NATURAL LANGUAGE PROCESSING COMM061

COURSEWORK

2023-2024

Table of Contents

SECTION 1

ANALYZE AND VISUALIZE THE DATASET

1. Overview of the dataset

- For this section, first we study the structure of the datasets and its type.

```
datasets

DatasetDict({
    train: Dataset({
        features: ['tokens', 'pos_tags', 'ner_tags'],
        num_rows: 1072
    })
    validation: Dataset({
        features: ['tokens', 'pos_tags', 'ner_tags'],
        num_rows: 126
    })
    test: Dataset({
        features: ['tokens', 'pos_tags', 'ner_tags'],
        num_rows: 153
    })
})
```

- We see the 'datasets' object is of type DatasetDict which is from the Hugging Face datasets library. It consists of the subsets: train, validation, and test datasets.
- Each of the splits (wiz. Train, Test & Validation) includes features such as tokens, POS tags and NER tags. We also see the number of rows in different splits.
- Next up, using a function that displays random rows for dataset, preview few rows of the train split of the dataset.

```
display_random_rows(datasets['train'])
```

	tokens	pos_tags	ner_tags
0	[Mice, that, need, to, be, perfused, with, PBS, and/or, 4, %, paraformaldehyde, (, PFA,), prior, to, tissue, harvest, were, anesthetized, with, a, combination, of, ketamine, (, 20, mg, /, kg, body, weight,), and, xylazine, (, 2, mg, /, kg, body, weight,), .]	[NOUN, PRON, VERB, ADP, AUX, VERB, ADP, PROPN, CCONJ, NUM, NOUN, NOUN, PUNCT, PROPN, PUNCT, ADV, ADP, NOUN, NOUN, AUX, VERB, ADP, DET, NOUN, ADP, NOUN, PUNCT, NUM, NOUN, SYM, NOUN, NOUN, NOUN, PUNCT, CCONJ, NOUN, PUNCT, NUM, NOUN, SYM, NOUN, NOUN, NOUN, PUNCT, PUNCT]	[B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-AC, B-O, B-O, B-O, B-LF, B-O, B-AC, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O]
1	[This, indicates, that, a, Chromosome, 4, bivalent, experiences, on, average, 1.76, CO, in, male, meiosis, ,, but, only, 1.05, CO, in, female, meiosis, (, M, /, F, ratio, 1.68,), .]	[PRON, VERB, SCONJ, DET, NOUN, NUM, ADJ, VERB, ADP, ADJ, NUM, PROPN, ADP, ADJ, NOUN, PUNCT, CCONJ, ADV, NUM, PROPN, ADP, ADJ, NOUN, PUNCT, PROPN, SYM, PROPN, NOUN, NUM, PUNCT, PUNCT]	[B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-AC, B-O, B-LF, B-O, B-O, B-O, B-O, B-O, B-AC, B-O, B-LF, B-O, B-O, B-AC, B-O, B-AC, B-O, B-O, B-O, B-O]
2	[Within, the, first, week, after, surgery, ,, 7.7, ±, 0.6, %, of, dendritic, spines, were, eliminated, and, 8.1, ±, 0.5, %, were, formed, in, sham, mice, ,, whereas, 11.4, ±, 1.1, %, and, 11.9, ±, 1.1, %, of, dendritic, spines, were, eliminated, and, formed, in, SNI, mice, ,, respectively, .]	[ADP, DET, ADJ, NOUN, ADP, NOUN, PUNCT, NUM, SYM, NUM, NOUN, ADP, ADJ, NOUN, AUX, VERB, CCONJ, NUM, SYM, NUM, NOUN, AUX, VERB, ADP, ADJ, NOUN, PUNCT, SCONJ, NUM, SYM, NUM, NOUN, CCONJ, NUM, SYM, NUM, NOUN, ADP, ADJ, NOUN, AUX, VERB, CCONJ, VERB, ADP, PROPN, NOUN, PUNCT, ADV, PUNCT]	[B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-AC, B-O, B-O, B-O, B-O]
3	[IQR, :, interquartile, range, Similarly, ,, assessment, of, other, quantitative, variables, is, essential, in, pre-, and, post, -, MI, conditions, which, primarily, includes, ,, plasma, glucose, levels, ,, body, mass, index, (, BMI,), ,, high, density, lipid, (, HDL,), ,, low, density, lipid, (, LDL,), and, triglycerides, levels, .]	[PROPN, PUNCT, ADJ, NOUN, ADV, PUNCT, NOUN, ADP, ADJ, ADJ, NOUN, AUX, ADJ, ADP, ADJ, CCONJ, ADJ, ADJ, PROPN, NOUN, PRON, ADV, VERB, PUNCT, NOUN, NOUN, NOUN, PUNCT, NOUN, NOUN, NOUN, PUNCT, PROPN, PUNCT, PUNCT, ADJ, NOUN, NOUN, PUNCT, PROPN, PUNCT, PUNCT, ADJ, NOUN, NOUN, PUNCT, PROPN, PUNCT, CCONJ, NOUN, NOUN, PUNCT]	[B-AC, B-O, B-LF, I-LF, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-AC, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-O, B-LF, I-LF, I-LF, B-O, B-AC, B-O, B-O, B-LF, I-LF, I-LF, B-O, B-AC, B-O, B-O, B-LF, I-LF, I-LF, B-O, B-AC, B-O, B-O, B-O, B-O, B-O]
4	[BMI, ,, body, mass, index, ;, HR, ,, hazard, ratio, ;, SD, ,, standard, deviation, .]	[PROPN, PUNCT, NOUN, NOUN, NOUN, PUNCT, PROPN, PUNCT, NOUN, NOUN, PUNCT, PROPN, PUNCT, ADJ, NOUN, PUNCT]	[B-AC, B-O, B-LF, I-LF, I-LF, B-O, B-AC, B-O, B-LF, I-LF, B-O, B-AC, B-O, B-LF, I-LF, B-O]

From this data, we observe that Tokens are the individual words/symbols extracted from the text with their respective POS tags and NER tags assigned to each token.

2. Token Analysis

- Start by displaying top 10 most frequently appearing tokens in each split of the dataset.

```
[(',', 2268),
 ('(', 1583),
 (')', 1575),
 ('the', 1136),
 ('.', 1073),
 ('of', 995),
 ('and', 903),
 ('-', 901),
 (';', 691),
 ('in', 585)]
```

1. Train Split

```
[(',', 244),
 ('(', 195),
 (')', 188),
 ('the', 168),
 ('of', 143),
 ('.', 126),
 ('-', 125),
 ('and', 111),
 ('in', 72),
 (']', 62)]
```

2. Validation Split

```
[(',', 264),
 ('(', 191),
 (')', 184),
 ('.', 152),
 ('the', 141),
 ('of', 135),
 ('and', 119),
 ('-', 115),
 ('in', 78),
 (';', 57)]
```

3. Test Split

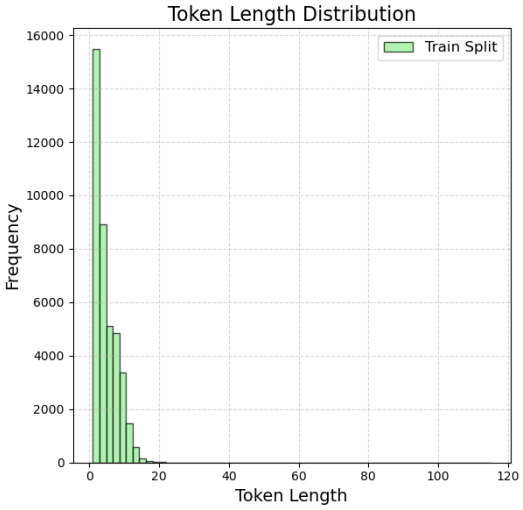
We observe that all of them are either punctuation marks or common English words (the, of, and etc) called stop words in NLP. This analysis helps us in understanding whether we go ahead with further preprocessing steps such as stop word removal or punctuation removal to clean the data to reduce noise. Doing so might help the model focus more on informative words rather than these which don't mean much.

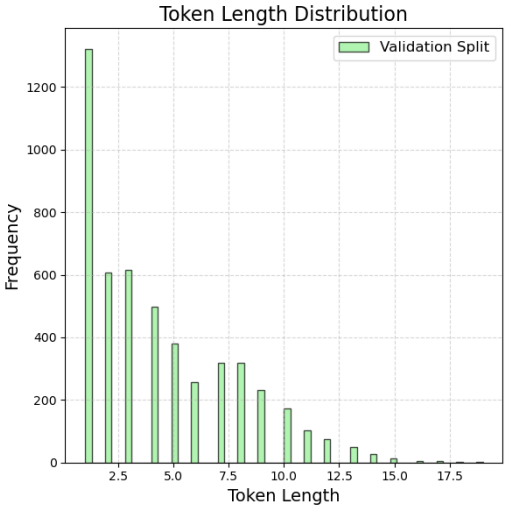
Upon additional analysis we see:

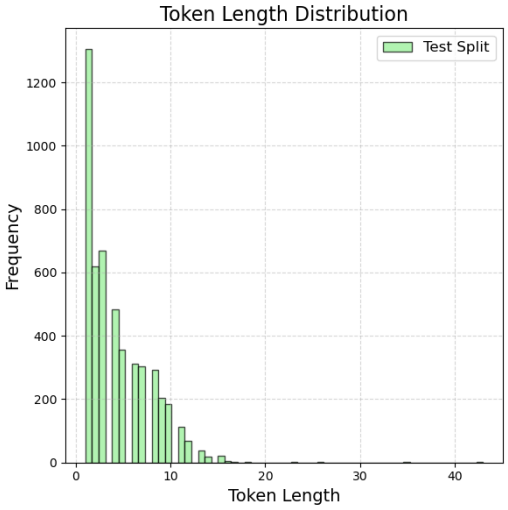
	TRAIN	***VALIDATION***	***TEST***
No. of Tokens	40000	5000	5000
No. of Unique Tokens	9133	1963	1974
No. of Rare Tokens	5604	1425	1411
Max Length Token	115	19	43
Min Length Token	1	1	1

Next step is to visualize the distribution of token length across train, validation, and test splits of the dataset.

- From the histogram we see the distribution is skewed towards tokens with shorter length which indicates the presence of many common words, stop words or punctuations.







3. Named Entity Recognition Analysis

Start by analysing the NER tags and their count for the train split.

```
Number of NER tags: {'B-O', 'B-AC', 'I-LF', 'B-LF'}
Distribution of NER Tag:
B-O = Count: 32971
B-LF = Count: 1462
I-LF = Count: 3231
B-AC = Count: 2336
```

We observe that 'B-O' tag shows dominance covering over 82% being the most frequent tag and 'B-AC' comprising about just 5.84%. This extremely large portion of 'B-O' tags suggests that large part of text does not even involve named entities.

.

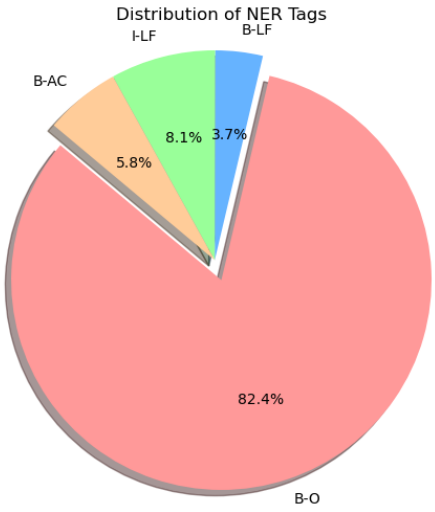
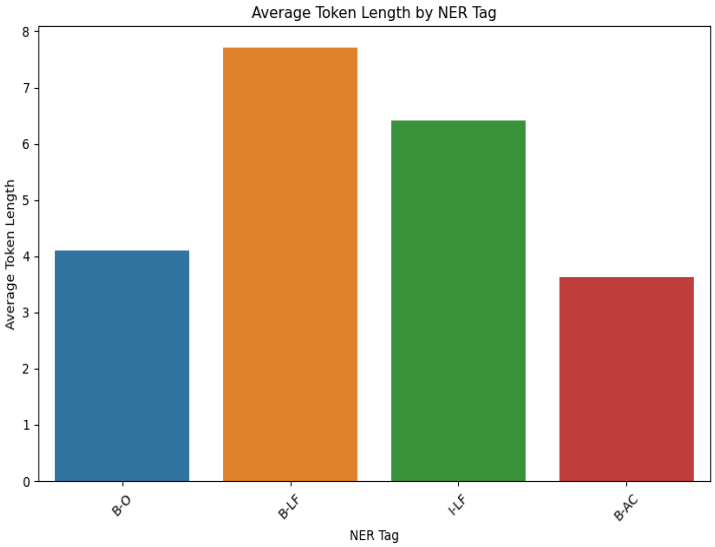
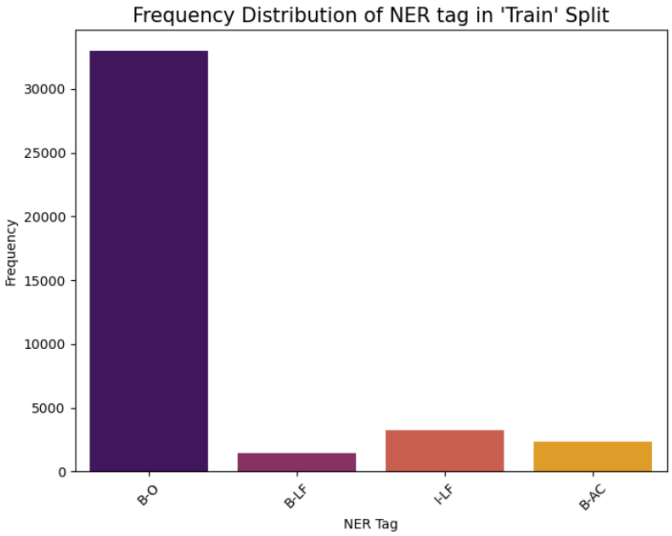


