# Machine Learning for Tagging of Biomedical Literature

Caroline Persson

Kongens Lyngby 2012
IMM-BSc-2012-33

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-BSc-2012-33

# Summary

*Introduction*:
Named entity recognition of gene terms plays a big role in the increasing challenge of extracting gene terms from literature. Gene terms exists in many variants and the amount of gene terms is growing continuously. The goal of this project is to understand how the tagging of gene terms works, especially the understanding of the algorithms behind the recognition systems. A good understanding of the learning mechanisms is a necessary part of improving existing methods.

*Methods*:
The steps for training a Naive Bayes classifier are explained in details throughout the report. Examples of how the training compute different probabilities, and how the classifier handles raw unlabelled text are showed and evaluated. Furthermore a Naive Bayed classifier is implemented in Python, and the performance are compared to similar tasks.

*Conclusion*:
A Naive Bayes classifier is definitely an useful tool for named entity recognition of gene terms. The performance is dependent of the selection of features, and the final performance of an implementation in Python receive an f-measure of 0.58. This is comparable, though in the lower end, of the results from the BioCreative I challenge task 1.A.

# Preface

This is my closing project of my bachelor degree in Mathematics and Technology at the Technical University of Denmark (DTU) (from here denoted 'the project'). The project was suggested by Ph.D. student Kasper Jensen from Center for Biological Sequence Analysis (CBS) at DTU, and supervised by Finn Aarup Nielsen from the Department of Informatics and Mathematical Modelling (IMM).

The project digs in to the field of textmining of biomedical literature, with the aim of creating a named entity recognition system (NER) used for tagging gene and protein terms in biomedical literature.
A lot of effort was put into finding relevant data and templates showing how research had dealt with similar tasks. It turned out that the BioCreative [1], and especially task 1.A from 2004, dealt with similar challenges as the formulation of this project. Really convenient was the ability to compare the evaluation of the implementation presented in the project with similar work using the same data sets.
Python was used for all implementation, and in particular the package for Natural Language Processing (nltk) turned out to be helpful, including the tutorial book [2].

# Contents

Chapter 1

# Introduction

Unsuspected amounts of biological literature are freely accessible through online databases mainly provided by the United States National Library of Medicine (NLM). Due to the growing amount of literature, the need of developing and inventing of new robust and computational efficient algorithms for NER systems has become an important field of today's research. In general for most natural language processing tasks, supervised machine-learning methods are a kind of state of art. This is also the case for NER tasks concerning biological literature. For extracting gene and protein terms different statistical strategies (e.g. Bayes classifiers, Support Vector Machines (SVM), Hidden Markov Models (HMM), ect.), combined with rule-based orthographic features, pre- and postprocessing of data and dictionary scannings are some of the frequently used approaches. A problematic point of supervised learning is that the size of training data is essential to achieve good performance, but building a training corpus by human labeling is time consuming and expensive. Another difficulty in biological NER is that each named entity exists in multiple variant forms. Even though if a named entity already is defined in a dictionary, it is still difficult to recognize. The project deals with the challenge of finding relevant and freely available annotated data sets which can be used for training and testing, as well as finding or creating a dictionary of good quality which can be used as a gazetteer for looking up gene and protein terms. Here follow handling of data, selection of NER methods, implementation of algorithms, evaluation and considerations about modifications and improvements.

Chapter 1 is an introduction to NER of gene and protein terms. Chapter 2 describes data set and dictionaries used for the project. Chapter 3-5 covers the approaches used in the project. The architecture of the NER system from raw text to labelled output are described in details. Through examples the algorithms behind training of a part of speech (POS) tagger and a Naive Bayes classifier are described. Chapter 5 presents a final implementation of a Naive Bayes classifier for gene/protein entities. The classifier is compared with results from similar papers using the same data set. The results of the final classifier are presented in chapter 5, and finally in chapter 6 a conclusion about the usability of the implemented classifier and its performance is stated.

During this paper there is not drawn any distinction between gene and protein terms, and they are consequently used in the same context. Meaning that whenever any form of the word 'gene' is used in a context where it does not refer to a specific gene, we do actually mean both 'gene' and 'protein'. This also stress the fact that the project digs into the field of NER for gene **and** protein terms.

Chapter 2

# Data

## 2.1 Corpus

A well annotated data set is crucial for gaining a good performing NER system. When considering gene terms in biomedical text, it's important that the data is annotated consequently through the whole corpus following some specific annotation rules regarding how to decide whether a gene term should be tagged or not. Different types of data sets can be downloaded freely at NCBI's (National Center for Biotechnology Information) FTP (file transfer protocol) site. According to [3] two annotated datasets is more frequently used for training and testing of biomedical NER systems, respectively the corpora GENETAG [4] and GENIA [5]. The main differences between the two datasets are that the GENIA corpus is restricted to consist of 2000 MEDLINE abstracts retrieved by searching for the PubMed query for the three MeSH terms 'human', 'blood cells', and 'transcription factors'. Furthermore the tags are allowed to be generic, which means that not only specific tags such as 'tat dna sequence', but also non-specific terms such as e.g. 'dna sequence' would be legally tagged. The GENIA corpus was created not only for the purpose of being used for NER tasks concerning gene entities, but for a much wider range of biologically terms. In fact the corpus is tagged with 47 different biologically categories, for more details see [5]. On the other hand the GENETAG corpus consists of 20K sentences selected from unrestricted abstracts from MEDLINE, and the tags are restricted to be

specific gene entities only.
Even though the two datasets seem to be equally well annotated by experts, the GENETAG corpus has a simpler construction and a more intuitive interpretation. Moreover it does not contain information which is irrelevant for the purpose of this project, and the format of the corpus is easier to handle and work with in Python, which favours it over GENIA in this case.

The GENETAG corpus was used for the BioCreative (Critical Assessment of Information Extraction systems in Biology) challenge, which is a still on-going community-wide effort for evaluating text mining and information extraction systems applied to the biological domain [1]. The free version of the GENETAG corpus is the version used for the first BioCreative competition, Task 1.A, in 2004. The corpus has been updated since but it's not possible to download the latest version for free.
To ensure the heterogeneity in the GENETAG corpus, 10K of the sentences were chosen randomly among sentences from MEDLINE having a low score for term similarity to documents with known gene names, and 10K sentences were chosen randomly among sentences having a high score. This ensures that the corpus consisted of sentences with respectively many and few occurrences of gene entities. The data is divided into 4 subsets called train set, test set, round1 and round2. The train set consist of 7500 sentences, the test set of 2500 and the round1 and round2 of 5000 sentences each. Only the train, test and round1 set is public, the round2 set is kept secret in case of new tasks or competitions. This results in an available dataset of a total of 15000 sentences for training and testing.
Table 2.1 shows some statistics of the GENETAG corpus.

| | train set | test set | round1 | round2 | total |
|---|---|---|---|---|---|
| # sentences | 7,500 | 2,500 | 5,000 | 5,000 | 20,000 |
| # words | 204,195 | 68,043 | 137,586 | 137,977 | 547,801 |
| # tagged genes/proteins (G) | 8,935 | 2,987 | 5,949 | 6,125 | 23,996 |
| # alternatives to G | 6,583 | 2,158 | 4,275 | 4,505 | 17,531 |
| # genes/proteins in G with alternatives | 4,675 | 1,522 | 3,057 | 3,186 | 12,440 |

