Jargon Detection
Indian Institute of Technology, Hyderabad

The AAAI-23 Workshop on Scientific Document Understanding

Tuhin Dutta
ai21mtech02002@iith.ac.in

Debika Dipak Samanta
cs22mtech12001@iith.ac.in

***Abstract*—This project aims to identify domain-specific technical terminology (jargon) used in scientific research papers. The task involves identifying specific words or commonly used terms with particular meanings within a particular field. The goal is to tag the main nouns in nominal phrases and model the task as a sentence-level sequence labelling problem. The project provides manually-labelled training and development datasets from three distinct scientific domains: Computer Science, Economics, and Physics. Each scientific domain has its own train/dev/test splits, and the dataset includes over 7000+ Computer Science sentences, 6000+ Economics sentences, and 8000+ Physics sentences. The evaluation is based on precision, recall, and F1 scores on the hidden test set computed for correct prediction for each sentence term. Finally, the Kaggle score estimates the model performance in the hidden private dataset.**

***Index Terms*—sequence-labeling, token-classification, contextual-embeddings, domain-specific-terms ;**

I. Literature Review

Jargon Detection deals with the detection of special terminology used by a particular group of people in that profession and this word may not be regularly used by the people of other domains. This comes under the Sequence Labeling task where we assign, to each word x_i in an input word sequence, a label y_i, so that the output sequence Y has the same length as the input sequence X. Jargon Detection is important as it enables us to identify the context-sensitive terminologies used in a specific domain.

Token classification, also known as part-of-speech (POS) tagging, is a fundamental problem in natural language processing (NLP) that involves assigning a grammatical category or label to each word in a sentence. This task has been widely studied in the literature, and various machine learning techniques have been applied to improve its accuracies, such as rule-based systems, Hidden Markov Models (HMM), Maximum Entropy Markov Models (MEMM), Conditional Random Fields (CRF), and neural network-based models. The availability of large annotated corpora has also contributed to developing more accurate and robust POS tagging models. In recent years, deep learning models, particularly those based on Recurrent Neural Networks (RNNs) [1], have shown remarkable success in POS tagging.

The classical token classification problem suffers from the following issues :

- Limited contextual understanding
- Lack of interpretability
- Over-reliance on annotated data
- Difficulty with out-of-vocabulary (OOV) words
- Difficulty with noisy or ambiguous data

II. Problem Statement

The use of technical terminology in scientific research papers can be a major obstacle for non-experts, as these terms have specific meanings that may not be easily understood by individuals outside the field. This could lead to misunderstandings or prevent readers from understanding scholarly writing altogether. To address this issue, it is essential to identify and recognize technical terms in scientific research papers, as this can facilitate the creation of a scientific-document reading system that helps readers better comprehend scholarly writing. By identifying technical jargon in scientific research papers, it is possible to reduce barriers to entry and improve the reader's comprehension. Non-experienced individuals may fail to understand technical terms or interpret common words in a different sense than intended, creating a significant entry barrier to reading scholarly writing.

- Task aims to identify domain-specific technical terminology or jargon used in scientific research papers.
- Each sentence is treated as a sequence of words, and the model assigns a label to each word in the sequence based on whether it is a technical term/jargon or not. The goal is to identify and tag all the technical terms or jargon in each sentence, which makes it a sequence labelling problem.

III. Dataset

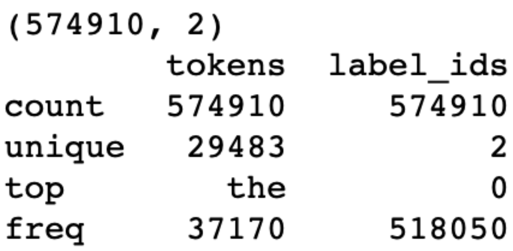
The dataset was provided in a Kaggle competition.

A. Description

- token id contains the domain, document ID, sentence ID, and token ID, in order, separated by "/".
- token contains the cased tokens.
- label contains either "O" or "TERM"
- dataset was available both in CSV as well as JSON format.

B. Specifications

The train data consist of a total of 574910 entries, where label 'O' has 520286 entries and label 'TERM' has 54624 entries. The validation data has 37143 entries. The test data is not labelled and has 42358 entries.



```
(574910, 2)
        tokens  label_ids
count   574910     574910
unique   29483          2
top        the          0
freq     37170     518050
```

Fig. 1 Train Dataset

C. Data Partitioning

The total data is divided into Train, Validation, and Test sets into three different files. Train data is used for training the model. The Validation set was used for the hyperparameter tuning of the model. The Test data was given with no labels and is used for the final evaluation of our models which has been achieved by submitting the prediction file in Kaggle.