Fig. Pie Chart Distribution of NER Tags



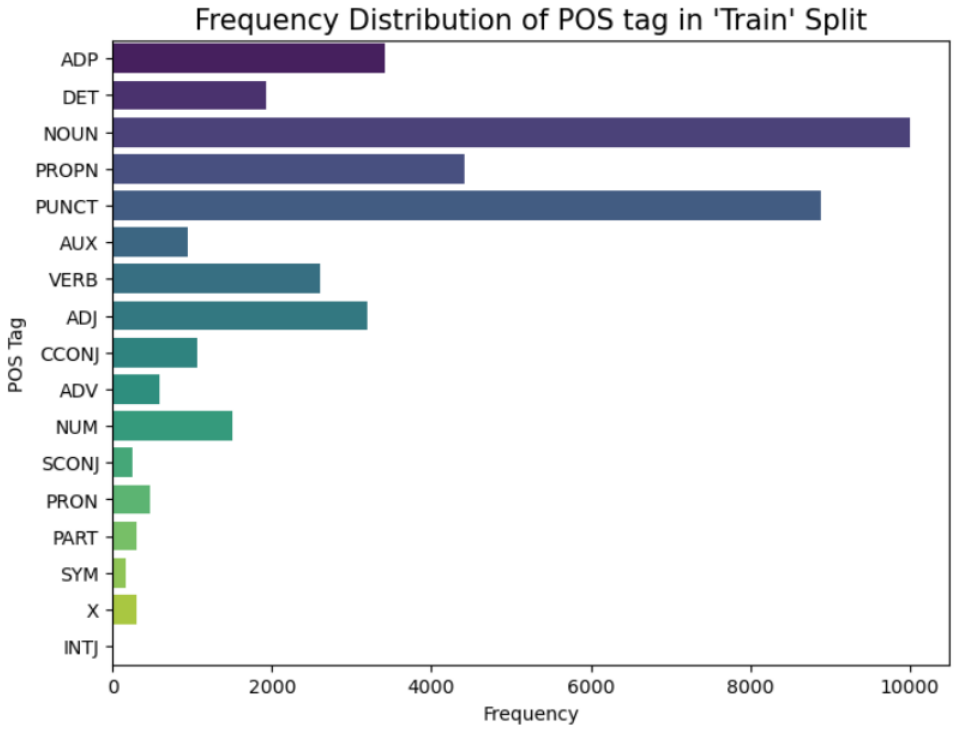


- The frequency distribution of NER tags gives a clear picture of how each tag contributes to the dataset. It shows imbalance in the dataset which is crucial when we are dealing with tasks like Named Entity Recognition to understand balance and potential bias as it helps in tuning the models to capture less frequent tags.

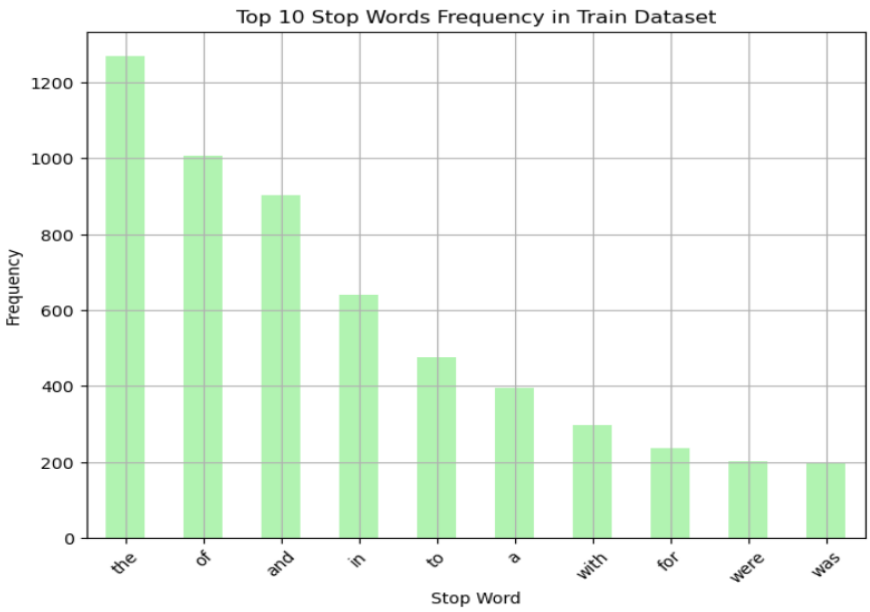
- The first bar chart illustrates the average token length associated with each NER tag. We observe the variation in token length across different entity types, which makes sense because tags like 'B-AC' are acronyms/abbreviations while 'B-LF' and 'I-LF' are the long forms of these acronyms.

4. Parts-Of-Speech Analysis

- We observe that 'NOUN' and 'PUNCT' are the most frequent POS tags, 'PROPN', 'ADJ' and 'VERB' are also common. This information can help in tuning model to focus more on relevant tags like NOUN or PROPN.



5. Stop Word Analysis



This visualization shows top 10 most frequent stop words in our train split of dataset.

For tasks like Named Entity Recognition, removing or retaining stop words play an important role depending on model's needs since it has its pros and cons. Removing stop words can help is noise reduction so the model can divert its attention more on words that are important and informative, however for some models it can aid in discovering important patterns and contribute to a better understanding to ultimately enhance the performance and accuracy of NER systems.

6. Word Cloud

Visualizing word cloud to get a quick visual picture of dataset's content. Words that appear larger like protein, cell, patient, data etc. are top non-stop words which also suggest the theme in the dataset.

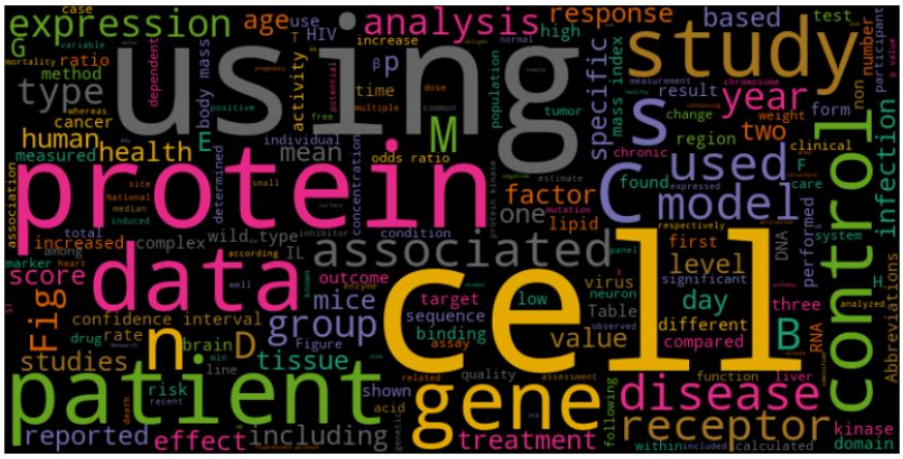


Fig. Word Cloud Train Split

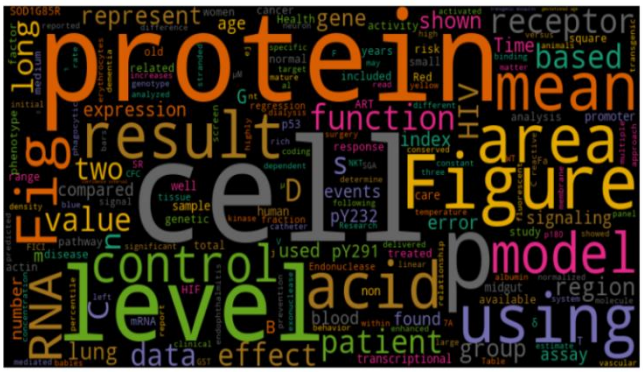


Fig.Word Cloud Validation Split

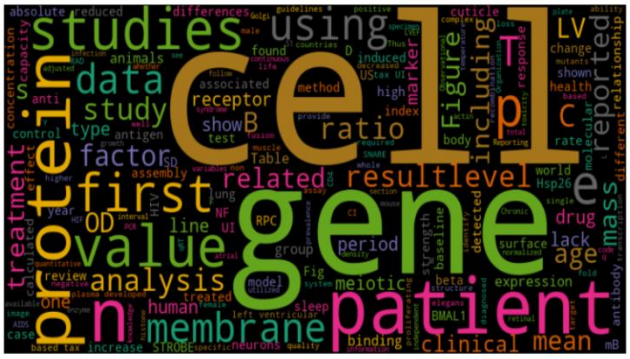


Fig.Word Cloud Test Split

SECTION 2

EXPERIMENTATION WITH FOUR DIFFERENT SETUPS

2.1 Comparing Different Preprocessing Techniques

In this experiment, I decided to compare the effectiveness of various Pre-processing Techniques for tasks like Named Entity Recognition. For this setup I decided to leverage the capabilities of Transformer model employing its *respective tokenizer for tokenization* while *prioritizing the F1-score* as the primary evaluation metric at the end for model performance.

For comparison, we will be implementing:

System 1: *Preprocessing by Lowercasing*

vs

System 2: *Preprocessing by Removing Stopwords and Punctuations*

vs

System 3: *Preprocessing by Lemmatizing*

vs

System 4: *Combining all Preprocessing Steps*

2.1.1 Rationale:

1. Lowercasing: It is a normalization technique that simplifies the data and reduces the vocabulary size and can lead to an improved model performance and efficiency. Many pre-trained model and embeddings are trained on text that is lowercased which might be a good setup for our experiment since our model of choice is going to be BERT. However, it's not always the case because lowercasing might remove critical information, especially for tasks like Named Entity Recognition.

2. Stopword/Punctuation Removal: Stopword are commonly used English words (ex. "the", "and", "is" etc.) that carry little to no meaningful information. Both preprocessing steps can help in reducing noise in the data and make downstream NLP models more efficient.

3. Lemmatization: It is a process in NLP where words are reduced to their root form. For example, "running" becomes "run", "better" becomes "good", "children" becomes "child" and so on. This process simplifies the text data by treating different forms of a particular word as on. It enhances the model's ability to learn from the data, since it doesn't treat different forms of same words as different words.

4. Combination of Preprocessing Steps: When combined, these preprocessing steps can help in reducing noise and variability in the data but combination of all may not always yield the best results. The aim of this setup is to highlight the importance of carefully selecting preprocessing methods to optimize the model performance.

2.1.2 Experimental setup:

Tokenization: We will tokenize the dataset using BERT tokenizer before training the model.

Model: The model of choice is BERT (Bidirectional Encoder Representations from Transformers). It is pre-trained on a large corpus of text data which helps the model to understand nuances which plays a significant role in

tasks like Named Entity Recognition. We will start with a pre-trained RoBERTa model ("roberta-base") and fine-tune it on the NER dataset.

Hyperparameters: We will use the same hyperparameters, including batch size, learning rate, and number of epochs, keeping them constant across all systems for a fair comparison.

2.1.3 Experimental Procedure:

- Fine-tune the pre-trained model for each loss function.
- Train the model for a fixed number of epochs and monitor its performance on validation dataset.
- Evaluate and assess the performance of model using SeqEval, prioritizing the F1 score as the primary evaluation metric.
- Obtain predictions on the test dataset for a final assessment to assess generalization performance.