**Table 2.1:** Statistics of the different parts of the data

All gene entities in the corpus are tagged with 'NEWGENE'. To distinguish two different entities adjacent to each other in a sentence, the tag 'NEWGENE1' is used. The rest of the tokens are tagged with their POS tag. A list of all POS tags is in appendix 1. Each set of the data comes with two files 'Gold.format' and 'Correct.Data'. 'Gold.format' contains the true gene tags also called the golden standard for the data. 'Correct.Data' contains alternative acceptable tags. A typical example for a tagged sentence in the corpus containing both 'NEWGENE' and 'NEWGENE1' tags:

```
@@76576683396 TH-SH3/NEWGENE binding/JJ in/IN vitro/FW is/VBZ
abolished/VBN by/IN specific/JJ ,/, single/JJ amino/JJ acid/NN
substitutions/NNS within/IN the/DT Btk/NEWGENE TH/NEWGENE1
domain/NEWGENE1 or/CC the/DT Fyn/NEWGENE SH3/NEWGENE1
domain/NEWGENE1 ./.
```

The gene tags in the above sentence are represented in 'Gold.format' as:

1. @@76576683396 | 0 0 | TH-SH3
2. @@76576683396 | 15 15 | Btk
3. @@76576683396 | 16 17 | TH domain
4. @@76576683396 | 20 20 | Fyn
5. @@76576683396 | 21 22 | SH3 domain

and the alternatives in 'Correct.Data' are represented as:

1. @@76576683396 | 15 16 | Btk TH
2. @@76576683396 | 16 16 | TH
3. @@76576683396 | 20 21 | Fyn SH3
4. @@76576683396 | 21 21 | SH3

The numbers in between the pipe characters represent the index numbers of the first and last token of the tag in the sentence. '@@76576683396' is the unique identification number of the sentence. The example does really stress the ambiguity of gene tags. E.g. in the part of the sentence saying `[...within the Btk TH domain or...]`, the golden standard would be `[...within the Btk/NEWGENE TH/NEWGENE1 domain/NEWGENE1 or...]`, but the variations `[...within the Btk/NEWGENE TH/NEWGENE domain or...]` and `[...within the Btk/NEWGENE TH/NEWGENE1 domain or...]` are acceptable alternatives according to 'Correct.Data'. Notice how the tag `[...within the Btk/NEWGENE TH/NEWGENE domain/NEWGENE or...]` is not acceptable. The sentences were first tagged using an automated classifier, and afterwards corrected by experts using the guidelines in appendix 2 [4].

It is a general problem that there is no syntactic or contextual difference between the two tags 'NEWGENE' and 'NEWGENE1'. To deal with this problem all gene tags are changed to 'B' for tokens being the first token of a gene tag, and 'I' for tokens being a part of a gene tag, but not the first, in cases where the gene tag consists of multiple tokens. The sentence from before would now be:

```
@@76576683396 TH-SH3/B binding/JJ in/IN vitro/FW is/VBZ
abolished/VBN by/IN specific/JJ ,/, single/JJ amino/JJ acid/NN
substitutions/NNS within/IN the/DT Btk/B TH/B domain/I or/CC
the/DT Fyn/B SH3/B domain/I ./.
```

This corresponds to the IOB-format, which is widely used in natural language processing. In NLP tokens syntactic belonging to each other, so called chunks, will be tagged with I (inside), O (outside), or B (begin). A token is tagged as B if it marks the beginning of a chunk. Subsequent tokens within the chunk are tagged I. All other tokens are tagged O [6].

## 2.2 Dictionary

To build a gazetteer useful for the purpose of this project different online gene and protein databases are considered. An obvious choice was to use the UniProt Knowledgebase (UniProtKB), which is a protein database partially curated by experts and automated extracting systems. The database consists of two sections: UniProtKB/Swiss-Prot (containing reviewed, manually annotated entries) and UniProtKB/TrEMBL (containing unreviewed, automatically annotated entries). Meanwhile it turned out that working with dictionaries in Python with much more than 10,000,000 entries was too challenging for the memory of a standard laptop. The TrEMBL part of UniProtKB consisted of more than 20,000,000 entries, and it was not convenient to work with a dictionary of that size.
Another possibility was Georgetown University (Washington D.C.), who provides the web-based system 'BioThesaurus', which is designed to map a comprehensive collection of protein and gene names to UniProtKB protein entries. It covers all UniProtKB protein entries, and consists of several millions of names extracted from multiple resources based on database cross-references from more than 30 different sources. It is possible to download a list of all the protein terms

in 'BioThesaurus'. From year 2005 to 2010 there have been released yearly updates of the list. The first version from 2005 consists of around 4.6M entries, with 2.9M of them being unique terms, and the rest being alternatives/synonyms. The list from 2005 was extracted from 13 different data sources. The last, and 7th, version from 2010 consists of more than 40M entries with around 16.6M of them being unique terms, and the terms are extracted from 34 different data sources. Due to the fast increasing size of the amount of known gene and protein terms over the past decade, only the first three versions of the 'BioThesaurus' are considered as gazetteers. The latest versions are again too big and causes trouble in the implementation. For more details about 'BioThesaurus' see http://pir.georgetown.edu/pirwww/iprolink/biothesaurus/statistics.shtml#dh.
Finally two lists created by automated extracting systems of gene and protein terms from MEDLINE abstracts were considered. The first one called 'SemCat' consists of a large number of semantically categorized names relevant to genomics. The entities of the database are divided into more than 70 different biological categories, and the category 'DNA molecule' turned out to be the most frequent category containing gene tags from the GENETAG corpus. Because SemCat contains entities extracted by automated systems the data is by no means a comprehensive set of biomedical entities in MEDLINE, and we have to assume that 'SemCat' contains some wrong or too generic entities [7]. The other list of automatically generated entities is 'Gene.Lexicon', according to [8] 'Gene.Lexicon consists of 82% (+/-3%) complete and accurate gene/protein names, 12% names related to genes/proteins (too generic, a valid name plus additional text, part of a valid name, etc.), and 6% names unrelated to genes/proteins. Both 'SemCat' and 'Gene.Lexicon' is created by Tanabe et al., which also created the data set 'GENETAG' used for the BioCreative tasks and this project.

| | gene entries/Mbytes | % train set | % test set | year |
|---|---|---|---|---|
| 'Gene.Lexicon' | 1,145,391/45.2 | 0.489 | 0.486 | 2003 |
| 'SemCat' (DNA molecule) | 891,656/26.7 | 0.629 | 0.640 | 2006 |
| 'SwissProt' | 705,800/23.2 | 0.417 | 0.419 | 2012 |
| 'BioThesaurus3' | 9,941,868/357.2 | 0.681 | 0.685 | 2007 |
| 'BioThesaurus2' | 8,606,914/309.9 | 0.680 | 0.683 | 2006 |
| 'BioThesaurus1' | 4,646,029/171.8 | 0.661 | 0.665 | 2005 |

**Table 2.2:** Showing size of dictionaries, how many percent of the gene entities from respectively the train and test set are contained in the different dictionaries and what year the dictionary were updated.

Table 2.2 shows the size of each of the considered dictionaries, respectively the

number of gene terms and the size of the dictionary in python in Mbytes. Each gene entry from the train set and the test set was looked up, and the percentages shows how many of them were present in the dictionaries, only exact matches were allowed. A gene entry was counted to be present if either the entry from 'Gold.format' or one of its alternatives from 'Correct.Data' were present. Finally the table shows the year of release for each dictionary.
It turned out that the two dictionaries created by automated extraction from MEDLINE abstracts ('SemCat' and 'Gene.Lexicon') supplemented each other very well, and in any combinations of dictionaries including 'SemCat' and 'Gene.Lexicon' the percentages of gene match increased significantly. These two dictionaries are created by the same author, who created the data set, so most probably some similar annotation guidelines have been applied, and explains the good coverage. 'SemCat'+'Gene.Lexicon' covers 76.5% of the gene from the train set and 76.3% from the test set. Including 'BioThesaurus2' results in respectively 79.3% and 79.1%. Including the 'SwissProt' results in respectively 77.3% and 77.1%.
Based on testing different combinations of the dictionaries in 2.2. The conclusion is that 'SemCat'+'Gene.Lexicon' should be sufficient enough in this project, and the size of the combined dictionary of 'SemCat' and 'Gene.Lexicon' is easy and fast to load in python as well. The list below shows the sites from where the different dictionaries can be downloaded freely.

