Natural Language Processing Coursework
COMM061

Rohit Krishnan
Student ID: 6839323

August 13, 2024

1 Introduction

In this report, I present the results of my experimentation on token classification for abbreviation and long form detection. The task involves labelling abbreviations and their corresponding long forms using a BIO tagging scheme. The dataset used for this coursework is derived from biomedical literature, specifically from the PLOS journal articles.

2 Dataset Analysis and Visualization

2.1 Dataset Overview

The dataset consists of 50k labelled tokens with the following labels: B-O, B-AC, I-AC, B-LF, I-LF. Each token is accompanied by its corresponding part-of-speech (POS) tag.

2.2 Data Visualization

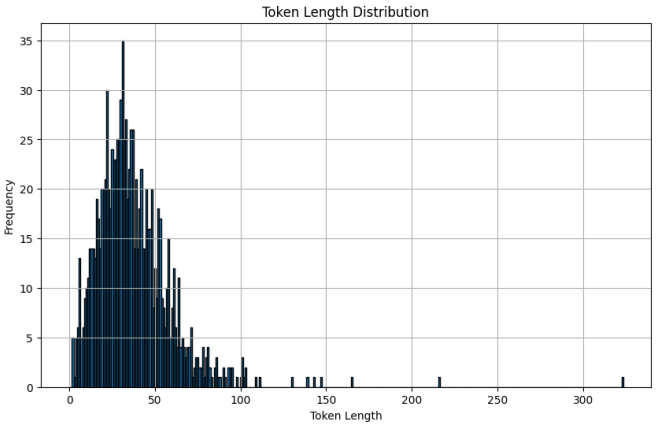

Figure 1: Visualization of the token length of samples from the PLOD-CW dataset

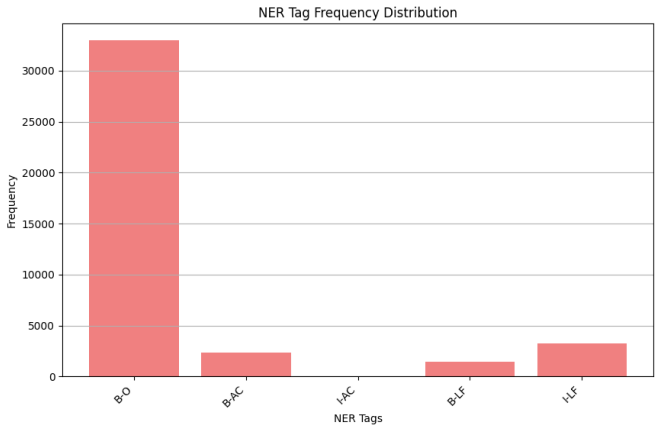

Figure 2: Visualization of the NER tag frequency in the dataset

The histogram of token lengths shows that most tokens have lengths ranging from 0 to 50, with a peak around the 2030 token range. The I-LF tag, which marks tokens inside a long form, also shows a modest frequency, indicating that long forms are often multi-token phrases.

