Assignment 3: NER System

Soham Pal (C)
Department of CSA,
Indian Institute of Science
sohampal@iisc.ac.in

1 Task

The goal of this assignment is to build an NER system. Every example is in the form of a 2-tuple (x, y), where $x = (x_1, x_2, \dots x_n)$ and $y = (y_1, y_2, \dots y_n)$ are equal length sequences. Here, the dataset $\mathcal{D} = \{(x, y)\}$ consists of titles of medical articles x, where each x_i is a dictionary word. The task is to recognize two named entities – diseases (D) and treatments (T) by labeling them appropriately, i.e. setting $y_j, \dots y_k = D$ or T respectively where $x_j, \dots x_k$ corresponds to a disease or treatment respectively. For all other words x_i, we omit O, indicating the label other.

The primarily challenges include:

1. The dataset size is fairly small, around 3000
2. The inherent skew in the dataset in the favor of the O token, as T and D are rarer than other words.

2 Data Preparation

We divide the data into splits of training, validation and test data in the ratio 7:1:2. We preserve all the word in the vocabulary, and do not make use of an out of vocabulary token while training the model.

3 Evaluation

Let the confusion matrix be denoted by M_{ij}, where i indicates the true label and j indicates the predicted label (so M_{ij} is the number of times a word with true label i was predicted to have the label j). For evaluation, we use the following two measures:

3.1 Accuracy

This is a simple word-level accuracy computed over each token.

$$\text{Accuracy} = \frac{\sum_i M_{ii}}{\sum_i \sum_j M_{ij}}$$

3.2 Precision, Recall and F_1-measure

As the dataset is highly skewed (there are 55,810 O tags, 4,889 T tags and 3,821 D tags in the dataset), a high accuracy may not be particularly indicative of good results. For instance, a classifier that naively labels every word O will obtain an accuracy of 86.50%!

To remedy this, we also calculate the F_1 measure for each symbol i as follows:

$$\text{Precision}_i = \frac{M_{ii}}{\sum_j M_{ij}}$$

$$\text{Recall}_i = \frac{M_{ii}}{\sum_j M_{ji}}$$

$$F_1 \text{ Measure}_i = \frac{2 \times \text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$$

4 Baseline

As the baseline, we use MALLET (Machine Learning for LanguagE Toolkit). In particular, we make use of SimpleTagger (`cc.mallet.fst.SimpleTagger`), which internally makes use of Conditional Random Fields.

4.1 Feature Engineering

As CRFs make use of discrete features, we must perform manual feature extraction on the dataset. The following features were chosen based on manually inspecting the dataset:

1. **Word features:** We introduce a feature for every word in the vocabulary. For each word, its corresponding word feature is associated with it.

2. **Parts-of-speech features:** We introduce a feature corresponding to each one of the 44 POS tags as defined in the Penn Treebank

(PTB). We run the `nltk` part-of-speech tagger on each input sentence and annotate each word with the feature corresponding to its POS tag.

3. **Wordnet features:** We introduced the wordnet features `body_part`, `medicine`, `treatment`, `disease` and `chemical`. We check to see if any *contiguous substring* of words in the input is a hyponym of any of the synsets corresponding to these features, and annotate it with the corresponding feature.

4. **MeSH features:** We extract all of the MeSH descriptors and qualifiers, as well as their corresponding entry terms from the MeSH dataset. If any *contiguous substring* in the dataset matches with either a qualifier or a descriptor or any of their entry terms, we annotate it with two features:
   - Whether it is a `qualifier`, or `descriptor` (both features may be added if it is both).
   - A feature that uniquely corresponds to that qualifier or descriptor.

5 Neural Model (BiGRU)

Against the baseline, we pit a BiGRU-based neural network model.

5.1 Architecture

We make use of stacked BiGRUs that have 150 neurons in eac direction, with a dropout of 0.5 applied to both the recurrent and non-recurrent layers. We make use of 3 layers of Stacked GRUs. This was found to produce the best results across a number of configurations we initially tested.

5.2 Embeddings

We experiment with three kinds of embeddings:

1. **From scratch:** We learn embeddings from scratch starting with a zero matrix.

2. **From GloVe:** We learn embeddings, starting with the pretrained 300-dimensional GloVe embeddings.

3. **From InDomainW2V:** We learn embeddings, starting with the 200-dimensional Word2Vec embeddings. These embeddings are learned from the unlabeled in-domain data provided as a part of the problem statement of this assignment.

4. **From GloVe+InDomainW2V:** We concatenate the two embeddings described above to arrive at a 500-dimensional embedding. These are used to initialize the embedding matrix and trained for improvement.

