CS779 Competition: Sentiment Analysis

Prakriti Prasad
230770
prakritip23@iitk.ac.in

Indian Institute of Technology Kanpur (IIT Kanpur)

Abstract

In recent years, deep learning models have revolutionized many fields including sentiment analysis by enabling more accurate understanding of textual data. So this report is an attempt to use these powerful deep learning models such as CNN, LSTM and their hybrid to solve Sentiment classification problem with three classes(negative, neutral, positive) combined with a One-vs-Rest (OvR) approach. The training data, was cleaned and analyzed to understand its structure and noise levels. CNNs were used to extract features and LSTMs to capture sequential patterns, along with BCEWithLogitsLoss for binary classification in the OvR setup. The final model performed well, achieving an F1-score of 0.61 on the test set and dev set both, handling imbalanced data and sentiment variability effectively.

1 Competition Result

Codalab Username: P_230770
Final leaderboard rank on the test set: 19
F1 Score wrt to the best rank: 0.61
Best F1 score on the leaderboard was 0.70

2 Problem Description

The goal of this competition is to solve a sentiment analysis problem by classifying textual data into one of three sentiment classes: negative , neutral , or positive .

Sentiment analysis is a fundamental task in NLP, and involves determining the emotional tone or opinion expressed in text. It has applications in diverse domains such as customer feedback analysis, social media monitoring, and product review interpretation, making it a practical and impactful problem to solve. The task poses several challenges, including handling ambiguous sentiments where weakly positive or negative sentences overlap with neutral ones.Moreover, achieving good generalization on test data is challenging, especially when the vocabulary overlap between the training and test datasets is limited.

The objective is to build a robust machine learning model that can accurately predict sentiment classes while addressing these challenges.

3 Data Analysis

3.1 Training Dataset Description

The training dataset consists of 92228 samples, each containing a unique text_id, a sentence, and a sentiment label (‘gold_label‘). The sentiment labels are represented as:

- +1: Positive Sentiment
- 0: Neutral Sentiment
- −1: Negative Sentiment

The class distribution in the training dataset is shown in Figure 1, where we can see that the dataset is slightly imbalanced with 0.296% of samples having positive sentiment, 0.493% neutral, and 0.210% negative.

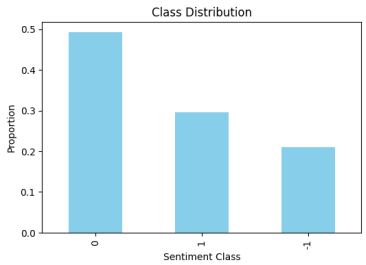


Figure 1: Class Distribution in Training Data

The average sentence length (i.e. no of words in a sentence) in the training data is 13.23_{avg} words, with a minimum of 1_{min} and a maximum of 258_{max}.

3.2 Test Dataset Analysis

The test dataset consists of 5110 samples with similar format as the training data. The vocabulary overlap between the two datasets is 20.51%.The average sentence length (i.e. no of words in a sentence) in the training data is 13.31_{avg} words, with a minimum of 1_{min} and a maximum of 143_{max}. The test corpus contains a total of 68016 words, out of which 12029 are unique.

3.3 Corpus Statistics and Noise Analysis

The training corpus contains a total of 1221019 words, out of which 76488 are unique. The top 5 most frequent words in the dataset excluding stopwords are

Word	Frequency
get	3919
like	3630
place	3509
would	3329
one	3070

Table 1: Top 5 Most Frequent Words in the Training Corpus

Word	Frequency
get	216
like	208
time	169
would	163
one	163

Table 2: Top 5 Most Frequent Words in the Test Corpus

Noise in the corpus includes:

- Special characters like , #, $, %, ; &, are present in 158_{noisy} sentences.
- Duplicate sentences identified in $14_{\text{duplicates}}$ samples.

3.4 Interesting Insights of Training dataset

Top Bigrams Per Class

An analysis of the top bigrams in the dataset reveals patterns that are distinct for each sentiment class.

Table 3: Top 10 Bigrams for Each Sentiment Class

Class	Bigram	Frequency
0 (Neutral)	decided try	115
	first time	370
	las vegas	226
-1 (Negative)	customer service	91
	even though	69
	feel like	61
1 (Positive)	best part	92
	come back	97
	customer service	186

(a) Training Dataset (b) Test Dataset

Figure 2: Figure presents a word cloud visualization of the most frequent words in the dataset.

Polarity Distribution

Neutral sentences (class 0) are heavily concentrated around 0, as expected, while positive (class 1) and negative (class -1) sentences show broader distributions with significant overlap near 0, indicating that there is widespread existence of ambiguous or weakly polarized sentiments in the Dataset.

