UPTEC X 10 022
Examensarbete 30 hp
Oktober 2010

Implementation and evaluation of a text extraction tool for adverse drug reaction information

Gunnar Dahlberg

UPPSALA UNIVERSITET

Bioinformatics Engineering Program

Uppsala University School of Engineering

UPTEC X 10 022	**Date of issue 2010-10**

Author

Gunnar Dahlberg

Title (English)

Implementation and evaluation of a text extraction tool for adverse drug reaction information

Title (Swedish)

Abstract

A text extraction tool was implemented on the .NET platform with functionality for preprocessing text (*removal of stop words*, *Porter stemming* and use of *synonyms*) and matching medical terms using permutations of words and spelling variations (*Soundex*, *Levenshtein distance* and *Longest common subsequence distance*). Its performance was evaluated on both manually extracted medical terms (semi-structured texts) from summary of product characteristics (SPC) texts and unstructured adverse effects texts from Martindale (i.e. a medical reference for information about drugs and medicines) using the WHO-ART and MedDRA medical term dictionaries. Results show that sophisticated text extraction can considerably improve the identification of ADR information from adverse effects texts compared to a verbatim extraction.

Keywords

Text extraction, Adverse drug reaction, Permutation, Soundex, Levenshtein distance, Longest common subsequence distance, Porter stemming

Supervisors

Tomas Bergvall
Uppsala Monitoring Centre

Niklas Norén
Uppsala Monitoring Centre

Scientific reviewer

Mats Gustafsson
Uppsala University

Project name	Sponsors
Language **English**	Security
ISSN 1401-2138	Classification
Supplementary bibliographical information	Pages **66**

Biology Education Centre Biomedical Center Husargatan 3 Uppsala
Box 592 S-75124 Uppsala Tel +46 (0)18 4710000 Fax +46 (0)18 471 4687

Implementation and evaluation of a text extraction tool for adverse drug reaction information

Gunnar Dahlberg

Sammanfattning

Inom ramen för Världshälsoorganisationens (WHO:s) internationella biverkningsprogram rapporterar sjukvårdspersonal och patienter misstänkta läkemedelsbiverkningar i form av spontana biverkningsrapporter som via nationella myndigheter skickas till Uppsala Monitoring Centre (UMC). Hos UMC lagras rapporterna i VigiBase, WHO:s biverkningsdatabas. Rapporterna i VigiBase analyseras med hjälp av statistiska metoder för att hitta potentiella samband mellan läkemedel och biverkningar. Funna samband utvärderas i flera steg där ett tidigt steg i utvärderingen är att studera den medicinska litteraturen för att se om sambandet redan är känt sedan tidigare (tidigare kända samband filtreras bort från fortsatt analys). Att manuellt leta efter samband mellan ett visst läkemedel och en viss biverkan är tidskrävande.

I den här studien har vi utvecklat ett verktyg för att automatiskt leta efter medicinska biverkningstermer i medicinsk litteratur och spara funna samband i ett strukturerat format. I verktyget har vi implementerat och integrerat funktionalitet för att söka efter medicinska biverkningar på olika sätt (utnyttja synonymer, ta bort ändelser på ord, ta bort ord som saknar betydelse, godtycklig ordföljd och stavfel). Verktygets prestanda har utvärderats på manuellt extraherade medicinska termer från SPC-texter (texter från läkemedels bipacksedlar) och på biverkningstexter från Martindale (medicinsk referenslitteratur för information om läkemedel och substanser) där WHO-ART- och MedDRA-terminologierna har använts som källa för biverkningstermer. Studien visar att sofistikerad textextraktion avsevärt kan förbättra identifieringen av biverkningstermer i biverkningstexter jämfört med en ordagrann extraktion.

Examensarbete 30hp—Oktober 2010
Civilingenjörsprogrammet i Bioinformatik
Uppsala Universitet

Implementation and evaluation of a text extraction tool for adverse drug reaction information

Gunnar Dahlberg

Abstract

Background: Initial review of potential safety issues related to the use of medicines involves reading and searching existing medical literature sources for known associations of drug and adverse drug reactions (ADRs), so that they can be excluded from further analysis. The task is labor demanding and time consuming.

Objective: To develop a text extraction tool to automatically identify ADR information from medical adverse effects texts. Evaluate the performance of the tool's underlying text extraction algorithm and identify what parts of the algorithm contributed to the performance.

Method: A text extraction tool was implemented on the .NET platform with functionality for preprocessing text (*removal of stop words*, *Porter stemming* and use of *synonyms*) and matching medical terms using permutations of words and spelling variations (*Soundex*, *Levenshtein distance* and *Longest common subsequence distance*). Its performance was evaluated on both manually extracted medical terms (semi-structured texts) from summary of product characteristics (SPC) texts and unstructured adverse effects texts from Martindale (i.e. a medical reference for information about drugs and medicines) using the WHO-ART and MedDRA medical term dictionaries.

Results: For the SPC data set, a verbatim match identified 72% of the SPC terms. The text extraction tool correctly matched 87% of the SPC terms while producing one false positive match using *removal of stop words*, *Porter stemming*, *synonyms* and *permutations*. The use of the full MedDRA hierarchy contributed the most to performance. Sophisticated text algorithms together contributed roughly equally to the performance. Phonetic codes (i.e. *Soundex*) is evidently inferior to string distance measures (i.e. *Levenshtein distance* and *Longest common subsequence distance*) for fuzzy matching in our implementation. The string distance measures increased the number of matched SPC terms, but at the expense of generating false positive matches. Results from Martindale show that 90% of the identified medical terms were correct. The majority of false positive matches were caused by extracting medical terms not describing ADRs.

Conclusion: Sophisticated text extraction can considerably improve the identification of ADR information from adverse effects texts compared to a verbatim extraction.