D. Data Preprocessing

The column in the dataset is in the form of a token, but the problem is not a simple token classification but it involves contextual understanding of the word. The tokens are arranged in a sequence of sentences. As per the model, we have transformed the data into sentences while adding 'PADwords' and also sometimes removed punctuations and stopwords in the training dataset to address the issue of class imbalance.

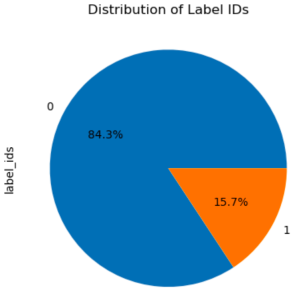


Fig. 2 Train data token distribution

E. Data Visualization

A Word Cloud is a collection or cluster of words depicted in different sizes. The bigger and bolder the word appears, the more often it is selected from a given field. Word Clouds are a powerful way to visualize the most frequently appearing terms belonging to a specified field as shown in Fig. 3. In Fig. 4 We observe the lengths of all sentences in the training dataset plotted in a histogram fashion Vs the frequency of such lengths.

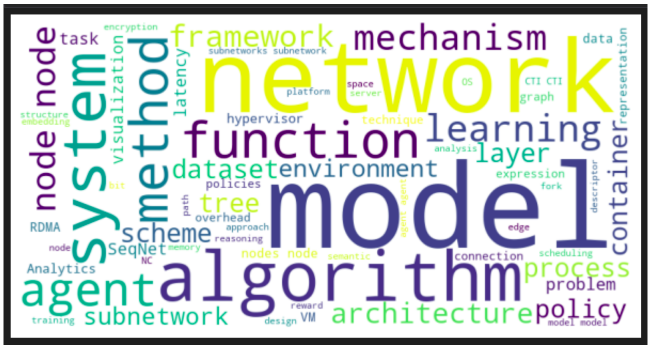


Fig 3.1: This figure shows the terms used in the Computer field.

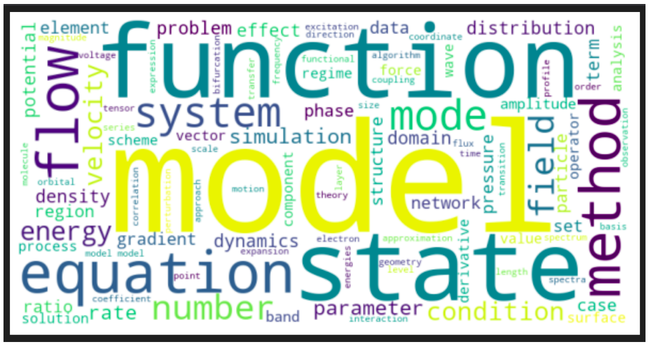


Fig 3.2: This figure shows the terms used in the physics field.

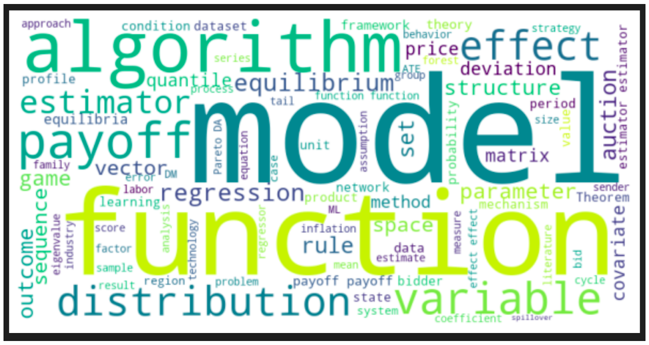


Fig 3.3: This figure shows the terms used in the economics field.

F. Sentence from token

The token in the dataset needs to have some contextual understanding to interpret the proper understanding.