'Gene.Lexicon': ftp://ftp.ncbi.nlm.nih.gov/pub/tanabe/Gene.Lexicon.gz
'SemCat': ftp://ftp.ncbi.nlm.nih.gov/pub/tanabe/SemCat.tar.gz
'SwissProt': ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/uniprot_sprot.dat.gz
'BioThesaurusX': ftp://ftp.pir.georgetown.edu/databases/iprolink/
'TrEMBL': ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/uniprot_trembl.dat.gz

# Chapter 3

# Methods

The NER system is a constructed in 2 steps. The first step is to train a POS-tagger that is able to recognize gene tags, as well as standard POS-tags. The second step is to generate a feature set for training a NB classifier, which classifies tokens to be 'B' for tokens being the first part of a gene/protein tag, 'I' for tokens following a 'B' in cases where the gene/protein tag consists of more than one token, and finally 'O' for tokens not being gene tags.
Everything is implemented in Python using the 'natural language tool kit' package (nltk). The 7500 sentences from the train set is used for training of a POS-tagger and a NB classifer, such that the performance is comparable with related NER tasks using the same data. The test set consisting of 2500 sentences is used for testing and improving of the classifier, while the performance on the round1 set consisting of 5000 sentences is used for final testing and comparing. The performance of the round1 set was the deciding factor for choosing the winner of the BioCreAtIvE tasks.

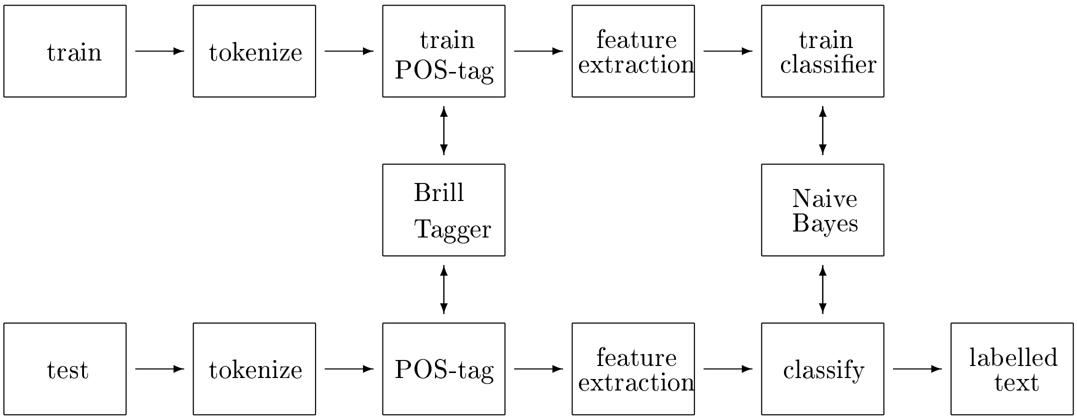

**Figure 3.1:** Overview of the NER system

Figure 3.1 shows the architecture of the NER system. The raw text is first tokenized, then tagged by a POS-tagger and analysed such that a feature set is generated. A NB classifier is fed with the feature set and determines the final classification of the tokens.

Evaluation is carried out relative to the tags that an expert would assign. Since we do not usually have access to an expert and impartial human judge, we make do instead with the gold standard and the alternatives for the test data represented in the files 'Gold.format' and 'Correct.Data'. The tagger is regarded as being correct if the tag it guesses for a given word is the same as the gold standard or one of the alternative tags.
Of course, the humans who designed and carried out the original gold standard annotation were only human. Further analysis might show mistakes in the gold standard, or may eventually lead to revised tag sets and more elaborate guidelines. Nevertheless, the gold standard is by definition "correct" as far as the evaluation of an automatic tagger is concerned.

## 3.1 Evaluation measure

For evaluation we count the true positives (tp), false positives (fp) and false negatives (fn), such that the recall, the precision and the f-measure can be determined.
The true positives are correctly classified gene entities. The false positive are non-gene entities which incorrectly are being classified as being genes. The false negatives are gene entities which are classified as not being a gene. Knowing

the counts of the tp, fp and fn it is possible to calculate the three measures:

$$\text{recall} = \frac{tp}{tp + fn} \tag{3.1}$$

$$\text{precision} = \frac{tp}{tp + fp} \tag{3.2}$$

$$\text{f-measure} = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3.3}$$

The recall is the fraction of the gene entities from the whole corpus which are retrieved by the classifier. On the other hand the precision is the fraction of correct gene entities among the retrieved ones. The F-measure is a weighted average of the precision and the recall, or more precisely denoted as the harmonic mean of the recall and the precision. Table 3.2 shows how the harmonic mean of the recall and the precision differs from a normal arithmetic mean.

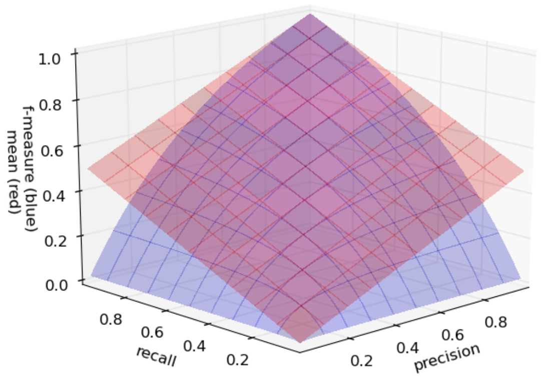

**Figure 3.2:** The graphical difference between the harmonic mean and the arithmetic mean of the precision and the recall.

The advantage of the harmonic mean is its availability to penalize the f-measure when the difference between the recall and the precision increases. If the recall and the precision are equal, then the f-measure will equal the arithmetic mean of the recall and the precision, which is desirable. If the recall and precision are subject to a mean-preserving spread, which means that they are "spread apart" from each other while leaving the arithmetic mean unchanged, then the harmonic mean will always decrease. A high f-measure ensures that both the recall and precision are high as well. Throughout this project, recall and precision are weighted equally.

## 3.2 Tokenization

The raw text (train/test) are tokenized before used it can be as train/test data. Tokenizing of the text has the purpose of identifying each element in a sentence. Each element is called a token and will be assigned a POS-tag, and later classified as being a gene or not. Hyphens "-" and the genetiv marker, "'s", are the only cases where signs are not individual tokens. The sentence

'A man's world, but it would be nothing without a woman.'

would be tokenized into the following 14 tokens:

A man 's world , but it would be nothing without a woman .

## 3.3 POS-tagger

The tools available in Pythons nltk-package have been used for training of a POS-tagger with the train set form GENETAG. To start with three different kinds of taggers are considered:

### 3.3.1 Regular expression tagger

The regular expression tagger, determines the tag of a given token from some user defined orthological rules. As an example we implement a regular expression POS-tagger with the following determining rules. If the token:

- consists of only numbers eventually with some symbols attached: 'CD'
- is 'The', 'the', 'A', 'a', 'An' or 'an': 'DT'
- ends on 'able', 'ible', 'esque', 'less', 'ive', 'ish', 'ous', 'ic', 'ical', 'ful': 'JJ'
- ends on 'ness', 'ity', 'ship', 'tion', 'sion', 'ment', 'ist', 'ism', 'dom', 'ence', 'ance', 'al', 'acy', 'er', 'or': 'NN'

- ends on ’ly’: ’RB’
- ends on ’s’: ’NNS’
- ends on ’ing’: ’VBG’
- ends on ’ed’: ’VBD’
- is not obeying any of the above rules: ’NN’

This means that a lot of tokens will just be assigned as ’NN’, but the advantage of this tagger is its ability to tag every token, including tokens which has never seen before.
See appendix 1 for a list of the different POS-tags in the train set.

