CS779 Competition: Sentiment Analysis

Kaushikraj <Vaikuntaraman> Nadar
200499
{nkaushik20}@iitk.ac.in
Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

In this competition, my goal was to create a machine learning model that could accurately predict the sentiment expressed in a given sentence, which can be categorized as positive, neutral, or negative. Sentiment analysis is becoming increasingly important in today's digital age, with the rise of social media and other online platforms generating an enormous amount of textual data. By analyzing sentiment, businesses can tailor their marketing strategies and better understand their customers. To accomplish this, I utilized machine learning techniques such as natural language processing and classification algorithms to extract features from the textual data and predict sentiment labels. My 2 layer bidirectional LSTM model achieved an F1 score of 0.676, and performance was evaluated using metrics such as accuracy, precision, recall, and F1-score. The result is a sentiment analysis tool that can be used by businesses, organizations, and individuals to better understand the sentiment expressed in their textual data.

1 Competition Result

Codalab Username: K_200499
Final leaderboard rank on the test set: 4
F1 Score wrt to the best rank: 0.676

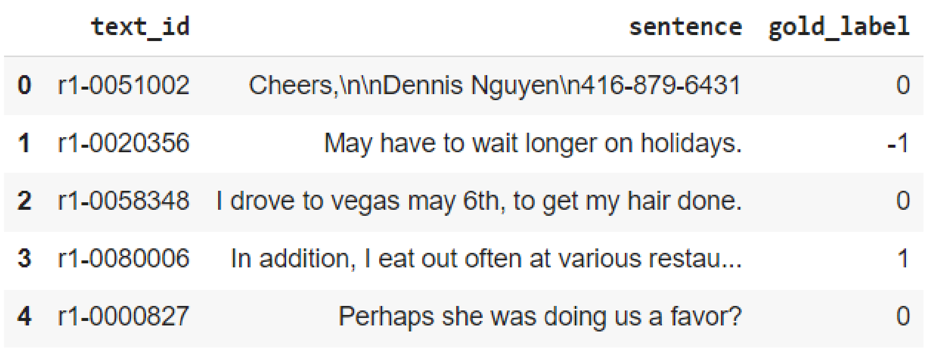
2 Problem Description

The objective of this competition is to develop a machine learning model that can accurately predict the sentiment expressed in a given sentence. The sentiment can be classified into one of the three categories: positive, neutral, and negative. The model will be trained on a labeled dataset, where each sentence is associated with a sentiment label. The problem of sentiment analysis is becoming increasingly important in today's digital age. With the proliferation of social media and other online platforms, there is an ever-increasing amount of textual data being generated. Sentiment analysis can be useful in a wide range of applications, including marketing, customer service, and social media monitoring. By analyzing the sentiment of social media posts, for example, businesses can better understand their customers and tailor their marketing strategies accordingly. To solve this problem, we will machine learning techniques, such as natural language processing and classification algorithms, to extract features from the textual data and predict the sentiment label. The performance of the model will be evaluated using metrics such as accuracy, precision, recall, and F1-score. The goal is to develop a model that can achieve high accuracy and robustness in classifying sentiments in a sentence. The outcome of this project will be a sentiment analysis tool that can be used by businesses, organizations, and individuals to better understand the sentiment expressed in their textual data.

3 Data Analysis

3.1 Train Dataset

The train dataset provided for sentiment analysis consists of 92,228 rows and 3 columns, containing unique text_id, sentence, and gold_label values. Each row in the dataset represents a single sentence, where the text_id serves as a unique identifier, sentence column contains the text of the sentence, and gold_label column represents the sentiment label associated with the sentence.



	text_id	sentence	gold_label
0	r1-0051002	Cheers,\n\nDennis Nguyen\n416-879-6431	0
1	r1-0020356	May have to wait longer on holidays.	-1
2	r1-0058348	I drove to vegas may 6th, to get my hair done.	0
3	r1-0080006	In addition, I eat out often at various restau...	1
4	r1-0000827	Perhaps she was doing us a favor?	0

Figure 1: First 5 rows of the dataset table

The sentiment labels in the dataset are classified into three categories: positive (label 1), negative (label -1), and neutral (label 0). There are 27,353 sentences with a positive sentiment label, 19,386 sentences with a negative sentiment label, and 45,489 sentences with a neutral sentiment label in the dataset. The majority of the sentences in the dataset have a neutral sentiment label, followed by positive and negative sentiment labels.