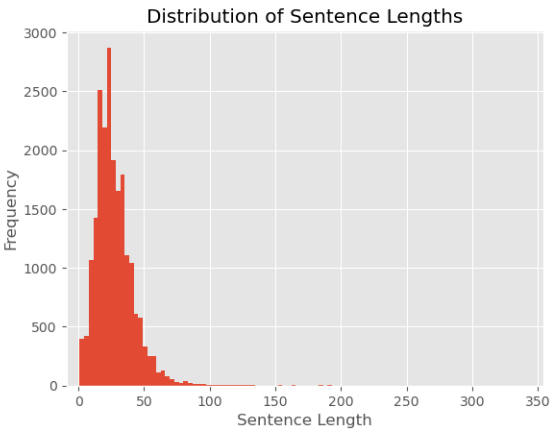


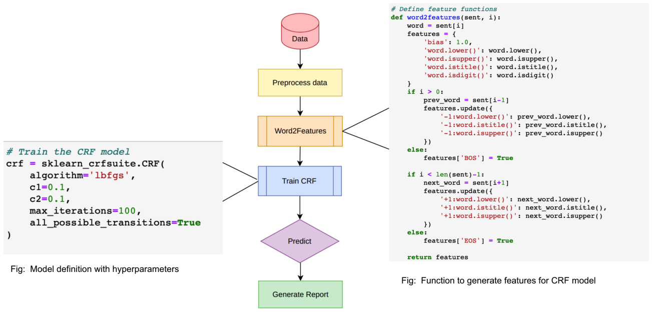
Fig 4: Graph showing sentence length distribution.

IV. Methodology

A. Baseline Model

1) Conditional Random Fields: Conditional Random Fields (CRFs) are a popular machine learning algorithm in natural language processing (NLP) for sequence labeling tasks such as part-of-speech (POS) tagging. CRFs model the dependencies between neighboring labels in a sequence to predict the most probable sequence of labels for a given input. They have outperformed other approaches such as Hidden Markov Models (HMMs) and rule-based methods for POS tagging.

CRF Model Architecture: In the model architecture, we first pre-process the data and then create features using the word2features function and use sklearn_crfsuite for defining the model. We then output the predictions.



```
# Train the CRF model
crf = sklearn_crfsuite.CRF(
    algorithm='lbfgs',
    c1=0.1,
    c2=0.1,
    max_iterations=100,
    all_possible_transitions=True
)
```

Fig: Model definition with hyperparameters

```
# Define feature functions
def word2features(sent, i):
    word = sent[i]
    features = {
        'bias': 1.0,
        'word.lower()': word.lower(),
        'word.isupper()': word.isupper(),
        'word.istitle()': word.istitle(),
        'word.isdigit()': word.isdigit()
    }
    if i > 0:
        prev_word = sent[i-1]
        features.update({
            '-1:word.lower()': prev_word.lower(),
            '-1:word.istitle()': prev_word.istitle(),
            '-1:word.isupper()': prev_word.isupper()
        })
    else:
        features['BOS'] = True

    if i < len(sent)-1:
        next_word = sent[i+1]
        features.update({
            '+1:word.lower()': next_word.lower(),
            '+1:word.istitle()': next_word.istitle(),
            '+1:word.isupper()': next_word.isupper()
        })
    else:
        features['EOS'] = True

    return features
```

Fig: Function to generate features for CRF model

Fig 5: Pure CRF Model

CRF Model specification: A CRF is a log-linear model that assigns a probability to an entire output (tag) sequence Y, out of all possible sequences Y, given the entire input (word) sequence X. We can think of a CRF as like a giant version of what multinomial logistic regression does for a single token. Recall that the feature function f in regular multinomial logistic regression can be viewed as a function of a tuple: a token x and a label y. In a CRF, the function F maps an entire input sequence X and an entire output sequence Y to a feature vector. Let's assume we have K features, with a weight w_k for each feature F_k :

$$p(Y|X) = \frac{\exp\left(\sum_{k=1}^{K} w_k F_k(X,Y)\right)}{\sum_{Y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y')\right)}$$

We'll call these K functions F (X, Y) global features since each one is a property of the entire input sequence X and output sequence Y., We compute them by decomposing into a sum of local features for each position I in Y :

$$F_k(X,Y) = \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i)$$

$$\begin{aligned}
\hat{Y} &= \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} P(Y|X) \\
&= \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} \frac{1}{Z(X)} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y)\right) \\
&= \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} \exp\left(\sum_{k=1}^{K} w_k \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i)\right) \\
&= \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} \sum_{k=1}^{K} w_k \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i) \\
&= \underset{Y \in \mathcal{Y}}{\operatorname{argmax}} \sum_{i=1}^{n} \sum_{k=1}^{K} w_k f_k(y_{i-1}, y_i, X, i)
\end{aligned}$$

The features function and the main training algorithm define the CRF model. We also define the Viterbi algorithm used within CRF and the loss function.

The 'word2features' function defined above is used for generating features for a Conditional Random Field (CRF) model. The features are designed to capture different characteristics of words in a sentence, and they include:

1) bias: A constant bias feature with a value of 1.0, which helps to capture any general bias in the data.
2) word.lower(): The lowercase form of the current word, which helps to normalize the word and capture its lexical information.
3) word.isupper(): A boolean feature that indicates whether the current word is in uppercase or not, which helps to capture the case information of the word.
4) word.istitle(): A boolean feature that indicates whether the current word is in titlecase or not, which helps to capture the titlecase information of the word.
5) word.isdigit(): A boolean feature that indicates whether the current word is a digit or not, which helps to capture numerical information in the word.

6) -1:{feature}: Similar to features as described above but for the previous word in the sentence, denoted by -1 prefix. These features capture the context of the previous word.
7) +1:{features}: Similar to features as described above but for the next word in the sentence, denoted by +1 prefix. These features capture the context of the next word.
8) BOS: A boolean feature that indicates whether the current word is at the beginning of the sentence.
9) EOS: A boolean feature that indicates whether the current word is at the end of the sentence.

The crf model defined is an instance of the CRF class from the sklearn_crfsuite library, which is used for training a Conditional Random Field (CRF) model for sequence labelling tasks. The different parameters used in the model definition are as follows:

1) Optimization algorithm=lbfgs: The optimization algorithm used for training the CRF model. In this case, the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) algorithm is used, which is a widely used optimization algorithm for training CRF models. The LBFGS algorithm is a type of quasi-Newton method that falls under the category of optimization algorithms known as iterative methods. It is particularly well-suited for optimizing smooth, unconstrained functions, such as the objective function used in CRF training. The algorithm approximates the Hessian matrix (a matrix of second-order derivatives) of the objective function using limited memory, which makes it computationally efficient and well-suited for large-scale problems.
2) c1 and c2: The L1 and L2 regularization strengths, respectively, are applied during training to prevent overfitting.
3) all_possible_transitions=True: A boolean flag that indicates whether to include all possible transitions in the CRF model. If set to True, the model will learn the transition probabilities for all possible label pairs, even if they do not occur in the training data. This can be useful for handling unseen label pairs during inference. These parameters collectively define the hyperparameters and configuration settings for the CRF model during training. Adjusting these parameters can impact the model's performance and generalization ability, and finding optimal values often involves experimentation and fine-tuning.

The Viterbi algorithm is used to find the most probable sequence of hidden states that generated a given sequence of observations. It is a dynamic programming algorithm that computes the maximum probability of each possible state sequence at each time step, and backtracks through the sequence to find the most likely path.

CRF Loss Function: The CRF loss function used in the sklearn_crfsuite algorithm module is based on the log-likelihood of the sequence labels given the input features and model parameters. It takes into account the transitions between labels, making it suitable for sequence labelling problems like part-of-speech tagging or named entity recognition. The objective is to maximize the log-likelihood of the correct sequence labels, which is achieved by optimizing the model parameters through gradient-based optimization algorithms like L-BFGS.

2) Bi-LSTMs: We now start with LSTMs and see how they dominate the field of POS tagging tasks. Long Short-Term Memory Networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. A sequence processing model consists of two LSTMs: one taking the input in a forward direction, and the other in a backwards direction. BiLSTMs effectively increase the amount of information available to the network, improving the context available to the algorithm.

Bi-LSTM Model Architecture: Model architecture consists of Bi-LSTMs and a Time-Distributed Dense layer on top of ELMo embeddings.