KEY WORDS: Text extraction, Adverse drug reactions, Permutation, Soundex, Levenshtein distance, Longest common subsequence distance, Porter stemming

Master Thesis 30hp—October 2010
Master of Science Bioinformatics Engineering
Uppsala University

Contents

Vocabulary

API Application Programming Interface

ADR Adverse Drug Reaction

CSV Comma-Separated Values

DBMS Database Management System

DLL Dynamically Linked Library

EMA European Medicines Agency

GUI Graphical User Interface

IC Information Component

ICSR Individual Case Safety Report

LCS Longest Common Subsequence

MedDRA Medical Dictionary for Regulatory Activities

SPC Summary of Product Characteristics

TextMiner TextMiner is the name of the application developed as part of this master thesis project.

UMC Uppsala Monitoring Centre

VigiBase™ The world's largest database of spontaneous individual case reports of suspected adverse drug reactions (contains over 5 million reports).

WHO World Health Organization

WHO-ART World Health Organization Adverse Reaction Terminology

XML eXtensible Markup Language

1 Introduction

This master thesis in medical bioinformatics has been conducted at Uppsala Monitoring Centre (UMC), the WHO Collaborating Centre for International Drug Monitoring in Uppsala. Supervisors for the project are Tomas Bergvall, research engineer, and Niklas Norén, manager of the research department. The project is about implementing and evaluating the performance of a text extraction tool that can isolate adverse drug reaction (ADR) information from free texts. Such a tool would provide valuable support in the initial review of potential drug safety signals at UMC.

1.1 Adverse Drug Reaction Surveillance

Pre-market clinical trials are limited in both time and scope. The post-market monitoring of drugs is therefore vital for establishing drug safety [25]. The WHO Programme for International Drug Monitoring was established in 1968 aiming to assess and monitor risks of drugs and other substances used in medicine to improve public health worldwide. Since 1978, UMC has the scientific and technical responsibility for the WHO programme. UMC is responsible, on behalf of WHO, for collecting, monitoring, analyzing and communicating drug safety information to member countries of the WHO programme [19]. The network of member countries has steadily grown from 10 in 1968 to 100 full member countries as of September 2010 [20]. The global individual case safety report (ICSR) database, VigiBase™, is the world's largest ICSR database [25] and contains reports submitted to the center since the WHO programme was initiated in 1968. The case reports are provided by physicians, other health care professionals and patients from member countries of the WHO programme [3]. As of September 2010, there are more than 5 million reports in VigiBase™.

One of the main responsibilities of UMC is to detect and communicate drug safety issues to all the national centers participating in the WHO programme [20]. On a quarterly basis UMC performs routine data mining of VigiBase™ to find new safety signals according to the WHO definition of a *signal*:
"Reported information on a possible causal relationship between an adverse event and a drug, the relation being unknown or incompletely documented previously. Usually more than one report is required to generate a signal, depending on the seriousness of the event and the quality of the information"[8].

The large size of the data set requires automatic methods for finding associations effectively. UMC uses a range of data mining and medical rule-based algorithms to mine pharmacovigilance data in order to find new potential drug-ADR signals [3, 25, 28]. The data mining systems allow for an automated way of highlighting those drug-ADR associations in VigiBase™ that require further

attention [28]. The automated method is a major improvement of the previous, completely manual, signal detection process and is today used as a routine method for detecting potential drug-ADR signals [3].

Potential signals are reviewed by the signal detection team at UMC. Initial review includes checking the quality of ICSRs and manual literature checks where medical literature is studied to see what is already known about the drug and the ADR. Already known and reported drug-ADR combinations are down-prioritized. Much time at UMC is spent on searching medical literature for known and well-described drug-ADR combinations. UMC estimates that in 2009 around 70-100 hours were spent on checking literature and the quality of ICSRs per signal cycle (last year there were 4 cycles), the major part being literature checking (Richard Hill, personal communication, September 17, 2010).

Remaining combinations are sent to a panel of international drug safety experts for in-depth review. Combinations that remain after the expert panel's clinical review fulfill the WHO signal definition and are summarized and reported in a quarterly released SIGNAL document. It is distributed to all national drug safety centers and pharmaceutical companies participating in the WHO Programme for International Drug Monitoring [28]. Figure 1 describes a schematic overview of the signal detection work flow at UMC.

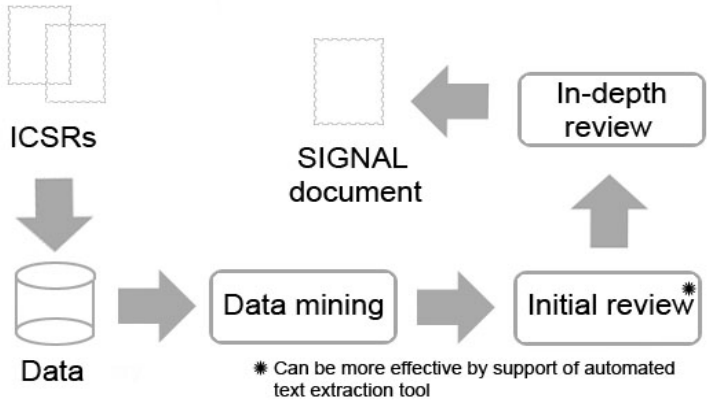

Figure 1: **Signal detection process**. Schematic overview of the sequential steps involved in the signal detection process at UMC. Potential drug-ADR combinations are detected by data mining methods—Bayesian disproportionality-based methods (using information component (IC) measure) and medical triage analysis [3, 28]. Initial review includes manual literature checks to search for and filter away known and described drug-ADR combinations. A text extraction tool can make this step more effective. In-depth review is performed by a panel of clinical experts.

Around 50% of all potential signals are filtered out because they are already in the literature (Richard Hill, personal communication, September 17, 2010).

The manual process can be improved: a text extraction tool to quickly find ADR information in medical texts would provide valuable support and enhancement. The tool is not supposed to replace human efforts, but to improve the speed and provide easy access to valuable and relevant information during the signal evaluation assessments. At UMC there is an earlier implementation of a text extraction algorithm considering some of the functionalities included in our text extraction tool [6]. Our implementation adds support for identifying ADR terms with spelling mistakes, has improved the efficiency for identifying ADR terms with word permutations and is developed on the .NET platform, the current production environment at UMC (the earlier implementation was developed in Perl).