Some notable insights from the data analysis:

- Sentences with positive sentiment tend to be longer on average than neutral or negative sentiment sentences.
- The top bigrams for both positive and negative sentiment include phrases like *"customer service"*.
- Noise in the dataset, such as special characters and emails, occurs more frequently in neutral sentiment samples.

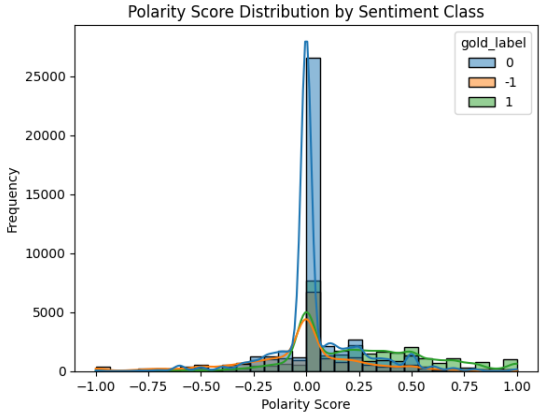


Figure 3: This plot represents the Polarity Score Distribution by sentiment class in training dataset.

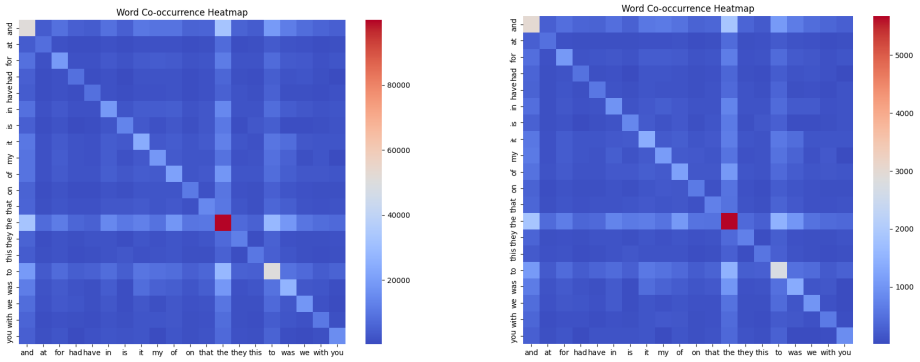
4 Model Description

This problem required the evolution of different models, each with its own distinct advantages and disadvantages. Some models handled issues like ambiguous sentiments, noisy data, class imbalance better than the others.

The initial approach was a hybrid CNN + LSTM model. The convolutional layers helped in extracting high level features from the text. The LSTM is used for short and long term memory, thus remembering the dependencies of the words with respect to the sentiments. This model performed reasonably well. One of its shortcoming was differentiating weakly polarized sentiments from neutral sentiment.

Next, an attention mechanism was incorporated after the CNN-LSTM architecture to better focus on important words contributing to sentiment. This CNN + LSTM + Attention model was better in terms of handling long and complex sentences. However, it did not significantly boost overall performance as most of the sentences in the dataset were short. In fact its F1 score even decreased a bit.

To handle class imbalance more effectively, the CNN + LSTM with One-vs-Rest (OvR) approach was implemented, using BCEWithLogitsLoss as the loss function. [1] Here each class was treated as an independent binary classification task, which allowed the model to focus on learning distinctive features for each class without being influenced by the others and to perform better, especially on the minority classes. This model performed the best on the test set. **Refer to figure 5**
The model starts by taking a tokenized input and converting it to dense numerical vectors using glove. The embeddings are then passed into three separate CNN layers of different kernel sizes(for eg. (1,3,5)), so as to extract features at varying n-gram levels. After each convolution, ReLU activation introduces non-linearity. Max pooling is applied along the sequence length dimension to select the most significant features from each filter output. The output is then concatenated into a single vector feature which is passed into bi directional LSTM. The bidirectional nature of the LSTM ensures that the text is processed in both forward and backward directions. This helps the model understand the sequential relationships



(a) Training Dataset (b) Test Dataset

Figure 4: Word co-occurrence heatmaps for the training and test datasets. The heatmaps visualize the frequency of word pair co-occurrence in each dataset.

and word dependencies in the text. The output from the LSTM is sent to three separate dense layers, one for each sentiment class: -1, 0 and 1. Each dense layer computes the probability for corresponding class in OVR approach. Finally the class with highest probability is chosen.