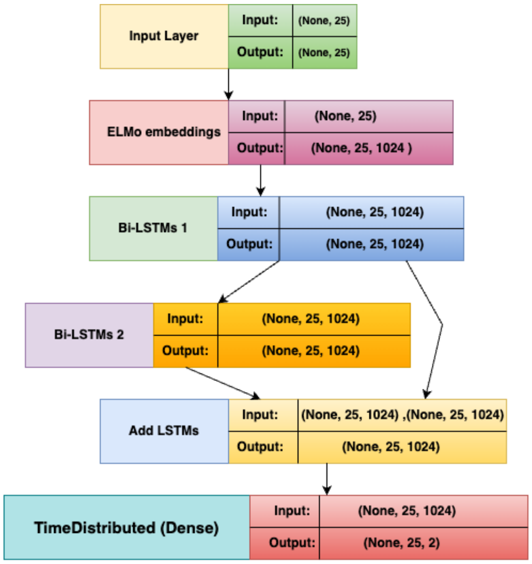


Fig 6: Bi-LSTM Model

Bi LSTM Model specification: The input is a string of sentences with maximum length. The input tensor is passed through an Elmo embedding layer, which generates a 1024-dimensional embedding for each word in the sentence. The Elmo embeddings are then passed through two bidirectional LSTM layers with 512 units, which allows the network to capture both past and future information about each word. The output of the LSTM layers is added with the output of the first LSTM layer to form a residual connection. Finally, a time-distributed dense layer with sigmoid activation is applied to each output of the LSTM layer to predict the probability of each token being a technical jargon or not.

Bi-LSTM Embedding Technique: ELMo (Embeddings from Language Models) is a neural language model that is pre-trained on large datasets and is used for generating high-quality contextualized word embeddings. These embeddings capture not only the syntactic and semantic information of words but also their context. In POS tagging or token classification tasks, where the meaning of a word is often determined by its context in a sentence, ELMo embeddings have shown remarkable improvements over traditional word embeddings. In ELMo, words are represented as vectors of different dimensionality based on their context, which are then passed through a deep, bi-directional LSTM model to generate the final embedding.

Bi-LSTMs Loss Function: BinaryCrossentropy calculates the cross-entropy loss between the predicted probability distribution and the actual probability distribution of the binary labels. The predicted distribution is obtained using a sigmoid activation function applied to the model output, which ensures the output lies in the range of [0,1].

3) BERT For Token Classification:

BERT Model Architecture:

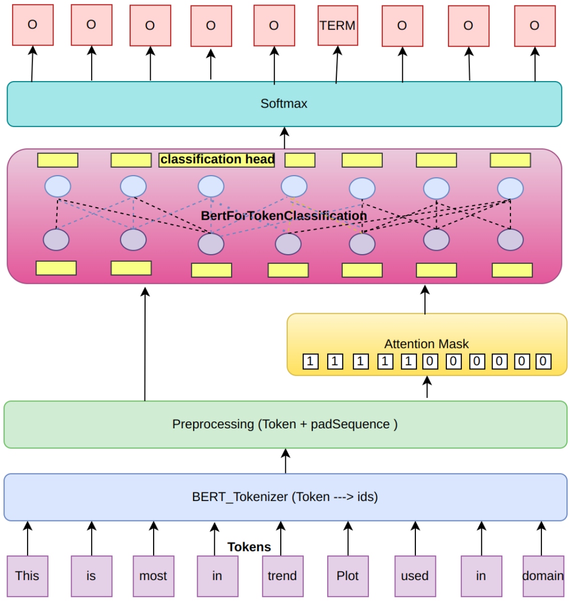


Fig 7: This is the BERT model architecture used for token classification.

BERT Model specification: Fine-tuning in NLP refers to the procedure of re-training a pre-trained language model using your own custom data. As a result of the fine-tuning procedure, the weights of the original model are updated to account for the characteristics of the domain data and the task you are interested in.

- BertForTokenClassification -'bert-base-uncased'
- BertForTokenClassification -'bert-base-cased'
- BertForTokenClassification -'bert-large-uncased'
- BertForTokenClassification -'bert-large-cased'

BERT Embedding Technique: BERT uses Wordpiece embeddings input for tokens. Along with token embeddings, BERT uses positional embeddings and segment embeddings for each token. In our use case, we would be using BERT Tokenizer. There are 4 main types in the BERT Tokenizer

- BertTokenizer -'bert-base-uncased'
- BertTokenizer -'bert-base-cased'
- BertTokenizer -'bert-large-uncased'
- BertTokenizer -'bert-large-cased'

BERT Loss Function: In Bert for token labeling the classification head is a linear layer that accepts the encoder's hidden states. The hidden states represent the learned features from each audio frame which can have varying lengths. To create one vector of fixed length, the hidden states are pooled first and then transformed into logits over the class labels. The cross-entropy loss is calculated between the logits and target to find the most likely class.

B. Hybrid Model

In the hybrid model architecture, we combine the power of both Bi-LSTMs and CRF.

1) Model Architecture: Here we keep the architecture the same as Bi-LSTMs but we add an extra CRF layer at the end to improve our evaluation metric scores. This layer is added to the model to predict the sequence labels. The model predicts two labels for each token.