### 3.3.2 Unigram tagger

The unigram tagger is based on a simple statistical algorithm. It simply assigns the most likely tag to a given token based on the counts of tags of that given token in the train set. Taking a closer look at the word ’function’, shows that the distribution of the tags of ’function’ in our train set are ’NN’ 6826 times, ’VB’ 403 times, ’VBP’ 101 times and ’MIDGENE’ 33 times. The unigram tagger will tag ’function’ as ’NN’ everytime it occurs with no concerns about the words context.

### 3.3.3 N-gram tagger

The N-gram tagger is a generalization of the unigram tagger, instead of only considering a single isolated token, N=1, the N-gram tagger considers every token in the context of N-1 preceding tags. This means that the tagger is still not able to tag tokens not being present in the train data, and furthermore an N-gram tagger will not able to tag tokens occuring in a context not present in the train data. Figure 3.3 shows the concept of a 3-gram tagger also known as a trigram tagger.

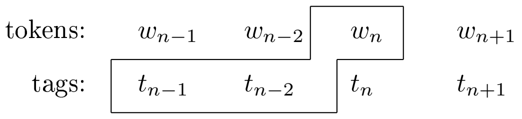

**Figure 3.3:** The concept of an trigram tagger

### 3.3.4 Combining taggers

As mentioned above the N-gram taggers are not able to tag tokens if the context they appear in have not been present in the train data. Assuming a bigram tagger (N=2) and the word 'function' occurring after a token tagged as 'JJ'. If 'function' never appeared in the train data with a preceding 'JJ' tag, then a bigram tagger will fail to assign 'function' in this context, and 'function' will be assigned the tag 'None'. The word following 'function' will then have the preceding tag 'None'. It is most probably that this following word has never occurred in the context with a preceding 'None' tag in the train data, and again the bigram tagger will fail to assign the word any tag but 'None'. This goes on throughout the sentence, and results in a very poor performance of the N-gram taggers. As N gets bigger the performance gets worse, due to the increasing specificity of the context.
Biomedical literature are often characterized as sparse text, because of many technical and specific terms, which occurs very seldom in common literature. This is a problem because as soon as an N-gram tagger meets a word not present in train data, the tagger is unable to assign it any tag, and a chain-reacting of 'None' tags throughout the sentence will begin.
To solve this problem it is possible to add one or more backoff-taggers to a given tagger, such that if e.g. a bigram tagger is not able to assign a tag to a given token, then it will try with a unigram tagger instead. In case the unigram tagger should fail as well, a regular expression tagger could be a backoff-tagger for the unigram tagger, such that 'None' tags are avoided.
After POS-tagging text, the tags are corrected such that an 'I' tag having neither a preceding 'I' or 'B' tag are changed to 'B', while no entities can start with an 'I' tag.

Table 3.1 shows the performances of different combinations of a regular expression, unigram, bigram and trigram tagger on the test set. The accuracy are the fraction of tokens in the test set, which are tagged with the correct POS-tag. The accuracy then tells us something about how well a given tagger performs not only on gene/protein tags but for all POS-tags. On the other hand the recall, precision and f-measure do only concern the gene/protein tags. Notice the poor performance of the N-gram taggers without any backoff-taggers, and in particular the poor ability to recognize gene/protein terms. Table 3.1 also shows that # 7 receives the best accuracy, while # 15 receives the best f-measure for the gene/protein tags. Tagger # 7 and # 15 are picked out and analysed further throughout this chapter.

| # | tagger | backoff 1 | backoff 2 | backoff 3 | accuracy | recall | precision | f-measure |
|---|---|---|---|---|---|---|---|---|
| 1 | reg. expr. | | | | 28.750% | - | - | - |
| 2 | unigram | | | | 80.944% | 38.768% | 40.632% | 39.678% |
| 3 | unigram | reg. expr. | | | 84.188% | 38.768% | 40.632% | 39.678% |
| 4 | bigram | | | | 15.251% | 4.720% | 58.506% | 8.736% |
| 5 | bigram | reg. expr. | | | 84.188% | 19.417% | 58.943% | 29.212% |
| 6 | bigram | unigram | | | 82.811% | 39.873% | 52.769% | 45.423% |
| 7 | bigram | unigram | reg. expr. | | **85.774**% | 39.839% | 58.419% | 47.373% |
| 8 | trigram | | | | 8.950% | 1.440% | 44.330% | 2.788% |
| 9 | trigram | reg. expr. | | | 73.385% | 7.800% | 47.648% | 13.406% |
| 10 | trigram | unigram | | | 82.555% | 40.074% | 48.092% | 43.718% |
| 11 | trigram | unigram | reg. expr. | | 85.531% | 40.074% | 52.500% | 45.453% |
| 12 | trigram | bigram | | | 15.177% | 4.720% | 58.506% | 8.736% |
| 13 | trigram | bigram | reg. expr. | | 81.025% | 19.652% | 58.119% | 29.372% |
| 14 | trigram | bigram | unigram | | 82.837% | 40.308% | 52.691% | 45.675% |
| 15 | trigram | bigram | unigram | reg. expr. | 85.661% | 40.241% | 58.208% | **47.585**% |

**Table 3.1:** Performance of different combinations of pos-taggers on the test set

### 3.3.5 Brill tagger

The Brill tagger is another type of POS-tagger, which improves a given initial tagger. Using a Brill tagger makes it possible to improve the N-gram taggers. The Brill tagger works the way that it starts out with a corpus tagged by a given initial POS-tagger. It looks up the mistakes in the tagged corpus, and given the mistake it tries to come up with some correction rules that repairs the mistakes. The tagged corpus gets updated with every new correction rule. This continues until a certain user specific number of correction rules have been applied to the POS-tagger or a user specified minimum score is reached. The Brill tagger has the advantage over the N-gram taggers that it is capable of looking up preceding and following words as well as tags. The user creates a template for the Brill tagger that tells it which patterns and sequences of preceding and following tags and words to take into account. The correction rules are of the structure "replace $T_1$ with $T_2$ in the context C" e.g. "replace 'NN' with 'I' if the tag of

the following word is 'I'" or "replace 'NNP' with 'B' if the preceding tag is 'JJ' and the following word is 'gene'", ect.
To correct a wrong tag, $t_0$, of a word, $w_0$, the Brill tagger is told to look at the specific tags $t_{-2}$, $t_{-1}$, $t_1$ and $t_2$ and the specific words $w_{-2}$, $w_{-1}$, $w_1$ and $w_2$. Additional the tagger is able to create correction rules based on occurrences of particular words or tags surrounding the wrong tag, e.g. a rule could be "replace $T_1$ with $T_2$ if the the tag/word $x$ occurs among the following/preceding $i$th tags/words". The Brill tagger are initiated such that $i$ can take the values 2 and 3.
The Brill tagger outputs a score for every generated correction rule. The score, $s$, is defined as $s = fixed - broken$, where $fixed$ is the number of times the rule changes an incorrect tag to a correct tag, and $broken$ is the number of times the rule changes a correct tag to an incorrect tag. The Brill tagger was trained with a limit of maximum of 1000 rules and a minimum score of 2.

| tagger | initial tagger 1 | accuracy | recall | precision | f-measure |
|---|---|---|---|---|---|
| # 7 | - | 85.774% | 39.839% | 58.419% | 47.373% |
| # 15 | - | 85.661% | 40.241% | 58.208% | 47.585% |
| Brill | # 7 | 87.010% | 42.250% | 62.599% | 50.450% |
| Brill | # 15 | 86.593% | 42.417% | 61.297% | 50.139% |

**Table 3.2:** Performance of Brill taggers compared to the best N-gram taggers from table 3.1

The Brill taggers in table 3.2 were trained on the train set, which was initial tagged by respectively tagger # 7 and tagger # 15 from table 3.1. The accuracy, recall, precision and f-measure are equivalent to 3.1 and based on the test set as well. The Brill tagger initiated by tagger # 7 receives the best f-measure for the gene/protein tags, this tagger will be used in the further implementation of a NER system.

