Preface

This master's thesis has been produced in the period of January 20th through June 30th 2005 at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. The title of the thesis is *"Identification of biomedical entities from Medline abstracts using a dictionary-based approach"*.

I would like to thank my advisor Heri Ramampiaro.

Trondheim, June 30th, 2005

Magnus Skuland

Abstract

The aim of this paper was to develop a system for identification of biomedical entities, such as protein and gene names, from a corpora of Medline abstracts. Another aim was to manage to extract the most relevant terms from the set of identified biomedical terms and make them readily presentable for an end-user.

The developed prototype, named iMasterThesis, uses a dictionary-based approach to the problem. A dictionary, consisting of 21K gene names and 425K protein names, was constructed in an automatic fashion. With the realization of the protein name dictionary as a multi-level tree structure of hash tables, the approach tries to facilitate a more flexible and relaxed matching scheme than previous approaches.

The system was evaluated against a golden standard consisting of 101 expert-annotated Medline abstracts. It is capable of identifying protein and gene names from these abstracts with a 10% recall and 14% precision. It seems clear that for further improvements of the obtained results, the quality of the dictionary needs to be increased, possibly through manual inspection by domain experts. A graphical user interface, presenting an end-user with the most relevant terms identified, has been developed as well.

Contents

List of Figures

List of Tables

1 Introduction

The amount of information in biological databases is growing exponentially, reflecting the research effort constantly being performed by teams of biologists and medical researchers world-wide. Medline, the largest biological database available, currently contains close to 15 million articles, growing at a rate of 4% per year. Given these huge amounts of data, researchers require assistance in extracting information and relationships between biomedical entities in this domain. Thus, over the last decade a large research effort has been put into development of various text mining tools that would improve this situation.

Text mining of standard texts, such as news texts, routinely may reach levels of 90-95% on information extraction, but biological text mining is a completely different business. Success rates in this field seldom surpasses 75%. There are many reasons for this phenomenon. For one, biological terms tend to have multiple syntactical variants and synonyms. Scientists in the field do not always stick to established terminologies. Another problem is the lack of data that have been "marked up", i.e. data indicating the roles played by and the relationship between the entities in the text. A third problem is inconsistencies between databases and the literature. The massive growth of data coupled with inefficencies in the process of transferring data into other data resources have lead to incomplete databases.

Research in biological text mining have concentrated on three directions: Rule-based approaches, machine-learning approaches and dictionary-based approaches. Additionally, some approaches have been hybrid approaches of the former three. Rule-based systems rely on expert-derived rules, which consist of specific syntactic and semantic properties, to identify biomedical entities in text. Machine-learning systems on the other hand, require the presence of an expert-annotated training corpus. This corpus is then used to automatically derive identification rules through the use of specific statistical algorithms. Dictionary-based systems require the presence of a rich list of protein and gene names to identify occurrences of biomedical entities in text. The identification is often being done by means of various substring matching techniques.

1.1 Problem definition

This paper is focused on the problem of identifying biomedical entitites, i.e. gene and protein names, in Medline abstracts. The authors have chosen to attack the problem using a dictionary-based approach. The problem also consists of extracting, from the identified biomedical entities, the most relevant terms to be presented for an end user. The final system should be graphically represented by a prototype interface.

1.2 Thesis outline

Below is a presentation of the contents of this paper.

Section 2 - Theoretical foundation: This section contains knowledge that is important for the understanding of this approach. Fields such as molecular biology, information retrieval and text mining are subjects of focus.

Section 3 - Previous work: This section addresses some of the interesting previous research efforts in the field.

Section 4 - Own approach: This section presents the reader with the core idea of this approach.

Section 5 - Implementation: This section provides information on vital implementational parts. It describes the implemented prototype application and gives insight into the development process.

Section 6 - Results: The results section describes the evaluation of the implemented system and contains information about the system performance.

Section 7 - Discussion: The Discussion section contains an assessment of the approach and its performance. It also contains a final conclusion on the system and outlines directions for further study.

2 Theoretical foundation

This section will provide the reader with an introduction to basic concepts of molecular biology as well as to fields such as bioinformatics, information retrieval and textmining.

2.1 Molecular biology

The study of molecular biology is essentially the study of life. All organisms on the planet are composed of cells, the basic unit of life. Some organisms are single-cell organisms like bacteria, others consist of multiple cells.

Biologists are currently working on problems that affect us in our daily lives every day. They work on finding cures to diseases as cancer and AIDS, and they try to find solutions to the world's rapidly increasing population. Their accumulation of knowledge is actually life-saving.

In the subsequent sections 2.1.1, 2.1.2, 2.1.3 and 2.1.4, the basic building stones of molecular biology are presented.

2.1.1 Proteins

There are four types of macromolecules that all organisms consist of. These are proteins, nucleic acids, lipids and carbohydrates. These macromolecules are assembled into organelles within the cells.

One can typically divide the functions of proteins into seven categories: Enzyme catalysis, defense, transport, support, motion, regulation and storage.

Enzymes are globular proteins that function as catalysts. They facilitate the breaking up of larger molecules into smaller subunits by stressing certain chemical bonds. Kinases are examples of enzymes. Some proteins have a defense purpose, e.g. spider venom consists of several neural toxins. Transport proteins transport specific ions and small molecules. An example is hemoglobin, that carries oxygen in the blood stream. Support proteins are typically fibers, like for example keratin which forms hair and nails in the human body. Actin and myosin are two motion proteins that facilitate muscle contraction. Regulatory proteins have a key role in controlling many of the body's functions. Hormones are one type of regulatory proteins. An example is insulin which controls blood glucose levels. Some proteins have a storage

function, in that for example iron can be stored in the cell by binding as ion to specific storage proteins.

2.1.2 Amino acids

Amino acids are the building blocks of proteins. All proteins are simply polymers of 20 different kinds of amino acids, in a specific order.

An amino acid group chemically contains an amino group (-NH_2), a carboxyl group (-COOH) and a hydrogen atom (H). All of these are bonded to a central carbon atom. See figure 1 for an illustration. The R group indicates the side group that gives each amino acid unique chemical properties.

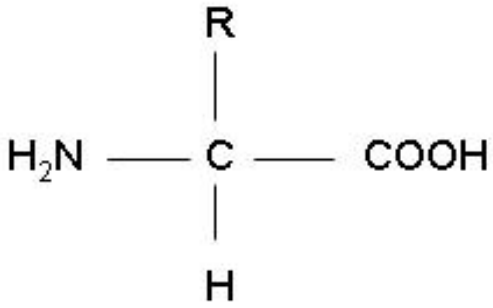

Figure 1: Amino acid structure

A protein consists of one or more long chains, polypeptides, of amino acids linked by peptide bonds. It was not until the 1950s that scientists, through the work of Sanger, became aware that every kind of protein had a specific amino acid sequence[16].

2.1.3 DNA

DNA[1] was discovered in 1869 by the German chemist Friedrich Miescher. The material he discovered seemed to be located in the nucleus of the cells and also was slightly acidic, hence the name "nucleic acid".

The three main components in a DNA molecule were determined by P.A. Levene (1920s). They are:

- Five-carbon sugar

[1] Abbreviation for deoxyribonucleic acid.

- Phosphate (PO_4) group
- Nitrogen-containing (nitrogenous) base

The base may be either a purine (adenine, A or guanine, G) or a pyrimidine (thymine, T or cytosine, C). In humans there is roughly a 30% proportion of both A and T, and a 20% proportion of both G and C. These numbers correspond to Chargaff's rules, which are given in definition 2.1 below:

Definition 2.1 *The proportion of A always equals that of T, and the proportion of G always equals that of C.*

The three main components form what is called a nucleotide, and a single strand of DNA consists of a series of nucleotides joined together in a long strand. Figure 2 shows the double helix shape of a DNA molecule.

Figure 2: The double helix shape of DNA

DNA is just one of the two types of nucleic acids. RNA[2] is the other. Together these two manage the incredible feat of storing and transferring genetic information. RNA, which is similar to DNA in structure, is used to read a cell's DNA-encoded information and is made as a transcripted copy of portions of the DNA. This transcript ventures out into the rest of the cell, serving as a blueprint specifying a protein's amino acid sequence. The information stored in DNA can also be passed down to an organism's descendants, thus the labeling of DNA as "hereditary material".

2.1.4 Genes

Genes are the sequence of nucleotides that determines the amino acid sequence of a protein. Most genes encode proteins or subunits of proteins, but others produce special forms of RNA.

[2]Abbreviation for ribonucleic acid.

The key of the gene expression was found in 1961 by Francis Crick. He discovered that the genetic code consists of blocks called codons consisting of three consecutive nucleotides. Each codon codes for one specific amino acid. For example GUU and GUC codes for valine, and AAA codes for lysine.

Figure 3: Rice grains (*Oryza sativa*). Rice actually has a genome consisting of close to 50 000 genes, considerably larger than the human genome.

In 2000, The Human Genome Project reported that they had successfully mapped and sequenced the entire 3.2 billion nucleotide human genome. Interestingly, the human genome consisting of only 30 000 genes plus, is only slightly larger than the genome of mice, and much smaller than that of rice (figure 3).

2.2 Bioinformatics

Bioinformatics can be defined as in definition 2.2 below (Tekaia, [15]):

Definition 2.2 *The mathematical, statistical and computing methods that aim to solve biological problems using DNA and amino acid sequences and related information.*

Thus, one can interpret this in a broader sense to imply that bioinformatics is a field that describes any use of computers to process biological information.

There are multiple fields closely related to bioinformatics. Among these are computational biology(field closer to evolutionary biology than cell and molecular biomedicine), medical informatics(discipline that is concerned with structures and algorithms for the manipulation of medical data), genomics(the science of exploring a species complete genome) and proteomics(studies of gene expression at the level of the functional proteins).

2.3 Information Retrieval

Information retrieval deals with the representation, storage, organization of, and access to information items[1]. It is important to differentiate between the two terms "information retrieval" and "data retrieval".

Given a collection of documents, a data retrieval merely determines which documents contain the keywords in a user query. This might be unsatisfactory to the user. A relational database system is an example of a data retrieval system.

Information retrieval on the other hand aims at retrieving information about a subject from mostly natural language text, whereas data retrieval systems works on data with well defined structure. The IR system has to extract syntactic and semantic information from the text and rank the documents according to relevance compared to the user query. The goal of such a system is to return the relevant documents.

There exists three classic IR models; the Boolean, the vector and the probabilistic models, and these will be briefly presented here.

2.3.1 The Boolean model

This model is a simple one based on set theory and Boolean algebra. The queries are specified as Boolean expressions. In this model the index terms are considered to be either present or absent in a document. Therefore the index term weights are assumed to be binary (0 or 1), thus leading to a prediction that a document is either relevant or non-relevant. This exact matching may lead to retrieval of too many or too few documents.

2.3.2 The vector model

The vector model differs from the Boolean model in that index terms can be given non-binary weights. This makes partial matching possible. Both the index terms in queries and in the documents are weighted. These can be represented by a query vector and a document vector. In this model it is possible to calculate the degree of similarity between documents, with regard to the query, as the correlation between the two vectors. This can be quantified by the cosine of the angle between them. Thus the vector model proposes a ranking of the document according to the degree of similarity to the query. The value of similarity is a number between 0 and +1.

In the vector model, the index term weights are often quantified using the *tf-idf-term-weighting scheme*. This is a model based on term frequency and inverse document frequency. The tf-idf scheme balances the effects of both intra-cluster similarity and inter-cluster dissimilarity. The normalized frequency $f_{i,j}$ of a term k_i in document d_j is given as in equation 1:

$$f_{i,j} = \frac{freq_{i,j}}{max_l \ freq_{l,j}} \tag{1}$$

where $freq_{i,j}$ is the raw frequency of term k_i in d_j. The maximum is computed over all mentioned terms in d_j. The inverse document frequency is given as in equation 2:

$$idf_i = \log \frac{N}{n_i} \tag{2}$$

where N is the total number of documents in the system and n_i is the number of documents in which the index term k_i appears. Given the normalized and inverse document frequencies, it is possible to define the term weight in this scheme as in equation 3:

$$w_{i,j} = f_{i,j} \cdot \log \frac{N}{n_i} \tag{3}$$

2.3.3 The probabilistic model

The probabilistic model, which was proposed already in 1976, takes as an assumption that there exists an ideal answer set R which contains exactly the relevant documents in a collection. All index term weights are binary, and the set R has to be initially guessed. The model assigns to each document a ratio which is the probability that the document is relevant to the query divided by the probability that it is non-relevant. This ratio is then used as a measure of the document's similarity to the query. Documents are ranked in decreasing order of their probability of being relevant.

2.4 Evaluation

To evaluate the performance of a certain information retrieval strategy, it is common to compare the documents retrieved by the strategy with the documents that have been found to be relevant by experts, with regards to similarity. The two most commonly used evaluation measures are presented in 2.4.1. Section 2.4.2 presents TREC, a well known test collection used in evaluation of information retrieval systems.

2.4.1 Precision and recall

When evaluating the performance of an information retrieval system, the two most frequently used measures are recall and precision. Recall is defined as the fraction of the relevant documents which has been retrieved. Precision is the fraction of the retrieved documents which is relevant [1].

Given a set R of relevant documents and a document answer set A, then the intersection of these two sets is identical to the relevant documents in the answer set. Recall and precision can then be described as in equations (4) and (5):

$$Recall = \frac{|Ra|}{|R|} \tag{4}$$

$$Precision = \frac{|Ra|}{|A|} \tag{5}$$

$|Ra|$ is the number of relevant documents in the answer set. Figure 4 illustrates this. There are however some disadvantages of using these two

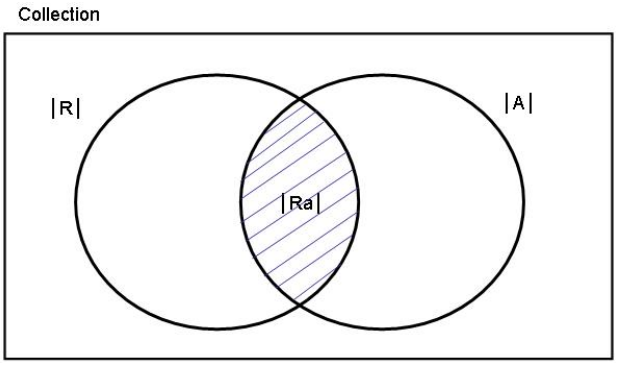

Figure 4: Precision and recall

measures. The estimation of maximum recall for a query requires detailed knowledge of all the documents in a collection. This implies that for large collections, such as biomedical corpora, it is impossible to estimate recall precisely. In some occations it might be of greater interest to use a single value that combines precision and recall. [2] introduces a measure called Precision-Recall break even point (PRBE), which is the point within ranking where precision equals recall.

2.4.2 TREC

TREC (Text REtrieval Conference) is a yearly conference started in the early 1990s in Maryland as a response to the lack of robust testbeds and benchmarks in IR. The first conference was held in 1992. Ninety-three groups from 22 countries were attending the TREC 2003. The last TREC was held November 2004[3]. Every participant at this conference has to work on the same collection, the TREC collection.

The TREC collection consists of documents, example information requests and a set of relevant documents for each example information request. The size of this collection has been growing considerably. As of 2004 the size was 426 GB[4].

The test collections and evaluation software are available online, thus facilitating testing of own retrieval systems at any time. TREC is therefore an user-friendly tool that might help developers test their own contributions in a standardized fashion.

2.5 Text Mining

Text mining is a field covering various techniques that might be applied to textual data in order to retrieve information from text. Thus it is closely related to IR. Text mining is a subterm of the more general term data mining. Data mining can be defined as in definition 2.3 ([5]):

Definition 2.3 *Data mining is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data. This encompasses a number of different technical approaches, such as clustering, data summarization, learning classification rules, finding dependency net works, analysing changes, and detecting anomalies.*

The definition of text mining is identical except the data is of textual type. The expression “previously unknown” means, in this context, either that it is information that not even the writer knows about, or it means that one should rediscover information that the author did encode in the text. The process of text mining, i.e. information extraction from textual data, is a process of several steps. As can be seen from figure 5, the text is initially preprocessed. This preprocessing consists of syntactic and semantic text analysis. Part-of-speech (POS) tagging is used here. This method labels each word with the corresponding part of speech (noun, verb or adjective). The text is also normally parsed, and a parse tree is generated for each sentence.

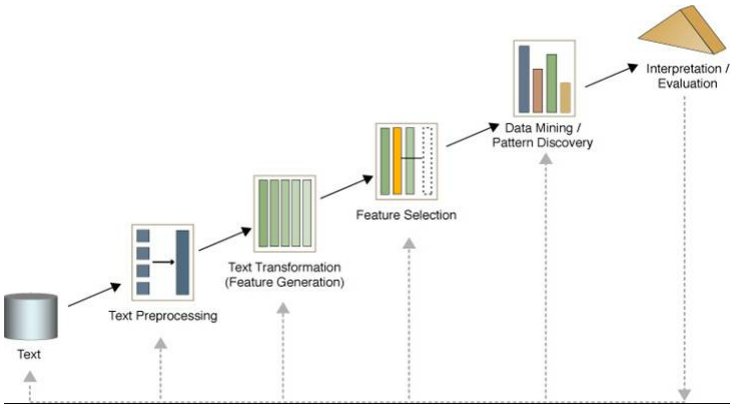

Figure 5: The text mining process

The second step is text transformation, in which the documents are divided into the single words. These words are then subject to stemming (the words are identified by their roots). In this step there is also removal of stop words.

In the feature selection step the dimensionality is reduced. Irrelevant features are also identified, so as to limit the noise.

In the text mining step the text may be subject to a supervised learning process(classification) or an unsupervised learning process (clustering). In a classification process, the data (observations, measurements) are accompanied by labels indicating the class of observations. These data are then split into training data and test data for model building purposes. Examples of classification techniques are:

- Bayesian classification
- Decision trees
- Neural networks
- Instance-based methods

In the case of a clustering process, class labels of training data are unknown. The system is then given a set of data with the aim of identifying classes or clusters in the data. The idea is that documents in one cluster are more similar to one another. Repetition of the process will result in more correct sets of documents.

2.6 Biological text mining

Research in the field of text mining these days is especially directed towards biological texts and databases. These databases offer information about genes, gene products, protein structure, metabolic pathways, diseases, organisms, DNA-sequences and so on.

One of the main reasons this field is of great interest is the enormous growth of size in the literature. As can be seen from figure 6, the number of articles in Medline is presently close to 15 million, and this is growing approximately 4% per year[6]. Medline will be further presented in section 2.6.1.

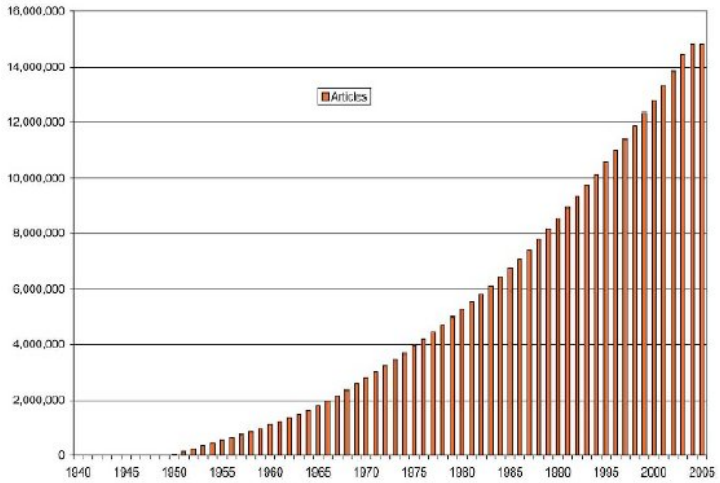

Figure 6: Growth of articles in Medline[6].

Text mining is a tough business. To this date it has yet not been developed tools that can analyse more than 30% of English sentences correctly and transform them into a structured formal representation[6]. The problem is that one always runs into Zipf's law, i.e. that frequent terms account for a large portion of the text, but a large fraction of the terms appear at low frequency and often only once. Examples of these frequent terms are the, of, and, in, to and also biological terms like $cells, protein$ etc.

The absolute standard for any biological text mining tool is the data returned by a curator. A curator is a trained domain-expert who is able to sort out any ambiguities as well as handle the high variability of language described by Zipf's law. No computer-based system can match this at present.

A major problem in identification of biological entities, is the multiple variants of terms. All variants of a term should be considered, these can be

synonyms and syntactical variants. There is also a problem with ambiguity, in that for example some gene symbols might correspond to disease names or experimental methods. In these situations the context has to be taken into consideration[3][7].

Some of the strategies that have been developed with the aim of identifying biological entities are ad hoc rule-based approaches, machine-learning techniques and dictionary-based approaches as well as hybrid approaches that combine different approaches. The first three will be further presented in sections 2.6.2, 2.6.3 and 2.6.4 respectively.

2.6.1 Medline

The huge MEDLINE database today consists of roughly 15 million articles. It is the NLM's[4] premier bibliographic database covering the fields of medicine, nursing, dentistry, veterinary medicine, the health care system and the preclinical sciences[10]. Through its Entrez query system it provides the standard "front end" for biomedical literature search[8][5]. It contains bibliographic citations and author abstracts from more than 4800 biomedical journals published in 71 countries. MEDLINE began as MEDLARS in 1964 with batch searching. The first online service was presented in 1971. In the early 90s Medline was made available on the Web. The system has become increasingly popular, now with more than 250M hits per year.

PubMedCentral, a product of NLM, was set up to be a barrier-free, biomedical repository that accepts full text and supporting data[9]. Each item has a unique and persistent PubMedID. Links are provided in this system to sequence information, in addition to full text documents. Entrez is the software that integrates across the various systems. It is a text-based search and retrieval system that is being used at NCBI[6] for services including PubMed, Nucleotide and Protein Sequences, Protein Structures, Complete Genomes, Taxonomy and others. Figure 7 below gives a high-level view of the Entrez databases. Authors and publishers are able to submit material to Medline through PubMedCentral, given that the material meets certain criteria.

A typical sample record in PubMed Medline contains data tags informing of the PubMed Unique Identifier, date, title, journal, abstract, authors, publi-

[3]Example: 'EGFR' may have the meaning 'epidermal growth factor receptor' as well as 'estimated glomerular filtration rate'.

[4]National Library of Medicine

[5]Entrez PubMed: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi

[6]National Center for Biotechnology Information

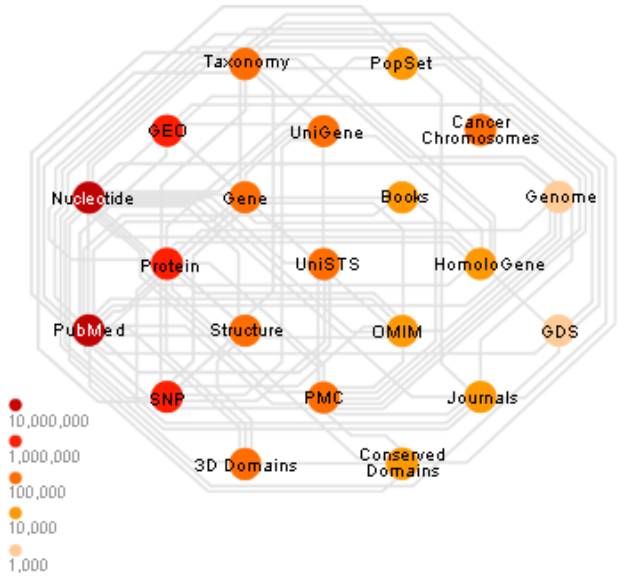

Figure 7: Entrez databases

cation type, MeSH terms and source among others. MeSH stands for Medical Subject Headings and is a controlled vocabulary thesaurus. The MeSH terms are descriptors that help arrange the articles in a hierarchical fashion. There are 22568 descriptors in MeSH[11].

2.6.2 Rule-based approach

The automatical tagging of protein names in text is an important first step towards automatic knowledge extraction from biomedical texts. One approach which has been proposed is systems based on hand-written rules. These rules are expert-derived and usually combine surface clues with word syntactic and semantic properties. Such systems require lots of human analysis, but are easy supportable, adjustable and expandable systems. One example of this approach is the Yapex system proposed by Franzén[12].

The Yapex approach consists of several lexical analysis steps. One is finding feature terms, words that describe the function or characteristics of a protein. Such words can indicate the presence of a protein name. Another step is finding core terms, i.e. words that end in -ase or -in for example. To eliminate false hits various filters are used. Examples of such can be pattern matching filters that filter out chemical formulas, arithmetic expressions and amino acid sequences.

Franzén reported results of precision and recall of roughly 83% under sloppy notion of correctness. This notion implies that any token of the hit matches some token of the answer key. Under strict notion the results were closer to 67%[12].

2.6.3 Machine-learning approach

Machine-learning algorithms have also been a proposed solution to the disambiguation of biomedical entities in text. Such an approach aims to fully automate the process of knowledge extraction. They rely on the presence of an expert-annotated training corpus to automatically derive the identification rules by means of various statistical algorithms[14]. GeneWays is an interesting research effort in this direction[13].

This system is an example of unsupervised learning, in that only raw text and no human input or annotation is available to the system. It uses an automated script to download articles that appear in HTML, then it converts these to the XML-format. Term identification is done by lookup over the GenBank database. It only considers terms that are multi-word entries in GenBank, or if single-words, do not appear in the more than 80 000-entry lexicon of common English words used by Brill's statistical part-of-speech tagger. For machine learning the system utilizes basic features of the terms. Words that appear near a term are considered basic features to which contextual information can be mapped.

There are several ways of adding positional information to the features. A word bag can be used for the words before a term, another word bag for the words after. Each word can also be annotated with its distance from the term.

Hatzivassiloglou[13] considers three learning techniques for construction of a prediction model over the word features. These are naive Bayesian learning, decision trees and inductive rule learning.

The naive Bayesian learning-method is a method that tries to assign to a term occurence the class c that maximizes the probability of c given the probability of ϵ ($P(c|\epsilon)$). ϵ is the evidence available to the machine learning algorithm for that occurence.

Decision trees (figure 8) learning is about recursive partitioning of the feature space into areas corresponding to each class label. The C4.5 implementation of decision tree learning (Quinlan, 1993) is well known. The process ends

with a leaf node corresponding to a class label.

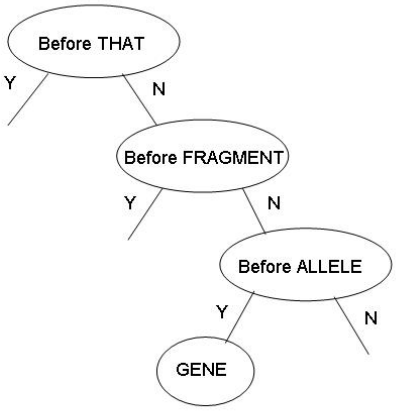

Figure 8: Example decision tree

Inductive rule learning is a method where rules involving tests on features are iteratively constructed. These rules map a specific combination of features to a class label, and are then applied in a sequential fashion during prediction, so that rules where the system has the highest confidence are applied first.

Hatzivassiloglou reported of results ranging from 74-76% accuracy for the three different machine learning techniques on two-way classification[7] over a 9 million words-corpus[13]. The naive Bayes is significantly faster in training and prediction than the two others though. Regarding pure protein-identification the results for precision and recall were 87 and 82% respectively.

2.6.4 Dictionary-based approach

This type of approach utilizes a provided list of protein terms to identify protein occurences in a text. This is normally done by means of various substring matching techniques. According to [14], the dictionary-based approaches outperform rule-based approaches on accuracy. Their performance is closely related to the quality of the dictionaries, and the creation and maintenance of these may be a non-trivial task. But the other two approaches require huge manual efforts as well.

[7] Labeling of genes or proteins only.

Egorov[14] presents an interesting effort named ProtScan. Their system uses a combination of curated and non-curated ("raw") protein name dictionaries. After applying tokenization and filtering algorithms to the target text (Medline abstracts), they initiate protein identification by loading both the raw and the curated dictionaries into the same hash table. The entry from the curated dictionary always takes precedence. The abstracts are processed one sentence at a time, and qualifying token sequences are searched for the presence of dictionary entries by trying all subsequences from long to short and from left to right. A subsequence is tested by calculating its hash value and then doing a hash table lookup.

The results of ProtScan were very promising. From a corpus of 1000 randomly selected and manually labeled Medline abstracts, it managed a 88% recall and 98% precision[14].

3 Previous work

Conventional textmining, i.e. information retrieval and extraction performed on standard texts (like news texts), is a research field that has already come of age. Extraction of person and place names from such texts can routinely reach success rates of 90-95%. But when it comes to textmining of biological texts, such as abstracts from the biomedical database Medline, socalled biological textmining, the success rates are much lower, often around 70%. Biological textmining is currently a field that is subject to huge amounts of research. It is a challenging field, where even something as trivial as a slash-sign may imply two different entities or a single compound.

The three main approaches for identification of biomedical entities in text are ad hoc rule-based approaches, machine-learning approaches and dictionary-based approaches as well as hybrid approaches of the former three.

The rule-based approach requires expert-derived hand-written rules. These systems require lots of human analysis, but are easy supportable and adjustable. The Yapex approach [12], as described in section 2.6.2, is an interesting research effort in this direction. The Yapex system uses multiple lexical analysis steps in searching for patterns in the text. Under strict notion of correctness, this effort reported values for precision and recall of around 67%.

GeneWays is an example of a machine-learning approach. This system facilitates unsupervised learning, in that no human input or annotation is available. The approach considers the usage of three different learning techniques: Naive Bayesian, decision trees and inductive rule learning. The effort reportedly has precision and recall values of 87 and 82% respectively[13]. GeneWays is further described in 2.6.3. Another research effort in this area is that of Wilbur et al.[23]. They analyzed and compared one lexical approach with two statistical machine-learning approaches. The approach that was most successful was one based on a bayesian classifier approach, with a reportedly 97% classification accuracy.

ProtScan[14] is a proposed dictionary approach with astonishingly good results. Their results are supposedly in the range of 88% recall and 98% precision on 1000 randomly selected Medline abstracts. This approach utilizes a combination of curated and non-curated protein name dictionaries. This approach is further described in section 2.6.4.

Textpresso[24] is a somewhat well known ontology-based approach in the world of biological text mining. Textpresso uses an ontology consisting of

33 different classes of biological concepts and a collection of full text of scientific articles split into individual sentences ready to be marked up. Textpresso is organism-specific, currently focusing on the 3800 full text-corpora of *C.elegans*. Values for precision and recall on abstracts are 52% and 45% respectively.

Nenadic[25] presents a terminology-driven approach with very good results. This approach is a hybrid solution combining lexical, syntactical and external similarity between terms. On a corpus of Medline abstracts they reportedly achieved a 99% precision and 74% recall.

4 Own approach

This chapter will provide the reader with an insight into the basic idea of this approach. Succeedingly, the dictionary, which is a core unit in the approach, will be addressed. The user will also be presented with class and sequence diagrams for increased understanding of the approach.

4.1 Basic idea

This approach is based primarily on the idea of identifying biological terms through lookup in a rich dictionary of biomedical names. Figure 9 shows a main view of the approach described in this paper. The quality of the results of this system is in direct correlation with the quality of the dictionary. Hence it is essential that the quality of the dictionary is as high as possible. Our solution is inspired by the work of Egorov et al.[14] and their proposed system; Protscan.

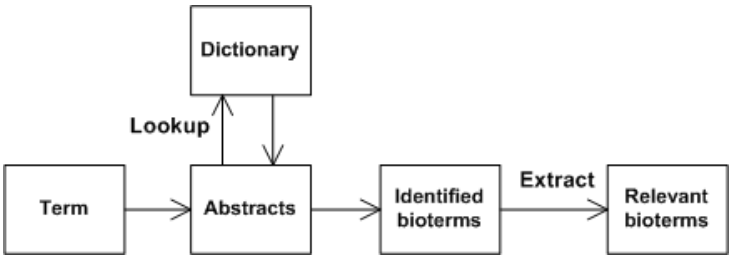

Figure 9: Main view of approach

The results of the Protscan system were almost suspiciously good, but this is a topic that will be addressed in section 7. Protscan used a combination of curated and non-curated ("raw") protein name dictionaries. For construction of the dictionaries they used the LocusLink database and additionally enriched this by incorporating protein names, aliases, gene names etc. from the linked GenBank, GoldenPath and HUGO database entries[8].

