CZI - Entity Recognition and Entity Linking

Industrial Mentor: Sunil Mohan, CZI PhD Mentor: Patrick Verga
Shikha Agarwal, Sneha Bhattacharya, Nathan Greenberg, Srijan Mishra, Yuvraj Singla

1 Abstract

We explore the use of deep learning techniques to automate entity recognition and linking in biomedical journals. We explore a Bi-LSTM and CRF model for entity recognition, and a separate LSTM based, modular neural model for entity linking. We compare the results of these models against the results of TaggerOne, a semi-Markov model that learns both of these tasks jointly.

2 Introduction

Named entity recognition (NER) is the problem of recognizing groups of words as entities. NER for biomedical papers is very important for tasks such as such as the correct identification of mention and span of diseases and genes in a particular text. Given the huge volume of biomedical research papers, we need accurate as well as efficient models for such tasks. Long-Short-Term Memory (LSTM) based models in combination with Conditional Random Fields (CRF) achieve high performance on a variety of NER tasks when provided with an appropriate training corpus and a relatively small investment in feature engineering.

Many end-user tasks also require entity linking, the identification of the concept mentioned within a controlled vocabulary or ontology. For this task, we have partially explored a tf-idf approach in conjunction with a neural linking model.

We have divided the task into two phases, the first is to perform NER followed by entity linking. In the second phase we use the joint learning model, TaggerOne for join NER and linking.

3 Related Work

3.1 Joint learning -NER and linking

In Taggerone[3], the authors propose a joint learning method to solve the problem of both entity recognition and linking. They introduce a semi-Markov based approach for NER and a cosine similarity function between tf-idf features for linking. The joint learning task is seen as finding the best NER segmentation which maximizes the sum of individual NER score and linking score.

They define a scoring function over the set of valid segmentations $\mathcal{Y}(X)$, so that the task of prediction becomes finding the segmentation $Y \in \mathcal{Y}(X)$ with the highest score:

$$f(X) = \underset{Y \in \mathcal{Y}}{\operatorname{argmax}}(Score(Y; s, t, W))$$

$$Score(Y; s, t, W) = \sum_{j=0}^{|Y|} Score(Y_j; s, t, W)$$

$$Score(Y_j; s, t, W) = Score_{NER}(Y_j; s) + Score_{Norm}(Y_j; t, W)$$

They model the NER scoring function as a structured classification problem using a multi-class linear classifier, which learns one weight vector s_l per label $l_i \in \mathcal{L}$. t_l is an additional term for the cosine similarity. Element (i, j) in matrix W_l can be interpreted as the correlation between token t_i appearing in a text segment with NER label l and token t_j appearing in any concept name for l from the lexicon. The model can thus learn a variety of relationships between tokens in text and names from the lexicon, including both synonymy and contrast.

3.2 Bi-directional LSTM with CRF

In Huang et al [1] the authors have shown the use of Bi-directional LSTM with CRF for a benchmark NLP tasks like sequence tagging and Named Entity Recognition. Becuase they use a bi-directional LTM, the model can effectively take into account the left and right context of each word. The tag transition information gets utilized because of the CRF layer. It has produced state of the art result in NER and part of speech tagging tasks.

3.3 Entity Linking

In Gupta et al [2]. the authors explore a neural, modular entity linking system that learns a unified dense representation for each entity using multiple sources of information. The model uses a CNN over a Wikipedia description of the entity, an LSTM over the context of a mention, and entity type information to build mention and entity embeddings – thus capturing different aspects of the "meaning" of an entity. Hence, they overcome the shortcomings of several existing models that do not capture all these aspects.

4 Experiments

4.1 Entity Recognition

Entity recognition is the problem of grouping the words of a sentence into entities. We are formulating the individual task of entity recognition as a sequence tagging problem. The input is a sequence of tokens, and the output is a sequence of entity types corresponding to each token. We are using IOB encoding for each of the entity types. B means the beginning of an entity, I means inside of an entity, and O means outside of an entity (i.e. not part of an entity). For example, "Due to pulmonary disease, ..." would be labeled as "O, O, B-Disease, I-Disease, ...".

For this task, we are using a bidirectional Long Short-Term Memory Network (LSTM) with an added conditional random field layer, similar to Huang et al[1]. The LSTM takes a sequence of word embeddings as its input.

We use a bidirectional LSTM which is able to use the left and right context for each word to produce its output. The output of the LSTM is passed through a hidden layer with a nonlinearity.

The output from the hidden layer is passed through a linear transformation to produce scores for each tag. The CRF models the transitions between tags, and uses the scores to produce the best sequence of tags.

We train all the parameters of the model jointly, using the log likelihood of the training dataset. We are using the Adam optimizer to optimize the log likelihood.

Once we have entity types for each token, we can decode the IOB encoding into entities.

4.1.1 Sentence Tokenization

We tokenized the dataset for NER in two ways. One, where we used the entire abstract and title and then use a word tokenizer (the GENIA tagger) over it. The other was tokenize the abstract into sentences and then tokenize those sentence into words. In this way, we can feed either a sentence into our model, or an entire abstract. We found that using the entire abstract worked better than using single sentences.

4.1.2 Character-level Embeddings

We also experimented with using character embeddings. We use an additional bidirectional LSTM model that takes a sequence of characters from a word as input, and outputs a single vector that represents the character-level representation of that word. Using this additional LSTM, we augment each word embedding with character-level information. This is helpful for taking advantage of the similar spelling of some words. We found that using character-level embeddings of words improved the performance of the model.

4.2 Entity Linking

For the entity linking task we are using the method as mentioned in Gupta et. al. The difference is that we are not using outside knowledge to create a description embedding. We have created encodings of a particular mention, locally, using an LSTM from the beginning of the sentence to the mention and from the mention to the end of the sentence, to capture the right and the left context of the mention. We also use a bag of mentions to capture the context of the mention at the abstract level. These are combined to get the full mention context. To generate the predictions, we plan to use a set of generated candidate entities and their prior scores. We predict the likelihood of a mention being each entity candidate by taking the dot product of the mention and entity vector representations and normalizing over all entity candidates.

4.2.1 Candidate reduction

We have trained a tf-idf vectorizer on all of the gold mentions present in all the types using character and word level features. Assuming that a perfect recognition step would give us the document type of the recognized mention, we tried to find the top 1, 5 and 100 closest matched pairwise cosine distance in the tf-idf transforms of the mention and that of the mentions in a particular document type. This helps us in reducing the number of candidates.

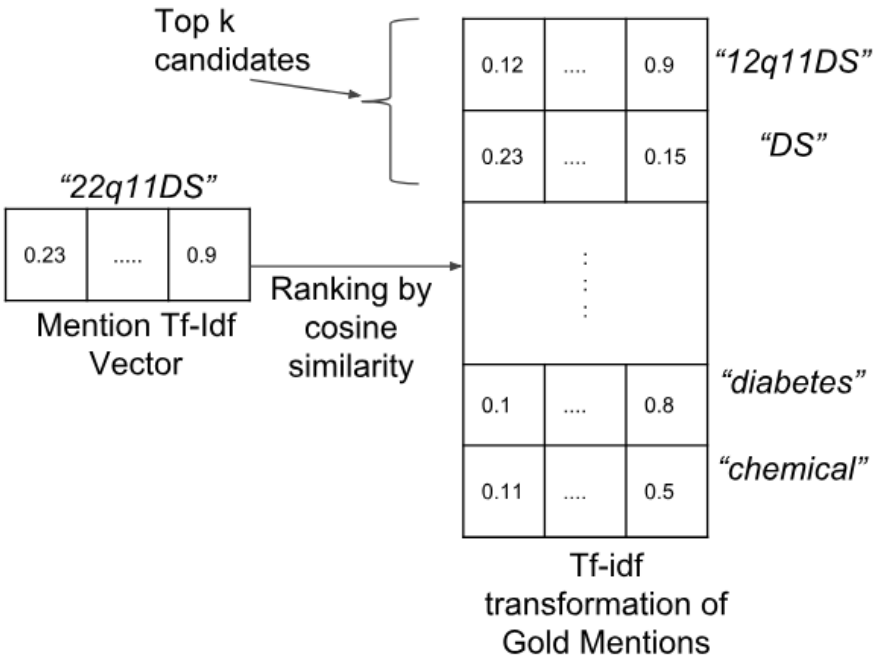

Figure 1: Candidate Set Reduction

4.3 TaggerOne

One of the challenges of having the two tasks of entity recognition and linking in a serial pipeline is that it cascades the errors from NER to linking. Also, the NER module is unable to directly exploit the lexical information captured by Linking. For combining the two tasks, we are using a joint learning model called TaggerOne[3].

Using Ab3P to identify the replace abbreviations took a significant amount of time to run. We thus use a static abbreviations file preprocessed for our dataset. The text is then broken into sentences. For NER, it uses a semi-Markov structured linear classifier, with a rich feature approach. Where a traditional Markov model tags each token to one of the lexicons, the model trains a semi markov structure to identify segments which may have multiple tokens. At the token level, the features used are token text, token stem, part of speech, character level n—grams, etc. It also uses feature templates at the segment level which includes the number of tokens in the segment, the characters and tokens surrounding the segment, the first and last token, whether the segment contains unbalanced parenthesis, etc. The NER feature vector for each segment is equal to the segment level feature values summed with each of the token level features for each token within the segment.

For linking, a supervised semantic indexing approach is used which converts both the segments and the names into vectors and then uses a weight matrix to score pairs of vectors. The normalized vector for each entity is a modification of the tf-idf vector considering the entities as documents and the entity's lexicon as the collection of documents. A cosine similarity is used to match the entity segments with the normalized entity representations. NER and linking is performed simultaneously by defining the score for each segment to be the sum of its NER and normalization scores. The combined objective function in effect learns the best segmentation which produces the best linking

and thus minimizing the overall loss.
For our TaggerOne experiments, we were able to get results for NER, as well as linking results given both the gold NER entities and TaggerOne's own predicted NER entities. The results are discussed below.

4.4 Data

We have titles and abstracts from 4300 biomedical papers with manually labeled entities with 19 UMLS entity types. The dataset is divided into 60% training set, 20% validation set, and 20% test set.

4.5 Word Embedding

We have used pre-trained word embeddings that have been trained on a text corpus of biomedical scientific literature that contains over five billion words. It was run using the skip-gram model with a window size of 5, hierarchical softmax training, and a frequent word sub-sampling threshold of 0.001 to create 200-dimensional vectors.
As the biomedical papers contain chemical names like "DCTN(4)" or "pirimiphos-methyl", we need a tokenizer that does not separate these words at '(' or '-'. Thus, we use Genia tokenizer that is specifically tuned for biomedical texts.

5 Results

Performance of the NER system is evaluated on the 20 percent of dataset available as test set. Since we have two sub-problems to solve, we can evaluate them separately. Specifically, we focus on the following questions:
1. How effective is the Bi-LSTM model in NER predictions?
2. How well does the entity linking model do with gold NER labels?
3. How well does the entity linking model do with predicted NER labels?

We measure NER model at the mention level, where we require the predicted span and entity type to exactly match the annotated span and entity type. For linking, we plan to measure at both the mention level and the abstract level. In the abstract level, we will compare the set of concepts predicted for the document to the set annotated, independent of their location within the text. We are using precision, recall and F1 score as evaluation metrics.

5.1 Results from TaggerOne

We trained and evaluated the TaggerOne model on our dataset. We report results for NER only with exact match in Table 1. We also have results from TaggerOne for entity linking. Table 2 has linking results using only gold entities and Table 3 has linking results using predicted entities.

Table 1: NER TaggerOne: micro average scores

F1	**Precision**	**Recall**
56.36	59.42	53.66

Table 2: Linking TaggerOne (Gold): micro average scores

F1	Precision	Recall
74.9	77.4	72.5

Table 3: Linking TaggerOne (Prediced): micro average scores

F1	Precision	Recall
44.16	46.5	42

5.2 Results for NER: Bi-LSTM

Our best LSTM + CRF model for NER takes in an entire abstract, instead of just a sentence. It also uses the character embedding model. Table 4 gives our best model's results for NER. We also report a confusion matrix across all the labels in Figure 1. The x-axis is the predicted labels and the y-axis is true labels. The results suggest our model does well in identifying correct labels. It also shows that the model is most likely to misclassify entities with the "Outside" label, as opposed cross-entity type missclassification.

Table 4: NER Bi-LSTM: micro average scores

	F1	Precision	Recall
sentence level, no character embeddings	58.36	58.39	58.33
abstract level, no character embeddings	58.32	60.33	56.43
sentence level, character embeddings	59.34	59.08	59.60
abstract level, character embeddings	**59.35**	60.49	58.25

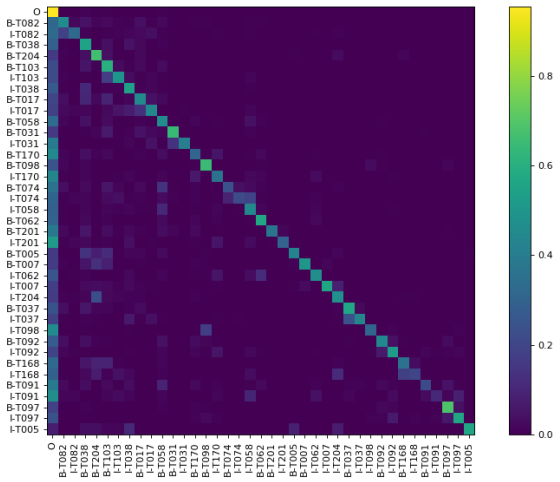

Figure 2: Confusion Matrix for NER

5.3 Comparison with TaggerOne

We have evaluated TaggerOne and the bi-LSTM model usng the same metrics. On the UMLS dataset using 19 entity types, the bi-LSTM model performs better than the TaggerOne by 3 points for the task of NER. Additionally, the bi-LSTM model takes about a day to train, while TaggerOne takes about a week to train. The bi-LSTM model is more efficient in regards to time, as well as more accurate.