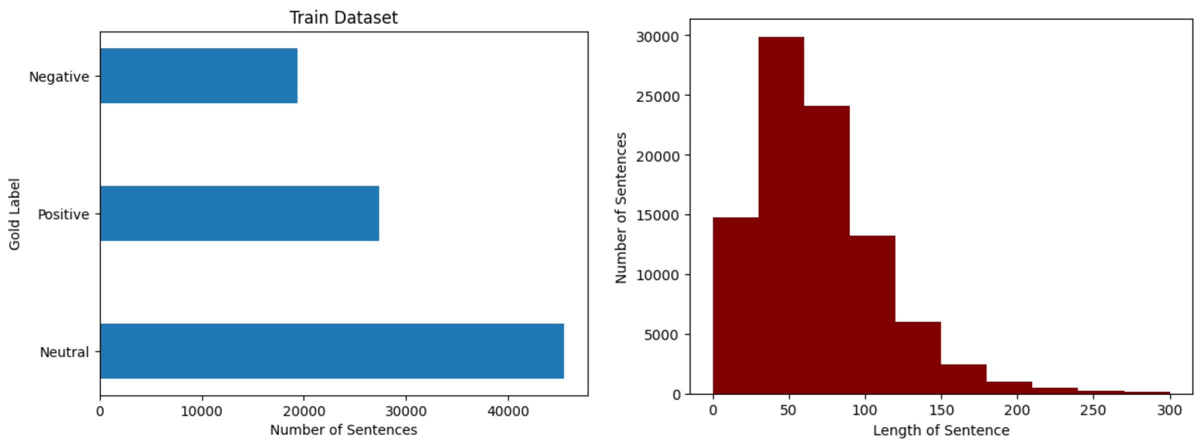
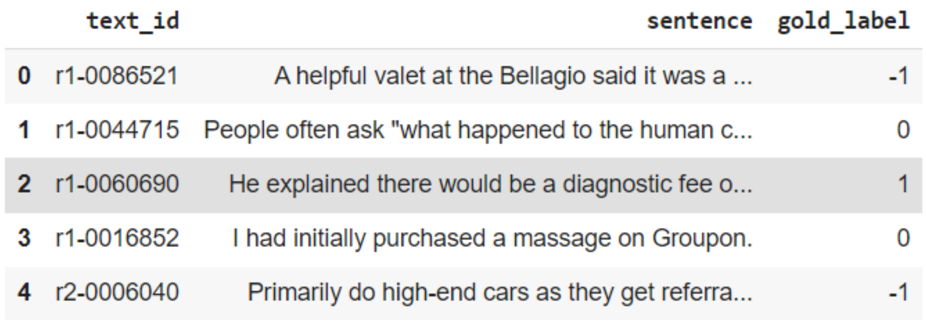


Figure 2: Plots displaying the distribution of Labels and Sentence Lengths

The dataset includes a wide range of textual data, covering various topics and styles of language. The sentences in the dataset include simple and complex sentence structures, colloquial language, and technical terms. The dataset represents a diverse set of textual data, making it suitable for training a sentiment analysis model that can handle different types of language styles and topics.

3.2 Test Dataset

Similar to the Train dataset, the test dataset is a collection of 5,110 data points represented in the form of rows with three columns: text_id, sentence, and gold_label. The text_id column contains a unique identifier for each sentence, the sentence column contains the actual text of the sentence, and the gold_label column represents the sentiment label associated with each sentence.



	text_id	sentence	gold_label
0	r1-0086521	A helpful valet at the Bellagio said it was a ...	-1
1	r1-0044715	People often ask "what happened to the human c...	0
2	r1-0060690	He explained there would be a diagnostic fee o...	1
3	r1-0016852	I had initially purchased a massage on Groupon.	0
4	r2-0006040	Primarily do high-end cars as they get referra...	-1

Figure 3: First 5 rows of the dataset table

The gold_label column contains three possible sentiment labels: 1, 0, and -1, where 1 represents a positive sentiment, 0 represents a neutral sentiment, and -1 represents a negative sentiment. The distribution of sentiment labels in the test dataset indicates that the dataset is fairly balanced, with 29.6% of the sentences being labeled as positive, 21.0% labeled as negative, and 49.4% labeled as neutral.