1.2 Text Extraction

In medical research the amount of new published articles with data pertaining to protein, gene and molecular interactions/functionalities increases rapidly. Much information is given as unstructured texts. Individual researchers are often unable to keep up with the fast pace of new information accumulation [11]. To handle all the data, systems have been developed to automatically extract knowledge about proteins, genes and other molecular interactions and relationships from the text of published articles and store the information in databases in a computer readable format [1, 7, 11, 29]. There are systems to automate extraction of molecular pathways [11], protein-protein interactions [12, 29] and gene/protein biological functions from unstructured text [1]. When the information is stored in a structured form, it allows for further analysis of the data. This provides an approach to manage the high rate of new information and making it available to researchers in a more accessible way.

It is easy to acknowledge the potential of a text extraction system that is able to parse and understand text. The task is, however, complex and daunting. The complexity originates from the fact that words can have more than one meaning (polysemy) and more than one word can be used to express a meaning (synonyms)—word and meaning have a many-to-many relationship. Natural language is also very flexible, it evolves rapidly—grammatical rules are stretched, new words are added, new modern expressions emerge and old expressions may be dropped [16]. The high rate of change makes it difficult to develop parsers that will last for a longer period of time. Sentences in text can often have more than one possible parse and determining the correct one requires additional information as context or other prior knowledge. In many cases, the single parsing of a sentence cannot be determined due to the ambiguous nature of the text [16].

Text extraction is a first step to perform text mining. Common functions of text mining applications are clustering/categorization of documents (documents within the same cluster are related based on certain characteristics), summarization of

documents and trend analysis [16]. Text mining tools have applications in various areas that share the characteristic of handling large volumes of text. Data mining and text mining share many common characteristics—both aim at finding hidden information (i.e. patterns, relationships, trends) in large sources of data using algorithms from machine learning, statistics and artificial intelligence. The big difference is that data mining deals with structured numerical data whereas text mining deals with unstructured text. In both text and data mining the results are heavily dependent on the source data.

1.3 Objective

The aim of this thesis project is to develop a text extraction tool that can be used to identify ADR information from unstructured text in existing literature sources. Different techniques for preprocessing and matching text will be used and evaluated in terms of number of false positive drug-ADR associations and number of drug-ADR associations missed by the algorithm.

To summarize, the objective of the project is to:

1. Implement a text extraction tool on the .NET platform to extract ADR information from free text

2. Evaluate the performance of the tool's underlying algorithm on adverse effects texts and extracted semi-structured medical terms under different parameter settings

3. Identify what parts of the algorithm contributed most to the performance

1.4 Outline

The report is organized in six main chapters. The first chapter provides the reader with an introduction to the fields of ADR surveillance and text extraction and states the objective of the project. The second chapter covers background theory of the algorithms used within the text extraction algorithm. The third chapter discusses the material and methods— it presents all data set sources and the text extraction algorithm. The fourth chapter presents technical results from implementation and results from the evaluation of the text extraction algorithm performance. In the fifth chapter we discuss the results and highlight some interesting areas for future work. The last chapter concludes the study. Five appendices provide supplementary information. Appendix A describes a classical permutation algorithm implemented in the text extraction tool, Appendix B summarizes the text extraction algorithm parameters, Appendix C gives the stop words and synonyms used in the algorithm performance evaluation, Appendix D shows the

graphical user interface (GUI) of TextMiner (the developed application) and Appendix E discusses how to use the *TextMiningTools* dynamically linked library to execute the text extraction algorithm through code.

2 Background—Algorithms

This section covers theory of the text algorithms used within the text extraction algorithm to provide its different matching capabilities. The theory will serve as background knowledge to the reader when the text extraction algorithm is explained in the next section.

2.1 Removal of Stop Words

The text extraction algorithm makes use of a list of stop words. Stop words are words that do not contain any significant information, e.g. prepositions and conjunctions such as *and*, *in*, *if*, *else*, *or*, *but* etc. Other words that have no meaning in the specific text search can also be included in the list. Stop words are removed from the medical texts and terms to reduce the number of words for the text extraction process.

2.2 Synonyms

Synonyms are words that are regarded as equivalent in the text extraction process. Medical texts and terms can contain words with different word stems that still share the same meaning, e.g. *decrease* and *lower*, *convulsion* and *seizure* etc. A list of synonyms is used by the text extraction algorithm to give a possibility to match terms with text where the words are completely different but still share the same meaning.

2.3 Stemming

Stemming is a process of removing suffixes from words and is often used in information retrieval (IR) systems [27]. In such systems, there is typically a collection of documents where each document is described by a vector of terms (words) extracted from the respective document. Similar terms with different suffixes often have the same meaning and by using suffix removal those terms can be grouped into the same term group based on equal word stems. This increases the performance and reduces the number of unique terms in the IR system which results in lower data complexity [27].

We use the Porter stemming algorithm for suffix removal on all words in the medical texts and on the words in the medical terms. The algorithm uses a list of suffixes and for each single suffix there are specific rules for when to remove it from a word to leave only the stem. Porter points out that the algorithm certainly will not give successful results in all cases. There are English words where the suffix completely changes the meaning of the word (Porter gives the example of 'probe' and 'probate') and in those cases it will be wrong to remove the suffix. There are also words where the word stem changes with the addition of a suffix (e.g. index and indices) [27].

The algorithm consists of several steps, where each step contains specific suffix stripping rules. A word to be stemmed will pass the different steps of the algorithm sequentially. Simple suffixes will be stripped in a single step whereas more complex suffixes will be stripped one part at a time by several steps. We do not intend to describe the details of all suffix stripping rules involved in the algorithm, but merely give the reader knowledge about the functionality the algorithm provides. For full cover of the algorithm and the suffix stripping rules for each step, please see the detailed description by Porter [27]. We use a freely available implementation of the Porter Stemming algorithm written in C# [2].

Table 1 shows examples of words and the results from applying the Porter stemming algorithm.

Table 1: **Porter stemming**

Original word	**Stemmed word**
hepatitis	hepat
generalizations	gener
hepatic	hepat
dyskinesia	dyskinesia
convulsions	convuls
hypertrichosis	hypertrichosi
osteoporosis	osteoporosi

Example of words and results from applying the Porter stemming algorithm. Note that both *hepatitis* and *hepatic* are conflated into the same word *hepat*. *Dyskinesia* is not affected by the Porter Stemming algorithm. Not all words will be affected by the algorithm. In a vocabulary of 10,000 words tested by Porter, there were 3650 words that were unaffected by the Porter stemming algorithm [27].

2.4 Permutation

In our application we have a need to find permutations of the words that constitute a medical term. In a medical term the same word can sometimes occur multiple times. The task is therefore to find complete permutations of a set or multiset of words. We have implemented two algorithms. First, a classical algorithm to generate the possible permutations in lexicographical order (see Appendix A).

A second algorithm uses a completely different scheme for finding term permutations. Instead of first generating all possible permutations of a medical term (using the algorithm above) and then performing the search with all permutations, a search is performed for each individual word within the medical term. All word matches within the medical text are stored together with the position in the text where they were found. When all individual words within the medical term have been used for searching we need to check all the found word matches. If there are word matches for each single word within the term and these word matches are positioned in the text within the length of the medical term, then we have found a term permutation.

2.5 Approximate String Matching

Approximate string matching is used to enhance information retrieval of free text. To utilize approximate string matching we need a method to assess whether two strings are similar. We must also be able to quantify the similarity or difference [10]. The text extraction algorithm provides approximate string matching by using two string distance measures (*Levenshtein distance* and *LCS distance*) and phonetic codes (*Soundex*). String distance measures are based on calculating a numerical estimate of the differences between two strings. These calculations are based on either the number of character differences between the character sequences of two strings or the number of edits that is required to transform one string into the other [31]. Phonetic codes transforms strings into phonetic codes and the strings are considered similar if their corresponding phonetic codes are identical [31].

2.5.1 Soundex

The Soundex algorithm is a phonetic algorithm (i.e. words are coded based on their pronunciation). The original Soundex algorithm was invented by Robert C. Russell and Margaret K. Odell and patented in the beginning of the 20th century [26]. There are several variations of the algorithm. The simplified Soundex algorithm became popular after being described by Donald Knuth [14]. It is the version used in our text extraction tool. We use a freely available implementation of the

simplified Soundex algorithm written in C# [4]. The simplified Soundex algorithm applies a set of well-defined rules to a string to generate a four-character code consisting of a letter followed by three digits. The rules are based on the English pronunciation of words. The point is that words with similar pronunciation will receive identical Soundex codes. This allows for fuzzy string matching based on the pronunciation of words instead of their exact literal spelling. A limitation of the simplified Soundex algorithm is that it only applies to English words, i.e. applying it to words of other languages will not give any meaningful results. An outline of the steps in the simplified Soundex algorithm is as follows [14]:

- All characters in the string except the English letters A to Z are ignored.
- Extract the first letter in the string. It is the first letter in the Soundex code.
- Transform the remaining characters to digits according to the following rules:
  - 0: A, O, U, E, I, H, W or Y
  - 1: B, F, P or V
  - 2: C, G, J, K, Q, S, X or Z
  - 3: D or T
  - 4: L
  - 5: M or N
  - 6: R
- When adjacent digits are the same, remove all digits except for one.
- Remove all zero characters.
- As a final step, force the Soundex code to be 4 characters long by appending zero characters if it is too short or use truncation if it is too long.

Below follows a couple of examples to illustrate how the algorithm works.

Example 1 *Let us say we want to assign a Soundex code to* **LEXICON**.
The first letter L is extracted to be the first letter of the Soundex code. Next, transforming remaining characters to digits give ***020205****. No adjacent digits are the same. Removal of zero characters gives* ***225****. The Soundex code is* ***L225****. No truncation or appending of zero characters is needed.*

Example 2 *Assign a Soundex code to* **DICTIONARY**.
The first letter D is extracted. Next, remaining characters are transformed to digits, which give us ***023005060****. Removal of adjacent digits results in* ***02305060****. Removal of zero characters gives us* ***2356****. The Soundex code is now* ***D2356****. As a final step, we force the Soundex code to be 4 characters long. Thus, the final Soundex code becomes* ***D235****.*

2.5.2 Levenshtein Distance

The Levenshtein distance is an edit distance metric for measuring the difference between two strings. The allowed edit operations are insertion, deletion and substitution of a single character. The Levensthein distance is the minimum number of edit operations (insertions, deletions and substitutions of single characters) required to turn one string into the other [17].

Our implementation of computing Levenshtein distance of two strings uses a dynamic programming approach to solve the task. A. Lew and H. Mausch define dynamic programming as *"a method that in general solves optimization problems that involve making a sequence of decisions by determining, for each decision, subproblems that can be solved in like fashion, such that an optimal solutions of the original problem can be found from optimal solutions of subproblems"* [18]. Dynamic programming approaches have been applied to numerous problems within bioinformatics involving string processing and sequencing. For dynamic programming to become computationally efficient the subproblems should be overlapping such that results from subproblems only need to be computed once and can then be reused within the algorithm [18].

Below is pseudo code for our implementation of computing Levenshtein distance of two strings.

```
1   FUNCTION LevenshteinDist(first : STRING, second : STRING) : INTEGER
2    SET m to length[first]
3    SET n to length[second]
4    SET editMatrix[0,0] to 0
5
6    FOR i = 1 to m
7     SET editMatrix[i,0] to editMatrix[i-1,0] + 1
8    FOR j = 1 to n
9     SET editMatrix[0,j] to editMatrix[0,j-1] + 1
10
11   FOR i = 1 to m
12    FOR j = 1 to n
13     SET option1 to editMatrix[i-1,j] + 1
14     SET option2 to editMatrix[i,j-1] + 1
15
16     IF (first[i-1] == second[j-1])
17      SET editCost to 0
18     ELSE
19      SET editCost to 1
20
```

```
21      SET option3 to editMatrix[i-1,j-1] + editCost
22      SET editMatrix[i,j] to min(option1,option2,option3)
23
24    RETURN editMatrix[m,n]
25  END FUNCTION
```

As seen in the pseudo code a matrix is used to store results of subproblems. In this way results of new subproblems can be computed with the help of results of previously calculated subproblems. First there is an initialization step where the matrix is initialized—the upper left position is assigned 0 and the cells in the first row and first column is initialized with values incremented with 1 for each step. The algorithm iterates over all rows in the matrix. For each row it iterates over all columns and computes the Levenshtein distance of substrings up to this position. To compute the Levenshtein distance value for a particular cell, it uses the previously computed Levenshtein distances of shorter substrings. The final Levenshtein distance for the two input strings is equal to the value found in the lowest right cell of the matrix.