Another experiment was reversing the sequence of CNN and LSTM layers (LSTM then CNN). Additionally, a multi-head attention mechanism was applied to the LSTM outputs to enable the model to focus on multiple aspects of the sentence simultaneously.

5 Experiments

5.1 Data Pre-processing

Careful preprocessing was used to clean the data and ready it for modelling.

Text sentences were first cleaned by removing urls, HTML tags, except punctuation marks that might hold sentiment relevance. All text was converted to lowercase, and common stopwords (e.g., “the,” “and”) were removed. Special characters (like @, #) were removed unless they added sentiment meaning (e.g., exclamation marks). Words were reduced to their base forms to avoid redundancy in vocabulary, and sequences were padded or truncated to a fixed length. Sentences were then tokenized using the spaCy NLP pipeline, excluding components such as Named Entity Recognition (NER) and the dependency parser. Pre-trained GloVe embeddings were used to initialize the embedding layer for meaningful word representations.

These preprocessing steps ensured that the model could focus on meaningful textual patterns while avoiding noise in the dataset.

5.2 Training Procedure

The training procedure involved experimenting with various configurations of optimizers, learning rates, and epochs across different models.

The **optim.adam optimizer** was used due to its efficient handling of sparse gradients and adaptability. Learning rates were tuned within the range of 10^{-4} to 10^{-2}, with 10^{-3} performing best for most configurations. Models were trained for a maximum of **10 epochs** with early stopping to prevent overfitting, and batch sizes of **256** were used.

Training time varied based on the complexity of the architecture:

- **CNN → LSTM:** ∼20 minutes per epoch.
- **CNN → LSTM + Attention:** ∼25 minutes per epoch.
- **CNN → LSTM with OvR:** ∼20 minutes per epoch.

The **BCEWithLogitsLoss** function was used for the OvR strategy. For others **Cross-Entropy loss** was used for other models.

5.3 Hyperparameter Tuning

Hyperparameters were optimized using **Optuna**, an automated hyperparameter optimization framework. Optuna conducted around 100 trials per model to arrive at the optimal configurations.

Table 4: Hyperparameter Tuning Results for the Final Model (CNN → LSTM with OvR)

Hyperparameter	**Range**	**Best Value**
Learning Rate	10^{-4} to 10^{-2}	10^{-3}
Dropout Rate	0.1 to 0.5	0.3
CNN Filters	64, 128, 256	128
Kernel Sizes (CNN)	(1, 3, 5) or (2, 4, 6)	(1, 3, 5)
LSTM Hidden Dimension	128, 256, 512	256
Number of LSTM Layers	1, 2, 3	2

Table 5: Hyperparameter Tuning Results for the CNN + LSTM + attention model)

Hyperparameter	**Range**	**Best Value**
Learning Rate	10^{-3} to 10^{-2}	10^{-3}
Dropout Rate	0.2 to 0.5	0.3
CNN Filters	50 to 200	98
Kernel Sizes (CNN)	(1, 3, 5) or (2, 4, 6) or (2,3,4)	(2, 3, 4)
LSTM Hidden Dimension	64 to 256	203
Number of LSTM Layers	1, 2, 3	1

6 Results

Table 6 shows the performance of different models on the development set across various phases. Metrics like accuracy, F1 score, precision, and recall are reported as provided by the evaluation scripts.

Table 6: F1 scores of different models on the dev data and test data in tabular format.

Model	**Dev Phase**	**Test Phase**
LSTM → CNN	0.608	0.609
CNN → LSTM + Attention	0.59	0.605
CNN → LSTM (OvR + BCE)	0.61	0.611
LSTM → CNN + MHA	0.603	0.606

All the architectures involve a combination of CNN and LSTM layers, which can extract both local features (via CNNs) and sequential dependencies (via LSTMs). This overlap in functionality leads to similar performance across models.

7 Error Analysis

The sentiment classification task is straightforward and doesn't require highly complex architectures, so simpler models like LSTM → CNN perform almost as well as more sophisticated ones.

