Named Entity Recognition Homework 1 - NLP 2022

Mirza Mujtaba Hussain
Email: mirza.1989740@studenti.uniroma1.it

1. Introduction

Named entity recognition (NER) is a classification task in the field of Natural Language Processing which aims at classifying the named entities in text or sentence into predefined categories such as groups, persons, locations, corporations, products, and creative work.In our case we are using the BIO tags ,thus we have 12 different named entities classes ('B-GRP', 'I-GRP', 'B-PER', 'I-PER', 'B-LOC', 'I-LOC', 'B-CW', 'I-CW', 'B-PROD', 'I-PROD', 'B-CORP','I-CORP') and one class (O) for all the other words. My approach to solve this problem was to use a baseline model which is a simple LSTM architecture. Later I used pre-trained GLOVE word embeddings while training which improved the results upto a certain extent.In the final stage , I switched to Bi-LSTM architecture which yielded better results when compared to LSTM architecture.

2. Data Encoding

Before feeding the training data into the Machine Learning models, it was necessary to encode the data . During this project, I experimented with encoding the data using pre-trained and random embeddings. The purpose was to compare the impact of pre-trained embeddings on the performance of the models and determine the best approach for this particular task. :

- **Without pre-trained Embeddings:** To encode the data without utilizing pre-trained embeddings, a lookup index was created for each word in the corpus that appeared more than three times. This approach was chosen as there were many misspelled words in the corpus that only occurred once or twice, and it was deemed necessary to eliminate these to ensure accurate results. To handle the variant length of sentences and prevent padding issues, the data was padded and two additional tokens were introduced: 'pad' and 'UNK', which represented unknown or out-of-vocabulary (OOV) words and padded tokens, respectively.
- **With pre-trained Embeddings:** The pre-trained GLOVE embeddings file was downloaded and used to create a lookup index for the data encoding process. As with the previous approach, the 'pad' and 'UNK' tokens were added to handle unknown data or padded tokens. The use of pre-trained embeddings aimed to provide a more robust representation of the text data and improved the performance of the models.

3. Training

During training phase, two types of embeddings were used as discussed earlier for training the models. Additionally, two different types of architectures were employed: a Bi-LSTM (Bidirectional Long Short-Term Memory) architecture and a standard LSTM architecture.

The goal of the experiment was to compare the impact of pre-trained embeddings on the performance of the models and determine the best approach for this particular NER task.The use of two different architectures allowed for an evaluation of the impact of architecture on the model's performance and provided insights into which architecture is best suited for this task.We prefer to use BiLSTM as they provide the context information in both forward and backward directions, resulting in a better representation of the text.

In first case,the models consisted of an embedding layer with a dimension of 300, followed by two layers of Bi-LSTM, and finally a fully connected layer.A similar approach was used when using LSTM cells.

Dropout training was also employed to reduce overfitting during training, with dropouts applied at the embedding level, between the Bi-LSTM layers, and before the final layer. This resulted in better validation loss.The optimizer used during training was the Adam optimizer and an SGD optimizer with a learning rate of 0.01 and a momentum of 0.9. The Adam optimizer allowed the model to converge faster compared to SGD, which was slower to converge.

4. Results

During this project, I used scikit-learn and Seqeval to calculate the f1-score and precision respectively. When evaluating the results, I found that using pre-trained word embeddings along with a Bi-LSTM architecture resulted in a higher precision score of **0.82**. A comparison between LSTM with embeddings, Bi-LSTM without embeddings, and Bi-LSTM with embeddings can be seen in the following figures. First I have plotted a normalized confusion matrix for all of these 3 cases which gives us insights into classes of labels that are perfectly classified by the model and the labels which are mostly misclassified. Next we compare the F1 score for each individual class for all these cases.