[8] *LocusLink* - an integrated querying and cross-referencing system now superceded by Entrez Gene (http://www.ncbi.nih.gov/entrez/query.fcgi?db=gene), *GenBank* - a genetic sequence database, an annotated collection of all publicly available DNA sequences, (accessible from http://www.psc.edu/machines/seq/seq.html), *GoldenPath* - a human genome browser gateway provided by the Genome Bioinformatics Group of UC Santa Cruz (http://genome.ucsc.edu/), *HUGO* - The Human Genome Organisation, established 1989 (http://www.hugo-international.org/)

The identification of biological entities in the ProtScan solution is done by processing one sentence at a time from the input abstracts. The sentences are scanned for the presence of sequences of tokens from a "protein words" set. These sequences are then scanned for the presence of dictionary entries by trying all of the subsequences from long to short and from left to right. All the dictionary entries are hash table entries, so the look up is done by calculating a potential protein name's hash value and then scanning through the hash table entries.

This approach differs from the Protscan-solution in that we try to facilitate a more relaxed matching-scheme. We have also implemented the dictionaries as hash tables, but instead of having the names at only one hash table level, we have implemented a tree structure. The tree structure is maximally 10 levels deep.

This approach iterates through every token in the input abstracts, and for each token it tries to match the hash key at the first level in the dictionary. If it is a match, then the consecutive word is tested for a match on the second level of the dictionary and so on. The approach will always try to return the longest possible match. This is an attempt to increase the flexibility since many of the protein names are very long and a 100% match of these seemed unfeasible.

4.2 The dictionary

The dictionary is the Achille's heel of this approach and of immense importance. There was built one gene name dictionary and one protein name dictionary. The gene name dictionary was developed based on resources from the HUGO Gene Nomenclature Committee[9]. This dictionary turned out to be about 21K names of size.

The protein name dictionary was developed from UniProt[10] (Universal Protein Resource). This is the world's most comprehensive catalog of information on proteins. It is a central repository created by joining the information contained in Swiss-Prot(protein sequence database distributed by European Molecular Biology Laboratory(EMBL)), TrEMBL(contains entries that eventually might be included in Swiss-Prot) and PIR(Protein Information Resource, located at Georgetown University Medical Center). The dictionary eventually became about 425K names of size.

[9] http://www.gene.ucl.ac.uk/nomenclature/
[10] http://www.ebi.uniprot.org/index.shtml

4.3 System description

This section gives an overview of the system design of this approach. Figure 10 gives an overview of implemented classes. There are 15 classes in total, but only 13 are executed at run-time. The ProcessDictionary class has only been used during the initial construction of the protein and gene dictionaries, and the Test class was only used during evaluation of system performance. Short descriptions of each class are given below.

Main:
As the name suggest, this class is the main entry point of the application. It starts the GUI window as well as communicating with the Entrez PubMed web resource through an URLConnection. It uses the Parse class for parsing of the output from PubMed. It also starts up the Tokenize class.

GUI:
This class represents the prototype user interface. It invokes the constructor in the Main-class upon a user click. It also starts the BioTermsWindow-class upon another user click.

Parse:
This class extends the DefaultHandler class which is common for classes implementing SAXParsers. It implements a standard Java SAXParser using the SAXParserFactory class. Parse is used to parse XML InputStreams from PubMed and return the results to the Main class.

Tokenize:
The Tokenize class implements logic for separating the individual tokens from the abstract texts. It does not simply remove all signs because that might affect system performance. But it separates all word tokens, and removes signs like e.g. full stop, comma and parantheses. But removing a sign like slash might be unwise, since it is a crucial part of many bio term names. Tokenize also starts up the PreProcessing class.

PreProcessing:
This class is responsible for preparing the sequences of tokens for lookup. It optionally removes purely numerical tokens as well as performing stemming on the tokens. Stopwords are also removed from the set of tokens. The PreProcessing class starts up the LookUp class.

Stopwords:
This class contains logic for removal of stopwords from text. The list of stopwords is based on the stopwords used in Medline.

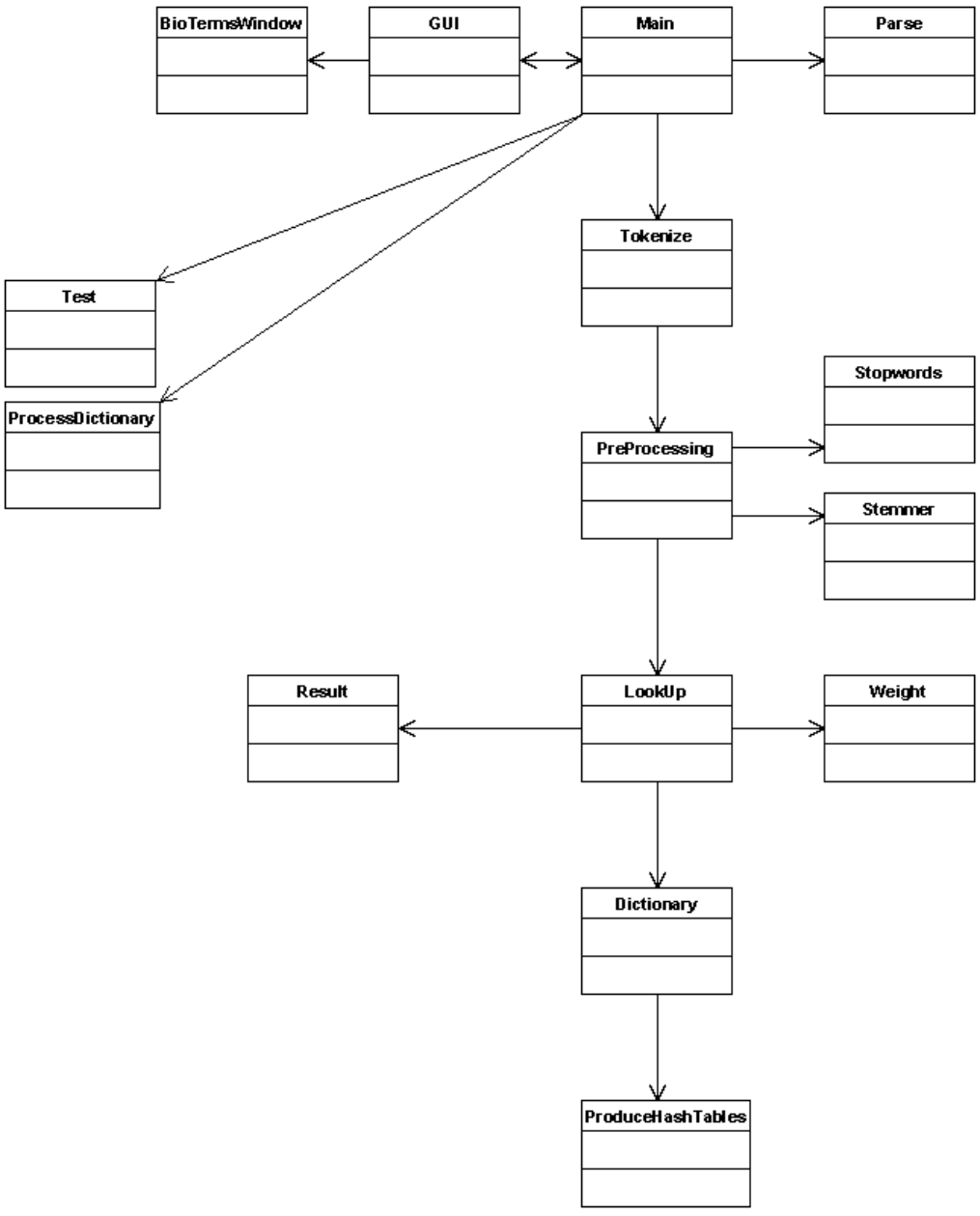

Figure 10: Class diagram - overview

Stemmer:
This class can be instantiated every time it is necessary to perform word stemming. The class is an implementation of the Porter stemming algorithm, freely available on the web.

LookUp:
The LookUp class is responsible for initiating the creation of the dictionaries and for the look-up itself. It also starts up the Weight and Result classes.

Weight:
This class performs calculations on the term frequencies. For each term it calculates the normalized frequency and the inverse document frequency. It then proceeds to calculate weights from these frequencies (see section 2.3.2 for the theory). The results are used to determine the relevance of each term in the LookUp class.

Result:
This is a simple class that contains a result ArrayList of bioterms. The results are sent to the GUI.

Dictionary:
The Dictionary class initates the ProduceHashTables class thus producing all the hash tables. After this initiation Dictionary contains hash tables representing the gene and protein name dictionaries, all ready for lookup processing.

ProduceHashTables:
This class contains logic for creation of both the gene and protein hash tables representing the respective dictionaries.

BioTermsWindow:
If the user wishes to view every single bioterm identified through the lookup process, the BioTermsWindow class will be initiated by a user click in the GUI. The BioTermsWindow class is a simple user interface implemented with the JScrollPane class, thus facilitating scrolling of the list of bioterms.

Figure 11 shows the flow of program control in the system. Program execution starts in Main. All arrows indicate initiation of a class with the object lifelines being the dotted vertical lines.

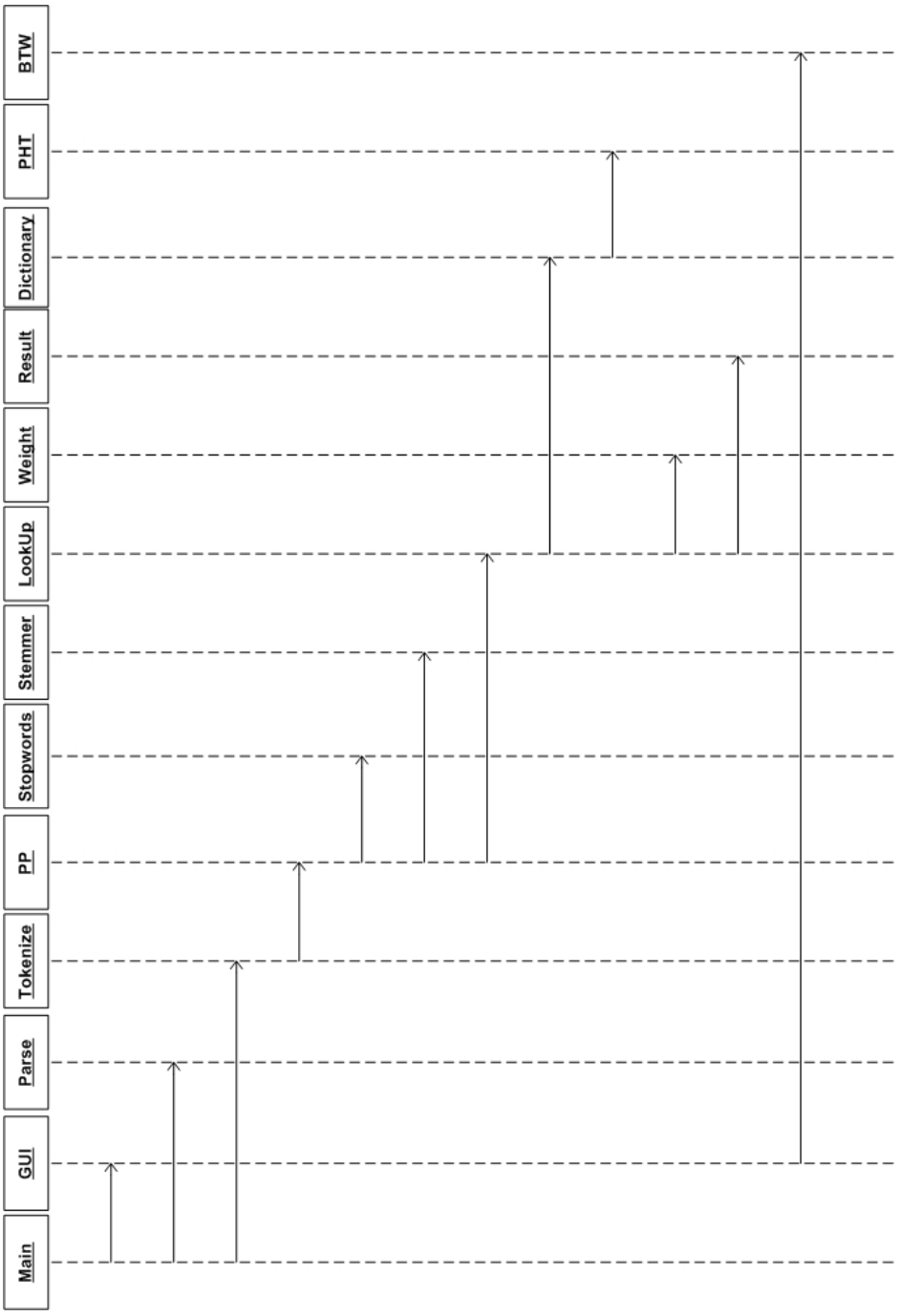

Figure 11: Sequence diagram. After some time Main initiates Parse and Tokenize. Program control then flows to Tokenize which in turn initiates PreProcessing. PreProcessing initiates Stopwords (and optionally Stemmer) before initiating LookUp. LookUp initiates Dictionary which in turn initiates ProduceHashTables. LookUp then starts up Weight and Result. Finally GUI might initiate BioTermsWindow upon user action.

5 Implementation

This section describes the implemented prototype as well as the general implementation process. Central implementational details are explored at an algorithmical level. Subsequently, some techniques that have been applied to the set of potential biomedical terms in the postprocessing stage are treated. Finally, the programming environment and major technologies are addressed. The source files can be found in appendix B.

5.1 The prototype application

The prototype was developed in Java using in excess of 4K of code lines. Figure 12 shows a screenshot. As the user starts the application from the command line or from a Java Web Start link, only the left window will appear at first. The user is then free to enter a search term in the search text field (the amino acid alanine is used in this example). It is also possible to adjust the settings regarding stemming and removal of purely numerical entities. The user setting in regard of stemming is a boolean that will be sent to the ProduceHashTables class. The application will then proceed to load either stemmed or unstemmed versions of the gene and protein name dictionaries. The setting for removal of numbers is implemented as a boolean that is sent to the PreProcessing class, where action is taken according to its value.

When the user hits enter or the button, the search term will be sent as part of a query to PubMed through its ESearch-service. The 10 first hits in PubMed will be returned as an XML file. This file is parsed, and the PubMed IDs are extracted. These IDs are then used as part of an EFetch call to PubMed, returning the corresponding abstracts as another XML file. This file is parsed, and the abstract texts are extracted. The abstracts are then sent on to Tokenize for further processing.

When the processing of the abstracts are finished the results are presented in the application window. The left column shows the potentially three most relevant terms from the abstracts on a general basis. The column to the right displays the five protein or gene names that the system has identified as the five most relevant. Below these columns, the total number of gene and protein names identified is shown. If the user then wants to look at all identified bio terms, a click at the bottom button will open the right window effectively displaying every single hit from every abstract. All hits are sorted by frequency on an ascending order using QuickSort.

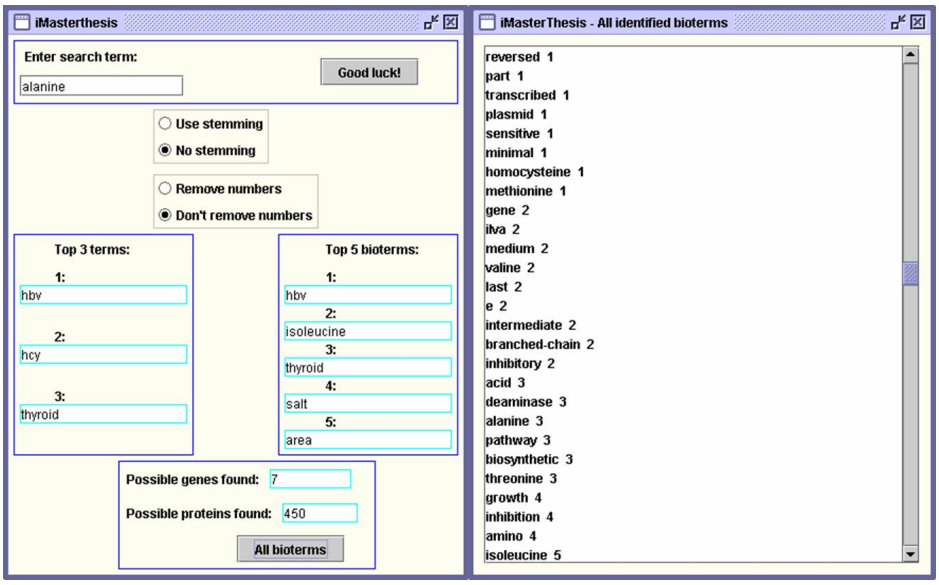

Figure 12: Screenshot of the prototype

5.2 The dictionary

Since the dictionary is such a vital part of this approach, the time-consuming process of creating it will be discussed here. The 2.6GB XML file, that originally was downloaded from UniProt and parsed[11], contained 2326498 entries. This file was far from ready to be used. Several hours had to be spent to improve the quality. Below is table 1 showing several of those steps this file went through. As can be seen from this, less than half of the entries were unique. The steps mentioned in the table consisted of removing names that gave absolutely no meaning. For example "UniRef100 entry" means that we removed entries on the form: "UPI00003A957A UniRef100 entry". These are just empty entries.

At this point the file had shrunk to a size of 493.814. Later on we discovered that there were areas of improvement in this file. Several of the names were still incapable of matching any names in the abstracts because of their forms. Some names were incredibly long. All names were tested for their length in number of tokens and we found out that the longest "name" was 181 tokens long. There were 12 names longer than 100 tokens, 218 longer than 40 tokens, 5424 longer than 20 tokens and 23 386 longer than 10 tokens. A decision was made to remove all entries longer than 10 tokens. The length

[11]The parsing and writing to a textfile alone took about 1.5 hrs using a BufferedWriter.

Step	Removed	New size	Percent
Original		2.326.498	100
Duplicates	1.322.705	1.003.734	43.1
"PREDICTED: hypohetical.."	12.406	991.328	42.6
"UniRef100 entry"	172.568	818.760	35.2
"..(h|H)ypothetical.."	244.113	574.647	24.7
"PREDICTED: .."	54.234	520.413	22.4
decimals	41	520.372	22.4
"(S|s)imilar to.."	26.558	493.814	21.2

Table 1: Processing of the protein dictionary

of the longest word decides the depth of the hashtable-tree-structure and it is anyhow extremely unlikely to get a match on such long names. After this removal the size was 470 089 entries (20.2%).

As part of a finalizing process, we removed the first word of names which started with *new, predicted, potential, probable, putative, related to*. A name like *potential kinase protein 2* is not likely to get a match. This effort affected a total of 45.909 names. A new removal of duplicates was run and the file shrank to a size of 427.045 (18.4% of original size). This process was then repeated on entries that started with *similarity to, possible, similarities with* and some other regular expressions. In total this last step affected 7632 entries. The resulting final protein name dictionary ended up of size 424.979 entries which corresponds to 18.3% of the original size.

The processing of the gene name dictionary was much smoother, mostly because the names were just neater. Some of the entries in the protein file were more than 100 words long, whereas all the gene name entries were single-word entries. From an original file of 23 569 approved symbols we found that all names were unique. But a total of 2 296 entries that were on the form *zgt09∼withdrawn* were removed. The resulting gene dictionary is of size 21 273 entries which corresponds to 90.3% of the original size. Examples of both the protein and gene name dictionaries can be found in appendix A.

5.3 The lookup mechanism

To realize the flexible approach that was described in section 4.1, we implemented the protein name dictionary as a 10-level deep tree of hash tables. Hash tables are extremely fast for doing lookup, the cost is only O(1) when

you lookup the value of a given key. That is assuming the hash function is an O(1) computation. But the downside is that they require quite some cost for creation and they take up huge amounts of space, this is especially true for this approach. The gene name dictionary on the other hand is 20 times smaller and is also only implemented as a 1-level hash table, hence the costs for its creation are negligible in comparison.

The creation of the hash tables that implement the dictionaries takes place in the ProduceHashTables class. Below is a little piece of pseudocode explaining this:

```
createProteinHashTables ()
1   for all lines in the protein names file
2     split the line into a string array s of tokens (max size 10)
3       for all tokens in s
4         if 0th token :
5           if hashtable at level 0 (ht0) already contains the token :
6             get the internal hashtable (ha0) which is the value of the token key
7             put the token key and internal hashtable to ht0
8           else :
9             put the token key and a new Hashtable to ht0
10        if 1st token :
11          if hashtable at level 1 (ht1) already contains the token :
12            get the internal hashtable (ha1) which is the value of the token key
13            put the token key and ha1 to ht1
14            get the internal hashtable at level 0 (ha0)
15            put the token key and a dummy variable to ha0
16            put the 0th token and ha0 to ht0
17          else :
18            put the token key and a new Hashtable to ht1
19            get the internal hashtable at level0
20            put the token key and a dummy variable to ha0
21            put the 0th token and ha0 to ht0
22        .
23        .
24        .
25        for levels 2 through 9 it is identical as for level 1
```

A hash table consists of (key,value)-pairs. In this implementation we use the protein names as keys, and for each key there is a hash table at the value position containing all the possible keys that this key maps to at the next level (the next token in the name). The number of unique keys at each level is naturally decreasing as there are fewer names of length 8 words than of length 3 words. Below is a list of the number of keys at every level:

1. 206 304
2. 24 391
3. 33 314
4. 23 086
5. 20 287
6. 9 578
7. 6 212
8. 3 717
9. 2 051
10. 925

Basically, this datastructure becomes very space- and resource-demanding. Our application spends about 24 seconds on building this structure. A huge part of this cost is naturally the I/O-cost of reading the textfile containing the 425K lines of protein names. The application was run on 512 MB of RAM, using the $java - Xmx500m$-command to make sure that the application did not run out of memory. It is not possible to write the hash table-structure to file and read it in again at runtime, this is too memory-demanding. It has to be created and loaded in to memory every time the application is run, but once it is created, lookup is obviously very fast. Figure 13 is an example illustration of the hashtable-structure at level 0 and 1.

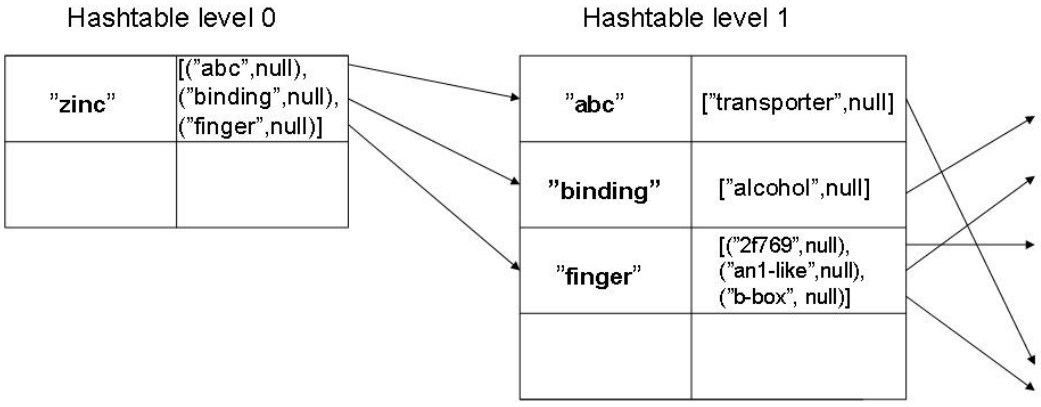

Figure 13: Example hash table-structure at level 0 and 1

The actual lookup itself is done in the readAbstract-method in the LookUp class. This method traverses every token in the abstract text and looks it up

in the gene and protein dictionaries. If it matches a gene, the token is added to the list of identified genes, and the traversal jumps to the next token. If it is a protein match, the lookup will then proceed to try and match the consecutive token in the abstract text with a key in this token's internal hash table. This will indicate a match of length 2, and the process may follow the pointers to the next level in the hash table-structure, as shown in figure 13, to try and get a match of length 3. The traversal of the abstract text's tokens will then jump the corresponding number of places, to make sure all tokens are only attempted matched once.

5.4 Postprocessing

The initial result set from the lookup process is rather large. To increase the level of precision, these tokens are subject to postprocessing in the Lookup class. This process consists of several steps. The first step is removing eventual stopwords. Secondly, purely numerical entities are removed. As a third step, entries consisting of 1-2 letters are promptly eliminated. The next step is removing entries that match any entry in a specific postprocessing stopwords-list. This stopwords-list has been developed by studying the 300 most frequent, non-relevant entries that were returned on different input-terms. The fifth step is removal of decimal number-entries. The succeeding step is the removal of entries of length 1 word that do not match any entry in a one-level hash table containing all the protein names in the dictionary. Only entries of length 1 that have a match in this hash table are considered potential protein names. This step dramatically increases precision because of removing entries that initially are assumed to be protein names because of the flexible matching-strategy, but in fact simply are uninteresting first words of longer protein names. The seventh and last step is the removal of entries containing only measures, e.g. *kda*[12]-entries. Figure 14 summarizes the steps in the postprocessing of the set of potential bioterms.

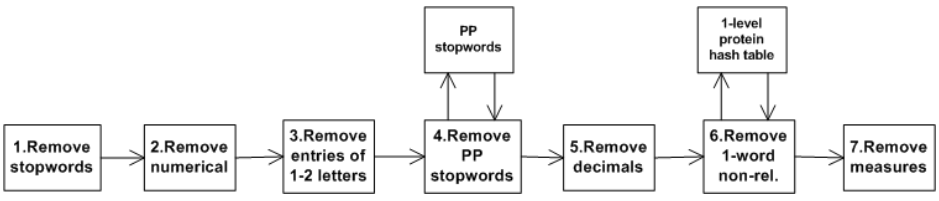

Figure 14: Steps in postprocessing

[12]Abbreviation for kilodalton.

5.5 Programming environment

Considering it was decided to develop the application in Java and make it available through Java Web Start, a short presentation of these technologies will be given in the consecutive subsections. XML, and parsing of XML documents, which are core technologies in this approach, are also addressed in this section.

5.5.1 XML

XML is a subset of SGML (Standard Generalized Markup Language), designed to ease the exchange of structured documents over the Internet. XML can be dated back to November 14, 1996, the publishing date of the oldest working draft regarding XML currently listed at the W3C, called "Extensible Markup Language (XML)"[18]. XML consists of logical entities called elements and has clearly defined start and end points. An XML document is extensible and self describing and can exist on its own without any further definition of its structure[17].

Alternatively, an XML document can be defined through a DTD (Document Type Definition) or through XML Schema. Both of these are W3C recommendations[13]. Drawbacks with DTD is the lack of XML-syntax and its limited expressive power. XML Schema on the other hand is more flexible. It is wellformed XML which allows the user to define datatypes such as string and integer. XML Schema also allows uniqueness constraints and references between elements.

5.5.2 Parsing

Our approach requires handling of XML documents at runtime. To work with XML in programming languages it is necessary to parse the XML documents. There exists two main parsing technologies: DOM and SAX.

The W3C DOM (Document Object Model) is providing a standard set of objects for representing HTML and XML documents and a standard interface for accessing and manipulating them. It is separated into the Core DOM, the XML DOM and the HTML DOM. Only the XML DOM is of interest in this context.

[13]This means that they are technologies that have reached a consensus both in- and outside of the World Wide Web Consortium.

The XML DOM is platform and language independent and is a W3C standard. It views XML documents as a tree structure of elements embedded within other elements. All elements can be accessed through the DOM tree. All the nodes in the DOM tree represent objects which have functions and an identity. DOM is very complex and not ideal for parsing large XML documents[19],[20]. JDOM is a simple and Java-specific API that has been developed to exploit special Java features and avoid the complexity of the general DOM API.

SAX (Simple API for XML) is clearly the best choice for parsing of huge XML documents. In SAX an XML tree is not viewed as a data structure, but as a stream of events that is generated by the parser. These events are e.g. encountering an element start tag, an element end tag or character data inside an element. These events invoke a callback method that the programmer writes. A data structure is not necessarily built during parsing, and this is convenient for memory usage reasons. An XML document can be piped through an application very quickly. To use SAX one needs:

- Java 1.1 or higher
- SAX2-compatible XML parser installed on the Java classpath
- The SAX2 distribution installed on the Java classpath

Since we had to parse an XML file of size close to 3 GB to build the protein dictionary, our choice landed on a SAXParser. It was the only choice considering the huge memory cost it actually is parsing such a large file. Using a DOM parser would most likely have required a larger RAM on the system.

5.5.3 Java

Java is an objectoriented programming language that can be traced back to May 1995. At this point Sun Microsystems released the first Java Development Kit (JDK) through the Internet, making it possible for developers world wide to download and start using it[21]. 1998 saw the release of Java 2 SDK, Standard Edition (J2SDK).

Java is tightly knit to the Web and is sometimes called the World Wide Web programming language, but it is a powerful programming system that is used in numerous applications. The Java technology consists of both a programming language and a number of specialized platforms. The programming language Java makes it possible to write programs that run in the browser,

from a desktop, on a server or any user application. Java programs are interpreted by a Java Virtual Machine (JVM) for the "native" operative system. This means that any computer system with a JVM installed, is able to run a Java program regardless of where the application was developed. This unique portability is perhaps the greatest strength of Java.

5.5.4 Java Web Start

Java Web Start is a technology that enables standalone Java software applications to be deployed with a single click over the network. It ensures that the most current version of the application will be deployed, as well as the most correct version of the Java Runtime Environment (JRE). Java Web Start is included in the JRE as part of J2SE.

In the spirit of Java's portability, Java Web Start works with any Web server and browser. The browser-independent architecture is ensured by the Java Network Launching Protocol & API (JNLP)[22].

To use the Java Web Start, a developer has to create a JAR (Java Archive) file containing all necessary .class-files and other resources. Additionally, a manifest-file has to be put inside the JAR file, containing a pointer to the main class of the application. If a client is to be given full access to the system, one also has to sign the JAR file. Finally, a JNLP file has to be produced. This is a small XML file containing additional information about the application.

6 Results

To evaluate the system performance in terms of precision and recall, it was imperative to have a *golden standard* available for testing. This golden standard had to be biomedical texts that had been tagged by experts, in particular these texts ought to be Medline abstracts. We managed to find suitable data sets for testing on the web[26]. As part of the project "Proteinhalt i text" (Protein concentration in text), a Swedish research group developed the protein name tagger Yapex (described in section 2.6.2). Through their web site they also provide access to two data sets of tagged Medline abstracts as XML files. These data sets were one reference set consisting of 99 abstracts and a test set of 101 abstracts. The protein names from all 200 abstracts were annotated by domain experts connected to the Yapex project. Table 2 shows the statistics of the golden standard data. As the reader will notice, the average numbers of protein names per abstract for these data sets are quite high.

Data set	**Size**	**Number of tags**	**Avg. tags/abstract**
yapex_ref_collection	99	1559	15.7
yapex_test_collection	101	1789	17.7

Table 2: The golden standard

The reference collection of 99 abstracts was used to develop and improve our system. We then later on proceeded to test and evaluate the system using the test collection. The results from testing can be found in table 3.

The table shows results from several of the postprocessing steps leading to the final results. We tested the system using four different settings, according to true/false-values of using stemming on the abstracts and removing purely numerical entities.

