MenuNER

Federico Lorenzo Pes
Politecnico di Torino
s287822@studenti.polito.it

Eleonora Poeta
Politecnico di Torino
s292481@studenti.polito.it

***Abstract*—Named entity recognition (NER) is a sub-task of information extraction that seeks to locate and classify named entities in texts, into pre-defined categories such as the names of persons, organizations, locations, etc. In this paper, we focus on Menu-entities extraction from online user reviews for the restaurants. The approach involves first adapting NER task on a new domain where large datasets are rarely available, compute the different embeddings and fine-tuning the network model BiLSTM with CRF, very popular in NER tasks, using extended feature vectors. The implementations details are available at https://github.com/eleonorapoeta/MenuNER.git**

***Index Terms*—Named Entity Recognition, BiLSTM, CRF, BERT, Domain Adaptation, POS, CharCNN, Embedding.**

I. Problem Statement

The Named Entity Recognition (NER) task consists in identifying portion of text referring to named entities. The two-fold aims are to detect word n-grams that explicitly refer to particular entities and to classify them as one of the predefined category of named entities. The problem addressed by *Syed et al.* [1], consists in extracting the named entities representing dishes belonging to a menu, from users' text reviews of restaurants.1
The challenge of this problem is represented by the application of the NER task, in a domain with few labelled data available. In order to solve this problem the idea is to exploit state-of-the-art models which are already pre-trained, for a general task, and further train them toward a domain specific task. In this implementation the pre-trained BERT is further trained on restaurant reviews, in order to obtain more meaningful word embedding tailored towards the food domain. The approach will be developed in detail in the Methodology II section. In order to enrich the word representation, and provide more information about the context, POS Tag and Characters embeddings are concatenated to the BERT word embedding. Subsequently the BiLSTM network, along with CRF architecture are used to handle each word as part of a sentence as to provide more consciousness about the context to the whole model. In the end, this model provides the labels related to each word in the sentence.

II. Methodology

The methodology of this work follows the one used by *Syed et al.* in [1], with some modifications due to lack of computational resources and too high training time required.

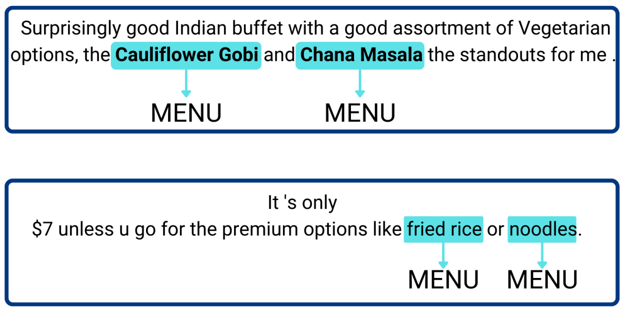
Surprisingly good Indian buffet with a good assortment of Vegetarian options, the Cauliflower Gobi and Chana Masala the standouts for me .
MENU
MENU
It 's only $7 unless u go for the premium options like fried rice or noodles.
MENU
MENU

Fig. 1. Two examples of how the NER task works into the Food Domain. In the end, the model should be able to identify and correctly classify the words that represent MENU entities.

A. Reviews retrieval and manipulation

First step consists in retrieving the reviews coming from the Yelp Dataset[1] and extract, from them, only the ones that are referred to Restaurant Business category. The Restaurants reviews serve as a new corpus of data that is domain-related to the foods.