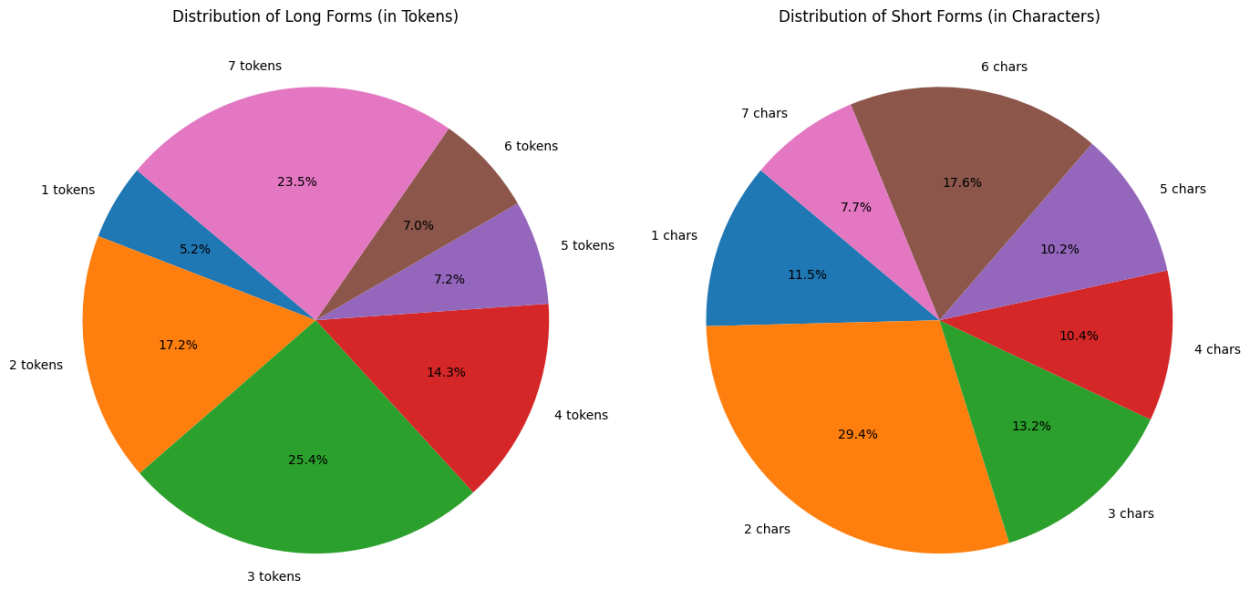

Figure 3: Distributions of long forms in no. of tokens (left) and short forms in no. of characters (right)

Abbreviations are typically concise, with a preference for short character sequences. Long forms tend to be multi-word phrases, with a tendency toward 3 to 7 tokens.

2.3 Observations

These visualizations suggest that the model needs to be robust in handling imbalanced data, capturing contextual information across multiple tokens, and efficiently processing sequences of varying lengths. Pre-trained language models like BERT or RoBERTa, which are adept at capturing contextual relationships, are well-suited for this task.

The distribution also implies that padding strategies need to be efficient. Since most sequences are short, using a fixed-length input size for training could lead to a lot of padding, which might waste computational resources and potentially introduce noise.

The dataset's characteristics imply that evaluation metrics should focus not only on overall accuracy but also on precision, recall, and F1-score for the minority classes (B-AC, B-LF, I-LF). This will ensure that the model is performing well on the key task of identifying abbreviations and long forms, rather than just predicting the majority class correctly. Since there are no instances of the I-AC class, dropping the class alltogether would help in improving the model's performance.

3 Experimentation and Evaluation

3.1 Experiment 1: Fine-Tuning Sequence Classification Models

Several pre-trained language models were fine-tuned, using the PLOD-CW dataset. The models were chosen based on their popularity and effectiveness in sequence classification tasks. Apart from BERT and RoBERTa that are widely accepted models for sequence classification, I also experimented with models pre-trained on biomedical data **TODO:cite**

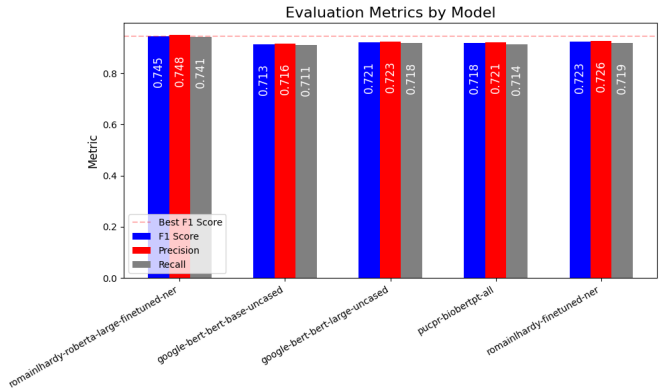

Figure 4: Visualization of the token length of samples from the PLOD-CW dataset

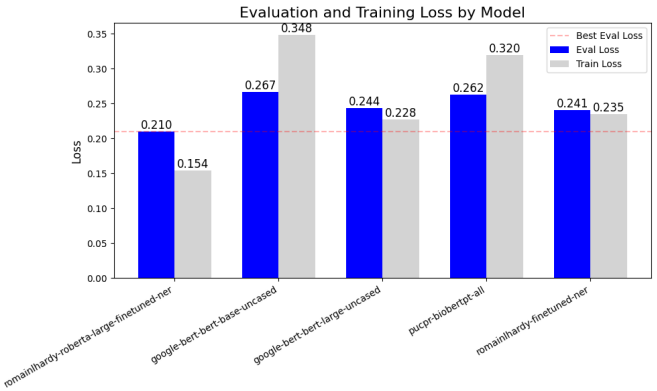

Figure 5: Visualization of the NER tag frequency in the dataset

Table 1: Training results for commonly used models

Model	Batch Size	Epochs	F1 Score ↑	Eval Loss ↓
google-bert/bert-base-uncased	8	20	0.913	0.267
pucpr/biobertpt-all	8	20	0.918	0.262
google-bert/bert-large-uncased	8	20	0.921	0.244
romainlhardy/finetuned-ner (bert-base)	8	20	0.923	0.241
roberta-large-finetuned-ner	8	20	**0.945**	**0.210**

3.2 Experiment 2: Hyperparameter Optimization

Of all the runs, **romainlhardy/roberta-large-finetuned-ner** showed the most promising results. I performed hyperparameter optimization using Optuna for the **romainlhardy/roberta-large-finetuned-ner** model as it had the highest F1 score.