The 10 correction rules having the highest score for tagger # 7 are shown in table 3.3. Comparing with the 5 rules having the smallest score in 3.4, it is clear how the rules get more concerned about particular words, whereas the rules in table 3.3 in particular concern the gene tags, 'B' an 'I', and tags in general.

| **Score** | **Fixed** | **Broken** | **Other** | **Rule** |
|---|---|---|---|---|
| 153 | 153 | 0 | 153 | NN -> I if the tag of the following word is 'I' |
| 148 | 149 | 1 | 8 | IN -> RB if the text of words i+1...i+2 is 'as' |
| 147 | 164 | 17 | 35 | JJ -> B if the tag of the following word is 'I' |
| 93 | 95 | 2 | 0 | ( -> I if the tag of the following word is 'I' |
| 91 | 99 | 8 | 99 | . -> CD if the tag of words i+1...i+3 is 'CD' |
| 65 | 67 | 2 | 0 | I -> B if the tag of the preceding word is 'DT', and the tag of the following word is 'I' |
| 57 | 64 | 7 | 14 | NN -> VB if the text of the preceding word is 'as', and the text of the following word is 'as' |
| 50 | 57 | 7 | 40 | NN -> B if the tag of the following word is 'I' |
| 46 | 46 | 0 | 3 | SYM -> I if the tag of the following word is 'I' |
| 40 | 41 | 1 | 1 | DT -> IN if the tag of the preceding word is 'VBD', and the tag of the following word is 'DT' |

**Table 3.3:** The 10 correction rules with the highest score from the training of the Brill tagger

| **Score** | **Fixed** | **Broken** | **Other** | **Rule** |
|---|---|---|---|---|
| 2 | 2 | 0 | 0 | VBP -> VB if the text of the preceding word is 'others' |
| 2 | 2 | 0 | 0 | VBZ -> NNS if the text of words i-2...i-1 is 'spirochaete' |
| 2 | 2 | 0 | 0 | WDT -> IN if the text of the preceding word is ')', and the text of the following word is 'encodes' |
| 2 | 2 | 0 | 0 | WDT -> IN if the text of words i-2...i-1 is 'cluster' |
| 2 | 2 | 0 | 0 | WDT -> IN if the text of the following word is 'on' |

**Table 3.4:** The 5 correction rules with the lowest score from the training of the Brill tagger

## 3.4 Naive Bayes Classifier

To do the final classification a Naive Bayes (NB) classifier is trained to distinguish genes from non-gene entities. The classifier distinguish between three classes; 'B', 'I' and 'O', which are initiated by the Brill tagger from previous section. The NB classifier is trained on a set of features, which are meant to identify the gene entities, and in particular the entities not found by the POS-tagger. A feature could e.g. be the POS-tag of the token, whether the token is a word consisting of both lower and upper case letters, whether the token is a digit or not, ect.

A NB classifier is trained in Python, using the nltk package. The classifier works such that a set of features relevant to the different classes are specified by the user. The main purpose is to determine the class of a token, $t$, given the feature set, $f_t$, belonging to the particular token by computing the posterior probability $P(class|f_t)$. Using Bayes theorem:

$$P(class|f_t) = \frac{P(f_t|class) \cdot P(class)}{P(f_t)} \tag{3.4}$$

### 3.4.1 Exploring feature sets

To demonstrate how Python's NB classifier works an example is provided. The example is a simplified version of the final implementation. Considering the following three features:

feature 1: token's POS-tag is 'B' or 'I'

feature 2: token occurs in the dictionary & POS-tag is not present in {IN, DT, CC, PRP, WDT, MD, PDT, WP, TO, EX, WRB, RP }

feature 3: token consists of at least one upper case letter and one digit

The NB classifier is trained separately on each of the 7500 train sentences, and to each token in each sentence belongs a feature set. Table 3.5 shows how the classifier handles an input sentence for training. The sentence are first POS-tagged by the previous trained brill tagger, and then a feature set for each of the tokens are generated. The brill tagger has an overall accuracy on the train set of 0.97, and evaluation measures for the gene/protein tags are: recall = 0.917, precision = 0.908 and f-measure = 0.912. The fact that not 100% of the gene/protein entities are recognized correctly in the train set, will result in some wrongly generated features for the features depending on the POS-tags. E.g. it will be possible to have a token which truely should have a gene tag, but has been assigned a wrong POS-tag by the brill tagger. The beginning of the following sentence is used to exemplify how the feature sets for the train data are generated and used for training.

```
The LMW FGF-2 up-regulated the PKC epsilon levels by 1.6-fold
; by contrast the HMW isoform down-regulated the level of this
PKC isotype by about 3-fold and increased the amount of PKC
delta by 1.7-fold .
```

| token | The | LMW | FGF-2 | up-regulated | the | PKC | epsilon | levels | by |
|---|---|---|---|---|---|---|---|---|---|
| true class | O | B | I | O | O | B | I | O | O |
| brill tag | DT | B | I | IN | DT | B | I | NNS | IN |
| feature set | | | | | | | | | |
| feature 1 | False | True | True | False | False | True | True | False | False |
| feature 2 | False | True | True | True | False | True | True | False | False |
| feature 3 | False | False | True | False | False | False | False | False | False |

**Table 3.5:** The feature sets for the first nine tokens of the sentence.

For this particular train sentence the NB classifier would get the train input as tuples representing each token, see figure 3.4.

$$\left[\left(\begin{array}{l}\text{feature 1: False}\\ \text{feature 2: False}\\ \text{feature 3: False}\end{array}, \text{'O'}\right), \left(\begin{array}{l}\text{feature 1: True}\\ \text{feature 2: True}\\ \text{feature 3: False}\end{array}, \text{'B'}\right), \left(\begin{array}{l}\text{feature 1: True}\\ \text{feature 2: True}\\ \text{feature 3: True}\end{array}, \text{'I'}\right)\ldots\right.$$

**Figure 3.4:** The first three tuples of the sentence from table 3.5.

To save time only the first 10 sentences of the train set is used for this example. The 10 sentences consist of 239 tokens and 13 gene entities. Most of the gene entities consist of more than one word, so the total number of 'B' tags is 13 and the total number of 'I' tags is 25, leaving 201 tokens with 'O' tags. Figure

3.5 shows the contingency tables of the counts for each feature. The posterior probability, $P(class|f_t)$ for the three classes are based upon the counts from the contingency tables in figure 3.5.

| | True | False | |
|---|---|---|---|
| O | 1 | 200 | 201 |
| B | 12 | 1 | 13 |
| I | 22 | 3 | 25 |
| | 35 | 204 | 239 |

