Biological Named Entity Recognition System

Kexin Hui and **Yuyuan Lou**
khui3@illinois.edu and ylou7@illinois.edu

Abstract

This document is the final report of the class project which discusses methods to apply named entity recognition (NER) algorithm to texts in biological area, typically classify protein from the data.

1 Introduction

In the realm of Natural Language Processing (NLP), efficiently and accurately finding a named-entity is a very important task for such as information extraction and retrieval, spelling correction etc. This can be applied to many text-based applications. There are mainly three sub-problems in NER: finding entity names such as persons, organizations and locations; finding temporal expression such as times and dates; finding numerical quantities such as money values and percentages. The latter two are comparably straightforward as the features for them are relatively fixed. On the other hand, features for entity names are quite ambiguous and flexible. Many sophisticated problems rely here. For example, semantic understanding of the text. Instead of just generally localizing the entity names, we want to focus on bio-related name entities, in this case, protein. It makes the task even more difficult since the biologically relevant words do not share any obvious features. For example, they can be formatted randomly by numerical numbers and alphabets. Even more, some common words can have an entirely different meaning and play a significant role in the biological world. For example, egg is a common type of food but it can also be viewed as a female reproductive cell.

In this case, we want to apply machine learning methodology to build an efficient, accurate and fast protein spotting system to finding biologically related names in relevant dataset. Basically, we use Support Vector Machine (SVM) linear classifier with our well-selected feature spaces. All the features are extracted based on our close scrutiny of the protein named entity from our data. Note that some names may contain multiple words instead of a single-word, which greatly increases the difficulty of accurate prediction. In this case, we consider merging words if they are predicted and they are close to each other. In terms of evaluating our method, accuracy is not sufficient here since true negatives are not available. We turn to use precision, recall and F-measure to evaluate our prediction.

This paper mainly describes the method we used to implement Biology NER and tis performance. All the work is done in Python.

2 Dataset

The dataset we are using is Genia Event Extraction for NFkB from BioNLP-Shared Tasks 2013. consider the dataset from BioNLP - Shared Task 2013. Specifically, GENIA is based on Genia corpus, a collection of documents for NF-kB. NF-kB is a protein complex that controls the transaction of DNA.

This dataset perfectly satisfies our needs that it is a professional document in the domain of molecular biology and medicine. In addition, it contains sufficient data for our training and testing: training data with 222 files, development data with 249 files, and test data with 305 files. For each set, we are given a txt file containing the original text, as well as an .a1

EBV **Latent Membrane Protein 1** Activates **Akt**, NFkappaB, and **Stat3** in B Cell Lymphomas **Latent membrane protein 1** (**LMP1**) is the major oncoprotein of Epstein-Barr virus (EBV). In transgenic mice, **LMP1** promotes increased lymphoma development by 12 mo of age. This study reveals that lymphoma develops in B-1a lymphocytes, a population that is associated with transformation in older mice. The lymphoma cells have deregulated cell cycle markers, and inhibitors of **Akt**, NFkappaB, and **Stat3** block the enhanced viability of **LMP1** transgenic lymphocytes and lymphoma cells in vitro. Lymphoma cells are independent of **IL4**/**Stat6** signaling for survival and proliferation, but have constitutively activated **Stat3** signaling. These same targets are also deregulated in wild-type B-1a lymphomas that arise spontaneously through age predisposition. These results suggest that **Akt**, NFkappaB, and **Stat3** pathways may serve as effective targets in the treatment of EBV-associated B cell lymphomas.

Figure 1: Sample Labeled Data

file with the category, location, and extracted named entity.

However, we are not able to finish the multi-class classification as proposed before. Because the test data in this dataset only contains the labels for proteins, no other biological categories. We are able to train and predict based on the training data and development data. Yet we are not able to evaluate our classification results.

3 Method

3.1 Classifier

Initially, inspired by course staff, we are interested in linear classifiers. Because under the framework of LBJava, NER uses Perceptron linear classifier and gives quite fast and accurate prediction. Yet we are not familiar with Java and it was really hard for us to understand that framework in such a short time. Therefore, we chose to implement the linear classifier ourselves. SVM is preferred since it maximizes margin and works quite well on high-dimensional feature space.