In our implementation we assume a penalty of 1 for deletions, substitutions and insertions. The algorithm and the dynamic programming solution were developed in the Soviet Union within the area of coding theory [17]. In the literature there are examples of extending the basic algorithm in different ways [21]. One example is to allow substitutions, deletions and insertions to have different weights depending on the characters involved in the edit operation. To develop and define penalty matrices for edit operations one can consider adjacency of characters on the keyboard [10].
The example below shows how to reason to find the Levenshtein distance measure.

Example 3 *Compute Levenshtein distance of* **ABDOMINAL** *and* **NOMINAL**. **ABDOMINAL** *can be transformed into* **NOMINAL** *by deletion of the first character* **A**, *the second character* **B** *and substitution of the third character* **D** *to* **N**. *Therefore the Levenshtein distance is 3.*

2.5.3 Longest Common Subsequence

The longest common subsequence (LCS) is another measure that can be used to quantify the difference between two strings and hence be used in approximate string matching [24]. A subsequence differs from a substring—in a subsequence the characters do not have to be consecutive. The text extraction algorithm has an implementation of computing the longest common subsequence of two strings. Below is pseudo code for our implementation.

```
1   FUNCTION LCS(first : STRING, second : STRING) : INTEGER
2    SET m to length[first]
3    SET n to length[second]
4    FOR i = 0 to m
5        FOR j = 0 to n
6           IF (i == 0 or j == 0)
7            SET resultMatrix[i,j] to 0
8           ELSE IF (first[i-1] == second[j-1])
9            SET resultMatrix[i,j] to
10                  resultMatrix[i-1,j-1] + 1
11          ELSE
12           SET resultMatrix[i,j] to
13                  max(resultMatrix[i-1,j], resultMatrix[i,j-1])
14
15   RETURN resultMatrix[m, n]
16  END FUNCTION
```

LCS is a similarity measure, where a higher value indicates a higher degree of similarity. Levenshtein distance, on the other hand, is a difference measure (i.e. higher values indicate a higher degree of difference). To transform LCS into a difference measure like Levenshtein distance, we use equation 1:

$$LCS_d = max(s_1.Length, s_2.Length) - LCS(s_1, s_2) \tag{1}$$

it guarantees, $0 \leq LCS_d \leq max(s_1.Length, s_2.Length)$

Below is an example of computing LCS and LCS_d for two strings.

Example 4 *Compute LCS and* LCS_d *of* **HORDOLEUM** *and* **HORDEOLUM**. *The longest common subsequence string is* **H**—**O**—**R**—**D**—**O**—**L**—**U**—**M**, *thus the longest common subsequence is 8. Both* **HORDOLEUM** *and* **HORDEOLUM** *have length 9, so* $LCS_d = 9 - 8 = 1$. *(***Note 1:** *the Levenshtein distance of* **HORDOLEUM** *and* **HORDEOLUM** *is 2. An* E *has to be both deleted and inserted to transform one string into the other.* **Note 2:** *the longest common substring is* **H**—**O**—**R**—**D**.*)*

3 Materials & Methods

The section begins with a discussion of the medical text and medical terminology sources that have been used. We then specify our implementation of the text extraction algorithm. The section then covers the methods used for evaluating the text extraction algorithm and implementing the text extraction tool.

3.1 Text Sources

The text extraction algorithm requires both a medical text source (consisting of free text descriptions of ADRs for drugs) and a medical terminology source as input data. Each medical term in the terminology is searched for in the medical text source.

The medical text source should consist of a list of medical entries where each entry must contain data for a specific drug and its ADR text. The ADR texts will be searched by the text extraction algorithm. In the project two different medical text sources have been utilized, namely Martindale: The Complete Drug Reference [23] and manually extracted Summary of Product Characteristics (SPC) texts.

The medical terminology source should consist of a list of medical terms. A terminology is defined as a *"set of terms representing the system of concepts of a particular subject field"*[13]. Terminologies can be simple enumerations of terms or have a more sophisticated organization where terms are assigned to classes, groups or categories [20]. WHO-ART and MedDRA were used as medical terminologies by the text extraction algorithm (both are hierarchical terminologies).

3.1.1 Martindale: the Complete Drug Reference

Martindale provides a medical reference for information about drugs and medicines of clinical interest internationally [30]. Besides providing information about drugs in clinical use other types of substances like vitamins, contrast media and toxic substances are included. The information about a substance is divided into sections that cover different properties and aspects. The type of information available vary among substances. A few examples of information that can be provided are molecular descriptions, interactions, pharmacokinetics, preparations, uses and administration, withdrawal, precautions, dependence, adverse effects and treatment of adverse effects. The first publication of Martindale came in 1883 [30]. The version of Martindale used in the project includes data up to October 2009 [23].

As discussed above, Martindale holds more information for each substance than is needed in our application. We are only interested in the known ADRs of each substance. Therefore as a first step the ADR texts for each substance must be isolated from Martindale. The extracted texts can then be provided as input to the text extraction algorithm (see figure 2).

Martindale is available in electronic format as an XML-file [23]. In the XML, a coding system with prime numbers is used to distinguish information in sections as belonging to different categories. Sometimes a specific section will contain information which is a combination of several categories. In those cases the individual prime numbers are multiplied. The extraction of ADR texts is accomplished in code by examining the prime numbers of sections and extracting the text for

sections containing adverse reaction information.

***Figure 2:* ADR texts isolation from Martindale**. A schematic view of the ADR texts isolation from Martindale. The process generates a list of substances and their respective ADR text. This list serves as input to the text extraction algorithm.

3.1.2 Extracted SPC Texts

SPC texts are found on the leaflet accompanying a drug when purchased. It provides information about dosage, manufacturer, usage, precautions, adverse effects etc. The data set of SPC texts used for the project originates from the European Medicines Agency (EMA). Each SPC text (belonging to a specific drug) has been preprocessed by manual extraction of ADR terms from the adverse effects section. The data set therefore consists of one or more texts (i.e. one for each extracted ADR term) for each drug.

As described above, the nature of the extracted SPC texts will differ from the ADR texts extracted from Martindale:

- Each text consists of text for what is assumed to be one isolated ADR term.
- The texts are much shorter.

3.1.3 WHO-ART

WHO Adverse Reaction Terminology (WHO-ART) is a medical terminology dictionary maintained by UMC (see figure 3). It has been developed specifically for the WHO Programme for International Drug Monitoring [20].

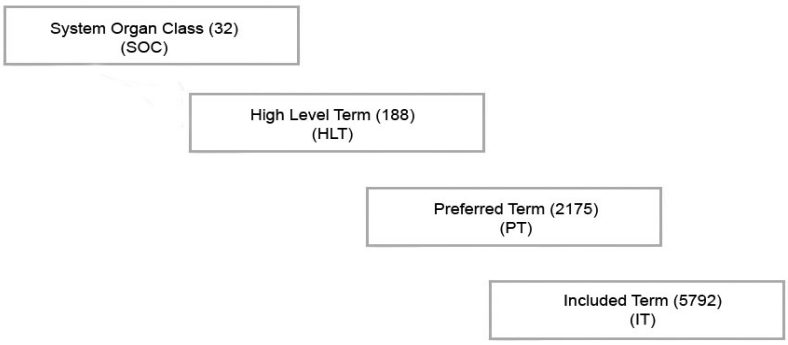

Figure 3: **WHO-ART hierarchy**. The WHO-ART hierarchy consists of four levels. The numbers in parenthesis show the number of terms for the respective hierarchy level (as of September 2010). Included terms are most detailed. The preferred terms (PTs) often allow precise identification of a reaction [20]. The set of terms for different levels can have overlapping elements.

3.1.4 MedDRA

In 1989 the UK Medicines Control Agency (MCA) identified a need for a new medical terminology to assist in storage of drug regulation data. This marks the start of the development of the Medical Dictionary for Regulatory Activites (MedDRA). MedDRA covers symptoms, diagnoses, therapeutic indications, adverse drug reactions, medical procedures and more. It is developed with the aim to provide a single comprehensive and internationally accepted medical dictionary to be used in pre- and postmarket regulatory processes [5].

MedDRA is structured into an hierarchical tree with five levels, see figure 4.

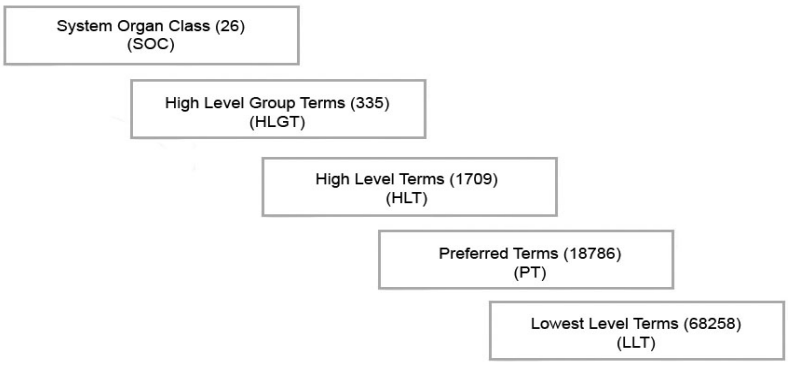

Figure 4: **MedDRA hierarchy**. A schematic view of the five levels in the MedDRA hierarchy. The number of terms for each hierarchy level in the latest version of MedDRA (version 13.0 from March 2010) is shown in parenthesis.

3.2 Text Extraction Algorithm

As described earlier, the algorithm requires a list of ADR texts and a medical term dictionary as input data. It has several parameters that will set up and affect how the text extraction process will be performed (see Appendix B for a complete list of parameters including descriptions).

The high level steps of the algorithm are:

- Initialize all algorithm parameters (if there are parameters not set, default values will be read from an XML-file)
- Sort all the medical terms (in descending order based on the number of letters they contain)
- Clean all medical terms (replace all non-alphanumeric characters in the text with spaces and replace long stretches of space with a single space character)
- Preprocess all medical terms (possibly using *removal of stop words*, *Porter stemming*, *Soundex*, *synonyms*)
- Go through each ADR text. For each text:
  - Clean ADR text (same procedure as for medical terms)
  - Preprocess the ADR text (using the same methods as when preprocessing the medical terms)
  - Search the preprocessed text using the preprocessed terms to get a list of matches (possibly using *permutation*, *Levenshtein distance*, *LCS distance*)
  - Filter away partial text matches, i.e. matches that cover a percentage of the words in the text below a certain threshold (if setting is turned on)
  - Filter out the *"best"* matches when matches are overlapping in the text (i.e. matches that originate from positions in the original text that overlap with each other; see section 3.2.1 for how the *"best"* matches are determined)
  - Search the original ADR text using the sorted original medical terms to get all verbatim matches (sorting is needed because found matches are removed from the searched text during the search, thus the order in which the terms are searched is important).
  - Mark each found term match as a verbatim match if it exists in the set of found verbatim matches

The cleaning step ensures that the text only consists of consecutive stretches of alphanumeric characters split by single space characters. It allows for a simple way of extracting words from the text—words are extracted by splitting the text on space characters. The Porter stemming algorithm and the Soundex algorithm both require the text to be partitioned into words, since the algorithms work with one word at a time.

Appendix E shows a code example to illustrate how to set up the algorithm parameters, listen to algorithm events and start a text extraction algorithm by code.

3.2.1 Function to determine *"best"* match

The text extraction algorithm uses a function to determine the *"best"* match for overlapping matches in the text (i.e. matches that originate from positions in the original text that overlap with each other). By default, the *"best"* match of two overlapping matches is given by:

- For each match: Compute a *"hit success value"*. It is calculated using equation 2:

$$HitSuccessValue = length(s_1) - 2 * mt_d \tag{2}$$

  where, s_1 denotes the matched string in the preprocessed ADR text

  mt_d denotes the matched preprocessed medical term's distance measure (i.e. the *Levenshtein distance* or *LCS distance* between s_1 and the matched preprocessed medical term)

- The match with the highest *"hit success value"* is the best.
- If both matches have the same *"hit success value"*, we check whether the matched medical terms contain stop words. The matched medical term that do not contain stop words is the *"best"*. If none or both contain stop words, they are considered equally good.

The above is the default implementation, however it is possible to define a custom function to determine the *"best"* match (see the HitComparerMethod parameter in Appendix B for details).

3.3 Text Extraction Algorithm Performance Analysis

The performance of the text extraction algorithm was analyzed on the SPC and Martindale data sets. Performance was evaluated in terms of precision and recall. A high precision ensures a low number of false positive drug-ADR associations.

A high recall ensures that as few positive drug-ADR associations as possible are missed by the algorithm. Results from both data sets were therefore analyzed to check the amount of correctly matched medical terms, false positives (i.e. falsely reported matches of medical terms) and unmatched medical terms (i.e. medical terms missed in the ADR texts) generated.

To evaluate and provide an objective count of the number of false positives, a framework with criteria for when a medical term match was considered correct and not needed to be established. For the analysis, the following criteria were used:

1. The matched medical term has a different meaning than the matched text → **False positive**

2. The matched medical term has the same meaning but is less detailed (more general) → **Correct match**

   Consider the examples: (text on the left, matched medical term on the right)
   Primary graft dysfunction → *Graft dysfunction*
   Genital pain male → *Genital pain*
   Unstable angina pectoris → *Angina pectoris*
   Oral soft tissue disorder → *Soft tissue disorder*
   Cerebral adverse reaction → *Adverse reaction*

3. The medical term is more detailed (i.e. specific) than the matched text → **False positive**

To illustrate the reasoning behind point 2 and 3 above, a *soft tissue disorder* **is not** necessarily an *oral soft tissue disorder* but an *oral soft tissue disorder* **is a** *soft tissue disorder*. Thus *oral soft tissue disorder* → *soft tissue disorder* is a correct match, whereas *soft tissue disorder* → *oral soft tissue disorder* is not (i.e. false positive). For point 2 we acknowledge that a more detailed medical term had been preferable, but we still consider the match as correct.

The algorithm can report more than one ADR term as a match for overlapping texts. This occurs when both matches are considered equally good by the algorithm (see section 3.2.1). As a consequence, there are many situations when evaluating the SPC data set that an extracted SPC term will be matched to several ADR terms, e.g. the SPC term *Metabolic acidosis* can be matched to both *Metabolic acidosis* and *Acidosis metabolic*.

Counts on number of unmatched ADR terms were provided by comparing the found ADR terms by the algorithm with manual extraction of ADR terms from the ADR texts. The manual identification was performed by a M.Sc. in pharmacy to provide a gold standard.

All runs included in the performance analysis that use the settings *removal of stop words* or *synonyms* have used the stop word list and synonyms list that can be found in Appendix C (the synonyms list is the same as one used in a previous text extraction implementation at UMC [6]). All runs that used *permutations* used the permutation algorithm based on storing the position of single word matches (see section 2.4)

3.3.1 SPC Data Set

The SPC data set consists of a total of 4270 extracted ADR terms from SPC texts of 75 different drugs. There are several cases where the same extracted ADR term occurs for multiple drugs in the data set. To avoid redundancies, the SPC data set was prepared by removing all duplicate extracted ADR terms. The formatted SPC data set consisted of 1785 unique extracted ADR terms.

15 text extraction runs were performed using the non-redundant SPC data set for the algorithm performance analysis. As medical terminology, runs used MedDRA Preferred Terms or all unique terms from the MedDRA hierarchy. Runs that used a string distance measure to allow for approximate string matching (i.e. *LCS distance* or *Levenshtein distance*) used, admittedly arbitrary, a 15% cut-off value on term distance and 25% cut-off value on word distance. The term and word distance cut-off values set the maximum allowed percentage deviation (i.e. percentage calculation based on character differences) for a single word or complete term match respectively (see Appendix B).

Since the nature of the SPC data set differs from the Martindale data set (each text consists of what is assumed to be one extracted medical term, see section 3.1.2), a *restriction on partial text matches* was set when running the performance analysis on the SPC data set (i.e. a threshold was imposed on minimum percentage of the words in the text that need to be matched by a term to result in a match). It limits the number of reported partial SPC matches. For the runs on the SPC data set, a *partial match restriction* of either 100% (to find verbatim matches) or 60% (a threshold value that required a little bit more than half of the words in the SPC term to be matched) was used.

The results of each run were analyzed to check the amount of correctly matched, false positives and unmatched SPC terms generated. For consecutive runs, algorithm parameters were set stepwise using forward selection within groups (i.e. the best combination of algorithm parameters were kept from each group). The algorithm parameters were tested using the following groups and order:

1. *Term list* (using MedDRA Preferred Terms or all unique terms within the MedDRA hierarchy) and *Restrict partial text matches* (using a 100% or 60% threshold)

2. *Removal of stop words*, *Permutations* and *Stemming*
3. *Synonyms*
4. *LCS distance*, *Levenshtein distance* (both using a 15% and 25% cut-off value on term distance and word distance respectively)

3.3.2 Martindale Data Set

5 text extraction runs were performed on the extracted ADR texts in Martindale. The results for each run were extensive and a complete analysis of all results from of each run was not possible due to limited resources. Therefore, to control and assess the quality of the results, a random sample of 10 ADR texts was drawn from the results of one text extraction run. A clinical evaluation of the extracted ADR terms from the texts was performed by a domain expert (M.Sc. in pharmacy). Two of the ADR texts in the random sample were strongly overlapping and therefore one was omitted from the evaluation. Hence, 9 ADR texts were evaluated. The domain expert was available to perform the clinical evaluation of the results at a point when the functionality for *synonyms* was not implemented. Therefore, the text extraction run for the clinical evaluation used *removal of stop words*, *stemming* and *permutations*, but not *synonyms*. WHO-ART Preferred Terms were used as term list.