**(a)** Counts for feature 1

| | True | False | |
|---|---|---|---|
| O | 50 | 151 | 201 |
| B | 11 | 2 | 13 |
| I | 23 | 2 | 25 |
| | 84 | 155 | 239 |

**(b)** Counts for feature 2

| | True | False | |
|---|---|---|---|
| O | 16 | 185 | 201 |
| B | 9 | 4 | 13 |
| I | 7 | 18 | 25 |
| | 32 | 207 | 239 |

**(c)** Counts for feature 2

**Figure 3.5:** Counts for the features

Three features with each two outcomes, True and False, results in a total of 8 feature sets. Table 3.6 shows how tokens will be classified according to what feature set they match. The probabilities are computed in Python simply using the command below for each of the classes, here shown with the class 'B':

```
>>> classifier.prob_classify({'feat1':True, 'feat2': True, ...
'feat3': True}).prob('B')
0.5253238661014447
```

| feature 1 | feature 2 | feature 3 | B | I | O | classification |
|---|---|---|---|---|---|---|
| True | True | True | 0.5253 | 0.4720 | 0.0026 | B |
| True | True | False | 0.1725 | 0.8070 | 0.0205 | I |
| True | False | True | 0.6627 | 0.2914 | 0.0459 | B |
| True | False | False | 0.2027 | 0.4642 | 0.3330 | I |
| False | True | True | 0.1290 | 0.1503 | 0.7207 | O |
| False | True | False | 0.0072 | 0.0434 | 0.9494 | O |
| False | False | True | 0.01271 | 0.0072 | 0.9800 | O |
| False | False | False | 0.0005 | 0.0016 | 0.9978 | O |

**Table 3.6:** Classifications of the different feature sets, and the posterior probabilities for each class

Table 3.6 shows the different feature sets, and the posterior probabilities of the classes for each of the sets. The class with the biggest posterior probability is assigned to the token. Let us assume that the sentence from table 3.5 was not one of the 10 train sentences, and using the feature sets from table 3.5, the classification according to table 3.6 would be:

| token | The | LMW | FGF-2 | up-regulated | the | PKC | epsilon | levels | by |
|---|---|---|---|---|---|---|---|---|---|
| true class | O | B | I | O | O | B | I | O | O |
| alternative | O | O | B | O | O | B | O | O | O |
| assigned class | O | **I** | **B** | O | O | **I** | **I** | O | O |
| corrected | O | **B** | **B** | O | O | **B** | **I** | O | O |

**Table 3.7:** The classes assigned to the sentence from table 3.5

Table 3.7 shows the classes assigned to the sentence from table 3.5. In the first try the classifier is wrong for both gene entities. Apparently the distinction between 'B' and 'I' is not that good, but it is irrelevant, while this is only meant as a demonstration explaining how the system works. The 'I' assigned to the token 'LMW' and 'PKC' is changed to B, because the preceding class is 'O', and a gene entity can not start with 'I'. This corrects the classification such that the gene entity 'PKC epsilon' is correct. The entity 'LMW FGF-2' is now classified as two separate gene entities 'LMW' and 'FGF-2'. Both 'LMW FGF-2' and 'FGF-2' alone will be considered as true positive, while the 'FGF-2' is a legal alternative to 'LMW FGF-2'. But 'LMW' can not stand alone. After correction of 'I' to 'B' the classification results in 2 true positives ('FGF-2' and 'PKC epsilon'), 1 false positive ('LMW') and 0 false negatives. Repeating the evaluation measure from chapter 3 we get:

$$\begin{aligned}\text{recall} &= \frac{tp}{tp+fn} = \frac{2}{2} = 1.0\\ \text{precision} &= \frac{tp}{tp+fp}\frac{2}{3} = 0.67\\ \text{f-measure} &= 2\cdot\frac{precision\cdot recall}{precision+recall} = \frac{2}{5} = 0.80\end{aligned}$$

### 3.4.2 Bayes Theorem and additive smoothing

Given the feature set and the counts from figure 3.5, the probabilities for the different classes are computed using equation 3.4:

$$P(class|f_t) = \frac{P(f_t|class)\cdot P(class)}{P(f_t)}$$

Because the denominator is the same for each of the three classes given a particular feature set, it is only necessary to compute the numerator. The token will be assigned the class that results in the largest numerator in equation 3.4. The Naive Bayes classifier assumes that the features are independent, such that the posterior probability for a feature set $f = \{f_1, f_2, f_3\}$, given a class, $C$, is:

$$P(f|C) = P(f_1|C)P(f_2|C)P(f_3|C) \tag{3.5}$$

Using equation 3.5 it is now possible to calculate the numerator of equation 3.4:

$$p^C_{num} = P(f|C)P(C) = P(f_1|C)P(f_2|C)P(f_3|C)P(C) \tag{3.6}$$

To find the most likely class given a feature set, we calculate $p^C_{num}$ for each class $C = \{$ 'B', 'I', 'O' $\}$ . $p^C_{num}$ is normalized by $\sum (p^B_{num}, p^I_{num}, p^O_{num})$, such that the probabilities sum to one.

To ensure that equation 3.6 will not equal zero, which will happen in the case where one of the counts in figure 3.5 is zero, additive smoothing with parameter $\alpha$ is applied when computing the probabilities. This means that instead of computing the likelihood $\frac{c}{N}$, with $c$ being the count and $N$ being the marginal totals for the given classes, the likelihoods are modified such that:

$$P(f|C) = P(f_1|C)P(f_2|C)P(f_3|C) = \prod_{i=1..C} \left( \frac{c_i + 0.5}{N + B \cdot 0.5} \right) \tag{3.7}$$

for $C$ being the number of classes and $\alpha = 0.5$.

Back to the example, we want find the most likely class given the feature set $f = \{\text{True, True, True}\}$. The $p^C_{num}$, is calculated for each class, $C = \{\text{B,I,O}\}$, and normalized by the sum $\sum_C p^C_{num}$.

$$P(B|f) = P(f_1 = \text{True}|\text{B})P(f_2 = \text{True}|\text{B})P(f_3 = \text{True}|\text{B})P(\text{B})$$
$$= \frac{12.5}{14.5} \cdot \frac{11.5}{14.5} \cdot \frac{9.5}{14.5} \cdot \frac{13.5}{240.5} \cdot \frac{1}{P_{sum}} = 0.5253$$

$$P(I|f) = P(f_1 = \text{True}|\text{I})P(f_2 = \text{True}|\text{I})P(f_3 = \text{True}|\text{I})P(\text{I})$$
$$= \frac{22.5}{26.5} \cdot \frac{23.5}{26.5} \cdot \frac{7.5}{26.5} \cdot \frac{25.5}{240.5} \cdot \frac{1}{P_{sum}} = 0.4720$$

$$P(O|f) = P(f_1 = \text{True}|\text{O})P(f_2 = \text{True}|\text{O})P(f_3 = \text{True}|\text{O})P(\text{O})$$
$$= \frac{1.5}{202.5} \cdot \frac{50.5}{202.5} \cdot \frac{16.5}{202.5} \cdot \frac{201.5}{240.5} \cdot \frac{1}{P_{sum}} = 0.0026$$

$$P_{sum} = \frac{12.5 \cdot 11.5 \cdot 9.5 \cdot 13.5}{14.5^3 \cdot 240.5} + \frac{22.5 \cdot 23.5 \cdot 7.5 \cdot 25.5}{26.5^3 \cdot 240.5} + \frac{1.5 \cdot 50.5 \cdot 16.5 \cdot 201.5}{202.5^3 \cdot 240.5} \approx 0.04787$$

This is how the NB-classifier in Python works and the results agree with the probabilities in table 3.6. 'B' is the most likely class, and 'B' is assigned to tokens with the feature set $\{True, True, True\}$.