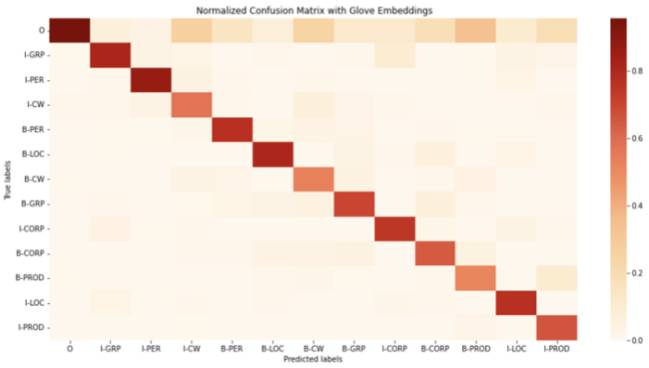

Figura 1. Confusion Matrix for Bi-lstm trained with Glove Embeddings

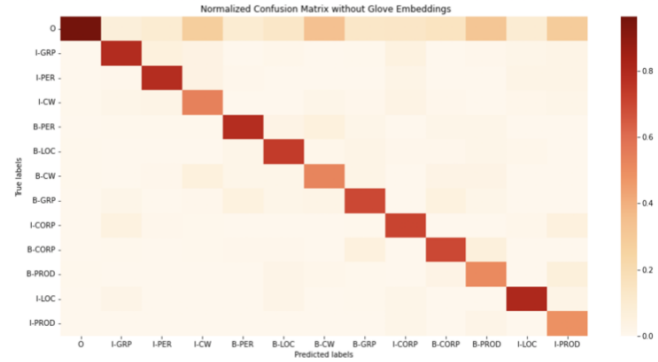

Figura 2. Confusion Matrix for Bi-lstm trained without Glove Embeddings

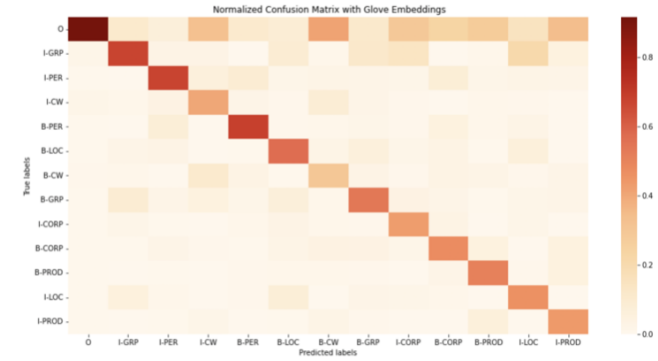

Figura 3. Confusion Matrix for LSTM trained with Glove Embeddings

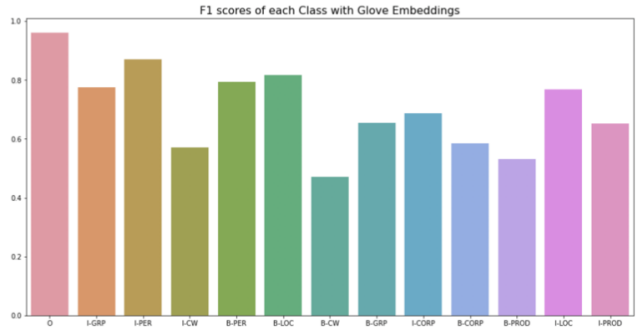

Figura 4. F1 Scores for each class for Bi-lstm trained with Glove Embeddings

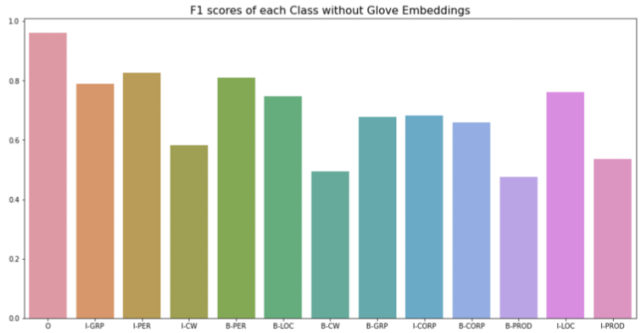

Figura 5. F1 Scores for each class for Bi-lstm trained without Glove Embeddings

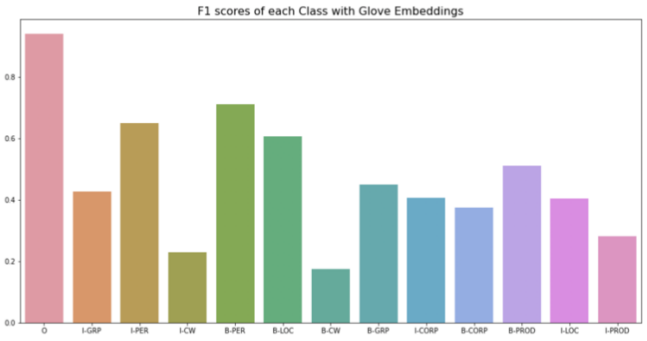

Figura 6. F1 Scores for each class for LSTM trained with Glove Embeddings

5. Conclusion

Clearly from the results, we can see that using the Bi-LSTM architecture performed better compared with using LSTM.

Also when using pre-trained embeddings from Glove,the result can be seen slightly better than that of using random embeddings.It was interesting to see that Bi-LSTM trained without Glove embeddings performed way better than LSTM trained with Glove Embeddings, although expected as Bi-LSTM could process the data in both forward and backward directions, capturing the dependencies in both directions.

The data was highly unbalanced with 192840 tokens labeled as ‘O’, while just 1710 ’I-PROD’ . As a result of this there is huge variance in f1 score of each class.The model performance can be made better by adding a lot more data having a balanced number of data points.