The first row of results indicates the percentage of stopwords initally removed from the abstracts. "Number of hits" indicates the number of protein hits that are found in the test sets initially. As can be seen, this number is between 3 to 4 times the number of tags in the golden standard. The next 9 rows provides the number of hits with length more than 1 word that have been found. The system did not find any stopwords amongst the bioterm hits. The next row indicates how many purely numerical entities have been removed from the hits. A zero indicates a rN-setting of true. "1-2 signs" indicates how many hits consisting of a word of length 1 to 2 signs that

	yapex_ref_collection				yapex_test_collection			
	¬uS,¬rN	¬uS,rN	uS,¬rN	uS,rN	¬uS,¬rN	¬uS,rN	uS,¬rN	uS,rN
Percent stopwords	0,374	0,374	0,374	0,374	0,363	0,363	0,363	0,363
Number of hits	6223	6050	5062	4868	6165	6050	4811	4692
Hit of length: 2	682	642	401	355	621	605	381	360
3	184	170	62	50	154	150	48	44
4	61	51	13	11	49	46	13	11
5	14	9	1	1	14	7	1	1
6	4	1	1	0	2	2	0	0
7	2	0	1	0	0	0	0	0
8	1	0	1	0	0	0	0	0
9	0	0	1	0	0	0	0	0
10	0	0	0	0	0	0	0	0
Removed stopw.	0	0	0	0	0	0	0	0
Num. entities	152	0	166	0	103	0	108	0
1-2 signs	309	309	367	367	320	320	313	314
PP stopwords	625	636	26	26	492	493	22	22
Decimalnumbers	6	0	6	0	5	0	5	0
Nonrel. 1-word	3909	3923	4122	4146	4014	4022	4004	4018
kda	17	0	22	0	7	0	7	0
New size	1205	1182	353	329	1224	1215	352	338
Number of unique	632	615	195	178	577	572	174	165
Precision	0,1402	0,1387	0,0283	0,0243	0,1422	0,1366	0,0398	0,0207
Recall	0,1084	0,1052	0,0064	0,0051	0,0973	0,0928	0,0078	0,0039

Table 3: Results from testing. uS and rN are short forms of use stemming and remove numbers.

have been identified and removed. "PP stopwords" informs the reader of hits that matched entries in the postprocessing stopwords-list. The next row indicates identified decimalnumber-hits. The "Nonrel. 1-word" row indicates the number of entries that have been removed because of being of length 1 and not matching any entry in the 1-level protein hash table. A few entries are also removed containing only measures (kda). The fourth-to-last row shows the new size of the set of potential protein names, which is the size that is to be used for calculation of precision and recall values. The succeeding row shows the number of unique entries in this set of potential hits. The last two rows presents values for precision and recall.

From the results we can immediately see that the number of initial hits is too high. In the test setting of not using stemming and not removing numerical entities, which is the one that gave best results, the number of hits is almost four times the number of tags in the golden standard. We can also notice that 840 hits (13.6%) were of length 2 or higher. Hence the large majority of returned hits are 1-word entries. A large portion of these entries are removed in the "Nonrel. 1-word" step. Out of the 5325 1-word entries in this test setting result set, 4014 were removed (75.4%). It is also clear that the fifth test setting gave the best results for precision and recall. Removing numbers in the preprocessing of the abstracts affects performance in a slightly negative

way. The usage of stemming gave very poor results in this approach. Figure 15 presents the results for precision and recall for all eight test settings in a graphical form. The maximum performance of this system, quantified by evaluating it on 101 previously untested Medline abstracts, is roughly 10% recall and 14% precision.

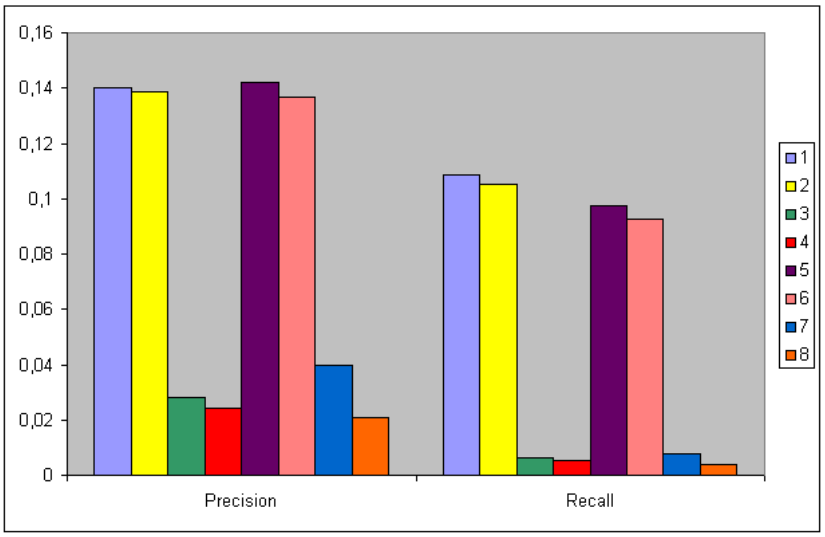

Figure 15: Diagram of test results. The left 8 columns show results for precision and the rightmost 8 show results for recall. The numbers 1-8 correspond to the 8 different test settings shown in table 3.

7 Discussion

In this section the reader will be provided with an assessment of the system. The system's performance will be assessed on basis of what was the initial idea and intentions of the approach. The section ends with a conclusion based on the discussion and finally outlines directions for future work.

An initial reaction to the system performance is slight disappointment. We had hoped to reach results in the region of 30%. But still, it is interesting and rewarding to analyze the results in an attempt to learn from the approach.

The ProtScan approach, which served as an inspiration in the initial stages of developing this approach, delivered results of 98% precision and 88% recall on a test set of 1000 "randomly selected and hand-tagged" Medline abstracts. What immediately stands out to us is the fact that these 1000 abstracts only contained 1914 protein names, i.e. an average of less than 2 protein names per abstract. In fact, 688 of the abstracts in the test set contained no proteins. This seems suspicious. However, it is unfair to compare the results of our approach to the results of the Protscan approach, so we will not dwell more with this.

The hard facts of this approach's performance is that it identifies approximately 10% of the relevant terms in the golden standard, and approximately 14% of the terms it identifies are relevant terms. The relevant terms that are identified are absolutely perfect matches[14].

The intentions of this approach (see section 4.1) was to facilitate a more flexible matching-scheme. The idea was to return the longest possible term that hopefully was a match. This still remains a good idea, and reading from the results in section 6, one can see that the best combination of test settings gave 2 matches on 6-word-entries, 14 matches on 5-word-entries, 49 matches on 4-word-entries, 154 matches on 3-word-entries as well as 621 matches on 2-word-entries. Another test setting even gave match to a 9-word-entry. We are certain that these hits would have been very difficult to obtain with a one-level hashtable-approach. Hence, we would argue that it is a good strategy for seeking to get longest possible matches. The weakness of this approach is the one-word non-relevant hits that are returned. Even after removal of all entries that are not present in the one-level protein hashtable, there are still multiple one-word non-relevant hits. While testing the system on the reference collection, many entries of this type were removed from the dictionary. For example there were unique entries in the protein dictionary

[14]The terms are compared with the equals()-method in Java.

for words like *"salt", "fit"* and *"area"*. Removal of every such entry needs to be done manually, preferably by experts and definitively requires a lot more resources than what was available for this project.

The results also show that the usage of stemming has a negative impact on performance in this approach. This has very obvious reasons. When the linked hash table structure is created in useStemming-mode, potentially wrongly linked names will occur. This is because two different second-level words in a name might potentially have identical word stems, and thus be linked to from the same first-level word. In general, stemming seems to have minimal to negative effect on performance on biomedical texts. As Saetre[27] pointed out, protein names are often rooted in Latin or have acronyms as their name. For example, the porter stem of *gas* (gastrin) becomes *ga*, which is wrong and too ambiguous.

The removal of numerical entities from the abstracts also seems to slightly decrease the values for recall and precision. This was intended to be a user-option in the interface, and the idea was that one could remove many irrelevant terms such as year-entries and other uninteresting purely numerical entries at the preprocessing step, thus reducing the size of the abstract texts. But this seems unwise, since numbers are a vital part of many protein names.

In the end, the success of this approach comes down to the quality of the dictionary, and then in particular the protein name dictionary. To increase the value of recall, it is imperative that the expert-tagged protein name actually is present in the dictionary. The dictionary used for this approach is not extensive enough, even though the UniProt-resource allegedly is the world's most comprehensive catalog of information on proteins. The Protscan-project was in possession of biology-experts that manually processed the dictionary clearing out ambiguities and thus dramatically increasing the quality. Usage of such resources was not feasible in this project, but the authors recognize the advantages of this.

Part of the problem with searching in abstracts as opposed to searching full text articles, is the fact that authors are forced to compress information, actually altering the writing style. This fact is reflected in the Textpresso-approach, where the authors actually reported recall values of 44.6% and 94.7% for searching abstracts and full text respectively. Clearly, comparisons with Textpresso puts this approach in a better light. Recall values in the 40%-range does not seem that out of reach, given the potential of improvement of the dictionaries.

7.1 Conclusion

This approach presented a prototype for identification of biomedical terms in Medline abstracts. The prototype allows an user to search for a biomedical term, before it fetches ten abstracts from the Entrez PubMed interface of Medline. These abstracts are then processed, and the user is provided with the most relevant terms identified from these abstracts. The user is also allowed to view all terms that have been identified.

Evaluation of the approach, on a golden standard of annotated data sets of Medline abstracts, shows that the maximum performance of the system equates to a 10% recall and 14% precision. The usage of stemming seems to have a negative impact on performance, partly because of many protein names being acronyms whose word stems might give directly wrong results, and partly because of the distinct core hash table structure. The removal of numerical entities from the abstracts also affects the performance negatively, but only slightly so.

The results revealed from the evaluation are in direct correlation with the quality of the dictionaries that are being used for lookup. Even though multiple hours have been spent improving these, the quality still seems to be inadequate. Improving the dictionaries, as well as expanding them by adding additional protein name resources, is a topic left for future work.

7.2 Future work

The improvement of the dictionaries seems to be an obvious direction for future work on this system. To obtain as perfect results as possible from such an approach, the dictionaries should be manually processed by domain experts, after automatic processing. Adding multiple new different protein name resources to the existing dictionary is also a possibility that should be considered. The gene name dictionary on the other hand seems to be adequate. But it is seldom that gene names are used in the abstracts, it is almost only protein names. So the focus should be on the protein names.

The approach was originally intended to be a hybrid approach, combining dictionary lookup with certain pattern-based rules for identification of biomedical entities. Because of time limitations, this approach was not realized, but it is definitively an interesting direction of study that could yield improved results.

A Dictionary

This appendix shows examples of the gene and protein name dictionaries that are being used by the system. Both examples contain 35 consecutive entries.

A.1 Gene name dictionary

```
1  lmcd1
2  lmln
3  lmna
4  lmnb1
5  lmnb2
6  lmnl1
7  lmnl2
8  lmo1
9  lmo2
10 lmo3
11 lmo4
12 lmo6
13 lmo7
14 lmod1
15 lmod2
16 lmod3
17 lmpdz1
18 lmtk2
19 lmtk3
20 lmx1a
21 lmx1b
22 lnpep
23 lnx
24 lnx2
25 loh11cr2a
26 loh12cr1
27 loh12cr2
28 loh18cr1
29 loh19cr1
30 loh1cr1
31 loh3cr2a
32 lor
33 lox
34 loxhd1
35 loxl1
```

A.2 Protein name dictionary

```
1  hydroxyethylthiazole kinase family
2  hydroxyethylthiazole kinase putative
3  hydroxyethylthiazole kinase thim/thik
4  hydroxyethylthiazole kinase-like protein
5  hydroxyethylthioazole kinase
6  hydroxyindole o-methyltransferase
7  hydroxyindole o-methyltransferase ec 2.1.1.4 hiomt acetylserotonin o-methyltransferase
       asmt
8  hydroxyindole-o-methyltransferase
9  hydroxyindole-o-methyltransferase isoform a
10 hydroxyisourate hydrolase
11 hydroxylacyl-coa dehydrogenase
12 hydroxylamine oxidoreductase
13 hydroxylamine oxidoreductase precursor
14 hydroxylamine oxydoreductase
15 hydroxylamine reductase
16 hydroxylamine reductase 1
17 hydroxylamine reductase 2
18 hydroxylaminobenzene mutase
19 hydroxylaminobenzene mutase hab
20 hydroxylaminochlorobenzene mutase
21 hydroxylase
22 hydroxylase alpha subunit
23 hydroxylase alpha subunit 2
```

```
24  hydroxylase beta subunit
25  hydroxylase beta subunit 2
26  hydroxylase blmf
27  hydroxylase component
28  hydroxylase for synthesis of 2-methylthio-cis-ribozeatin in trna
29  hydroxylase large subunit
30  hydroxylase molybdopterin-containing subunit
31  hydroxylase ncnh
32  hydroxylase of the polyketide produced by the pks cluster
33  hydroxylase protein
34  hydroxylase putative
35  hydroxylase small component
```

B Source code

This appendix presents all the Java source files. There are 15 files in total. They are presented in the following order: *BioTermsWindow* (page 47), *GUI* (page 49), *Main* (page 60), *Parse* (page 65), *Tokenize* (page 71), *PreProcessing* (page 81), *Stopwords* (page 88), *Stemmer* (page 90), *Result* (page 96), *Weight* (page 97), *Dictionary* (page 104), *ProduceHashTables* (page 105), *Test* (page 113), *ProcessDictionary* (page 118) and *LookUp* (page 128).

BioTermsWindow.java

```
1  package master;
2
3  import java.awt.Color;
4  import java.util.Vector;
5
6  import javax.swing.BorderFactory;
7  import javax.swing.BoxLayout;
8  import javax.swing.JFrame;
9  import javax.swing.JList;
10 import javax.swing.JPanel;
11 import javax.swing.JScrollPane;
12 import javax.swing.UIManager;
13
14 /*
15  *Denne klassen skal vise alle bioterm-treff i et enkelt vindu
16  *kun bestående av et JTextArea.
17  */
18
19 /**
20  *@author skuland
21  *
22  *
23  */
24
25 public class BioTermsWindow{
26
27   private Vector v;
28   private int xCoordinate;
29   private int yCoordinate;
30   private int jframeheight;
31   private int jframewidth;
32
33   /*
34   *Konstruktoren. Denne får som input navnet som skal
35   *brukes i hovedvinduet, en vektor med alle biotermene,
36   *samt koordinater som bestemmer hvor på skjermen vinduet
37   *vil være.
38   */
39   public BioTermsWindow(String name, Vector v, int xCoordinate, int yCoordinate,
       int jframewidth, int jframeheight)
40   {
41     this.v = v;
42     this.xCoordinate = xCoordinate;
43     this.yCoordinate = yCoordinate;
44     this.jframewidth = jframewidth;
```

```
45      this.jframeheight = jframeheight;
46
47      createGUI(name);
48    }
49
50    /*
51    *Denne metoden skaper GUIen med navnet som hentes
52    *inn i konstruktoren.
53    */
54    private void createGUI(String name)
55    {
56      JFrame.setDefaultLookAndFeelDecorated(true);
57
58      JFrame frame = new JFrame(name);
59      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
60      frame.setSize(440, 540);
61      frame.setLocation(xCoordinate + jframewidth + 1, yCoordinate);
62      frame.setResizable(false);
63
64      JPanel contentPane = new JPanel();
65      contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.PAGE_AXIS));
66      contentPane.setOpaque(true);
67      frame.setContentPane(contentPane);
68
69      Color col = new Color(255,255,240);
70      contentPane.setBackground(col);
71
72      JPanel panel = new JPanel();
73      panel.setLayout(new BoxLayout(panel, BoxLayout.PAGE_AXIS));
74      panel.setBackground(col);
75      panel.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
76
77      JList list = new JList(v);
78      JScrollPane scrollPane = new JScrollPane(list);
79
80      panel.add(scrollPane);
81
82      contentPane.add(panel);
83
84      try
85      {
86        UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
87      }
88      catch (Exception e){}
89
90      frame.setVisible(true);
91    }
92  }
```

GUI.java

```
1  package master;
2
3  import java.awt.Color;
4  import java.awt.Dimension;
5  import java.awt.GraphicsConfiguration;
6  import java.awt.Rectangle;
7  import java.awt.event.ActionEvent;
8  import java.awt.event.ActionListener;
9  import java.awt.event.KeyEvent;
10 import java.awt.event.KeyListener;
11 import java.util.ArrayList;
12 import java.util.Vector;
13
14 import javax.swing.BorderFactory;
15 import javax.swing.Box;
16 import javax.swing.BoxLayout;
17 import javax.swing.ButtonGroup;
18 import javax.swing.JButton;
19 import javax.swing.JFrame;
20 import javax.swing.JLabel;
21 import javax.swing.JPanel;
22 import javax.swing.JRadioButton;
23 import javax.swing.JTextArea;
24 import javax.swing.JTextField;
25 import javax.swing.UIManager;
26
27 /*
28  *Denne klassen skal skape et lite brukergrensesnitt for
29  *prototypen.
30  */
31
32 /**
33  *@author skuland
34  *
35  */
36 public class GUI implements ActionListener, KeyListener{
37
38   private JTextField searchTextField;
39   private JTextArea geneArea, proteinArea;
40   private JTextArea first,second,third;
41   private JTextArea one,two,three,four,five;
42   private boolean useStemming = false;
43   private boolean removeNumbers = false;
44   private boolean readyToInsertText = false;
45   private boolean enterPressed = false;
46
47   private int jframewidth = 440;
48   private int jframeheight = 540;
49
50   private final int jFrameXCoordinate = 200;
51   private final int jFrameYCoordinate = 242;
52
53   private int xCoordinate;
54   private int yCoordinate;
55   private int height;
56   private int width;
57
58   private Vector v;
```

```
59    public Main main;
60    public Dictionary d;
61
62    /*
63    *Konstruktoren initierer GUIen
64    */
65    public GUI()
66    {
67      createGUI();
68
69    }
70
71    /*
72    *Dette er metoden som setter i gang både Main
73    *og BioTermsWindow-klassen ved handling fra brukeren.
74    */
75    public void actionPerformed(ActionEvent e)
76    {
77
78      /*
79      *Dersom noen har trykket på "Good luck!"-knappen
80      *(eller enter) og tekstfeltet ikke er tomt
81      */
82      if( ( (e.getActionCommand().equals("Good luck!"))  )  && !(searchTextField.
          getText().equals(null)))
83      {
84        String searchTerm = searchTextField.getText();
85        //d = new Dictionary(useStemming);
86        main = new Main(searchTerm, this, useStemming, removeNumbers);
87      }
88      //dersom brukeren ikke ønsker stemming
89      else if(e.getActionCommand().equals("Use stemming"))
90      {
91        System.out.println("stemming here");
92        useStemming = true;
93      }
94
95      else if(e.getActionCommand().equals("Remove numbers"))
96      {
97        System.out.println("Remove numbers!!");
98        removeNumbers = true;
99      }
100
101     else if((e.getActionCommand().equals("All bioterms")) && readyToInsertText)
102     {
103       System.out.println("aaaaalllllll bioterms");
104       BioTermsWindow btw = new BioTermsWindow("iMasterThesis - All identified
            bioterms", v, jFrameXCoordinate, jFrameYCoordinate, jframewidth,
            jframeheight);
105     }
106
107
108   }
109
110   public void keyTyped(KeyEvent ke)
111   {
112
113   }
114
115   public void keyPressed(KeyEvent ke)
116   {
117
```

```
118   }
119
120   /*
121   *Denne metoden gjør at et enter-trykk har samme
122   *effekt som å trykke på "good luck!"-knappen dersom
123   *man har fylt inn et søkeord i søkefeltet.
124   */
125   public void keyReleased(KeyEvent ke)
126   {
127     if( ((ke.getKeyCode()) == KeyEvent.VK_ENTER)   && !(searchTextField.getText()
        .equals(null)))
128     {
129       String searchTerm = searchTextField.getText();
130       main = new Main(searchTerm, this, useStemming, removeNumbers);
131     }
132   }
133
134   /*
135   *Denne metoden får inn arraylisten resultList som
136   *inneholder samtlige gen og protein-treff sortert etter
137   *frekvens oppadgående. Størrelsen på geneSortedList
138   *og protSortedList skal være lik.
139   */
140   public Vector receiveResultList(ArrayList resultList)
141   {
142     v = new Vector();
143     int abstractNumberCounter = 0;
144     int protein = 0;
145     int gene = 0;
146     String blank = " ";
147
148     ArrayList geneSortedList = (ArrayList)resultList.get(0);
149     ArrayList protSortedList = (ArrayList)resultList.get(1);
150
151     //k skal her være 0,2,4,...,18
152     System.out.println("genesortedList.size: "+geneSortedList.size());
153     for(int k = 0; k< geneSortedList.size()-1;k=k+2)
154     {
155       System.out.println(k);
156       String abstractString = new String("Abstract number: "+
          abstractNumberCounter);
157       v.add(abstractString);
158       v.add(blank);
159       v.add(blank);
160
161       abstractNumberCounter++;
162
163       ArrayList sortedGeneKeys = (ArrayList)geneSortedList.get(k);
164       ArrayList sortedGeneFrequencies = (ArrayList)geneSortedList.get(k+1);
165
166       ArrayList sortedProteinKeys = (ArrayList)protSortedList.get(k);
167       ArrayList sortedProteinFrequencies = (ArrayList)protSortedList.get(k+1);
168
169       String geneNames = new String("Gene occurences: ");
170       v.add(geneNames);
171       v.add(blank);
172
173       for(int i = 0; i < sortedGeneKeys.size(); i++)
174       {
175         String geneName = (String)sortedGeneKeys.get(i);
176         gene++;
```

```
177           String geneFrequency = ((Integer)sortedGeneFrequencies.get(i)).toString()
                ;
178           String geneString = new String(geneName + " "+ " " +geneFrequency);
179           v.add(geneString);
180         }
181         v.add(blank);
182
183         String proteinNames = new String("Protein occurences: ");
184         v.add(proteinNames);
185         v.add(blank);
186
187         for(int j = 0; j < sortedProteinKeys.size(); j++)
188         {
189           String proteinName = (String)sortedProteinKeys.get(j);
190
191           protein++;
192           String proteinFrequency = ((Integer)sortedProteinFrequencies.get(j)).
                toString();
193           String proteinString = new String(proteinName + " "+ " " +
                proteinFrequency);
194           v.add(proteinString);
195         }
196         v.add(blank);
197       }
198
199       System.out.println("Vector size: "+v.size());
200       System.out.println("Antall mulige gen: "+gene);
201       System.out.println("Antall mulige protein: "+protein);
202
203       insertHits(gene, protein);
204
205       readyToInsertText=true;
206
207       return v;
208     }
209
210     /*
211     *Dette er metoden som skaper GUIen.
212     */
213     public void createGUI()
214     {
215       JFrame.setDefaultLookAndFeelDecorated(true);
216
217       JFrame frame = new JFrame("iMasterthesis");
218       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
219
220       frame.setSize(jframewidth, jframeheight);
221       frame.setResizable(false);
222
223       frame.setLocation(jFrameXCoordinate, jFrameYCoordinate);
224
225       JPanel contentPane = new JPanel();
226       contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.PAGE_AXIS));
227
228       contentPane.setOpaque(true);
229       frame.setContentPane(contentPane);
230
231
232       //get the GraphicsConfiguration
233       GraphicsConfiguration gcon = contentPane.getGraphicsConfiguration();
234       Rectangle bounds = gcon.getBounds();
235       xCoordinate = bounds.x;
```

```
236     yCoordinate = bounds.y;
237     height = bounds.height;
238     width = bounds.width;
239
240     Color col = new Color(255,255,240);
241     contentPane.setBackground(col);
242
243     //panelLeftUpper
244     JPanel panelLeftUpper = new JPanel();
245     panelLeftUpper.setLayout(new BoxLayout(panelLeftUpper, BoxLayout.PAGE_AXIS));
246     panelLeftUpper.setBackground(col);
247
248     JLabel searchTerm = new JLabel("Enter search term:");
249     searchTextField = new JTextField(25);
250     searchTextField.addActionListener(this);
251     searchTextField.addKeyListener(this);
252     searchTextField.setMaximumSize(new Dimension(300,20));
253
254     panelLeftUpper.add(searchTerm);
255     panelLeftUpper.add(Box.createRigidArea(new Dimension(0,10)));
256     panelLeftUpper.add(searchTextField);
257
258     //panelRightUpper
259     JPanel panelRightUpper = new JPanel();
260     panelRightUpper.setLayout(new BoxLayout(panelRightUpper, BoxLayout.PAGE_AXIS)
        );
261     panelRightUpper.setBackground(col);
262
263     //søkeknappen som skal være i JPanel panelRightUpper
264     JButton searchButton = new JButton("Good luck!");
265     searchButton.addActionListener(this);
266     searchButton.addKeyListener(this);
267
268     panelRightUpper.add(searchButton);
269
270     //JPanel som er øverst, layout fra venstre til høyre
271     JPanel panelUpper = new JPanel();
272     panelUpper.setLayout(new BoxLayout(panelUpper, BoxLayout.X_AXIS));
273     panelUpper.setBackground(col);
274     panelUpper.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
275     panelUpper.add(panelLeftUpper);
276     panelUpper.add(Box.createRigidArea(new Dimension(5,0)));
277     panelUpper.add(panelRightUpper);
278     panelUpper.add(Box.createHorizontalGlue());
279     panelUpper.setPreferredSize(new Dimension(200,150));
280     panelUpper.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
        createLineBorder(Color.BLUE),panelUpper.getBorder()));
281
282     //JPanel som skal inneholde radioButtons - valg for stemming bl.a.
283     JPanel radioButtonPanel = new JPanel();
284     radioButtonPanel.setLayout(new BoxLayout(radioButtonPanel, BoxLayout.
        PAGE_AXIS));
285     radioButtonPanel.setBackground(col);
286
287     ButtonGroup bg = new ButtonGroup();
288     JRadioButton stemming = new JRadioButton("Use stemming");
289     stemming.setActionCommand("Use stemming");
290     stemming.addActionListener(this);
291     stemming.setBackground(col);
292     JRadioButton nostemming = new JRadioButton("No stemming");
293     nostemming.setActionCommand("No stemming");
294     nostemming.setSelected(true);
```

```
295     nostemming.addActionListener(this);
296     nostemming.setBackground(col);
297
298     bg.add(stemming);
299     bg.add(nostemming);
300
301     ButtonGroup bg2 = new ButtonGroup();
302     JRadioButton removeNumbers = new JRadioButton("Remove numbers");
303     removeNumbers.setActionCommand("Remove numbers");
304     removeNumbers.addActionListener(this);
305     removeNumbers.setBackground(col);
306     JRadioButton notRemoveNumbers = new JRadioButton("Don't remove numbers");
307     notRemoveNumbers.setActionCommand("Don't remove numbers");
308     notRemoveNumbers.setSelected(true);
309     notRemoveNumbers.addActionListener(this);
310     notRemoveNumbers.setBackground(col);
311
312     bg2.add(removeNumbers);
313     bg2.add(notRemoveNumbers);
314
315     JPanel stemmingPanel = new JPanel();
316     stemmingPanel.setLayout(new BoxLayout(stemmingPanel, BoxLayout.PAGE_AXIS));
317     stemmingPanel.setBackground(col);
318     JPanel numbersPanel = new JPanel();
319     numbersPanel.setLayout(new BoxLayout(numbersPanel, BoxLayout.PAGE_AXIS));
320     numbersPanel.setBackground(col);
321
322     stemmingPanel.add(stemming);
323     stemmingPanel.add(Box.createRigidArea(new Dimension(0,1)));
324     stemmingPanel.add(nostemming);
325     stemmingPanel.setBackground(col);
326     stemmingPanel.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
          createLineBorder(Color.lightGray),stemmingPanel.getBorder()));
327
328     numbersPanel.add(removeNumbers);
329     numbersPanel.add(Box.createRigidArea(new Dimension(0,1)));
330     numbersPanel.add(notRemoveNumbers);
331     numbersPanel.setBackground(col);
332     numbersPanel.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
          createLineBorder(Color.lightGray),numbersPanel.getBorder()));
333
334     radioButtonPanel.add(stemmingPanel);
335     radioButtonPanel.add(Box.createRigidArea(new Dimension(0,10)));
336     radioButtonPanel.add(numbersPanel);
337     radioButtonPanel.setBackground(col);
338
339     JPanel radioButtonPanelContainer = new JPanel();
340     radioButtonPanelContainer.setLayout(new BoxLayout(radioButtonPanelContainer,
          BoxLayout.LINE_AXIS));
341     radioButtonPanelContainer.add(radioButtonPanel);
342     radioButtonPanelContainer.setBackground(col);
343
344     //JPanel for de tre beste treff
345     JPanel topthreePanel = new JPanel();
346     topthreePanel.setLayout(new BoxLayout(topthreePanel, BoxLayout.PAGE_AXIS));
347     topthreePanel.setBackground(col);
348
349     JLabel top3 = new JLabel("Top 3 terms:");
350
351     //1. mest relevante term
352     JLabel fir = new JLabel("1:");
353     first = new JTextArea(1,10);
```

```
354     first.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
          createLineBorder(Color.cyan),first.getBorder()));
355     first.setEditable(false);
356     first.setMaximumSize(new Dimension(300,60));
357
358     //2.mest relevante
359     JLabel sec = new JLabel("2:");
360     second = new JTextArea(1,10);
361     second.setMaximumSize(new Dimension(300,60));
362     second.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
          createLineBorder(Color.cyan), second.getBorder()));
363     second.setEditable(false);
364
365     //3.mest relevante
366     JLabel thir = new JLabel("3:");
367     third = new JTextArea(1,10);
368     third.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
          createLineBorder(Color.cyan), third.getBorder()));
369     third.setEditable(false);
370     third.setMaximumSize(new Dimension(300,60));
371
372     topthreePanel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
373     topthreePanel.add(top3);
374     topthreePanel.add(Box.createRigidArea(new Dimension(0,10)));
375     topthreePanel.add(fir);
376     topthreePanel.add(first);
377     topthreePanel.add(Box.createVerticalGlue());
378     topthreePanel.add(sec);
379     topthreePanel.add(second);
380     topthreePanel.add(Box.createVerticalGlue());
381     topthreePanel.add(thir);
382     topthreePanel.add(third);
383     topthreePanel.add(Box.createVerticalGlue());
384     topthreePanel.setMinimumSize(new Dimension(170,208));
385     topthreePanel.setMaximumSize(new Dimension(170,208));
386     topthreePanel.setSize(new Dimension(170,208));
387     topthreePanel.setPreferredSize(new Dimension(170,208));
388
389
390     //JPanel for de fem beste biotreff
391     JPanel topfivePanel = new JPanel();
392     topfivePanel.setLayout(new BoxLayout(topfivePanel, BoxLayout.PAGE_AXIS));
393     topfivePanel.setBackground(col);
394
395     JLabel top5bio = new JLabel("Top 5 bioterms:");
396
397     JLabel oneLabel = new JLabel("1:");
398     one = new JTextArea(1,5);
399     one.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
          createLineBorder(Color.cyan), one.getBorder()));
400     one.setEditable(false);
401     one.setMaximumSize(new Dimension(300,60));
402
403
404     JLabel twoLabel = new JLabel("2:");
405     two = new JTextArea(1,10);
406     two.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
          createLineBorder(Color.cyan), two.getBorder()));
407     two.setEditable(false);
408     two.setMaximumSize(new Dimension(300,60));
409
410     JLabel threeLabel = new JLabel("3:");
```

```
411     three = new JTextArea(1,10);
412     three.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
          createLineBorder(Color.cyan), three.getBorder()));
413     three.setEditable(false);
414     three.setMaximumSize(new Dimension(300,60));
415
416     JLabel fourLabel = new JLabel("4:");
417     four = new JTextArea(1,10);
418     four.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
          createLineBorder(Color.cyan), four.getBorder()));
419     four.setEditable(false);
420     four.setMaximumSize(new Dimension(300,60));
421
422     JLabel fiveLabel = new JLabel("5:");
423     five = new JTextArea(1,10);
424     five.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
          createLineBorder(Color.cyan), five.getBorder()));
425     five.setEditable(false);
426     five.setMaximumSize(new Dimension(300,60));
427
428     topfivePanel.add(top5bio);
429     topfivePanel.add(Box.createRigidArea(new Dimension(0,10)));
430     topfivePanel.add(oneLabel);
431     topfivePanel.add(one);
432     topfivePanel.add(twoLabel);
433     topfivePanel.add(two);
434     topfivePanel.add(threeLabel);
435     topfivePanel.add(three);
436     topfivePanel.add(fourLabel);
437     topfivePanel.add(four);
438     topfivePanel.add(fiveLabel);
439     topfivePanel.add(five);
440     topfivePanel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
441     topfivePanel.setMinimumSize(new Dimension(170,208));
442     topfivePanel.setMaximumSize(new Dimension(170,208));
443     topfivePanel.setSize(new Dimension(170,208));
444     topfivePanel.setPreferredSize(new Dimension(170,208));
445
446     //JPanel som inneholder top3 og top5
447     JPanel topPanel = new JPanel();
448     topPanel.setLayout(new BoxLayout(topPanel, BoxLayout.LINE_AXIS));
449     topPanel.setBackground(col);
450     topPanel.add(topthreePanel);
451     topPanel.add(Box.createHorizontalGlue());
452     topPanel.add(Box.createRigidArea(new Dimension(10,0)));
453     topPanel.add(topfivePanel);
454
455     topthreePanel.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
          createLineBorder(Color.BLUE), topthreePanel.getBorder()));
456
457     topfivePanel.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
          createLineBorder(Color.BLUE), topfivePanel.getBorder()));
458
459     //JPanel med JButton for å få oversikt over alle proteintreff
460     JPanel bottomPanel = new JPanel();
461     bottomPanel.setLayout(new BoxLayout(bottomPanel, BoxLayout.PAGE_AXIS));
462     bottomPanel.setBackground(col);
463
464     JLabel geneLabel = new JLabel("Possible genes found:");
465     geneArea = new JTextArea();
466     geneArea.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
          createLineBorder(Color.cyan), geneArea.getBorder()));
```

```
467       geneArea.setEditable(false);
468       geneArea.setMaximumSize(new Dimension(80,20));
469       JPanel genePanel = new JPanel();
470       genePanel.setLayout(new BoxLayout(genePanel, BoxLayout.LINE_AXIS));
471       genePanel.setBackground(col);
472       genePanel.setBorder(BorderFactory.createEmptyBorder(2,2,2,2));
473       genePanel.add(geneLabel);
474       genePanel.add(Box.createRigidArea(new Dimension(10,0)));
475       genePanel.add(geneArea);
476
477       JLabel proteinLabel = new JLabel("Possible proteins found:");
478       proteinArea = new JTextArea();
479       proteinArea.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
            createLineBorder(Color.cyan), proteinArea.getBorder()));
480       proteinArea.setEditable(false);
481       proteinArea.setMaximumSize(new Dimension(80,20));
482       JPanel proteinPanel = new JPanel();
483       proteinPanel.setLayout(new BoxLayout(proteinPanel, BoxLayout.LINE_AXIS));
484       proteinPanel.setBackground(col);
485       proteinPanel.setBorder(BorderFactory.createEmptyBorder(2,2,2,2));
486       proteinPanel.add(proteinLabel);
487       proteinPanel.add(Box.createRigidArea(new Dimension(10,0)));
488       proteinPanel.add(proteinArea);
489
490       JButton bioTermsButton = new JButton("All bioterms");
491       bioTermsButton.addActionListener(this);
492       bottomPanel.add(genePanel);
493       bottomPanel.add(Box.createRigidArea(new Dimension(0,10)));
494       bottomPanel.add(proteinPanel);
495       bottomPanel.add(Box.createRigidArea(new Dimension(0,10)));
496       bottomPanel.add(bioTermsButton);
497       bottomPanel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
498       bottomPanel.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
            createLineBorder(Color.BLUE), bottomPanel.getBorder()));
499
500       contentPane.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
501       contentPane.add(panelUpper);
502       contentPane.add(Box.createRigidArea(new Dimension(0,5)));
503       contentPane.add(radioButtonPanelContainer);
504       contentPane.add(Box.createRigidArea(new Dimension(0,5)));
505       contentPane.add(topPanel);
506       contentPane.add(Box.createRigidArea(new Dimension(0,5)));
507       contentPane.add(bottomPanel);
508
509       try {
510         UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
511         //UIManager.setLookAndFeel("com.sun.java.swing.plaf.mac");
512       } catch (Exception e) { }
513
514
515       frame.setVisible(true);
516
517       System.out.println(bioTermsButton.size());
518       System.out.println(bioTermsButton.getLocation());
519       System.out.println(bottomPanel.size());
520
521     }
522
523     /*
524     *Denne metoden setter inn de fem mest
525     *relevante biotermene i GUIen etter indeks i
526     */
```

```
527   public void insertTopFiveRelevantBioTerms(String term, int i)
528   {
529     if(i==0)
530     {
531       one.insert(term,0);
532     }
533     if(i==1)
534     {
535       two.insert(term,0);
536     }
537     if(i==2)
538     {
539       three.insert(term,0);
540     }
541     if(i==3)
542     {
543       four.insert(term,0);
544     }
545     if(i==4)
546     {
547       five.insert(term,0);
548     }
549   }
550
551   /*
552   *Denne metoden setter inn de tre
553   *mest relevante generelle termene.
554   */
555   public void insertTopThreeRelevantTerms(String term, int i)
556   {
557     if(i==0)
558     {
559       first.insert(term,0);
560     }
561     if(i==1)
562     {
563       second.insert(term,0);
564     }
565     if(i==2)
566     {
567       third.insert(term,0);
568     }
569     else
570     {
571       //uinteressant i
572     }
573   }
574
575   /*
576   *Denne metoden setter inn antallet
577   *identifiserte gen og protein i GUIen.
578   */
579   public void insertHits(int countGeneHits, int countProteinHits)
580   {
581     Integer i = new Integer(countGeneHits);
582     String geneHits = i.toString();
583     geneArea.insert(geneHits,0);
584
585     Integer j = new Integer(countProteinHits);
586     String proteinHits = j.toString();
587     proteinArea.insert(proteinHits,0);
588   }
```

```
589
590 }
```