2.1.4 Detailed Analysis of the Code:

Here's a step-by-step breakdown of the code for all the systems:

1. Load the Dataset
   - The script begins by loading the PLOD-CW[1] dataset using the *'load_dataset'* function from the Hugging Face Transformers Library
2. Load the RoBERTa Tokenizer and Model
   - The RoBERTa tokenizer from the pre-trained '*roberta-base'* model is loaded. This tokenizer converts text into tokens that can be processed by the RoBERTa model.
   - We load a RoBERTa based NER model using the *'AutoModelForTokenClassification'* class from the Hugging Face Transformers library. This model is specifically designed for token classification task like NER.
   - Next, we check if CUDA(GPU) is available, and if so, it assigns the device to CUDA, otherwise, it defaults to CPU.
3. Tokenization Function
   - We define a function called '*tokenize_and_align_labels'* where we tokenize the input text and then align the labels with the tokenized inputs.
   - This function takes a dictionary as input with two keys: *'tokens'* (a list of tokens) and *'ner_tags'* (a list of NER tags corresponding to the tokens).
   - It uses the RoBERTa tokenizer to tokenize input tokens, ensuring that the tokenized inputs are padded and truncated as necessary. The '*is_split_into_words'* is set to True since the input tokens are already split into words.
   - The function the iterates over the NER tags for each example and aligns the labels with the tokenized inputs.
     i. It obtains the word IDs for the tokenized inputs using the 'word_ids' method.
     ii. Iterated over the word_ids and appends corresponding label ID (from the '*label_encoding'* dictionary) to a list of label IDs.
     iii. If a word_id is None (indicating a padding token), the function appends '-100' to the list of label IDs. This is common approach to indicates that these labels should be ignored during training or evaluation.
   - The function then adds the list of label ids to the *'labels'* key in the *'tokenized_inputs'* dictionary and returns it.
4. Apply Tokenization to the Datasets
   - The tokenization function is applied to the train, validation and test datasets using the 'map' function to prepare the data for the model.

[1] surrey-nlp/PLOD-CW · Datasets at Hugging Face

5. Define Metrics Computation Function
   - Function '*compute-metrics'* is designed to evaluate the performance of the model. It uses the 'seqeval' metric, which is specifically suited for sequence labelling tasks like NER.
     i. The function takes parameter *'p'* which is a tuple containing predictions and labels.
     ii. *'predictions = np.argmax(predictions, axis=2)'* uses *'np.argmax'* to convert the predictions (which are in a probabilistic format) into actual class predictions by selecting the class with the highest probability.
     iii. *'true_predictions'* generated a list of lists where each sub list contains the predicted labels for the tokens that are not special tokens.
     iv. *'true_labels'* generates a list of lists of the true labels corresponding to the non-special tokens.
6. Set Up Training Arguments
   - Training arguments such as the number of epochs, batch size, and learning rate are defined using the *'TrainingArguments'* class.
7. Preprocessing Functions
   - **System 1: Preprocessing by Lowercasing**
     - For this system, a function named '*preprocess_by_lowercasing'* takes a dataset as input, converts all tokens in the dataset to lowercase and returns the modified tokens.
     - Applies preprocessing to each element in tokenized datasets using .map() method and stores in '*preprocessed_dataset_1'*.
   - **System 1: Preprocessing by Removing Stopword and Punctuations**
     - For this system, a function named '*preprocess_by_stopword_punc'* takes a dataset as input.
     - Initializes empty lists '*tokens', 'pos_tags', and 'ner_tags'* to store the filtered tokens for pre-processed dataset.
     - Iterates over each token and checks if tokens are not a stopword and not a punctuation, and if the token passes the check it is appended to the tokens list with its respective pos_tags and ner_tags.
     - Applies preprocessing to each element in tokenized datasets using .map() method and stores in '*preprocessed_dataset_2'*.
   - **System 1: Preprocessing by Lemmatizing**
     - WordNetLemmatizer Initialization
       - First create an instance of '*WordNetLemmatizer*' class from '*nltk.stem'*.
     - POS Tag to WordNet Tag Mapping
       - Next step is to define function '*get_wordnet_pos'* that converts Treebank POS tags (like 'JJ', 'VB', 'NN', 'RB') to their corresponding WordNet POS tags (ADJ, VERB, NOUN, ADV).
       - This function checks the input POS tag and returns the WordNet tag.
       - If the input tag doesn't match any, it defaults to NOUN.
     - Lemmatization Function
       - Function '*preprocess_lemmatize'* is defined to perform lemmatization on the dataset.
       - It initializes empty lists to store lemmatized tokens with its corresponding pos and ner tags.
       - For each token, it gets the corresponding WordNet POS tag and lemmatizes the token.
       - The lemmatized token, its pos tag and ner tag are appended to the lists and the final pre-processed dataset is returned.
     - Applies preprocessing to each element in tokenized datasets using .map() method and stores in '*preprocessed_dataset_3'*.
   - **System 1: Combining all Preprocessing Steps**
     - For this system, a function named '*preprocess_all'* is defined that follows all the steps as described in the previous setups to preprocess the datasets.

- Applies preprocessing to each element in tokenized datasets using .map() method and stores in '*preprocessed_dataset_4'*.

8. Training the Model with Pre-processed Data
   - The model is trained by calling the '*trainer.train()'* method of the trainer.
9. Evaluate the model with the Validation Dataset
   - Model is evaluated on validation dataset with *'trainer.evaluate()*' and various metrics including precision, recall, F1-score, and accuracy are computed using the '*compute-metrics'* function.
10. Obtain Predictions on Test Set
    - The test dataset is prepared the same way, we obtain true predictions and true labels and then compute metrics.
    - Model predictions are obtained and evaluated using '*trainer.predict()'*.
11. Visualization and Error Analysis
    - A classification report is generated using the '*seqeval_classification_report'* function from seqeval.metrics.
    - Functions are defined for visualizing the Confusion Matrix, Error Distribution, and Error Types across different NER classes.

In the provided code, I've adopted certain functions and structures inspired by a similar implementation. The original source of inspiration can be found here.[2]

Note: After training and evaluating the model by System 1 , the script repeats the process for System 2, 3 and 4. This allows for a comparison of their effects on performance using different preprocessing methods.

2.2 Comparing Loss Functions

In this experiment, I decided to compare the effectiveness of different loss functions that are specially tailored towards tasks like Named Entity Recognition. For this setup I decided to *ditch the traditional preprocessing steps* and instead leverage the capabilities of Transformer model employing its *respective tokenizer for tokenization* while *prioritizing the F1-score* as the primary evaluation metric at the end for model performance.

For comparing loss functions, we will be implementing:

System 1: *Categorical Cross Entropy Loss*

vs

System 2: *Label Smoothing with Categorical Cross Entropy*

vs

System 3: *Focal Loss*

2.2.1 Rationale for Loss Functions:

1. Categorical Cross Entropy: It is standard choice when it comes to sequence labelling tasks like Named Entity Recognition. It measures the difference between the predicted probabilities and the actual class labels for multi-class classification tasks and penalizes the model for incorrect classification. It is widely used in classification tasks, including NER, where the goal is to encourage the model to produce more accurate predictions for entity recognition.

2. Label Smoothing: Label Smoothing is a regularization technique used to make the model less confident about its predictions by adjusting the target labels (i.e. softening the targets). Its

[2] For more information, check out the GitHub repository for PLOD Abbreviation Detection.

combination with Categorical Cross Entropy can significantly enhance model's performance by addressing overfitting and improving its generalization capabilities.

3. Focal Loss: In NER, some entities are more challenging to recognize than others due to various factors like context dependency or ambiguity. Focal Loss is a variant of cross-entropy loss designed to handle class imbalances by making the model focus more on hard examples. It modifies the cross-entropy loss by adding a factor that reduces the contribution from easy examples whilst increasing the contribution from hard examples thereby improving the model performance on minority classes. The underlying loss function used here is categorical cross-entropy.

2.2.2 Experimental setup:

Tokenization: We will tokenize the dataset using RoBERTa tokenizer before training the model.

Model: The model of choice is RoBERTa (Robustly Optimized BERT-Pretraining Approach). It is pre-trained on a large corpus of text data which helps the model to understand nuances which plays a significant role in tasks like Named Entity Recognition. We will start with a pre-trained RoBERTa model ("roberta-base") and fine-tune it on the NER dataset.

Hyperparameters: We will use the same hyperparameters, including batch size, learning rate, and number of epochs, keeping them constant across all systems for a fair comparison.

2.2.3 Experimental Procedure:

- Fine-tune the pre-trained model for each loss function.
- Train the model for a fixed number of epochs and monitor its performance on validation dataset.
- Evaluate and assess the performance of model using SeqEval, prioritizing the F1 score as the primary evaluation metric.
- Obtain predictions on the test dataset for a final assessment to assess generalization performance.

2.2.4 Detailed Analysis of the Code:

Here's a step-by-step breakdown of the code for all the systems:

1. Load the Dataset
   - The script begins by loading the PLOD-CW dataset using the *'load_dataset'* function from the Hugging Face Transformers Library
2. Load the RoBERTa Tokenizer and Model
   - The RoBERTa tokenizer from the pre-trained '*roberta-base'* model is loaded. This tokenizer converts text into tokens that can be processed by the RoBERTa model.
   - We load a RoBERTa based NER model using the *'AutoModelForTokenClassification'* class from the Hugging Face Transformers library. This model is specifically designed for token classification task like NER.
   - Next, we check if CUDA(GPU) is available, and if so, it assigns the device to CUDA, otherwise, it defaults to CPU.
3. Tokenization Function
   - We define a function called '*tokenize_and_align_labels'* similar to Experiment 1 where we tokenize the input text and then align the labels with the tokenized inputs.
4. Apply Tokenization to the Datasets
   - The tokenization function is applied to the train, validation and test datasets using the 'map' function to prepare the data for the model.

5. Define Metrics Computation Function
   - Function '*compute-metrics'* is designed to evaluate the performance of the model similar to Experiment 1.
6. Set Up Training Arguments
   - Training arguments such as the number of epochs, batch size, and learning rate are defined using the *'TrainingArguments'* class.
7. Custom Loss Functions
   - **System 1: Cross Entropy Loss**
     - For this system, we will use the Hugging Face '*Trainer*' class without customizing it for our loss function because by default 'Trainer' class uses cross-entropy loss. It internally computed the loss between predicted probabilities and the true labels.
   - **System 2: Label Smoothing**
     - Initialize the Trainer:
       - For this system, a '*CustomTrainer*' class is defined, inheriting from the '*Trainer'* class. This allows for the use of a custom loss function during training.
     - Custom Loss Function:
       - Class Name: *'LabelSmoothingCrossEntropy'*
       - Base Class: Inherits from '*nn.Module'*
       - Constructor('__init__')
         - Parameter: 'smoothing'
           - This parameter controls the degree of smoothing. For example, a value of 0.1 means that 10% of the confidence will be redistributed to other classes, reducing the model's confidence for easy to classify class.
       - Forward Method ('forward')
         - Parameters: '*input', 'target'*
           - *'input'*: The logits from the model
           - *'targe*t': The true labels for the data
         - Process:
           - Constructs a mask *('valid_labels_mask')* to identify valid labels which are not equal to -100.
           - Apply mask to filter out invalid labels and corresponding logits.
           - Next step is to compute log probabilities i.e. the log of softmax probabilities of the inputs. This is a standard step in cross-entropy loss.
           - Next up we adjust weights for label smoothing. Create weight tensor that initially assigns a small portion of probability to all classes as per smoothing parameter.
           - We adjust the weights such that the true class gets a higher probability (1-smoothing), while other classes share the remaining probability.
           - Next, we compute the loss (weighted negative log likelihood), and this is done by multiplying the adjusted weights by the log probs, summing over the classes, and then averaging over all samples in the batch.
           - '*criterion = LabelSmoothingCrossEntropy(smoothing=0.1)*': An instance of the class is created with a smoothing factor of 0.1 ready to be used as a loss function in the training loops.

   - **System 3: Focal Loss**
     - Initialize the Trainer:
       - For this system, we use the same *'CustomTrainer'* defined earlier.

- Custom Loss Function[3]:
  - Class Name: *'FocalLoss*
  - Base Class: Inherits from *'nn.Module'*
  - Constructor('__init__')
    - Parameters: '*alpha', 'gamma' & 'reduction'*
      - This parameters *'alpha'* and *'gamma'* control the class weighting and focusing parameter respectively. The '*reduction*' parameter specifies how loss should be aggregated (either 'mean' or 'sum')
  - Forward Method ('forward')
    - Parameters: '*input', 'target'*
      - '*input'*: The logits from the model
      - '*target'*: The true labels for the data
    - Process:
      - Like Label Smoothing, we first compute log probabilities i.e. the log of softmax probabilities of the inputs. This operation converts logits to log probabilities.
      - A mask is created to filter out any special tokens or padded areas in the targets (like '-100').
      - Using 'gather' method, the log probs corresponding to the actual target classes are extracted. This step aligns the predicted probabilities with their ground truth labels.
      - Now we calculate the focal loss with the formula *'-self.alpha * (1 - probs) ** self.gamma * targets_log_probs'*. Here 'probs' are the probabilities of the target classes derived from log probabilities. *'(1-probs) ** self.gamma'* scales the log probs based on the correctness of the classification – scaling is higher for incorrect misclassifications (hard examples).
      - The calculated loss in then multiplied by mask to make sure no loss is calculated for the padded positions.
      - Finally, the loss values are aggregated according to the *'reduction'* parameter to produce a single scalar that represents the loss for the batch.
      - *'criterion = FocalLoss(alpha=0.25, gamma=2.0)'*: An instance of the class is created ready to be used as a loss function in the training loops.

