NLP: Event Detection

Nazgul Mamasheva

1 Introduction

Event Detection (ED) is a natural language processing task that involves identifying and categorizing events within a given text or document. This task involves locating event triggers, which are words or phrases that indicate the occurrence of an event, and associating each token (word or subword) in the text with the corresponding event label. The goal is to recognize and classify different types of events, such as actions, changes, possessions, scenarios, sentiments, or any other predefined categories, depending on the specific context or application. For our task we are given 5 predefined labels: *action*, *change*, *possession*, *scenario* and *sentiment*. Event Detection is commonly used in information extraction and text analysis applications to extract meaningful insights from unstructured textual data.

2 Preprocessing

To prepare the data for training, several preprocessing steps are performed. First, tokens and labels are retrieved and stored separately from the corresponding datasets. The next step involves constructing vocabularies from the obtained tokens and labels. These vocabularies are exclusively built using the training dataset. The third step consists of encoding. Once our vocabularies are prepared, we proceed to encode our tokens and labels. In simpler terms, this involves converting them from textual values to numerical values, as the model comprehends and demands numerical inputs. The last step of preprocessing involves organizing batches and padding the samples within those batches. We specify the batch size and apply batching to inputs. When batching is performed, padding becomes necessary as the model demands inputs of same length in each batch. In these batches, our encoded tokens and labels from previous step serve as inputs, and since sentence lengths can differ, the padding function ensures the same length. It achieves this by extending shorter sentences with <*pad*> values to match the length of the longest sentence in the each batch.

3 Model

Our model consists of embedding, Bidirectional LSTM (Bi-LSTM), and Linear classifier layers.

3.1 Embedding

The initial layer in our model is the embedding layer, which can be constructed using pre-trained word embeddings. Word embeddings are trained on tons of data and in this way provide us the word vectors that represent semantic relationships and similarities between words. The similar and related words are assigned with vector representations that are close to each other. Utilizing pre-trained word embeddings can help our model in comprehending the connections and relationships between words. So, a pre-trained word embedding was utilized, to be specific, a Glove with a dimension size of 50. Glove offers 4 distinct pre-trained models with dimensions: 50, 100, 200, and 300. I experimented with all of them and found that utilizing Glove with the smallest dimension is sufficient for the embedding step, as the results of 4 dimensions were comparable the same. Moreover, the training process is faster with the smaller dimension compared to its higher dimensions. To perform embedding, we utilize our token vocabulary. Initially, we search each token from the token vocabulary in Glove. If the token exists in Glove, we assign its vector to the token; otherwise, we initialize it with random vectors. The token vocabulary contains around 38K unique tokens, with approximately 19.1K of them found in Glove and initialized with pre-trained vectors values, while the remaining 18.9K tokens were initialized randomly.

3.2 Base model

Event Detection involves recognizing and learning the features of an object to detect it. To address

this, Recurrent Neural Network (RNN) and its variant, Long short-term memory (LSTM), are suitable for recognition and detection tasks. LSTMs are good at maintaining information in memory for extended periods compared to RNNs. While RNNs are better suited for tasks like predicting the next word. The LSTM can function as unidirectional, preserving information only from past to future, or bidirectional (Bi-LSTM), preserving information in both directions - from past to future and from future to past, making it proficient at understanding the given text's context. Thus, the bidirectional LSTM (Bi-LSTM) is more fitting for our context-aware task. Additionally, the structure of the training dataset suggests choosing Bi-LSTM, as event triggers may span more than one token and thus implying that the model should be able to move backward to capture entire span of event triggers.

I trained the model with a base model as LSTM, Bi-LSTM, RNN, and Bi-RNN. The F1 scores and accuracy results for these models, trained with pre-trained Glove-50, are presented in Table 1. The F1 scores on the test dataset range from 0.68 to 0.70. As we can see, LSTM and RNN result similar performance, while bidirectional models outperformed their unidirectional counterparts. Specifically, Bi-LSTM demonstrated better performance compared to Bi-RNN. Based on these results, Bi-LSTM was selected as the base model. The number of layers was set to the default value of 1, as increasing the layer number from 1 to 2 didn't improve performance.

3.3 Classifier

The final layer in a model is the output layer that produces the model's predictions or outputs. It is responsible for transforming the learned features from the previous layers into the desired output format. The final layer of our model is a Linear classifier, since the model is required to produce a predicted label for each input token. Linear classifier makes predictions by applying a linear combination of input features. The input size of the classifier is a predetermined hidden size, set as 128 in our model, which is considered optimal, neither too small nor too large. In the case of a bidirectional base model, the input size becomes the double of the hidden size, i.e., 256, as the bidirectional module outputs forward and reverse hidden states. The output size of the final layer has to match the number of all possible labels in the label vocabulary, which is 12 (in BIO format) for our model. The classifier generates 12 logits for each token. By selecting the logit with a maximum value among each set of 12 logits for each token, we obtain predictions for input tokens. Subsequently, decoding is performed to convert these numerical values into understandable text labels.

4 Training

The training process utilizes the Cross-Entropy loss function, as it is appropriate for classification task where the goal is to optimize the model's predicted class probabilities to match the true class distribution in the training data, and our model falls under this category.

For optimizer we utilize Adam optimizer, as it is generally acknowledged as an effective optimizer, that provides good performance across various tasks and is widely used in practice.

The graph of training and validation losses is presented in Figure 1 with training number - 200 and batch size - 32. The figure illustrates that the model was trained faster and better during first 50 epochs and continued to perform well until around 100 epochs, then the learning process is gradually decreased.