B. Data Labelling

The authors of the paper perform a hand-labeling of the reviews dataset, dividing each review before into sentences and after into word-tokens exploiting the NLTK tokenizer. Next, each token is equipped with three labels:

- Part-of-Speech tag (POS) : it is chosen between the standard set of POS tags available in NLTK.
- Chunk Id label : not relevant to our objective.
- Menu Label : there are three possible values that take into account the possibility that a food name is composed of multiple tokens.
  - **B-MENU** : indicates that that is the first token of a food-word.
  - **I-MENU** : indicates that the token that token follows the previous as being part of the same food-word.
  - **O** : it stands for ”outside menu” and is assigned to all those tokens that do not represent food

Since to operate this part of manual data labeling would have been time consuming, we opted to use text files for both train, test and validation that had already been annotated.Table

[1]Available at: https://www.yelp.com/dataset/

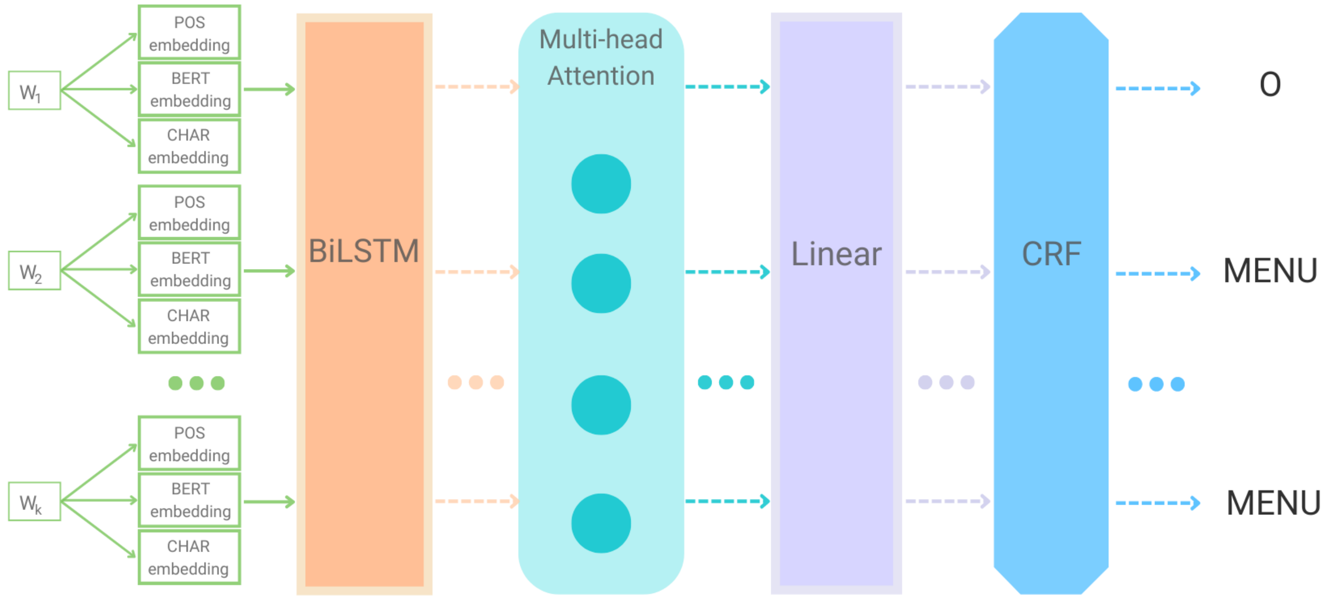

Fig. 2. MenuNER architecture used in this work. Each word of each sentence is embedded with POS, BERT and CHAR embedding layer (green). Then the BiLSTM (orange), along with the MultiHeadAttention (cyan), the Linear layer(lilac) and the CRF module (turquoise) work to predict the label of the word between O and MENU.

III shows a resume of all the dataset specifics.
During the data gathering we perform a merge of the labels *B-MENU* and *I-MENU*, to obtain only the label *MENU*. This is done to have to distinguish only menu-inside entities from the ones outside the menu.

C. BERT domain adaptation

Bidirectional Encoder Representations from Transformers (BERT) [3] is a transformer-based machine learning technique for natural language processing (NLP). BERT is an already pre-trained model on a large corpus, English written, and so it already has some general ”knowledge”. For the purposes of MenuNER, BERT is further pre-trained, to give it the domain-related knowledge that, in this case, is food-related. To do so, it is called an instance of the *BertForPretraining* model, over the reviews collected as mentioned before, choosing as pre-trained model the *’bert-base-cased’*. This choice is made taking into account that many of the food terms are indicated in the reviews with capital letters.