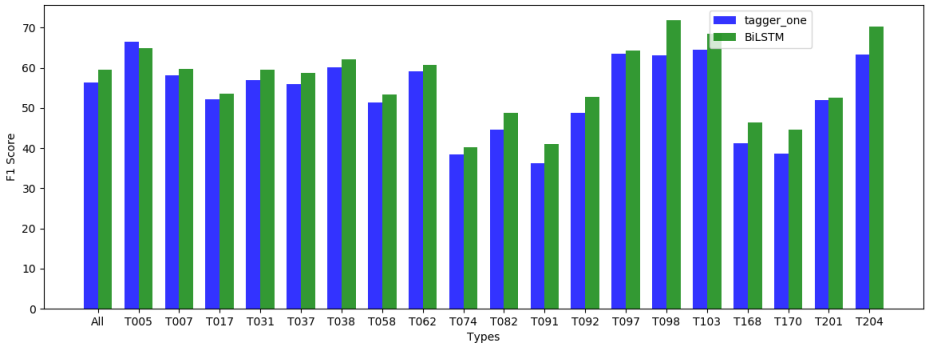

Figure 3: Type specific F1 scores

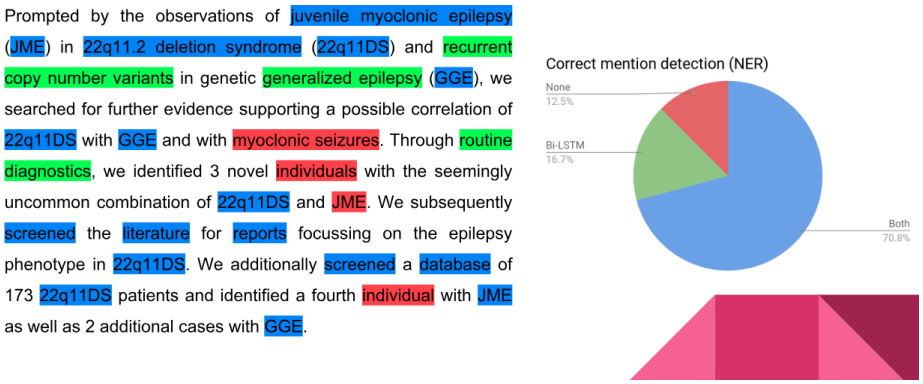

Figure 4: Correct mention detection(NER)

5.4 Linking experiment

We tried training the neural linking model over the training set using small dimensions (100 dimensions for word embeddings and LSTM state size) and all the possible 17,000 entities as candidates for each mention. This achieved around 30% accuracy on the development set. The model used a lot of memory, and we were not achieving good results, so we decided to focus on candidate set reduction instead of getting precise results for linking. We believe that with proper candidate set reduction, we will be able to achieve numbers comparable to TaggerOne. This is because we will be able to train a model with larger dimensions more efficiently.

6 Conclusion and Future Work

Using a neural NER model, we were able to achieve better NER results on our biomedical dataset than TaggerOne. We were also able to train the neural model much more efficiently than TaggerOne. We believe that this shows promise for this model as part of a named entity recognition and linking pipeline. However, we were not able to properly test our neural entity linking model, due to time constraints.

In the future, we would like to train the neural linking model using a reduced set of entities from our candidate set reduction step, as opposed to using all the entities in the dataset. As previously discussed, we believe that this will allow the model to achieve competitive results. Additionally, the original neural linking model uses Wikipedia descriptions of entities to augment the training process. We would also like to implement this. Since entity linking can be such a difficult task, we believe using outside knowledge of enties could greatly reduce the ambiguity of the linking task.

With a proper entity linking model, we will be able to test the full named entity recognition and linking pipeline. The results from this pipeline would be very informative when comparing against TaggerOne, and we would fully be able to explore the magnitude of the cascading errors problem.

References

[1] Zhiheng Huang, Wei Xu, and Kai Yu. *Bidirectional lstm-crf models for sequence tagging.* arXiv preprint arXiv:1508.01991 (2015)

[2] Gupta, Nitish, Sameer Singh, and Dan Roth. *Entity linking via joint encoding of types, descriptions, and context.* Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. 2017.

[3] Leaman, Robert, and Zhiyong Lu. *TaggerOne: joint named entity recognition and normalization with semi-Markov Models.* Bioinformatics 32.18 (2016): 2839-2846.