model trained with the best set of hyperparameters found on the testing set.

We also created functions to **save** and **load** the trained models in order to not have to train them each time we want to use them.

And creating the **ensemble classifier** was straightforward in the sense we used the soft voting approach: we just made a prediction with each one of the classifiers and made the sum of each probability obtained and took the biggest sum as a prediction. Like this, we do what we call **soft voting**.

And finally, we add a cell in our Jupiter Notebook to **predict all the data that need to be predicted in the test CSV file** using our ensemble classifier.

Results and discussion

After training many several models and tuning all the hyperparameters as we can that’s what we got as the final model for each one of the 4 (for a vocabulary size of 1000).

Convolutional neural network:

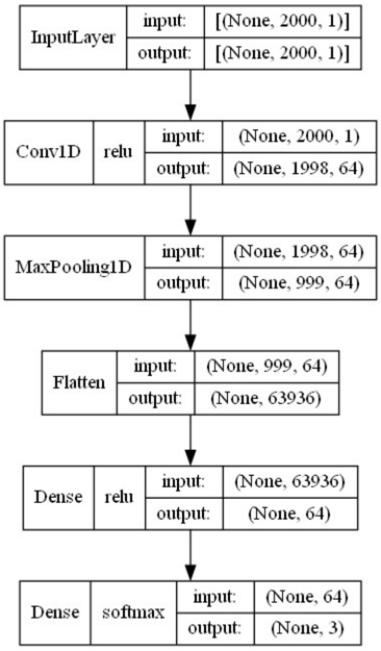

InputLayer | input: | [(None, 2000, 1)]
output: | [(None, 2000, 1)]
Conv1D | relu | input: | (None, 2000, 1)
output: | (None, 1998, 64)
MaxPooling1D | input: | (None, 1998, 64)
output: | (None, 999, 64)
Flatten | input: | (None, 999, 64)
output: | (None, 63936)
Dense | relu | input: | (None, 63936)
output: | (None, 64)
Dense | softmax | input: | (None, 64)
output: | (None, 3)

Image 1: Diagram of the CNN Model

This model contains 4 million parameters and took approximately 10 seconds to train per epoch. We used a batch size of 128. So, this network was fast to train as expected.

Bidirectional LSTM:

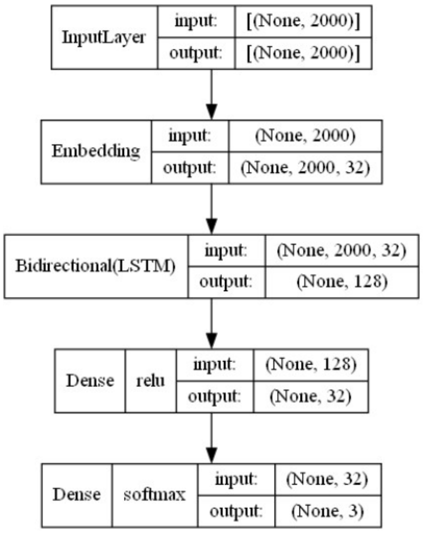
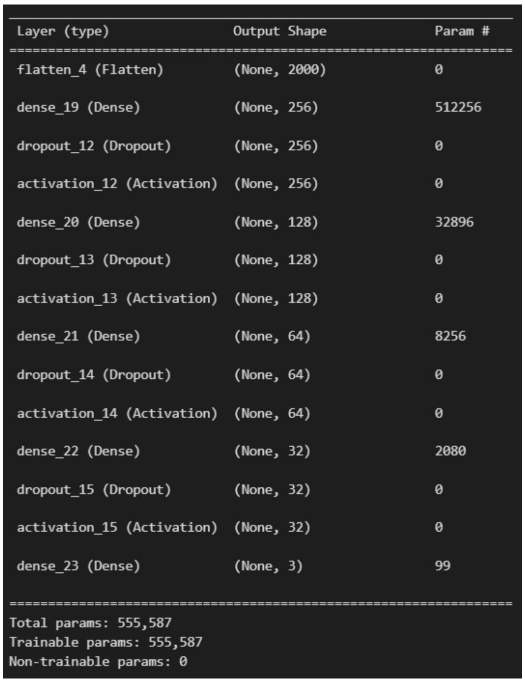