Main.java

```
1  package master;
2
3  import java.io.BufferedReader;
4  import java.io.BufferedWriter;
5  import java.io.DataOutputStream;
6  import java.io.File;
7  import java.io.FileNotFoundException;
8  import java.io.FileOutputStream;
9  import java.io.FileReader;
10 import java.io.FileWriter;
11 import java.io.InputStream;
12 import java.io.InputStreamReader;
13 import java.io.PrintStream;
14 import java.io.PrintWriter;
15 import java.net.URL;
16 import java.net.URLConnection;
17 import java.util.ArrayList;
18
19 import org.xml.sax.helpers.DefaultHandler;
20
21 /*
22  *The Main Class. This class initiates the GUI and starts off
23  *the whole chain of events.
24  */
25
26 /**
27  *@author skuland
28  *
29  *
30  */
31
32
33 public class Main extends DefaultHandler{
34
35     private FileOutputStream out; // declare a file output object
36     private PrintStream p; // declare a print stream object
37     private String pubmedid, a = "", b = "", term, retmode = "xml",
         pubmedIDString;
38     private int reldate, retmax;
39     private final int retstart = 0;
40     public File abstracts, pubmediditxt, eSearchResults, eSearchResultsCopy,
         eFetchResultsCopy, eFetchResults;
41     private FileReader fr;
42     private BufferedReader br;
43
44     private URL url, urlFetch;
45     private URLConnection urlConnection, urlConnectionFetch;
46     private DataOutputStream printout;
47     private InputStream inputStream, inputStreamFetch;
48     private InputStreamReader inputStreamReader, inputStreamReaderFetch;
49     private BufferedReader bufferedReader, bufferedReaderFetch;
50     private FileWriter filewriter, filewriterFetch;
51     private BufferedWriter bufferedwriter, bufferedwriterFetch;
52     private PrintWriter printwriter, printwriterFetch;
53     private ArrayList pubmedIDList, abstractTextList;
54
55     private BufferedReader geneReader, geneReader2;
56     private PrintWriter geneWriter;
```

```
57
58      /*
59      *Konstruktoren i Main-klassen. Denne blir kalt
60      *fra GUI-klassen under kjøring av prototypen. Den
61      *får inn searchTerm, guiobjektet, og boolean-verdier
62      *for useStemming og removeNumbers. Den kaller
63      *Tokenize.
64      *
65      */
66      public Main(String searchTerm, GUI g, boolean useStemming, boolean
          removeNumbers)
67      {
68          retmax(10);
69          reldate(100);
70          String x = searchTerm;
71
72          pubmedIDString = eSearch(x);
73          abstractTextList = eFetch(pubmedIDString);
74
75          //Dersom jeg ønsker å parse uniref100.xml
76          //Parse p = new Parse();
77          //p.parseProteinNamesXMLFile();
78
79          Tokenize tok = new Tokenize(pubmedIDString, abstractTextList, g,
              useStemming, removeNumbers);
80
81          //ProcessDictionary pd = new ProcessDictionary();
82          //long now = System.currentTimeMillis();
83          //ProduceHashTables pht = new ProduceHashTables();
84          //long after = System.currentTimeMillis();
85          //System.out.println((double)(after-now)/1000);
86
87      }
88
89      /*
90      *Denne metoden kontakter PubMed og henter abstractene
91      *ut fra den gitte pubmedIDString. Returnerer en
92      *ArrayList med abstract-tekstene.
93      */
94      public ArrayList eFetch(String pubmedIDString)
95      {
96          try{
97              String eFetchURL = "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/
                  efetch.fcgi?db=pubmed&id="+pubmedIDString+"&retstart="+retstart+"&
                  retmax="+retmax+"&retmode=xml&rettype=abstract";
98              urlFetch = new URL(eFetchURL);
99              urlConnectionFetch = urlFetch.openConnection();
100             urlConnectionFetch.setDoInput(true);
101             urlConnectionFetch.setDoOutput(true);
102             urlConnectionFetch.setUseCaches(false);
103             urlConnectionFetch.setRequestProperty("Content-Type", "application/x-
                  www-form-urlencoded");
104             urlConnectionFetch.connect();
105             inputStreamFetch = urlConnectionFetch.getInputStream();
106             inputStreamReaderFetch = new InputStreamReader(inputStreamFetch);
107             bufferedReaderFetch = new BufferedReader(inputStreamReaderFetch);
108
109             Parse pa = new Parse();
110             //abstractTextList = pa.returnAbstract(eFetchResults);
111             abstractTextList = pa.returnAbstract(inputStreamFetch);
112         }
113         catch(Exception e)
```

```
114         {
115             e.printStackTrace();
116         }
117         return abstractTextList;
118     }
119 
120     /*
121     *Denne metoden kontakter PubMed med en søketerm og
122     *får returnert en XML-fil med bl.a. pubmed-IDer.
123     *Metoden returnerer en pubmedIDString.
124     */
125     public String eSearch(String term)
126     {
127         this.term = term;
128 
129         try{
130             String eSearchURL = "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/
                  esearch.fcgi?db=pubmed&term="+term+"&reldate="+reldate+"&retstart="
                  +retstart+"&retmax="+retmax+"&retmode="+retmode;
131             System.out.println(eSearchURL);
132             url = new URL(eSearchURL);
133             urlConnection = url.openConnection();
134             urlConnection.setDoInput(true);
135             urlConnection.setDoOutput(true);
136             urlConnection.setUseCaches(false);
137             urlConnection.setRequestProperty("Content-Type", "application/x-www-
                  form-urlencoded");
138             urlConnection.connect();
139             inputStream = urlConnection.getInputStream();
140             inputStreamReader = new InputStreamReader(inputStream);
141             bufferedReader = new BufferedReader(inputStreamReader);
142 
143             Parse ga = new Parse();
144 
145             pubmedIDList = ga.returnPubMedIDs(inputStream);
146 
147             for(int j = 0; j<pubmedIDList.size();j++)
148             {
149                 if(j<pubmedIDList.size()-1)
150                 {
151                     a = (String)pubmedIDList.get(j)+ ",";
152                     b = b + a;
153                 }
154                 else
155                 {
156                     a = (String)pubmedIDList.get(j);
157                     b = b + a;
158                 }
159             }
160 
161             pubmedIDString = b;
162 
163         }
164         catch(Exception e)
165         {
166             e.printStackTrace();
167         }
168 
169         return pubmedIDString;
170     }
171 
172     public String inputSearchTerm(String searchTerm)
```

```
173     {
174         return searchTerm;
175     }
176
177     public void retmax(int j)
178     {
179         retmax = j;
180     }
181
182     public void reldate(int i)
183     {
184         reldate = i;
185     }
186
187     /*
188     *Main entry point.
189     */
190     public static void main(String args[]){
191         GUI gui = new GUI();
192         //Main main = new Main();
193
194     }
195
196     /*
197     *Metode som ble kalt under produksjon av
198     *genDictionary.
199     */
200     private void geneDictionaryToLowerCase()
201     {
202         try{
203             geneReader = new BufferedReader(new FileReader(new File("geneSorted.
                  txt")));
204             geneReader2 = new BufferedReader(new FileReader(new File("geneSorted2
                  .txt")));
205             }catch(FileNotFoundException fe){fe.printStackTrace();}
206         /*
207         try{
208             geneWriter = new PrintWriter(new BufferedWriter(new FileWriter(new
                  File("geneDictionary.txt"), true)));
209             } catch(IOException io){io.printStackTrace();}
210         */
211         String line = "";
212         int countA = 0;
213         int countB = 0;
214         try{
215         while( (line = geneReader.readLine()) != null )
216         {
217             countA++;
218                 if(line.length() > 0)
219                 {
220
221                     String newline = line.toLowerCase();
222                     geneWriter.println(newline);
223                     //geneWriter.println(countB);
224                     countB++;
225                 }
226
227         }
228         }catch(Exception e){}
229         try{
230             while( (line = geneReader2.readLine()) != null )
231             {
```

```
232                 countA++;
233                     if(line.length() > 0)
234                     {
235 
236                         String newline = line.toLowerCase();
237                         geneWriter.println(newline);
238                         //geneWriter.println(countB);
239                         countB++;
240                     }
241 
242             }
243             }catch(Exception e){}
244         System.out.println("Verdi til countA: "+countA);
245         System.out.println("Verdi til countB: "+countB);
246         System.out.println("geneDictionary er ferdig:");
247     }
248 
249 }
```

Parse.java

```
1  package master;
2
3  import java.io.BufferedWriter;
4  import java.io.CharArrayWriter;
5  import java.io.File;
6  import java.io.FileWriter;
7  import java.io.InputStream;
8  import java.io.PrintWriter;
9  import java.util.ArrayList;
10
11 import javax.xml.parsers.ParserConfigurationException;
12 import javax.xml.parsers.SAXParser;
13 import javax.xml.parsers.SAXParserFactory;
14
15 import org.xml.sax.Attributes;
16 import org.xml.sax.InputSource;
17 import org.xml.sax.Locator;
18 import org.xml.sax.SAXException;
19 import org.xml.sax.SAXNotRecognizedException;
20 import org.xml.sax.SAXNotSupportedException;
21 import org.xml.sax.SAXParseException;
22 import org.xml.sax.XMLReader;
23 import org.xml.sax.helpers.DefaultHandler;
24
25 /*
26  *This class provides the parsing of all
27  *XML files used in the system.
28  */
29
30 /*
31  *@author skuland
32  *
33  */
34 public class Parse extends DefaultHandler{
35
36   public String abstracttext;
37   private SAXParser parser2;
38   public File parsed_abstract;
39   public FileWriter fw;
40   public File uniref100;
41   public File proteinnames;
42   public BufferedWriter bw;
43   public PrintWriter pw;
44   private Locator locator;
45   public PrintWriter pw2;
46   public FileWriter fw2;
47   public BufferedWriter bw2;
48
49   private int counter = 0;
50   private StringBuffer sb;
51   private XMLReader parser;
52   public ArrayList pmidList, abstractTextList;
53   private CharArrayWriter charwriter;
54   boolean ampersand = false;
55   boolean b = false;
56
57   /*
58   *Konstruktoren i Parse-klassen.
```

```
59    *Denne skaper en SAXParser fra
60    *SAXParserFactory.
61    */
62    public Parse()
63    {
64      pmidList = new ArrayList();
65      abstractTextList = new ArrayList();
66      sb = new StringBuffer();
67
68      try{
69        uniref100 = new File("C:\\uniref100.xml\\uniref100xml");
70      }
71      catch(Exception e)
72      {
73        e.printStackTrace();
74        System.out.println("feil");
75      }
76
77      //Denne metoden prøvde jeg først med SAXParser fra parserfactory, men det gav
           problemer med UTF8 malformed char
78      try{
79      SAXParserFactory parserFactory = SAXParserFactory.newInstance();
80
81      parserFactory.setFeature("http://xml.org/sax/features/namespaces", true);
82      parserFactory.setFeature("http://xml.org/sax/features/namespace-prefixes",
          false);
83      //parserFactory.setFeature("http://xml.org/sax/features/unicode-normalization
          -checking", true);
84      //parserFactory.setFeature("http://xml.org/sax/features/validation", true);
85      parser2 = parserFactory.newSAXParser();
86      parser2.getXMLReader().setContentHandler(this);
87      parser2.getXMLReader().setDTDHandler(this);
88      parser2.getXMLReader().setErrorHandler(this);
89      parser2.getXMLReader().setEntityResolver(this);
90
91      } catch (SAXNotRecognizedException e) {
92        e.printStackTrace();
93      } catch (SAXNotSupportedException e) {
94        e.printStackTrace();
95      } catch (ParserConfigurationException e) {
96        e.printStackTrace();
97      } catch (SAXException e) {
98        e.printStackTrace();
99      }
100
101   }
102
103   /*
104   *Metode som ble kalt for å parse den initielle
105   *2.6GB XML protein fila
106   */
107   public void parseProteinNamesXMLFile()
108   {
109     try{
110     parser2.parse(uniref100, this);
111     }
112     catch(Exception e)
113     {
114       e.printStackTrace();
115     }
116   }
117
```

```
118   /*
119   *Metoden som kalles fra Main som returnerer
120   *pubmedIDene som en ArrayList
121   */
122   public ArrayList returnPubMedIDs(InputStream is)
123   {
124     try{
125       parser2.parse(is,this);
126     }catch(Exception e)
127     {
128       e.printStackTrace();
129     }
130
131     return pmidList;
132   }
133
134   /*
135   *Metoden som kalles fra Main og som returnerer
136   *abstractene som en ArrayList
137   */
138   public ArrayList returnAbstract(InputStream is)
139   {
140     try{
141       parser2.parse(is,this);
142     }
143     catch(Exception e)
144     {
145     System.out.println("exception "+e.toString());
146     e.printStackTrace();
147     }
148
149     return abstractTextList;
150
151   }
152
153   /*Standardmetode som kalles hver gang en XML
154   *startnode møtes
155   */
156   public void startDocument()
157   {
158
159     //System.out.println("Starten av dokumentet");
160
161   }
162
163   int number=0;
164
165   /*
166   *Standardmetode som kalles hver gang starten
167   *på et element inntreffer
168   */
169   public void startElement(String uri, String localName, String qName, Attributes
        attributes)throws SAXException
170   {
171     try{
172       //Id-elementet i eSearchResults.xml
173       if(localName == "Id")
174       {
175         counter = 1;
176         //System.out.println("pmid");
177       }
178
```

```
179         //name-elementet i uniref100.xml
180         else if(localName == "name")
181         {
182           counter = 3;
183         }
184
185         //AbstractText-elementet i eFetchResults.xml
186         else if(localName == "AbstractText")
187         {
188           //abstractFinished = false;
189           number++;
190           //abstractTextElementFinished = true;
191           counter = 2;
192           //System.out.println("AbstractText begins: ");
193         }
194         else
195         {
196           counter=0;
197         }
198       }catch(Exception e)
199       {
200         e.printStackTrace();
201       }
202     }
203
204     /*
205     *Standardmetode som kalles hver gang det
206     *finnes characters inne i et XML element
207     */
208     public void characters(char[] ch, int start, int length) throws SAXException
209     {
210       //Id-elementet i eSearchResults.xml
211       if(counter == 1)
212       {
213         try{
214
215           for(int i = start; i <= start+length-1; i++)
216           {
217             sb.append(ch[i]);
218           }
219           String pubmedid = sb.toString();
220           pmidList.add(pubmedid);
221           sb.delete(0,sb.length());
222           counter = 0;
223         }
224         //Forsøke å fange en Malformed UTF-8 char SAXParseException
225         catch(Exception e)
226         {
227           e.printStackTrace();
228           counter = 0;
229         }
230       }
231       //Abstract-elementet i eFetchResults.xml
232       else if(counter == 2)
233       {
234         String s = new String(ch,start,length);
235         sb.append(s);
236       }
237
238       else if(counter == 3)
239       {
240           counter = 0;
```

```
241       }
242
243     }
244
245     /*
246     *Standardmetode som kalles hver gang man
247     *når en slutt-tag i et XML element
248     */
249     public void endElement(String uri, String localName, String qName) throws
          SAXException
250     {
251       if(localName == "AbstractText")
252       {
253         String t = sb.toString();
254         abstractTextList.add(t);
255         sb.delete(0,sb.length());
256       }
257     }
258
259     public void setDocumentLocator (Locator locator)
260        {
261           this.locator = locator;
262        }
263
264     public InputSource resolveEntity (String publicId, String systemId) throws
          SAXException
265     {
266       return null;
267     }
268
269     public void error(SAXParseException e) throws SAXException
270     {
271       System.out.println("Super1feil"+e.toString());
272     }
273
274     public void fatalError(SAXParseException e) throws SAXException
275     {
276       System.out.println("Super2feil"+e.toString());
277     }
278
279     public void warning(SAXParseException e) throws SAXException
280     {
281       System.out.println("Super3feil"+e.toString());
282     }
283
284     /* (non-Javadoc)
285      * @see javax.xml.parsers.SAXParserFactory#getFeature(java.lang.String)
286      */
287     public boolean getFeature(String arg0) throws ParserConfigurationException,
          SAXNotRecognizedException, SAXNotSupportedException
288     {
289       return false;
290     }
291
292     /* (non-Javadoc)
293      * @see javax.xml.parsers.SAXParserFactory#newSAXParser()
294      */
295     public SAXParser newSAXParser() throws ParserConfigurationException,
          SAXException
296     {
297       return null;
298     }
```

```
299
300     /* (non-Javadoc)
301      * @see javax.xml.parsers.SAXParserFactory#setFeature(java.lang.String, boolean
          )
302      */
303     public void setFeature(String arg0, boolean arg1) throws
          ParserConfigurationException, SAXNotRecognizedException,
          SAXNotSupportedException
304     {
305
306     }
307 }
```