8. Training the Model with Pre-processed Data
   - The model is trained by calling the '*trainer.train()'* method of the trainer.
9. Evaluate the model with the Validation Dataset
   - Model is evaluated on validation dataset with '*trainer.evaluate()*' and various metrics including precision, recall, F1-score, and accuracy are computed using the '*compute-metrics'* function.
10. Obtain Predictions on Test Set
    - The test dataset is prepared the same way, we obtain true predictions and true labels and then compute metrics.
    - Model predictions are obtained and evaluated using '*trainer.predict()'*.
11. Visualization and Error Analysis
    - A classification report is generated using the '*seqeval_classification_report'* function from seqeval.metrics.
    - Functions are defined for visualizing the Confusion Matrix, Error Distribution, and Error Types across different NER classes.

[3] https://github.com/kornia/kornia/blob/main/kornia/losses/focal.py

In the provided code, I've adopted certain functions and structures inspired by a similar implementation. The original source of inspiration can be found here.[4]

Note: The default trainer in the experiment employs Cross Entropy Loss unless customized. The custom trainer was utilized to incorporate Label Smoothing Cross Entropy and Focal Loss into training process for comparative analysis.

2.3 Comparing Different Word Embeddings

In this experiment, I chose to compare the effectiveness of different word embeddings in the context of Named Entity Recognition tasks and to assess their impact on model performance. To conduct this comparison, I experimented with different word embeddings like Word2Vec and FastText.

For comparing word embeddings, I will be implementing:

System 1: *Word2Vec Embeddings*

vs

System 2: *FastText Embeddings*

2.3.1 Rationale for Word Embeddings:

1. Word2Vec: Word2Vec, developed by researchers at Google, consists of Skip-Gram and CBOW (Continuous Bag of Words) models that efficiently produce word embeddings. Word2Vec is effective when it comes to capturing semantic relationships among words, which is essential for identifying entities accurately. It's ability to generate word embeddings based on contextual information helps NER models understand the surrounding words and their semantic significance, which helps in entity recognition.

2. FastText: FastText, from Facebook's AI Research lab, extends Word2Vec by learning vectors for sub-words (n-grams) alongside words. By incorporating sub-words information, FastText can generate better word embeddings for rare words which might be useful for us since our data contains a lot of rare words as seen during EDA.

2.3.2 Experimental setup:

Preprocessing: The same preprocessing steps will be applied uniformly across all systems. Preprocessing steps will include lowercase conversion, stopword removal, and punctuation removal.

Model: The model of choice is a Bidirectional Long Short-Term Memory (BiLSTM) Model implemented using TensorFlow's Keras API.

Hyperparameters: We will use the same hyperparameters, including batch size, learning rate, and number of epochs, keeping them constant across all systems for a fair comparison.

2.1.3 Experimental Procedure:

- Preprocess the data.
- Train a Word2Vec model and FastText model using tokens from dataset.
- Train the model using the prepared datasets for a fixed number of epochs and batch size and monitor its performance on validation dataset.
- Evaluate and assess the performance of model using SeqEval, prioritizing the F1 score as the primary evaluation metric.

[4] For more information, check out the GitHub repository for PLOD Abbreviation Detection.

- Obtain predictions on the test dataset for a final assessment to assess generalization performance.

2.1.4 Detailed Analysis of the Code:

Here's a step-by-step breakdown of the code for all the systems:

1. Load the Dataset
   - The script begins by loading the PLOD-CW dataset using the *'load_dataset'* function from the Hugging Face Transformers Library
2. Data Preprocessing
   - The dataset is pre-processed by removing stopwords and punctuations and converting tokens to lowercase.
3. Creating Label Mapping
   - Create mappings between labels and their corresponding indices.
4. Dataset Preparation for Training
   - A '*prepare_dataset'* function is defined that converts the dataset into a format suitable for training a neural network model and returns input (X) and output (y) data.
   - This function converts the words in the 'tokens' to their corresponding indices in the Word2Vec or FastText model.
   - Pads the sequences to the fixed length.
   - Transforms the '*ner_tags'* to one-hot encoded labels.
5. BiLSTM Model Architecture
   - BiLSTM Model is defined with model architecture as follows:
     - *Embedding Layer*: This layer utilizes the pre-trained word embedding (Word2Vec/FastText) model to convert words into embeddings.
     - *Bidirectional LSTM Layer*: This layer processes the input sequence from both directions i.e. forward and backward. BiLSTMs are particularly useful in sequence labelling tasks like NER because they provide additional context from both directions of the sequence.
     - *Dense and Dropout Layer*: These layers are used for prediction and regularization respectively.
     - *TimeDistributed Dense Layer*: Applies a dense layer to every timestep of the LSTM output which allows the model to make predictions at each step of the input sequence.
   - The model is then trained on prepared training and validation datasets, using the embeddings and the model architecture defined.
6. Word Embedding Model Training
   - **System 1: Word2Vec Embedding**
     i. A Word2Vec model is trained from the pre-processed train dataset tokens with various hyperparameters such as number of features, minimum word count, window size, and subsampling rate.
     ii. The trained model is saved to a file named '*my_word2vec_model'* and later loaded for use in NER model.
   - **System 2: FastText Embedding**
     i. A FastText model is trained from the pre-processed train dataset tokens with various hyperparameters such as number of features, minimum word count, window size, using skip-gram approach for better handling of infrequent words.
     ii. The minimum and maximum length of character n-gram set to 3 and 6 respectively. This allows FastText to generate better representations for rare words using their sub-word information.
7. Model Definition and Compilation
   - Creating an instance of BiLSTM Model with Word2Vec/FastText Embeddings and compile the model.
8. Dataset Preparation for Training

- The datasets are prepared by calling *'prepare_dataset'* function which converts the text data into word embeddings using pre-trained Word2Vec/FastText embeddings.

9. Model Training
    - The model is then trained on prepared training and validation datasets, using the embeddings and the model architecture defined.
10. Predictions on Test Set
    - The trained model makes predictions on the test set, which is then used for evaluation.
    - The performance metrics (loss and accuracy) are reported, and a detailed classification report is generated using the 'seqeval_classification_report' function from seqeval.metrics to assess model's effectiveness in correctly identifying and classifying named entities.
11. Visualization and Error Analysis
    - A classification report is generated using the *'seqeval_classification_report'* function from seqeval.metrics.
    - Functions are defined for visualizing the Confusion Matrix, Error Distribution, and Error Types across different NER classes.

Note: After training the Word2Vec based model, the script repeats the process using a FastText based model. This allows for a comparison of performance between the two embeddings techniques.

2.4 Extracting Additional Train/Validate Dataset from PLOD-Filtered

In this experiment, I decided to extract additional train and validate dataset from additional PLOD-Filtered dataset and combine it with the PLOD-CW dataset. The reason for this was to explore how different proportions of subsets extracted and combined with the original dataset (PLOD-CW) affect the accuracy and overall metrics.

For this experiment, I will be implementing:

System 1: *Train solely on PLOD-CW*

vs

System 2: *Additional 1% of data from PLOD-Filtered combined with PLOD-CW*

vs

System 3: *Additional 5% of data from PLOD-Filtered combined with PLOD-CW*

2.4.1 Rationale

The decision to extract additional data from the optional dataset and concatenate with the existing PLOD-CW is driven by the pursuit of potential improvement in model performance. While there is no guarantee of significant enhancement, the inclusion of additional data might still provide valuable information or capture more diverse patterns in the data.

2.4.2 Experimental setup:

Upon analysis of the conducted experiments, it was evident that the most promising system among all is the one from Experiment 2 – System 2. This system, employing Label Smoothing with Cross Entropy, gave the most promising results so considering these findings, setup from Experiment 2, System 2 will be the baseline for this experiment.

Since we're already covered the experiment setup earlier in the report, this section will mainly dive into how we're getting our data ready and combining different datasets for analysis.

2.4.3 Detailed Analysis of the Code:

Here's a step-by-step breakdown of the code for all the systems:

Note: In the analysis I will be covering only the data preparation steps and skipping the model setup explanation.

1. Load the Dataset
   - The script begins by loading both PLOD-CW and PLOD-Filtered[5] (Additional Dataset) datasets using the *'load_dataset'* function from the Hugging Face Transformers Library
2. Displaying Rows
   - Function to display random rows for both the datasets in defined with the intention of studying the contents of both the datasets to see if the features align or not before extraction and concatenation.
   - After displaying the data, we notice that the ner_tags and pos_tags of PLOD-Filtered are encoded.
3. Encoding POS tags and NER tags.
   - For POS tags, I decided to encode the POS tags in the PLOD-CW dataset using Label Encoder.
   - For NER tags, I converted the encoded tags in PLOD-Filtered to their string label.
   - Next up I checked if the features in both the datasets were aligned and noticed that the pos_tags in both the datasets were different. So, I modified the feature type for pos_tags to int64 in PLOD-CW .
4. Extraction and Concatenation
   - Adding Tuple Representation
     i. Since neither of the datasets had a unique identifier column that could be used to directly identify and remove duplicate entries, using tuple representations of token sequences allows to create unique fingerprints for each data instance.
     ii. Each tuple generated from the tokens in one dataset uniquely represents a specific combination of tokens. This uniqueness ensures that using these tuples will help filter duplicates.
   - Removing Duplicate Entries from PLOD-Filtered
     i. Here we collect token tuples from PLOD-CW datasets and use these tuples to filter duplicates in the other dataset.
   - Function to Extract Subset
     i. The '*extract_subset'* function takes a dataset and extracts a subset of a specified size.
     ii. The function utilizes the '*train_test_split'* method to split the dataset and uses the specified 'test size' parameter as '*subset_size'*.

   Next steps are similar to Experiment 2: System 2

   System 1: Additional 1% of data from PLOD-Filtered combined with PLOD-CW

   - Extraction and Concatenation
     - Here we call the '*extract_subset'* function with the subset size (0.1) which then returns the train and validation subsets from PLOD-Filtered.
     - Using '*concatenate_datasets'* from datasets library, we concatenate the train_subset and validation_subset with the PLOD-CW train and validation.
     - We then tokenize the concatenated datasets.
     - The next steps are similar to the model training steps, evaluation and predictions.

   System 2: Additional 5% of data from PLOD-Filtered combined with PLOD-CW

   - The extract subset function is called with subset size of 0.5.

Note: Due to limited computational resources and processing capabilities, only 1% and 5% of the large dataset were extracted for experimentation purposes.

[5] surrey-nlp/PLOD-filtered · Datasets at Hugging Face

SECTION 3

EXPERIMENTAL ANALYSIS (TESTING AND ERROR ANALYSIS)

3.1 EXPERIMENT 1: Comparing Different Preprocessing Techniques

➢ **Evaluation Results on Validation Set:**

- System 1 (Lowercasing) showed relatively high performance with an evaluation loss of 0.2163, precision of 93.58%, recall of 92.92%, F1-score of 93.25%, and accuracy of 92.85%.
- System 2 (Stopword and Punctuation Removal) resulted in an evaluation loss of 0.2649, precision of 93.52%, recall of 93.27%, F1-score of 93.39%, and accuracy of 93.02%.
- System 3 (Lemmatization) led to an evaluation loss of 0.3383, precision of 93.48%, recall of 93.32%, F1-score of 93.39%, and an accuracy of 92.91%.
- System 4 (Combination of all Preprocessing Steps) resulted in an evaluation loss of 0.3843, precision of 93.54%, recall of 93.45%, F1-score of 93.50%, and accuracy of 92.99%.

➢ **Predictions on Test Set:**

Details of Precision (P), Recall (R) and F1-score (F) for Abbreviations (AC) and Long-Form (LF).

	Abbreviations			Long-Forms			Other		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
SYSTEM 1	0.77	0.84	0.80	0.65	0.73	0.69	0.97	0.94	0.95
SYSTEM 2	0.76	0.83	0.79	0.64	0.73	0.68	0.96	0.94	0.95
SYSTEM 3	0.76	0.84	0.79	0.65	0.76	0.70	0.96	0.94	0.95
SYSTEM 4	0.77	0.84	0.80	0.65	0.73	0.69	0.97	0.94	0.95

➢ **Error Analysis:**

1. **Confusion Matrix:** It shows the number of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) for each class in the model's classification task.