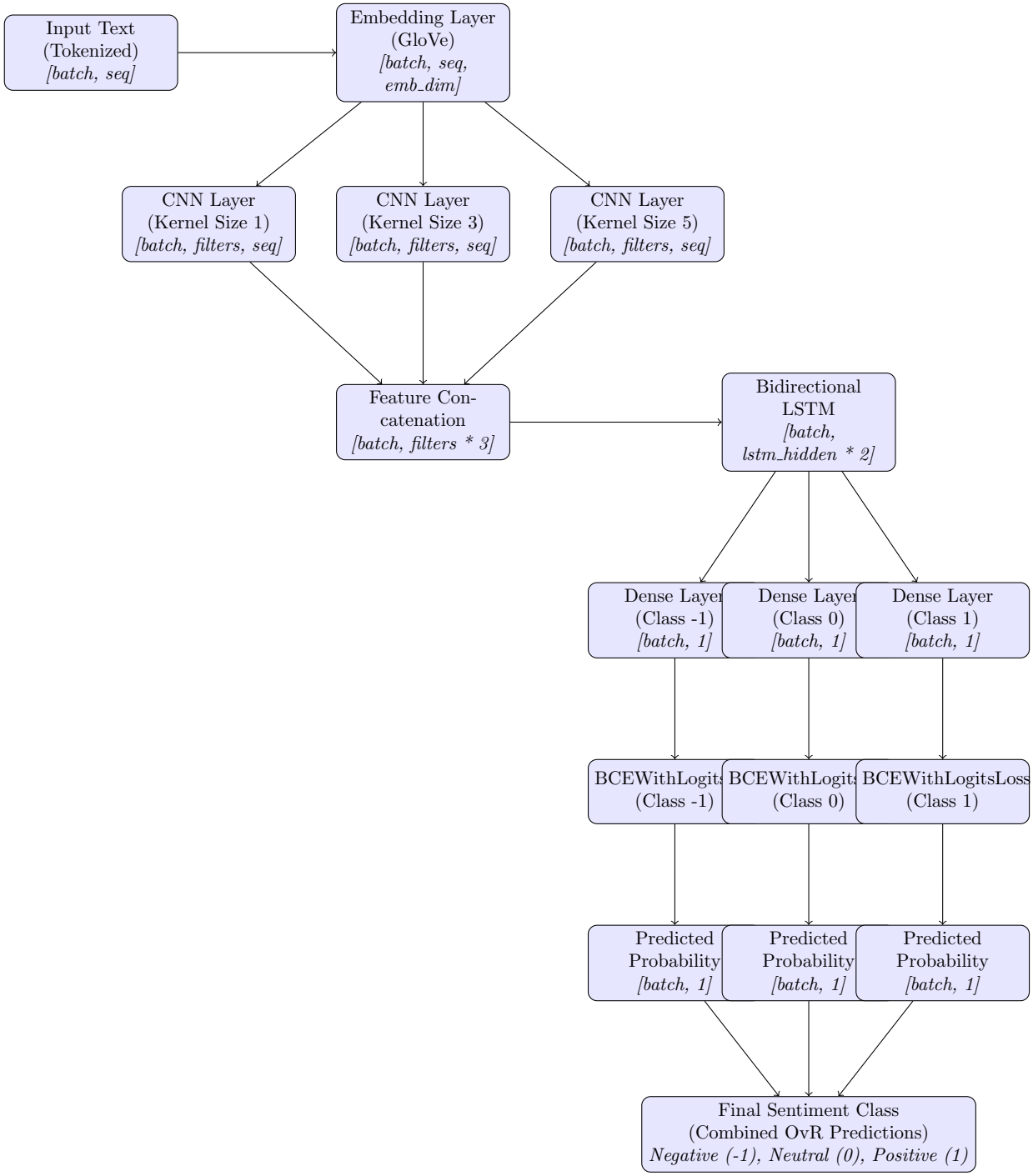
- Why CNN → LSTM (OvR + BCE) performed better
  - OvR allows the model to focus on each class individually. This provides an advantage in handling class imbalance issues.
  - The model combined CNNs for local features and LSTMs for sequential patterns.
- Why did CNN → LSTM + Attention not perform
  - Attention Mechanism is useful for long complex sentences where the added context weights makes a significant difference. The given dataset did not contain a significant number of such sentences.

8 Conclusion

The model demonstrates strong results on both the test and dev dataset, highlighting its ability to perform well even in challenging scenarios like this. Despite its strong performance it still has difficulties in distinguishing weak polarized sentences to the neutral ones. Future work can involve exploring transformer based models, using data augmentation to address the imbalance of classes and to introduce mechanisms for sarcasm detection or contextual understanding. Heat maps can also be developed to highlight which parts of the input influence predictions the most.

References

[1] Bohang Chen, Qiongxia Huang, Yiping Chen, Li Cheng, and Riqing Chen. Deep neural networks for multi-class sentiment classification. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 854–859, 2018.



It predicts probabilities for each class independently using BCEWithLogitsLoss in a One-vs-Rest setting. The final sentiment class is determined by combining these probabilities.

Figure 5: Detailed Final Model Architecture: CNN + LSTM with OvR, BCE-WithLogitsLoss, and Sentiment Classification