Tokenize.java

```
1  package master;
2
3  import java.io.BufferedReader;
4  import java.io.BufferedWriter;
5  import java.io.File;
6  import java.io.FileReader;
7  import java.io.FileWriter;
8  import java.io.PrintWriter;
9  import java.text.StringCharacterIterator;
10 import java.util.ArrayList;
11 import java.util.regex.Matcher;
12 import java.util.regex.Pattern;
13
14 /*
15  *
16  *Denne klassen tokenizer teksten.
17  */
18
19 /**
20  *@author skuland
21  *
22  *
23  */
24 public class Tokenize {
25
26   private String pmid, a;
27   private String[] tokens, tok;
28   private ArrayList abstractList, tokenizedAbstractsList, alist, toStopWords;
29   private ArrayList proteinCandidates, geneCandidates;
30   private File testread, testwrite;
31   private FileReader fr;
32   private BufferedReader br;
33   private FileWriter fw;
34   private BufferedWriter bw;
35   private PrintWriter pw, print;
36
37   /*
38   *Konstruktoren.
39   */
40   public Tokenize(String pmid, ArrayList abstractList, GUI g, boolean useStemming
       , boolean removeNumbers)
41   {
42     this.pmid = pmid;
43     this.abstractList = abstractList;
44     alist = new ArrayList();
45     tokenizedAbstractsList = new ArrayList();
46     System.out.println(pmid);
47
48     tokenizedAbstractsList = tokenize(abstractList);
49
50     PreProcessing pp = new PreProcessing(pmid, tokenizedAbstractsList, g,
         useStemming, removeNumbers);
51   }
52
53   /*
54   *Denne metoden får inn ArrayList abstractList med
55   *n abstract-tekst-objekter og returnerer dem tokenized
56   */
```

```
57    private ArrayList tokenize(ArrayList abstractList)
58    {
59      //System.out.println("ABSTRACTLIST er " +abstractList.size());
60      for(int y = 0; y < abstractList.size(); y++)
61      {
62        //Abstract-teksten
63        String s = (String)abstractList.get(y);
64        tokens = s.split("\\s");
65
66        //legger alle tokens fra arrayet til en arraylist alist
67        for(int i = 0; i <tokens.length; i++)
68        {
69          alist.add(i, tokens[i]);
70        }
71        //leser gjennom alle tokene i arraylisten
72        for(int x = 0; x < alist.size(); x++)
73        {
74          //ord etterfulgt av punktum, fjerne punktum
75          Pattern p = Pattern.compile(".+\\u002E");
76          Matcher m = p.matcher(((String)alist.get(x)));
77
78          //"." fjerne rene punktum
79          Pattern punktum = Pattern.compile("\\u002E");
80          Matcher mpunktum = punktum.matcher(((String)alist.get(x)));
81
82          //kolon
83          Pattern kolon = Pattern.compile("\\u003A");
84          Matcher mkolon = kolon.matcher(((String)alist.get(x)));
85
86          //semikolon
87          Pattern semikolon = Pattern.compile("\\u003B");
88          Matcher msemikolon = semikolon.matcher(((String)alist.get(x)));
89
90          //bindestrek
91          Pattern phyphen = Pattern.compile("\\u002D");
92          Matcher mhyphen = phyphen.matcher(((String)alist.get(x)));
93
94          //komma
95          Pattern pcomma = Pattern.compile("\\u002C");
96          Matcher mcomma = pcomma.matcher(((String)alist.get(x)));
97
98          //ord etterfulgt av komma, fjerne komma
99          Pattern pa = Pattern.compile(".*\\u002C");
100         Matcher ma = pa.matcher(((String)alist.get(x)));
101
102         //Venstre og høyre parentes fjernes
103         //(HAT) , (GRP-R    , NMB-R)
104         Pattern pat = Pattern.compile("\\u0028.*\\u0029");
105         Pattern pat1 = Pattern.compile("\\u0028.*");
106         Pattern pat2 = Pattern.compile(".*\\u0029");
107
108         Matcher mat = pat.matcher(((String)alist.get(x)));
109         Matcher mat1 = pat1.matcher(((String)alist.get(x)));
110         Matcher mat2 = pat2.matcher(((String)alist.get(x)));
111
112         //klammeparanteser
113         Pattern psquarebracket1 = Pattern.compile("\\u005B.*\\u005D");
114         Pattern psquarebracket2 = Pattern.compile("\\u005B.*");
115         Pattern psquarebracket3 = Pattern.compile(".*\\u005D");
116
117         Matcher msquare1 = psquarebracket1.matcher(((String)alist.get(x)));
118         Matcher msquare2 = psquarebracket2.matcher(((String)alist.get(x)));
```

```
119         Matcher msquare3 = psquarebracket3.matcher(((String)alist.get(x)));
120
121         //For å få treff på GRP-R (GRPReceptor) fjernes -
122         //dette gjelder generelt gen-navn som BRS-3, fjern -
123         Pattern p13 = Pattern.compile("\\p{Upper}{2,4}\\u002D(\\p{Upper}){1}");
124         Matcher m13 = p13.matcher(((String)alist.get(x)));
125
126         //uttrykk som (-1)), altså flere non-whitespace charact. før/etter et
              parantesuttrykk
127         //  ((3a)  , (3a))-    ,  d*(3a)-d   , 16).
128         Pattern parantheses1 = Pattern.compile("(\\S+\\u0028.*\\u0029)");
129         Pattern parantheses2 = Pattern.compile("\\u0028.*\\u0029\\S+");
130         Pattern parantheses3 = Pattern.compile("(\\S+\\u0028.*\\u0029\\S+)");
131         Pattern parantheses4 = Pattern.compile(".+\\u0029\\S+");
132
133         Matcher mparantheses1 = parantheses1.matcher(((String)alist.get(x)));
134         Matcher mparantheses2 = parantheses2.matcher(((String)alist.get(x)));
135         Matcher mparantheses3 = parantheses3.matcher(((String)alist.get(x)));
136         Matcher mparantheses4 = parantheses4.matcher(((String)alist.get(x)));
137
138         // eks. angina:   120;    kolon, semikolon etter ord
139         Pattern pangina = Pattern.compile(".+[\\u003A|\\u003B]+");
140         Matcher mangina = pangina.matcher(((String)alist.get(x)));
141
142         boolean hasMatched = false;
143
144         //Fjerner tomme elementer og rene punktum, kolon, semikolon, bindestrek -
              innslag
145         if((((String)alist.get(x)).length()== 0)  || mpunktum.matches() || mkolon
              .matches() || msemikolon.matches() || mhyphen.matches())
146         {
147           alist.remove(x);
148           x--;
149         }
150
151         else if(msquare1.matches())
152         {
153           //System.out.println((String)alist.get(x));
154           String newstring = ((String)alist.get(x)).substring(1, ((String)alist.
                get(x)).length()-1);
155           alist.remove(x);
156           alist.add(x, newstring);
157           x--;
158         }
159
160         else if(msquare2.matches())
161         {
162           //System.out.println("jlooo"+(String)alist.get(x));
163           String newstring = ((String)alist.get(x)).substring(1, ((String)alist.
                get(x)).length());
164           alist.remove(x);
165           alist.add(x, newstring);
166           x--;
167         }
168
169         else if(msquare3.matches())
170         {
171           //System.out.println("jlooo"+(String)alist.get(x));
172           String newstring = ((String)alist.get(x)).substring(0, ((String)alist.
                get(x)).length()-1);
173           alist.remove(x);
174           alist.add(x, newstring);
```

```
175           x--;
176         }
177
178         else if(m13.matches())
179         {
180           //System.out.println("GRS-3 treff her");
181           //System.out.println((String)alist.get(x));
182           String newstring = ((String)alist.get(x)).replaceAll("\\u002D", "");
183           alist.remove(x);
184           alist.add(x, newstring);
185           x--;
186         }
187
188         else if(mparantheses1.matches() || mparantheses2.matches() ||
              mparantheses3.matches())
189         {
190           //leser stringen og finner start og sluttindeks, skriver substringen
                til ny string
191           String par = (String)alist.get(x);
192           StringCharacterIterator sci = new StringCharacterIterator(par, 0, par.
                length(), 0);
193           int beginIndex=0;
194           int endIndex=0;
195
196           while(sci.getIndex() < sci.getEndIndex()){
197
198             char c = sci.current();
199             char n = sci.next();
200
201             if(c== '(' && n != '(' )
202             {
203               beginIndex = sci.getIndex();
204             }
205             else if(c != ')' && n == ')')
206             {
207               endIndex = sci.getIndex();
208             }
209
210           }
211
212           String paranthesesGone = par.substring(beginIndex, endIndex);
213
214           alist.remove(x);
215           alist.add(x, paranthesesGone);
216           x--;
217         }
218         else if(mparantheses4.matches())
219         {
220           String par = (String)alist.get(x);
221           StringCharacterIterator sci = new StringCharacterIterator(par, 0, par.
                length(), 0);
222           int beginIndex=0;
223           int endIndex=0;
224           while(sci.getIndex() < sci.getEndIndex())
225           {
226             char c = sci.current();
227             char n = sci.next();
228             if(c != ')' && n == ')')
229             {
230               endIndex = sci.getIndex();
231             }
232           }
```

```
233             String str = par.substring(beginIndex, endIndex);
234             alist.remove(x);
235             alist.add(x, str);
236             x--;
237           }
238           else if(mat.matches())
239           {
240             hasMatched = true;
241             String oldstring = ((String)alist.get(x));
242             String newstring = oldstring.substring(1, oldstring.length()-1);
243             alist.remove(x);
244             alist.add(x, newstring);
245             x--;
246           }
247           else if(mat1.matches() && !hasMatched)
248           {
249             hasMatched = true;
250             String oldstring = ((String)alist.get(x));
251             String newstring = oldstring.substring(1, oldstring.length());
252             alist.remove(x);
253             alist.add(x, newstring);
254             x--;
255           }
256           else if(mat2.matches() && !hasMatched)
257           {
258             hasMatched = true;
259             String oldstring = ((String)alist.get(x));
260             String newstring = oldstring.substring(0, oldstring.length()-1);
261             alist.remove(x);
262             alist.add(x, newstring);
263             x--;
264           }
265           else if(m.matches())
266           {
267             String oldstring = ((String)alist.get(x));
268             String newstring = oldstring.substring(0, oldstring.length()-1);
269             alist.remove(x);
270             alist.add(x, newstring);
271             x--;
272           }
273           else if(ma.matches())
274           {
275             String oldstring = ((String)alist.get(x));
276             String newstring = oldstring.substring(0, oldstring.length()-1);
277             alist.remove(x);
278             alist.add(x, newstring);
279             x--;
280           }
281           else if(mangina.matches())
282           {
283             String oldstring = ((String)alist.get(x));
284             String newstring = oldstring.substring(0, oldstring.length()-1);
285             alist.remove(x);
286             alist.add(x, newstring);
287             x--;
288           }
289         }
290         //alist skal her inneholde et abstract som er blitt tokenized
291         //blir cloned til blist som legges til tokenizedAbstractsList
292         ArrayList blist = (ArrayList)alist.clone();
293         tokenizedAbstractsList.add(blist);
294         alist.clear();
```

```
295       }
296       return tokenizedAbstractsList;
297     }
298
299     /*
300     *Denne metoden blir ikke kalt under kjøring av systemet
301     *fordi hybrid-løsningen ble for tidkrevende.
302     *Var meningen å prøve å finne noen mønstre i biologiske
303     *tekster som: ..activates..
304     *Rule-based part of the approach.
305     */
306     private void searchForPatterns(ArrayList a)
307     {
308       String nextSeq = "NEXTSEQ:";
309       proteinCandidates.add(nextSeq);
310       for(int x = 0; x < a.size(); x++)
311       {
312         //FACTOR, RECEPTOR, ENZYME, PROTEIN
313         //treff her gir sterk indikasjon på protein-navn
314         Pattern p1 = Pattern.compile(".*(factor).*");
315         Pattern p2 = Pattern.compile(".*(receptor).*");
316         Pattern p3 = Pattern.compile(".*(enzyme).*");
317         Pattern p4 = Pattern.compile(".*(protein).*");
318
319         Matcher m1 = p1.matcher(((String)a.get(x)));
320         Matcher m2 = p2.matcher(((String)a.get(x)));
321         Matcher m3 = p3.matcher(((String)a.get(x)));
322         Matcher m4 = p4.matcher(((String)a.get(x)));
323
324         if(m1.matches() || m2.matches() || m3.matches() || m4.matches())
325         {
326           String this_token = ((String)a.get(x));
327           String previous_token = "";
328           String next_token = "";
329           if(x > 0)
330           {
331             previous_token = ((String)a.get(x-1));
332             proteinCandidates.add(previous_token);
333           }
334           else
335           {
336             previous_token = "";
337           }
338           proteinCandidates.add(this_token);
339           if(x < a.size()-1)
340           {
341             next_token = ((String)a.get(x+1));
342             proteinCandidates.add(next_token);
343           }
344           else
345           {
346             next_token = "";
347           }
348           proteinCandidates.add(nextSeq);
349         }
350
351         //..A FAMILY OF..
352         //er interessert i ordet etter denne sekvensen
353         Pattern p5 = Pattern.compile(".*(a).*");
354         Pattern p6 = Pattern.compile(".*(family).*");
355         Pattern p7 = Pattern.compile(".*(of).*");
356
```

```
357         //må teste for nullpointer, dvs. at indeks er size-3
358         Matcher m5 = p5.matcher("");
359         Matcher m6 = p6.matcher("");
360         Matcher m7 = p7.matcher("");
361 
362         if(x<a.size()-3)
363         {
364           m5 = p5.matcher(((String)a.get(x)));
365           m6 = p6.matcher(((String)a.get(x+1)));
366           m7 = p7.matcher(((String)a.get(x+2)));
367         }
368         else
369         {
370           m5 = p5.matcher("");
371           m6 = p6.matcher("");
372           m7 = p7.matcher("");
373         }
374         if(m5.matches() && m6.matches() && m7.matches())
375         {
376           String potential = (String)a.get(x+3);
377           proteinCandidates.add(potential);
378           //a.remove(x+3);
379           a.remove(x+2);
380           a.remove(x+1);
381           a.remove(x);
382           proteinCandidates.add(nextSeq);
383         }
384         //....-LIKE
385         //er interessert i dette ordet og ordet etter
386         Pattern p8 = Pattern.compile(".*\u002D(like)");
387         Matcher m8 = p8.matcher("");
388         if(x<a.size()-1)
389         {
390         m8 = p8.matcher(((String)a.get(x)));
391         }
392         else
393         {
394           m8 = p8.matcher("");
395         }
396         if(m8.matches())
397         {
398           String this_token = ((String)a.get(x));
399           String next_token = ((String)a.get(x+1));
400 
401           proteinCandidates.add(this_token);
402           proteinCandidates.add(next_token);
403           proteinCandidates.add(nextSeq);
404         }
405         //....-RELEASING
406         //er interessert i dette ordet og ordet etter
407         Pattern p9 = Pattern.compile(".*\u002D(releasing)");
408         Matcher m9 = p9.matcher("");
409         if(x<a.size()-1)
410         {
411         m9 = p9.matcher(((String)a.get(x)));
412         }
413         else
414         {
415           m9 = p9.matcher("");
416         }
417         if(m9.matches())
418         {
```

```
419           String this_token = ((String)a.get(x));
420           String next_token = ((String)a.get(x+1));
421
422           proteinCandidates.add(this_token);
423           proteinCandidates.add(next_token);
424           proteinCandidates.add(nextSeq);
425         }
426           //....-ASE, -IN
427           //er interessert i dette ordet (protein)
428           Pattern p10 = Pattern.compile(".*(ase)");
429           Pattern p11 = Pattern.compile(".*(in)");
430
431           Matcher m10 = p10.matcher("");
432           Matcher m11 = p11.matcher("");
433           if(x<a.size())
434           {
435             m10 = p10.matcher(((String)a.get(x)));
436             m11 = p11.matcher(((String)a.get(x)));
437           }
438           if(m10.matches() || m11.matches())
439           {
440             String this_token = ((String)a.get(x));
441
442             proteinCandidates.add(this_token);
443             proteinCandidates.add(nextSeq);
444           }
445           //Gen-navn, korte ord
446           //2-8bokstaver, store b. Eks: HAT, HATs
447           //vil også ha treff på navn som BRS3
448           Pattern p12 = Pattern.compile("([\\w&&[^\\p{Lower}]]{2,8})");
449           Matcher m12 = p12.matcher("");
450           if(x<a.size())
451           {
452             m12 = p12.matcher(((String)a.get(x)));
453           }
454           if(m12.matches())
455           {
456             String this_token = ((String)a.get(x));
457             geneCandidates.add(this_token);
458           }
459           //hormone..
460           //Etterfølgende ord sannsynlig protein
461           Pattern p13 = Pattern.compile("(hormone)");
462           Matcher m13 = p13.matcher("");
463           if(x<a.size()-1)
464           {
465             m13 = p13.matcher(((String)a.get(x)));
466           }
467           if(m13.matches())
468           {
469             String this_token = ((String)a.get(x));
470             String next_token = ((String)a.get(x+1));
471
472             proteinCandidates.add(next_token);
473             proteinCandidates.add(nextSeq);
474           }
475           //release of..., inhibitor of
476           //inhibited by, mutation of, (over)expression of
477           //regulator of, modification of, control of, production of
478           //enhancement of
479           //Etterfølgende ord sannsynlig protein
480           Pattern p14 = Pattern.compile("(inhibit).*");
```

```
481         Pattern p15 = Pattern.compile("(of|by)");
482         Pattern p16 = Pattern.compile("(mutation)");
483         Pattern p17 = Pattern.compile("(of)");
484         Pattern p18 = Pattern.compile("(over)?(expression)");
485         Pattern p19 = Pattern.compile("(of)");
486         Pattern p20 = Pattern.compile("(regulator)");
487         Pattern p21 = Pattern.compile("(of)");
488         Pattern p22 = Pattern.compile("(modification)");
489         Pattern p23 = Pattern.compile("(of)");
490         Pattern p24 = Pattern.compile("(control)");
491         Pattern p25 = Pattern.compile("(of)");
492         Pattern p26 = Pattern.compile("(production)");
493         Pattern p27 = Pattern.compile("(of)");
494         Pattern p28 = Pattern.compile("(enhancement)");
495         Pattern p29 = Pattern.compile("(of)");
496
497         Matcher m14 = p14.matcher("");
498         Matcher m15 = p15.matcher("");
499         Matcher m16 = p16.matcher("");
500         Matcher m17 = p17.matcher("");
501         Matcher m18 = p18.matcher("");
502         Matcher m19 = p19.matcher("");
503         Matcher m20 = p20.matcher("");
504         Matcher m21 = p21.matcher("");
505         Matcher m22 = p22.matcher("");
506         Matcher m23 = p23.matcher("");
507         Matcher m24 = p24.matcher("");
508         Matcher m25 = p25.matcher("");
509         Matcher m26 = p26.matcher("");
510         Matcher m27 = p27.matcher("");
511         Matcher m28 = p28.matcher("");
512         Matcher m29 = p29.matcher("");
513
514         if(x<a.size()-2)
515         {
516           m14 = p14.matcher(((String)a.get(x)));
517           m15 = p15.matcher(((String)a.get(x+1)));
518
519           m16 = p16.matcher(((String)a.get(x)));
520           m17 = p17.matcher(((String)a.get(x+1)));
521
522           m18 = p18.matcher(((String)a.get(x)));
523           m19 = p19.matcher(((String)a.get(x+1)));
524
525           m20 = p20.matcher(((String)a.get(x)));
526           m21 = p21.matcher(((String)a.get(x+1)));
527
528           m22 = p22.matcher(((String)a.get(x)));
529           m23 = p23.matcher(((String)a.get(x+1)));
530
531           m24 = p24.matcher(((String)a.get(x)));
532           m25 = p25.matcher(((String)a.get(x+1)));
533
534           m26 = p26.matcher(((String)a.get(x)));
535           m27 = p27.matcher(((String)a.get(x+1)));
536
537           m28 = p28.matcher(((String)a.get(x)));
538           m29 = p29.matcher(((String)a.get(x+1)));
539         }
540         if((m14.matches() && m15.matches()) || (m16.matches() && m17.matches())
              || (m18.matches() && m19.matches()) || (m20.matches() && m21.matches())
               || (m22.matches() && m23.matches()) || (m24.matches() && m25.matches()
```

```
            ) || (m26.matches() && m27.matches()) || (m28.matches() && m29.matches
            ()))
541         {
542           String proteincandidate = ((String)a.get(x+2));
543
544           proteinCandidates.add(proteincandidate);
545           proteinCandidates.add(nextSeq);
546         }
547       }
548     }
549   }
```

PreProcessing.java

```
1  package master;
2
3  import java.io.BufferedWriter;
4  import java.io.File;
5  import java.io.FileWriter;
6  import java.io.PrintWriter;
7  import java.util.ArrayList;
8  import java.util.HashMap;
9  import java.util.Iterator;
10 import java.util.Set;
11 import java.util.regex.PatternSyntaxException;
12
13 /*
14  *Denne klassen skal preprosessere abstractene før de
15  *blir sendt for lookup. De har da allerede blitt
16  *tokenized, og skal gjennom fjerning av stoppord og
17  *eventuelt stemmes.
18  *De preprosesserte ferdige abstractene skal sendes til LookUp.
19  *
20  */
21
22 /*
23  *@author skuland
24  */
25
26 public class PreProcessing {
27
28   private String pmid;
29   private ArrayList tokenizedAbstractsList, removedStopwords, stemmedList,
       finalList;
30   private File frequency;
31   private HashMap tokensCounted;
32   private ArrayList sortedKeys, sortedFrequencies, tokFreq;
33   private PrintWriter frequencyPrintWriter, print, print2, print3;
34   private FileWriter filewriter;
35   private BufferedWriter bufferedwriter;
36
37   /*
38   *Konstruktoren. Den får inn tokenized abstracts.
39   *Starter opp LookUp-klassen.
40   */
41   public PreProcessing(String pmid, ArrayList tokenizedAbstractsList, GUI gui,
       boolean useStemming, boolean removeNumbers)
42   {
43     this.pmid = pmid;
44     this.tokenizedAbstractsList = tokenizedAbstractsList;
45     tokensCounted = new HashMap();
46     sortedKeys = new ArrayList();
47     sortedFrequencies = new ArrayList();
48     tokFreq = new ArrayList();
49     removedStopwords = removeStopwordsFromAbstracts(tokenizedAbstractsList);
50
51     //Stemmer bare abstractene dersom brukeren ønsket dette i GUI
52     if(useStemming)
53     {
54       stemmedList = stemAbstracts(removedStopwords);
55     }
56       else
```

```
57      {
58      stemmedList = removedStopwords;
59      }
60
61      //finalList inneholder etter dette steget alle abstractene ferdig stemmet og
          stoppordfjernet
62      finalList = removeStopwordsFromAbstracts(stemmedList);
63      ArrayList z = finalList;
64
65      if(removeNumbers)
66      {
67        finalList = removePurelyNumericalEntries(finalList);
68      }
69      else
70      {
71        finalList = z;
72      }
73
74      ArrayList al = sortTokensByFrequency(finalList);
75
76      LookUp lu = new LookUp(pmid, finalList, gui, al, this, useStemming);
77    }
78
79    /*
80    *Metode som sender abstracts til Stopword-klassen
81    *for å fjerne stoppord
82    */
83    private ArrayList removeStopwordsFromAbstracts(ArrayList tokenizedAbstractsList
        )
84    {
85      double e = 0;
86      double average = 0;
87      Stopwords stop = new Stopwords();
88      for(int k = 0; k<tokenizedAbstractsList.size();k++){
89          ArrayList a = stop.toLowerCase((ArrayList)tokenizedAbstractsList.get(k));
90          double d = stop.quote;
91          e = e + d;
92          ArrayList b = stop.removeStopwords(a);
93          tokenizedAbstractsList.remove(k);
94          tokenizedAbstractsList.add(k,b);
95      }
96      average = e / tokenizedAbstractsList.size();
97      System.out.println("Gjennomsnittlig antall prosent fjernet fra abstractene: "
          +average);
98
99      for(int z = 0; z<2;z++)
100     {
101       ArrayList ar = (ArrayList)tokenizedAbstractsList.get(z);
102     }
103
104     return tokenizedAbstractsList;
105   }
106
107   /*
108   *Metode som skal sende abstractene til Stemmer-klassen
109   *og returnere stemmed-versjon av abstractene.
110   */
111   private ArrayList stemAbstracts(ArrayList alist)
112   {
113     Stemmer st = new Stemmer();
114
115     //for alle abstract
```

```
116     for(int k =0; k<alist.size();k++){
117
118       ArrayList xy = (ArrayList)alist.get(k);
119
120       //for alle tokens i hvert abstract
121       for(int i = 0; i < xy.size();i++)
122       {
123         int b = ((String)xy.get(i)).length();
124
125         for(int a = 0; a < b; a++){
126           st.add(((String)xy.get(i)).charAt(a));
127         }
128         st.stem();
129         String stemmedToken = st.toString();
130         if(!((String)xy.get(i)).equals(stemmedToken))
131         {
132           xy.remove(i);
133           xy.add(i,stemmedToken);
134         }
135       }
136       alist.remove(k);
137       alist.add(k,xy);
138     }
139     return alist;
140   }
141
142   /*
143   *Metode som fjerner numeriske innslag. Valgbar.
144   */
145   private ArrayList removePurelyNumericalEntries(ArrayList finalList)
146   {
147     int county = 0;
148     for(int z = 0; z < finalList.size();z++){
149       ArrayList al = (ArrayList)finalList.get(z);
150       try{
151         for(int a = 0; a<al.size();a++)
152         {
153           if (( ((String)al.get(a)).matches("[\\d]+") ) &&  (! ((String)al.get(a)
              ).matches("[\\D]+") ))
154           {
155             county++;
156             al.remove(a);
157             a--;
158           }
159           //fjerne desimaltall også
160           else if ( ((String)al.get(a)).matches("\\d*(\\u002E)\\d*") )
161           {
162             county++;
163             al.remove(a);
164             a--;
165           }
166
167         }
168       }catch(PatternSyntaxException e)
169       {
170         e.printStackTrace();
171       }
172       finalList.remove(z);
173       finalList.add(z, al);
174     }
175
176     return finalList;
```

```
177   }
178
179   /*
180   *Denne metoden får inn en arraylist med alle abstractene.
181   *Den går så gjennom et og et, og sorterer de etter
182   *frekvens ved quicksort-algoritmen. Den returnerer arraylisten
183   *tokFreq som inneholder samtlige abstract sortert.
184   */
185   public ArrayList sortTokensByFrequency(ArrayList finalList)
186   {
187     for(int e = 0; e < finalList.size(); e++){
188
189       ArrayList list = (ArrayList)finalList.get(e);
190
191       //legger til nøkkel-frekvens mappinger til HashMap'et tokensCounted
192       for(int i = 0; i < list.size();i++)
193       {
194         String m = ((String)list.get(i));
195
196         if(tokensCounted.containsKey(m))
197         {
198           Integer a = (Integer)( (tokensCounted.get(m)) );
199           int b = a.intValue();
200           b++;
201           Integer c = new Integer(b);
202           tokensCounted.put(m, c);
203         }
204         else
205         {
206           int x = 1;
207           Integer y = new Integer(x);
208           tokensCounted.put(m, y);
209         }
210       }
211       Set s = tokensCounted.keySet();
212       Iterator it = s.iterator();
213       int n = 0;
214       int[] frequency = new int[tokensCounted.size()];
215
216       while(it.hasNext())
217       {
218
219         String m = ((String)it.next());
220         Integer i = (Integer)tokensCounted.get(m);
221         int a = i.intValue();
222
223         frequency[n] = a;
224         n++;
225       }
226
227       //Kaller quicksort på frequency-arrayet
228       try{
229       sort(frequency);
230       }
231       catch(Exception ex){ex.printStackTrace();}
232
233       //må så iterere gjennom hele arrayet frequency og mappe frekvenser mot
          stringene
234       Set newset = tokensCounted.keySet();
235       Iterator newit = newset.iterator();
236
237       //Fyller opp ArrayListene med tomme objekter
```

```
238         Integer io = new Integer(0);
239         String xyz = "";
240         for(int xy = 0; xy < newset.size(); xy++){
241         sortedKeys.add(xyz);
242         sortedFrequencies.add(io);
243         }
244
245         int test = 0;
246         boolean found = false;
247         //For alle mappinger i HashMap tokensCounted
248         while(newit.hasNext())
249         {
250           test++;
251           found = false;
252           String m = ((String)newit.next());
253           //System.out.println("String m er: " +m);
254           Integer i = (Integer)tokensCounted.get(m);
255           //System.out.println("Verdi er: "+i);
256           int a = i.intValue();
257           if(!found){
258             //itererer over frequency-arrayet
259             for(int k = 0; k < frequency.length;k++)
260             {
261               if(!found){
262                 //System.out.println("k er: "+k);
263                 //Hvis verdi a fra tokensCounted er lik verdi i frequency-array
264                 if(a == frequency[k])
265                 {
266                   sortedKeys.set(k,m);
267                   sortedFrequencies.set(k,i);
268                   frequency[k]=0;
269                   found = true;
270                 }
271               }
272             }
273           }
274         }
275
276         //nuller ut arrayet for hver iterasjon e
277         frequency = null;
278
279
280         //Har nå to ArrayLists sortedKeys og sortedFrequencies som inneholder
281         //sorterte nøkler etter frekvens for abstract nr. e
282         //Putter så disse to ArrayListene til et ArrayList tokFreq
283
284         ArrayList sortedKeysClone = (ArrayList)sortedKeys.clone();
285         ArrayList sortedFrequenciesClone = (ArrayList)sortedFrequencies.clone();
286
287         tokFreq.add(sortedKeysClone);
288         tokFreq.add(sortedFrequenciesClone);
289
290         sortedKeys.clear();
291         sortedFrequencies.clear();
292         tokensCounted.clear();
293
294       }
295       System.out.println("Tokfreq size: "+tokFreq.size());
296
297       ArrayList tokFreqClone = (ArrayList)tokFreq.clone();
298       tokFreq.clear();
299
```

```
300       return tokFreqClone;
301
302     }
303
304     private void testWrite(String s)
305     {
306       try{
307       }
308       catch(Exception e){
309         e.printStackTrace();
310       }
311     }
312
313     private void createFile()
314     {
315       try{
316         frequencyPrintWriter = new PrintWriter(new BufferedWriter(new FileWriter(
            new File("frequency.txt"), false)));
317
318       }
319       catch(Exception e){
320         e.printStackTrace();
321       }
322     }
323
324
325        /** This is a generic version of C.A.R Hoare's Quick Sort
326        * algorithm.  This will handle arrays that are already
327        * sorted, and arrays with duplicate keys.<BR>
328        *
329        * If you think of a one dimensional array as going from
330        * the lowest index on the left to the highest index on the right
331        * then the parameters to this function are lowest index or
332        * left and highest index or right.  The first time you call
333        * this function it will be with the parameters 0, a.length - 1.
334        *
335        * @param a         an integer array
336        * @param lo0     left boundary of array partition
337        * @param hi0     right boundary of array partition
338        */
339        private void quickSort(int a[], int l, int r) throws Exception
340        {
341             int M = 4;
342             int i;
343             int j;
344             int v;
345
346             if ((r-l)>M)
347             {
348                     i = (r+l)/2;
349                     if (a[l]>a[i]) swap(a,l,i);     // Tri-Median Methode!
350                     if (a[l]>a[r]) swap(a,l,r);
351                     if (a[i]>a[r]) swap(a,i,r);
352
353                     j = r-1;
354                     swap(a,i,j);
355                     i = l;
356                     v = a[j];
357                     for(;;)
358                     {
359                             while(a[++i]<v);
360                             while(a[--j]>v);
```

```
361                             if (j<i) break;
362                             swap (a,i,j);
363
364
365                     }
366                     swap(a,i,r-1);
367
368                     quickSort(a,l,j);
369                     quickSort(a,i+1,r);
370             }
371         }
372
373         private void swap(int a[], int i, int j)
374         {
375                 int T;
376                 T = a[i];
377                 a[i] = a[j];
378                 a[j] = T;
379         }
380
381         private void insertionSort(int a[], int lo0, int hi0) throws Exception
382         {
383                 int i;
384                 int j;
385                 int v;
386
387                 for (i=lo0+1;i<=hi0;i++)
388                 {
389                         v = a[i];
390                         j=i;
391                         while ((j>lo0) && (a[j-1]>v))
392                         {
393                                 a[j] = a[j-1];
394                                 j--;
395                         }
396                         a[j] = v;
397                 }
398         }
399
400         public void sort(int a[]) throws Exception
401         {
402                 quickSort(a, 0, a.length - 1);
403                 insertionSort(a,0,a.length-1);
404         }
405 }
```

Stopwords.java

```
1  package master;
2
3  import java.util.ArrayList;
4
5  /*
6   * Denne klassen inneholder logikk for å fjerne
7   * stoppord fra abstractene. Stoppordlista er hentet
8   * fra Medline.
9   */
10
11 /**
12  * @author skuland
13  *
14  */
15
16 public class Stopwords {
17
18   private String pmid;
19   private ArrayList tokens;
20   private String[] stopwords = {"a", "about", "again", "all", "almost", "also", "
       although", "always", "among", "an", "and", "another", "any", "are", "as", "at
       ", "be", "because","been", "before", "being", "between", "both", "but", "by",
        "can", "could", "did", "do", "does", "done", "due", "during", "each", "
       either", "enough", "especially", "etc", "for", "found", "from", "further", "
       had", "has", "have", "having", "here", "how", "however", "i", "if", "in", "
       into", "is", "it", "its", "itself", "just", "kg", "km", "made", "mainly", "
       make", "may", "mg", "might", "ml", "mm", "most", "mostly", "must", "nearly",
       "neither", "no", "nor", "obtained", "of", "often", "on", "our", "overall", "
       perhaps", "quite", "rather", "really", "regarding", "seem", "seen", "several"
       , "should", "show", "showed", "shown", "shows", "significantly", "since", "so
       ", "some", "such", "than", "that", "the", "their", "theirs", "them", "then",
       "there", "therefore", "these", "they", "this", "those", "through", "thus", "
       to", "upon", "use", "used", "using", "various", "very", "was", "we", "were",
       "what", "when", "which", "while", "with", "within", "without", "would", "
       abstract"};
21   private ArrayList tokenList;
22   private ArrayList finalTokenList;
23   public double quote;
24
25   /*
26   *Konstruktoren.
27   */
28   public Stopwords()
29   {
30
31   }
32
33   /*
34   *Metoden som kalles fra andre klasser for å fjerne stoppord.
35   */
36   public ArrayList removeStopwords(ArrayList tokenList)
37   {
38     int g = tokenList.size();
39     int count = 0;
40     boolean foundStopword = false;
41     for(int i = 0; i < tokenList.size(); i++)
42     {
43       foundStopword = false;
```

```
44        if(!foundStopword){
45        for(int x = 0; x < stopwords.length; x++)
46        {
47          if(!foundStopword){
48          //må passe på indeksen her, dersom den er null, kan den bli negativ
49          if(((String)(tokenList.get(i))).equals(stopwords[x]))
50          {
51            count++;
52
53            if(i > 0)
54            {
55            tokenList.remove(i);
56            i--;
57            foundStopword = true;
58            }
59            else
60            {
61              tokenList.remove(i);
62              i = -1;
63              foundStopword = true;
64            }
65          }
66          }
67        }
68        }
69      }
70      quote = (double)count/g;
71      return tokenList;
72
73    }
74
75    /*
76    *Metode som gjør et string array til en ArrayList
77    */
78    public ArrayList toArrayList(String[] tokens)
79    {
80      for(int i = 0; i < tokens.length; i++)
81      {
82        tokenList.add(tokens[i]);
83      }
84      return tokenList;
85    }
86
87    /*
88    *Metode som setter en arraylist med abstracts til lowercase
89    */
90    public ArrayList toLowerCase(ArrayList alist)
91    {
92      for(int x = 0; x < alist.size(); x++)
93      {
94        String m = ((String)alist.get(x)).toLowerCase();
95        alist.remove(x);
96        alist.add(x,m);
97
98      }
99      return alist;
100   }
101 }
```