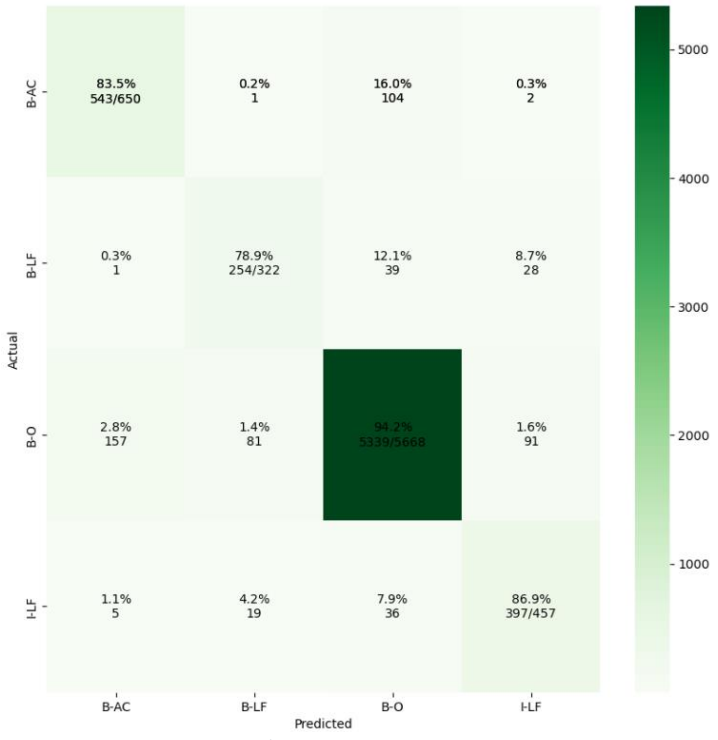


Fig. System 1

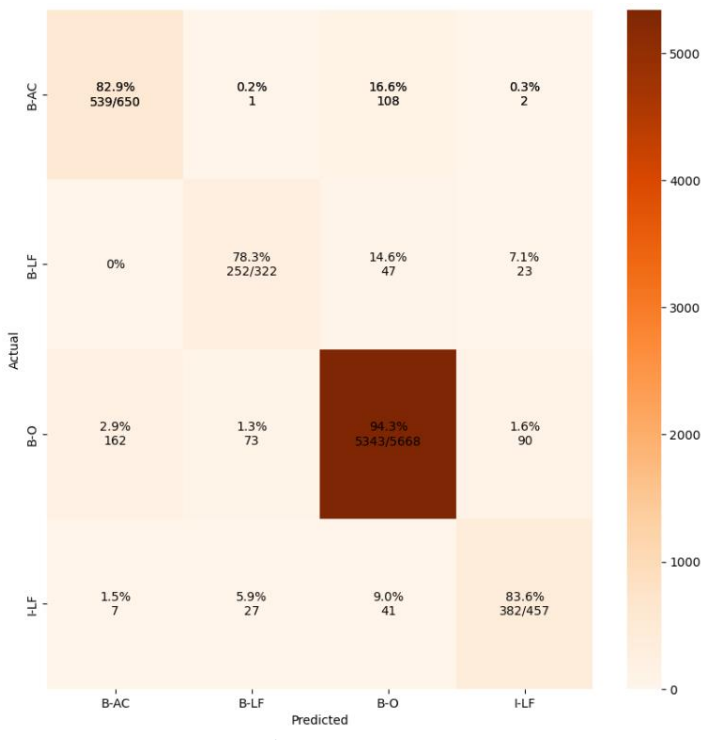


Fig. System 2

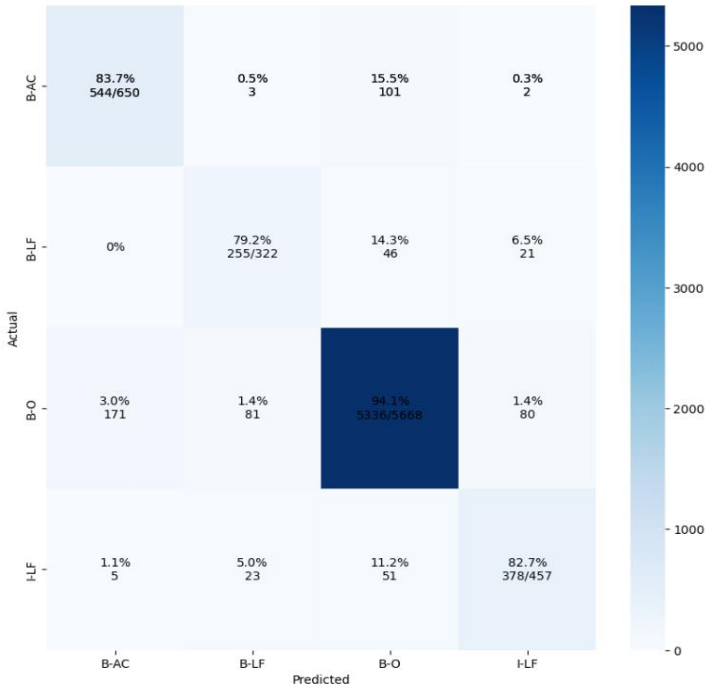


Fig. System 3

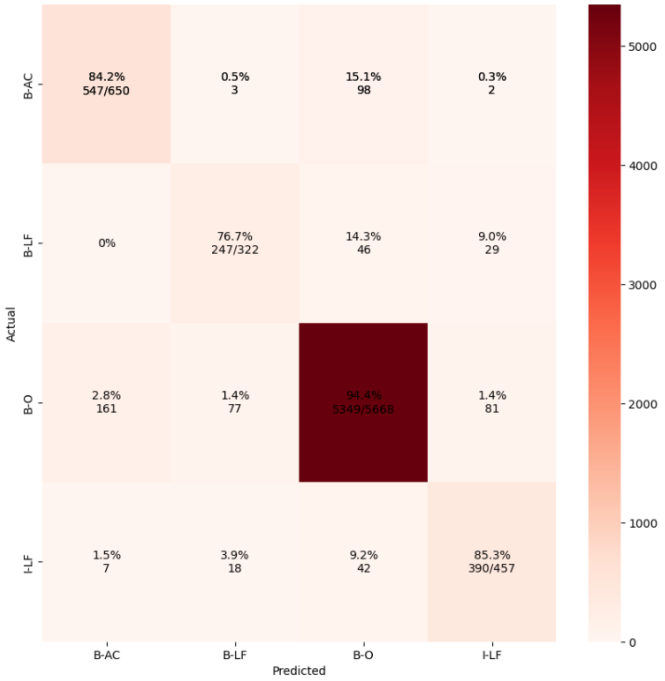


Fig. System 4

2. Error Type Analysis:

B-AC		B-LF		I-LF		B-O	
SYSTEM 1							
TP	53	**TP**	254	**TP**	397	**TP**	5339
TN	6284	**TN**	6674	**TN**	6519	**TN**	1250
FP	163	**FP**	101	**FP**	121	**FP**	179
FN	107	**FN**	68	**FN**	60	**FN**	329
SYSTEM 2							
TP	539	**TP**	252	**TP**	382	**TP**	5343
TN	6278	**TN**	6674	**TN**	6525	**TN**	1233
FP	169	**FP**	101	**FP**	115	**FP**	196
FN	111	**FN**	70	**FN**	75	**FN**	325
SYSTEM 3							
TP	544	**TP**	255	**TP**	378	**TP**	5336
TN	6271	**TN**	6668	**TN**	6537	**TN**	1231
FP	176	**FP**	107	**FP**	103	**FP**	198
FN	106	**FN**	67	**FN**	79	**FN**	332
SYSTEM 4							
TP	547	**TP**	247	**TP**	390	**TP**	5349
TN	6279	**TN**	6677	**TN**	6528	**TN**	1243
FP	168	**FP**	98	**FP**	112	**FP**	186
FN	103	**FN**	75	**FN**	67	**FN**	319

a. All preprocessing techniques (System 4) contribute to reducing false positives and false negatives across all labels. This shows that these techniques help improve model performance by reducing the number of misclassifications.

b. Across all systems, class “O” tends to have to have the highest number of true positives followed by “AC” and “LF”.

c. While false positives and false negatives are generally reduced with preprocessing, there are slight variations in performance between the different methods. System 3 (Lemmatization) appears to have a slightly higher false positive rate compared to other systems.

d. Overall, System 4 seems to offer the best balance between reducing false positives and false negatives while maximizing the true positives.

3. <u>Visualization of Error Types:</u>

The visualization reveals a nearly uniform distribution of true positives, true negatives, false positives, and false negatives with slight variations.

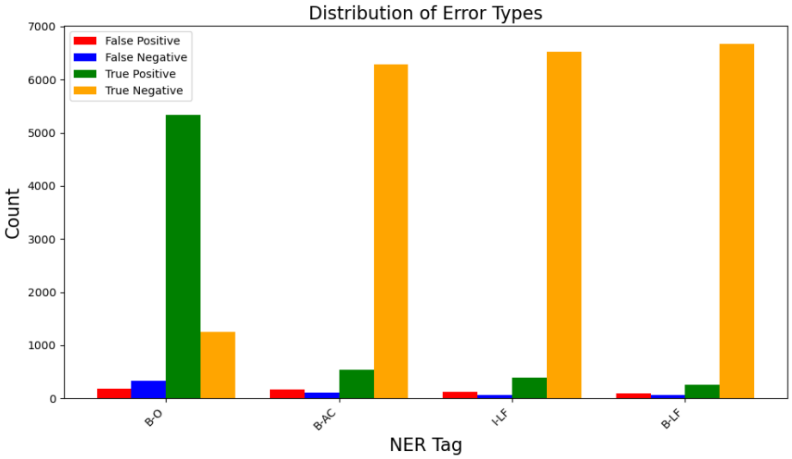


Fig. System 1

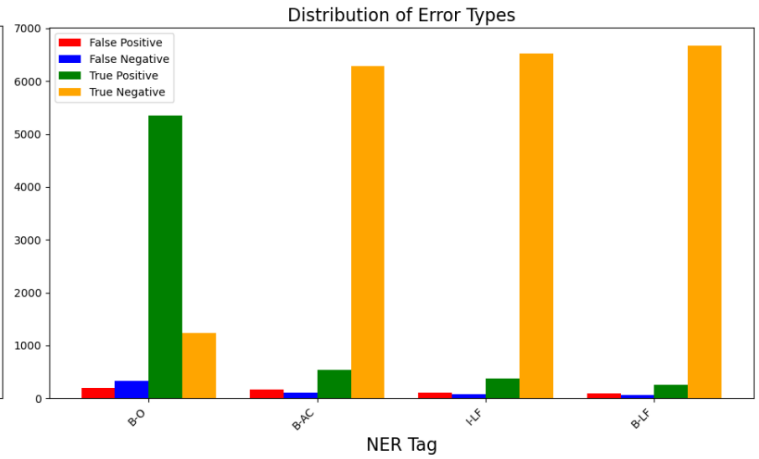


Fig. System 2

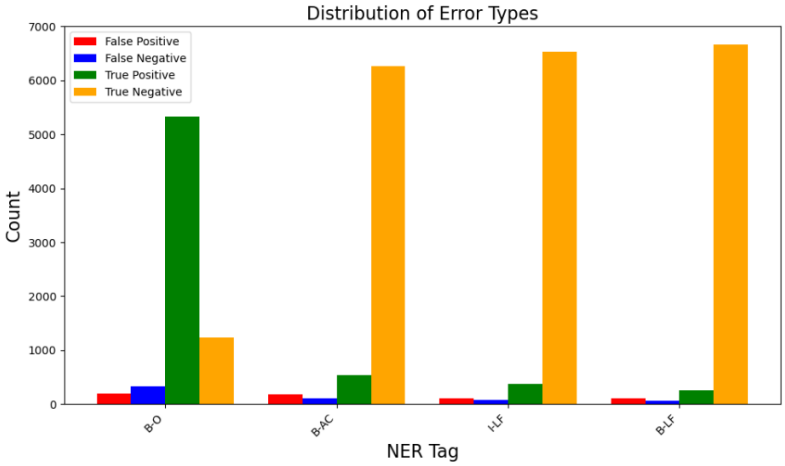


Fig. System 3

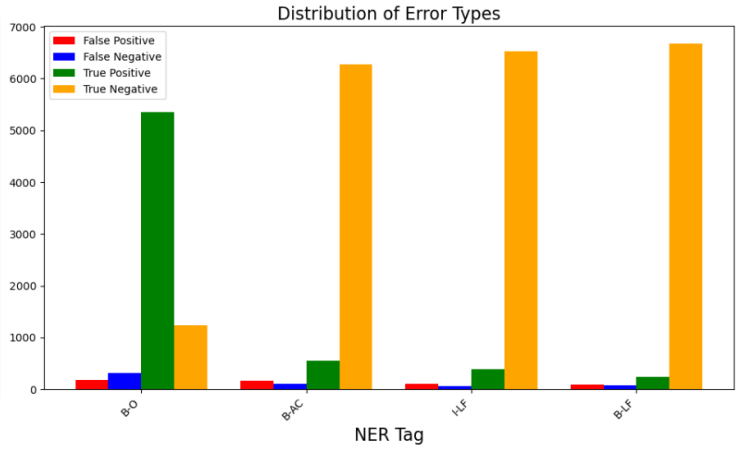


Fig. System 4

4. <u>Error Distribution by Class:</u>

This chart shows the error rate for each class. It appears that B-LF has the highest error rate, which might indicate difficulties in correctly classifying this tag, followed by B-O.