InputLayer | input: | [(None, 2000)]
output: | [(None, 2000)]
Embedding | input: | (None, 2000)
output: | (None, 2000, 32)
Bidirectional(LSTM) | input: | (None, 2000, 32)
output: | (None, 128)
Dense | relu | input: | (None, 128)
output: | (None, 32)
Dense | softmax | input: | (None, 32)
output: | (None, 3)

Image 2: Diagram of the Bidirectional LSTM Model

This model contains 117k parameters and took approximately 300 seconds (5 minutes) to train per epoch. We used a batch size of 128. So, this network was super long to train as expected and even if there is way more parameters to fit.

Multi-Layer Perceptron (MLP):



```
Layer (type)                Output Shape              Param #
=============================================================
flatten_4 (Flatten)         (None, 2000)              0

dense_19 (Dense)            (None, 256)               512256

dropout_12 (Dropout)        (None, 256)               0

activation_12 (Activation)  (None, 256)               0

dense_20 (Dense)            (None, 128)               32896

dropout_13 (Dropout)        (None, 128)               0

activation_13 (Activation)  (None, 128)               0

dense_21 (Dense)            (None, 64)                8256

dropout_14 (Dropout)        (None, 64)                0

activation_14 (Activation)  (None, 64)                0

dense_22 (Dense)            (None, 32)                2080

dropout_15 (Dropout)        (None, 32)                0

activation_15 (Activation)  (None, 32)                0

dense_23 (Dense)            (None, 3)                 99

=============================================================
Total params: 555,587
Trainable params: 555,587
Non-trainable params: 0
```

Image 3: Summary of the MLP Model

This model contains 555k parameters and took approximately 7 seconds to train per epoch. We used a batch size of 128. So, this network was very fast to train and this is expected because it’s a simple model with not so much parameters.

Transformer Model:

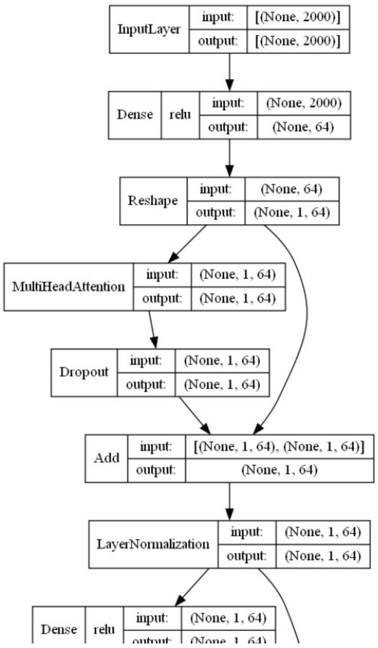
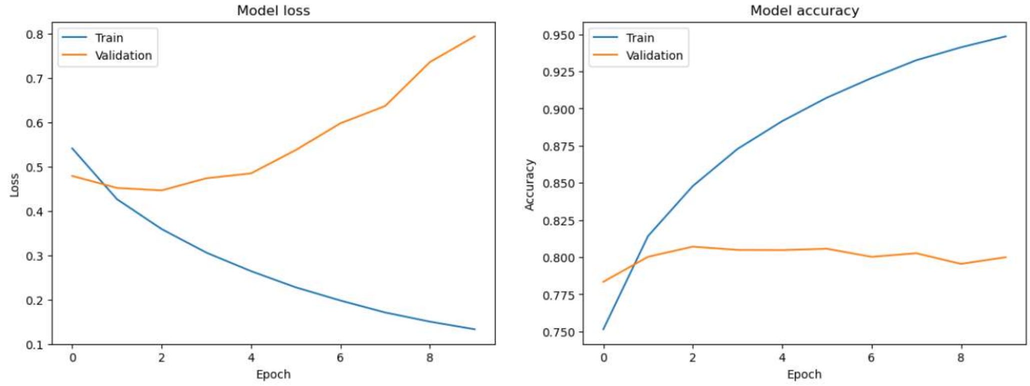


Image 4: Part of the diagram of the Transformer Model

This model contains 280k parameters and took approximately 33 seconds to train per epoch. We used a batch size of 128. So, this network was quite fast to train and this is expected because it’s a way more complex model to train even if it has fewer parameters.

And here is what we got as plots during training (that helped us to decide about the ideal number of epochs without overfitting) (the results below are still with a vocabulary size equal to 1000):

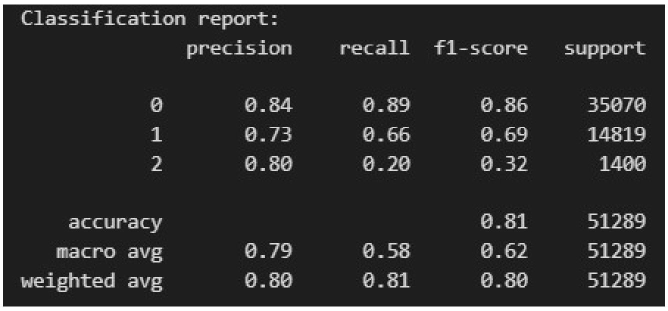
Convolutional neural network:



Graph 1: Accuracy and Loss plot for CNN model

We trained over 10 epochs the model and we noticed that we started to overfit after 3 epochs, so relatively early. we of course tried other structures that add regularization, etc. but nothing really improved. So, then we took this model and fit it with the validation and training set on 3 epochs and got a global accuracy of **80.78%.**

And here are more metrics related to the model on the test set:

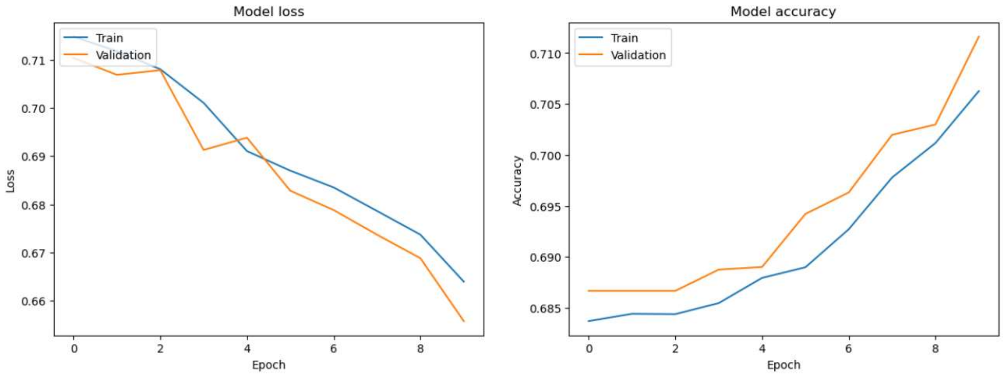


```
Classification report:
              precision    recall  f1-score   support

           0       0.84      0.89      0.86     35070
           1       0.73      0.66      0.69     14819
           2       0.80      0.20      0.32      1400

    accuracy                           0.81     51289
   macro avg       0.79      0.58      0.62     51289
weighted avg       0.80      0.81      0.80     51289
```

Table 1: Classification Report for CNN model

The results are quite satisfying for this model, especially since the training was fast and we got good results for a simple model.

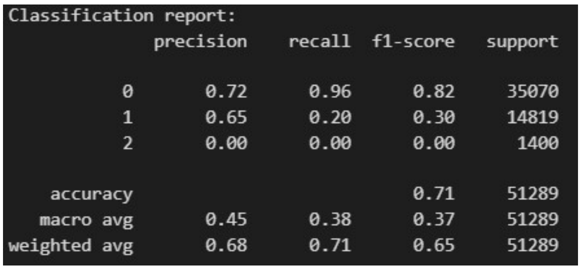
Bidirectional LSTM:



Graph 2: Accuracy and Loss plot for RNN model

We trained over 10 epochs of the model and we noticed that we did not overfit and we are still underfitting. So, we expect to have even better results with more epochs, but we had limited computational resources, so we stopped the training after 10 epochs, especially since training took a very large amount of time, but this was expected because LSTM units are super computationally expensive. So, then we took this model and fit it with the validation and training set on 10 epochs and got a global accuracy of **71.10%.**

And here are more metrics related to the model on the test set:

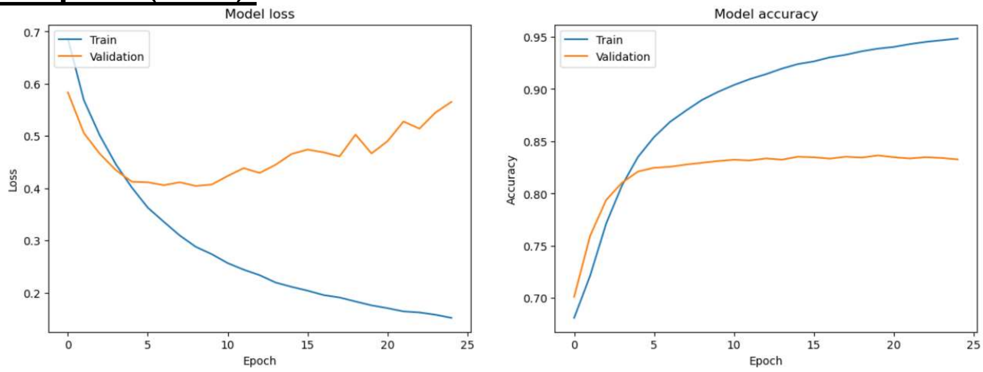


```
Classification report:
              precision    recall  f1-score   support

           0       0.72      0.96      0.82     35070
           1       0.65      0.20      0.30     14819
           2       0.00      0.00      0.00      1400

    accuracy                           0.71     51289
   macro avg       0.45      0.38      0.37     51289
weighted avg       0.68      0.71      0.65     51289
```

Table 2: Classification Report for RNN model

The results are quite bad for this model because we expect to get something way better for an RNN model, but we can understand why, and this is due to our very early stopping of the training due to our limited computational power.

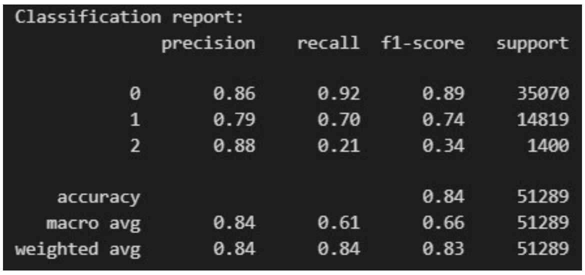
<u>Multi-Layer Perceptron (MLP):</u>



Graph 3: Accuracy and Loss plot for MLP model

We trained over 25 epochs the model and we noticed that we started to overfit after 9 epochs. We added a lot of Dropout layers to overfit more lately in order to have a better model, and this time it worked and that's why we overfit after 9 epochs (initially it was after 2 epochs). So, then we took this model and fit it with the validation and training set on 9 epochs and got a global accuracy of **83.87%.**

And here are more metrics related to the model on the test set:

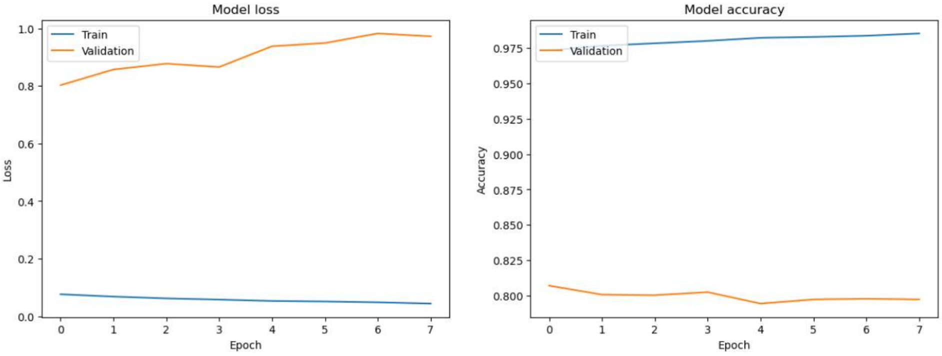


```
Classification report:
              precision    recall  f1-score   support

           0       0.86      0.92      0.89     35070
           1       0.79      0.70      0.74     14819
           2       0.88      0.21      0.34      1400

    accuracy                           0.84     51289
   macro avg       0.84      0.61      0.66     51289
weighted avg       0.84      0.84      0.83     51289
```

Table 3: Classification Report for MLP model

The results are quite satisfying for this model, especially since the training was fast and we got good results for a model with not so many parameters. We can also notice that this model outperforms the other models in terms of the F1 Score and have obviously a better-weighted average.

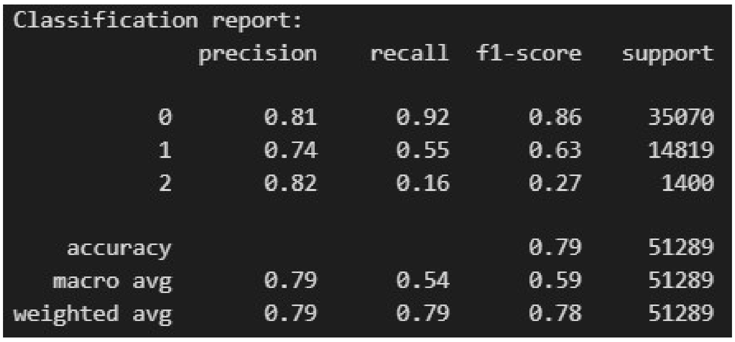
Transformer Model:



Graph 4: Accuracy and Loss plot for Transformer model

We trained over 8 epochs the model and we noticed that we started to overfit after 1 epoch. We tried to add some regularization layers such as Batch Normalization, Dropout, etc., but nothing really improved, we also played with the learning rate, and the same thing nothing avoided this super early overfitting. We believe that this is probably due to the complexity of this model and maybe we need more data to avoid this overfitting. So, at the end, we took this model and fit it with the validation and training set on 1 epoch and got a global accuracy of **79.15%.**

And here are more metrics related to the model on the test set:



```
Classification report:
              precision    recall  f1-score   support

           0       0.81      0.92      0.86     35070
           1       0.74      0.55      0.63     14819
           2       0.82      0.16      0.27      1400

    accuracy                           0.79     51289
   macro avg       0.79      0.54      0.59     51289
weighted avg       0.79      0.79      0.78     51289
```

Table 4: Classification Report for Transformer model

The results are quite satisfying for this model, especially since we overfitted after one epoch and therefore we did not exploit all the potential of this model. We can also notice that this model still has good performance in comparison to the others even if not the best and also have a comparable F1 score in comparison to the others.

Remark: 0 means “Unrated”, 1 means “Agreed” and 2 means “Disagreed”. And we can notice that in all the models we got terrible recall and f1-score for the “Disagreed” category and this is probably due to the sample size in comparison to the other 2 categories.

Now another interesting thing to compare could be the impact of the vocabulary size on the models using the same hyperparameters and that's what we got:

Vocabulary Size	CNN Accuracy	RNN Accuracy	MLP Accuracy	Transformer Accuracy
100	75.41%	**71.48%**	77.31%	73.98%
1000	**80.78%**	71.10%	**83.87%**	**79.15%**

Table 5: Accuracy comparison with different vocabulary sizes

Vocabulary Size	CNN Weighted Avg. F1 Score	RNN Weighted Avg. F1 Score	MLP Weighted Avg. F1 Score	Transformer Weighted Avg. F1 Score
100	0.73	**0.66**	0.75	0.72
1000	**0.80**	0.65	**0.83**	**0.78**

Table 6: Weighted Average F1 Score comparison with different vocabulary sizes

So, we can see that all the models that have 1000 as a vocabulary size outperformed by 5 points (on average) the same model trained with a vocabulary size of 100. We can also notice that increased by 6 points, on average the weighted Average F1 score of the models by using 10 times more vocabulary size. We also noticed a slight speed-up in the training time, but the RNN training time per epoch dropped from 5 minutes to 1 minute.

However, the Bidirectional LSTM model did not perform better, and performed as the other one, and we believe that this is due to the fact that both modes are still at the early stage of training and therefore both are still underfitting the data and could not be considered as a good final version of the training, so we can't really see a difference in their performance.

