CHALMERS

Automatic Searching in Electronic Health Records

Master of Science Thesis in the Programme Computer Science – Algorithms, Languages and Logic

MARGARETHA MICHNIK

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
GÖTEBORG, SWEDEN, JUNE 2010

Automatic Searching in Electronic Health Records

MARGARETHA MICHNIK

Examiner: PETER DAMASCHKE

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2010

Abstract

Modern day patient records consist largely of structured information. Such information makes it easy for healthcare professionals to get an overview of a patient's health status. The patient records also contain a large amount of unstructured data, in other words, descriptions of patients in plain text. This information is much more difficult for the healthcare professionals to find, since there is often a lot of text to read through. At each new hospital visit the patient must often repeat what diseases he or she has. Important information may be lost if the patient forgets to inform or can not inform (for instance an unconscious patient) the staff of a particular disease he or she may suffer from. In this case, it would be helpful to have a program that automatically searches for diseases in a patient record. This master thesis examines if it is feasible that one could introduce automatic searches that would work in the context of retrieving information from electronic health records. Different algorithms were examined to see if they could be a part of the solution to such an automatic search. In addition to the theoretical examinations, a java application was developed in order to test these algorithms. The result shows that an automatic search could work in reality but much further research needs to be done in order to make such a program totally trustworthy. It was unfortunately impossible to access real patient records since they are classified. Therefore, fictitious patient records were used in the testing of the application.

Sammanfattning

Dagens patientjournaler består till stor del av strukturerad information, som gör det lätt för vårdpersonalen att få en överblick över en patient. Men patientjournalerna innehåller även en stor mängd ostrukturerad information, det vill säga beskrivningar om en patient i löpande text. Denna information är mycket mer svåråtkomlig för vårdpersonalen eftersom det ofta kan bli mycket text att gå igenom. Ofta måste patienten upprepa vilka sjukdomar han eller hon har vid varje nytt sjukhusbesök. Viktig information kan gå förlorad om patienten glömmer bort att säga eller inte kan säga, (till exempel en medvetslös patient), om han eller hon har en speciell sjukdom. I sådana fall skulle det vara till hjälp att ha ett program som automatiskt söker efter sjukdomar i en patientjournal. I detta examensarbete studerades det om det är möjligt att utföra automatiska sökningar i elektroniska patientjournaler. Olika algoritmer undersöktes om de skulle kunna vara en del av lösningen för en sådan automatisk sökning. Förutom den teoretiska undersökningen, har även en javaapplikation skrivits för att testa dessa algoritmer. Resultatet visar att en automatisk sökning skulle kunna bli verklighet men mycket kvarstår att undersöka för att få ett sådant program att vara helt och hållet tillförlitligt. Eftersom patientjournaler är hemligstämplade och då det inte fanns möjlighet att få tillgång till dessa under arbetets gång användes istället fiktiva patientjournaler till testningen av applikationen.

Preface

This report is a Master Thesis in Computer Science at the Department of Computer Science and Engineering, Chalmers University of Technology. The master thesis was conducted at Know IT Göteborg AB. I would like to thank my supervisor Fredric Landqvist and Head of Unit Fredrik Abrahamson at the company. I would also like to thank my examiner Peter Damaschke at Chalmers, Dag Wedelin at Chalmers, Dimitrios Kokkinakis at University of Gothenburg and Svetoslav Marinov at Findwise AB who were helpful and responsive in regard to my questions. A special thanks to the medical students Maria Mårin, Sara Fredh, Mattias Svensson, Emelie Wiktorson and the intern Johanna Berg for their contribution of the fictitious records.

The following services were helpful in the process of accomplishing this work: Talbanken05 (Swedish treebank of around 300 000 words), hunpos (an open source hidden Markov model tagger), the web service Nyckelordtjänst developed by Findwise AB and the file with Swedish abbreviations from Öresunds Översättningsbyrå.

Table of Contents

1 Introduction

1.1 Background

A big problem within the Swedish healthcare system is the keeping of electronic health records (EHR). There are several different EHR systems since different practices use different terminologies to describe a patient's health status. The information about a patient in one EHR system may be important for a person who treats that patient but uses another EHR system. The easiest option would be if a patient only had one health record so that all the information about the patient would be in one place. To cover all the different healthcare systems the EHR would have to be very big and that would not be very user-friendly. The solution to this problem would be to let different health departments select the EHR system which meets their needs the best, but at the same time make the EHR systems interoperate with each other. This would make it easier for both the patient and the person who treats the patient. For example, the patient would not have to repeat to the healthcare professionals over and over again what he or she is allergic to or what disease he or she had before or has now. Such things would automatically pop-up when the doctor or nurse searches for information regarding the patient. Such a solution would also be needed when, for instance, a patient who comes unconscious to the hospital needs urgent care. An unconscious patient cannot share what he or she is allergic to and if the doctors and nurses do not find out, there could be terrible consequences. If there was a program which could search through all the patient's old records to find such alerts, the risk of medical errors would decrease.

When considering such a solution, several problems become apparent. The first problem concerns structured and unstructured data in an EHR system. Structured data is data which is easy to interpret. If there is, for example, an attribute called “blood pressure” one may assume that it involves some information about the patient's blood pressure. The unstructured data is all the information which is given in plain text. The problem is to interpret this unstructured data and obtain relevant information from the text. Let us look at an easy example: In one EHR system there is an attribute which lists a patient's allergies and in another system the patient's allergies may appear in plain text. But regardless of where this information is, the healthcare professionals will be alerted about the patient's allergies.

Another problem is that different health departments use different terminologies with different abbreviations. An abbreviation can have different signification at two different health departments. The challenge is to recognize what real medical term corresponds to one specific abbreviation.

1.2 Purpose

The purpose of this work is to find possible algorithms which search through patient records and output any diseases the patients may suffer from. Medical terms from MeSH[1] (Medical Subject Headings) will be used to exemplify possible diseases described in the patient records. “MeSH, Medical Subject Headings, is a list of controlled terms used for indexing, searching and cataloging of biomedical journals.”[2] Only a few medical terms from MeSH are examined and tested in this master thesis. To test all existing medical terms would be impossible within the time constraints of this work. Consequently, the algorithms of this work and the analysis will build on those few terms. A demo application in Java[3] will show how the algorithms can be used practically. This Java application will be tested with fictitious patient records[4][5][6][7] and fictitious sentences which describe various diseases. This is due to the fact that real patient records are classified. Therefore, the tests will unfortunately not be based on real data. However, they will still convey interesting facts about the algorithms in question. The demo application may interpret sentences describing a particular disease and ignore the rest of the sentences. In other words, the application may not interpret the whole text, just the parts that include information of possible relevance for the healthcare professionals.

1.3 Limitations

Since this subject is very extensive some limitations were inevitably needed. One big limitation was to not try to solving this problem so that it suited all the medical terms in MeSH. Only a few diseases' names from MeSH were covered. The work consists of several programming steps and to develop each step from scratch would not have been realistic. Therefore, some already finished programs were used to facilitate the work process. One open source project that was used is hunpos[8] which is part of speech tagger. It tags the words, periods, commas etc. in a text with a corresponding word class. To construct such a program from scratch would have taken too much time from the actual problem. Another open source project that was used in this master thesis is Apelon DTS[9], “(...) that provides comprehensive terminology services in distributed application environments.”[9]. Findwise AB[10] is a company in Gothenburg that has build web services[11] on Apelon DTS. The web service Nyckelordtjänst[11] is one of them and its functions are used in this work. The functions made it possible to search after diseases in MeSH. The use of these functions saved a lot of time. To simplify the problem even more, it is assumed that the sentences do not refer to each other. A sentence is only describing itself.

2 Analysis

To make a computer understand a whole text is very difficult. In this work the computer does not need to understand the whole text. It is sufficient that it can interpret the sentences in the text. So the first thing that must be done is to divide the text into sentences. In section 2.1 it is described how text can be divided into sentences and into even smaller pieces. The computer must only understand a sentence if it is describing a disease. Hence, many words which have nothing to do with the diseases can be eliminated. How to know which words can be eliminated is described in section 2.2. After the computer has divided the text into smaller parts and removed some words, it should be easier for the computer to interpret the sentences. In section 2.3, possible solutions regarding this, are described.

2.1 Tokenization

In order for a computer to understand a text, it must first tokenize the text. That means the text must be divided into sentences and tokens. A token can be a word, period, comma, exclamation mark etc. When tokenizing a text the computer must follow some rules and not just separate words by spaces. Let us look at some Swedish sentences as an example:

> "Maria vann 200 000 kr, hon blev väldigt glad." (Maria won 200 000 kr, she became very happy.)
>
> "Äpplen kan t.ex. vara gröna och gula." (Apples can for example be green and yellow.)
>
> "Vad heter du? frågade Anna." (What is your name? Anna asked.)

If a human being would tokenize this, she would not have any problem recognizing what parts of the sentences constitute a token. But the computer must be able to handle for example the following cases:

kr, This is an abbreviation for the word "kronor" (SEK) without a period, since in the Swedish language there is no period after abbreviations consisting of the first and last letter of the word. [12] This string consists of two tokens: "kr" and "," .

glad. The period following this word shows that the sentence ends. But how will the computer know if it is an abbreviation or an end of a sentence? This is also two tokens.

t.ex. These two punctuation marks signify that this word is an abbreviation of the expression "till exempel" (for example).

du? Here the question mark implies that there is a question, but the sentence continues after the question mark since the word after the mark starts with a lower case.

There are plenty of sequences of characters that a computer does not know how to interpret. The computer does not know that “kr,” is the same type as “kr”. Therefore the computer needs many rules to follow.[13]

When dividing text into sentences the rule is to check if the next word after a sentence terminator such as period, question mark, exclamation mark or semicolon begins with a capital letter or a small letter. If the next word has a small letter the conclusion is that it is not the end of a sentence. But if the word following a sentence terminator does have a capital letter, the punctuation mark may indicate that it is indeed the end of a sentence. However, it may also denote that it is an abbreviation followed by a name of something, as in the sentence below:

“Patienten är allergisk mot en del läkemedel, t.ex. Penicillin, Aspirin och Kodein.”

(The patient is allergic to some drugs e.g. Penicillin, Aspirin and Codeine.)

How will the computer know the sentence does not end after the word “t.ex.” (e.g.)? Such situations must be taken into account because if such a sentence is wrongly interpreted the computer can miss that a patient has hypersensitivities to some drugs. The sentence above can be interpreted by the computer in two different ways. Either it is interpreted as the correct sentence above or it is interpreted as two sentences. To make the computer choose as accurately as possible it can be a good idea to let the computer choose the combination with the highest probability. This is described more in section 3.1.[13]

2.2 Stemming of words and removing irrelevant words

The purpose of stemming is to reduce the number of types in a document. Very often there are words which are not written in their root form. Let us for instance look at the Swedish word “stolar” (chairs), with the root form “stol” (chair). To change a word to its root form is called stemming. Each language has its own stemming algorithm. Often it is easy to remove or replace a word's suffix based on simple rules. But there are words which are not so easy to stem. Take for example the Swedish word “fick” (got) with the stem “få” (get). Such words are much more complicated since they do not follow any rules. Another problem emerges when a word has different meanings as for example the Swedish word “satt” which has different meanings in the sentences “Han satt.” (He sat.) and “Han är satt.” (He is stocky.). In the first sentence “satt” is the preterite form of “sitta” (sit), and in the second sentence it is an adjective which means stocky. Therefore, a stemming algorithm may be insufficient and it may be a good idea to have a stemming dictionary also.[14]

It would be favorable if the interpreted text could be cleaned. In other words, remove all unimportant words, so that only those words that can be possible medical terms remain. An easy way to do this is to look at what words a document consists of, how often these words appear, how ordinary they are in other documents etc. Words that appear several times in the records can be removed. These words, for example the words “patient” or “he”, are probably not very important since they do not describe the health of a patient. [15]

2.3 Making the computer recognize a sentence

A disease can be described in different ways. For instance, in Swedish you can describe the fact that someone has nut hypersensitivity as “Han tål inte nötter.” (He does not tolerate nuts.), “Han reagerade negativt mot nötter.” (He reacted negatively to nuts.), “Han är allergisk mot nötter.” (He is allergic to nuts.) etc. We see that the word “hypersensitivity” does not even need to appear in the sentence to describe an allergy. A human being can easily recognize that all sentences above relate to hypersensitivity but, a computer can not. To make a computer recognize when a sentence describes nut hypersensitivity one must teach the computer. In other words one must tell the computer which words describe the disease nut hypersensitivity. One method could be to collect all words that describe nut hypersensitivity. This method can of course cause problems, considering, for example, the following sentence: “Han är inte allergisk mot nötter.” (He is not allergic to nuts.). The words “allergisk” (allergic) and “nötter” (nuts) describe nut hypersensitivity but the word “inte” (not) is saying that the person does not have any allergy to nuts. How can one collect words that describe a particular disease in a smooth manner? The solution is to teach the computer which word combinations describe a disease and which combinations do not describe a disease.

If one were to tell the computer that the sentences “Han tål inte nötter.” (He does not tolerate nuts.) and “Han reagerade negativt mot nötter.” (He reacted negatively to nuts.) describe nut hypersensitivity then the computer may think the key words that describe nut hypersensitivity are “han” (he) and “nötter” (nuts) since they appear in both sentences. To avoid such misinterpretation, words can be removed as described in section 2.2. Then the number of words in the sentences would be smaller. Words such as “han” (he) probably not only appear in sentences that define an allergy and therefore such words would be removed. In other words, the solution is to remove as many words as possible that certainly do not describe a disease.

When a computer has learned which sentences describe a particularly disease a text can be reviewed to check if it contains a sentence which defines a particular disease.

2.3.1 Concept learning

As mentioned in section 2.3, the computer must be taught to recognize whether a sentence describes a particularly disease. The solution is to use a machine learning method: Concept learning. By giving good and bad training examples where good examples are members in the concept and bad are not you can construct a boolean-valued function that tells if next coming instance is in the category or not. A concept is a subset of the instances that are positive, and therefore members. An instance consists of a collection of conditions that together decide if it is a member or nonmember of the concept. One condition consist of an attribute and its value. Each attribute can choose between several values.[16]

Let us take a look at the following example: There is a couple with a child and they want to learn what their child likes to eat. They have the following training examples:

The food consists of a banana.	Wants to eat.
The food consists of an apple.	Wants to eat.
The food consists of a carrot.	Does not want to eat.

In this example there are three attributes: *banana*, *apple* and *carrot*. Each attribute can choose between the values *yes* and *no*. In Table 1 you can see the same example but more structured.

In this example there is a total of eight possible instances since there are three attributes with two possible values each. In an arbitrary example of concept learning with n attributes, where each attribute have $a_1, a_2, \ldots, a_n$ values, there are total $a_1 * a_2 * \ldots * a_n$ instances. Without seeing any training examples you have to do assumptions of which instances are members of the concept and which are not. This is called to do a hypothesis. There can be several hypotheses. More specifically, if there are m instances then there are 2^m possible hypotheses, since every instance may or may not be in a hypothesis. All these hypotheses are elements in a hypothesis space H. One of the hypotheses equals the target concept. The purpose of concept learning is to reduce the hypotheses so that one gets as close to the target concept as possible[17][18]

Banana	Apple	Carrot	Wants to eat
Yes	No	No	Yes
No	Yes	No	Yes
No	No	Yes	No

Table 1: Training examples for the concept "Wants to eat".

One way to reduce the hypothesis space is to let a hypothesis be a vector with attribute constraints, which conveys which attributes are good. Either the constraint is assigned a value *v* of an attribute which means that this value is the correct one, or it has a question mark, $?$, which means that all values of an attribute are good, or it is assigned $\emptyset$, which means that none of the values of an attribute are good. So a constraint in a hypothesis can assign two ($?$ and $\emptyset$) plus the number of values of the attribute that the constraint represents. If there are n attributes and each attribute has a_k values where $1 \leqslant k \leqslant n$ then there are $(2+a_1)*(2+a_2)*...*(2+a_n)$ hypotheses which is much lesser than $2^{a_1*a_2*...*a_n}$ hypotheses.[19]

The hypothesis $h_1 = \langle yes, ?, ? \rangle$, says that it contains the instances (yes, no, no), (yes, no, yes), (yes, yes, no) and (yes, yes, yes). The hypothesis $h_2 = \langle yes, no, ? \rangle$ is more *specific*[20] than the hypothesis h_1, and h_1 is more *general*[20] than h_2. In other words, h_1 contains the same positive instances as h_2 but also some more positive instances. The most general hypothesis $\langle ?, ?, ? \rangle$ is the one that says that no matter what values the attributes have, the attributes give a positive instance. The most specific hypothesis $\langle \emptyset, \emptyset, \emptyset \rangle$ is the one that says that none of the instances are positive.

2.3.2 Applying concept learning on sentences

The technique in the previous section is a possible solution to make a computer understand the main purpose of a sentence. One can produce training examples by seeking the records and finding sentences that describe a particular disease. Assume that the sentences "Han tål inte nötter." (He does not tolerate nuts.) and "Han kan inte äta nötter." (He can not eat nuts.) were found in some records. By applying concept learning on these sentences a table as the one in Table 2 can be constructed. Looking at the table, it is evident that the mutual words are "han" (he), "inte" (not) and "nötter" (nuts). Therefore, if such words are in a sentence the computer can assume that this sentence is about nut hypersensitivity. But suppose the computer encounter a sentence which has "hon" (she) instead of "han" (he). In this case the computer will not recognize the sentence as nut hypersensitivity. By elimination of unimportant words, described in section 2.2, the number of words in the table would be decreased.

han he	**tål** tolerate	**inte** not	**nötter** nuts	**kan** can	**äta** eat	**nut hypersensitive**
yes	yes	yes	yes	no	no	**yes**
yes	no	yes	yes	yes	yes	**yes**

Table 2: Instances constructed from sentences.

Another problem is for example the two sentences “Hon är allergisk mot vete.” (She is allergic to wheat.) and “Hon har en allergi mot vete.” (She has an allergy to wheat.). The two words “allergisk” (allergic) and “allergi” (allergy) refer to the same thing but will be two different words in the table. To avoid this, each word can go through a stemmer to obtain the root form of the word as described in section 2.2.

Diseases are divided into categories, for example, nut hypersensitivity and wheat hypersensitivity belong to the category food hypersensitivity.[21] Instead of having many different tables, one for each disease, another option is to have one table for the whole category. On the upside, lesser training examples would be needed since sentences containing both nut and wheat hypersensitivity etc. would be used in the same table. But this would not solve the problem of finding the correct disease. The table would only show to which category the sentence belongs, if any. Knowing that a sentence belongs to a category it is sufficient to search after a word that characterizes a disease's name in the category. If, for instance, the computer has detected a sentence that belongs to the category food hypersensitivity it is enough to search after the words nut, wheat, milk, peanut and egg which categorize each disease under the category food hypersensitivity.[21] In section 2.3.4 it is described more precisely how to solve this.

After having cleaned the words in the training examples it is time to choose the right algorithm that constructs hypotheses which are as close as possible to the target concept. There are some algorithms to choose between. In the next section it is analyzed what algorithm is best for this problem.

2.3.3 Algorithms to construct hypotheses

There are algorithms which construct hypotheses going from the most general hypothesis to more specific or the opposite namely begins with the most specific and generate more general hypotheses. Before any explanation of the different algorithms it would be good to explain what *version space* is. Version space is a subset of the hypothesis space *H* but it is consistent with all training examples. It contains all possible versions of the target concept.[22]

The List-Then-Eliminate algorithm is an algorithm that uses version space. From the beginning the version space contains all possible hypotheses. The List-Then-Eliminate algorithm eliminates all hypotheses which are inconsistent with any training example. If for example the hypotheses $h_1 = \langle ?, ?, ? \rangle$ and $h_2 = \langle ?, ?, yes \rangle$ are in the version space and there is a training example $t = (yes, no, no)$ then h_2 is not consistent with t and will be eliminated. This method is not suitable for the problem which is treated in this work since if one shall collect training examples from the records it is unlikely to find negative training examples, that is sentences which for example says that a patient does not have a disease. For example, if a patient is not allergic to wheat then there probably is not anything written about it, the sentence “Han är inte allergisk mot vete.” (He is not allergic to wheat.) would probably never appear in a record. The negative examples are

important when using the List-Then-Eliminate algorithm since a hypothesis as h_1 would never be eliminated. Suppose there are three attributes “tål” (tolerate), “inte” (not) and “allergisk” (allergic). Then if, and of course it is a small probability but the risk still remains, a sentence as “Hon är inte allergisk mot nötter.” (She is not allergic to nuts.) appears in a record the attributes “inte” (not) and “allergisk” (allergic) would obtain the value *yes* and “tål” (tolerate) would obtain the value *no.* Therefore, this sentence would be classified as a positive instance even if it is negative. So to summarize, the algorithm would also count sentences which negate diseases to positive instances.[23]

Another algorithm in concept learning is the Candidate Elimination algorithm which is similar to the List-Then-Eliminate algorithm but instead of remembering all the hypotheses in the version space it stores only the most specific and most general hypotheses. Then the hypotheses between are automatically included. Since this algorithm also depends on negative instances it is not suitable for this problem.[24]

Find-S is also an algorithm in concept learning and this algorithm depends only on positive training examples. This algorithm starts with the most specific hypothesis and after each positive training example it generalize the hypothesis so that the training example fits the hypothesis. Negative training examples are ignored. For each new positive training example, Find-S goes through all attribute constraints in the hypothesis and compare to related attributes. If the attribute constraint is consistent with the attributes value then nothing is done but if not then the attribute constraint is replaced with a more general one.[25]

Let us apply the Find-S algorithm at the training examples in Table 1. From the beginning the hypothesis *h* is as specific as possible:

$$h = \langle \emptyset , \emptyset , \emptyset \rangle$$

Looking at the first training example $t_1 = (yes , no , no)$, which is positive, one can see that *h* must be generalized. But only such that t_1 fits *h*. The new hypothesis is:

$$h = \langle yes , no , no \rangle$$

This hypothesis contains only one instance. Looking at the next example $t_2 = (no , yes , no)$ which also is positive the conclusion can be made that *h* must be generalized once again:

$$h = \langle ?, ?, no \rangle$$

Next training example is negative, $t_3 = (no, no, yes)$. As mentioned above, negative examples are ignored. One can see that *h* satisfies t_3. *h* says that the child likes to eat apple, banana, apple with banana and nothing, but this can of course be wrong, maybe the child does not like apple with banana. If this is the case then *h* does not equal the target concept and this is of course a problem. Regardless of the order of the training examples, the hypothesis is the same. Another negative aspect of the Find-S algorithm becomes evident when the training examples have errors.[26] If an example indicates that it is positive, but it actually is negative then the hypothesis becomes wrong.

The problem named above, that a negative instance can be found positive by the hypothesis, is an issue for this project. Assume that there are two sentences “Hon tål inte nötter.” (She does not tolerate nuts.) and “Hon är allergisk mot nötter.” (She is allergic to nuts.). From these two sentences the training examples as in Table 3 are constructed. When applying the Find-S algorithm on these examples the following steps are calculated:

$$h = \langle yes, yes, no, no \rangle$$
$$h = \langle ?, ?, ?, ? \rangle$$

This hypothesis creates a problem. Assume the sentence “Han är inte allergisk mot vete.” (He is not allergic to wheat.) which gets the instance (no, yes, yes, yes). This instance is negative but the hypothesis finds it positive. Or even more probable, if there is a sentence which includes none of these words then it classifies the sentence as belonging to this category. To avoid such errors the hypothesis must be modified a little to fit this project. Rules must be introduced and the hypothesis will not contain only one vector with attribute constraint but instead a disjunction of vectors. So the example from Table 3 will get the hypothesis

$$h = \langle yes, yes, no, no \rangle \vee \langle no, no, yes, yes \rangle .$$

tål tolerate	**inte** not	**är** is	**allergisk** allergic	**food hypersensitive**
yes	yes	no	no	**yes**
no	no	yes	yes	**yes**

Table 3: Constructed training examples from the sentences "Hon tål inte nötter." and "Hon är allergisk mot nötter.".

If an instance suits one of the vectors then it is consistent with the hypothesis. But now another problem has appeared. If a sentence as “Han tål inte nötter, dvs. han är allergisk.” (He does not tolerate nuts, that is, he is allergic.) which constructs the instance (yes, yes, yes, yes). The hypothesis will find this instance negative even if the sentence is describing food hypersensitivity. A solution to this problem is to replace the value *no* to *?*. So the hypothesis will instead be

$$h = \langle yes, yes, ?, ? \rangle \vee \langle ?, ?, yes, yes \rangle .$$

Of course this will contribute to categorize sentence as “Han är inte allergisk mot vete.” (He is not allergic to wheat.) as a positive instance but sentences which have none of these words will not be classified as members of the hypothesis, which is desired. It is better that the computer wrongly assumes that a patient has a disease even if he or she does not have any, than the other way around. In section 3.4 it is described more precisely what rules are used and what calculations are needed.

2.3.4 How to go further after the categorization of a sentence?

When a sentence is classified as belonging to a category the problem still remains of finding out what disease is described in the sentence. By saving all the names of diseases, along with their categories, one would know which possible diseases the sentence is describing. Several thousands possible diseases are reduced to maybe between ten and twenty, which is much easier to handle. MeSH medical terms are sorted in a tree structure, see Illustration 1. As one can see in the illustration, wheat hypersensitivity, nut hypersensitivity etc. belongs to category food hypersensitivity which in turn belongs to the category hypersensitivity, immediate. A node can have several parents, meaning one disease can belong to several categories. The functions from Findwise's web service Nyckelordtjänst make it possible to search for medical terms in MeSH. When searching for a word one gets returned all possible nodes this word can represent. If, for example, one searches for “nöt” (nut), then the returned nodes are “nötkreatur” (cattle), “nötkreatursjukdomar” (cattle diseases), “nötöverkänslighet” (nut hypersensitivity) and “nötter” (nuts). One can also get a node's parents, (the categories the node belongs to), and children if it has any.

If someone is allergic to, for example, nuts, then the word nut or its inflection must exist in the sentence in some way. So when searching for that word one receives four nodes. If a node belongs to the same category as this sentence the conclusion is that the sentence is describing this node. If one was to use Findwise's web service Vokabulärtjänst[11] instead, one could find the correct disease by searching for the category, acquiring all children to the corresponding node and comparing them to the words in the sentence. If a word equals the beginning of a node's name then it can be concluded that the sentence is describing that specific node (disease). These two

methods are probably equivalent, but with the latter method one needs the path to the node which can cause problems when, for example, a path changes. Therefore, the first method seems to be better for this work.

When searching on a word the received nodes are only those that begin with that word. If for example one searches on the word “nut” and there is a node containing that word but not in the beginning, then the node is not returned. Each node name has synonyms and if a synonym starts with the search word then the node will be returned also. In Swedish most of the compound words have their primary importance at the last part of the word.[27] Therefore the word “nötöverkänslighet” (nut hypersensitivity) is a hypersensitivity (“överkänslighet”) and nut (“nöt”) is describing which hypersensitivity it is. The descriptive words are in most cases the search words, hence it is not often a problem. But sometimes it is a problem, for example the disease “typ 2-diabetes” (diabetes mellitus, type 2) can be described as “diabetes mellitus, icke-insulinberoende” (diabetes mellitus, not insulin dependent) with the descriptive word “icke-insulinberoende” (not insulin dependent). A solution to this is presented in section 3.5.

Another problem is that if for example a sentence that belongs to the category diabetes mellitus and describes the disease diabetes mellitus type 2, consists of the words “diabetes”, “mellitus” and “icke-insulinberoende” among others, a search will be done on each of the word. The only hit is obtained by the word “diabetes”, but there are several node names beginning with “diabetes” that belong to the category diabetes. So the problem is to find only the node that is the correct one. In section 3.5 it is also described how this problem is handled.

Illustration 1: A screen shot from MeSH searching on "Food Hypersensitivity".

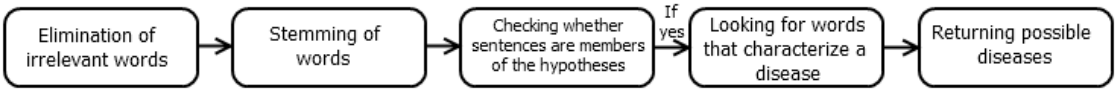

Illustration 2: An overview over the main steps in the demo program.

3 Method

The aim with this work is to make a computer understand and retrieve important information from electronic health records. In this chapter all used methods and algorithms are described and how they were implemented practically. A demo application in Java was written and in the following sections the main classes that were constructed for the application are described. Illustration 2 shows the main steps in the demo program.

3.1 Tokenization

As mentioned in section 2.1 a text must be divided into tokens. There are a lot of things to take in consideration when writing an application which tokenize. There are a couple of rules that are rather easy for a computer to follow in order to recognize a sentence. But there are also many difficult rules and there may be exceptions which are very hard to handle. A class called SentenceDivider was written which divides text into sentences and tokens. The first thing SentenceDivider does is to send text to WordDivider, which reads a string and divides it on spaces but also when a word starts or ends with for example the marks: *” (* , etc. WordDivider does not separate when a word ends with a sentence terminator.

The next thing SentenceDivider does is to check if a word ends with a sentence terminator. If this is the case then it determines whether the word following the sentence terminator starts with an upper case. If it does not then SentenceDivider assumes the sentence continues. But if the next word starts with an upper case then it checks if the word is an abbreviation. All abbreviations shall be listed in the file Abbreviations.txt. If it is not an abbreviation then SentenceDivider assumes that the sentence ends after this word. But if it is an abbreviation then SentenceDivider must calculate which combination of the words gives the highest probability. So the different possible combinations of the sentences are sent to HunposString and after that to ClassCombinationReducer, both of which are described in 3.2. The result consists of the sequence of word classes together with the probability that this sequence will occur. If there is a combination of words that can be interpreted as one sentence or as two then the probability that one sentence occurs is compared to the probability that the two sentences will occur. The probability that it will be two sentences is computed from the probabilities that these two sentences will occur by them self multiplying with each other and then multiplied with a , where $a = 10^{-2 \cdot \text{the number of the sentences}}$. The reason for this procedure is that when testing, the output was much better when multiplying the product of the probabilities that the sentences will occur with a . This is shown in the result in section 4.3. If sending the sentence

Sentence	Word class sequence	Probability that the sequence of word classes will occur
Patienten är allergisk mot en del läkemedel, t.ex. Penicillin, Aspirin och Kodein.	NN – VV – AJ – PR – EN – NN – NN – IK – AB – NN – IK – NN – ++ – NN – IP	$6.84 \cdot 10^{-11}$
Patienten är allergisk mot en del läkemedel, t.ex.	NN – VV – AJ – PR – EN – NN – NN – IK – NN – IP	$3.24 \cdot 10^{-9}$
Penicillin, Aspirin och Kodein.	NN – IK – NN – ++ – NN – IP	$3.97 \cdot 10^{-3}$

Table 4: Sentences which SentenceDivider can choose between.

"Patienten är allergisk mot en del läkemedel, t.ex. Penicillin, Aspirin och Kodein."

(The patient is allergic to some drugs e.g. Penicillin, Aspirin and Codeine.)

to SentenceDivider it can choose between sentences showed in Table 4. If multiplying the probabilities from row two and three and multiplying it with a, one obtains the probability $1.27 \cdot 10^{-15}$, which is less than $6.84 \cdot 10^{-13}$ so SentenceDivider chooses the combination from row one in Table 4. In the end the SentenceDivider has hopefully divided the text correctly in sentences and tokens.

3.2 Assigning correct word class to each token in a text

Hunpos[8] is a hidden Markov model-based part of speech tagger. A Markov process is a stochastic process but without memory. A stochastic model is a deterministic model which means it depends on previous cases. A Markov process does not care about what has passed, it only takes into account the current state. In Illustration 3 there is a state which tells that if a student studies for ten hours then the probability that the student will fail an exam is 80%. If a student has failed an exam it is with a certainty of 50% that he or she next time will study thirty hours for the exam. But this probability does not depend on the previous state. One does not know if the student has studied ten hours or thirty before the exam. The nodes "glad", "nervous", "tired" and "sad" are observations of what a student feels when he or she jumps to the respective state. So if a student is first tired and then glad one know that the student has studied for thirty hours and passed the exam. In a hidden Markov model there can be several possible observations to a node as in Illustration 4. So if a student is tired and then glad one can not say which states he or she has passed and therefore the state sequence is hidden.[28][29]

The patient records are in Swedish, hence a Swedish model was needed to hunpos. A class HunposArrayList and HunposString were written which sends an ArrayList as well as a list of strings to hunpos and receives the output from it. The output is the words together with the assigned hunpos word classes. Some word classes from hunpos are listed in Appendix C.

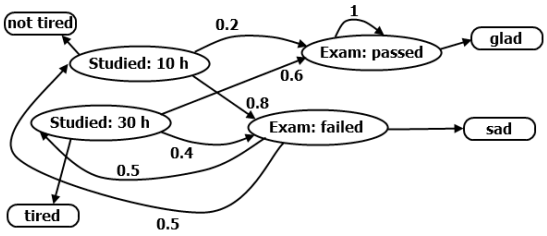

Illustration 3: A Markov model.

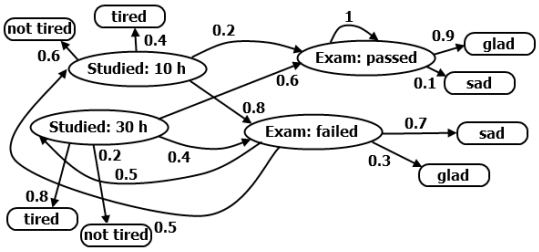

Illustration 4: A Hidden Markov model.

Word class from HunPos	**Word class from Talbanken05**	
	NN	ID
nn.utr.sin.def.gen	99.07%	
pm.nom	72.83%	24.4%

Table 5: An example on how hunpos is assigning word classes to words comparing with the word classes of Talbanken05. For explanations, see Appendix B. and Appendix C.

To see how good and precise hunpos is it was tested with Swedish sentences from files in Talbanken05[30]. Talbanken05 contains Swedish sentences divided into words, periods, commas etc. with corresponding word classes. A class HunposVSTalbanken was written which reads from .conll files in Talbanken05 and collects statistic about how often a word class from hunpos is representing a word class from Talbanken05. When HunposVSTalbanken reads from the .conll files it sends all sentences to hunpos and the output from hunpos is compared to the word classes in the files. The result was written to the file OutputStatUTF8kop.txt. Some word classes from hunpos are almost always correct and some are much more uncertain. See the example in Table 5. If a word class from hunpos matches a word class from Talbanken05 with less than 4% it is ignored since it is assumed that it is a misinterpretation from hunpos which does not occur as many times. If all values would be taken into account (also those beneath 4%) it would cause problems later on, which is described below in this section. As shown in Table 5 the word class pm.nom can correspond to a noun (NN) with 72.83% certainly and to a multi-word unit (ID) with 24.4% certainly. Why is this? In which circumstances does pm.nom appear as NN and when as ID?

To obtain an answer to this question, sequences of word classes from files in Talbanken05 were analyzed. A class called CalcWordSeq was written which reads from .conll files. CalcWordSeq reads all the sentences in the files and collects all combinations of three word classes that occur in the files and also all sequences each sequence can jump to. It also stores sequences of word classes which start or end a sentence. Some word classes in Talbanken05 are converted into a more general one, for example adjectival noun (AN) is converted into a usual noun (NN), thus there will be fewer different sequences which will be easier to take care of. When all sequences are calculated the result is written to the file OutputStatComb3.txt. CalcWordSeq is based

on a Markov process, since the probability to jump to a sequence depends only on the last sequence. The description of a Markov process was described in the beginning of this section.

Combining the information from the files OutputStatComb3.txt and OutputStatUTF8kop.txt must give more correct answers than if one would only take information from the last named file. Lets look at the following sentence which is sent to hunpos: “Idag är det mycket hetsigare och stressigare.” (Today it is much more heated and stressful.). The output is shown in Table 6. In Table 7 it is shown which word classes from Talbanken05 the output from hunpos could be assigned to. If one would take only these word classes from Talbanken05 with the highest probability the sentence above would get the following sequence of word classes:

AB VV PO AJ AJ ++ AJ IP

Words	**Word classes from hunpos**
Idag	ab
är	vb.prs.akt
det	pn.neu.sin.def.sub/obj
mycket	ab.pos
hetsigare	jj.kom.utr/neu.sin/plu.ind/def.nom
och	kn
stressigare	jj.kom.utr/neu.sin/plu.ind/def.nom
.	mad

Table 6: The output from hunpos when giving the sentence "Idag är det mycket hetsigare och stressigare." as input. For explanations, see Appendix C.

ab	AB	90.63%				
vb.prs.akt	VV	99.31%				
pn.neu.sin.def.sub/obj	PO	98.24%				
ab.pos	AJ	49.41%	AB	43.02%		
jj.kom.utr/neu.sin/plu.ind/def.nom	AJ	78.02%	PO	15.73%	AB	5.544%
kn	++	80.12%	UK	12.35%		
jj.kom.utr/neu.sin/plu.ind/def.nom	AJ	78.02%	PO	15.73%	AB	5.544%
mad	IP	98.64%				

Table 7: Word classes from hunpos representing the word classes from Talbanken05. For explanations, see Appendix B. and Appendix C.

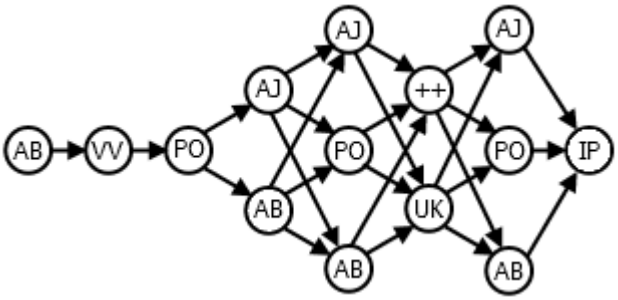

Illustration 5: A tree with all possible sequences of word classes. For explanations, see Appendix B.

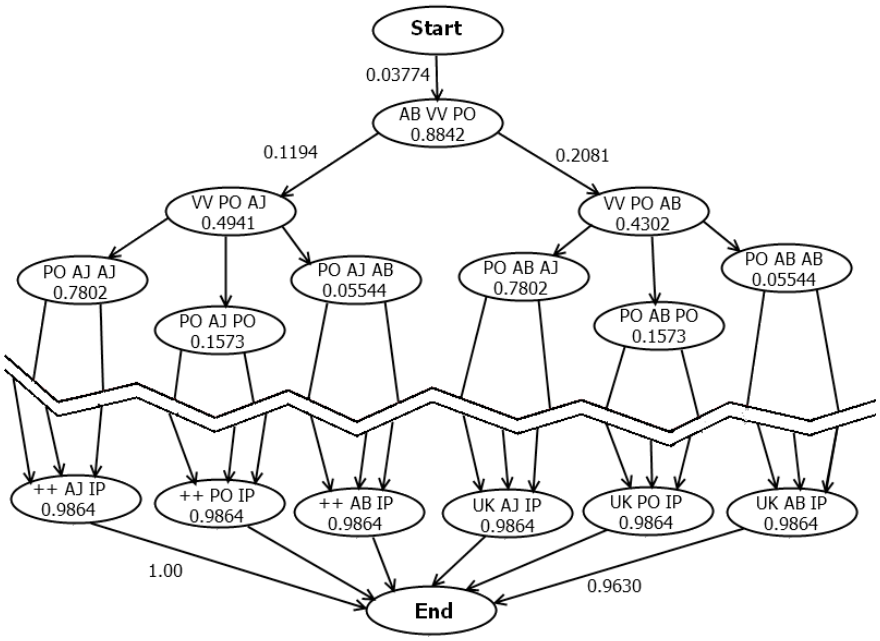

Illustration 6: A graph, where the nodes are assigned probabilities that the word classes will occur. The most likely path from the Start node to the End node is AB VV PO AB AJ ++ AJ IP. For explanations, see Appendix B.

But if one would choose these word classes which together with belonging sequences give the highest probability (see Illustration 6), one would instead get the following sequence of word classes:

AB VV PO AB AJ ++ AJ IP

This sequence is also the correct one. Of course this method is not a hundred percent right but it makes it more likely that more correct sequences of word classes would be assigned to a text, which is shown in the result in 4.2. The class

ClassCombinationReducer was written to make the computer choose the most probable sequence of word classes from Talbanken05 when inputting a sequence of word classes from hunpos. The ClassCombinationReducer reads and stores the information from the files OutputStatComb3.txt and OutputStatUTF8kop.txt. First ClassCombinationReducer creates a tree, that lists all possible sequences of word classes from Talbanken05 which the input sequence can represent, see Illustration 5. It goes through all the nodes and together with the information from the files mentioned above it creates nodes with three word classes in each and calculates the probabilities to jump to respective node, see Illustration 6. If there is no representation for a sequence then the sequence is ignored since it is assumed that such sequences are unlikely.

In Illustration 6, the percentage given in the first node following the start node represents the product of the respective accuracy of the three different word classes. The percentages in the subsequent nodes only represent the probability that the last word class in each node is correct, since the first two word classes are already included in the calculations. An edge shows the probability to jump to the next node taken into account the node that the edge came from. When calculating the probability of a path all probabilities on the way must be multiplied. ClassCombinationReducer uses Dijkstra's algorithm[31] to calculate the most probable way in the graph. According to the example above, the following path is the one with the highest probability:

$$
\begin{aligned}
&P(\text{AB VV PO}|\text{Start}) * P(\text{AB}|\text{ab}) * P(\text{VV}|\text{vb.prs.akt}) * P(\text{PO}|\text{pn.neu.sin.def.sub/obj}) * \\
&P(\text{VV PO AB}|\text{AB VV PO}) * P(\text{AB}|\text{ab.pos}) * \\
&P(\text{PO AB AJ}|\text{VV PO AB}) * P(\text{AJ}|\text{jj.kom.utr/neu.sin/plu.ind/def.nom}) * \\
&P(\text{AB AJ ++}|\text{PO AB AJ}) * P(\text{++}|\text{kn}) * \\
&P(\text{AJ ++ AJ}|\text{AB AJ ++}) * P(\text{AJ}|\text{jj.kom.utr/neu.sin/plu.ind/def.nom}) * \\
&P(\text{++ AJ IP}|\text{AJ ++ AJ}) * P(\text{IP}|\text{mad}) * \\
&P(\text{End}|\text{++ AJ IP}) = 1.006 * 10^{-6}
\end{aligned}
$$

As mentioned above in this section the probabilities that a word class from hunpos corresponds to a word class from Talbanken05 were ignored if they were beneath 4%. Looking at the example above there are already many paths to choose from. If all probabilities would be taken into account the number of paths would grow exponentially and it would be too time-consuming for the computer to find the best path. Hence there will be outputs from hunpos that could not be interpreted correctly as for example the sentence “Detta vill jag bestämt bemöta.”[30] (This, I would certainly like to address.) with the right word class sequence being

PO VV PO AJ VV IP

The output from hunpos with corresponding probabilities is shown in Table 8. The word class AJ is not even given as a possibility. Therefore, such situations will unfortunately always be associated with the wrong word classes.

Output from hunpos	Word classes from Talbanken05 that the output from hunpos can represent.			
pn.neu.sin.def.sub/obj	PO	98.24%		
vb.prs.akt	VV	99.31%		
pn.utr.sin.def.sub	PO	99.07%		
vb.sup.akt	VV	92.81%	TP	4.80%
vb.inf.akt	VV	97.90%		
mad	IP	98.64%		

Table 8: The output from hunpos when giving the sentence "Detta vill jag bestämt bemöta." as input. For explanations, see Appendix B. and Appendix C.

++	Coordinating conjunction	IQ	Colon
PO	Pronoun	IR	Parenthesis
EN	Indefinite article or numeral "en", "ett" (one)	IM	Infinitive marker
YY	Interjection	PU	List item (bullet or number)
IK	Comma	I?	Question mark
PR	Preposition	IS	Semicolon
IP	Period	IT	Dash
IC	Quotation mark	IU	Exclamation mark
UK	Subordinating conjunction		

Table 9: If a word has any of these tags the word is removed.

A few sentences may not get a translation between the output from hunpos to Talbanken05's word classes. This can occur in three different situations. Either the sentence is too short, it consists of less than three tokens or there is a lack of nodes that fit the possible word class sequences. If so, the only thing to do is to choose the word classes with the highest probability. The third thing it can depend on is when a word class from hunpos has no corresponding word class from Talbanken05. If such a situation occur, the sentence's word classes from hunpos are not converted to Talbanken05's word classes.

3.3 Removing unimportant words and stemming

After one has sent the text to ClassCombinationReducer and received the different word classes it is time to clean the text. The class WordElimination goes through all the text and eliminates tokens which are prepositions, conjunctions etc. Such tokens are not interesting when looking for a medical significance. All tokens that belong to some of the word class in Table 9 are eliminated. After the elimination of words and marks, the

rest of the words go to the class Stemmer which removes the end of a word such that it is in the root form. Stemmer builds on a Swedish stemmer algorithm[32]. The Swedish stemmer algorithm looks for the region after the first consonant following a vowel. This region is the ending of the word which will be modified. If there is no such consonant then there is no ending of the word which can be modified. The region to be modified in the word “allergic” is “lergic”. When this region is defined Stemmer removes letters from the ending of this region depending on which word class the word belongs to. Normally the Stemmer only takes into account the word classes from Talbanken05 but if the word is a noun, it also looks at the word class from hunpos since nouns can have different inflections depending if the word is in plural, definite, genitive, non-neuter etc. When Stemmer knows which word class the word belongs to it first determines if the longest removable ending is in the region and then the shorter one and so on. All removed endings are listed in Appendix E. Stemmer also searches for words that equal the words in the file Wordlist.txt. Wordlist.txt lists all words and their inflections, together with belonging word classes, that distinguish one disease from another. If a word class and the word are conformable with some word in the file Wordlist.txt then the word is replaced with the word in the base form. The reason for doing this is that it is more important that these words are in the base form since in most cases they are in base form in MeSH.

3.3.1 Construct a table with most frequent words

Different types of documents consist of different types of words. A historian uses different words in his research texts than a natural scientist. When care staff write patient records they also use specific words to describe patients and their diseases. Some words appear more often than others, such as the word “patient”. As mentioned in section 2.2 such words are often not that important since they generally do not describe any specific disease about a patient. Therefore, they can be removed. But to know which words can be removed all words in the patient records must be counted and divided into right types. A class ConstructWordTable was constructed to handle this. ConstructWordTable reads from text files and counts how many times different types of words appears. First it sends all the text to the SentenceDivider, described in section 3.1, so that all text will be divided into words, dots, commas etc. Then the text is sent to WordEliminator so that unimportant words can be removed and also so that words go through the stemmer. After the text is cleaned all words are counted and stored in a table. When the whole table is finished the words that appear most often are written to the file MostFrequentWords.txt. Words which occurred at least as many times as there were records were classified as frequent words, since these words do not seem to be unique.

3.4 Implementation of concept learning

As it is described in section 2.3 there must be an algorithm that recognizes a sentence. In that section it was established that the Find-S algorithm is the best option in this case, but the hypothesis must be a little modified. The purpose is to construct hypotheses which later can be used when analyzing sentences. The easiest way to do this is to

construct tables of words that show which word combinations are accepted, in other words hypothesis tables which list keywords of diseases. As previously mentioned, hypotheses' attribute constraints can either be a value of the corresponding attribute, a question mark or the empty mark. In this case, the empty mark can be reduced at the beginning since a word is either in a sentence or not, or it does not matter if the word is in the sentence at all. It never occurs that none of the values are correct. To have something that describes the attribute constraints a new data type, Value, was constructed. Value can have three different values. EXIST, which means that the word (attribute) must exist in the sentence. EXIST_NOT, which means that the word may not appear in the sentence and EXIST_DOES_NOT_MATTER, meaning that it does not matter if a word is in the sentence or not.

To construct a hypothesis table several components are needed. The main class which was constructed was TrainingData. This class reads from a file with sentences that describe a category (the name of the file) and writes the calculated hypothesis to a file. At the first row in the file all words which distinguish one disease from another in this category are listed. So if the category is food hypersensitivity one can see from Illustration 1 that the words which distinguish one disease from another are “ägg” (egg), “mjölk” (milk), “nöt” (nut), “jordnöt” (peanut) and “vete” (wheat). Therefore, all these words shall be listed in the first row of the file separated with spaces. The reason why these words shall be listed is because they will be ignored when the computer learns from the sentences. They do not characterize the category but instead only describe more precisely the diseases which belong to the category. Such words are not desirable in the table. Since there are not that many such describing words, it is not hard to list all such words in the beginning of the file. From second row down to the last, all possible sentences that belong to that category are listed. When TrainingData reads such a file it sends all sentences to the class SentenceDivider described in section 3.1. The reason is to separate the words from periods, commas etc. After this is done, all sentences are sent to the class WordEliminator, also described in 3.1, to eliminate unimportant words and also to stem the words.

After the sentences have been cleaned, all words are examined to determine whether they start with any of the words that shall be ignored. Looking at the previous example, this would involve determining whether some of the words start with “ägg” (egg), “mjölk” (milk) etc. If this is the case then that part is replaced by a star. For example, let us look at the word “nötöverkänslig” (nut hypersensitive). Since “nöt” (nut) shall be ignored the word is replaced with “*överkänslig”, meaning there may exist words which end on “överkänslig”. When removing such words the table becomes broader since “*överkänslig” also covers words as “mjölköverkänslig” (milk hypersensitive), “jordnötsöverkänslig” (peanut hypersensitive) etc. instead of only “nötöverkänslig”. Also, frequently occurring words are removed. Finally there are only those words left that may have something to do with the description of the category. When this is done all training examples can be constructed. A table that lists what word combinations occurs in the sentences is constructed. If a word occurs in a sentence then the attribute constraint assigns the value EXIST and if the word does not exist in the sentence the constraint gets the value EXIST_NOT. If two words are almost equal meaning they only

differ regarding one letter and if the length of one of the words is more than six letters then these words are treated as one word since such words can be treated as misspelling or that the stemmer has done something wrong. For example if one word is “överkänslig”, and another is “överkänsli” then both words are actually the same word. To calculate how many letters differ between two words Levenshtein Distance[33] is used which is described in section 3.5.

When the table is complete the hypothesis must be calculated. FindSModel is a class which constructs a hypothesis by adding training examples, one by one. A training example is represented by a Vector object. The class Vector consists of a list with Values and a boolean which tells if it is a positive or negative instance. The FindSModel adds only positive training examples as the Find-S algorithm. The FindSModel contains a list with all possible vectors represented by Vector objects. When adding a new training example the hypothesis is updated by iterating through all vectors in the hypothesis. If a training example can change a vector, so that this training example becomes a member of the hypothesis, the change will be done. Otherwise the training example will be added to the rest of the vectors. The requirement of changing a vector is that at least one value must have value EXIST at the same place in both the vector and the training example. If that is the case all values of the vectors are compared to corresponding values of the training example. If the value of a vector in the hypothesis is EXIST and the value of the training example is EXIST_NOT then the value of the vector is assigned EXIST_DOES_NOT_MATTER. The same thing is done if the value of the training example is EXIST and the value of the vector is EXIST_NOT. When a vector's value is assigned EXIST_DOES_NOT_MATTER then all other values in the vectors which are at the same place are also assigned EXIST_DOES_NOT_MATTER unless a value is the only attribute constraint in the vector which has the value EXIST, then this value is not changed. A hypothesis before and after a new training example:

$$h_{beforet} = \langle yes, no, no, no \rangle \vee \langle no, yes, no, yes \rangle \vee \langle no, no, yes, no \rangle$$

$$t = (yes, no, yes, no)$$

$$h_{after\,t} = \langle yes, no, ?, no \rangle \vee \langle ?, yes, ?, yes \rangle \vee \langle ?, no, yes, no \rangle$$

The reason to do it like this is that much more cases will be covered. If it does not matter if a word exists in a sentence it probably does not matter if it exists in another sentence also. For example if the computer has found that it does not matter if the word “har” (have) exists then the computer has hopefully found that this word has nothing to do with the category so it is needless to care about it in another kind of sentence.

When all training examples are added, one has a hypothesis table with vectors which have values EXIST, EXIST_NOT and EXIST_DOES_NOT_MATTER. In 2.3.3 it was described about a sentence that is describing a certain disease and therefore should be classified as a member of a hypothesis contains words which are consistent with a

vector's positive attribute constraints but the sentence also contains some words which are negative in the vector. Such sentence will be classified as a non member of the hypothesis. To avoid this all attribute constraints that have the value EXIST_NOT are replaced with EXIST_DOES_NOT_MATTER. Thus such sentences will be classified as members of the hypothesis. After all vectors are complete they are, together with all the words, written to a file. A hypothesis table is constructed and can be used in the categorization of a sentence.

3.5 Search in text

When all preparations are done it is time to search through a text. Search is a class which receives a text that will be checked through. Search reads from files with the hypothesis tables, described in 3.4 so that it stores all hypothesis tables. Then it sends the text to the SentenceDivider in order to tokenize the text. After this is done, the text is sent to WordElimination so that irrelevant words are eliminated. Frequently used words are also eliminated. Then for each sentence Search goes through all the hypothesis tables and checks if the remaining words are satisfying the attributes in the tables. If there is a vector in the hypothesis table which can represent a sentence then the conclusion is that this sentence belongs to the category which the table represents, but it is not certain that this sentence describes a disease in this category.

Since a word can be misspelled and the stemmer may not always stem as desired, words that are similar to some attribute are regarded as this attribute. To decide if a word is sufficiently similar to an attribute, an algorithm called Levenshtein Distance is used, which measures the difference between two strings. Levenshtein Distance calculates how many changes must be done to one string for it to equal another string, meaning it calculates the number of deletions, insertions and substitutions.[33] The distance between the words “allergic” and “allergy” is two since these two words are equal up until letter *g*. Then the letter *y* in “allergy” is substituted to *i* and after that an insertion of *c* is done. The whole algorithm is shown in Appendix D. In Dimitrios Kokkinakis papers[34] it is written that one can assume that words which have seven or more letters are more likely to be misspelled. In the Dimitrios Kokkinakis's research, words that have a Levenshtein Distance of at most one are treated as the same word. This work also deals with MeSH terminologies as the Dimitrios Kokkinakis research does, hence it can be a good idea to follow these restrictions. Hence, the words “allergy” and “alergy” are classified as the same word. Here “alergy” has only six letters but since “allergy” has seven Search will accept it as a candidate to calculate the Levenshtein Distance.

When a sentence is classified as belonging to a category, one still can not know which disease, if any, the sentence is describing. But the problem has become less severe considering the fact that the sentence originally could describe many thousand different diseases and now only up to maybe twenty. To find out what disease the sentence describes, the sentence is sent to SearchApelon. As described in 2.3.4 functions from Findwise's web service Nyckelordtjänst are used for this work. The main function

searches and outputs all possible nodes which begin with the words in a string. So if a cleaned sentence that consists of the words “patient” (patient), “är” (is), “allergi” (allergy) and “nöt” (nut) is the input to the Findewise's function one receives all nodes that begin with these words. To find out what categories these nodes belong to, their parents are fetched. If any parent equals the category of the sentence then one of these nodes is the name of the disease which the sentence is describing. The nodes in which parents are not equal to the category are ignored. If none of the parents of the nodes equal the category then the conclusion is made that this sentence does not describe any disease.

Assume that a sentence contains the words “sockersjuka” (another word for diabetes in Swedish) and “icke-insulinberoende” (not insuline-dependent) and the sentence belongs to the category “diabetes” (diabetes mellitus) then no node will be found since there is no node with synonyms that start with these two words and with parent nodes that equal the category. One solution is to add the name of the category to the words in the sentence. In that case, the search string will instead be “sockersjuka icke-insulinberoende diabetes”. This will solve the situations in where a node is describing a category and is not in the beginning of the word. Unfortunately there will be cases that will not be covered.

As mentioned in 2.3.4 one can receive several nodes that belong to the same category as the sentence. To solve this problem, the node with name or synonyms most similar to the words in the sentence is chosen as the final disease. To measure how similar a sentence is to a node's synonyms all words which are a substring of the synonyms are combined in all possible combinations. If a word with over six letters has a Levenshtein Distance to a word in the synonyms which is at most one it is also tested in the combination. For each of these combinations the Levenshtein Distance is calculated to the synonyms. The smallest distance is counted as this node's distance to the sentence. If for example one will calculate the distance between the words “överkänslig” (hypersensitive) and “nöt” (nut) and the node's name “nötöverkänslighet” (nut hypersensitivity) then it calculates the distance between “överkänslig nöt” and “nötöverkänslighet” which is six. It then calculates the distance between “nöt överkänslig” and “nötöverkänslighet” which is four. Therefore the distance between these two words no matter the combination and the node is four. To obtain a fair rating of which node is closest to the sentence, the percentage distances are compared. The percentage distance is calculated from the distance between two strings divided by the sum of the letters in these two strings. The node that has the smallest percentage distance to the sentence is chosen as the disease that the sentence is describing. No node is returned when only the name of the category is similar to the nodes, that is, no other words in the search string are substrings to the nodes' names or synonyms. Sometimes several nodes can have the same percentage distance to a sentence. If this is the case, all these nodes are returned as diseases.

In conclusion, the output from SearchApelon is the disease the sentence is describing, or nothing if the sentence is not describing any disease. The output from Search are all diseases which are described in the input text.

4 Result

In this section results from the different parts of the work are presented. Although some of the results are not very remarkable, reviewing them may still be interesting.

4.1 Frequently used words

As mentioned in the beginning of this report fictitious patient records were used. In Table 10 one can see the words which appear most frequently in the records after eliminating the irrelevant words in the WordEliminator. All words that have appeared fourteen or more times (there are fourteen fictitious patient records) are listed in the table. These words are eliminated from the sentences that are training examples and from the text which is scanned.

4.2 Using Talbanken05

All possible sequences of three word classes which occurred in Talbanken05 were collected. The probability of jumping from one sequence to another was calculated. There are a lot of sequences, hence it is hard to show them all here. Table 11 shows some few probabilities to jump from one sequence to another. The node START refers to the beginning of a sentence, a jump from START to a sequence means that a sentence starts with this sequence. If a sequence jumps to the END node it means that a sentence ends with this sequence.

Word	Number of occurrence	The percentage occurrence
har	39	2.78%
normal	31	2.21%
ej	30	2.14%
patient	25	1.78%
höger	20	1.43%
dag	20	1.43%
mg	20	1.43%
t	19	1.35%
u.a.	17	1.21%
nu	16	1.14%
blodtryck	16	1.14%
sedan	15	1.07%
arm	15	1.07%
vänster	15	1.07%
var	14	1.00%

Table 10: Most frequent words in the patient records.

From node	To node	PO VV PO	PO VV AB	AB VV AB	AB VV PO	END
START		2.79%	7.93%	0.778%	3.77%	-
IK PO VV		14.9%	20.8%	-	-	-
PO AB VV		-	-	11.4%	19.0%	-
++ PO VV		13.6%	40.3%	-	-	0.188%

Table 11: Probabilities to jump from one sequence of three word classes to another. A jump from START to a sequence means that a sentence starts with this sequence. A jump from a sequence to END means that a sentence ends with this sequence.

Word classes from HunPos	Word classes from Talbanken05	NN	++	VV	EN	AJ	IK	PO	IP	ID	UK	IQ	I?
nn.neu.sin.def.nom		0.97											
kn			0.8								0.12		
nn.utr.sin.def.nom		0.96											
vb.prs.akt				0.99									
dt.utr.sin.ind					0.9			0.1					
jj.pos.utr.sin.ind.nom						0.78		0.17					
nn.utr.sin.ind.nom		0.93								0.05			
mid							0.77					0.05	0.05

Table 12: Probabilities that a word class from HunPos is representing a word class from Talbanken05.

Section 3.2 covers the research of the precision of hunpos. Sentences from Talbanken05, which are divided into words, punctuation marks etc. with corresponding word class were sent to hunpos. The word classes which were allocated the tokens in hunpos were compared with the word classes in Talbanken05. In Table 12 one can see some of the probabilities that a word class from hunpos is representing a word class from Talbanken05. To show all probabilities would take up to much space.

To get a better result when assigning word classes to tokens, the output from hunpos were combined with the probabilities as in Table 11 and Table 12. This is described in 3.2. Tests were done to see how good this solution was. The sentences from Talbanken05 were sent to hunpos. When converting the output from hunpos to the word classes from Talbanken05 only the highest probabilities as those shown in Table 12 were taken into account. If, for example, one would get the word class kn from hunpos and wanted to get the representing word class from Talbanken05 one would always choose ++ and skip UK. The assigned word classes were then compared to the right word classes in Talbanken05, the error was 10.31%. The error is instead 9.47% when taking into account the probabilities in both Table 11 and Table 12 and choosing the

combination of word classes which gives the highest probability when combining the probabilities from these two tables. This may seem like not that big of an improvement, but the error has nevertheless been reduced by 8.16% which of course is better.

Sentences sent to SentenceDivider	**No multiplication with a**	**Multiplication with a**
Jag är allergisk mot en del läkemedel som t.ex. Penicillin, Alvedon och Kortison.		x
Jag har övers. Nyhetsposten tre gånger.		x
Jag har bott i många städer, bl.a. Warszawa, Berlin och Göteborg med sina fina områden, t.ex. Slottskogen, Delsjön m.m. Där brukar jag gå på promenader.	x	x
Jag tycker om Anna, Lisa och Mia m.m. Men jag tycker inte om Lova.	x	x
Jag har haft en del mobiler, bl.a. Samsung och Nokia.	x	x
Hon ligger på avd. Smörblomman.	x	x
Hon ligger på en avd. Hon mår inte så bra.	x	x
Jag vill sälja min beg. Volvo740, den är ganska risig.		x
Jag vill sälja min beg. Volvo740. Den är ganska risig.	x	x
Olle kommer hit m. Anna och deras dotter.		x
Jag har besökt Ida, Elisabet m.fl. De var alla jättesnälla.	x	x
I n.ö. Göteborg ligger Angered, Bergsjön m.m. I Angered ligger Vättlefjäll.	x	x
Det är tre nya tjejer, näml. Anna, Tova och Lisa.	x	x
Vi säljer äpplen, päron, apelsiner o.s.v. Men vi har även godis.	x	x
Vi måste hålla stängt p.g.a. Lisas operation.		x
Målet rör. Olle går snabbt framåt, sa Johan iron. Ja precis, svarade Linda.		
Min opp. Lisa ska själv ha redovisning om två v. Hon förbereder sig redan.	x	x
Här jobbar vaktm. Olle som har jobbat här i fyra år.		x
Den ligger u. Lisas arm, ser du?		x
Vi hade det trevl. Maria, jag, Sara m.fl. Vi tittade på Saras kort som hon har tagit i bl.a. London och Paris.		

Table 13: Sentences with abbreviations followed by a word which starts with an uppercase. The column "No multiplication with a" means that only the possible sentence's probabilities are multiplied and "Multiplication with a" means that the possible sentence's probabilities are multiplied and the product is multiplied with a, $a = 10^{-2 \cdot \text{the number of the sentences}}$*. An "x" means that the sentence is interpreted correctly.*

4.3 Determining whether there is a new sentence after an abbreviation

In 3.1 it is shown that when a sentence contains an abbreviation followed by a word which starts with an uppercase, SentenceDivider must determine whether there is a new sentence after the abbreviation or whether the sentence continues. SentenceDivider chooses the combination which gives the highest number when multiplying the probabilities of the occurrence of each sentence and then multiplying with a where $a = 10^{-2*the\ number\ of\ the\ sentences}$. Table 13 shows sentences that contain abbreviations followed by a word starting with uppercase. The table shows which sentences are interpreted correctly when only the possible sentences' probabilities are multiplied with each other and when possible sentences' probabilities are multiplied with each other and then multiplied with a. From the table, one can see that when there is no multiplication with a the interpretation is not that bad; eleven out of twenty sentences are interpreted correctly. But when multiplying with a eighteen sentences are interpreted correctly which is much better.

4.4 Construction of the hypothesis tables

The algorithm which constructs hypothesis tables was tested with some training examples. “Diabetes” was the first hypothesis table which was constructed. The file, as in Illustration 7 was sent to the class TrainingData, described in 3.4, and the constructed hypothesis table was as that in Table 14. This hypothesis table is pretty accurate. Sentences containing words as “åldersdiabetes” (diabetes of old age), “sockersjuka” (another word for diabetes) and “diabetisk” (diabetic) will be categorized as possible to describe a sub-disease to diabetes. This table is of course not ideal considering for example a sentence including the expression “prediabetisk fas” (prediabetic state). In this case, according to the table, the word “konstatera” (find out), “konstater” in the table, must also be in the sentence. This word has nothing to do with diabetes. Hence the table also consists of errors. To see how good this algorithm is it was also tested with similar sentences, but some of them were longer. In Illustration 8 one can see the file that was sent to TrainingData and in Table 15 the hypothesis table that was constructed. As one can see some words that occur in Table 14 are missing in Table 15, such as “*diabeti”. Therefore, the sentence “Är diabetisk.” (Is diabetic.) would not be classified as a member of this hypothesis. Another factor that will classify a sentence as a member of the hypothesis is the occurrence in the sentence of the word “uppsikt” (supervision), which is represented by the word “uppsik” in the table. The word “uppsikt” has nothing to do with diabetes, so the sentence will probably be wrongly classified.

```
 experimentell 1- autoimmun insulinberoende juvenil ungdoms 2- icke-insulinberoend
e ålders graviditets gestations ketoacidos pre
 T Lider av typ 1-diabetes.
 T Har haft graviditetsdiabetes.
 T Fick ungdomsdiabetes vid 14-årsåldern.
 T Fick diagnosen icke-insulinberoende diabetes.
 T Patienten har sockersjuka, är insulinberoende.
 T Prediabetisk fas konstaterades hos patienten.
 T Har sockersjuka, icke-insulinberoende.
 T Diagnoserades med åldersdiabetes.
 T Har MODY.
 T Patienten lider av NIDDM.
 T Har donohues syndrom.
 T Patienten har diabetesketoacidos.
 T Är diagnoserad med diabetes - juvenil.
 T Har autoimmun diabetes.
 T Är diabetiker, autoimmun.
 T Testades positiv för donohues syndrom.
 T Har lidit av sockersjuka sen ungdomen.
```

Illustration 7: A file with training examples which will be used when constructing the hypothesis table for diabetes. The words in the first row are words that will be ignored when the computer constructs the hypothesis table.

```
 experimentell 1- autoimmun insulinberoende juvenil ungdoms 2- icke-insulinberoend
e ålders graviditets gestations ketoacidos pre
 T Blodprovet för diabetes togs, prediabetisk fas konstaterades.
 T Blodprov för diabetes var positivt, misstänks ha donohues syndrom.
 T Lider av typ 1-diabetes.
 T Har haft graviditetsdiabetes under andra graviditeten.
 T Fick ungdomsdiabetes vid 14-årsåldern men är under kontroll.
 T Fick diagnosen icke-insulinberoende diabetes för några år sedan.
 T Patienten har sockersjuka, är insulinberoende, tar sprutor regelbundet.
 T Prediabetisk fas konstaterades hos patienten, får vara under uppsikt.
 T Har sockersjuka, icke-insulinberoende.
 T Diagnoserades med åldersdiabetes, får äta specialkost.
 T Har MODY.
 T Patienten lider av NIDDM.
 T Har donohues syndrom.
 T Patienten har diabetesketoacidos.
 T Är diagnoserad med diabetes - juvenil.
 T Har autoimmun diabetes.
 T Är diabetiker, autoimmun.
 T Testades positiv för donohues syndrom.
 T Har lidit av sockersjuka sen ungdomen.
```

Illustration 8: A file with somewhat longer sentences which will be used when constructing the hypothesis table for diabetes. The words in the first row are words that will be ignored when the computer constructs the hypothesis table.

*diabetes	sockersjuk	*diabeti	fas	konstater	mody	niddm	donohue	syndrom	diabetes-ketoacidos	diabe
yes	?	?	?	?	?	?	?	?	?	?
?	yes	?	?	?	?	?	?	?	?	?
?	?	yes	yes	yes	?	?	?	?	?	?
?	?	?	?	?	yes	?	?	?	?	?
?	?	?	?	?	?	yes	?	?	?	?
?	?	?	?	?	?	?	yes	yes	?	?
?	?	?	?	?	?	?	?	?	yes	?
?	?	?	?	?	?	?	?	?	?	yes

Table 14: The hypothesis table which was constructed after sending the file in Illustration 7 to TrainingData. Words that have the value ? in all rows were omitted. This hypothesis table shall represent sentences that describe diabetes.

*diabetes	syndrom	sockersjuk	uppsik	mody	niddm	diabetes-ketoacidos	diabe
yes	?	?	?	?	?	?	?
?	?	yes	?	?	?	?	?
?	?	?	yes	?	?	?	?
?	?	?	?	yes	?	?	?
?	?	?	?	?	yes	?	?
?	yes	?	?	?	?	?	?
?	?	?	?	?	?	yes	?
?	?	?	?	?	?	?	yes

Table 15: The hypothesis table which was constructed after sending the file in Illustration 8 to TrainingData. Words that have the value ? in all rows were omitted. This hypothesis table shall represent sentences that describe diabetes.

The algorithm was also tested with sentences that describe food hypersensitivity. Again, the algorithm was tested with short and long sentences. In Illustration 9 and Illustration 10, the two files that were sent to TrainingData are shown, and Table 16 and Table 17 contain the hypothesis tables that were constructed from each file. The same problem arises as in the previous tests. Words that have nothing to do with the diseases occur in the table. But many words that do describe the diseases are actually listed in the table which is very good. The hypothesis table based on the shorter sentences is somewhat better than the other table. But if there were more sentences in the file with the longer sentences, the table would probably be better. For example, if there was an additional sentence containing the word "över." (an abbreviation of hypersensitive) it probably would not contain the word "barndom" (childhood) so the latter word would also be assigned the value *?* which would make the table more adjusted to reality.

```
ägg mjölk nöt jordnöt vete
T Patienten är allergisk mot nötter.
T Patienten tål inte nötter.
T Reagerade mot vete.
T Kan inte äta mat med mjölk i.
T Visade tecken på allergi mot nötter.
T Patienten är överkänslig mot nötter och mjölk.
T Patienten klarar inte av nötter.
T Man gjorde ett test som visade överkänslighet mot mjölk.
T Patienten har mjölkallergi.
T Lider av överkänslighet av typ 1 mot nötter.
T Patienten har omedelbar överkänslighet mot ägg.
T Är över. mot nötter.
T Har mjölkall.
T Patienten lider av mjölkintol.
T När patienten drack mjölken så fick han en allergisk reaktion.
```

Illustration 9: A file with training examples which will be used when constructing the hypothesis table for food hypersensitivity. The words in the first row are words that will be ignored when the computer constructs the hypothesis table.

```
ägg mjölk nöt jordnöt vete
T Patienten säger sig vara allergisk mot nötter.
T Patienten tål inte nötter.
T Reagerade mot vete får några år sedan.
T Kan inte äta mat med mjölk i, måste ha specialmat.
T Visade tecken på allergi mot nötter.
T Patienten är överkänslig mot nötter och mjölk.
T Patienten klarar inte av nötter.
T Man gjorde ett test som visade överkänslighet mot mjölk.
T Patienten har mjölkallergi men kan dricka det i små mängder.
T Lider av överkänslighet av typ 1 mot nötter.
T Patienten har omedelbar överkänslighet mot ägg.
T Är över. mot nötter sen barndomen.
T Har mjölkall., får ont i magen vid intag.
T Patienten lider av mjölkintol. klarar dock av det i små mängder.
T När patienten drack mjölken så fick han en allergisk reaktion.
```

Illustration 10: A file with somewhat longer sentences which will be used when constructing the hypothesis table for food hypersensitivity. The words in the first row are words that will be ignored when the computer constructs the hypothesis table.

*allergi	inte	reagera	överkänsl	över.	*all	*intol
yes	?	?	?	?	?	?
?	yes	?	?	?	?	?
?	?	yes	?	?	?	?
?	?	?	yes	?	?	?
?	?	?	?	yes	?	?
?	?	?	?	?	yes	?
?	?	?	?	?	?	yes

Table 16: The hypothesis table which was constructed after sending the file in Illustration 9 to TrainingData. Words that have the value ? in all rows were omitted. This hypothesis table shall represent sentences that describe food hypersensitivity.

*allergi	inte	får	överkänsl	över.	barndom	*intol.	dock	små
yes	?	?	?	?	?	?	?	?
?	yes	?	?	?	?	?	?	?
?	?	yes	?	?	?	?	?	?
?	?	?	yes	?	?	?	?	?
?	?	?	?	yes	yes	?	?	?
?	?	?	?	?	?	yes	yes	yes

Table 17: The hypothesis table which was constructed after sending the file in Illustration 10 to TrainingData. Words that have the value ? in all rows were omitted. This hypothesis table shall represent sentences that describe food hypersensitivity.

4.5 Searching for diseases in a patient record

The final aspect of the work was to see if the demo application could go through a text in a patient record and output all diseases that are mentioned in the record. Since this report covers only the sub-diseases to diabetes and food hypersensitivity, tests were done only in order to find these particular diseases. The tests are divided into three parts. The first one tests patient records that are not describing any disease. The second one is to test patient records that include sentences that are classified as a members of a hypothesis table but are not describing a sub-disease. The third test is to test records that include sentences describing sub-diseases to diabetes and food hypersensitivity. All tests were conducted using the hypothesis tables from Table 14 and Table 16.

Sentences placed randomly in the records	Output from Search
Har pollenallergi.	x
Har inte visat tecken på andnöd.	x
Kan inte umgås med pälsdjur.	x
Reagerat med utslag för länge sedan på något hon tror var penicillin.	x
Har överkänslighet mot Penicillin.	x
Allergisk mot hamster och kiwifrukt.	x
Har diabetes och hypertoni, behandlas med tabletter.	x
Har diabetes i släkten.	x
Har sockersjuka.	x
Patientens pappa var diabetiker.	Typ 1-diabetes, Typ 2-diabetes
Allergisk mot kåvepenin (reagerade med utslag och klåda efter en pneumonibehandling -92).	x
Pupiller reagerar liksidigt på direkt och indirekt ljus.	x
Ingen känd allergi.	x

Table 18: Sentences which are members of the hypotheses in Table 14 and Table 16 but which are not describing any sub-diseases to the diseases that the hypotheses represent. The sentences were placed randomly in the patient records. An "x" in the column "Output from Search" means that the output was empty.

When the first test was implemented all answers were positive. Fourteen records, that did not describe any sub-diseases to diabetes and food hypersensitivity, were sent to Search and the output was empty. Some of the sentences in the records were classified as members of the hypotheses but further searches did not find any relation between the sentences and the sub-diseases. The purpose of the subsequent test was to examine if the output is empty or not when one sends records with sentences that are classified as members of the hypotheses but are not describing sub-diseases to the diseases the hypothesis tables represent. Sentences in Table 18 were placed randomly in different patient records. Only one sentence out of thirteen in the table outputs names on diseases that are assumed to be described.

The last test was aimed at determining whether or not the computer can capture diseases that are written down in the records. Sentences in Table 19 were placed randomly in the patient records and the records were sent to Search. Some of the words in the sentences are intentionally misspelled. The aim is that the computer should be able to find the correct diseases despite of the spelling errors. Most of the diseases are found by the computer, but some sentences are more difficult for the application to understand. The sentence "Har diabetes ketoacidos." (Has diabetic ketoacidosis.) is misspelled, the correct spelling being "Har diabetesketoacidos.". When Nyckelordtjänst's function has as input the word "diabetes" it should be able to find the disease "diabetesketoacidos"

since the disease starts with the word “diabetes”, but for some unknown reason it does not. The application classifies this sentence correctly, hence the error does not depend on the constructed algorithms. The sentence “Är insuliberoende, har diabetes.” (Is insulin dependent, has diabetes.) has the misspelled word “insuliberoende” which should be “insulinberoende” (insulin dependent). Since the word becomes stemmed to “insuliberoend” the Levenshtein Distance between this word and “insulinberoende” becomes two which does not fit the criteria for being the same word. The word “allergisk” (allergic) in the sentence “Fick allergisk chock av nötter.” (Got allergic shock from nuts.) is assigned the word class pronoun but the real word class is adjective. Since all words that are pronouns are eliminated, this word is eliminated also. Therefore the sentence is classified as a nonmember of the hypotheses and the disease is not found. The sentence “Över. mot nöter.” (Hypersensitive to nuts.) is classified correctly as a member of the hypothesis representing food hypersensitivity, but since the word “nöter” is misspelled, the correct spelling being “nötter” (nut), it is stemmed wrongly and Nyckelordtjänst's function is not able to find the correct disease. The last sentence that was misinterpreted is “Regerade med utslag då patienten drack mjölk.” (Reacted with a rash when the patient drank milk.). The word “regerade”, which is misspelled, the correct spelling being “reagerade” (reacted) instead, is stemmed to “regerad” and the Levenshtein Distance between this word and the word “reagera” in the hypothesis in Table 16 is two. This number is too high, hence the sentence is not classified as a member of this hypothesis.

Sentences placed randomly in the records	Output from Search	Correct disease
Har diabetes, typ 1.	Typ 1-diabetes	Typ 1-diabetes
Fick diabetes under graviditeten.	Graviditetsdiabetes	Graviditetsdiabetes
Tablettbehandlad diabetes mellitus typ II.	Typ 1-diabetes Typ 2-diabetes	Typ 2-diabetes
Har diabetesketoacidos.	Diabetesketoacidos	Diabetesketoacidos
Har diabetes ketoacidos.	x	Diabetesketoacidos
Är insulinberoende, har diabetes.	Typ 1-diabetes	Typ 1-diabetes
Lider av juvenildiabetes.	Typ 1-diabetes	Typ 1-diabetes
Har åldersdibetes.	Typ 2-diabetes	Typ 2-diabetes
Sockersjuka, typ 1.	Typ 1-diabetes	Typ 1-diabetes
Är insuliberoende, har diabetes.	x	Typ 1-diabetes
Kan inte dricka mjölk.	Mjölköverkänslighet	Mjölköverkänslighet
Fick allergisk chock av nötter.	x	Nötöverkänslighet
Kan inte äta jordnötter.	Jordnötsöverkänslighet	Jordnötsöverkänslighet
Har veteall.	Veteöverkänslighet	Veteöverkänslighet
Reagerade negativt mot ägg.	Äggöverkänslighet	Äggöverkänslighet
Överkänslig mot nötter.	Nötöverkänslighet	Nötöverkänslighet
Över. mot nöter.	x	Nötöverkänslighet
Regerade med utslag då patienten drack mjölk.	x	Mjölköverkänslighet
Är mjölkintol sen några år tillbaka.	Mjölköverkänslighet	Mjölköverkänslighet
Har alergi mot nötter.	Nötöverkänslighet	Nötöverkänslighet

Table 19: Sentences which are describing sub-diseases to food hypersensitivity and diabetes. The sentences were placed randomly in the patient records. An "x" in the column "Output from Search" means that the output was empty.

5 Discussion

There are some questions that are interesting to discuss. Is the method described in this report good? Could it be better? If yes, then how? Can healthcare professionals rely on this method? Are there other methods that may be more suitable? In the following paragraphs these questions are discussed among others.

When looking at the results shown in 4.5 one can see that this method is rather satisfying. The algorithms could at least find many of the diseases which were mentioned in the patient records. But of course there could have been more rightly interpreted diseases. The question is whether this would work on other diseases and not only on the ones that were tested in this master thesis. Since many diseases have similar structure this method would probably work on other diseases too, but probably some more rules would have to be added. Of course this theory needs to be tested. To make this method more accurate there are some aspects that could probably be improved on. First there is the issue of elimination of words. In 4.2 it is written that the error from hunpos is 10.31%, but when using hunpos together with combinations of three word classes the error is reduced to 9.47%. If this error could be reduced even more, additional irrelevant words would be eliminated and errors such as the one in 4.5, where the word “allergisk” (allergic) was assigned the wrong word class and was incorrectly eliminated, would also be decreased. Maybe a combination of four or more word classes instead of three would contribute to a better output. But the longer the combinations of the word classes, the longer it would take for the computer to find the correct sequences. One could also ask if the error, when sending the patient records to hunpos, is not even more extensive, considering that the sentences in the patient records are not always grammatically correct. For example, healthcare professionals often write short and concrete sentences in a record.[35] For instance, the sentence “He got rashes.” may be reduced to “Got rashes.” which can be more difficult for the tagger to interpret. So maybe the patient records should have a tagger of their own. Since many medical terms are used in the patient records and these are important it could potentially be very dangerous if they were eliminated. Maybe a word-list which lists all medical terms that can not be removed could be a solution? Another improvement on the search in records would be to produce a better stemmer. For instance, a stemmer with more specific rules and which also has a word-list with all irregular words.

If the words would be eliminated and stemmed correctly the hypothesis tables would become more consistent with reality. But they can still fail to classify a positive sentence as a member of the hypothesis, as for example in 4.5, which had the word “reagerade” (reacted) misspelled and wrongly stemmed to “regerad”. If the word were to be correctly spelled and stemmed it would fit the keyword “reagera” (react) in Table 16 but now the Levenshtein Distance is two which contributes to wrong classification of that sentence. Perhaps the limit, that the distance between two words may at most be one, is too low. Maybe one should allow the distance between two words to be two when the words get stemmed. Another problem when constructing these hypothesis

tables is if the wrong word is assigned the value *?*. But if there are many training examples (sentences) the keyword may appear again with other words and thus in another vector the word may obtain the value *yes* again.

When a sentence is classified as a member of a hypothesis it still remains to be found which disease it relates to. The problem here was that misspelled words could not return correct nodes. The word “nötter” (nuts) in 4.5 with the stemmed form “nöt” (nut) was misspelled to “nöter” and therefore it did not get stemmed correctly. This word is a short one so it is difficult for the computer to guess what the correct word is. Maybe the misspelled word already has a meaning as in this case in which the word “nöter” means “wears”. That makes it even harder to find out which real meaning this word should have. Of course one could have a word-list with commonly misspelled words but there can always occur new spelling errors. To avoid problems, also mentioned in 4.5, concerning the words “insuliberoend” and “insulinberoende” (insulin-dependent) being classified as two different words, one could allow a bigger Levenshtein Distance between longer words, since it is more likely that the long words will be misspelled and wrongly stemmed.

Another big question is if healthcare professionals can rely on this method. This method must of course be developed further and then maybe it could become a helpful tool for professionals. But, since computers can always miss a disease maybe one can not rely on them completely. But, if, for example, an unconscious patient arrives to the emergency room and the healthcare professional does not know anything about what he or she is allergic to then such a application may be helpful. A problem arises if a disease is negated in a record. If, for example, tests are done so that a disease can be excluded then it is probably mentioned in the record. In this case, the sentence which negates the disease can be found by the application as describing the disease. This can cause problems.

There may exist methods more suitable to finding diseases in a text. Maybe an algorithm which also takes negative training examples into account so that the issues described in the last paragraph do not appear. This is of course a very big subject which can be investigated extensively. In this master thesis the application assumes that a patient does not have any diseases from the beginning and then searches if he or she has one. Maybe it would be better to assume that a patient has all diseases until the opposite is proven.

This work only treats sentences that refer to themselves, but there can of course be sentences that refer to each other. Therefore, an algorithm that takes this into account would be better. Also, English medical words can occur in the records, hence a collection of these are a good idea also.

6 Conclusion

The subject automatic searching in electronic health records is probably solvable and it is absolutely worth further investigations. There may of course exist much better methods than the method described in this report. A good idea would be to make surveys and examine various methods to perform automatic searching in electronic health records. By implementing several tests one can find the necessary rules and reduce the error rate. Of course this requires a lot of extensive research. Maybe one can not rely on this method a hundred percent but it may still prove to be an aid in healthcare.

A program such as the demo application can in addition to searching through a patient's health record and alerting the healthcare professionals to what diseases he or she has, also be used for other things. For example if there are scientists who need a group of people with a particular disease to test a new medicine on, they would only need to search through all the records and find these people very quickly. Another situation in which such a program could be used is when a healthcare professional writes in a patient record. The program could then automatically detect if he or she writes about for example an allergy and alert the healthcare professional to the fact that this information should be put in a special field called Allergy. Such a program could also be useful in other areas. Of course the program must be adapted in order to suit the given area.

Another aspect which would be interesting to investigate is whether a similar method could work when making a search on the Internet. That is, when one searches for something on the Internet, could one also receive the documents belonging to the same category as the search phrase even if they do not contain the particular search words. This is of course a much more difficult problem since the Internet is very big, it might even be too big. However, this would be a very interesting research subject.

References

[1] Karolinska Institutet, *Svensk MeSH - Sökhjälp - medicinsk information - Biblioteket - ki.se*, http://mesh.kib.ki.se/swemesh/swemesh_se.cfm, 2010-05-11.
[2] Karolinska Institutet, *MeSH Indexing Manual - A Guide to MeSH Indexing - Karolinska Institutet*, http://ki.se/ki/jsp/polopoly.jsp;jsessionid=aeNI9cLM010hAkuuhI?l=en&d=4358&a=11716, 2010-05-11.
[3] Oracle Corporation and/or its affiliates, *Java SE Downloads - Sun Developer Network (SDN)*, http://java.sun.com/javase/downloads/index.jsp, 2010-05-11.
[4] G. Petersson, *diktatex*, http://www.oron.mas.lu.se/utbildning/diktatex.html, 2010-05-13.
[5] Linköpings universitet, *Journalexempelvt2010.pdf (application/pdf-objekt)*, http://www.hu.liu.se/lakarprogr/t5/dokumentarkiv/1.178223/Journalexempelvt2010.pdf, 2010-05-13.
[6] Kandidaterna, *Kandidaterna*, http://www.kandidaterna.se/, 2010-05-13.
[7] *Medical students at Läkarprogrammet*, University of Gothenburg, 2010
[8] Google, *hunpos - Project Hosting on Google Code*, http://code.google.com/p/hunpos/, 2010-05-11.
[9] Apelon Inc., *Apelon DTS - Welcome to Apelon DTS*, http://apelon-dts.sourceforge.net/, 2010-05-11.
[10] Findwise, *Findwise | Findwise - Search Driven Solutions*, http://findwise.se/, 2010-05-11.
[11] Google, *Wiki Pages - oppna-program-metadata-service - Project Hosting on Google Code*, http://code.google.com/p/oppna-program-metadata-service/w/list, 2010-05-11.
[12] Regeringskansliet, *Språkliga frågor och svar*, http://www.regeringen.se/sb/d/2729, 2010-05-13.
[13] S. M. Weiss, N. Indurkhya, T. Zhang, F. J. Damerau, *TEXT MINING Predictive Methods for Analyzing Unstructured Information*, 20-21. Springer Science+Business Media, Inc., New York ,2005.
[14] S. M. Weiss, N. Indurkhya, T. Zhang, F. J. Damerau, *TEXT MINING Predictive Methods for Analyzing Unstructured Information*, 21, 23. Springer Science+Business Media, Inc., New York ,2005.
[15] S. M. Weiss, N. Indurkhya, T. Zhang, F. J. Damerau, *TEXT MINING Predictive Methods for Analyzing Unstructured Information*, 25-26. Springer Science+Business Media, Inc., New York ,2005.
[16] T. M. Mitchell, *MACHINE LEARNING*, 20-21. McGraw-Hill Companies, Inc., United States of America ,1997.
[17] P. Damaschke, *The course in Algorithms for machine learning and inference (TDA231)*, Chalmers University of Technology, January-March 2009.
[18] T. M. Mitchell, *MACHINE LEARNING*, 22-23. McGraw-Hill Companies, Inc., United States of America ,1997.
[19] T. M. Mitchell, *MACHINE LEARNING*, 21. McGraw-Hill Companies, Inc., United States of America ,1997.
[20] T. M. Mitchell, *MACHINE LEARNING*, 22. McGraw-Hill Companies, Inc., United States of America ,1997.

[21] Karolinska Institutet, *Food Hypersensitivity*, http://mesh.kib.ki.se/swemesh/show.swemeshtree.cfm?Mesh_No=C20.543.480.370&tool=karolinska, 2010-05-11.
[22] T. M. Mitchell, *MACHINE LEARNING*, 26-30. McGraw-Hill Companies, Inc., United States of America ,1997.
[23] T. M. Mitchell, *MACHINE LEARNING*, 30. McGraw-Hill Companies, Inc., United States of America ,1997.
[24] T. M. Mitchell, *MACHINE LEARNING*, 32-33. McGraw-Hill Companies, Inc., United States of America ,1997.
[25] T. M. Mitchell, *MACHINE LEARNING*, 26. McGraw-Hill Companies, Inc., United States of America ,1997.
[26] T. M. Mitchell, *MACHINE LEARNING*, 28. McGraw-Hill Companies, Inc., United States of America ,1997.
[27] T. G. Hultman, *Svenska Akademiens språklära*, 32. Svenska Akademien, Stockholm ,2003.
[28] M. W. Kadous , *Hidden Markov models*, http://www.cse.unsw.edu.au/~waleed/phd/tr9806/node12.html, 2010-05-11.
[29] J. Enger, J. Grandell, *MARKOVPROCESSER OCH KÖTEORI*, 9-11. Department of Mathematical Sciences Chalmers University of Technology and Göteborgs University, Göteborg ,2009.
[30] Lund University, *Talbanken05*, http://stp.lingfil.uu.se/~nivre/research/Talbanken05.html, 2010-05-11.
[31] M. T. Goodrich, R. Tamassia, *Data Structures & Algorithms in Java*, 621-625. John Wiley & Sons, Inc., United States of America ,2006.
[32] M. Porter or R. Boulton, *Swedish stemming algorithm*, http://snowball.tartarus.org/algorithms/swedish/stemmer.html, 2010-05-11.
[33] M. Gilleland, *Levenshtein Distance*, http://www.merriampark.com/ld.htm#REFS, 2010-05-11.
[34] D. Kokkinakis, *Lexical granularity for automatic indexing and means to achieve it - the case of Swedish MeSH®.*, 25-26. In Information Retrieval in Biomedicine : Natural Language Processing for Knowledge Integration. Prince V. and Roche M. (eds). pp. 11-37., IGI Global, 2009.
[35] J. Berg, *Facts from the intern Johanna Berg*, Gothenburg, 2010-05-07
[36] Öresunds Översättningsbyrå, *Oversættelse svensk dansk - Statsaut. translatører - Online ordbog*, http://www.dansk-og-svensk.dk/, 2010-05-12.
[37] G. Andersson, *Den stora svenska ordlistan*, http://dsso.se/download.html, 2010-05-13.
[38] J. Nivre, *Lexical categories in MAMBA*, http://stp.lingfil.uu.se/~nivre/research/MAMBAlex.html, 2010-05-11.
[39] E. Ejerhed, D. Ridings, *PAROLE -> SUC tagset*, http://spraakbanken.gu.se/parole/tags.phtml, 2010-05-11.
[40] E. Andersson, *Grammatik från grunden - En koncentrerad svensk satslära*, 28-30, 34-35, 42-44, 46-47, 48. Hallgren & Fallgren Studieförlag AB ,1993.

[41] D. Kokkinakis, *Lexical granularity for automatic indexing and means to achieve it - the case of Swedish MeSH®.*, 19-20. In Information Retrieval in Biomedicine : Natural Language Processing for Knowledge Integration. Prince V. and Roche M. (eds). pp. 11-37., IGI Global, 2009.

Bibliography

These books were used in the research of this master thesis

Andersson E. 1993. *Grammatik från grunden - En koncentrerad svensk satslära*, 28-30, 34-35, 42-44, 46-47, 48. Hallgren & Fallgren Studieförlag AB.

Enger J. & Grandell J. 2009. *MARKOVPROCESSER OCH KÖTEORI*, 9-11. Department of Mathematical Sciences Chalmers University of Technology and Göteborgs University, Göteborg.

Goodrich M. T. & Tamassia R. 2006. *Data Structures & Algorithms in Java*, 621-625. John Wiley & Sons, Inc., United States of America.

Hultman T. G. 2003. *Svenska Akademiens språklära*, 32. Svenska Akademien, Stockholm.

Kokkinakis D. 2009. *Lexical granularity for automatic indexing and means to achieve it - the case of Swedish MeSH®.*, 19-20, 25-26. In Information Retrieval in Biomedicine : Natural Language Processing for Knowledge Integration. Prince V. and Roche M. (eds). pp. 11-37., IGI Global.

Mitchell T. M. 1997. *MACHINE LEARNING*, 20-23, 26-30, 32-33. McGraw-Hill Companies, Inc., United States of America.

Weiss S. M., Indurkhya N., Zhang T. & Damerau F. J. 2005. *TEXT MINING Predictive Methods for Analyzing Unstructured Information*, 20-21, 23, 25-26. Springer Science+Business Media, Inc., New York.

These homepages were used in the research of this master thesis

Gilleland M. [2010-05-11] *Levenshtein Distance*, http://www.merriampark.com/ld.htm#REFS

Kadous M. W. 1998. [2010-05-11] *Hidden Markov models*, http://www.cse.unsw.edu.au/~waleed/phd/tr9806/node12.html

Karolinska Institutet [2010-05-11] *Svensk MeSH - Sökhjälp - medicinsk information - Biblioteket - ki.se*, http://mesh.kib.ki.se/swemesh/swemesh_se.cfm

Porter M. and Boulton R. [2010-05-11] *Swedish stemming algorithm*, http://snowball.tartarus.org/algorithms/swedish/stemmer.html

Regeringskansliet [2010-05-13] *Språkliga frågor och svar*, http://www.regeringen.se/sb/d/2729.

These are other sources used in the research of this master thesis

Berg J. [2010-05-07] *Facts from the intern Johanna Berg*, Gothenburg.

Damaschke P. [January-March 2009] *The course in Algorithms for machine learning and inference (TDA231)*, Chalmers University of Technology.

Services from these sources have been used in this master thesis

Andersson G. 2003. [2010-05-13] *Den stora svenska ordlistan*, http://dsso.se/download.html

Apelon Inc. 2007-2010. [2010-05-11] *Apelon DTS - Welcome to Apelon DTS*, http://apelon-dts.sourceforge.net/

Ejerhed E. & Ridings D. 2010. [2010-05-11] *PAROLE -> SUC tagset*, http://spraakbanken.gu.se/parole/tags.phtml

Findwise 2010. [2010-05-11] *Findwise | Findwise - Search Driven Solutions*, http://findwise.se/

Google 2010. [2010-05-11] *hunpos - Project Hosting on Google Code*, http://code.google.com/p/hunpos/

Google 2010. [2010-05-11] *Wiki Pages - oppna-program-metadata-service - Project Hosting on Google Code*, http://code.google.com/p/oppna-program-metadata-service/w/list

Kandidaterna [2010-05-13] *Kandidaterna*, http://www.kandidaterna.se/

Linköpings universitet 2010. [2010-05-13] *Journalexempelvt2010.pdf (application/pdf-objekt)*, http://www.hu.liu.se/lakarprogr/t5/dokumentarkiv/1.178223/Journalexempelvt2010.pdf

Lund University 2006. [2010-05-11] *Talbanken05*, http://stp.lingfil.uu.se/~nivre/research/Talbanken05.html

Medical students at Läkarprogrammet 2010. University of Gothenburg

Nivre J. [2010-05-11] *Lexical categories in MAMBA*, http://stp.lingfil.uu.se/~nivre/research/MAMBAlex.html

Oracle Corporation and/or its affiliates 2010. [2010-05-11] *Java SE Downloads - Sun Developer Network (SDN)*, http://java.sun.com/javase/downloads/index.jsp

Petersson G. 1999. [2010-05-13] *diktatex*, http://www.oron.mas.lu.se/utbildning/diktatex.html

Öresunds Översättningsbyrå [2010-05-12] *Oversættelse svensk dansk - Statsaut. translatører - Online ordbog*, http://www.dansk-og-svensk.dk/

Appendixes

This chapter contains some clarifications to the report. The first appendix contains an explanation of how one installs and uses the demo application. The remaining appendixes contain some explanations.

Appendix A. The demo application – user guide

Make sure you have Java[3] installed on your computer and that .jar files are opened with Java. To install the demo application just unpack the file ExjobbDemo.rar in a folder. Make sure that none of the parent folders have names with spaces etc. After the unpacking you can start the application by opening the file demo.jar located in the folder ExjobbDemo. If you want to add a record make sure that the file is UTF-8 encoded and place the file in the folder ExjobbDemo\Files\Records. Add headlines that appear in the file but which are missing in the file Headlines1.txt. The file Headlines1.txt is located in the folder ExjobbDemo\Files\IgnoreHeadlines. When you want to search in a record write the name of the record in the field and click on the button Search, see Illustration 11. Illustration 12 shows the result after a search. Table Table 20 lists the files and folders, located in ExjobbDemo\Files, and what they contain.

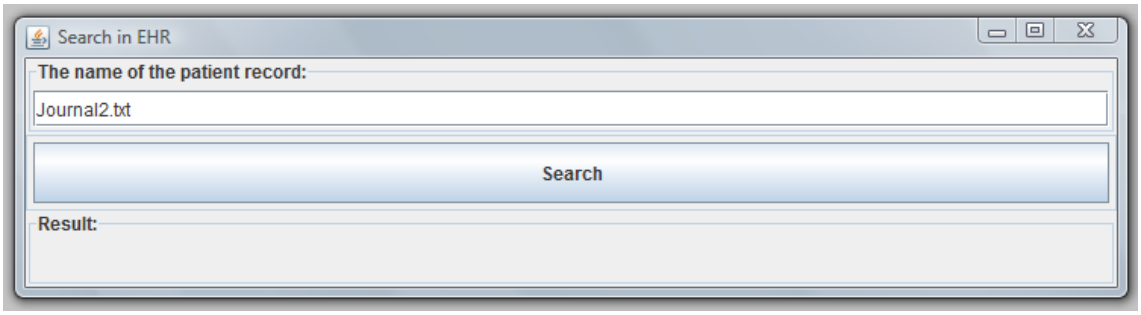

Illustration 11: Before the search in the patient record "Journal2.txt".

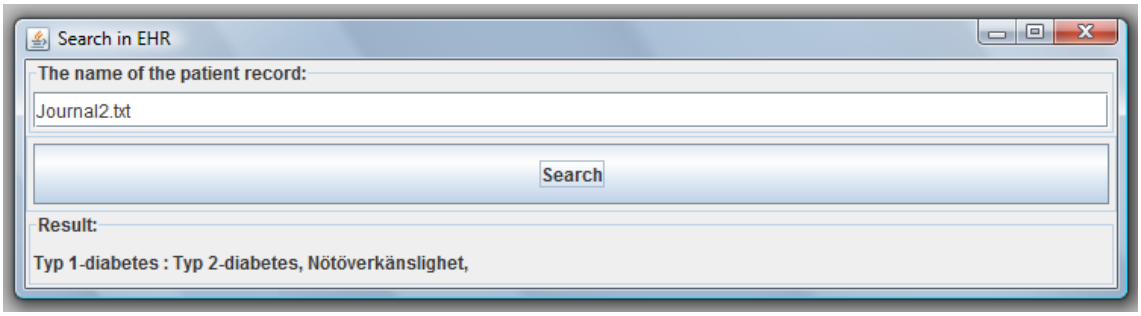

Illustration 12: After the search in the patient record "Journal2.txt".

Abbreviations.txt	A file with Swedish abbreviations from Öresunds Översättningsbyrå[36].
OutPutStatComb3.txt	The file contains combinations of three word classes and the probabilities that sequences of word classes can jump to another sequences.
OutputStatUTF8kop.txt	The file contains probabilities that word classes from hunpos are representing word classes from Talbanken05.
Wordlist.txt[37]	The file contains words, with inflections and belonging word classes, that distinguish one disease from another but which belong to the same category.
HypothesisTables	The folder contains files with hypothesis tables. The file names must be equal to the name of the categories that these files are representing.
IgnoreHeadlines	The folder contains the file Headlines1.txt. The file lists all headlines that appear in the patient records.
Records	The folder contains patients records. They are UTF-8 encoded text files.
TableHypothesesLong	The folder contains files with hypothesis tables. These hypothesis tables are constructed from somewhat longer sentences. The file names must be equal to the name of the categories that these files are representing.
TrainingExamples	The folder contains files that include sentences which are used to construct hypothesis tables. The first row in each file lists words that distinguish one sub-disease to a category from another sub-disease.
WordFrequence	The folder contains the file MostFrequentWords.txt which lists the most frequently occurring words.

Table 20: Files and folders located in ExjobbDemo\Files.

Appendix B. Word classes from Talbanken05

Abbreviations of word classes with explanations[30][38], that were used in this work.

ID	part of idiom (multi-word unit)	UK	subordinating conjunction
NN	noun	IK	comma
PO	pronoun	IP	period
EN	indefinite article or numeral “en”, “ett” (one)	I?	question mark
RO	other numeral	IU	exclamation mark
AJ	adjective	IQ	colon
VV	verb	IS	semicolon
TP	perfect participle	IT	dash
SP	present participle	IR	parenthesis
AB	adverb	IC	quotation mark
PR	preposition	PU	list item (bullet or number)
IM	infinitive marker	IG	other punctuation mark
++	coordinating conjunction	YY	interjection
XX	unclassifiable part-of-speech		

Appendix C. Word classes from hunpos

The word classes from hunpos that were mentioned in this report.[39]

nn.utr	noun non-neuter
nn.utr.sin.def.gen	noun non-neuter singular definite genitive
nn.utr.sin.def.nom	noun non-neuter singular definite nominative
nn.utr.sin.ind.nom	noun non-neuter singular indefinite nominative
nn.utr.sin.ind.gen	noun non-neuter singular indefinite genitive
nn.utr.plu.def.gen	noun non-neuter plural definite genitive
nn.utr.plu.ind.gen	noun non-neuter plural indefinite genitive
nn.utr.plu.def.nom	noun non-neuter plural definite nominative
nn.utr.plu.ind.nom	noun non-neuter plural indefinite nominative
nn.neu	noun neuter
nn.neu.sin.def.nom	noun neuter singular definite nominative
nn.neu.sin.def.gen	noun neuter singular definite genitive
nn.neu.sin.ind.gen	noun neuter singular indefinite genitive
nn.neu.sin.ind.nom	noun neuter singular indefinite nominative
nn.neu.plu.def.gen	noun neuter plural definite genitive
nn.neu.plu.ind.gen	noun neuter plural indefinite genitive
nn.neu.plu.def.nom	noun neuter plural definite nominative
nn.neu.plu.ind.nom	noun neuter plural indefinite nominative
pm.nom	proper noun nominative
ab	adverb
vb.prs.akt	verb present active
pn.neu.sin.def.sub/obj	pronoun neuter singular definite
ab.pos	adverb positive
jj.kom.utr/neu.sin/plu.ind/def.nom	adjective comparative non-neuter/ neuter singular/plural indefinite/ definite nominative
kn	conjunction
mad	sentence separation punctuation
pn.utr.sin.def.sub	pronoun non-neuter singular definite subject form
vb.sup.akt	verb past perfect participle active
vb.inf.akt	verb infinitive active
dt.utr.sin.ind	determiner non-neuter singular indefinite
jj.pos.utr.sin.ind.nom	adjective positive non-neuter singular indefinite nominative
mid	punctuation

Appendix D. Levenshtein Distance

Illustration 13 shows Levenshtein Distance that was used in this work.[33]

```
public int calcLevenshteinDistance(String s1, String s2) {

    int x = s1.length() + 1;
    int y = s2.length() + 1;
    int[][] matrix = new int[x][y];

    for (int i = 0; i < x; ++i) {
        matrix[i][0] = i;
    }

    for (int i = 0; i < y; ++i) {
        matrix[0][i] = i;
    }

    for (int i = 1; i < x; ++i) {
        for (int j = 1; j < y; ++j) {

            if (s1.charAt(i-1) == s2.charAt(j-1)) {
                matrix[i][j] = matrix[i-1][j-1];
            } else {
                int del = matrix[i-1][j] + 1;
                int ins = matrix[i][j-1] + 1;
                int sub = matrix[i-1][j-1] + 1;
                matrix[i][j] = Math.min(Math.min(del,ins),sub);
            }

        }
    }

    return matrix[x-1][y-1];
}
```

Illustration 13: Levenshtein Distance written in Java.

Appendix E. The removed suffixes in the stemmer

If a word belongs to the following word classes from Talbanken05 or hunpos and it ends with corresponding suffixes listed to the right, the suffix is removed from the word.[32] [40][41] See Appendix B. and Appendix C. for explanation of the abbreviations of the word classes.

AB	re, st
SP	anden, andes, endes, ande, ende, nde
TP	ersta, erst, aste, ast, are, ste, sta, re, st, dd, de, d, t, a, e
VV	dde, te, de, er, tt, it, r, t
AJ	ersta, erst, aste, ast, are, ste, sta, re, st, sk, ad, ig, a, e, t
NN	ion, ing, tik, ik
nn.neu.plu.def.gen	arnas, ernas, enas, rnas, ars, ens, nas, s
nn.neu.-.-.-	ets, ts, et, t, s
nn.neu.plu.ind.gen	ars, ers, ens, rs, ns, s
nn.neu.plu.def.nom	arna, erna, ena, rna, ar, en, na
nn.neu.plu.ind.nom	en, ar, er, n
nn.neu.sin.def.gen	ets, ts, s
nn.neu.sin.ind.gen	es, s
nn.neu.sin.def.nom	et, t
nn.neu.sin.ind.nom	e
nn.utr.-.-.-	ens, ns, en, n, s
nn.utr.plu.def.gen	arnas, ernas, ornas, nas, s
nn.utr.plu.ind.gen	ars, ers, ors, s
nn.utr.plu.def.nom	arna, erna, orna, na
nn.utr.plu.ind.nom	ar, er, or, r
nn.utr.sin.def.gen	ans, ens, ns, s
nn.utr.sin.ind.gen	as, es, s
nn.utr.sin.def.nom	an, en, n
nn.utr.sin.ind.nom	e, a
All words	heterna, hetens, heter, heten, arens, andet, arne, aren, ades, erns, ade, het, ern, at, lig, els, ig
All words. Removes only the last letter if the word ends with any of these suffixes.	löst, fullt, dd, gd, nn, dt, gt, kt, tt