3.2 Features

Features are one of the most significant factors affecting the performance of NER. They would provide us with the distinguished characteristics of a word categorized as a named entity in order to help us determine its correlation to a named entity. Based on our common sense and the normal features of a named entity extraction system, we consider these four normal features as follows.(Baluja et al., 2000)

1. All capital letters
   They are usually short-cut forms of a named entity. For example, ICSBP, also known as interferon consensus sequence-binding protein.

2. Mixed upper-class and lower-class letters
   By the same token, these are also short-cut forms. For example, IgH, immunoglobulin heavy chain, one kind of an antibody.

3. Initial capital letters
   Often names would start with a capital letter. For example, However, the initial word of a sentence is always capitalized. This would be confusing sometimes. Therefore, this feature for a word is activated when it is initially capitalized and it is not the starting word of the sentence.

4. Part of speech tagger
   Basically, there are eight parts of speech in a sentence: nouns, pronouns, verbs, adjectives, adverbs, conjunctions, prepositions and interjunctions. It is obvious that a named entity is a noun.

Yet we are not dealing with the common world, but a biological world. A sample data is shown in Figure 1. There are many specific naming rules for biological words. For example, they might share many unique prefixes and suffixes: enzymes commonly have accepted names ending in aseref. After close observation of the data, we need to take into account several more features.

5. Alphanumeric
   They are possibly a combination of letters and Arabic numerals. For example, CD3, cluster of differentiation 3.

6. Greek letters
   Naming of a protein may contain a Greek letter. For example, TNF-alpha, tumor necrosis factor alpha.

7. '+', '-', '/'
   There may be some special symbols here to denote a protein name. We find three in our dataset as above. There could be more. For example, IRF-4, interferon regulatory factor 4.

8. Similarity
   We want to use similarity to find the words that share the same prefix and suffix. In addition, naming for protein of the same family/type will

also contain some same names, for example, CD4 and CD4+. Thus, we need to come up with some distance measurement between two words.

a. Hamming distance
Hamming distance is a most common distance measurement between two strings. It measures the number of positions at which the string are different. Yet it requires two equal-length string. This is not applicable since the character counts of a protein name can be any number.

b. Levenshtein distance
Levenshtein distance is superior than Hamming distance that it doesn't require the two strings to be the same length. However, based on Equation 1, Levenshtein distance works by recursively calling itself. This would cause a big problem on running time. The complexity of this algorithm is $O(3^n)$.

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) \quad if\ min(i,j) = 0 \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{a_i \neq b_j} \end{cases} \\ otherwise \end{cases} \tag{1}$$

c. Our own similarity function
Since Levensthein distance is not reliable to compute the similarity feature in a reasonable amount of time, we have to implement our own similarity function. To match the prefix and suffix, we want to align the two words by the front and by the end respectively, and compute the number of continuous same letters.

It is worth mentioning that some protein names in our dataset will contain multiple words, for example, IL-10 receptor. However, the features for these multiple-word named entity are not explicit since there are actually no general rules. Initially, we want to add a bigram feature to measure this. Yet this only works for two-word name. Moreover, this doesn't improve our result in the end. On the other hand, each word of that multi-word named entity can be classified accurately based on our current features. Therefore, we do not add additional features.

Algorithm 1 Pseudocode for similarity

```
train_words ← array of words to compare
Initialize max_sim ← -1
for word in train_words do
    sim ← 0
    for i from 0 to min(length) do
        if test[i] == word[i] then
            sim += 1
        else
            break
    for i from 1 to min(length) + 1 do
        if test[-i] == word[-i] then
            sim += 1
        else
            break
    max_sim = max(max_sim, sim)
score = max_sim / len(test)
return score
```

In total, we will have 8 major features. In the end, we construct a feature list for each based on the 8 features we summarized above. The list will be fed into the classifier.

3.3 Training

We first loop through all the *.a1 files to get a dictionary of all the proteins, with protein name as key and 1 as data. Then for each txt file, we split it into an array of words. After striping the stop words and punctuation marks, we construct a feature vector mentioned above for each word and use them to train the NER.

3.4 Dealing with Multi-word

As mentioned above, we cannot come up with much more effective features to classify a multi-word name. Since each word can be classified accurately, we plan to solve this when testing. That is, merge consecutive words to form a complete multi-word name entity.