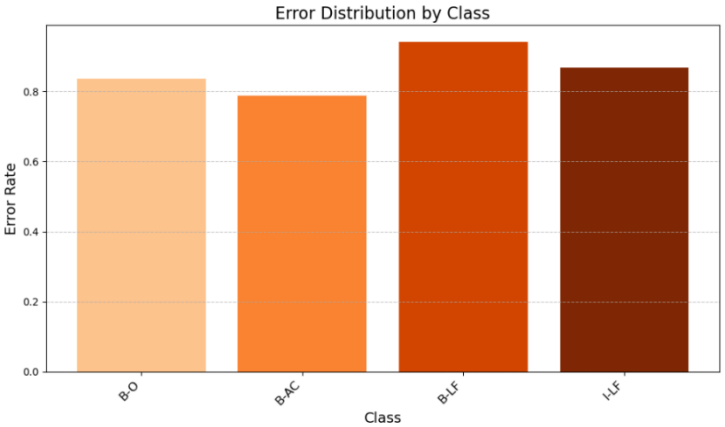


Fig. System 1

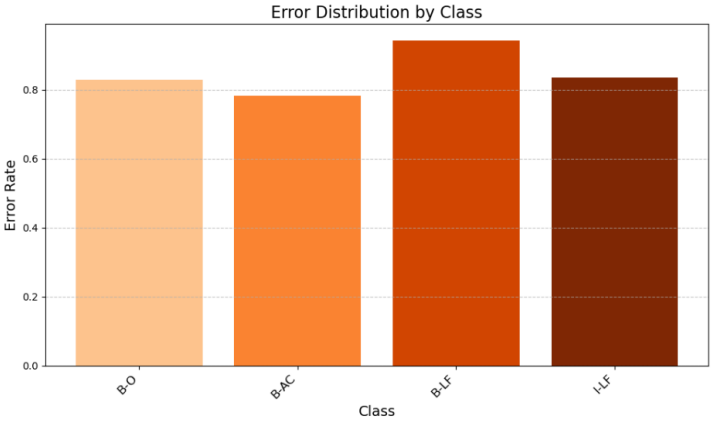


Fig. System 2

Fig. System 3

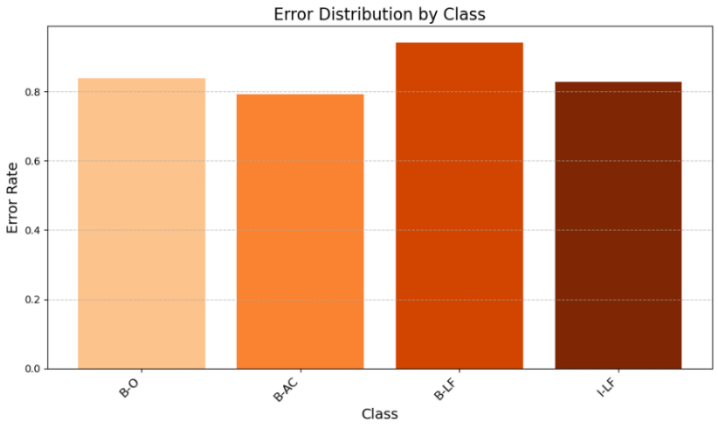
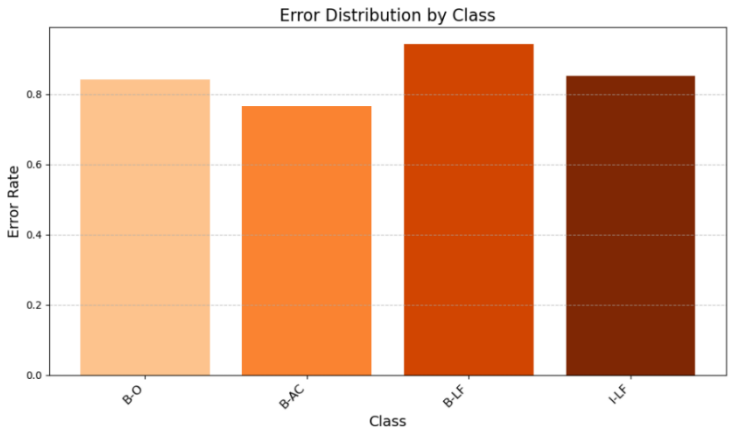


Fig. System 4



3.2 EXPERIMENT 2: Comparing Different Loss Functions

➢ **Evaluation Results on Validation Set:**

- System 1 (Cross Entropy Loss) resulted in an evaluation loss of 0.1827, precision of 94.74%, recall of 94.13%, F1-score of 94.43%, and accuracy of 93.91%.
- System 2 (Label Smoothing Cross Entropy) resulted in an evaluation loss of 0.5543, precision of 95.42%, recall of 94.84%, F1-score of 95.13%, and accuracy of 94.46%.
- System 3 (Focal Loss) led to an evaluation loss of 0.0228, precision of 95.07%, recall of 94.62%, F1-score of 94.85%, and an accuracy of 94.09%.

➢ **Predictions on Test Set:**

Details of Precision (P), Recall (R) and F1-score (F) for Abbreviations (AC) and Long-Form (LF).

	Abbreviations			Long-Forms			Other		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
SYSTEM 1	0.78	0.84	0.81	0.70	0.79	0.75	0.97	0.95	0.96
SYSTEM 2	0.78	0.85	0.81	0.74	0.79	0.76	0.97	0.95	0.96
SYSTEM 3	0.75	0.86	0.80	0.72	0.80	0.76	0.97	0.94	0.96

➢ **Error Analysis:**

1. **<u>Confusion Matrix:</u>**

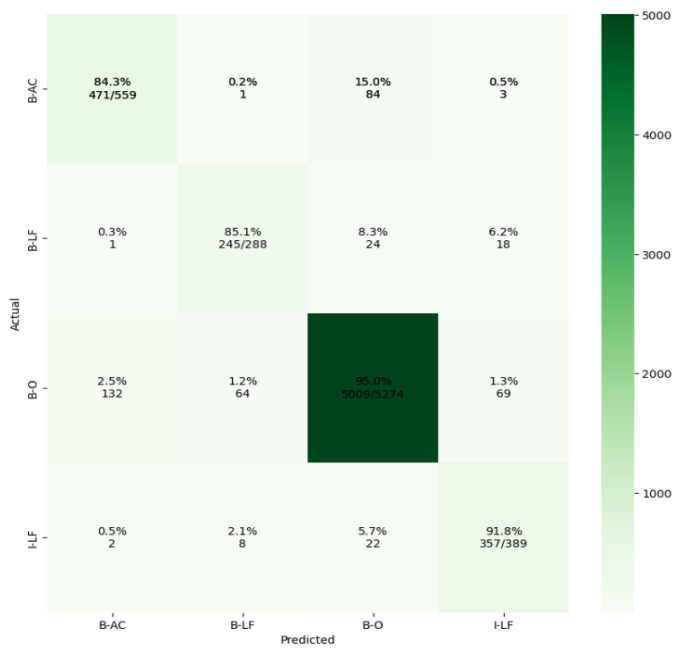


Fig. System 1

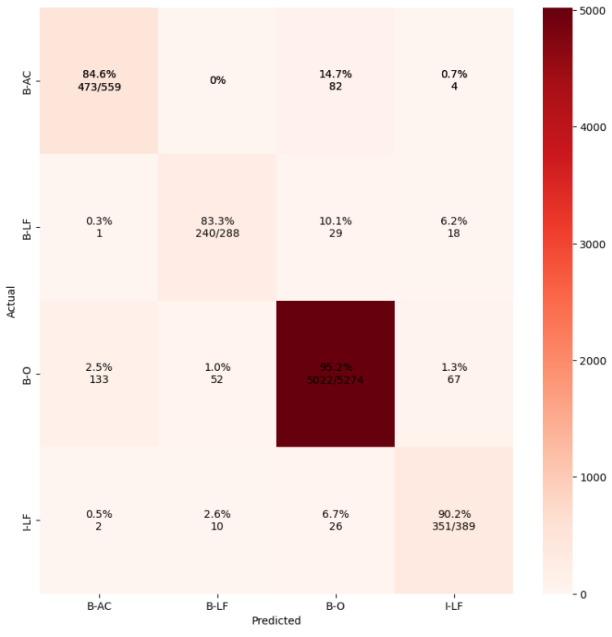
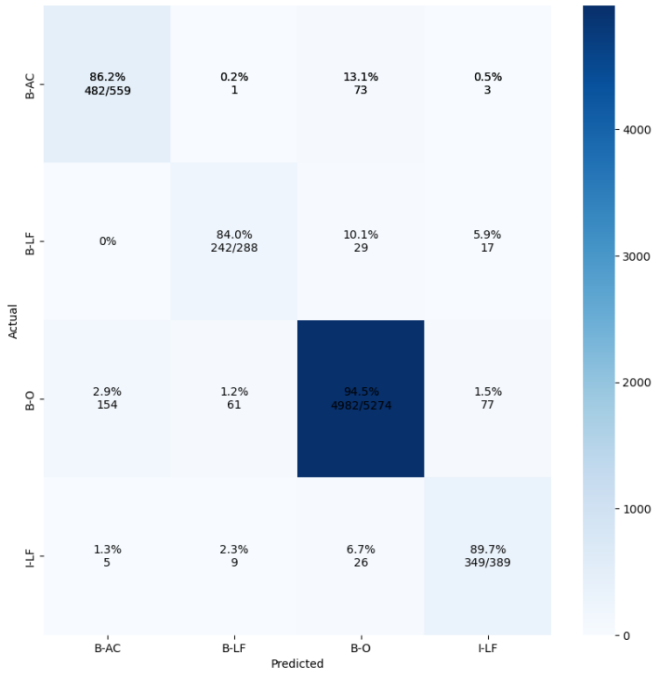


Fig. System 2

Fig. System 3



2. **Error Type Analysis:**
   a. Across different systems (loss functions), true positives, true negatives, false positives, and false negatives show similar distributions.
   b. The slight variation observed indicate nuanced differences in model performance, but overall comparable results are achieved.
   c. This consistence indicates that the choice of loss function does not significantly impact model's ability to correctly classify instances.

B-AC		B-LF		I-LF		B-O	
SYSTEM 1							
TP	471	**TP**	245	**TP**	357	**TP**	5000
TN	5816	**TN**	6149	**TN**	6031	**TN**	1106
FP	135	**FP**	73	**FP**	90	**FP**	130
FN	88	**FN**	43	**FN**	32	**FN**	265
SYSTEM 2							
TP	473	**TP**	240	**TP**	351	**TP**	5022
TN	5815	**TN**	6160	**TN**	6032	**TN**	1099
FP	136	**FP**	62	**FP**	89	**FP**	137
FN	86	**FN**	48	**FN**	38	**FN**	252
SYSTEM 3							
TP	482	**TP**	242	**TP**	349	**TP**	4982
TN	5792	**TN**	6151	**TN**	6024	**TN**	1108
FP	159	**FP**	71	**FP**	97	**FP**	128
FN	77	**FN**	46	**FN**	40	**FN**	292

3. **Visualization of Error Types:**

The visualization illustrates a consistent pattern across different loss functions. Despite minor variations in values, the overall similarity suggests that the choice of loss function has minimal impact on model performance in this context.

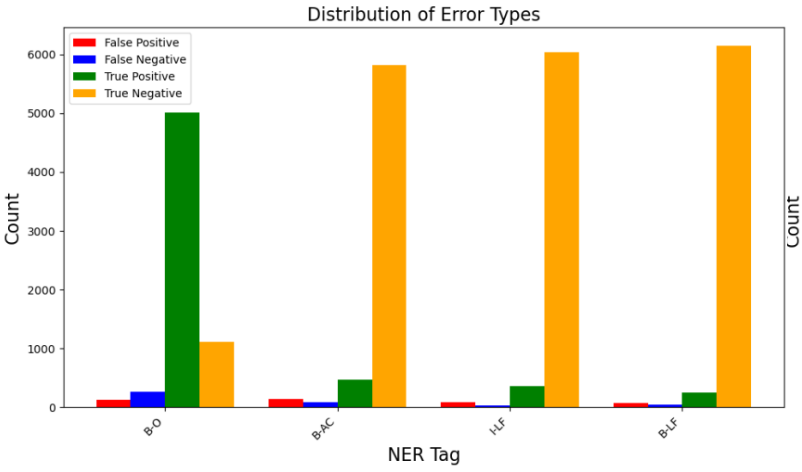


Fig. System 1

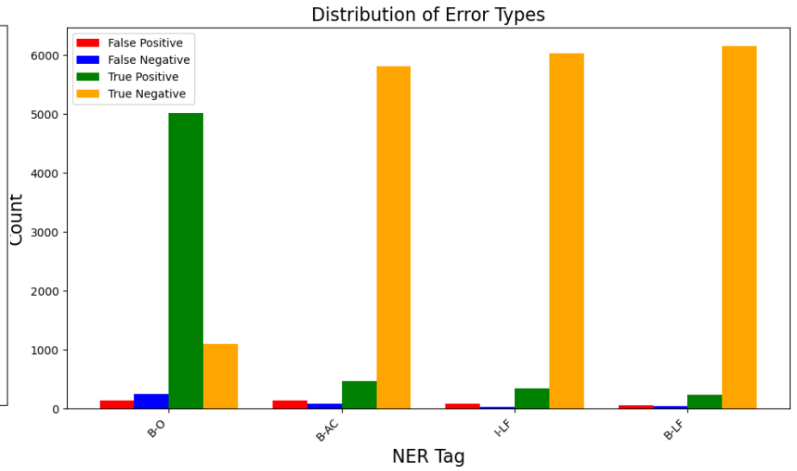


Fig. System 2

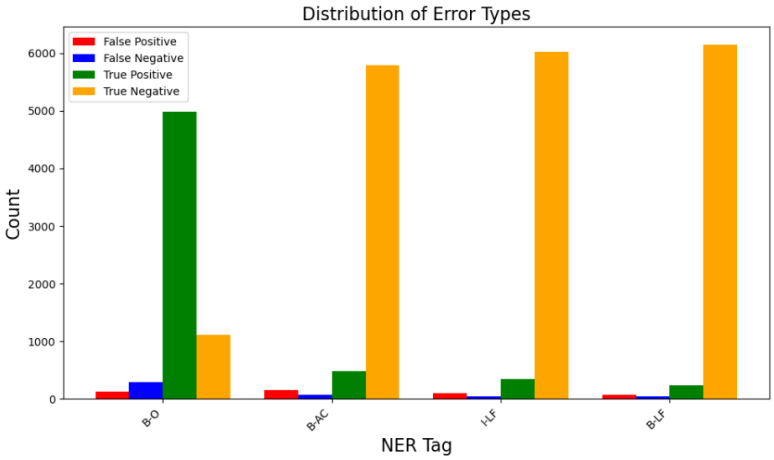


Fig. System 3

4. **Error Distribution by Class**

- The visualization shows a similar pattern in error rates across all systems.
- B-LF has the highest error rate, followed by I-LF