D. Data Preparation

The data used needs some preprocessing since they comes from a manually-labeled dataset.

1) Tokenization: First shrewdness is related to the typology of tokenization that is used. In fact, when reading the input files, each sentence of each review is given as tokenized with NLTK tokenizer. However, since the model exploits the BERT-domain previously trained, and its BERT-Tokenizer, to obtain the embeddings of each token, it becomes necessary doing an alignment of the two possible tokenization. This is done by taking a single token and performing another tokenization with BERT-Tokenizer, thus extending the original label and the POS tag, in case more tokens are obtained as a result of BERT-Tokenizer.

2) Feature extraction: After the label-alignment, it is performed the conversion from tokens and labels, to features that are useful for the model training. Depending on the features, different actions are performed:

- Token id : the id of each token is extracted after the BERT-tokenization exploiting the function *convert_tokens_to_ids()*.
- Label id : labels are mapped to integer values to obtain each id. Moreover, the labels *”B-MENU”* and *”I-MENU”* are merged into a generic label *”MENU”*
- POS id : this is computed scanning all the possible values of POS tags present in a dictionary and then it is assigned the corresponding integer.
- Char id : here it is applied the function *batch_to_ids()* coming from the AllenNLP library. This function converts each batch of tokenized sentences into a tensor representing the sentences with encoded characters.
- Attention : this is a mask that is used to indicate where the tokens are real or are pads.

3) Padding and creation of the dataset: Once the feature are extracted, other two manipulations are made on their representation. First one is the insertion of the corresponding encoding, *0*, for the *[CLS]* token, at the beginning of each sentence representation and, then, all the representation are padded up to a *max_length = 512*. Lastly, the MenuDataset is created as an instance of the *torch.Dataset* class and is then being passed to the *torch.DataLoader*.

E. Model

The model is composed of different parts, each of them committed to different purposes. Fig 2 shows each block comping the architecture and how they interact with themselves.

1) Embedding Layer - BERT: At the beginning of the model takes place the Embedding of some of the features described above. Each word of each sentences passes through three different embedding steps, all characterized by the *nn.Embedding()* layer of PyTorch, and at the end, the result is the concatenation of the aforementioned. The embeddings performed are:

- POS Embedding: this embedding exploits POS Tag features to extract syntactic features, which are helpful to disambiguate the words from their grammatical role in the sentence. The embedding is generated through an Embedding layer which works as a look-up table, providing a vector of $dimension = 64$ for each id.
- BERT-domain Embedding: it is a model capable of creating contextual word representations that capture high semantic and syntactic meaning. Each sequence is padded up to 512 token length and also truncation is True if the sequence surpasses this length. The embedding of the word is created through the hidden states of the model. Specifically, for this task, the last four hidden states are extracted and then, by applying mean pooling, we obtain the word embedding. The dimension of the embedding is 768, which is obtained from the size of the hidden states.
- Char-level Embedding : this embedding includes passing the ids of each character of each token both to the Embedding layer and the CharCNN network. The main purpose of including also this step is to give the character-level information to the model, permitting a more accurate analysis. The dimension of the Char Embedding is 64.

2) CharCNN: The Character-level CNN (CharCNN) is introduced by *Zhang et al.* in [2] for text classification purpose. The implementation done in this work involves two Convolutional layers in 1D having $input\_channels = 25$ that corresponds to the length of the char embedding. Then, are used $output\_channels = 32$ and the kernel dimensions of the two convolutions are $kernel\_sizes = [3, 9]$ After the convolutions, the relu activation function and a max-over-time pooling operation are applied, and at the end everything is concatenated.

3) BiLSTM: Bidirectional Long short-term memory (BiLSTM) Layers are networks capable of learning long-range dependencies, so their usage in tasks like NER is optimal since they permit full use of the context information of the input sequence. In this implementation, there are actually two BiLSTM stacked by concatenating the forward $h^{\rightarrow}$ and the backward $h^{\leftarrow}$ hidden states. The forward LSTM layer extracts the past information while the backward layer captures the future information of the sequence and produces the final output y_t. In this BiLSTM network, since there are two instances of this, the output y_t of the lower BiLSTM layer feeds into the upper level BiLSTM layer. The BiLSTM has $hidden\_units = 512$, so that the output representation dim is 1024 for each token in each stacked layer of the network. Also a $dropout = 0.1$ is applied.

4) MultiHeadAttention: The Multi-head self-attention (MHA) layer is applied on the top of the stacked BiLSTMs to provide the model the power to encode multiple contextualized relationships for each word. In this implementation the number of head is set to $num\_of\_head = 4$.

TABLE I
PARAMETER DETAILS

Parameter	Value
Max Length Representation	512
POS embedding dimension	64
BERT embedding dimension	768
Char embedding dimension	64
In channels CharCNN	25
Out channels CharCNN	32
Number of hidden units in the BiLSTM	512

5) CRF module: The Conditional Random Field (CRF) is a conditional probability model that is used for segmenting and labeling sequential data. The CRF layer is added right after a nn.Linear() layer and takes the logits from the Bi-LSTM as inputs. In this matter, it is important to clarify that in this implementation the CRF layer is taken from *pytorch-crf*. This layer is used in two ways. The first one consists in decoding the logits coming from the Linear layer and creating the output of the entire model. The other is computing the log_likelihood that is needed to calculate the Loss function of the model.

Therefore we have :

$$output\_model = CRF.decode(output\_Linear) \qquad (1)$$

$$log\_likelihood = CRF(output\_Linear) \qquad (2)$$

$$Loss = -1 \cdot log\_likelihood \qquad (3)$$

III. EXPERIMENTS

A. Data Description

a) BERT training data: Data are handled in different ways for the two steps. In order to train BERT we extract from the YELP Dataset 250000 reviews, which are all from the category *Restaurants*.
This reviews are then divided into sentences and prepared for the Masked Language Model (MLM) and Next Sentence Prediction (NSP) tasks. For the first task 15% of the tokens

are selected and masked with MASK special token. Then, for the second task, first 50% of the sentences are paired with a sentence that follows them, and the last 50% are paired with a randomly chosen sentence. The details regarding the training settings are provided in paragraph *BERT training settings*.

b) BiLSTM and CRF training data: To train the whole model for the NER task, the authors of the MenuNER paper provide a hand-labelled dataset where for each line there is a tuple containing the word, the POS Tag provided by NLTK, the chunk id (which is not relevant for our analysis) and finally the label. Each sentence is separated by a blank line. The dataset is already divided into Training , Validation and Test. The relative dataset specifics are described in the table I.

B. Experimental environment

All the training and inference steps are designed in order to run on the Google Colab framework that ensures the reproducibility of experiments, despite having computation limits.

a) BERT training settings: The training of BERT on the food corpus is done for 3 epochs which correspond to $90,000$ steps, taking approximately 10 hours. Every $10,000$ steps we create a checkpoint in order to resume the training from the checkpoint if needed. The optimizer is AdamW and has the epsilon parameter $\epsilon = \mathbf{10^{-8}}$ and the learning rate of $3 \cdot 10^{-5}$. Furthermore it is also exploited the *fp16* optimization on the BERT Trainer in order to reduce the total training time.

b) MenuNER training settings: The training for MenuNER task is also deployed on Google Colab, using AdamW optimizer with the learning rate $lr = \cdot 10^{-3}$, $\epsilon = 10^{-8}$ and $weight\_decay = 0.01$. Moreover, the training is done with a $batch\_size = 8$ because of the computational limitations of Google Colab, number of epochs equal to 30 and the patience value equals to 7 for the Early Stopping. During this part of the training, the BERT model is not trained and it is only used to provide the embedding for the model.

c) Metrics: The metrics used in this work are calculated exploiting the *seqeval.metrics* library by Hugging Face. This choice was made since those metrics are well-suited for the kind of output of this model. The following three metrics are calculated: Precision(P) 4, Recall(R)5 and F1 score(F1)6. Furthermore F1 score is the criterion to choose which model to save during training. All the metrics and their components are explained below in details.

- True Positve(TP): The number of Menu entities which are correctly classified as Menu.
- False Positive(FP): The number of Outside(O) non entities classified as Menu entity.
- True Negative(TN): The number of Outside(O) non entities classified as O.
- False Negative(FN): The number of Menu entities which are classified as O.

Then, to evaluate the model, the following three metrics are calculated: Precision(P) 4, Recall(R)5 and F1 score(F1)6.

$$P = \frac{TP}{TP + FP} \tag{4}$$

$$R = \frac{TP}{TP + FN} \tag{5}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6}$$

TABLE II
DATASET SPECIFICS

Dataset Description	**Dataset splits**		
	Training	***Test***	***Validation***
#Sentences	5517	3037	2211
#Total Entities	1026	637	516
#Unique Entities	2626	1424	1106

IV. RESULTS

A. Ablation study

For research purposes different model configurations were tested. The main focus of these experiments is the Embedder block and its different components. The entire model configuration is split in different components that are trained, and evaluated, in different combination as indicated below.

- BERT-domain and Char Embedding
- BERT-domain and POS Embedding
- BERT-domain Embedding

B. Experimental results

It can be affirmed that, generally, the combination of all blocks performs better than its subsets. This is probably because, thanks to the POS embedding, we have more syntactical information about the word in the sentence that actually enriches the context information for the single word. Furthermore, thanks to the Char embedding we can overcome the problem of misspellings and out of vocabulary words, providing also an important contribute to the task.

TABLE III
ABLATION STUDY: SCORES CALCULATED ON THE TEST

Embedder Modules	**Measures**		
	Precision	***Recall***	***F1 score***
BERT-domain	88.12%	88.60%	88.36%
BERT-domain and POS	87.21%	85.21%	86.17%
BERT-domain and Char	89.17%	88.72%	88.94%
BERT base, Pos and Char	90.17%	88.87%	89.52%
Complete Embedding	90.23%	89.50%	**89.86**%

The second part of the ablation study regards the investigation of the effectiveness of the domain adaptation of BERT model in the food domain. BERT is a model which already provides good performances and meaningful word embeddings for downstream tasks. However, doing the further training of BERT on the Restaurant reviews, provides the model an enrichment for the idea of the context in which is applied, leading to better performances.

V. Conclusion

The aim of this work was to carry on a Named Entity recognition task on the Food Domain and, in the end, shows a valuable technique to handle the critical problem of limited dataset in certain domains. This problem is addressed by domain adaptation of models, such as BERT, that are pre-trained on large general data sets. These, in and of themselves, lack specific knowledge and therefore will certainly perform less well. The further pre-training of BERT, on a specific-domain dataset, provides the model the contextual word embeddings that are probably more aware of the domain itself. This work also shows how enriching information with syntactical features can be a meaningful way of providing the model more awareness of the sentence and the role that each word plays in it. For future works and improvements would be interesting to use the BERT model both for obtaining the embeddings and to fine-tune it along with the rest of the model.

For example it could work on two stages where the first one is used to train the rest of the network, and after BERT could be finetuned along with the whole model.

References

[1] M. H. Syed and S.-T. Chung, "MenuNER: Domain-Adapted BERT Based NER Approach for a Domain with Limited Dataset and Its Application to Food Menu Domain," Applied Sciences, vol. 11, no. 13, p. 6007, Jun. 2021, doi: 10.3390/app11136007.

[2] Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. Advances in neural information processing systems, 28.

[3] Devlin, J., Chang, M. W., Lee, K., Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.