Stemmer.java

```
1  package master;
2
3
4  /*
5   *  Klasse som implementer Porter stemming algoritmen.
6   */
7
8  /**
9   *  @author skuland
10  *
11  */
12
13 /*
14  *  Stemmer, implementing the Porter Stemming Algorithm
15  *
16  *  The Stemmer class transforms a word into its root form.  The input
17  *  word can be provided a character at time (by calling add()), or at once
18  *  by calling one of the various stem(something) methods.
19  */
20
21 public class Stemmer{
22
23   private char[] b;
24   private int i,     /* offset into b */
25               i_end, /* offset to end of stemmed word */
26               j, k;
27   private static final int INC = 50;
28                       /* unit of size whereby b is increased */
29   public Stemmer()
30      {
31       b = new char[INC];
32       i = 0;
33       i_end = 0;
34      }
35
36      /**
37       * Add a character to the word being stemmed.  When you are finished
38       * adding characters, you can call stem(void) to stem the word.
39       */
40      public void add(char ch)
41      {  if (i == b.length)
42         {  char[] new_b = new char[i+INC];
43            for (int c = 0; c < i; c++) new_b[c] = b[c];
44            b = new_b;
45         }
46         b[i++] = ch;
47      }
48
49
50      /** Adds wLen characters to the word being stemmed contained in a portion
51       * of a char[] array. This is like repeated calls of add(char ch), but
52       * faster.
53       */
54      public void add(char[] w, int wLen)
55      {  if (i+wLen >= b.length)
56         {  char[] new_b = new char[i+wLen+INC];
57            for (int c = 0; c < i; c++) new_b[c] = b[c];
58            b = new_b;
```

```
59          }
60          for (int c = 0; c < wLen; c++) b[i++] = w[c];
61      }
62
63      /**
64       * After a word has been stemmed, it can be retrieved by toString(),
65       * or a reference to the internal buffer can be retrieved by getResultBuffer
66       * and getResultLength (which is generally more efficient.)
67       */
68      public String toString() { return new String(b,0,i_end); }
69
70      /**
71       * Returns the length of the word resulting from the stemming process.
72       */
73      public int getResultLength() { return i_end; }
74
75      /**
76       * Returns a reference to a character buffer containing the results of
77       * the stemming process.  You also need to consult getResultLength()
78       * to determine the length of the result.
79       */
80      public char[] getResultBuffer() { return b; }
81
82      /* cons(i) is true <=> b[i] is a consonant. */
83
84      private final boolean cons(int i)
85      {  switch (b[i])
86         {  case 'a': case 'e': case 'i': case 'o': case 'u': return false;
87            case 'y': return (i==0) ? true : !cons(i-1);
88            default: return true;
89         }
90      }
91
92      /* m() measures the number of consonant sequences between 0 and j. if c is
93         a consonant sequence and v a vowel sequence, and <..> indicates arbitrary
94         presence,
95
96            <c><v>       gives 0
97            <c>vc<v>     gives 1
98            <c>vcvc<v>   gives 2
99            <c>vcvcvc<v> gives 3
100           ....
101     */
102     private final int m()
103     {  int n = 0;
104        int i = 0;
105        while(true)
106        {  if (i > j) return n;
107           if (! cons(i)) break; i++;
108        }
109        i++;
110        while(true)
111        {  while(true)
112           {  if (i > j) return n;
113                 if (cons(i)) break;
114                 i++;
115           }
116           i++;
117           n++;
118           while(true)
119           {  if (i > j) return n;
120              if (! cons(i)) break;
```

```
121                 i++;
122               }
123               i++;
124             }
125       }
126
127       /* vowelinstem() is true <=> 0,...j contains a vowel */
128       private final boolean vowelinstem()
129       {  int i; for (i = 0; i <= j; i++) if (! cons(i)) return true;
130          return false;
131       }
132
133       /* doublec(j) is true <=> j,(j-1) contain a double consonant. */
134       private final boolean doublec(int j)
135       {  if (j < 1) return false;
136          if (b[j] != b[j-1]) return false;
137          return cons(j);
138       }
139
140       /* cvc(i) is true <=> i-2,i-1,i has the form consonant - vowel - consonant
141          and also if the second c is not w,x or y. this is used when trying to
142          restore an e at the end of a short word. e.g.
143
144             cav(e), lov(e), hop(e), crim(e), but
145             snow, box, tray.
146
147       */
148       private final boolean cvc(int i)
149       {  if (i < 2 || !cons(i) || cons(i-1) || !cons(i-2)) return false;
150          {  int ch = b[i];
151             if (ch == 'w' || ch == 'x' || ch == 'y') return false;
152          }
153          return true;
154       }
155
156       private final boolean ends(String s)
157       {  int l = s.length();
158          int o = k-l+1;
159          if (o < 0) return false;
160          for (int i = 0; i < l; i++) if (b[o+i] != s.charAt(i)) return false;
161          j = k-l;
162          return true;
163       }
164
165       /* setto(s) sets (j+1),...k to the characters in the string s, readjusting
166          k. */
167       private final void setto(String s)
168       {  int l = s.length();
169          int o = j+1;
170          for (int i = 0; i < l; i++) b[o+i] = s.charAt(i);
171          k = j+l;
172       }
173
174       /* r(s) is used further down. */
175       private final void r(String s) { if (m() > 0) setto(s); }
176
177       /* step1() gets rid of plurals and -ed or -ing. e.g.
178
179             caresses  ->  caress
180             ponies    ->  poni
181             ties      ->  ti
182             caress    ->  caress
```

```
183             cats      ->  cat
184
185             feed      ->  feed
186             agreed    ->  agree
187             disabled  ->  disable
188
189             matting   ->  mat
190             mating    ->  mate
191             meeting   ->  meet
192             milling   ->  mill
193             messing   ->  mess
194
195             meetings  ->  meet
196
197         */
198         private final void step1()
199         {  if (b[k] == 's')
200            {  if (ends("sses")) k -= 2; else
201               if (ends("ies")) setto("i"); else
202               if (b[k-1] != 's') k--;
203            }
204            if (ends("eed")) { if (m() > 0) k--; } else
205            if ((ends("ed") || ends("ing")) && vowelinstem())
206            {  k = j;
207               if (ends("at")) setto("ate"); else
208               if (ends("bl")) setto("ble"); else
209               if (ends("iz")) setto("ize"); else
210               if (doublec(k))
211               {  k--;
212                  {  int ch = b[k];
213                     if (ch == 'l' || ch == 's' || ch == 'z') k++;
214                  }
215               }
216               else if (m() == 1 && cvc(k)) setto("e");
217            }
218         }
219
220         /* step2() turns terminal y to i when there is another vowel in the stem. */
221         private final void step2() { if (ends("y") && vowelinstem()) b[k] = 'i'; }
222
223         /* step3() maps double suffices to single ones. so -ization ( = -ize plus
224            -ation) maps to -ize etc. note that the string before the suffix must
                give
225            m() > 0. */
226         private final void step3() { if (k == 0) return; /* For Bug 1 */ switch (b[k
               -1])
227         {
228             case 'a': if (ends("ational")) { r("ate"); break; }
229                       if (ends("tional")) { r("tion"); break; }
230                       break;
231             case 'c': if (ends("enci")) { r("ence"); break; }
232                       if (ends("anci")) { r("ance"); break; }
233                       break;
234             case 'e': if (ends("izer")) { r("ize"); break; }
235                       break;
236             case 'l': if (ends("bli")) { r("ble"); break; }
237                       if (ends("alli")) { r("al"); break; }
238                       if (ends("entli")) { r("ent"); break; }
239                       if (ends("eli")) { r("e"); break; }
240                       if (ends("ousli")) { r("ous"); break; }
241                       break;
242             case 'o': if (ends("ization")) { r("ize"); break; }
```

```
243                 if (ends("ation")) { r("ate"); break; }
244                 if (ends("ator")) { r("ate"); break; }
245                 break;
246             case 's': if (ends("alism")) { r("al"); break; }
247                 if (ends("iveness")) { r("ive"); break; }
248                 if (ends("fulness")) { r("ful"); break; }
249                 if (ends("ousness")) { r("ous"); break; }
250                 break;
251             case 't': if (ends("aliti")) { r("al"); break; }
252                 if (ends("iviti")) { r("ive"); break; }
253                 if (ends("biliti")) { r("ble"); break; }
254                 break;
255             case 'g': if (ends("logi")) { r("log"); break; }
256         } }
257
258         /* step4() deals with -ic-, -full, -ness etc. similar strategy to step3. */
259         private final void step4() { switch (b[k])
260         {
261             case 'e': if (ends("icate")) { r("ic"); break; }
262                 if (ends("ative")) { r(""); break; }
263                 if (ends("alize")) { r("al"); break; }
264                 break;
265             case 'i': if (ends("iciti")) { r("ic"); break; }
266                 break;
267             case 'l': if (ends("ical")) { r("ic"); break; }
268                 if (ends("ful")) { r(""); break; }
269                 break;
270             case 's': if (ends("ness")) { r(""); break; }
271                 break;
272         } }
273
274         /* step5() takes off -ant, -ence etc., in context <c>vcvc<v>. */
275         private final void step5()
276         {   if (k == 0) return; /* for Bug 1 */ switch (b[k-1])
277             {  case 'a': if (ends("al")) break; return;
278                case 'c': if (ends("ance")) break;
279                          if (ends("ence")) break; return;
280                case 'e': if (ends("er")) break; return;
281                case 'i': if (ends("ic")) break; return;
282                case 'l': if (ends("able")) break;
283                          if (ends("ible")) break; return;
284                case 'n': if (ends("ant")) break;
285                          if (ends("ement")) break;
286                          if (ends("ment")) break;
287                          /* element etc. not stripped before the m */
288                          if (ends("ent")) break; return;
289                case 'o': if (ends("ion") && j >= 0 && (b[j] == 's' || b[j] == 't'))
                              break;
290                                                  /* j >= 0 fixes Bug 2 */
291                          if (ends("ou")) break; return;
292                          /* takes care of -ous */
293                case 's': if (ends("ism")) break; return;
294                case 't': if (ends("ate")) break;
295                          if (ends("iti")) break; return;
296                case 'u': if (ends("ous")) break; return;
297                case 'v': if (ends("ive")) break; return;
298                case 'z': if (ends("ize")) break; return;
299                default: return;
300             }
301             if (m() > 1) k = j;
302         }
303
```

```
304       /* step6() removes a final -e if m() > 1. */
305       private final void step6()
306       {  j = k;
307          if (b[k] == 'e')
308          {  int a = m();
309             if (a > 1 || a == 1 && !cvc(k-1)) k--;
310          }
311          if (b[k] == 'l' && doublec(k) && m() > 1) k--;
312       }
313
314       /** Stem the word placed into the Stemmer buffer through calls to add().
315        * Returns true if the stemming process resulted in a word different
316        * from the input.  You can retrieve the result with
317        * getResultLength()/getResultBuffer() or toString().
318        */
319       public void stem()
320       {  k = i - 1;
321          if (k > 1) { step1(); step2(); step3(); step4(); step5(); step6(); }
322          i_end = k+1; i = 0;
323       }
324    }
```

Result.java

```
1  package master;
2
3  import java.util.ArrayList;
4
5  /*
6   *
7   *Denne klassen inneholder resultat arrayList med alle gen-
8   *og protein-treff sortert etter frekvens.
9   *
10  */
11
12 /**
13  *@author skuland
14  *
15  *
16  */
17 public class Result {
18
19   private ArrayList resultList;
20   private GUI gui;
21
22   /*
23   *Konstruktoren Result tar inn resultat ArrayListen og gui-pekeren
24   */
25   public Result(ArrayList resultList, GUI gui)
26   {
27     System.out.println("ER I RESULTKLASSEN");
28     this.resultList = resultList;
29     this.gui = gui;
30     displayAllBioTermsInGUIWindow(resultList);
31   }
32
33   /*
34   *Metode som setter inn resultatene i GUIen
35   */
36   public void displayAllBioTermsInGUIWindow(ArrayList resultList)
37   {
38     gui.receiveResultList(resultList);
39   }
40
41 }
```

Weight.java

```
1  package master;
2
3  import java.util.ArrayList;
4
5  /*
6   *
7   *Denne klassen beregner normalisert og invers frekvens
8   *for alle termer i alle abstracts. Deretter beregnes
9   *vekter for alle termene. De tre termene med høyest
10  *vekt sendes tilbake til LookUp.
11  *
12  *N er antall dokumenter(abstracts) i systemet.
13  *
14  */
15
16 /**
17  *@author skuland
18  *
19  *
20  */
21
22 public class Weight {
23
24    private ArrayList tokFreq;
25    private ArrayList normalizedFrequencies;
26    private ArrayList inverseFrequencies;
27    private ArrayList weightList, superWeightList;
28    private double normalizedFrequency = 0.0;
29    private double inverseDocumentFrequency = 0.0;
30    private int N;
31
32    /*
33    *Konstruktor Weight.
34    */
35    public Weight()
36    {
37      normalizedFrequencies = new ArrayList();
38      inverseFrequencies = new ArrayList();
39      weightList = new ArrayList();
40      superWeightList = new ArrayList();
41    }
42
43    /*
44    *Metode som skal beregne de 5 mest relevante biotermene
45    *(protein- eller gennavn). Den tar inn en arraylist med
46    *sorterte gentreff og vekter, samme for protein
47    */
48    public ArrayList calculateFiveMostRelevantBioTerms(ArrayList gene, ArrayList
       protein)
49    {
50      double largestWeight = 0;
51      double secondLargestWeight = 0;
52      double thirdLargestWeight = 0;
53      double fourthLargestWeight = 0;
54      double fifthLargestWeight = 0;
55      double a = 0;
56      double b = 0;
57      double c = 0;
```

```
58      double da = 0;
59
60      int filpIndex = 0;
61      int filuIndex = 0;
62      int folpIndex = 0;
63      int foluIndex = 0;
64      int tlpIndex = 0;
65      int tluIndex = 0;
66      int slpIndex = 0;
67      int sluIndex = 0;
68      int lpIndex = 0;
69      int luIndex = 0;
70
71      for(int u = 0; u < protein.size(); u++)
72      {
73        ArrayList abstractWeights = (ArrayList)protein.get(u);
74
75        for(int p = 0; p < abstractWeights.size(); p++)
76        {
77          if( ((Double)abstractWeights.get(p)).doubleValue() > fifthLargestWeight)
78          {
79            c = fifthLargestWeight;
80            fifthLargestWeight = ((Double)abstractWeights.get(p)).doubleValue();
81            filpIndex = p;
82            filuIndex = u;
83
84            if( ((Double)abstractWeights.get(p)).doubleValue() >
                fourthLargestWeight)
85            {
86              da = fourthLargestWeight;
87              fourthLargestWeight = ((Double)abstractWeights.get(p)).doubleValue();
88              folpIndex = p;
89              foluIndex = u;
90              fifthLargestWeight = c;
91
92              if( ((Double)abstractWeights.get(p)).doubleValue() >
                  thirdLargestWeight)
93              {
94                a = thirdLargestWeight;
95                thirdLargestWeight = ((Double)abstractWeights.get(p)).doubleValue()
                    ;
96                tlpIndex = p;
97                tluIndex = u;
98                fourthLargestWeight = da;
99
100               if( ((Double)abstractWeights.get(p)).doubleValue() >
                    secondLargestWeight)
101               {
102                 b = secondLargestWeight;
103                 secondLargestWeight = ((Double)abstractWeights.get(p)).
                      doubleValue();
104                 slpIndex = p;
105                 sluIndex = u;
106                 thirdLargestWeight = a;
107
108                 if( ((Double)abstractWeights.get(p)).doubleValue() >
                      largestWeight)
109                 {
110                   largestWeight = ((Double)abstractWeights.get(p)).doubleValue();
111                   lpIndex = p;
112                   luIndex = u;
113                   secondLargestWeight = b;
```

```
114                 }
115
116               }
117             }
118           }
119         }
120       }
121
122     }
123
124     //de fem mest relevant proteintermer
125     String mostRelevantTerm = (String)((ArrayList)tokFreq.get(luIndex*2)).get(
          lpIndex);
126     String secondMostRelevantTerm = (String)((ArrayList)tokFreq.get(sluIndex*2)).
          get(slpIndex);
127     String thirdMostRelevantTerm = (String)((ArrayList)tokFreq.get(tluIndex*2)).
          get(tlpIndex);
128     String fourthMostRelevantTerm = (String)((ArrayList)tokFreq.get(foluIndex*2))
          .get(folpIndex);
129     String fifthMostRelevantTerm = (String)((ArrayList)tokFreq.get(filuIndex*2)).
          get(filpIndex);
130
131     //prosesserer genArrayListen
132     for(int g = 0; g< gene.size(); g++)
133     {
134       ArrayList geneAbstractWeights = (ArrayList)protein.get(g);
135       for(int ba = 0; ba < geneAbstractWeights.size(); ba++)
136       {
137         if( ((Double)geneAbstractWeights.get(ba)).doubleValue() >  largestWeight)
138         {
139           mostRelevantTerm = (String)((ArrayList)tokFreq.get(g*2)).get(ba);
140         }
141         else if( ((Double)geneAbstractWeights.get(ba)).doubleValue() >
              secondLargestWeight)
142         {
143           secondMostRelevantTerm = (String)((ArrayList)tokFreq.get(g*2)).get(ba);
144         }
145         else if( ((Double)geneAbstractWeights.get(ba)).doubleValue() >
              thirdLargestWeight)
146         {
147           thirdMostRelevantTerm = (String)((ArrayList)tokFreq.get(g*2)).get(ba);
148         }
149         else if( ((Double)geneAbstractWeights.get(ba)).doubleValue() >
              fourthLargestWeight)
150         {
151           fourthMostRelevantTerm = (String)((ArrayList)tokFreq.get(g*2)).get(ba);
152         }
153         else if( ((Double)geneAbstractWeights.get(ba)).doubleValue() >
              fifthLargestWeight)
154         {
155           fifthMostRelevantTerm = (String)((ArrayList)tokFreq.get(g*2)).get(ba);
156         }
157       }
158     }
159
160     ArrayList relevantBioTermsList = new ArrayList();
161     relevantBioTermsList.add(mostRelevantTerm);
162     relevantBioTermsList.add(secondMostRelevantTerm);
163     relevantBioTermsList.add(thirdMostRelevantTerm);
164     relevantBioTermsList.add(fourthMostRelevantTerm);
165     relevantBioTermsList.add(fifthMostRelevantTerm);
166
```

```
167       return relevantBioTermsList;
168     }
169
170     /*
171     *Metode som beregner normalized frequency for generelle termer
172     *Dersom en term ikke opptrer i et dokument skal den normaliserte
173     *frekvensen være 0
174     */
175     public ArrayList calculateNormalizedFrequency(ArrayList sortedKeys, ArrayList
          sortedFrequencies)
176     {
177         int maxFrequency = ((Integer)(sortedFrequencies.get(sortedFrequencies.size
              ()-1))).intValue();
178
179         for(int i = 0; i < sortedKeys.size(); i++)
180         {
181           normalizedFrequency = ((double)(((Integer)sortedFrequencies.get(i)).
                intValue()) / ( (double)maxFrequency));
182           Double d = new Double(normalizedFrequency);
183           normalizedFrequencies.add(d);
184         }
185
186         ArrayList normalizedFrequenciesClone = (ArrayList)normalizedFrequencies.
              clone();
187         normalizedFrequencies.clear();
188
189         return normalizedFrequenciesClone;
190     }
191
192     /*
193     *Dette blir en metode som returnerer antall abstracts
194     *som en gitt term opptrer i. Denne variabelen kalles
195     *n_i. Tall mellom 1 og N.
196     */
197     public int returnNumberOfTermOccurences(String term)
198     {
199       boolean isFoundHere = false;
200       int occurences = 0;
201
202       //nr. 0,2,4,6,8,10,12,14,16,18
203       for(int w = 0; w < tokFreq.size(); w=w+2)
204       {
205         ArrayList sortedKeys = (ArrayList)tokFreq.get(w);
206         if(!isFoundHere){
207           for(int r = 0; r < sortedKeys.size(); r++)
208           {
209             if(!isFoundHere){
210               if(( (String)sortedKeys.get(r)).equals(term))
211               {
212                 occurences++;
213                 isFoundHere=true;
214               }
215             }
216           }
217         }
218         isFoundHere = false;
219       }
220       return occurences;
221     }
222
223     /*
224     *Metode som beregner inverse document frequency
```

```
225   *for generelle termer
226   */
227   public ArrayList calculateInverseDocumentFrequency(ArrayList sortedKeys)
228   {
229     //idf_i = log (N/n_i)
230
231     for(int i = 0; i < sortedKeys.size(); i++)
232     {
233       int n_i = returnNumberOfTermOccurences((String)sortedKeys.get(i));
234       double ratio = (((double)N) / ((double)n_i));
235       inverseDocumentFrequency = Math.log(ratio);
236
237       Double e = new Double(inverseDocumentFrequency);
238       inverseFrequencies.add(e);
239     }
240     ArrayList inverseFrequenciesClone = (ArrayList)inverseFrequencies.clone();
241     inverseFrequencies.clear();
242
243     return inverseFrequenciesClone;
244   }
245
246   /*
247   *Metode som returnerer superWeightList, som inneholder N
248   *antall arrayLister med vekter i stigende rekkefølge.
249   */
250   public ArrayList calculateWeight(ArrayList tokFreq)
251   {
252     this.tokFreq = tokFreq;
253     this.N = tokFreq.size()/2;
254     //for alle abstracts, beregn frekvens og invers frekvens på alle termer
255     //I arraylisten tokFreq er annenhvert objekt en sortedKeys og en
        sortedFrequencies-ArrayList
256     for(int x = 0; x < tokFreq.size()-1; x=x+2)
257     {
258       ArrayList sortedKeys = (ArrayList)tokFreq.get(x);
259       ArrayList sortedFrequencies = (ArrayList)tokFreq.get(x+1);
260
261       //dersom et abstract har fått minst ett protein- el gen-treff
262       if(!(sortedFrequencies.size()==0)){
263
264       ArrayList norm = calculateNormalizedFrequency(sortedKeys, sortedFrequencies
          );
265       ArrayList inverse = calculateInverseDocumentFrequency(sortedKeys);
266
267       for(int t = 0; t < norm.size(); t++)
268       {
269         double weight =( ((Double)norm.get(t)).doubleValue() ) * ( ((Double)
            inverse.get(t)).doubleValue() );
270         Double e = new Double(weight);
271         //weightList får alle vektene til et abstract
272         weightList.add(e);
273       }
274       ArrayList weightListClone = (ArrayList)weightList.clone();
275       superWeightList.add(weightListClone);
276       weightList.clear();
277       }
278       //dersom abstractet ikke har fått treff(skjer på gen-treff i blant)
279       //sortedFrequencies.size = 0
280       else
281       {
282         double weight = 0;
283         Double e = new Double(weight);
```

```
284         weightList.add(e);
285         ArrayList weightListClone = (ArrayList)weightList.clone();
286         superWeightList.add(weightListClone);
287         weightList.clear();
288       }
289     }
290     //System.out.println("superWeightList-size:"+superWeightList.size());
291     ArrayList superWeightListClone = (ArrayList)superWeightList.clone();
292     superWeightList.clear();
293 
294     return superWeightListClone;
295   }
296 
297   /*
298   *Metode som trekker ut de tre mest relevante generelle
299   *termer fra supervekt-listen. Returnerer en arraylist.
300   */
301   public ArrayList extractTopThreeMostRelevant(ArrayList superWeightList)
302   {
303     double largestWeight = 0;
304     double secondLargestWeight = 0;
305     double thirdLargestWeight = 0;
306     double b = 0;
307     double a = 0;
308     int tlpIndex = 0;
309     int tluIndex = 0;
310     int slpIndex = 0;
311     int sluIndex = 0;
312     int lpIndex = 0;
313     int luIndex = 0;
314 
315     for(int u = 0; u < superWeightList.size(); u++)
316     {
317       ArrayList abstractWeights = (ArrayList)superWeightList.get(u);
318 
319       for(int p = 0; p < abstractWeights.size(); p++)
320       {
321 
322 
323         if( ((Double)abstractWeights.get(p)).doubleValue() > thirdLargestWeight)
324         {
325           a = thirdLargestWeight;
326           thirdLargestWeight = ((Double)abstractWeights.get(p)).doubleValue();
327           tlpIndex = p;
328           tluIndex = u;
329 
330           if( ((Double)abstractWeights.get(p)).doubleValue() >
              secondLargestWeight)
331           {
332             b = secondLargestWeight;
333             secondLargestWeight = ((Double)abstractWeights.get(p)).doubleValue();
334             slpIndex = p;
335             sluIndex = u;
336             thirdLargestWeight = a;
337 
338             if( ((Double)abstractWeights.get(p)).doubleValue() > largestWeight)
339             {
340               largestWeight = ((Double)abstractWeights.get(p)).doubleValue();
341               lpIndex = p;
342               luIndex = u;
343               secondLargestWeight = b;
344             }
```

```
345
346             }
347           }
348         }
349
350       }
351
352       String mostRelevantTerm = (String)((ArrayList)tokFreq.get(luIndex*2)).get(
            lpIndex);
353       String secondMostRelevantTerm = (String)((ArrayList)tokFreq.get(sluIndex*2)).
            get(slpIndex);
354       String thirdMostRelevantTerm = (String)((ArrayList)tokFreq.get(tluIndex*2)).
            get(tlpIndex);
355
356       ArrayList relevantList = new ArrayList();
357       relevantList.add(mostRelevantTerm);
358       relevantList.add(secondMostRelevantTerm);
359       relevantList.add(thirdMostRelevantTerm);
360
361       return relevantList;
362     }
363
364 }
```

Dictionary.java

```
1  package master;
2
3  import java.util.ArrayList;
4  import java.util.Hashtable;
5
6  /*
7   *Denne klassen skal inneholde gen- og proteindictionaries.
8   *Både stemmede og ustemmede versjoner skal kunne loades inn i minnet.
9   */
10
11 /**
12  *@author skuland
13  *
14  */
15
16 public class Dictionary {
17
18   public Hashtable geneDictionary;
19   private ArrayList a;
20   public Hashtable proteinDictionary_level_0, proteinDictionary_level_1,
       proteinDictionary_level_2, proteinDictionary_level_3,
       proteinDictionary_level_4, proteinDictionary_level_5,
       proteinDictionary_level_6, proteinDictionary_level_7,
       proteinDictionary_level_8, proteinDictionary_level_9;
21
22   /*
23   *Konstruktoren Dictionary som kaller ProduceHashTables-klassen
24   *Gen og protein hash tables blir loaded inn i minnet.
25   *useStemming bestemmer om stemmet eller ustemmet protein
26   *hash table skal loades.
27   */
28   public Dictionary(boolean useStemming)
29   {
30     ProduceHashTables pht = new ProduceHashTables();
31     this.geneDictionary = pht.createGeneHashTable();
32     this.a = pht.createProteinHashtables(useStemming);
33     this.proteinDictionary_level_0 = (Hashtable)a.get(0);
34     this.proteinDictionary_level_1 = (Hashtable)a.get(1);
35     this.proteinDictionary_level_2 = (Hashtable)a.get(2);
36     this.proteinDictionary_level_3 = (Hashtable)a.get(3);
37     this.proteinDictionary_level_4 = (Hashtable)a.get(4);
38     this.proteinDictionary_level_5 = (Hashtable)a.get(5);
39     this.proteinDictionary_level_6 = (Hashtable)a.get(6);
40     this.proteinDictionary_level_7 = (Hashtable)a.get(7);
41     this.proteinDictionary_level_8 = (Hashtable)a.get(8);
42     this.proteinDictionary_level_9 = (Hashtable)a.get(9);
43   }
44 }
```

ProduceHashTables.java

```
1  package master;
2
3  import java.io.BufferedReader;
4  import java.io.IOException;
5  import java.io.InputStreamReader;
6  import java.io.PrintWriter;
7  import java.net.URL;
8  import java.net.URLConnection;
9  import java.util.ArrayList;
10 import java.util.Hashtable;
11 import java.util.Iterator;
12 import java.util.Set;
13 import java.util.jar.JarFile;
14 import java.util.regex.Matcher;
15 import java.util.regex.Pattern;
16
17 /*
18  *Klasse for å skape Hashtable-datastrukturene som skal benyttes
19  *som proteindictionaries.
20  */
21
22 /**
23  *@author skuland
24  *
25  */
26
27 public class ProduceHashTables{
28
29   private JarFile jf = null;
30   private BufferedReader br, genereader;
31   private PrintWriter pw;
32   private Hashtable nulltable;
33   private Hashtable ht0, ht1, ht2, ht3, ht4, ht5, ht6, ht7, ht8, ht9;
34   private Hashtable pointer0, pointer1, pointer2, pointer3, pointer4, pointer5,
       pointer6, pointer7, pointer8, pointer9;
35
36   /*
37   *Konstruktoren ProduceHashTables. Hash tables initialiseres.
38   */
39   public ProduceHashTables()
40   {
41     try{
42       jf = new JarFile("prototype.jar");
43     }catch(Exception e)
44     {}
45
46     ht0 = new Hashtable();
47     ht1 = new Hashtable();
48     ht2 = new Hashtable();
49     ht3 = new Hashtable();
50     ht4 = new Hashtable();
51     ht5 = new Hashtable();
52     ht6 = new Hashtable();
53     ht7 = new Hashtable();
54     ht8 = new Hashtable();
55     ht9 = new Hashtable();
56
57     pointer0 = new Hashtable();
```

```
58      pointer1 = new Hashtable();
59      pointer2 = new Hashtable();
60      pointer3 = new Hashtable();
61      pointer4 = new Hashtable();
62      pointer5 = new Hashtable();
63      pointer6 = new Hashtable();
64      pointer7 = new Hashtable();
65      pointer8 = new Hashtable();
66
67      nulltable = new Hashtable();
68    }
69
70    /*
71     *Denne metoden skaper gen hashtabellen. Returnerer hashtabellen.
72     */
73    public Hashtable createGeneHashTable()
74    {
75      try{
76        // Get current classloader
77        ClassLoader cl = this.getClass().getClassLoader();
78        URL geneURL = cl.getResource("final_dictionaries/geneDictionary.txt");
79        URLConnection geneConnection = geneURL.openConnection();
80        geneConnection.setDoInput(true);
81        geneConnection.connect();
82        genereader = new BufferedReader(new InputStreamReader(geneConnection.
            getInputStream()));
83
84        }catch(Exception fe){fe.printStackTrace();}
85
86      Hashtable genetable = new Hashtable();
87      Integer i = new Integer(0);
88      String line = "";
89      try{
90        while( (line = genereader.readLine()) != null )
91        {
92          if(line.length() > 0)
93          {
94            genetable.put(line, i);
95          }
96
97        }
98        System.out.println("genetable size"+genetable.size());
99
100     }catch(IOException io){io.printStackTrace();}
101
102     return genetable;
103   }
104
105   /*
106   *Denne metoden returnerer protein hashtabellen.
107   *Den blir skapt enten som stemmet eller ustemmet versjon.
108   */
109   public ArrayList createProteinHashtables(boolean b)
110   {
111     ClassLoader cl = this.getClass().getClassLoader();
112
113     if(b){
114       try{
115         URL stProtURL = cl.getResource("final_dictionaries/
              stemmedProteinDictionary2.txt");
116         URLConnection stProtConnection = stProtURL.openConnection();
117         stProtConnection.setDoInput(true);
```

```
118         stProtConnection.connect();
119         br = new BufferedReader(new InputStreamReader(stProtConnection.
              getInputStream()));
120
121         }catch(Exception fe){fe.printStackTrace();}
122       }
123       else
124       {
125         try{
126           URL protURL = cl.getResource("final_dictionaries/proteinDictionary2.txt")
              ;
127           URLConnection protConnection = protURL.openConnection();
128           protConnection.setDoInput(true);
129           protConnection.connect();
130           br = new BufferedReader(new InputStreamReader(protConnection.
              getInputStream()));
131
132           }catch(Exception fe){fe.printStackTrace();}
133       }
134
135       int n = 11;
136       String line = "";
137       int length = 0;
138       int j = 0;
139       System.out.println(line.equals(null));
140       System.out.println(br.equals(null));
141
142       try{
143         //leser inn alle proteinnavnene, totalt 425K
144         while( (line = br.readLine()) != null )
145         {
146           if(line.length() > 0)
147           {
148             String[] s = line.split("\\s");
149
150             /*
151             *s inneholder nå alle tokens på denne linja i input-fila.
152             *Det er maksimalt 10 tokens på en linje som skal fordeles
153             *på Hashtables
154             *
155             *Itererer over tokenene i et proteinnavn, fra 0 til max 9
156             *Alle de interne hashtabellene (pointer0-9) skal ha null
157             *på value, men det er ikke tilfelle for ht0 til ht9. Disse
158             *vil initielt ha null, men ved gjennomlesing av linje nr. 2
159             *må man ikke slette tidligere satt verdi (som kan være en
160             *hashtable med innslag på neste nivå).Fjerner paranteser,
161             *samt komma og punktum, kolon og semikolon
162             */
163             for(int k = 0; k < s.length; k++)
164             {
165               //0.nivå
166               if(k==0)
167               {
168                 //Dette ordet er allerede lagt inn i hashtabell på nivå 0
169                 if(ht0.containsKey(s[k]))
170                 {
171                   //Hashtable ha0 er den interne hashtabellen som er på value-
                      plassen
172                   //til hashtabell ht0. ha0 skal inneholde de nøkler som er mulige
173                   //kombinasjoner for ord nr. 2 i proteinnavnet
174                   Hashtable ha0 = (Hashtable)ht0.get(s[k]);
175                   ht0.put(s[k],ha0);
```

```
176               }
177               //Nytt ord på nivå 0
178               else
179               {
180                 ht0.put(s[k], new Hashtable());
181               }
182             }
183             //1.nivå
184             else if(k==1)
185             {
186               //Hvis dette ordet har blitt lagt til på 1.nivå før
187               if(ht1.containsKey(s[k]))
188               {
189                 Hashtable ha1 = (Hashtable)ht1.get(s[k]);
190                 ht1.put(s[k],ha1);
191
192                 Hashtable ha0 = (Hashtable)ht0.get(s[k-1]);
193                 ha0.put(s[k], nulltable);
194                 ht0.put(s[k-1],ha0);
195               }
196               //Dersom det er første gang dette ordet opptrer på 1.nivå
197               else
198               {
199                 //Legger til nøkkel på nivå 1 til hashtable på nivå 1
200                 ht1.put(s[k], new Hashtable());
201
202                 //Legger til nøkkel på 1.nivå til intern hashtable på nivå 0
203                 Hashtable ha0 = (Hashtable)ht0.get(s[k-1]);
204                 ha0.put(s[k], nulltable);
205                 ht0.put(s[k-1],ha0);
206
207               }
208             }
209             //2.nivå
210             else if(k==2)
211             {
212               if(ht2.containsKey(s[k]))
213               {
214                 Hashtable ha2 = (Hashtable)ht2.get(s[k]);
215                 ht2.put(s[k],ha2);
216
217                 Hashtable ha1 = (Hashtable)ht1.get(s[k-1]);
218                 ha1.put(s[k], nulltable);
219                 ht1.put(s[k-1],ha1);
220               }
221               else
222               {
223                 ht2.put(s[k], new Hashtable());
224
225                 Hashtable ha1 = (Hashtable)ht1.get(s[k-1]);
226                 ha1.put(s[k], nulltable);
227                 ht1.put(s[k-1],ha1);
228               }
229             }
230             //3.nivå
231             else if(k==3)
232             {
233               if(ht3.containsKey(s[k]))
234               {
235                 Hashtable ha3 = (Hashtable)ht3.get(s[k]);
236                 ht3.put(s[k],ha3);
237
```

```
238                 Hashtable ha2 = (Hashtable)ht2.get(s[k-1]);
239                 ha2.put(s[k], nulltable);
240                 ht2.put(s[k-1],ha2);
241               }
242               else
243               {
244                 ht3.put(s[k], new Hashtable());
245
246                 Hashtable ha2 = (Hashtable)ht2.get(s[k-1]);
247                 ha2.put(s[k], nulltable);
248                 ht2.put(s[k-1],ha2);
249               }
250             }
251             //4.nivå
252             else if(k==4)
253             {
254               if(ht4.containsKey(s[k]))
255               {
256                 Hashtable ha4 = (Hashtable)ht4.get(s[k]);
257                 ht4.put(s[k],ha4);
258
259                 Hashtable ha3 = (Hashtable)ht3.get(s[k-1]);
260                 ha3.put(s[k], nulltable);
261                 ht3.put(s[k-1],ha3);
262               }
263               else
264               {
265                 ht4.put(s[k], new Hashtable());
266
267                 Hashtable ha3 = (Hashtable)ht3.get(s[k-1]);
268                 ha3.put(s[k], nulltable);
269                 ht3.put(s[k-1],ha3);
270               }
271             }
272             //5.nivå
273             else if(k==5)
274             {
275               if(ht5.containsKey(s[k]))
276               {
277                 Hashtable ha5 = (Hashtable)ht5.get(s[k]);
278                 ht5.put(s[k],ha5);
279
280                 Hashtable ha4 = (Hashtable)ht4.get(s[k-1]);
281                 ha4.put(s[k], nulltable);
282                 ht4.put(s[k-1],ha4);
283               }
284               else
285               {
286                 ht5.put(s[k], new Hashtable());
287
288                 Hashtable ha4 = (Hashtable)ht4.get(s[k-1]);
289                 ha4.put(s[k], nulltable);
290                 ht4.put(s[k-1],ha4);
291               }
292             }
293             //6.nivå
294             else if(k==6)
295             {
296               if(ht6.containsKey(s[k]))
297               {
298                 Hashtable ha6 = (Hashtable)ht6.get(s[k]);
299                 ht6.put(s[k],ha6);
```

```
300
301                 Hashtable ha5 = (Hashtable)ht5.get(s[k-1]);
302                 ha5.put(s[k], nulltable);
303                 ht5.put(s[k-1],ha5);
304               }
305               else
306               {
307                 ht6.put(s[k], new Hashtable());
308
309                 Hashtable ha5 = (Hashtable)ht5.get(s[k-1]);
310                 ha5.put(s[k], nulltable);
311                 ht5.put(s[k-1],ha5);
312               }
313             }
314             //7.nivå
315             else if(k==7)
316             {
317               if(ht7.containsKey(s[k]))
318               {
319                 Hashtable ha7 = (Hashtable)ht7.get(s[k]);
320                 ht7.put(s[k],ha7);
321
322                 Hashtable ha6 = (Hashtable)ht6.get(s[k-1]);
323                 ha6.put(s[k], nulltable);
324                 ht6.put(s[k-1],ha6);
325               }
326               else
327               {
328                 ht7.put(s[k], new Hashtable());
329
330                 Hashtable ha6 = (Hashtable)ht6.get(s[k-1]);
331                 ha6.put(s[k], nulltable);
332                 ht6.put(s[k-1],ha6);
333               }
334             }
335             //8.nivå
336             else if(k==8)
337             {
338               if(ht8.containsKey(s[k]))
339               {
340                 Hashtable ha8 = (Hashtable)ht8.get(s[k]);
341                 ht8.put(s[k],ha8);
342
343                 Hashtable ha7 = (Hashtable)ht7.get(s[k-1]);
344                 ha7.put(s[k], nulltable);
345                 ht7.put(s[k-1],ha7);
346               }
347               else
348               {
349                 ht8.put(s[k], new Hashtable());
350
351                 Hashtable ha7 = (Hashtable)ht7.get(s[k-1]);
352                 ha7.put(s[k], nulltable);
353                 ht7.put(s[k-1],ha7);
354               }
355             }
356             //9.nivå
357             else if(k==9)
358             {
359               if(ht9.containsKey(s[k]))
360               {
361                 Hashtable ha9 = (Hashtable)ht9.get(s[k]);
```

```
362                 ht9.put(s[k],ha9);
363
364                 Hashtable ha8 = (Hashtable)ht8.get(s[k-1]);
365                 ha8.put(s[k], nulltable);
366                 ht8.put(s[k-1],ha8);
367               }
368               else
369               {
370                 ht9.put(s[k], new Hashtable());
371
372                 Hashtable ha8 = (Hashtable)ht8.get(s[k-1]);
373                 ha8.put(s[k], nulltable);
374                 ht8.put(s[k-1],ha8);
375               }
376             }
377           }
378
379         }
380       }
381       br.close();
382
383       System.out.println(ht0.size());
384       System.out.println(ht1.size());
385       System.out.println(ht2.size());
386       System.out.println(ht3.size());
387       System.out.println(ht4.size());
388       System.out.println(ht5.size());
389       System.out.println(ht6.size());
390       System.out.println(ht7.size());
391       System.out.println(ht8.size());
392       System.out.println(ht9.size());
393
394     }
395     catch(IOException ioe){ioe.printStackTrace();}
396
397     ArrayList listOfProteinHashTables = new ArrayList();
398
399     listOfProteinHashTables.add(ht0);
400     listOfProteinHashTables.add(ht1);
401     listOfProteinHashTables.add(ht2);
402     listOfProteinHashTables.add(ht3);
403     listOfProteinHashTables.add(ht4);
404     listOfProteinHashTables.add(ht5);
405     listOfProteinHashTables.add(ht6);
406     listOfProteinHashTables.add(ht7);
407     listOfProteinHashTables.add(ht8);
408     listOfProteinHashTables.add(ht9);
409
410     return listOfProteinHashTables;
411
412   }
413
414   /*
415   *Metode som har blitt brukt for å lage
416   *stemmet versjon av protein hashtabellen.
417   */
418   private void stemProteinNames()
419   {
420     try{
421     Stemmer st = new Stemmer();
422
423     String line = "";
```

```
424     while( (line = br.readLine()) != null )
425     {
426       if(line.length() > 0)
427       {
428         String[] s = line.split("\\s");
429
430         for(int i = 0; i < s.length;i++)
431         {
432           int b = s[i].length();
433           for(int a = 0; a < b; a++){
434             st.add((s[i]).charAt(a));
435           }
436           st.stem();
437           String stemmedToken = st.toString();
438           if(!(s[i]).equals(stemmedToken))
439           {
440             s[i] = stemmedToken;
441           }
442           pw.print(s[i] + " ");
443         }
444         pw.print("\r");
445       }
446
447     }
448     pw.close();
449     }
450     catch(Exception e)
451     {
452       e.printStackTrace();
453     }
454
455   }
456
457 }
```