CHAPTER 4

# Implementation of NB-classifier

## 4.1 Feature selection

A good choice of features is important for receiving a good performance. Because the number of non-gene terms in data ('O'-tagged tokens) are many times bigger than the number of gene terms ('B' and 'I' tags), it is hard to find features which do favour the 'B' and 'I' tags over 'O'. An exceptions are the features 'POS-tag == 'B'' and 'POS-tag == 'I'', where 'POS-tag' refer to the tag assigned by the brill tagger prior to the classification. The features resulting in the best classification were features aimed to identify gene terms.
Only boolean features were considered, table 4.1 shows a list of the orthographic features used for classification. To supplement the orthographic features a dictionary of generic high frequent gene tokens were extracted from the train set. The list contained words such as 'gene', 'mutant', 'factor', 'dna', 'receptor', 'protein', ect. which can never be tagged as individual entities, but often occur somewhere inside an entity, e.g. 'p53 gene', 'mutant EphB1 receptor', ect. The dictionary, $D$, from chapter 3 and this generic dictionary, $d$, were used as part of the features as well, e.g. '$t_n$ exists in $D$ and $t_{n+1}$ exists in $d$' would be true for 'p53 gene', where $t_n$ = 'p53'.
Features were generated for one sentence at a time, which made it possible to include previous tokens and their POS-tags. Each token is represented as a

| fetaure | example |
|---|---|
| All caps | RXR, AE, YREA,.. |
| Mix caps | DNase, CopG, apoB,.. |
| Alpha + digit | CPA1, p53, Rat7p,.. |
| Natural number | 0,1,2,.. |
| Real number | 0.05, 23.5, 1,345 |
| Roman | VIII, ix, II,.. |
| Contain dash | COUP-TFII, c-fos, 5-HT2C-R,.. |
| Sequence | TTACTA, GGGCTA, UGCAA,.. |

**Table 4.1:** List of the orthographic features.

tuple $(w_n, pt_n)$, where $w_n$ is the word/token and $pt_n$ is the POS-tag. A possible feature could be "$w_n$ is a roman number and $pt_n ==$ 'B'", which would be true for roman numbers following a 'B' POS-tag.

## 4.2 Evaluation

Table 4.2 shows the performance of the classifier for the test set and round1:

| | precision | recall | f-measure |
|---|---|---|---|
| test set | 0.561 | 0.594 | 0.577 |
| round1 | 0.565 | 0.613 | 0.588 |

**Table 4.2:** Evaluation measure of NB-classifier

Figure 4.1 shows how our final f-measure would have been ranked in the BioCreative task 1.A, where the same data set was used for training a NER system for extracting genes and protein terms.

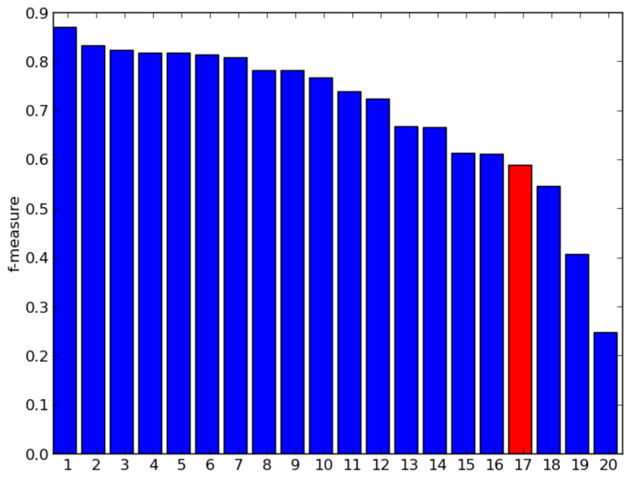

**Figure 4.1:** Graphical view of the f-measures received by the teams participating in BioCreative task 1.A (blue) [9], and the NB-classifier (red).

CHAPTER 5

# Further work and improvements

The NB classifier is a fast an efficient learning algorithm, and it is just a few seconds for training with Python. A NB classifier has the advantages that it is robust to irrelevant features, which will be uniformly distributed over all classes, and the conditional distribution for the given feature will have no impact on the computation of the posterior probabilities.
The learning algorithms showing best results in the BioCreative tasks have been conditional random fields. Using more advanced statistical algorithm will most likely improve performance. Also a more comprehensive feature selection and extraction will as well show improvements. The NB classifier is not suited for dependent features, and it can be hard in practise to ensure independence among all features.
During the project different values were tried for the probability limit trying to catch some false positives. E.g. a token would only be assigned 'O' if $P(O|f_t) > 0.7$, if $P(O|f_t) < 0.7$ the token would be assigned to either 'B' or 'I' dependent on which of $P(B|f_t)$ and $P(I|f_t)$ had the biggest value. It turned out that the increase of catching false negatives did not match the decrease of getting false positives.

The following shows some of the sentences from the final classification of the test set which have terms not classified correctly. It is worth noticing that the classifier manage to recognize most of the terms, but have some trouble determining the borders of the terms. The second output shows a typical mistake where the classifier tags the beginning of the term 'Factor VIII' but fails to get the rest 'related antigen' (notice the misspelled word 'antigen'). The third output shows a more crucial mistake, the classifier has tagged 'Raf-independent', words ending on -dent, -able, -ible, ... are adjectives and will never be legal stand-alone gene terms, so there is definitely some improvement to be done here. The last output shows a case where 'mutant' has been tagged as gene term, this is against the annotation guidelines for not tagging generic terms such as 'mutant', 'dna', 'peptide'..ect. To fix some of these problems some lexical rules or some post-processing could be a solution. Mistakes are also seen in tagging of parentheses and other signs.

```
index:  @@7969135416
genes:  '|6 6|Cln2', '|7 8|PEST domain'
gold:   '|6 8|Cln2 PEST domain',
alt:    '|6 6|Cln2', '|6 7|Cln2 PEST'

index:  @@94233104179
genes:  '|3 3|cystic', '|17 17|CD34', '|19 20|Factor VIII'
gold:   '|17 17|CD34', '|19 22|Factor VIII related antigen',

index:  @@110058084808
genes:  '|5 5|Raf-independent', '|6 7|ERK MAPK'
gold:   '|6 7|ERK MAPK',
alt:    '|6 6|ERK'

index:  @@18424982266
genes:  '|16 16|phytohemagglutinin', '|18 18|PHA'
gold:   '|16 18|phytohemagglutinin ( PHA',

index:  @@94306612817
genes:  '|3 3|mutant', '|4 4|EphB1', '|10 10|Nck',
        '|13 13|EphB1', '|17 17|JNK'
gold:   '|3 5|mutant EphB1 receptors, '|7 7|Y594F', '|10 10|Nck',
        '|13 13|EphB1', '|17 17|JNK'
alt:    '|3 4|mutant EphB1', '|4 4|EphB1','|4 5|EphB1 receptors' ,
        '|4 8|EphB1 receptors ( Y594F )','|3 8|mutant EphB1 receptors ( Y594F )',
        '|13 13|EphB1'
```

Improving of features and applying of some more robust postprocessing rules, that could ensure that adjectives tagged alone would be filtered out. Carefulness

is important, while changing a gene tagged adjective to a non-gene tag, would maybe not solve the problem. Maybe the adjective does belong to some bigger context of a gene entity, and the change would not improve anything.
Misspellings and wrong annotations in the data set does exist, e.g. the missing parenthesis in output number three in the tags from the golden standard: '|16 18|phytohemagglutinin ( PHA'. A closing parenthesis does definitely miss here. The standard of the data can be discussed. This is the first version of the data set, and the GENETAG data has been updated and modified multiple times since the publications of the first version, but these up-to-date version are not freely accessible.