5 Results

From confusion matrix represented in Figure 2, we can say that our trained model performed better in predicting "B-" tag (beginning of the event trigger's span) labels than "I-" tag (inside of the event trigger's span) labels, except I-SCENARIO label. The model correctly predicted B-ACTION by 68%, B-CHANGE by 74%, B-POSSESSION by 71%, B-SCENARIO by 68% and B-SENTIMENT by 66%. "B-" tag labels are predominantly mispredicted as O tag (outside of any span) label by 19%-28%. "I-" tag labels, except the I-SCENARIO, are largely mispredicted as O tag label by 86%-100%. We can conclude that our trained model is proficient at labeling the first word of event triggers rather than the subsequent words. The reason of this could be the limited occurrence of event triggers with inside spans in the training dataset.

6 Conclusion

As the results showed, Bi-LSTM with word embedding Glove-50 gives a good performance for our Event Detection task, achieving F1 score of 0.7035 and an accuracy of 0.9502.

Model	F1 score	Accuracy
LSTM	0.6840	0.9487
Bi-LSTM	0.7035	0.9502
RNN	0.6882	0.9482
Bi-RNN	0.6998	0.9489

Table 1: F1 score and Accuracy results of models.

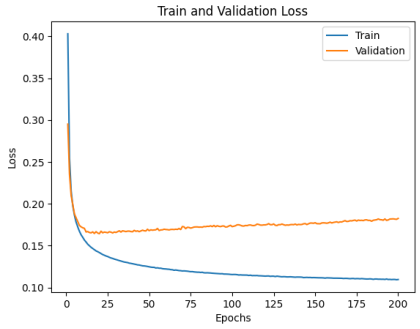

Figure 1: Train and Validation Losses.

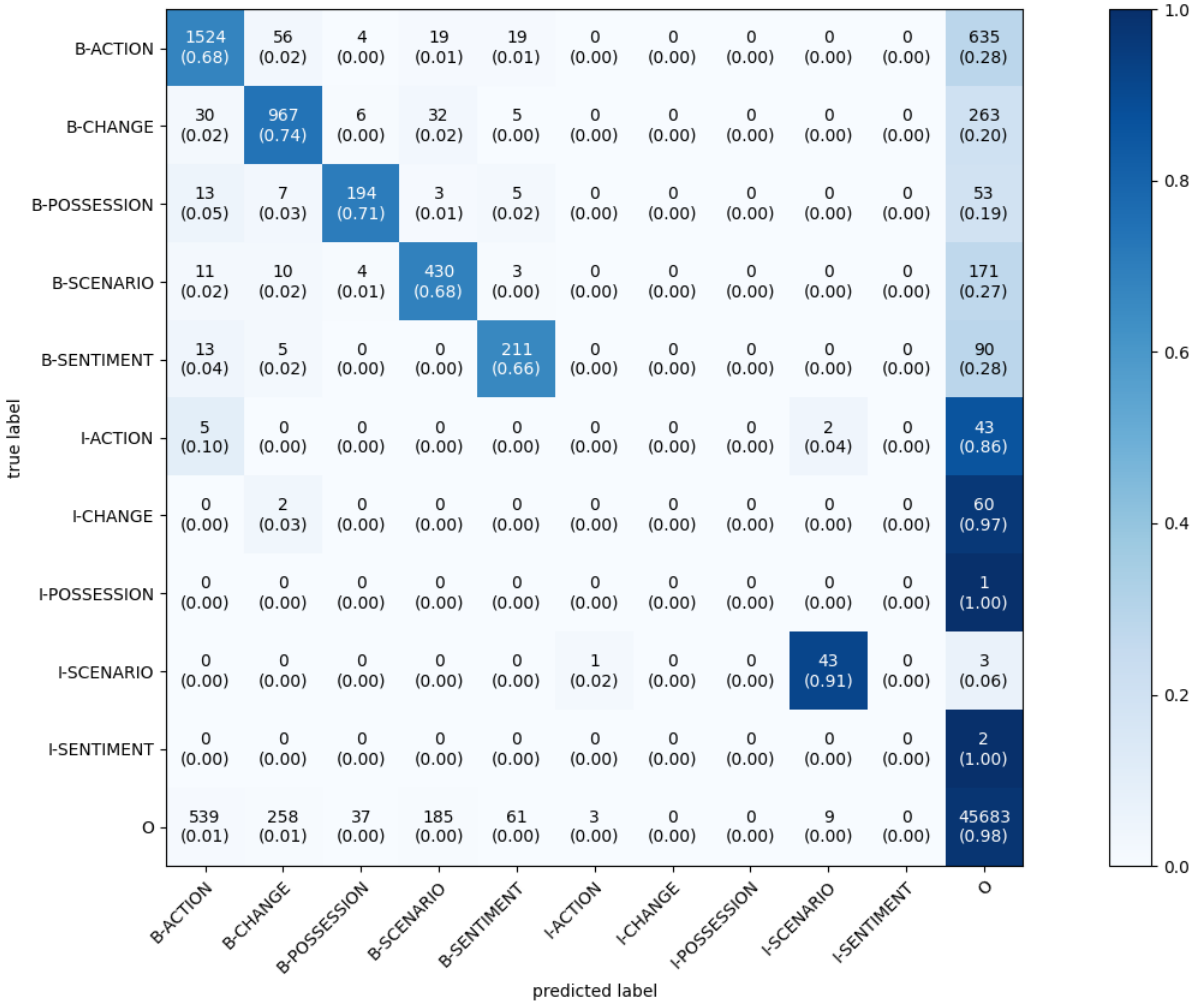

Figure 2: Confusion matrix on test dataset.