Test.java

```
1  package master;
2
3  import java.io.BufferedReader;
4  import java.io.File;
5  import java.io.FileInputStream;
6  import java.io.FileReader;
7  import java.io.PrintWriter;
8  import java.util.ArrayList;
9
10 import javax.xml.parsers.ParserConfigurationException;
11 import javax.xml.parsers.SAXParser;
12 import javax.xml.parsers.SAXParserFactory;
13
14 import org.xml.sax.Attributes;
15 import org.xml.sax.SAXException;
16 import org.xml.sax.SAXNotRecognizedException;
17 import org.xml.sax.SAXNotSupportedException;
18 import org.xml.sax.helpers.DefaultHandler;
19
20 /*
21  *Denne klassen er laget for å forsøke å få testet systemet
22  *mot ferdig taggede abstracts. Målet er å få tall på
23  *precision og recall.
24  */
25
26 /**
27  *@author skuland
28  *
29  */
30
31 public class Test extends DefaultHandler{
32
33   private SAXParser parser2;
34   private int counter = 0;
35   private PrintWriter proteinWriter;
36   private PrintWriter pmidWriter;
37   private BufferedReader proteinReader;
38   private BufferedReader pmidReader;
39   private StringBuffer sb1, sb2;
40   private boolean insideAbstract = false;
41   private ArrayList pmidList, taggedProteinList;
42   private String a = "", b = "", pmidString = "";
43
44   /*
45    *Konstruktoren skaper en SAXParser.
46    */
47   public Test()
48   {
49     pmidList = new ArrayList();
50     taggedProteinList = new ArrayList();
51     try{
52       SAXParserFactory parserFactory = SAXParserFactory.newInstance();
53       parserFactory.setFeature("http://xml.org/sax/features/namespaces", true);
54       parserFactory.setFeature("http://xml.org/sax/features/namespace-prefixes",
           false);
55       parser2 = parserFactory.newSAXParser();
56       parser2.getXMLReader().setContentHandler(this);
57       parser2.getXMLReader().setDTDHandler(this);
```

```
58        parser2.getXMLReader().setErrorHandler(this);
59        parser2.getXMLReader().setEntityResolver(this);
60      } catch (SAXNotRecognizedException e) {
61        e.printStackTrace();
62      } catch (SAXNotSupportedException e) {
63        e.printStackTrace();
64      } catch (ParserConfigurationException e) {
65        e.printStackTrace();
66      } catch (SAXException e) {
67        e.printStackTrace();
68      }
69
70      //parseFile();
71      //readYapexTaggedProteins();
72      //readYapexPmids();
73    }
74
75    /*
76    *Metode som leser inn alle taggede proteinnavn og putter de
77    *i en arraylist taggedProteinList. Denne inneholder alle 1559
78    *proteinene som er tagget i Yapex-treningsfila.
79    */
80    public ArrayList readYapexTaggedProteins(){
81      String line = "";
82      try{
83        //proteinReader = new BufferedReader(new FileReader(new File("
            YAPEX_TAGGED_PROTEINS.txt")));
84        proteinReader = new BufferedReader(new FileReader(new File("
            YAPEX_TAGGED_PROTEINS_TEST.txt")));
85        }catch(Exception e){}
86      try{
87        while( (line = proteinReader.readLine()) != null )
88        {
89          if(line.length() > 0)
90          {
91            taggedProteinList.add(line);
92          }
93        }
94      }
95      catch(Exception e){}
96
97      System.out.println("taggedProteinList.size: "+taggedProteinList.size());
98      return taggedProteinList;
99    }
100
101   /*
102    *Metode som returnerer string med 99pmider som brukes til trening.
103    */
104   public String readYapexPmids()
105   {
106     String line = "";
107     try{
108       //pmidReader = new BufferedReader(new FileReader(new File("YAPEX_PMIDS.txt
            ")));
109       pmidReader = new BufferedReader(new FileReader(new File("YAPEX_PMIDS_TEST.
            txt")));
110     }catch(Exception e){}
111     try{
112     while( (line = pmidReader.readLine()) != null )
113     {
114
115       if(line.length() > 0)
```

```
116         {
117           pmidList.add(line);
118         }
119
120       }
121       }
122       catch(Exception e){}
123
124       System.out.println("pmidlist.size: "+pmidList.size());
125
126       for(int j = 0; j<pmidList.size();j++)
127       {
128         if(j<pmidList.size()-1)
129         {
130           a = (String)pmidList.get(j)+ ",";
131           b = b + a;
132         }
133         else
134         {
135           a = (String)pmidList.get(j);
136           b = b + a;
137         }
138       }
139
140       //Her har jeg en pmidstring på form tall,tall,tall og størrelse 99
141       pmidString = b;
142
143       return pmidString;
144
145     }
146
147     /*
148      *Metode som initierer parsing av test XML fil
149      */
150     public void parseFile()
151     {
152
153       try{
154         //File yapex = new File("master/yapex_ref_collection.txt");
155         File yapex = new File("master/yapex_test_collection.txt");
156         FileInputStream fis = new FileInputStream(yapex);
157
158         //pmidWriter = new PrintWriter(new BufferedWriter(new FileWriter(new File("
              YAPEX_PMIDS_TEST.txt"),false)));
159         //proteinWriter = new PrintWriter(new BufferedWriter(new FileWriter(new
              File("YAPEX_TAGGED_PROTEINS_TEST.txt"),false)));
160
161
162         sb1 = new StringBuffer();
163         sb2 = new StringBuffer();
164
165         parser2.parse(fis, this);
166         pmidWriter.close();
167         proteinWriter.close();
168         System.out.println("Antall pmider:"+count);
169         System.out.println("Antall protein i abstractene:"+proteinCount);
170         System.out.println("Antall slutttagger PMID: "+ countPMIDEND);
171         System.out.println("Antall slutttagger Protname: "+ countProtnameEnd);
172         System.out.println("Counter == 1: "+county);
173       }
174       catch(Exception e)
175       {
```

```
176         e.printStackTrace();
177       }
178     }
179
180     public void startDocument()
181     {
182
183     }
184
185     int count = 0;
186     int proteinCount = 0;
187     public void startElement(String uri, String localName, String qName, Attributes
          attributes)throws SAXException
188     {
189       try{
190         if(localName == "PMID")
191         {
192           counter = 1;
193           System.out.println("YAPEX PMID");
194           count++;
195         }
196         else if(localName == "AbstractText")
197         {
198           insideAbstract = true;
199         }
200         else if(localName =="Protname" && insideAbstract)
201         {
202           proteinCount++;
203           counter = 2;
204         }
205
206       }catch(Exception e)
207       {
208         e.printStackTrace();
209       }
210     }
211
212     int county = 0;
213     public void characters(char[] ch, int start, int length) throws SAXException
214     {
215
216       if(counter == 1)
217       {
218         county++;
219
220         String s = new String(ch,start,length);
221         sb1.append(s);
222       }
223
224       if(counter ==2)
225       {
226         String s = new String(ch,start,length);
227         sb2.append(s);
228       }
229
230     }
231
232     int countPMIDEND = 0;
233     int countProtnameEnd = 0;
234     public void endElement(String uri, String localName, String qName) throws
        SAXException
235     {
```

```
236       if(localName == "PMID")
237       {
238         countPMIDEND++;
239
240         counter = 0;
241         String t = sb1.toString();
242         pmidWriter.println(t);
243         pmidWriter.println("\r\n");
244         sb1.delete(0,sb1.length());
245
246
247       }
248       else if(localName =="Protname")
249       {
250         countProtnameEnd++;
251         counter = 0;
252
253         String t = sb2.toString();
254         proteinWriter.println(t);
255         //proteinWriter.println("\r");
256         sb2.delete(0,sb2.length());
257
258       }
259
260       else if(localName == "AbstractText")
261       {
262         insideAbstract = false;
263       }
264     }
265
266   }
```

ProcessDictionary.java

```
1  package master;
2
3  import java.io.BufferedReader;
4  import java.io.BufferedWriter;
5  import java.io.File;
6  import java.io.FileReader;
7  import java.io.FileWriter;
8  import java.io.PrintWriter;
9  import java.util.ArrayList;
10 import java.util.Calendar;
11 import java.util.Collections;
12 import java.util.Hashtable;
13 import java.util.Iterator;
14 import java.util.Set;
15 import java.util.regex.Pattern;
16
17 /*
18  *Class that has been used during construction
19  *of the gene and protein dictionaries.
20  *Contains mostly regular expression-logic.
21  */
22
23 /**
24  *@author skuland
25  *
26  */
27 public class ProcessDictionary {
28
29   private File proteinnames, removedDuplicates, removedNumbers, testy, testfile,
       updatedFile, updatedFile2, alfaSorted;
30   private File genenames, dictionary, genenames2, genenames3;
31
32   private FileWriter fw, fw2, fw3, updatedFileWriter, dictionaryFileWriter, fw4,
       fw5;
33   private BufferedWriter bw, bw2, bw3, updatedBufferedWriter,
       dictionaryBufferedWriter,bw4, bw5;
34   private PrintWriter pw, pw2, pw3, printwriter, dictionaryPrintWriter,pw4, pw5;
35
36   private FileReader fr, fr2, fr3, fr4, fr5, fr6;
37   private BufferedReader br, br2, br3, br4, br5, br6;
38
39   private Hashtable proteintable, proteintable2, genenamestable;
40   private ArrayList list1, list2, sortProteinNamesList, sortGeneNamesList;
41   private Calendar cal, cal2, cal3, cal4;
42
43   /*
44    *Konstruktoren brukes til å sette i gang
45    *diverse prosesseringsmetoder.
46    */
47   public ProcessDictionary()
48   {
49     try{
50       list1 = new ArrayList();
51       list2 = new ArrayList();
52       sortProteinNamesList = new ArrayList();
53       sortGeneNamesList = new ArrayList();
54       proteintable = new Hashtable();
55       proteintable2 = new Hashtable();
```

```
56        genenamestable = new Hashtable();
57      }
58      catch(Exception e)
59      {
60        e.printStackTrace();
61      }
62
63      cal3 = Calendar.getInstance();
64      long before2 = cal3.getTimeInMillis();
65
66      //removeEntries();
67      //testremoveEntries();
68      //processGeneNames();
69      //alphabeticallySortProteinNames();
70      //alphabeticallySortGeneNames();
71
72      //magnus();
73      //removeDuplicateNames();
74      //removeEntries();
75      //alphabeticallySortProteinNames();
76      stemProteinNames();
77
78      cal4 = Calendar.getInstance();
79      long after2 = cal4.getTimeInMillis();
80      long time2 = after2 - before2;
81      System.out.println("Tidsforbruk: "+ (double)time2/1000 + " sekunder");
82    }
83
84    /*
85     *Denne metoden stemmer protein-navnene.
86     */
87    private void stemProteinNames()
88    {
89      BufferedReader br = null;
90      PrintWriter pri = null;
91      try{
92        br = new BufferedReader(new FileReader(new File("proteinDictionary2.txt")))
            ;
93        pri = new PrintWriter(new BufferedWriter(new FileWriter(new File("
            stemmedProteinDictionary2.txt"),false)));
94      }
95      catch(Exception e){}
96
97      try{
98      Stemmer st = new Stemmer();
99
100     String line = "";
101     while( (line = br.readLine()) != null )
102     {
103       if(line.length() > 0)
104       {
105         String[] s = line.split("\\s");
106
107         for(int i = 0; i < s.length;i++)
108         {
109           int b = s[i].length();
110           for(int a = 0; a < b; a++){
111             st.add((s[i]).charAt(a));
112           }
113           st.stem();
114           String stemmedToken = st.toString();
115           if(!(s[i]).equals(stemmedToken))
```

```
116           {
117             s[i] = stemmedToken;
118           }
119           pri.print(s[i] + " ");
120         }
121         pri.print("\r");
122       }
123
124     }
125     br.close();
126     pri.close();
127     }
128     catch(Exception e)
129     {
130       e.printStackTrace();
131     }
132
133   }
134
135   /*
136    *Regular-expression-metode.
137    */
138   public void magnus()
139   {
140     BufferedReader br = null;
141     PrintWriter pri = null;
142     try{
143       br = new BufferedReader(new FileReader(new File("kournikova.txt")));
144       pri = new PrintWriter(new BufferedWriter(new FileWriter(new File("UPDATED-
          DICTIONARY.txt"),false)));
145     }
146     catch(Exception e){}
147
148     //((String)al.get(a)).matches("[\\d]+")
149
150     int tell = 0;
151     int county=0;
152     StringBuffer sb = new StringBuffer();
153
154     try{
155       String line = "";
156       String newline = "";
157       while( (line = br.readLine()) != null )
158       {
159         if(line.length() > 0)
160         {
161
162           county++;
163           /*
164            //tegn ).
165           if(line.matches(".*\\u0029\\u002E"))
166           {
167             tell++;
168             line = line.substring(0, line.length()-2);
169           }
170           */
171           //tegn )
172           if(line.matches(".*\\u0029"))
173           {
174             tell++;
175             line = line.substring(0, line.length()-1);
176           }
```

```
177           pri.println(line);
178
179           /*
180           county++;
181           String[] s = line.split("\\s");
182           if(s[0].equals("similarities") && s[1].equals("with"))
183           {
184             tell++;
185
186             for(int h = 2; h< s.length;h++)
187             {
188               sb.append(s[h]+" ");
189             }
190             //sb.append(s[s.length-1]);
191             int lastindex = sb.length()-1;
192             System.out.println(lastindex);
193             if(lastindex > 0){
194               sb.deleteCharAt(lastindex);
195             }
196             line = sb.toString();
197             System.out.println(line);
198
199           }
200           pri.println(line);
201           sb.delete(0,sb.length());
202           */
203         }
204
205       }
206       br.close();
207       pri.close();
208     }catch(Exception e){}
209
210
211
212     System.out.println("Antall linjer lest: "+county);
213     System.out.println("Antall linjer med predicted... : "+tell);
214
215
216   }
217
218   /*
219    *Metode som alfabetisk sorterer gen-navnene i tekstfila
220    */
221   public void alphabeticallySortGeneNames()
222   {
223     try
224     {
225       String line = "";
226       while( (line=br6.readLine()) != null)
227       {
228         if(line.length() > 0)
229         {
230           sortGeneNamesList.add(line);
231         }
232       }
233       br6.close();
234       System.out.println("Size: "+sortGeneNamesList.size());
235
236       Collections.sort(sortGeneNamesList);
237
238       for(int i = 0; i< sortGeneNamesList.size(); i++)
```

```
239         {
240           pw5.println((String)sortGeneNamesList.get(i));
241         }
242         pw5.close();
243 
244       }
245       catch(Exception e)
246       {
247         e.printStackTrace();
248       }
249     }
250 
251     /*
252      *Metode som alfabetisk sorterer protein-navnene i tekstfila
253      */
254     public void alphabeticallySortProteinNames()
255     {
256       BufferedReader br = null;
257       PrintWriter pri = null;
258     try{
259       br = new BufferedReader(new FileReader(new File("kournikova.txt")));
260       pri = new PrintWriter(new BufferedWriter(new FileWriter(new File("UPDATED-
          DICTIONARY.txt"),false)));
261       }
262     catch(Exception e){}
263 
264       try
265       {
266         String line = "";
267         while( (line=br.readLine()) != null)
268         {
269           if(line.length() > 0)
270           {
271             sortProteinNamesList.add(line);
272           }
273         }
274         br.close();
275         System.out.println("Size: "+sortProteinNamesList.size());
276 
277         Collections.sort(sortProteinNamesList);
278 
279         for(int i = 0; i< sortProteinNamesList.size(); i++)
280         {
281           pri.println((String)sortProteinNamesList.get(i));
282         }
283         pri.close();
284       }
285       catch(Exception e)
286       {
287         e.printStackTrace();
288       }
289     }
290 
291     /*
292      *Metode for prosessering av gennavnene
293      */
294     public void processGeneNames()
295     {
296       try{
297         Integer i = new Integer(0);
298         String line = "";
299         int size = 0;
```

```
300       int count =0;
301       String withdrawn = ".*\\p{Punct}(withdrawn).*";
302
303       while( (line = br5.readLine()) != null )
304       {
305         if(line.length() > 0)
306         {
307           count++;
308           boolean b = Pattern.matches(withdrawn,line);
309           if(!b)
310           {
311             genenamestable.put(line, i);
312           }
313
314         }
315       }
316       br5.close();
317       size = genenamestable.size();
318
319       //skrive genenamestable-objektet til fil
320
321       String genenames_to_dictionary = genenamestable.toString();
322       dictionaryPrintWriter.println(genenames_to_dictionary);
323       dictionaryPrintWriter.close();
324
325       Set keys = genenamestable.keySet();
326       Iterator it = keys.iterator();
327
328       while(it.hasNext())
329       {
330         String key = (String)it.next();
331         pw3.println(key);
332       }
333       pw3.close();
334       System.out.println("Antall unike: " + size);
335       System.out.println("Fjernede elementer: " + (count-size));
336       System.out.println("Prosent av opprinnelig fil: "+ (double)size/count);
337     }
338     catch(Exception e)
339     {
340       e.printStackTrace();
341     }
342   }
343
344   /*
345    *Metode som fjerner duplikat-innslag
346    */
347   public void removeDuplicateNames()
348   {
349
350     BufferedReader br = null;
351     PrintWriter pri = null;
352   try{
353     br = new BufferedReader(new FileReader(new File("UPDATED-DICTIONARY.txt")));
354     pri = new PrintWriter(new BufferedWriter(new FileWriter(new File("kournikova.
        txt"),false)));
355     }
356   catch(Exception e){}
357
358       int size = 0;
359
360     try{
```

```
361         Integer i = new Integer(0);
362         String line;
363         int count =0;
364         while( (line = br.readLine()) != null )
365         {
366           //den skriver ut annenhver linje tom pga. \r -tegnet, derfor må teste
              lengde større enn 0
367           if(line.length() > 0)
368           {
369             count++;
370             //System.out.println(line);
371             proteintable.put(line, i);
372           }
373         }
374         br.close();
375         size = proteintable.size();
376
377         Set keys = proteintable.keySet();
378         Iterator it = keys.iterator();
379
380         while(it.hasNext())
381         {
382           String key = (String)it.next();
383           pri.println(key);
384         }
385         pri.close();
386
387         System.out.println("Antall unike: " + size);
388         System.out.println("Fjernede elementer: " + (count-size));
389         System.out.println("Prosent av opprinnelig fil: "+ (double)size/count);
390
391       }
392       catch(Exception e)
393       {
394         e.printStackTrace();
395       }
396
397     }
398
399     /*
400      *Regular expression-metode som fjerner diverse entries.
401      */
402     private void removeEntries()
403     {
404       BufferedReader br = null;
405       PrintWriter pri = null;
406     try{
407       br = new BufferedReader(new FileReader(new File("kournikova.txt")));
408       pri = new PrintWriter(new BufferedWriter(new FileWriter(new File("UPDATED-
            DICTIONARY.txt"),false)));
409       }
410     catch(Exception e){}
411
412       try{
413         Integer i = new Integer(0);
414         int countTotalLines = 0;
415         int countLinesInUpdatedFile = 0;
416         String line;
417
418         while( (line = br.readLine()) != null )
419         {
```

```
420         //den skriver ut annenhver linje tom pga. \r -tegnet, derfor må teste
              lengde større enn 0
421         if(line.length() > 0)
422         {
423           countTotalLines++;
424
425           String numbers1 = "[\\d]+";
426           String numbers2 = "[\\D]+";
427           String oneletter = "[a-zA-Z]{1}";
428           String kda = "\\d{1,4}\\p{Punct}{0,1}\\d{0,4}\\s(kDa){1}\\s(protein){1}
                ";
429           String predicted = "(PREDICTED:)\\s(hypothetical).*";
430           String uniref = ".*\\s(UniRef100)\\s(entry).*";
431           String hypothetical = "(H|h)(ypothetical).*";
432           String predicted2 = "(PREDICTED:).*";
433           String decimal = "\\d*\\p{Punct}{0,1}\\d*";
434           String similar = "(S|s)(imilar)\\s(to).*";
435
436           //hvis linja ikke er rene tall, skriv til fil
437
438            if( !((line.matches(numbers1) && (! line.matches(numbers2)))) )
439            {
440            countLinesInUpdatedFile++;
441            pri.println(line);
442            }
443            /*
444            //hvis linja er en bokstav som "A"
445             if(!line.matches(oneletter))
446             {
447             countLinesInUpdatedFile++;
448             printwriter.println(line);
449             }
450
451             //Hvis linje er på form "23 kDa protein" el. "23.3 kDa protein"
452              if(!line.matches(kda))
453              {
454              countLinesInUpdatedFile++;
455              printwriter.println(line);
456              }
457
458              //Hvis linje på form "PREDICTED: hypothetical.."
459               if(!line.matches(predicted))
460               {
461               countLinesInUpdatedFile++;
462               printwriter.println(line);
463               }
464
465
466               //Hvis linje på form ".. UniRef100 entry.."
467                if(!line.matches(uniref))
468                {
469                countLinesInUpdatedFile++;
470                printwriter.println(line);
471                }
472
473
474                //Hvis linje på form "..hypothetical.."
475                 if(!line.matches(hypothetical))
476                 {
477                 countLinesInUpdatedFile++;
478                 printwriter.println(line);
479                 }
```

```
480
481           //Hvis linje på form PREDICTED:....
482           if(!line.matches(predicted2))
483           {
484             countLinesInUpdatedFile++;
485             printwriter.println(line);
486           }
487
488           //Hvis 9.8
489           if(!line.matches(decimal))
490           {
491             countLinesInUpdatedFile++;
492             proteintable.put(line, i);
493             printwriter.println(line);
494           }
495
496
497           //Hvis Similar to:....
498           if(!line.matches(similar))
499           {
500             countLinesInUpdatedFile++;
501             proteintable.put(line, i);
502             printwriter.println(line);
503           }
504           */
505         }
506       }
507       br.close();
508       pri.close();
509
510       //legge til proteintable til dictionary-fila
511       //System.out.println(proteintable.size());
512       //String proteinnames_to_dictionary = proteintable.toString();
513       //dictionaryPrintWriter.println(proteinnames_to_dictionary);
514
515       System.out.println("Antall linjer: " + countTotalLines);
516       System.out.println("Fjernede linjer: " + (countTotalLines-
            countLinesInUpdatedFile));
517       System.out.println("Prosent av opprinnelig fil: "+ (double)
            countLinesInUpdatedFile/countTotalLines);
518
519
520     }catch(Exception e)
521     {
522       e.printStackTrace();
523     }
524
525   }
526
527   /*
528    *Testmetode.
529    */
530   private void testremoveEntries()
531   {
532     try{
533       int countTotalLines = 0;
534       int countLinesWithoutNumbers = 0;
535       String line;
536
537       while( (line = br3.readLine()) != null )
538       {
```

```
539           //den skriver ut annenhver linje tom pga. \r -tegnet, derfor må teste
                lengde større enn 0
540           if(line.length() > 0)
541           {
542             countTotalLines++;
543             if(line.matches("(H|h)(ypothetical).*"))
544             {
545               System.out.println("treff");
546             }
547
548           }
549         }
550         br3.close();
551         pw3.close();
552         System.out.println("Antall linjer: " + countTotalLines);
553         System.out.println("Fjernede linjer: " + (countTotalLines-
              countLinesWithoutNumbers));
554         System.out.println("Prosent av opprinnelig fil: "+ (double)
              countLinesWithoutNumbers/countTotalLines);
555
556
557       }catch(Exception e)
558       {
559         e.printStackTrace();
560       }
561
562     }
563
564   }
```

