Named Entity Recognition and Normalization. A Domain-Specific Language Approach

Miguel Vazquez[1], Monica Chagoyen[2,3], and Alberto Pascual-Montano[2]

[1] Departamento de Ingeniería del Software e Inteligencia Artificial.
Corresponding author: miguel.vazquez@fdi.ucm.es
[2] Dpto. Arquitectura de Computadores, Universidad Complutense de Madrid, Madrid, Spain.
[3] Biocomputing Unit, Centro Nacional de Biotecnología - CSIC, Madrid, Spain

Summary. We present a tool that performs Named Entity Recognition and Normalization of gene and protein mentions on biomedical text. Several freely available tools perform the recognition of named entities. None that we known of, however, performs the Normalization -matching these names with the actual entities they refer to. The tool we present not only offers a complete solution to the problem, but it does so by providing easily configurable framework, that abstracts the algorithmic details from the domain specific. Configuration and tuning for particular tasks is done using domain specific languages, clearer and more succinct, yet equally expressive that general purpose languages. An evaluation of the system is carried using the BioCreative datasets.

1 Introduction

Three factors have motivated the biomedical community to get interested in literature mining [9]. The first one is thanks to recent high throughput techniques, like DNA microarrays, that, by involving large collections of entities, place bigger demands in the researcher's use of previous knowledge. The second factor is the fast pace at which scientific articles are being published, making it hard for researchers to keep up to date. Coupled with these is the trend followed by many editorials of making the articles available on-line, thus, making them readily available for analysis by software tools.

Named entity recognition (NER) is an important tool for literature mining as it is found as an initial step in many information extraction applications [10]. Named entity recognition has been used in domains other than the biomedical, like news wire text. It is in the biomedical domain, however, where it presents the most challenges. Gene and protein names are the most common subjects for NER; they have several characteristics that make them challenging. They are very numerous, millions, the number grows continuously, and the names used to refer to them are not necessarily standardized

[6]. Furthermore, in many cases the names are the same across different genes or match some common English words, producing a significant amount of ambiguity [1]. These problems not only affect finding the mentions in the text, but also the subsequent task of identifying the actual entity they refer to. This identification step is usually referred to as normalization, and is as much a challenging task as NER itself [2].

Several initiatives have dealt with the problem of NER in the biomedical context. We have studied closely the BioCreative competition [10, 2], which had several tasks relating to NER and normalization. We used some of the resources they provided to evaluate our system. As for the availability of tools for NER and normalization, we have found several freely available tools for the NER step alone. Some of them, like Abner and Banner [8, 5] we have studied in detail. We did not, however, find any available tool that implemented the normalization step. This has motivated us to develop our own.

Examining the state of the art in NER reveals a strong trend towards the use of conditional random fields (CRF). These probabilistic models have proven to be appropriate for text labeling [4], which is how NER is usually approached. For NER we use CRF to predict labels for a sequence of words, given a set of features associated with each word. Most of the work on NER using CRF differs basically in what these features are. In fact, the nature of the CRF algorithm reduces the problem to determining a suitable set of features. Most common features are orthographic: Has the word any uppercase letter? Any digits? Is there a slash character in the word? Etc.

We developed a system for both NER and normalization that implemented a standard approach, but one that offers a practical way to be extended and configured. For NER we implemented the same ideas from Abner and Banner, abstracted the core working and committed all the details on creating the features to a domain specific language (DSL). For the normalization task we developed our own solution, based on many ideas from the BioCreative competition, and also separated all the configuration options to domain specific languages.

2 System Overview

This is an overview of the process we have implemented. Following sections will cover each aspect in more detail, but this section should be useful to gain perspective for further discussion. The CRF model for NER is trained off-line. The training text is chunked into words, each word is labeled and turned into features. This labels and features are used to train the model. The normalization also needs preparation beforehand. A lexicon is read by the system (a file containing all the genes of interest along with their most common synonyms. Usually for one particular organism) and genes are processed one at a time. For each gene the system takes each of its synonyms, runs them through several index cue generators, each turning the synonym into a set of

variations. It then maps each cue to the given gene in the appropriate index, one for each cue generator. There are several indexes so that some could be more restrictive than others and, thus, preferred. The same cue might get mapped to several genes.

When new text arrives, the NER system chunks it into words, turns them into features and uses the model trained before to predict the labels. Using the labels, the system figures out the mentions and their boundaries. These mentions are then feed to the normalizer system.