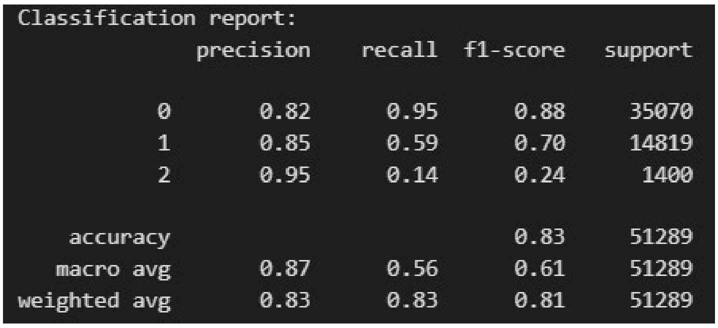
Conclusion:

So, we showed that **increasing the vocabulary size allows models to better capture nuanced patterns in text data**, leading to improved accuracy and F1 score. However, a larger vocabulary size can also increase the computational complexity and memory requirements, potentially affecting training time. So, the **trade-off** between the benefits of a larger vocabulary and the associated computational costs must be carefully considered when designing and training models for text classification tasks. And we were unable to use an even larger vocabulary size for our purpose because we directly face memory issues.

However, **we believe that the relationship between the increase of vocabulary size and the performance of the models is not linear**, and there is likely an optimal vocabulary size for each model beyond which the performance gains diminish or even decrease. Identifying this optimal vocabulary size is an essential part of model selection and design. In practice, the trade-off between model performance, computational complexity, and memory requirements often dictates the choice of vocabulary size, especially when dealing with memory constraints or large-scale datasets.

By the way, we also tried to do some **feature engineering**, by adding some features like "presence or not of exclamation points or interrogation points' (we used this feature for example because we believe that in fake news we try to attract the reader and this kind of elements could attract us to read and believe in this news), etc. by trying to find patterns in fake news titles and therefore help our models to perform better. But we **failed to find some good features** and our models were not improved (improved by 0.001 points, but it's not significant, so we gave up to implement this part of the project because we concluded that it was useless).

Finally, we implemented our **Ensemble Classifier** with soft voting and tested our model, we decided to use all our models for the voting process even if the Bidirectional LSTM model is not as good as the others, but it's still an important model that analyses things that are not analyzed by the other models, and here the results that we got on the testing set: **82.79% of accuracy.**



```
Classification report:
              precision    recall  f1-score   support

           0       0.82      0.95      0.88     35070
           1       0.85      0.59      0.70     14819
           2       0.95      0.14      0.24      1400

    accuracy                           0.83     51289
   macro avg       0.87      0.56      0.61     51289
weighted avg       0.83      0.83      0.81     51289
```

Table 6: Classification Report for Ensemble Classifier

So, it's almost 1 point less than our model MLP took individually for accuracy and 2 points less for the weighted average F1-Score than the MLP model.

Remark: 0 means "Unrated", 1 means "Agreed" and 2 means "Disagreed". And we can notice that in all the models we got terrible recall and f1-score for the "Disagreed" category and this is probably due to the sample size in comparison to the other 2 categories.

Conclusion:

We could then think that we should take the MLP model to make a prediction because have better performance metrics overall. However, the Ensemble Classifier still offers some advantages. By incorporating the strengths of all four models, including the Bidirectional LSTM, Transformers, and CNN models, the ensemble approach can provide better generalization and robustness against unseen data, making it a viable option even with slightly lower accuracy and weighted average F1-Score compared to the best individual model.

GENERAL CONCLUSION:

In conclusion, this project aimed to develop a fake news detection system using deep learning models like CNN, Bidirectional LSTM, MLP, and Transformer. We explored the impact of vocabulary size on performance and created an Ensemble Classifier to leverage the strengths of each model. Although the MLP model showed the best individual performance, the Ensemble Classifier provided a more robust and generalizable solution. Future work could involve exploring additional model architectures, data preprocessing, optimizing hyperparameters, and investigating advanced feature engineering techniques to further enhance the performance of the fake news detection system.

However, through this project, we learned the importance of selecting the right model architectures, handling different vocabulary sizes, and combining multiple models to improve overall performance. This experience demonstrates the value of exploring various techniques in building effective and robust solutions for real-world challenges like fake news detection.

References

We used the **TensorFlow** library along with some of our **class materials** and the **Stackoverflow** website to help us to do this project.