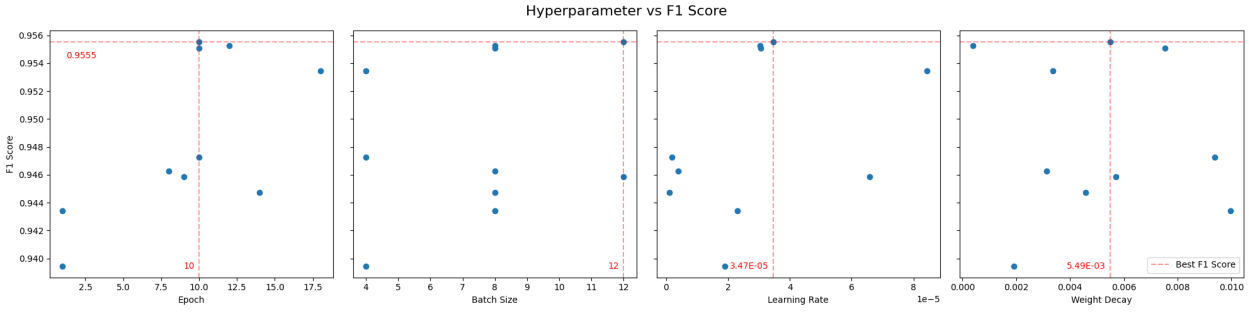

Figure 6: Hyperparameter optimization using Optuna

Table 2: Hyperparameter Optimization Results for google-bert/bert-large-uncased

Learning Rate	Batch Size	Epochs	F1 Score ↑	Eval Loss ↓
0.000008	12	20	0.9062	0.2880
0.000016	12	10	0.9202	0.2509
0.000003	4	10	0.9357	0.2625
0.000003	8	10	0.9322	0.2320
0.000072	12	20	0.9066	0.2694
0.000010	8	15	0.9352	0.3304
0.000004	8	10	0.9342	0.2344
0.000006	12	20	0.9376	0.3362
0.000011	4	15	**0.9394**	0.4690
0.000002	4	20	0.9337	**0.2736**

3.3 Testing and Error Analysis

[Include confusion matrices, classification reports, and any error analysis conducted.]

4 Testing and Deployment

4.1 Best Results and Adjustments

[Discuss the best results obtained, any adjustments made, and their impact.]

4.2 Model Evaluation and Efficiency

[Evaluate the overall performance of the models and discuss the trade-offs between accuracy and efficiency.]

4.3 Model Serving Options

[Research and discuss different model serving options, and justify the choice made for deployment.]

4.4 API/Web-Service Deployment

We deployed the best-performing model as an API endpoint using [framework or tool used]. The architecture is simple, running locally on the machine with the following key components:

- Model loading and inference.
- API endpoint setup using [tool used, e.g., Flask].
- Input/output handling for sequence classification.

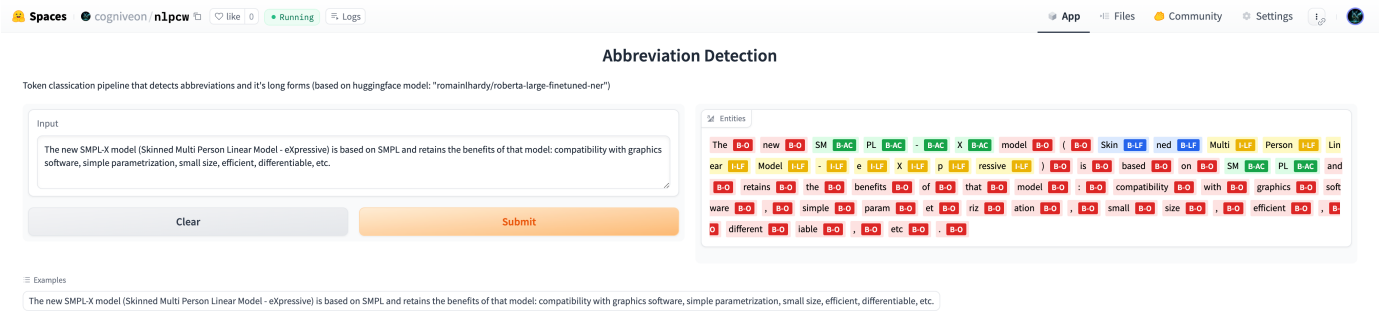

Figure 7: Screenshot of the API response.

5 Conclusion

In this report, we demonstrated the process of fine-tuning pre-trained models for sequence classification, hyperparameter optimization, and deploying the model as an API service. The results show that [summarize your key findings and the effectiveness of your approach].

6 References

[Include all the references for papers, code, and resources you used.]