As an aside, we did experiment with disabling the training of the embedding matrix, but the results obtained were always significantly worse. Thus, we always train the embedding matrix end-to-end along in line with the RNN.

5.3 InDomainW2V

In this subsection, we explain how we preprocess the unlabeled data for training the word2vec model. Each document in the unlabeled dataset is in the form of an XML file. For each such XML document, we retrieve all the `<p>` (paragraph) tags in the document. We extract all the text from each `<p>`, including any text that may lie in nested tags. This is then inlined to produce a single paragraph to train word2vec on.

The resulting file on which training is performed contains 647,309,369 words across 12,028,847 lines (i.e. abstracts).

Google's Word2Vec implementation is used to train a Continuous Bag of Words (CBoW) model of size 200 over a window size of 8. Negative sampling is used with k set to a value of 25. For training, we perform 15 iterations over the entire unlabeled dataset.

6 Results

All results are reported for the test set, corresponding to the model that performs the best on the validation dataset.

6.1 Baseline (ablation study)

First, we look at results with only *word features* in Table 1.

	O	T	D
Precision	99.18	10.84	21.46
Recall	87.89	76.67	75.96
F_1 Measure	93.19	18.99	33.46
Accuracy	87.53		

Table 1: Baseline with word features.

Next, we add the *POS features* in Table 2.

	O	T	D
Precision	98.06	21.32	31.30
Recall	89.13	77.35	64.37
F_1 Measure	**93.38**	**33.43**	**42.12**
Accuracy	**88.00**		

Table 2: Baseline with features 1-2.

	O	T	D
Precision	97.93	21.32	41.83
Recall	89.91	70.70	71.19
F_1 Measure	**93.75**	32.76	**52.70**
Accuracy	**88.69**		

Table 3: Baseline with features 1-3.

Next, we add the *Wordnet features* in Table 3. Finally, we add the *MeSH features* in Table 4.

	O	T	D
Precision	98.02	22.38	42.03
Recall	90.00	71.16	72.74
F_1 Measure	**93.84**	**34.05**	**53.28**
Accuracy	**88.85**		

Table 4: Baseline with all features.

From the progression shown from Table 1 to Table 4, it is easy to see that manual feature engineering has helped double the F_1 measure of T from 18.99 to 34.05, as also boost the F_1 measure of D from 33.46 to 53.28. However, instead of meticulously engineering these features manually, we choose to exploit the automatic end-to-end feature extraction capability of neural models. The results are shown in the following subsection.

6.2 Neural Model (ablation study)

First, we report results using the *from scratch* embeddings in Table 5.

	O	T	D
Precision	97.52	48.88	64.47
Recall	93.41	80.74	76.97
F_1 Measure	**95.42**	**60.90**	**70.17**
Accuracy	**91.86**		

Table 5: Neural model (from scratch).

Even without any semantic information, we see that the neural model is able to outperform the CRF model quite easily. Thanks to the strong regularlization, the neural model is able to learn even from a good model from a limited number of training examples. Next, we report results using the *from GloVe* embeddings in Table 6.

	O	T	D
Precision	97.19	69.14	79.72
Recall	96.03	80.97	80.76
F_1 Measure	**96.61**	**74.59**	**80.24**
Accuracy	94.04		

Table 6: Neural model (from GloVe).

It is easy to see that the GloVe results show a clear improvement over learning the embeddings from scratch. This is to be expected, owing to the sparsity of data available to us (3000 training examples). The pretrained vectors contain semantic information that can be easily exploited. Next, we report results using the *from InDomainW2V* embeddings in Table 7.

	O	T	D
Precision	95.54	79.98	85.63
Recall	97.34	72.70	76.25
F_1 Measure	96.43	**76.16**	**80.67**
Accuracy	93.78		

Table 7: Neural model (from InDomainW2V).

Note that while the overall accuracy has dropped, the F_1 Measure on the two infrequent tags has actually improved! This demonstrates the gains obtained from training on in-domain data! Finally, we report the results using the *from GloVe+InDomainW2V* embeddings in Table 8.

	O	T	D
Precision	96.23	80.33	86.91
Recall	97.43	78.57	77.66
F_1 Measure	**96.83**	**79.44**	**82.03**
Accuracy	94.49		

Table 8: Neural model (combined).

Clearly, using both GloVe and InDomainW2V results in dramatic boost in performance, pushing the F_1 measure for both O and D to 80! We speculate that this is the result of the two embeddings complementing each other well, each one making up with information that the other might lack. Thus, this table (i.e. Table 8) tabulates our final results.