The normalizer takes each of the mentions and begins a process of identifying the gene it refers to. The first thing it does is to turn the mention into cues, using the same index cue generators as before, and uses those cues to query the indexes in order. As soon as a match is made the matching stops, since the sooner the match is made the better the quality, if the indexes are sorted appropriately. The results are submitted to the next step. If no match is made the system drops this mention and moves along.

The index match might have render one gene or several, in any case, for each gene the system finds the synonyms in the lexicon and compares them to the mention. This comparison is done by breaking both the mention and the synonym into tokens, identifying the nature of this tokens and evaluating their similarities and differences. For each gene the best matching synonym is considered, and the value of the comparison servers both as a filter and as a ranking. A threshold is established so that genes not having similar enough synonyms are dropped.

If any genes pass the threshold, the best ranking one is selected. If there is a draw between several genes, the system disambiguates them. This is done by taking each of the genes, translating them to Entrez Gene codes, and taking the text from their description there. This descriptive text is then compared to the text from where the mention came from, and the gene with the most similar text is selected, or both, if the draw still holds.

3 Named Entity Recognition

The approach we follow for NER is based on that of Abner. We will describe briefly how it works, a more detailed description can be found in [4] or [8].

Finding mentions in text means determining the words that are likely to constitute a gene name. This is commonly done by labeling the words with their position relative to the mentions. The `IOB` schema that we use [7], uses the label `B` to mark the initial word on a mention, `I` marks any other word in the mention, and `O` is used for words that are outside of any mention. Named Entity Recognition boils down to determining what these labels would be for a given sequence of words.

The labeling of a sequence of words using Conditional Random Fields is done by representing each word by a set of features, and using a probabilistic model to determine the sequence of labels that most likely would have produce

a sequence of features like the one been analyzed. This model is built in an off-line phase using example data, in our case, the one provided as training for the BioCreative competition.

Our system uses CRF++ [3] to build the model and produce the labels, as it provides bindings for Ruby. Abner and Banner use Mallet, a Java implementation of CRF of similar characteristics. The main difference between all three systems strides in the features used to represent each word. Abner defines a set of features based on morphological features, these features are further expanded in Banner to include semantic features as well. Our system extracts all the feature generation from the rest of the system, and provides a DSL language to specify them. The features used to evaluate our system include those in Abner and Banner, excluding the syntactic based ones, as they slowed the system down considerably and they did not seem to enhance performance in our application significantly.

We will now take a look at the DSL used to define the features. Figure 1 shows the definition of three of the features that exemplify the three ways of defining features. The first one defines the feature `hasDigits` with a regular expression, so that if the word has a digit the feature is true. In the second case, `prefix_3` is defined with a regular expression capture, in the case the features value is what ever is captured in by the parenthesis, the first three characters. The last example uses a code block, the results of which, the word in lower case, is assigned to the feature. These three features, along with 25 others, are described in the default configuration file, and can be easily over written, and extended. Regular expressions are used as they are themselves a DSL for string matching, arguably the best way to capture the requisites of this particular domain.

```
hasDigits     /\d/
prefix_3      /^(...)/
downcase      do |w| w.downcase end
```

Fig. 1. Feature declaration

The CRF tool that we use, CRF++ [3], has a particular characteristic of being able to consider a certain context around each word at any particular step, as opposed to just the current state and word features. This context may be defined in another section of our DSL, as shown in figure 2, which just states that features `special, token2, isPunctuation and isDelim` from words surrounding the current one in distance of up three should be considered. The features used here must have been defined in the previous DSL.

```
context_features   %w(special token2 isPunctuation isDelim)
context_window     %w(1 2 3 -1 -2 -3)
```

Fig. 2. CRF++ context

4 Normalization

The Named Entity Recognition described before can only identify what could constitute a reference to a gene in the text. The actual gene that the mention refers is not determined by this process. Matching gene mentions to the actual genes is known as Normalization, although some systems have been proposed by BioCreative competitors that merge NER and Normalization in one single process. Normalization usually uses a lexicon that specifies, for each gene in a particular genome, all the synonyms used to refer to that gene in the literature. Similarity between the mention and the gene names is the main way to stablish matches. Our approach is based on a three step pipeline, finding first candidates genes for the mention, then selecting only those viable and, finally, choosing the best match among those, if any remain.