Chapter 6

# Conclusion

The learning algorithm for a Naive Bayes classifier has been explained in details through examples. The understanding of how the training and classification steps works for the learning algorithm is an important part of applying and improving the classifier. It has been shown that a NB classifier can be used for NER of gene terms. The performance does not equal the more advanced statistical methods, but the performance does show some significant skills of extracting gene terms. A well chosen feature set combined with some lexical rules for filtering out false negatives and false positives are important for receiving a good performance.
A final f-measure of 0.58 ranks in the lower end of the submitted NER solutions from the participating teams of the BioCreative I challenge task 1.A. A more comprehensive investigation of features and their influence, and maybe considerations about using a more advanced statistical learning algorithm would help improving the performance. Furthermore many of the false positives are actually true gene tags having a wrong parenthesis, missing a terminating word or a generic term. Application of some robust correcting rules after classification could probably be a solution to helping out in this gene border problem.

Appendix A

# Tags

| entry | definition | examples |
|---|---|---|
| NN | noun, common, singular or mass | cabbage thermostat investment |
| IN | preposition or conjunction, subordinating | astride among into if beside |
| JJ | adjective or numeral, ordinal | third oiled battery-powered |
| NEWGENE | | |
| DT | determiner | all them these this those |
| NNS | noun, common, plural | undergraduates scotches muses jobs |
| CD | numeral, cardinal | mid-1890 nine-thirty 271,124 dozen |
| . | sentence terminator | . ! ? |
| , | comma | , |
| CC | conjunction, coordinating | & ’n and both but either vs. |
| VBN | verb, past participle | multihulled experimented desired |
| RB | adverb | occasionally swiftly pitilessly |
| VBD | verb, past tense | dipped pleaded soaked |
| NNP | noun, proper, singular | Escobar Kreisler CTCA Liverpool |
| SYM | symbol | % & ’ ” ”. ) ). * + ,. < = > @ |
| VBZ | verb, present tense, 3rd person singular | bases pictures emerges seduces |
| ( | opening parenthesis | ( [ |
| VBP | verb, present tense, not 3rd person singular | predominate wrap |
| VB | verb, base form | ask bless boil bomb |
| PRP | pronoun, personal | hers himself hisself me myself |
| VBG | verb, present participle or gerund | stirring angering judging stalling |
| : | colon or ellipsis | : ; |
| WDT | WH-determiner | that what whatever |
| MD | modal auxiliary | can cannot could dare may |
| JJR | adjective, comparative | busier calmer clearer closer |
| FW | foreign word | gemeinschaft K’ang-si oui |
| NEWGENE1 | | |
| PDT | pre-determiner | all both half many quite such |
| WP | WH-pronoun | that what whosoever |
| TO | "to" as preposition or infinitive marker | to |
| JJS | adjective, superlative | calmest cheapest darkest deadliest |
| NNPS | noun, proper, plural | Americans Andalusians Andes |
| EX | existential there | there |
| WRB | Wh-adverb | how however whence whenever |
| CELL | | |
| RBR | adverb, comparative | further higher however larger |
| CHEM | | |
| ORG | | |
| RP | particle | around at back behind for from |
| RBS | adverb, superlative | best hardest most nearest |
| ) | closing parenthesis | ) ] |
| POS | genitive marker | ’ ’s |

**Table A.1:** A simple table

Appendix B

# Annotation guidelines for GENETAG corpus

The following are some rules about which words are considered as part of a single gene/protein entity.

1. Mutants

   *p53 mutant*

2. Parentheses at start or end when embedded in the name

   *(IGG) receptor*

3. Motifs, elements, and domains **with a gene name**

   *POU domain*
   *src domain*
   *RXR-responsive element*
   *Ras-responsive enhancer*
   **but not**
   serum response element
   AC element
   B-cell-specific enhancer element

dioxin responsive transcriptional enhancer

4. Plurals and families

   *immunoglobulins*

5. Fusion proteins

   *p53 mdm2 fusion*

6. The words light/heavy chain, monomer, codon, region, exon, orf, cdna, reporter gene, antibody, complex, gene, product, mrna, oligomer, chemokine, subunit, peptide, message, transactivator, homolog, binding site, enhancer, element, allele, isoform, intron, promoter, operon, etc. **with a gene name.**

7. Particular qualifiers such as alpha, beta, I, II, etc. **with a gene name.**

   For example, *topo* is not an allowable alternative to *topo II.*

8. If the context suggests that a word is necessary, require that word in the allowable alternatives, even if it is still a name without the word.

   *rabies immunoglobulin* (RIG) ( *immunoglobulin)*
   designated *HUG1*, for *Hox11 Upstream Gene* (not *Hox11*)

9. Viral promoters, LTRs and enhancers **with specific virus name**.

   *HIV long terminal repeat*
   *Simian virus 40 promoter*
   *HPV 18 enhancer*

10. Antigen receptor region gene segment genes

    *C genes*
    *Tamarin variable region genes*

11. Peptide hormones

    *Vasopressin, prolactin, FSH*

12. Embedded names – tag **only the gene/protein part of the name**

    *p53-mediated*

The following generally **do not** qualify as gene/protein entities:

1. Generic terms

   *zinc finger* alone (but *zinc finger protein* is an accepted gene/protein entity)

2. Mutations

   p53 mutations

3. Parentheses at start and end which 'wraps' the whole gene/protein name

   (*IGG*)

4. TATA boxes

5. Receptors: if a receptor is specified, the gene name without "receptor" is not considered to be a valid alternative.

6. Synonyms: if a synonym is given in the sentence which implies certain words are necessary to the gene name, they will be required in the alternatives

   For *rabies immunoglobin (RIG)*, "immunoglobin" alone will not be a valid alternative because RIG implies that "rabies" is part of the name in this context.

7. Non-peptide hormones

8. DNA and protein sequences

# Bibliography

[1] The BioCreative Organizing Committees. Critical assessment of information extraction in biology. http://www.biocreative.org/about/background/description/, August 2012.

[2] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly, Beijing, 2009.

[3] Tomoko Ohta, Jin-Dong Kim, Sampo Pyysalo, Yue Wang, and Jun'ichi Tsujii. Incorporating genetag-style annotation to genia corpus. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing*, BioNLP '09, pages 106–107, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[4] L. Tanabe, N. Xie, L. H. Thom, W. Matten, and W. J. Wilbur. Genetag: a tagged corpus for gene/protein named entity recognition. *BMC bioinformatics*, 6 Suppl 1, 2005.

[5] Jin-Dong Kim, Tomoko Ohta, Yuka Tateisi, and Jun ichi Tsujii. Genia corpus - a semantically annotated corpus for bio-textmining. In *ISMB (Supplement of Bioinformatics)*, pages 180–182, 2003.

[6] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*, chapter 7. O'Reilly, Beijing, 2009.

[7] L. Tanabe, L.H. Thom, W. Matten, D.C. Comeau, and W.J. Wilbur. Semcat: Semantically categorized entities for genomics. *AMIA Annu Symp Proc*, 2006.

[8] L. Tanabe and W. J. Wilbur. Generating of a large gene/protein lexicon by morphological pattern analysis. *J Bioinform Comput Biol.*, Jan;1(4):611–626, 2004.

[9] Alexander Yeh, Alexander Morgan, Marc Colosimo, and Lynette Hirschman. BioCreAtIvE Task 1A: gene mention finding evaluation. *BMC Bioinformatics*, 6(Suppl 1):S2+, 2005.