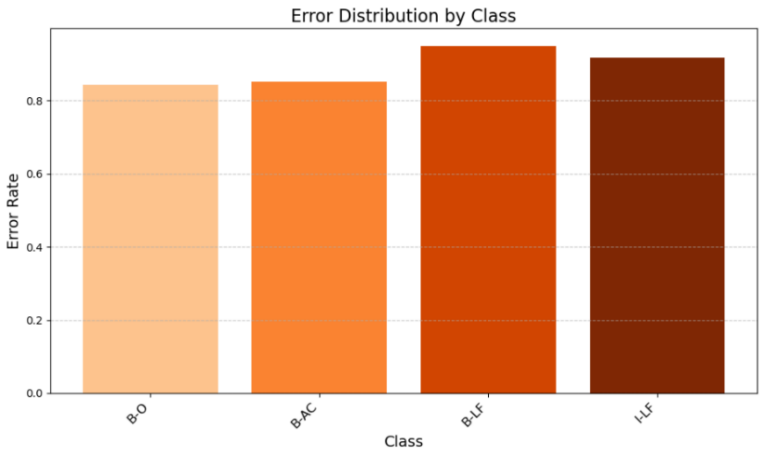


Fig. System 1

Fig. System 2

Fig. System 3

3.3 EXPERIMENT 3: Comparing Different Word Embeddings

➢ **Evaluation Results:**

- Despite different embeddings, both systems showed similar performance in terms of accuracy and loss.
- When analysing, precision, recall, and f1-score, discrepancies were observed. Both the system performed better on "O" as compared to "AC" and "LF" which is due to the data imbalance.
- System 1 achieved an accuracy of 94.56% and System with accuracy of 94.73%.
- The model underperformed greatly in NER if compared to systems in Experiment 1 and 2.

➢ **Predictions on Test Set:**

Details of Precision (P), Recall (R) and F1-score (F) for Abbreviations (AC) and Long-Form (LF).

	Abbreviations			Long-Forms			Other		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
SYSTEM 1	0.80	0.13	0.23	0.24	0.05	0.09	0.92	0.99	0.96
SYSTEM 2	0.77	0.15	0.25	0.21	0.05	0.08	0.92	0.99	0.96

➢ **Error Analysis: 1. <u>Confusion Matrix</u>:**

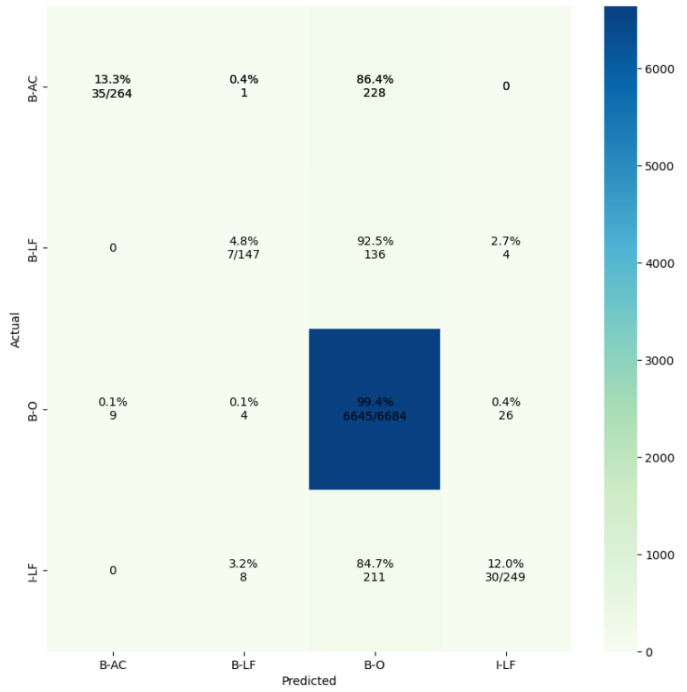


Fig. System 1

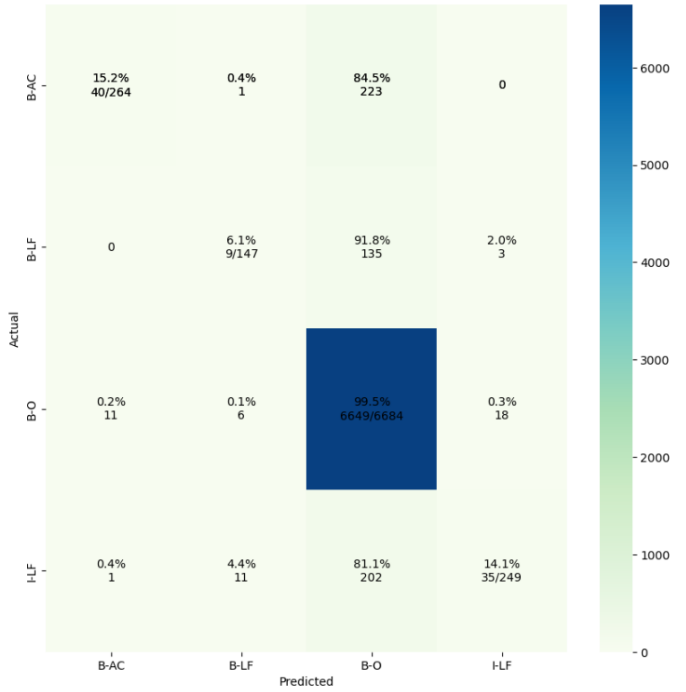


Fig. System 2

➢ **Error Type Analysis:**

a. Both systems showed a strong performance in accurately identifying “O” entities which is the majority class.

b. System 2 slightly outperformed System 1 in correctly classifying “AC” and “LF” entities.

c. System 1 however showed fewer false positives across all entity types.

B-AC		B-LF		I-LF		B-O	
SYSTEM 1							
TP	35	**TP**	7	**TP**	30	**TP**	6645
TN	7071	**TN**	7184	**TN**	7065	**TN**	85
FP	229	**FP**	140	**FP**	219	**FP**	39
FN	9	**FN**	13	**FN**	30	**FN**	575
SYSTEM 2							
TP	40	**TP**	9	**TP**	35	**TP**	6649
TN	7068	**TN**	7179	**TN**	7074	**TN**	100
FP	224	**FP**	138	**FP**	214	**FP**	35
FN	12	**FN**	18	**FN**	21	**FN**	560

➢ **Visualization of Error Types:**

The visualization illustrates a consistent pattern across different word embeddings. Despite minor variations in values, the overall similarity suggests that the choice of word embeddings has minimal impact on model performance in this context.

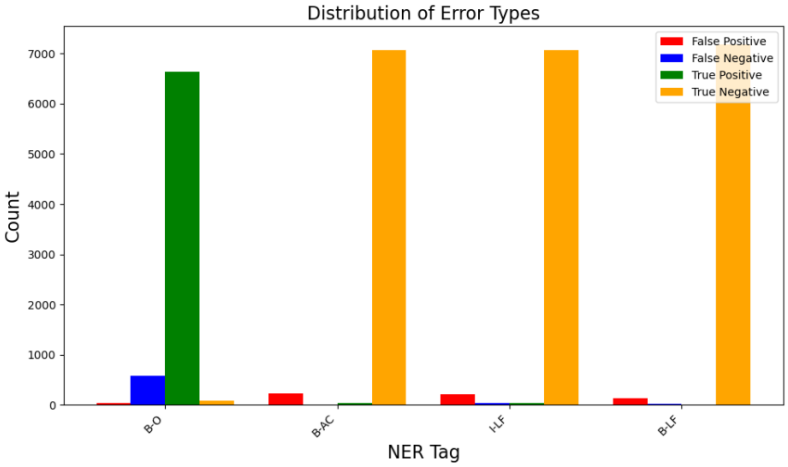


Fig. System 1

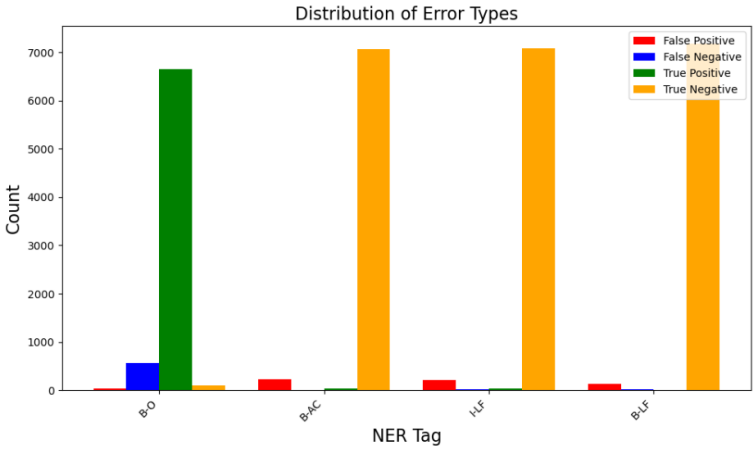


Fig. System 2

5. Error Distribution by Class

The distribution shows a very high error rate for I-LF for both the system which indicates difficulty in correctly identifying this tag.

Fig. System 1

Fig. System 2

3.4 EXPERIMENT 4: Extracting Additional Train/Validate Dataset From PLOD-Filtered

➢ **Evaluation Results on Validation Set:**

- System 1 resulted in an evaluation loss of 0.5543, precision of 95.42%, recall of 94.84%, F1-score of 95.13%, and accuracy of 94.46%.
- System 2 resulted in an evaluation loss of 0.5681, precision of 95.24%, recall of 94.47%, F1-score of 94.85%, and accuracy of 94.42%.
- System 3 led to an evaluation loss of 0.5754, precision of 94.63%, recall of 93.66%, F1-score of 94.14%, and an accuracy of 93.62%.

➢ **Predictions on Test Set:**

Details of Precision (P), Recall (R) and F1-score (F) for Abbreviations (AC) and Long-Form (LF).

	Abbreviations			Long-Forms			Other		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
SYSTEM 1	0.78	0.85	0.81	0.74	0.79	0.76	0.97	0.95	0.96
SYSTEM 2	0.89	0.87	0.88	0.79	0.86	0.82	0.99	0.97	0.97
SYSTEM 3	0.89	0.87	0.88	0.79	0.86	0.82	0.98	0.97	0.97

➢ Here from these observations, we see that the additional data did improve model performance but the addition of 1% and 5% additional from PLOD-Filtered to PLOD-CW dataset does not significantly impact the model's performance, as both setups yielded similar results.

➢ This similarity indicates that the model might have already learned sufficient information from the original dataset, and the slight increase in dataset size does not lead to any notable improvements in performance.

For System 2: Additional 1% of data from PLOD-Filtered combined with PLOD-CW

- *Evaluation results and predictions on the test set are almost identical to System 1 with a very slight improvement, indicating similar performance despite using a smaller dataset.*

For System 3: Additional 5% of data from PLOD-Filtered combined with PLOD-CW

- *Evaluation results and predictions on the test set remain consistent with the previous systems, suggesting that increasing the additional dataset size to 5% does not significantly impact performance.*

➢ **Error Analysis:**

1. **<u>Error Types:</u>**
   a. All the systems show strong performance with System 2 and 3 exhibiting strong performance in correctly identifying entities.
   b. System 2 and 3 slightly outperformed system 1 in identifying 'AC' and 'LF' entities, showing a better performance of the model.
   c. The high true negative rates suggest that all systems performed well but 2 and 3 performed a little better in accurately classifying.