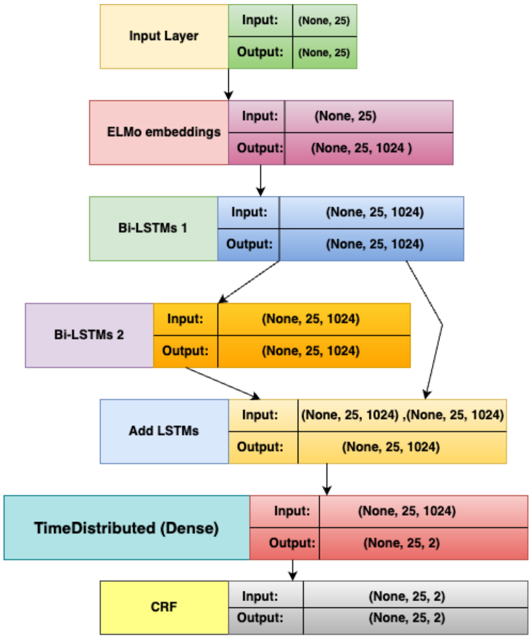


Fig 8: Bi-LSTM with CRF model architecture

2) Bi-LSTMs with CRF Loss Function: The loss function used in this model is the negative log-likelihood. This loss function takes into account the joint probability distribution of the predicted sequence and the true sequence labels. Also, the model uses crf_accuracy as a metric, which computes the proportion of labels that are predicted correctly by the model. The accuracy and loss are plotted against three epochs as measured during training.

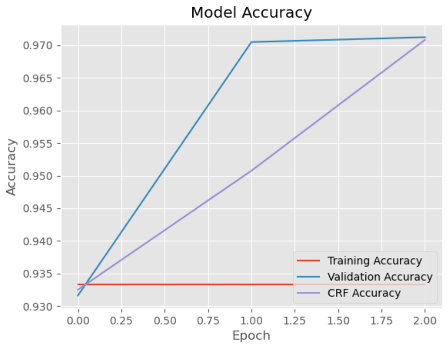


Fig 9: Accuracy plot during model training

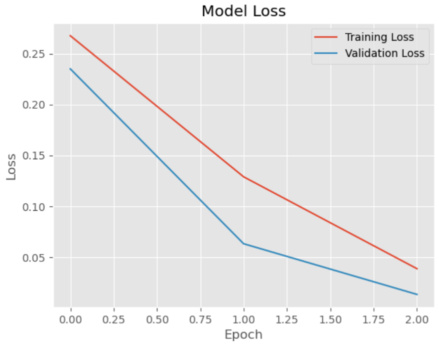


Fig 10: Loss plot during model training

V. Evaluation Metrics

	Actual Positive	Actual Negative
Predicted positive	L_{TP}	L_{FP}
Predicted negative	L_{FN}	L_{TN}

$$Label_{Total} = \begin{cases} L_{TPi} \textbf{ if } y_i = 1 \textbf{ and } w_\theta(X_i) \approx 1 \\ L_{TNi} \textbf{ if } y_i = 0 \textbf{ and } w_\theta(X_i) \approx 0 \\ L_{FPi} \textbf{ if } y_i = 0 \textbf{ and } w_\theta(X_i) \approx 1 \\ L_{FNi} \textbf{ if } y_i = 1 \textbf{ and } w_\theta(X_i) \approx 0 \end{cases}$$

1) Precision: proportion of true positive of all positive prediction.

$$Precision = \frac{L_{TP}}{L_{TP} + L_{FP}} \tag{1}$$

2) Recall: proportion of true positive of all actual positives.

$$Recall = \frac{L_{TP}}{L_{TP} + L_{FN}} \tag{2}$$

3) F1 score: harmonic mean of precision and recall

$$F1 = \frac{2 * L_{TP}}{2 * L_{TP} + L_{FP} + L_{FN}} \tag{3}$$

4) Accuracy: evaluation score (eval dataset) and test score(comparing the Kaggle score)

$$Accuracy = \frac{L_{TP} + L_{TN}}{L_{TP} + L_{TN} + L_{FP} + L_{FN}} \tag{4}$$

VI. Results

Accuracy score of each model

Model Name	Test Accuracy
LSTM + ELMO + CRF	96.292
LSTM + ELMO	96.103
BertTokenClassification(base-case)	88.240
BertTokenClassification(base-uncase)	86.422
BertTokenClassification(large-case)	90.602
BertTokenClassification(large-uncase)	87.201
Roberta(Roberta base)	49.657
CRF	88.099