LookUp.java

```
1  package master;
2
3  import java.io.BufferedReader;
4  import java.io.BufferedWriter;
5  import java.io.File;
6  import java.io.FileReader;
7  import java.io.FileWriter;
8  import java.io.IOException;
9  import java.io.InputStreamReader;
10 import java.io.PrintWriter;
11 import java.net.URL;
12 import java.net.URLConnection;
13 import java.util.ArrayList;
14 import java.util.Hashtable;
15 import java.util.Iterator;
16 import java.util.Set;
17 import java.util.regex.Matcher;
18 import java.util.regex.Pattern;
19
20 /*
21  *Denne klassen skal foreta selve oppslaget i
22  *protein og gen-dicionaries. Den får som input
23  *ArrayListen finalList som inneholder ferdig
24  *preprosesserte tokens fra n abstracts.
25  *Den kaller Dictionary-klassen.
26  *Den får også som input pmid-strengen, GUI-objektet, en peker
27  *til PreProcessing, en ArrayList med de taggede proteinene,
28  *en long-variabel som inneholder start-tiden
29  *og arraylisten tokFreq som inneholder alle abstractene
30  *sortert på nøkkel.
31  */
32
33 /**
34  *@author skuland
35  */
36
37 public class LookUp {
38
39   private String pmid;
40   private PrintWriter pw;
41   private ArrayList finalList, tokFreq;
42   private ArrayList superGeneList, superProteinList, superList;
43   private Dictionary d;
44   private String  token = "", tokenP1 = "", tokenP2 = "", tokenP3 = "", tokenP4 =
       "", tokenP5 = "", tokenP6 = "", tokenP7 = "", tokenP8 = "", tokenP9 = "";
45   private GUI gui;
46   private PreProcessing pp;
47   private ArrayList relevantList;
48   private ArrayList relevantBioTermsList;
49   private ArrayList allBioTermHits;
50   private ArrayList yapexTaggedProteins;
51   private ArrayList sortedKeys, sortedFrequencies;
52   private ArrayList postProcessingStopwords;
53   private BufferedReader preader;
54   private Hashtable ptable;
55
56   public LookUp(String pmid, ArrayList finalList, GUI gui, ArrayList tokFreq,
       PreProcessing pp, ArrayList yapexTaggedProteins, long now)
```

```
57    {
58      this.pmid = pmid;
59      this.finalList = finalList;
60      this.gui = gui;
61      this.pp = pp;
62      this.tokFreq = tokFreq;
63      this.yapexTaggedProteins = yapexTaggedProteins;
64      superGeneList = new ArrayList();
65      superProteinList = new ArrayList();
66      superList = new ArrayList();
67      allBioTermHits = new ArrayList();
68
69      sortedKeys = new ArrayList();
70      sortedFrequencies = new ArrayList();
71
72      initializeStopwordsArray();
73
74      d = new Dictionary(false);
75      //printGoldenStandard();
76      Weight w = new Weight();
77
78      //beregner de tre mest relevante termer generelt
79      ArrayList a = w.calculateWeight(tokFreq);
80      relevantList = w.extractTopThreeMostRelevant(a);
81
82
83      /*
84      *Kommentert ut for å kjøre Test
85      *
86      *insertRelevantTermsToGUI(relevantList);
87      */
88
89      //leser abstractene og returnerer bioterm-treffene
90      ArrayList b = readAbstract(finalList);
91
92      System.out.println("allBioTermHits.size before stopword-removal: " +
          allBioTermHits.size());
93
94      //Fjerner stoppord
95      Stopwords stop = new Stopwords();
96      allBioTermHits = stop.removeStopwords(allBioTermHits);
97      System.out.println("allBioTermHits.size etter stopword-removal: " +
          allBioTermHits.size());
98      removeBioTermEntries();
99      System.out.println("allBioTermHits : Ny størrelse: "+allBioTermHits.size());
100     sortBioTerms();
101     System.out.println("allBioTermHits : Ny størrelse: "+allBioTermHits.size());
102
103     ArrayList allBioTermHitsClone = (ArrayList)allBioTermHits.clone();
104     ArrayList yapexTaggedProteinsClone = (ArrayList)yapexTaggedProteins.clone();
105
106     //sammenligne de to arraylistene allBioTermHits og yapexTaggedProteins for å
          finne recall og precision
107     int answerSetSize = allBioTermHitsClone.size();
108     int relevantSetSize = yapexTaggedProteinsClone.size();
109     int measurementcounter = 0;
110
111     //for all retrieved names
112     for(int size = 0; size < allBioTermHits.size();size++)
113     {
114       //for all names in relevant set
115       for(int u=0;u<yapexTaggedProteins.size();u++)
```

```
116         {
117           if( ((String)yapexTaggedProteins.get(u)).equals( (String)allBioTermHits.
              get(size) ) )
118           {
119             measurementcounter++;
120             allBioTermHits.remove(size);
121             yapexTaggedProteins.remove(u);
122             if(size != 0)
123             {
124               size--;
125             }
126             if(u !=0)
127             {
128               u--;
129             }
130           }
131         }
132       }
133
134       double precision = (double)measurementcounter/answerSetSize;
135       double recall = (double)measurementcounter/ relevantSetSize;
136
137       System.out.println("PRECISION: "+precision);
138       System.out.println("RECALL: "+ recall);
139
140       //sender superGeneList og superProteinList til sortering
141       ArrayList b0 = (ArrayList)b.get(0);
142       ArrayList b1 = (ArrayList)b.get(1);
143
144       /*
145       *geneSortedList og protSortedList skal nå inneholde
146       *sortedKeys, sortedFrequencies - AL-par
147       */
148       ArrayList geneSortedList = pp.sortTokensByFrequency(b0);
149       ArrayList protSortedList = pp.sortTokensByFrequency(b1);
150
151       ArrayList resultList = new ArrayList();
152       resultList.add(geneSortedList);
153       resultList.add(protSortedList);
154       System.out.println("JENNIFER");
155
156       /*
157        *Kommenterte ut for å kjøre Test
158        *
159        *Result r = new Result(resultList, gui);
160       */
161
162       ArrayList c = w.calculateWeight(geneSortedList);
163       ArrayList d = w.calculateWeight(protSortedList);
164
165       /*
166       *Kommentert ut for å kjøre Test
167       *
168       //beregner de fem mest relevante biotermer
169       relevantBioTermsList = w.calculateFiveMostRelevantBioTerms(c,d);
170       insertRelevantBioTermsToGUI(relevantBioTermsList);
171       */
172
173       long after = System.currentTimeMillis();
174       System.out.println("Timeconsumption: "+(double)(after-now)/1000);
175
176     }
```

```
177
178   /*
179    *Denne metoden initialiserer postprocessing stoppordlisten
180    *som består av 300 termer
181    */
182   private void initializeStopwordsArray()
183   {
184     BufferedReader br = null;
185
186     try{
187     br = new BufferedReader(new FileReader(new File("postProcessStopwords.txt")))
        ;
188     }
189     catch(Exception e){}
190
191     postProcessingStopwords = new ArrayList();
192
193     try{
194     String line = "";
195     while( (line = br.readLine()) != null )
196     {
197       if(line.length() > 0)
198       {
199         postProcessingStopwords.add(line);
200       }
201
202     }
203
204     }catch(Exception e){}
205
206     System.out.println("Størrelse på stopwordsAL: "+postProcessingStopwords.size
        ());
207
208   }
209
210   /*
211    *Denne metoden sorterer de identifiserte
212    *biotermene ved hjelp av quicksort.
213    */
214   private void sortBioTerms()
215   {
216
217     PrintWriter pri = null;
218     try{
219       pri = new PrintWriter(new BufferedWriter(new FileWriter(new File("
          myBioTermsFrequency.txt"),false)));
220     }
221     catch(Exception e){}
222
223     Hashtable ht = new Hashtable();
224     int j = 0;
225     Integer i = new Integer(1);
226     for(int c = 0; c < allBioTermHits.size(); c++)
227     {
228       if(ht.containsKey( (String)allBioTermHits.get(c) ) )
229       {
230         Integer u = (Integer)ht.get((String)allBioTermHits.get(c));
231         j= u.intValue();
232         j++;
233         u = new Integer(j);
234         ht.put((String)allBioTermHits.get(c) , u);
235
```

```
236         }
237         else
238         {
239           ht.put((String)allBioTermHits.get(c),i);
240         }
241
242       }
243
244       System.out.println("Antall unike nøkler: "+ht.size());
245       Set s = ht.keySet();
246       Iterator it = s.iterator();
247       int n = 0;
248       int[] frequency = new int[ht.size()];
249       while(it.hasNext())
250       {
251         String bio = ((String)it.next());
252         Integer fr = (Integer)ht.get(bio);
253         int a = fr.intValue();
254
255         frequency[n] = a;
256         n++;
257
258
259       }
260       try{
261         sort(frequency);
262         }
263         catch(Exception ex){ex.printStackTrace();}
264
265         Set newset = ht.keySet();
266         Iterator newit = newset.iterator();
267
268         Integer io = new Integer(0);
269         String xyz = "";
270         for(int xy = 0; xy < newset.size(); xy++)
271         {
272         sortedKeys.add(xyz);
273         sortedFrequencies.add(io);
274         }
275         int test = 0;
276         boolean found = false;
277         //For alle mappinger i HashMap tokensCounted
278         while(newit.hasNext())
279         {
280           test++;
281           found = false;
282           String m = ((String)newit.next());
283           Integer e = (Integer)ht.get(m);
284           int a = e.intValue();
285           if(!found){
286             //itererer over frequency-arrayet
287             for(int k = 0; k < frequency.length;k++)
288             {
289               if(!found){
290                 //Hvis verdi a fra tokensCounted er lik verdi i frequency-array
291                 if(a == frequency[k])
292                 {
293                   sortedKeys.set(k,m);
294                   sortedFrequencies.set(k,e);
295                   frequency[k]=0;
296                   found = true;
297                 }
```

```
298             }
299           }
300         }
301       }
302
303       for(int y = 0; y< sortedKeys.size(); y++)
304       {
305         pri.println((String)sortedKeys.get(y) +" " + "," + " "+ (Integer)
              sortedFrequencies.get(y));
306       }
307     pri.close();
308   }
309
310   /*
311    *Metode som printer til en goldenstandard-tekstfil
312    */
313   private void printGoldenStandard()
314   {
315     PrintWriter printx=null;
316     try{
317       printx = new PrintWriter(new BufferedWriter(new FileWriter(new File("
            goldenstandard.txt"),false)));
318     }catch(IOException io){}
319
320     for(int e = 0; e<yapexTaggedProteins.size();e++)
321     {
322       printx.println((String)yapexTaggedProteins.get(e));
323     }
324     printx.close();
325   }
326
327   /*
328    *En I/O-metode.
329    */
330   private void printMyBioTermHits()
331   {
332     /*
333      *Her inneholder nå allBioTermHits alle protein/gen-treff
334      *som systemet fant gjennom test på Yapex-teksten.
335      *Sammenligner de med yapexTaggedProteins for å få målinger
336      *på recall og precision.
337      */
338     PrintWriter printy = null;
339     try{
340       printy = new PrintWriter(new BufferedWriter(new FileWriter(new File("
            myBioTerms.txt"),false)));
341     }catch(IOException io){}
342     for(int e = 0; e<allBioTermHits.size();e++)
343     {
344       printy.println((String)allBioTermHits.get(e));
345
346     }
347     printy.close();
348   }
349
350   /*
351   *Denne metoden skaper en one-level hash table
352   *av proteinnavnene. Her må en hardkode dersom
353   *man ønsker at systemet skal benytte stemming
354   *istedet for ikke stemming. Denne one-level
355   *hash tabellen brukes for å fjerne irrelevante
356   *treff av lengde 1 ord
```

```
357   */
358   private void pHT()
359   {
360     try{
361       ClassLoader cl = this.getClass().getClassLoader();
362       URL pURL = cl.getResource("final_dictionaries/proteinDictionary2.txt");
363       URLConnection pConnection = pURL.openConnection();
364       pConnection.setDoInput(true);
365       pConnection.connect();
366       preader = new BufferedReader(new InputStreamReader(pConnection.
            getInputStream()));
367     }catch(Exception e){}
368     ptable = new Hashtable();
369     Integer i = new Integer(0);
370     String line = "";
371     try{
372       while( (line = preader.readLine()) != null )
373       {
374         if(line.length() > 0)
375         {
376           ptable.put(line, i);
377         }
378
379       }
380       System.out.println("One-level protein hashtable size: "+ptable.size());
381
382     }catch(IOException io){io.printStackTrace();}
383   }
384
385   /*
386    *Metoden kaller pHT og skaper one-level hash tabellen.
387    *Deretter fjerner den en rekke innslag fra settet
388    *med antatte protein navn.
389    */
390   private void removeBioTermEntries()
391   {
392     pHT();
393
394     int number = 0;
395     int number1 = 0;
396     int number2 = 0;
397     int number3 = 0;
398     int number4 = 0;
399     int number5 = 0;
400
401     Pattern p = Pattern.compile("\\d+"); //rene tallinnslag
402     Pattern p1 = Pattern.compile("\\D{1,2}"); //et ikke-tall-innslag 1 til 2 tegn
          langt
403     Pattern p2 = Pattern.compile("\\d*(\\u002E)\\d*");//desimaltall
404     Pattern p3 = Pattern.compile("\\d+\\s(kda)(\\s((P|p)rotein))?.*");//kda
          innslag
405
406     for(int v = 0; v<allBioTermHits.size();v++)
407     {
408       String bioTerm = (String)allBioTermHits.get(v);
409
410       String[]t = bioTerm.split("\\s");
411       int numberoftokens = t.length;
412
413       Matcher m = p.matcher(bioTerm);
414       Matcher m1 = p1.matcher(bioTerm);
415       Matcher m2 = p2.matcher(bioTerm);
```

```
416         Matcher m3 = p3.matcher(bioTerm);
417
418         if(m.matches())
419         {
420           number++;
421           allBioTermHits.remove(v);
422           v--;
423         }
424         else if(m1.matches())
425         {
426           number1++;
427           allBioTermHits.remove(v);
428           v--;
429         }
430         else if(m2.matches())
431         {
432           number3++;
433           allBioTermHits.remove(v);
434           v--;
435         }
436         else if(m3.matches())
437         {
438           number5++;
439           allBioTermHits.remove(v);
440           v--;
441         }
442         else if(numberoftokens==1 && !ptable.containsKey(bioTerm))
443         {
444           number4++;
445           allBioTermHits.remove(v);
446           v--;
447         }
448         else{
449         for(int q = 0; q < postProcessingStopwords.size();q++)
450         {
451           if(bioTerm.equals((String)postProcessingStopwords.get(q)))
452           {
453             number2++;
454             allBioTermHits.remove(v);
455             v--;
456           }
457         }
458         }
459       }
460
461       System.out.println("Antall rene tallinnslag fjernet: "+number);
462       System.out.println("Antall 1 til 2 tegn lange ord fjernet: "+number1);
463       System.out.println("Antall ord som matcher postprocstopwords og er fjernet: "
            +number2);
464       System.out.println("Antall desimaltall fjernet: "+number3);
465       System.out.println("Antall 1-ord lange innslag som ikke er selvstendige: "+
            number4);
466       System.out.println("Antall kda-innslag fjernet: "+number5);
467       printMyBioTermHits();
468     }
469
470     /*
471      *Metode som setter inn de fem mest relevante
472      *biotermene i GUIen.
473      */
474     private void insertRelevantBioTermsToGUI(ArrayList relevantBioTermsList)
475     {
```

```
476       String one = (String)relevantBioTermsList.get(0);
477       String two = (String)relevantBioTermsList.get(1);
478       String three = (String)relevantBioTermsList.get(2);
479       String four = (String)relevantBioTermsList.get(3);
480       String five = (String)relevantBioTermsList.get(4);
481
482       gui.insertTopFiveRelevantBioTerms(one,0);
483       gui.insertTopFiveRelevantBioTerms(two,1);
484       gui.insertTopFiveRelevantBioTerms(three,2);
485       gui.insertTopFiveRelevantBioTerms(four,3);
486       gui.insertTopFiveRelevantBioTerms(five,4);
487     }
488
489     /*
490      *Denne metoden setter inn de tre mest relevante termene
491      *funnet i abstractene, gjennom Weight-klassen til GUIen.
492      */
493     private void insertRelevantTermsToGUI(ArrayList relevantList)
494     {
495       String first = (String)relevantList.get(0);
496       String second = (String)relevantList.get(1);
497       String third = (String)relevantList.get(2);
498
499       gui.insertTopThreeRelevantTerms(first,0);
500       gui.insertTopThreeRelevantTerms(second,1);
501       gui.insertTopThreeRelevantTerms(third,2);
502     }
503
504     /*
505      *Dette er selve kjerne-metoden i systemet.
506      *Metoden får inn en ArrayList med ferdige preprosesserte
507      *abstracts. Disse blir så kjørt gjennom en lookup
508      *prosess, hvor antatte protein eller gen navn
509      *blir tatt vare på i respektive ArrayLister. Metoden
510      *returner en ArrayList superList som inneholder
511      *ArrayLister med gen og protein-treff for samtlige
512      *abstracts.
513      */
514     public ArrayList readAbstract(ArrayList finalList)
515     {
516       String s = " ";
517       int countGeneHits=0;
518       int countProteinHits=0;
519
520       int length2 = 0;
521       int length3 = 0;
522       int length4 = 0;
523       int length5 = 0;
524       int length6 = 0;
525       int length7 = 0;
526       int length8 = 0;
527       int length9 = 0;
528       int length10 = 0;
529
530       boolean outOfIndex = false;
531       ArrayList geneHits = new ArrayList();
532       ArrayList proteinHits = new ArrayList();
533
534       System.out.println("finalList.size: "+finalList.size());
535       for(int q = 0; q < finalList.size(); q++)
536       {
537         int geneHit = 0;
```

```
538         int proteinHit =0;
539         int bioTermHit=0;
540
541         ArrayList abstractText = (ArrayList)finalList.get(q);
542
543         for(int a = 0; a < abstractText.size(); a++)
544         {
545           outOfIndex = false;
546           token = (String)abstractText.get(a);
547
548           if(a < abstractText.size()-1){ tokenP1 = (String)abstractText.get(a+1);}
549           else{outOfIndex = true;}
550
551           if(a < abstractText.size()-2){ tokenP2 = (String)abstractText.get(a+2);}
552           else{outOfIndex = true;}
553
554           if(a < abstractText.size()-3){ tokenP3 = (String)abstractText.get(a+3);}
555           else{outOfIndex = true;}
556
557           if(a < abstractText.size()-4){ tokenP4 = (String)abstractText.get(a+4);}
558           else{outOfIndex = true;}
559
560           if(a < abstractText.size()-5){ tokenP5 = (String)abstractText.get(a+5);}
561           else{outOfIndex = true;}
562
563           if(a < abstractText.size()-6){ tokenP6 = (String)abstractText.get(a+6);}
564           else{outOfIndex = true;}
565
566           if(a < abstractText.size()-7){ tokenP7 = (String)abstractText.get(a+7);}
567           else{outOfIndex = true;}
568
569           if(a < abstractText.size()-8){ tokenP8 = (String)abstractText.get(a+8);}
570           else{outOfIndex = true;}
571
572           if(a < abstractText.size()-9){ tokenP9 = (String)abstractText.get(a+9);}
573           else{outOfIndex = true;}
574
575           boolean gene = performGeneDictionaryLookup(token);
576           boolean protein = performProteinDictionaryLookup(token, 0);
577           //Hvis gennavn, adder til gen-navnliste
578           if(gene)
579           {
580             geneHit++;
581             bioTermHit++;
582             geneHits.add(token);
583
584             //allBioTermHits er laget for å kjøre Test
585             allBioTermHits.add(token);
586           }
587           //Hvis proteinnavn, adder til protein-navnliste
588           //undersøk om neste ord er på neste nivå i dictionary osv..
589           else if(protein)
590           {
591             String tokenString = "";
592             proteinHits.add(proteinHit, token);
593             tokenString = token;
594
595             allBioTermHits.add(bioTermHit,tokenString);
596
597             //2.ord i navnet
598             if( !outOfIndex && performProteinDictionaryLookup(tokenP1,1))
599             {
```

```
600             //System.out.println("Fått treff på andre nivå");
601             length2++;
602             proteinHits.remove(proteinHit);
603             proteinHits.add(proteinHit, token+s+tokenP1);
604             a++;
605             tokenString = token+s+tokenP1;
606
607             allBioTermHits.remove(bioTermHit);
608             allBioTermHits.add(bioTermHit,tokenString);
609
610             //3.ord
611             if( !outOfIndex && performProteinDictionaryLookup(tokenP2,2))
612             {
613               //System.out.println("Fått treff på tredje nivå");
614               length3++;
615               proteinHits.remove(proteinHit);
616               proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2);
617               a++;
618               tokenString = token+s+tokenP1+s+tokenP2;
619
620               allBioTermHits.remove(bioTermHit);
621               allBioTermHits.add(bioTermHit,tokenString);
622
623               //4.ord
624               if( !outOfIndex && performProteinDictionaryLookup(tokenP3,3))
625               {
626                 //System.out.println("Fått treff på fjerde nivå");
627                 length4++;
628                 proteinHits.remove(proteinHit);
629                 proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2+s+tokenP3);
630                 a++;
631                 tokenString = token+s+tokenP1+s+tokenP2+s+tokenP3;
632                 //System.out.println("4.nivaa: "+tokenString);
633
634                 allBioTermHits.remove(bioTermHit);
635                 allBioTermHits.add(bioTermHit,tokenString);
636
637                 //5.ord
638                 if( !outOfIndex && performProteinDictionaryLookup(tokenP4,4))
639                 {
640                   length5++;
641                   proteinHits.remove(proteinHit);
642                   proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2+s+tokenP3
                      +s+tokenP4);
643                   a++;
644                   tokenString = token+s+tokenP1+s+tokenP2+s+tokenP3+s+tokenP4;
645
646                   allBioTermHits.remove(bioTermHit);
647                   allBioTermHits.add(bioTermHit,tokenString);
648
649                   //6.ord
650                   if( !outOfIndex && performProteinDictionaryLookup(tokenP5,5))
651                   {
652                     length6++;
653                     proteinHits.remove(proteinHit);
654                     proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2+s+
                        tokenP3+s+tokenP4+s+tokenP5);
655                     a++;
656                     tokenString = token+s+tokenP1+s+tokenP2+s+tokenP3+s+tokenP4+s
                        +tokenP5;
657
658                     allBioTermHits.remove(bioTermHit);
```

```
659                     allBioTermHits.add(bioTermHit,tokenString);
660
661                     //7.ord
662                     if( !outOfIndex && performProteinDictionaryLookup(tokenP6,6))
663                     {
664                       length7++;
665                       proteinHits.remove(proteinHit);
666                       proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2+s+
                          tokenP3+s+tokenP4+s+tokenP5+s+tokenP6);
667                       a++;
668                       tokenString = token+s+tokenP1+s+tokenP2+s+tokenP3+s+tokenP4
                          +s+tokenP5+s+tokenP6;
669
670                       allBioTermHits.remove(bioTermHit);
671                       allBioTermHits.add(bioTermHit,tokenString);
672
673                       //8.ord
674                       if( !outOfIndex && performProteinDictionaryLookup(tokenP7
                          ,7))
675                       {
676                         length8++;
677                         proteinHits.remove(proteinHit);
678                         proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2+s+
                            tokenP3+s+tokenP4+s+tokenP5+s+tokenP6+s+tokenP7);
679                         a++;
680                         tokenString = token+s+tokenP1+s+tokenP2+s+tokenP3+s+
                            tokenP4+s+tokenP5+s+tokenP6+s+tokenP7;
681
682                         allBioTermHits.remove(bioTermHit);
683                         allBioTermHits.add(bioTermHit,tokenString);
684
685                         //9.ord
686                         if( !outOfIndex && performProteinDictionaryLookup(tokenP8
                            ,8))
687                         {
688                           length9++;
689                           proteinHits.remove(proteinHit);
690                           proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2+s
                              +tokenP3+s+tokenP4+s+tokenP5+s+tokenP6+s+tokenP7+s+
                              tokenP8);
691                           a++;
692                           tokenString = token+s+tokenP1+s+tokenP2+s+tokenP3+s+
                              tokenP4+s+tokenP5+s+tokenP6+s+tokenP7+s+tokenP8;
693
694                           allBioTermHits.remove(bioTermHit);
695                           allBioTermHits.add(bioTermHit,tokenString);
696
697                           //10.ord
698                           if( !outOfIndex && performProteinDictionaryLookup(
                              tokenP9,9))
699                           {
700                             length10++;
701                             proteinHits.remove(proteinHit);
702                             proteinHits.add(proteinHit, token+s+tokenP1+s+tokenP2
                                +s+tokenP3+s+tokenP4+s+tokenP5+s+tokenP6+s+tokenP7+
                                s+tokenP8+s+tokenP9);
703                             a++;
704                             tokenString = token+s+tokenP1+s+tokenP2+s+tokenP3+s+
                                tokenP4+s+tokenP5+s+tokenP6+s+tokenP7+s+tokenP8+s+
                                tokenP9;
705
706                             allBioTermHits.remove(bioTermHit);
```

```
707                             allBioTermHits.add(bioTermHit,tokenString);
708
709                           }
710                         }
711                       }
712                     }
713                   }
714                 }
715               }
716             }
717           }
718           proteinHit++;
719           bioTermHit++;
720         }
721         //Ordet er uinteressant
722         else
723         {
724
725         }
726       }
727
728       ArrayList geneHitsClone = (ArrayList)geneHits.clone();
729       ArrayList proteinHitsClone = (ArrayList)proteinHits.clone();
730
731       countGeneHits = countGeneHits+geneHitsClone.size();
732       countProteinHits = countProteinHits+proteinHitsClone.size();
733
734       superGeneList.add(geneHitsClone);
735       superProteinList.add(proteinHitsClone);
736
737       geneHits.clear();
738       proteinHits.clear();
739
740
741
742     }
743
744     System.out.println("allBioTermHits.size: "+allBioTermHits.size());
745
746     System.out.println("Antall ord med lengde 2: "+length2);
747     System.out.println("Antall ord med lengde 3: "+length3);
748     System.out.println("Antall ord med lengde 4: "+length4);
749     System.out.println("Antall ord med lengde 5: "+length5);
750     System.out.println("Antall ord med lengde 6: "+length6);
751     System.out.println("Antall ord med lengde 7: "+length7);
752     System.out.println("Antall ord med lengde 8: "+length8);
753     System.out.println("Antall ord med lengde 9: "+length9);
754     System.out.println("Antall ord med lengde 10: "+length10);
755
756     superList.add(superGeneList);
757     superList.add(superProteinList);
758
759     return superList;
760
761   }
762
763
764   /*
765    *Denne metoden foretar Lookup i protein-dictionary.
766    *Returnerer en boolean som forteller om det
767    *aktuelle token er del av et proteinnavn eller ikke.
768    */
```

```
769   public boolean performProteinDictionaryLookup(String possibleProtein, int
        hashTableLevel)
770   {
771     boolean protein = false;
772
773     switch(hashTableLevel)
774     {
775     case 0:   protein = d.proteinDictionary_level_0.containsKey(possibleProtein);
776               break;
777
778     case 1:   Hashtable intern0 = (Hashtable)d.proteinDictionary_level_0.get(
          token);
779               protein = intern0.containsKey(possibleProtein);
780               break;
781
782     case 2:   Hashtable intern1 = (Hashtable)d.proteinDictionary_level_1.get(
          tokenP1);
783               protein = intern1.containsKey(possibleProtein);
784               break;
785
786     case 3:   Hashtable intern2 = (Hashtable)d.proteinDictionary_level_2.get(
          tokenP2);
787               protein = intern2.containsKey(possibleProtein);
788               break;
789
790     case 4 :  Hashtable intern3 = (Hashtable)d.proteinDictionary_level_3.get(
          tokenP3);
791               protein = intern3.containsKey(possibleProtein);
792               break;
793
794     case 5:   Hashtable intern4 = (Hashtable)d.proteinDictionary_level_4.get(
          tokenP4);
795               protein = intern4.containsKey(possibleProtein);
796               break;
797
798     case 6:   Hashtable intern5 = (Hashtable)d.proteinDictionary_level_5.get(
          tokenP5);
799               protein = intern5.containsKey(possibleProtein);
800               break;
801
802     case 7:   Hashtable intern6 = (Hashtable)d.proteinDictionary_level_6.get(
          tokenP6);
803               protein = intern6.containsKey(possibleProtein);
804               break;
805
806     case 8:   Hashtable intern7 = (Hashtable)d.proteinDictionary_level_7.get(
          tokenP7);
807               protein = intern7.containsKey(possibleProtein);
808               break;
809
810     case 9:   Hashtable intern8 = (Hashtable)d.proteinDictionary_level_8.get(
          tokenP8);
811               protein = intern8.containsKey(possibleProtein);
812               break;
813
814     default : protein = false;
815               break;
816     }
817     return protein;
818   }
819
820   /*
```

```
821     *Denne metoden foretar lookup i gen-dictionary.
822     *Returnerer en boolean som indikerer om
823     *tokenet er et gen.
824     */
825    public boolean performGeneDictionaryLookup(String possibleGene)
826    {
827      //System.out.println("geneDictionary er lik null: "+d.geneDictionary.equals(
         null));
828      boolean gene = d.geneDictionary.containsKey(possibleGene);
829      return gene;
830    }
831
832    /*
833     *Metodene nedenfor er del av quicksort-algoritmen.
834     */
835    private void quickSort(int a[], int l, int r) throws Exception
836    {
837            int M = 4;
838            int i;
839            int j;
840            int v;
841
842            if ((r-l)>M)
843            {
844                    i = (r+l)/2;
845                    if (a[l]>a[i]) swap(a,l,i);     // Tri-Median Methode!
846                    if (a[l]>a[r]) swap(a,l,r);
847                    if (a[i]>a[r]) swap(a,i,r);
848
849                    j = r-1;
850                    swap(a,i,j);
851                    i = l;
852                    v = a[j];
853                    for(;;)
854                    {
855                            while(a[++i]<v);
856                            while(a[--j]>v);
857                            if (j<i) break;
858                            swap (a,i,j);
859
860
861                    }
862                    swap(a,i,r-1);
863
864                    quickSort(a,l,j);
865                    quickSort(a,i+1,r);
866            }
867        }
868
869        private void swap(int a[], int i, int j)
870        {
871                int T;
872                T = a[i];
873                a[i] = a[j];
874                a[j] = T;
875        }
876
877        private void insertionSort(int a[], int lo0, int hi0) throws Exception
878        {
879                int i;
880                int j;
881                int v;
```

```
882
883                 for (i=lo0+1;i<=hi0;i++)
884                 {
885                         v = a[i];
886                         j=i;
887                         while ((j>lo0) && (a[j-1]>v))
888                         {
889                                 a[j] = a[j-1];
890                                 j--;
891                         }
892                         a[j] = v;
893                 }
894         }
895
896         public void sort(int a[]) throws Exception
897         {
898                 quickSort(a, 0, a.length - 1);
899                 insertionSort(a,0,a.length-1);
900         }
901 }
```

References

[1] *Modern Information Retrieval*, Ricardo Baeza-Yates, Berthier Ribeiro-Neto, ISBN: 0-201-39829-X, 1999

[2] *Ranking for BioMinT: Investigating Performance, Local Search and Homonymy Recognition*, Alexander K. Seewald, Austrian Research Institute for Artificial Intelligence, Freyung 6/6, A-1010 Vienna, Austria

[3] *Text REtrieval Conference (TREC) Home Page*, http://trec.nist.gov/

[4] *Indri at TREC 2004: UMass Terabyte Track Overview*, http://ciir.cs.umass.edu

[5] *Data Mining Notes*, http://www.pcc.qub.ac.uk/tec/courses/datamining

[6] *Analysis of Protein/Protein Interactions Through Biomedical Literature: Text Mining of Abstracts vs. Text Mining of Full Text Articles*, Martin et al., 2004

[7] *Text-mining approaches in molecular biology and biomedicine*, Krallinger et al., March 2005

[8] *Tough Mining*, Steven Dickman, PLoS Biology

[9] *CENDI PA Meeting April 2000*, http://cendi.dtic.mil/minutes/pa_0400.html

[10] *PubMed Overview*, http://www.ncbi.nlm.nih.gov/entrez/query/static/overview.html#Medline

[11] *Medical Subject Headings (MESH) Fact Sheet*, http://www.nlm.nih.gov/pubs/factsheets/mesh.html, 2004

[12] *Protein names and how to find them*, Franzén et al., Stockholm, 2002

[13] *Disambiguating proteins, genes, and RNA in text: a machine learning approach*, Hatzivassiloglou et al., Columbia University, 2001

[14] *A Simple and Practical Dictionary-based Approach for Identification of Proteins in Medline Abstracts*, Egorov et al., Ariadne Genomics, 2004

[15] *Bioinformatics.org*, http://bioinformatics.org/faq/, Fredj Tekaia, Institut Pasteur

[16] *Biology, Seventh Edition*, Raven et al., McGraw-Hill, 2004, ISBN: 0-07-243731-6

[17] *World Wide Web Consortium*, http://www.w3.org

[18] *XMLHistory*, http://www.perfectxml.com/om/XMLHistory.PDF

[19] *What is the Document Object Model?*, http://www.w3.org/TR/DOM-Level-3-Core/introduction.html

[20] *XML DOM Introduction*, http://www.w3schools.com/dom

[21] *Developing Java Software*, Russel Winder og Graham Roberts, 2000, ISBN: 0 471 60696 0

[22] *Java Web Start Overview*, http://java.sun.com/products/javawebstart/overview.html

[23] *Analysis of Biomedical Text for Chemical Names: A Comparison of Three Methods*, W.John Wilbur et al., National Center for Biotechnology Information (NCBI)

[24] *Textpresso: An Ontology-Based Information Retrieval and Extraction System for Biological Literature*, Müller et al. , California Institute of Technology, 2004

[25] *Terminology-driven mining of biomedical literature*, Nenadic et al. , University of Salford, UK, 2002

[26] *Proteinhalt i text*, http://www.sics.se/humle/projects/prothalt/

[27] *Semantic Annotation of Biomedical Literature using Google*, Saetre et al., NTNU, May 2005