In the merge process, if we encounter several consecutive words classified as protein, we merge them into one entity. If conjunction words (e.g. 'and') are

encountered, we will take the part of speech tag into account. If nouns are appearing on either side of the conjunction word, they should be classified as two separate entities. Otherwise, we will keep merging the words until a word is not classified as protein. If prepositions (e.g. 'of') are encountered, we will keep appending the words to the list until a word is not classified as protein. Pseudocode in Algorithm 2 below.

Algorithm 2 Pseudocode for Merge process

```
Initialize protein_list
for txt file in test set do
    Initialize append
    Split txt into array
    for word in array do
        if word = 'and' or word = 'of' then
            append ← word
        else if word classified as protein then
            if Noun appears after 'and' then
                append pop
                protein_list ← str(append)
                append ← word
        else
            protein_list ← str(append)
            append = []
return protein_list
```

3.5 Testing

For every txt file in the test set, we first split the text into an array of words. Then we pass every word in the array to the trained NER. By this method, we are able to determine whether a single word is a protein. We also handle the problem of multi-word as described above.

3.6 Evaluation

We would like to quantitatively evaluate the performance of the NER, so we would like to introduce the following measure of Evaluation.

1. Accuracy measures the percentage of test set tuples that are correctly classified by the NER.

$$Accuracy = \frac{TP + TN}{P + N}$$

where

$TP = \#$ of proteins correctly labeled
$TN = \#$ of non proteins correctly labeled
$P = \#$ of true proteins in the text
$P = \#$ of true non proteins in the text

However, for this dataset, it is hard to compute accuracy because TN is not available.

2. Precision measures the percentage of tuples that the NER labeled as protein are actually protein.

$$Precision = \frac{TP}{TP + FP}$$

where

$FP = \#$ of words labeled as protein by NER that are actually not protein.

Precision is useful in evaluating the performance of our NER, but when the NER predicts nothing in the text is a protein, this measure is meaningless since we will get 0 for both TP and FP.

3. Recall measures the percentage of protein entity that are labeled by NER.

$$Recall = \frac{TP}{TP + FN}$$

where

$FN = \#$ of proteins that are not labeled by NER

Recall is another useful way to evaluate the performance. However, when the true label of the test file is empty, recall is meaningless since we will get 0 for both TP and FN.

4. F-measure is a harmonic mean of precision and recall.

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

We use F-measure as a overall measure of performance. We also notice that the test data is not accurate because it missed some obvious protein (e.g. NFkappaB) in the true label.

Therefore, FP will be higher than it's actual value. As a result, recall will be a more reliable measure than precision. Hence, we are assigning more weight to recall in F-measure by using F_2.

4 Results

We first run the classification without merge algorithm, and get the following data

$$\begin{aligned} Precision &= 0.108196229206 \\ Recall &= 0.492586269327 \\ F_1 &= 0.177421869079 \\ F_2 &= 0.2879708040268277 \end{aligned}$$

Then, we run the classification with merge algorithm, and get the following data

$$\begin{aligned} Precision &= 0.158030439289 \\ Recall &= 0.370736182116 \\ F_1 &= 0.221600983679 \\ F_2 &= 0.29210319540696067 \end{aligned}$$

Both method takes about 360 seconds to train, 160 seconds to predict and calculate the evaluation statistics.

We notice that by adding the merge process, the precision goes up by 0.05, recall goes down by 0.12, both F_1 and F_2 is improved. This is because by merging single words, some fragments of protein words are combined to form the protein appearing in the true label, so we get a higher precision. On the other hand, since we are inevitably predicting less entities, we will suffer a lower recall.

5 Future Work

5.1 Dataset

We would like to find a more accurate dataset with more sufficient labels for multiple classes. Accordingly, we can improve our prediction results as well as finish the multi-class classification.

5.2 BILOU

Additionally, we could implement non-local features like BILOU or BIO to handle the multiple occurrences of entities in the text. (Ratinov and Roth, 2009)

References

Shumeet Baluja, Vibhu O. Mittal, and Rahul Sukthankar. 2000. Applying machine learning for high performance named-entity extraction. *Computational Intelligence*, 16(4):586–595.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition.