Protein Name Extraction

Xavier Adam
adamxavi@utt.fr

Olle Rundgren
Department of Computer Science
Lund University
olle.rundgren.913@student.lu.se

Abstract

This article describes a simple information extraction system using pattern recognition. The intended use of the program is to extract protein names from research articles. The article gives a short introduction to the background of our project and then goes on describing the implementation and structure of our system. Finally we evaluate our system and discuss what conclusions can be made from our work.

1 Introduction

Within the area of biotechnology and the life sciences there are vast amounts of information generated every day. The need for organizing this information is therefore of biggest concern and any method of automating this task is a welcome one. The FetchProt[1] and the Yapex[2] projects of the Swedish Institute of Computer Science (SICS[3]) adresses this need of automation and they were the starting point of our project. The FetchProt project is an ongoing project building a complete system. It will be able to extract proteins and their functions and inserting these into a database for wider use, both academic as well as industrial use. The Yapex project is the protein name tagger that FetchProt uses for the extraction of the protein names. We were interested in the information extraction part of the project and therefore decided to build a small system similar to the Yapex project. Our aim was to build a system that matched the results of the Yapex system using a pretty simple approach of pattern recognition.

[1] http://www.sics.se/humle/projects/fetchprot/
[2] http://www.sics.se/humle/projects/prothalt/
[3] http://www.sics.se/

2 Implementation and Structure

2.1 Language

We used Java for our implementation partly because of personal preferences but mostly because of its wide use and possibilty to find free modules to use from within our program. As it turned out we only used standard Java but found resources that would have been interesting to incorporate into our program. A specifically useful package was the standard package for regular expressions.

2.2 Corpus

The Corpus we used was downloaded from the Yapex site and was relatively small. Since it was already tagged we had to clean it from all tags before we could use it. The Corpus was originally collected from research articles from the MEDLINE resource and consists of the abstracts of randomly chosen articles. The training collection contains 1745 protein names and the test collection contains 1966 protein names. All of the abstracts were originally annotated by domain experts connected to the Yapex project.

2.3 System

Our system has a direct and simple structure. When given an input text we create a new object that we chose to call a BioText which holds a dictionary that can store protein names and a buffer to store the text. On this object we can call all methods associated with the different tasks. If the text needs to be cleaned from tags we call the cleaning method and store the new clean text in the buffer instead. For the task at hand (tagging of Protein names) we have two different methods that both takes regular expressions written as strings as parameters and which both fills the dictionary with protein names. The two methods differ in that one of them is for applying more specialized patterns catching few protein names with a high precision while the other method is for applying

a broad pattern cathing lots of protein names. Because of this implementation choice we could experiment with the patterns to make them better in a fast and easy way. We also have some methods to help us in analyzing the effectiveness of the patterns we write. For example we have a contextwriter that writes the context of each of the tagged protein names so that we can analyze what needs to be done. For the evaluation we had methods that could be applied directly to our BioText objects to compute recall and precision.

2.3.1 Different types of patterns

The most important part of our program is the patterns which detects the protein names to be tagged. They are all applied using the Java standard package for regular expressions though we wrote convenient methods to be able to just write the patterns as strings. At first we tried to apply all-catching patterns that catches a lot of protein names and we got a lot mismatches and didn't get all the names either. Therefore we realized that we had to build different kinds of patterns to get better results. The method we settled for was to have one broad pattern whith a filter to get rid of mismatches and several high precision patterns that catches pretty few names but with a higher precision (fewer mismatches). This choice resulted in two different methods which made it easy to experiment with different patterns. The first method is used for the high precision patterns and is typically used to detect multiword protein names. The second method is used to apply the broad pattern and also includes a filter to avoid common mismatches of names that look like protein names but in fact are not proteins for example DNA, RNA, UV and PH. This way to implement the pattern recognition was pretty fruitful since it was easy to write new patterns and to test and analyze these new patterns. The ability to test new patterns fast was very important to us since we are no experts on proteins and the structure of the names are not completely standardized. There were also the occurence of similar names that had to be taken into account like DNA and the introduction of the filter was a very important improvement and could easily be extended if other recurring mismatches were found. This iterative process was typical for our workflow. Both of the methods also had functionality to avoid tagging the same proteins more than once.

2.3.2 Main Parts of the Biotext class - The dictionary and the Buffer

As mentioned shortly in an earlier section each Biotext object holds a dictionary to store each occurence of protein names and a buffer to store the text to be tagged and later the tagged text. The dictionary was implemented with a map and has the protein names as keys and the number of occurences as values. When a pattern is applied with the appropriate method each protein that the pattern catches is stored in the dictionary. The Biotext objects also incorporates a buffer to store the text to be tagged. After each pattern applied the buffer changes accordingly with the proteins that the pattern has caught tagged in the buffer. When all patterns are applied the dictionary contains all the protein names that has been tagged and the buffer contains all the tagging of these protein names. Then different methods can be called to compute recall, precision and the result file can be written to a file. This makes the system easy to extend, improve and test and was also done several times before we reached the final results. The actual running of the program was as implied done from a seperate testclass.

3 Evaluation

For the evaluation we used methods for computing of recall, precision and using those two it was easy to make a printout for the harmonic mean (which is a combination of the two). We first built a baseline case which only tagged proteins in the test corpus which had been stored in the dictionary after applying patterns on the training corpus. That is we didn't apply any patterns at all on the test corpus. This gave us pretty poor results which was as expected. After that we applied patterns to the testcorpus as well and the results got better so we knew we were on the right track. We then wrote better patterns for the training corpus . The more patterns added the better it got and we used the evaluation methods to see how effective new patterns were. The wide pattern was the one with the biggest increase of recall and writing of the specialized patterns only made smaller increases but still good increase. It was also during early evaluation that we became really aware of deficiencies in our patterns that tagged names that shouldn‘t be tagged and therefore the evaluation process became an iterative process done several times before acceptable results were reached. Finally we settled for

a couple of high precision patterns and the filtered broad pattern and ran them on both the training corpus and the test corpus. The results including the original baseline case can be viewed in the table below.

	recall	precision
base	10.04%	45.08%
train	56.19%	52.24%
test	55.5%	52.6%

Table 1: Results

For further evaluation it would have been interesting to run some arbitrary but relevant articles through the system as well but due to time restrictions that was never done.

4 Conclusions

As with any project you are left with lots of thoughts about things that could have been done better and things that you are pretty happy about afterwards. With the methods we used we don't think we could do much better. Yes, we could write more of the specialized patterns but there wouldn't be any bigger point since the system isn't going to be used and increases after a while becomes very small. If the system were to be used for a "real" project we realize that we would have to do better than we have done so far but in that case we would need other methods to combine with the ones we have used. We looked into adding some kind of part-of-speech tagger and had actually started writing some methods for that as well. After a while though we felt that it was better to concentrate completely on one method since we didn't think we would make it in time otherwise. That was probably a wise choice. We experienced some trouble with the reading and understanding of the texts that we were working on. It was not an easy task to read and analyze text so out of context and so highly specialized. Finding effective patterns therefore became more difficault than we first thought. All in all it was very good training with the whole project since we got to adress problems that is probably very common in natural language processing and to work freely with something always inspire. It was also interesting to see that we could reach fairly good results with rather simple means.

5 Acknowledgements

We would like to thank both Pierre Nugues and Richard Johansson for the help we received during the project.

6 References

- http://www.sics.se/humle/projects/prothalt/
- http://www.sics.se/humle/projects/fetchprot/
- An Introduction to Language Processing with Perl and Prolog, Pierre Nugues