To find the candidate genes we use the idea of a name index. An index is built where the keys are names of genes and the values are the gene identifiers for genes that have that name. We then use a mention to query this index. To be more permissive, we allow these index to work, not only with the actual names and mentions, but with simplifications of them designed to increase matching. This way we stablish a hierarchy of indexes from more strict to more permissive. Each mention is matches against these indexes in order, stopping at the first match. The actual simplifications for each index are specified using a DSL, as exemplified in figure 3. Each name or mention is turned to several sets of cues, where each set of cues is aimed at a different index.

```
equal    do |w| [w] end
standard do |w| [w.downcase.split(/\s+/).sort.join("")] end
words    do |w|  w.scan(/[a-z]+/i)  end
```

Fig. 3. Declaration of name indexes

In the example figure 3 the first index we define is composed of the name or mention as-is; the second takes each word in the mention, sorts them, and joins them together in lowercase –this is a more accommodating cue; whereas the third index makes a cue for each word in the name or mention.

The candidates for each mention can contain plenty of false negatives, especially if very accommodating cues are used, therefore, it is necessary to filter out the worst and to select amongst the rest. In order to do so we created a

method that compares two names and evaluates their similarity. This method consists in chunking the name or mention down into it different components: number, letters, Greek names, Roman numerals, etc. For example, the mention `ASD-2 promoter` becomes the components `ASD, 2` and `promoter`. Each component –token from now on– is then identified and named. This identification is again defined using a DSL:

```
roman        /^[IV]+$/
promoter
greek        do |w| $greek[w.downcase] != nil end
```

Fig. 4. Token types

The three examples illustrate the three ways to define the token types. The first one identifies a token as a roman numeral using a regular expression. If only the name of the type is given, a default regular expression is attached, matching the same name, case insensitive, and with a possible "`s`" character at the end, to account for possible plurals. The last example uses a code block to make this check, in this particular case, it makes use of a hash of Greek letter names to check if the token is one of them, again, this is done case insensitive.

We now have each mention or name broken down into tokens, and the token types identified. In order to evaluate how similar a mention is to a certain name we need to evaluate the differences and similarities. This is done, again, using a DSL.

```
same.greek     5
miss.greek     -3
diff.promoter  -10
```

Fig. 5. Token comparison weights

Example figure 5 specifies three rules. The first states that if the same Greek letter name is found in the two mentions, a value of 5 is added to the similarity measure. The second states that if the mention is missing a Greek letter, 3 is subtracted, and the last one states that if one of them has the token `promoter` and the other does not, 10 is subtracted. We have four operators: `same, miss, extra` and `diff`. Meaning that they have a token in common, the mention is missing one, the mention has an extra token or that one of them has one the other does not. For each operator we can associate any of the token types declared in the previous step.

In order to add more flexibility to this method, we have added two other operators: `transform`, used to change tokens from one type to another, and `compare`, that allows comparing a certain type of tokens using a code block, thus giving more flexibility.

```
transform.roman do |t| [t[0].arabic, 'number'] end
compare.number do |l1,l2|
val = 0
val -= 4 if (l1 - l2).length >0 || (l2 - l1).length >0
val -= 8 if l1[0]  != l2[0] &&  l1[0]
val += 3 if l1[0]  == l2[0]
val
end
```

Fig. 6. Advanced comparisons: Transformations and custom comparisons

Figure 6 holds an example of both. The first one turns a roman to a number, allowing the number `1` to match `I` in roman form, for example. The second compares the numbers of mention and name in a finner grained manner.

We now have the ability to assign a similarity score between a mention and each of its candidate genes. For each candidate gene the best score amongst those from all it's known synonyms is selected. We use these score to rule out those candidates that score to low, and also to establish a ranking amongst those. If there are several candidate genes containing the same synonym (as it often occurs) they will score the same. In order to resolve the draws, we have a final step of disambiguation.

The disambiguation step is done by comparing the context in which the mention occurs, the paragraph for example, with the description of that gene in the Entrez Gene database. A code block can be provided in order to translate gene identifiers to Entrez Gene format if needed. The comparison is made by finding the words in common between the text in the mentions context and the text in the gene description, and adding their word weights. The word weights are their inverse document frequency calculated from a corpus of example text drawn from PubMed article abstracts.