B-AC		B-LF		I-LF		B-O	
SYSTEM 1							
TP	473	**TP**	240	**TP**	351	**TP**	5022
TN	5815	**TN**	6160	**TN**	6032	**TN**	1099
FP	136	**FP**	62	**FP**	89	**FP**	137
FN	86	**FN**	48	**FN**	38	**FN**	252

SYSTEM 2							
TP	486	**TP**	258	**TP**	365	**TP**	5098
TN	5892	**TN**	6170	**TN**	6034	**TN**	1131
FP	59	**FP**	52	**FP**	87	**FP**	105
FN	73	**FN**	30	**FN**	24	**FN**	176
SYSTEM 3							
TP	486	**TP**	258	**TP**	365	**TP**	5098
TN	5892	**TN**	6170	**TN**	6034	**TN**	1131
FP	59	**FP**	52	**FP**	87	**FP**	105
FN	73	**FN**	30	**FN**	24	**FN**	176

- **Visualizing Error Distribution:**

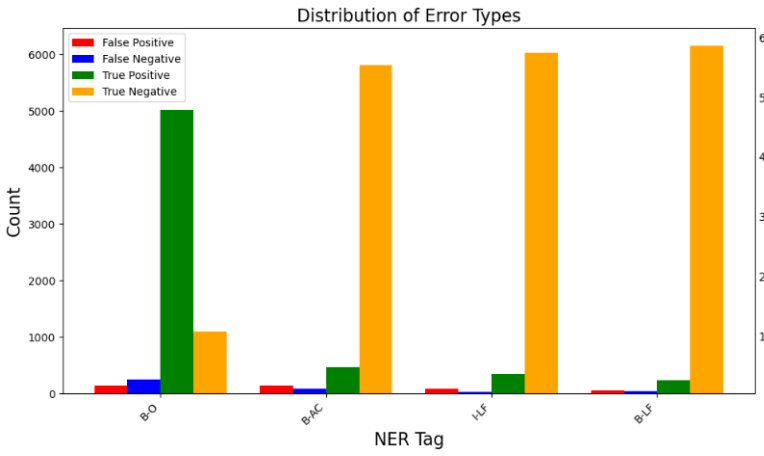


Fig. System 1

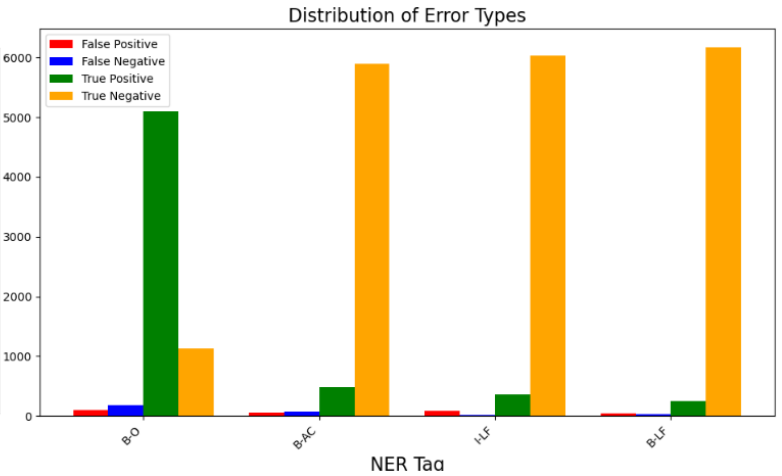


Fig. System 2 and 3

- **Confusion Matrix:** Confusion Matrix shows slight improvement in using Setup in System 2.

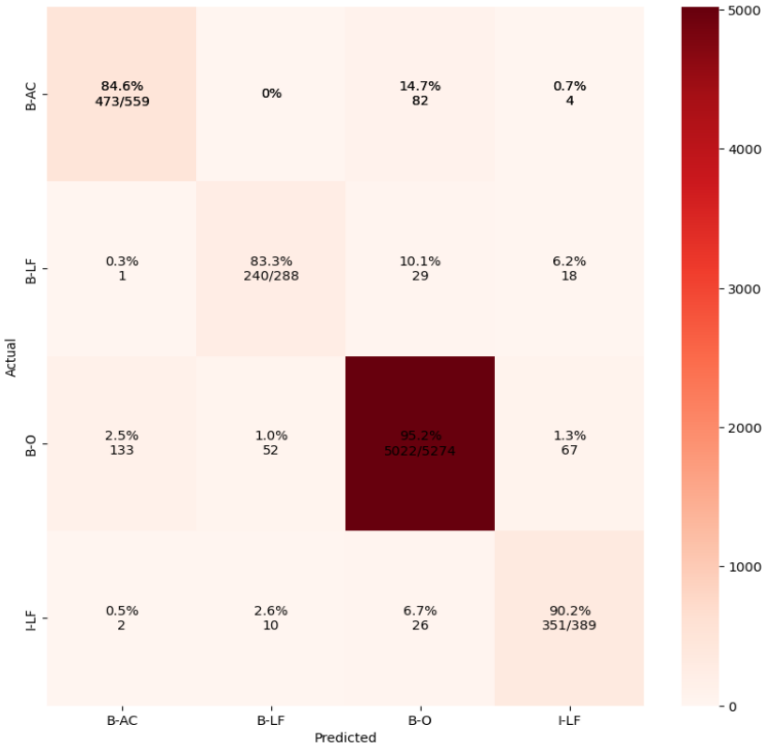


Fig. System 1

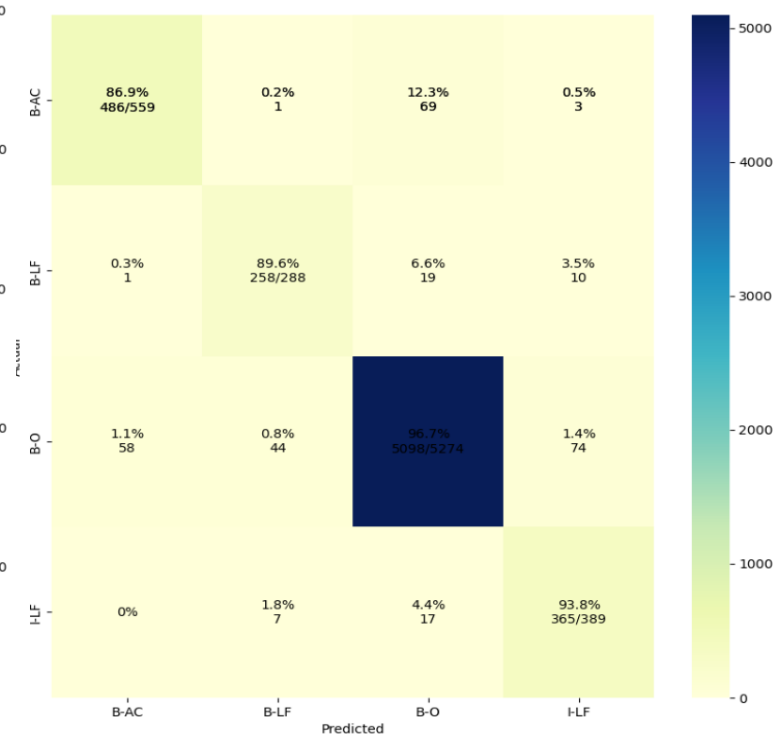


Fig. System 2

SECTION 4

DISCUSSING THE BEST RESULTS

4.1 EXPERIMENT – 1

System 1: *Preprocessing by Lowercasing*
System 2: *Preprocessing by Removing Stopwords and Punctuations*
System 3: *Preprocessing by Lemmatization*
System 4: *Preprocessing by Combining all the above*

- Based on the metrics, System 4 achieved the highest overall F1-score of 0.9252 in the test data.
- Combination of multiple preprocessing techniques effectively cleans the data and reduces noise.
- Overall System 4 offers the best performance in terms of accurately identifying named entities and recognizing entities. However, the rates of false positives and negatives are similar to other systems indicating areas for further improvement.

Best Result: These findings indicate that System 4 might be considered the best choice overall.

4.2 EXPERIMENT – 2

System 1: *Cross-Entropy Loss*
System 2: *Label Smoothing with Cross Entropy*
System 3: *Focal Loss*

- All systems perform well but System 2 stands out slightly with the highest F1-Score (0.951) compared to System 1 (0.944) and System 3 (0.948).
- System 2 and 3 also have a longer runtime.
- If computational efficiency is a priority without sacrificing quality, then System 1 is also a good setup. But here our goal is to maximise F1-score hence System 2 is a preferred choice.

Best Result: These findings indicate that System 2 might be considered the best choice overall.

4.3 EXPERIMENT – 3

System 1: *Word2Vec Embeddings*
System 2: *FastText Embeddings*

- Based on evaluation and testing, both systems achieved similar overall F1-scores on the test set, with System 2 slightly outperforming System 1.
- System 2 achieved an overall F1-score of around 0.93, which is higher than System 1, indicating that the model using FastText embeddings performed better in classifying named entities.

Best Result: These findings indicate that System 2 might be considered the best choice overall.

4.4 EXPERIMENT – 4

System 1: *Train solely on PLOD-CW*
System 2: *Additional 1% of data from PLOD-Filtered combined with PLOD-CW*
System 3: *Additional 5% of data from PLOD-Filtered combined with PLOD-CW*

- Based on the F1-scores, System 2 and System 3 have the highest F1-Score of 0.9572 on the test set, indicating that they perform equally well. However, System has a slightly higher F1-score on the validation set compared to System 3.

- In addition to having the highest F1-score, System 2, which also involves on a small subset from PLOD-Filtered, is computationally more efficient.
- Since it involves s smaller percentage of the dataset, it requires lesser time for training compared to System 3.

Best Result: These findings indicate that System 2 might be considered the best choice overall.

Overall, the best performing models across the experiments were:-

Experiment 1: System 4 (Combining all Preprocessing Steps)
Experiment 2: System 3 (Label Smoothing with Cross Entropy)
Experiment 3: System 2 (FastText Embedding)
Experiment 4: System 2 (Additional 1% of data from PLOD-Filtered combined with PLOD-CW)

Ways to Improve the Model for This Setup:

1. **Using BERT Variant:**
   - BERT has many variants out of which I have experimented with BERT-base and Roberta-base. Since the dataset has tokens related to medical and clinical research, using domain-specific pre-trained BERT models like BioBERT or ClinicalBERT may somewhat enhance the model's performance.
2. **Experimenting with Different Ratios:**
   - With better computational resources and GPUs, we can test with different ratios of PLOD-Filtered data to PLOD-CW data to find an optimal balance that might enhance model performance further.
3. **Hyperparameter Tuning and Optimization:**
   - Using Grid search or Bayesian Optimization to find the best model hyperparameters for the combined dataset. Fine-tuning hyperparameters such as learning rate, batch sizes, dropout rates etc can contribute to improvement.
4. **Sampling Techniques:**
   - Using sampling techniques when dealing with imbalanced datasets can help prevent model from being biased towards the majority class (which in the dataset is B-O).
5. **Adding Class Weights:**
   - Assigning different weights to each class for underperforming models (like models in Experiment 3) during training can help the model focus more on the minority class. This method is very useful when the dataset is highly imbalanced and minority classes are underrepresented.

SECTION 5

EVALUATING OVERALL ATTEMPT AND OUTCOME

A. Can the models fulfil their purpose?

- The models built in these experiments aim to perform Named Entity Recognition. They fulfil their purpose but show moderate to good performance for Abbreviations and Long-Forms. This suggests while the models are generally effective, there is still room for improvement.
- Experiment 1 and 2 models both show strong performance for non-named entities 'O' and good performance for named entities 'AC' and 'LF' but have room for improvement.
- Experiment 3 models struggle significantly with named entities, indicating it may not fully meet its intended purpose without further refinement.
- Experiment 4 models show excellent performance across all class labels, suggesting it effectively fulfils its purpose.

B. What is good enough F1/Accuracy?

- A good enough F1-score or accuracy can vary depending upon the task at hand. The F1-scores for AC and LF in Experiment 4, which are 0.88 and 0.82 respectively, without overall F1-score of 0.957 and overall accuracy 0.953 represent a strong benchmark for good enough performance in this context.

C. If any of the models did not perform well, what is needed to improve?

- Models in Experiment 3 needs significant improvement, especially in boosting its very low F1-scores for AC and LF.
- Techniques targeted towards handling of less frequent classes like class weighting, sampling methods or using a pre-trained word embeddings instead of training from scratch can enhance model performance.

D. If any of the models performed well, could/would you make it more efficient and sacrifice some quality?

- Systems in Experiment 4 shows the best overall performance, suggesting a slight increase in data can significantly enhance the model's effectiveness.
- In named-entity recognition tasks, both efficiency and quality are important.
- High quality models give accurate identification of named entities.
- In scenarios where computational resources are a concern, it may be beneficial to trade off a slight decrease in quality for improved efficiency.

Accuracy vs. Effectiveness

I. The choice between the most accurate and most effective solution is guided by the specific needs and requirements of the task.

II. In applications where accuracy is critical (like medical, financial, or legal documents), the most accurate and highest performing model (like those in Experiment 4) should be prioritized even if it's less efficient.

III. In less critical scenarios where the computational resources are limited, a slightly less accurate model but an efficient model would be ideal, especially if it leads to significant cost savings without drastically compromising performance.