A. Accuracy comparison among the models ?

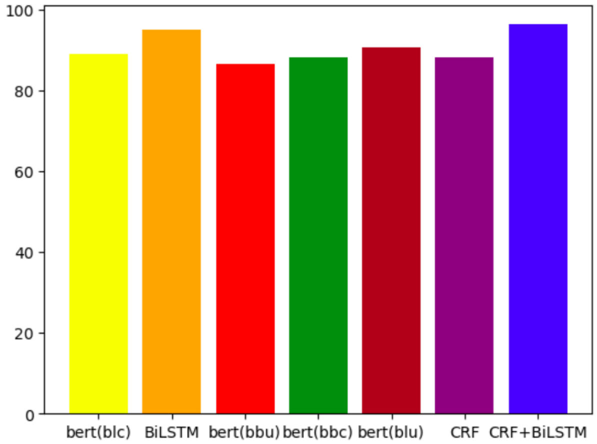


Fig 11: This figure compares the performance of different models.

B. Where the model fails ?

1) Pre-trained language models are trained on the generalized dataset, so they fail at domain-specific terminology that the model has not encountered before.
2) Model might struggle to learn the label of most rare tokens.
3) Model may encounter new terms during the testing phase, and find it difficult to classify it.
4) Accuracy also gets affected due to null entries in the dataset(null entries should be replaced with a substitute like a space character).

Conclusion

In this project, we explored different approaches to tackle the task of identifying domain-specific technical terminology [3] in scientific research papers. We started by implementing a simple CRF model with hand-crafted features and achieved moderate performance. Next, we incorporated ELMo

[5] embeddings for contextual representation along with Bi-LSTMs [2], which resulted in a significant improvement in the model's performance on all evaluation metrics. We also tried with BERTForTokenClassification [4] pre-trained model by fine-tuning its training on our dataset [7], which also gave comparable performance with ELMo and Bi-LSTMs. Our findings suggest that the ELMo-based model outperformed the simple CRF model and performed comparably to the BERT-based model. Overall, our experiments highlight the importance of contextual representation in NLP tasks and the significance of using advanced models such as ELMo and BERT for achieving state-of-the-art results. We came up with a hybrid model (ELMo + Bi-LSTM + CRF) [6], which gave the highest performance. We are constantly exploring ways to fine-tune our model hyperparameters and combine various model architectures in order to enhance the performance level.

The notebooks, dataset and output files can be found in this repo: github//CS5700-IITH-NLP

Acknowledgment

We would like to express our heartfelt gratitude to everyone who contributed to the success of this project. Firstly, we extend our sincere thanks to our project supervisor, Mr. Kaushal Kumar Maurya, and our faculty guide, Dr. Maunendra Sankar Desarkar, for their valuable guidance and unwavering support throughout the project. Their insightful feedback was instrumental in shaping the direction of our work. We would also like to acknowledge Planet and WiNET labs for providing us with access to their facilities and resources, which were essential to the success of this project. Lastly, we would like to thank our peers in the AI department who collaborated with us on this project.

References

[1] Chiche, A., Yitagesu, B. Part of speech tagging: a systematic review of deep learning and machine learning approaches. J Big Data 9, 10 (2022). https://doi.org/10.1186/s40537-022-00561-y

[2] Amir Jafari , "Comparison Study Between Token Classification and Sequence Classification In Text Classification," November 28, 2022.

[3] Han Wang, Qin Xiang Ng, Angeline Tung, A Weakly-Supervised Named Entity Recognition Machine Learning Approach for Emergency Medical Services Clinical Audit, International Journal of Environmental Research and Public Health, July 2021.

[4] Zhiyong He, Zanbo Wang, Wei Wei, Shanshan Feng, Xianling Mao, Sheng Jiang, A Survey on Recent Advances in Sequence Labeling from Deep Learning Models, 13 Nov 2020.

[5] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, Luke Zettlemoyer, Deep contextualized word representations, Allen Institute for Artificial IntelligencePaul G. Allen School of Computer Science Engineering, University of Washington, March 2018.

[6] Zhiheng Huang, Wei Xu, Kai Yu, Bidirectional LSTM-CRF Models for Sequence Tagging, CoRR, 2015.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, CoRR, 2018.