5 Evaluation

The NER system comes with a set of default configurations mostly copied from Abner, but with a few additions. The system performs fairly well with these defaults for gene mention recognition. We plugged our system into the BioCreative competition evaluation sets using these defaults. The results are shown in table 1, we downloaded Abner and Banner and performed the same tests. RNer, our system, seems to outperform Abner slightly. Banner performs

significantly better than both, possibly because it uses syntactic information as features, which our default configuration did not include.

System	Precision	Recall	F-Measure
RNer	0.819	0.780	0.799
Abner	0.789	0.741	0.764
Banner	0.836	0.828	0.832

Table 1. Results for the BioCreative I task 1-B

The Normalization step also comes with a default configuration. We show results for the BioCreative I task 1-B in tables 2 and 3. These results are obtained using only the default configuration, no specific tuning is performed for either organism, nor is the provided training data used in any way. We have performed the analysis using our NER system, as well as Abner and Banner. Our NER system outperforms Abner and Banner in the yeast dataset, and Banner significantly outperforms both in the mouse dataset.

NER	Precision	Recall	F-Measure
RNer	0.936	0.863	0.898
Abner	0.941	0.809	0.870
Banner	0.933	0.816	0.870

Table 2. Results for the yeast dataset of the BioCreative I task 1-B.

NER	Precision	Recall	F-Measure
RNer	0.695	0.645	0.669
Abner	0.680	0.649	0.664
Banner	0.666	0.686	0.675

Table 3. Results for the mouse dataset of the BioCreative I task 1-B.

6 Discussion

Ruby, our implementation language, proved to be very useful due to its clear syntax and meta-programming possibilities. Ruby can also be compiled into Java bytecode and used in any Java framework. The result is a simple to use system that works out of the box but at the same time is flexible and easy to configure and adapt to other domains. Domain specific languages allow us to describe the specifics of the system in a more clear and succinct way.

While the system does not, for the time being, beat any record in performance, it shows competitive and promising results, even though no organism based tuning is performed and the training data for normalization is not being used.

7 Availability

This system was developed as part of another system called SENT, available at `http://sent.dacya.ucm.es`. Very likely, this system will be made available to public domain when SENT is published, along with the rest of the code that composes the SENT. In the meantime it will be provided free upon request.

8 Acknowledgements

This work has been partially funded by the Spanish grants BIO2007-67150-C03-02, S-Gen-0166/2006, CYTED-505PI0058, TIN2005-5619. APM acknowledges the support of the Spanish Ramón y Cajal program.

References

1. L. Chen, H. Liu, and C. Friedman. Gene name ambiguity of eukaryotic nomenclatures. *Bioinformatics*, 21(2):248–256, 2005.
2. L. Hirschman, M. Colosimo, A. Morgan, and A. Yeh. Overview of BioCreAtIvE task 1B: normalized gene lists. *BMC Bioinformatics*, 6(1):S11, 2005.
3. T. Kudo. Crf++: Yet another crf toolkit., 2005.
4. J.D. Lafferty, A. McCallum, and F.C.N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. *Proceedings of the Eighteenth International Conference on Machine Learning table of contents*, pages 282–289, 2001.
5. R. Leaman and G. Gonzalez. Banner: An executable survey of advances in biomedical named entity recognition. In *Pacific Symposium on Biocomputing*, volume 13, pages 652–663. Citeseer, 2008.
6. U. Leser and J. Hakenberg. What makes a gene name? Named entity recognition in the biomedical literature. *Briefings in Bioinformatics*, 6(4):357–369, 2005.
7. B. Settles. ABNER: an open source tool for automatically tagging genes, proteins and other entity names in text. *Bioinformatics*, 21(14):3191, 2005.
8. B. Settles, N. Collier, P. Ruch, and A. Nazarenko. Biomedical Named Entity Recognition using Conditional Random Fields and Rich Feature Sets. *COLING 2004 International Joint workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA/BioNLP) 2004*, pages 107–110, 2004.
9. H. Shatkay and R. Feldman. Mining the Biomedical Literature in the Genomic Era: An Overview. *Journal of Computational Biology*, 10(6):821–855, 2003.
10. A. Yeh, A. Morgan, M. Colosimo, and L. Hirschman. BioCreAtIvE task 1A: gene mention finding evaluation. *BMC Bioinformatics*, 6:1, 2005.