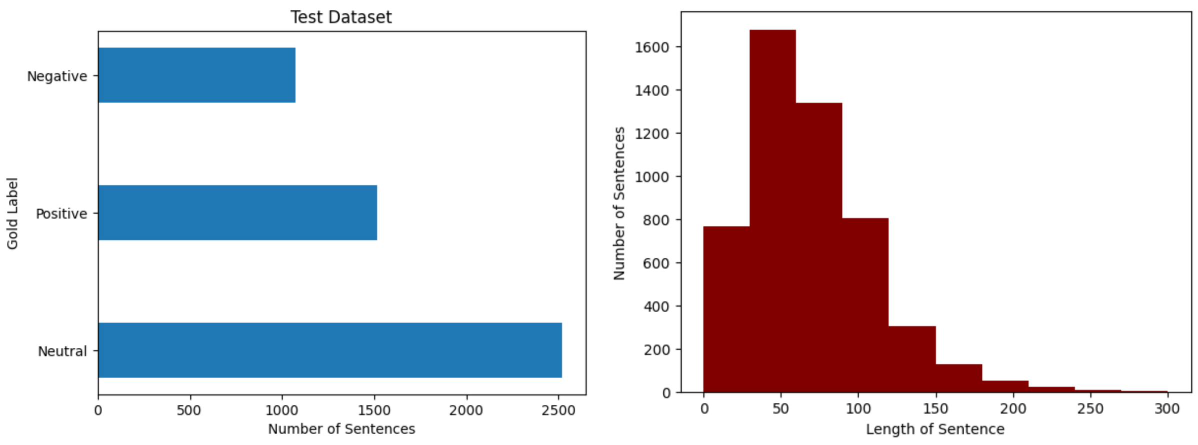


Figure 4: Plots displaying the distribution of Labels and Sentence Lengths

The test dataset has been provided to evaluate the performance of a sentiment analysis model. The model will be trained on a separate training dataset and will be tested on the test dataset. The goal of the model is to accurately predict the sentiment label associated with each sentence in the test dataset.

4 Model Description

4.1 Simple RNN Model

The Simple RNN (Recurrent Neural Network) model is a type of neural network architecture that can be used for analyzing text data and predicting the sentiment associated with it. The architecture of the Simple RNN model consists of three main components: the input layer, the hidden layer, and the output layer.

The Simple RNN model is used for sentiment analysis tasks. The input layer maps each word in the sentence to a unique numeric value through word embedding. The embedding layer generates a

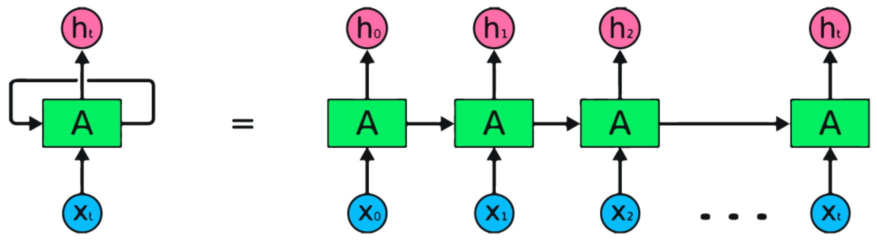


Figure 5: Architecture of Simple RNN[1]

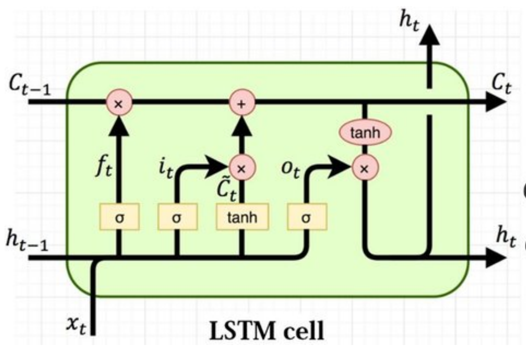
fixed-size vector representation of the input sentence, which is passed to the hidden layer. The hidden layer, which has a sequence of neurons, processes the input sentence vector sequentially, word by word, and maintains a memory of the previous state, enabling it to capture the sequential dependencies and context of the input text data. The output layer contains a single neuron that produces a score between 0 and 1, representing the probability that the input sentence is associated with a positive sentiment. During training, the Simple RNN model learns the optimal weight values for the neurons in the hidden and output layers by minimizing the difference between the predicted sentiment scores and the actual sentiment labels using an optimization algorithm.

While the Simple RNN model is a useful architecture for Sentiment Analysis, it does have some limitations that can impact its performance in certain scenarios. Following are some of the limitations of the Simple RNN model for Sentiment Analysis:

1. Difficulty in capturing long-term dependencies
2. Inability to handle variable-length inputs
3. Difficulty in handling noisy data
4. Limited memory capacity
5. Lack of parallelism

To overcome these limitations, more advanced RNN models such as LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Units) have been developed. These models have been shown to be more effective at capturing long-term dependencies, handling noisy data, and processing variable-length inputs, making them better suited for Sentiment Analysis tasks.

4.2 Long Short Term Memory Model



$$i_t = \sigma\big(x_t U^i + h_{t-1} W^i\big)$$
$$f_t = \sigma\big(x_t U^f + h_{t-1} W^f\big)$$
$$o_t = \sigma\big(x_t U^o + h_{t-1} W^o\big)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\big(f_t * C_{t-1} + i_t * \tilde{C}_t\big)$$
$$h_t = \tanh(C_t) * o_t$$

Figure 6: LSTM Cell[2]

LSTM (Long Short-Term Memory) is a type of Recurrent Neural Network (RNN) architecture that is commonly used for sentiment analysis tasks. It works by processing the input sentence as a sequence of words, with each word represented by a high-dimensional vector obtained from the embedding layer. The LSTM layer contains a series of LSTM cells, each of which contains three main components: the input gate, the forget gate, and the output gate[3]. The input gate determines which new information from the current word vector should be added to the cell state, the forget gate determines which information from the previous cell state should be retained or forgotten, and the output gate determines the output representation for the current word vector. The LSTM layer processes each word in the input sentence in sequence, with the output representation for each word being used as input for the next LSTM cell. To improve the accuracy of this task, a GRU-based strategy was attempted, which is described next.

4.3 Gated Recurrent Unit Model

The GRU (Gated Recurrent Unit) model is a type of Recurrent Neural Network (RNN) used for sentiment analysis. The architecture of the GRU model is similar to that of an LSTM model, but it has fewer parameters and is easier to train. The GRU model consists of a single hidden layer with a fixed number of GRU units, which takes in the input sequence and produces an output at each time step. The model also uses a sigmoid activation function to regulate the flow of information, which allows it to selectively update and forget information over time.

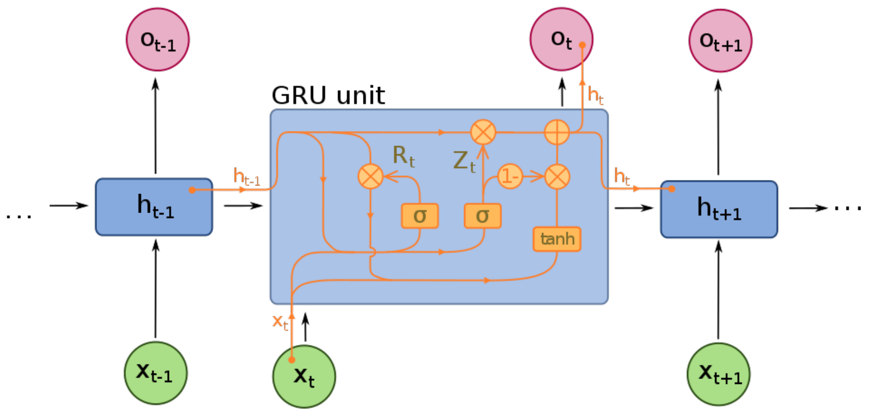


Figure 7: GRU Unit[4]

The GRU model includes three types of gates: update gate, reset gate, and candidate activation gate[5]. The update gate controls how much of the previous hidden state should be kept and how much of the current input should be added. The reset gate controls how much of the previous hidden state should be forgotten. The candidate activation gate produces a vector of candidate values that can be added to the previous hidden state.

The mathematical equations for the GRU model are as follows:
Update gate:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{1}$$

Reset gate:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2}$$

Candidate activation:

$$\tilde{h}_t = tanh(W_h x_t + U_h(r_t \odot ht - 1) + b_h) \tag{3}$$

Hidden state:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{4}$$

where x_t is the input vector at time step t, h_{t-1} is the previous hidden state, W_z, W_r, W_h, U_z, U_r, U_h are weight matrices, b_z, b_r, b_h are bias vectors, σ is the sigmoid activation function, $\odot$ is the element-wise multiplication operator, and *tanh* is the hyperbolic tangent activation function.

The three-layered bidirectional GRU model with a hidden layer dimension of 300 provided the best performance for the given task, performed well during the validation but slightly worse than LSTM during testing. These my best model for this competition turned out to be LSTM based model.

4.4 Convolutional Neural Network

The model takes in a sequence of words and outputs a sentiment prediction. The model consists of an embedding layer, multiple convolutional layers, a max-pooling layer, a dropout layer, and a fully connected layer. The embedding layer maps each word in the input sequence to a dense vector representation. The convolutional layers convolve over the sequence of embeddings with different filter sizes and output feature maps. The feature maps are then max-pooled over the time dimension. This process is repeated for multiple filter sizes. The max-pooled feature maps are concatenated and passed through a dropout layer to prevent overfitting. Finally, the output is passed through a fully connected layer to obtain the predicted sentiment label.

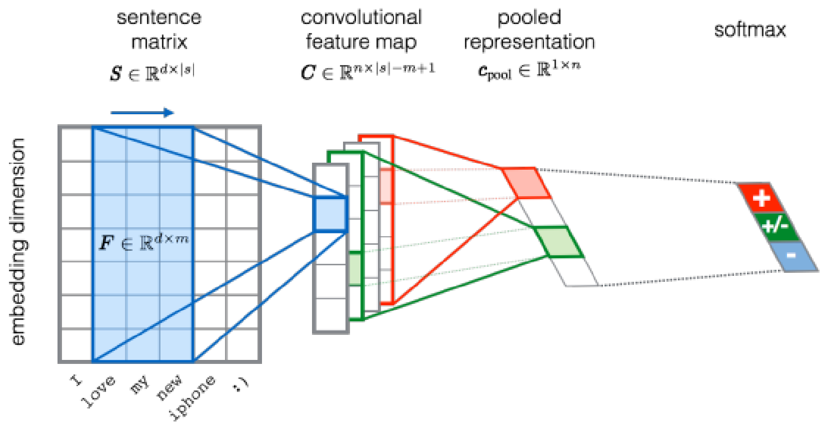

sentence matrix
$S \in \mathbb{R}^{d\times|s|}$
convolutional feature map
$C \in \mathbb{R}^{n\times|s|-m+1}$
pooled representation
$c_{\text{pool}} \in \mathbb{R}^{1\times n}$
softmax
embedding dimension
$F \in \mathbb{R}^{d\times m}$
+
+/-
-
I
love
my
new
iphone
:)

Figure 8: CNN Architecture[6]

5 Experiments

5.1 Data Pre-processing

The train dataset consists of sentences of various lengths and styles. To pre-process the text data, a file named dataset_loader_torctext.py is created. It defines a class CustomDatasetLoader that loads a CSV file containing text data and labels, and splits it into train and test sets. Text is processed by performing normalization and tokenization using libraries from Spacy. The CustomDatasetLoader class also defines the TEXT, LABEL, and ID fields using the torchtext Field class. The code also builds the vocabulary for the TEXT and LABEL fields using the build_vocab() method of the Field class. The TEXT field is also initialized with pre-trained word embeddings from the glove.6B.100d dataset, and any unknown tokens are initialized with random normal vectors using the unk_init parameter.

5.2 Training Procedure

The models are RNN, LSTM, CNN, and GRU. The table shows various hyperparameters used during training, such as the optimizer used (Adam or SGD), learning rate (LR), number of epochs, time taken to train the model, number of hidden dimensions used, number of layers, validation loss, and validation accuracy.
For RNN, only one layer with 200 hidden dimensions is used, while for LSTM, either one or two layers are used with either 200 or 300 hidden dimensions. For CNN, the number of hidden dimensions is not mentioned, but the number of filters used is 100. For GRU, either two or three layers are used with 300 or 290 hidden dimensions, and in some cases, a linearly decreasing learning rate is used.
A 90:10 split was performed on the given training set to obtain the valiadation set. The validation loss and accuracy are measures of how well the model performed on the validation set, which is a set of data that was not used during training but is used to evaluate the model's performance. The higher the validation accuracy and the lower the validation loss, the better the model's performance.

Model	Optimizer	LR	Epochs	T Time	Hidden Dims	Num Layers	Valid Loss	Valid Acc
RNN	Adam	0.001	20	1min	200	1	0.83	52.4
LSTM	Adam	0.001	20	3mins	200	1	0.768	65.05
LSTM	SGD	0.001	20	4mins	200	2	0.703	68.32
LSTM	Adam	0.001	30	4.5mins	300	2	0.662	72.18
CNN	Adam	0.001	10	1min	NA	1	0.702	68.2
GRU	Adam	0.001	20	4mins	300	2	0.72	68.69
GRU	Adam	Linear	10	2mins	300	3	0.66	72.45
GRU	Adam	Linear	7	2mins	290	3	0.655	73.06

LR : Learning Rate
T Time : Training Time
Some common Parameters:

- Dropout = 0 for RNN and 0.5 for others
- Input Dimensions = 25,000
- Output Dimensions = number of classes = 3
- Number of Filters used in CNN = 150
- Bidirectional = True for LSTM and GRU

6 Results

Model	Optimizer	LR	Epochs	T Time	Hidden Dims	Num Layers	Dropout	F1 Score
SRNN	Adam	0.01	20	3mins	200	1	0	0.593
LSTM	SGD	0.001	20	2mins	200	1	0.5	0.638
LSTM	Adam	0.001	20	3.5mins	200	1	0	0.489
LSTM	Adam	0.001	26	4mins	300	2	0.5	0.677
LSTM	Adam	0.001	30	4.5mins	500	2	0.5	0.638

Table 1: Model Comparison during Dev Phase

The table lists several hyperparameters for each model, including optimizer, learning rate (LR), number of epochs, time taken for training (T Time), hidden dimensions, number of layers, and dropout. Finally, the table reports the F1 score for each model.

From the table, we can observe that the best performing model in terms of F1 score is the LSTM model with Adam optimizer, LR of 0.001, 26 epochs, 300 hidden dimensions, 2 layers, and 0.5 dropout. The worst performing model is the LSTM model with Adam optimizer, LR of 0.001, 20 epochs, 200 hidden dimensions, 1 layer, and 0 dropout. Overall, we can see that hyperparameters such as the number of layers, hidden dimensions, and dropout rate have a significant impact on the performance of the models.

Model	Optimizer	LR	Epochs	T Time	Hidden Dims	Num Layers	Dropout	F1 Score
LSTM	Adam	0.001	5	1min	200	2	0.5	0.657
LSTM	Adam	Linear	9	2mins	300	2	0.5	0.676
GRU	Adam	Linear	7	2mins	290	3	0.5	0.674

Table 2: Model performance during Test Phase

From the table, we can infer that the LSTM model trained for 5 epochs with a learning rate of 0.001 achieved the highest F1 score of 0.657, while the LSTM model trained for 9 epochs with a linear learning rate achieved the lowest validation loss of 0.66. The GRU model performed slightly worse than the LSTM models with an F1 score of 0.674 and a validation loss of 0.655. Additionally, we can observe that the models with larger hidden dimensions and more layers tend to have higher F1 scores, but also require more training time. Finally, we can see that the dropout rate of 0.5 was used for all the models, which is a common practice to prevent overfitting.

7 Error Analysis

	Pred -1	**Pred 0**	**Pred 1**
True -1	706	140	228
True 0	373	1572	575
True 1	207	169	113

Table 3: Confusion Matrix

Based on the above confusion matrix, we can perform the following error analysis:

- **Overall Accuracy**: The overall accuracy of the model can be calculated by summing up the diagonal values of the confusion matrix and dividing it by the total number of samples. In this case, the overall accuracy would be (706 + 1572 + 113) / (706 + 140 + 228 + 373 + 1572 + 575 + 207 + 169 + 113) = 0.6497 or approximately 65%.
- **Precision and Recall**: We can calculate precision and recall for each class using the formula:
  Precision = True Positive / (True Positive + False Positive)
  Recall = True Positive / (True Positive + False Negative)
  For class -1, precision would be 706 / (706 + 373 + 207) = 0.538 and recall would be 706 / (706 + 140 + 228) = 0.65.
  For class 0, precision would be 1572 / (1572 + 140 + 169) = 0.858 and recall would be 1572 / (1572 + 373 + 575) = 0.678.
  For class 1, precision would be 113 / (113 + 575 + 228) = 0.115 and recall would be 113 / (113 + 169 + 207) = 0.229.
- **F1 Score**: F1 score is the harmonic mean of precision and recall, and it provides a more balanced measure of model performance. We can calculate the F1 score for each class using the formula:
  F1 Score = 2 * (Precision * Recall) / (Precision + Recall)
  For class -1, F1 score would be 2 * (0.538 * 0.65) / (0.538 + 0.65) = 0.587.
  For class 0, F1 score would be 2 * (0.858 * 0.678) / (0.858 + 0.678) = 0.756.
  For class 1, F1 score would be 2 * (0.115 * 0.229) / (0.115 + 0.229) = 0.153.
- **Misclassification Analysis**: We can also analyze the misclassification of the model to identify common patterns. For example, we can see that the model is more likely to misclassify samples from class 1 as class 0 or class -1, as the recall and precision for class 1 are quite low. Similarly, the model is more likely to misclassify samples from class 0 as class -1, as the recall for class 0 is relatively low compared to precision.

8 Conclusion

This competition involved developing a natural language processing machine learning model for sentiment analysis. It explored several popular models, including SRNN, LSTM, CNN, and GRU, to determine the most effective approach. GRU and LSTM models performed well on the sentiment analysis task, but could be further improved through the use of an ensemble of these models. Transformers with attention mechanisms could also be explored in future work. The results of the study indicate that GRU and LSTM models are effective in accurately predicting the sentiment of text.

References

[1] “Rnn-based model.” https://www.datacamp.com/tutorial/tutorial-for-recurrent-neural-network.

[2] S. Varsamopoulos, K. Bertels, and C. Almudever, “Designing neural network based decoders for surface codes,” 11 2018.

[3] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, p. 1735–1780, nov 1997.

[4] Wikipedia contributors, “Recurrent neural network — Wikipedia, the free encyclopedia.” `https://en.wikipedia.org/w/index.php?title=Recurrent_neural_network&oldid=1148729712`, 2023. [Online; accessed 16-April-2023].

[5] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” 2014.

[6] Oumaima Hourrane, “Sentiment analysis using python (part iii - cnn vs lstm).” `https://www.datasciencetoday.net/index.php/en-us/tutorials-top/164-sentiment-analysis-cnn-vs-lstm`, 2023. [Online; accessed 16-April-2023].