For the 4 other runs, statistics were calculated for minimum, maximum, mean and median number of matched medical terms per text. These runs used the parameter settings *removal of stop words*, *stemming*, *permutations* and *synonyms*. For these runs, we varied the term list to evaluate its impact on the results—WHO-ART Preferred Terms, MedDRA Preferred Terms and all unique terms in the WHO-ART and MedDRA hierarchy were used respectively.

3.4 Implementation Methods for TextMiner

The text extraction system was developed using the .NET platform. The production systems at UMC are developed on .NET and using the same technical platform allows for easier integration of the text extraction solution. Robert Martin describes principles used when deciding how to design the system into components and manage the dependencies between components [22]. The system was designed so that the text extraction algorithm functionality was collected in a standalone component within the system with a well-defined interface. It allows the component to be easier maintained and reused as part of other .NET implementations.

A graphical user interface using Windows Forms technology was developed to allow user interaction with the system. Data to support and run the text extraction algorithm as well as output from the algorithm were stored in a SQL Server 2008

database. A data layer to provide functionality for the communication with the database was also created.

3.4.1 High-level Architecture

Figure 5 shows a high-level architecture of the text extraction system. First a collection of ADR texts and medical terms stored in a dictionary are fed as input to the system. A text preprocessing step transforms the ADR texts and medical terms to a preprocessed form. Then preprocessed texts and terms are used as input for the term extraction. The preprocessed medical terms are searched for and extracted from the preprocessed text. Finally the extracted terms are used to generate drug-ADR associations. The user interacts with the system through a user interface to set up, start, stop text extraction runs etc.

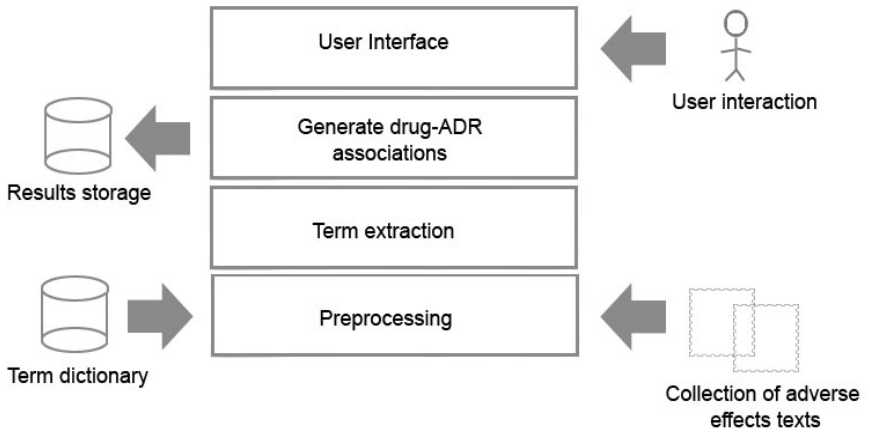

Figure 5: **System architecture**. A high-level conceptual view of the text extraction system and how the different functionalities fit together.

4 Results

The results section is divided into two parts. The first part presents results from the text extraction performance analysis on the SPC and Martindale data sets. The second part describes the results of the technical implementation.

4.1 Text Extraction Algorithm Performance Analysis

The section presents the results of the text extraction algorithm performance analyses of the SPC and Martindale data sets.

4.1.1 Summary of Product Characteristics Data Set

The SPC data set used for the performance analysis was a non-redundant version of the original SPC data set as outlined in section 3.3.1 (i.e. duplicate extracted ADR terms were removed). It consisted of 1785 unique extracted ADR terms. Table 2 shows results from 14 of the text extraction runs included in the performance analysis of the SPC data set. All runs used a cleaning step where non-alphanumeric characters were removed from the text (see section 3.2).

One more text extraction run was performed that used a 60% threshold on *partial match restriction*, *removal of stop word* and *Soundex* to allow approximate string matching. Initial evaluation of the results indicated that over 500 of the SPC matches were false positives. The total number of medical terms matched were above 5500—significantly higher than for the other runs. Since the number of false positives was very high and obvious reasons for this were identified (i.e. different medical terms were encoded identical Soundex codes) , we did not continue with an exhaustive results analysis.

Run 1 did not allow partial matches, i.e. 100% of the words in the extracted SPC term needed to be matched by the dictionary term to result in a match. It used the MedDRA Preferred Terms as term dictionary. It resulted in 987 SPC terms that found a match. 979 of the 987 matches were verbatim, thus 8 matches were non-verbatim. The non-verbatim matches were found due to the cleaning step in the algorithm where non-alphanumeric characters are removed from the medical terms and texts (see section 3.2 for details). Table 3 shows examples of non-verbatim matches.

Run 2 used the same settings as run 1 with the exception of a 60% threshold on *partial match restriction* and resulted in an additional 38 matched SPC terms. All of these additional matches were verbatim matches where the matching medical term was more general than the extracted SPC term (the medical term consisted of a subset of words found in the extracted SPC term). Table 4 gives some examples.

Run 3 did not allow partial matches and used all terms in the MedDRA hierarchy as term dictionary. It resulted in 1294 SPC terms being matched and 1287 of these were verbatim matches. The remaining 7 matches were the result of non-verbatim matches. Similar to run 1, these matches were all found due to the cleaning step of the algorithm.

Run 4 used a 60% threshold on *partial match restriction* (similar to run 2). It used all terms in the MedDRA hierarchy as term dictionary. Run 4 matched 322 additional SPC terms compared to run 2—a result of using all terms in the MedDRA hierarchy as opposed to the MedDRA Preferred Terms as term dictionary. Run

Table 2: **Results from SPC runs**

Run	Full MedDRA hierarchy	Allow partial text matches (60%)	Permutation	Remove stop words	Stemming	Synonyms	Levenshtein distance	LCS distance	SPC Terms Matched	SPC Terms Not Matched	False Positive SPC Matches	Total Term Matches	Total False Positive Matches
1									987	798	0	987	0
2		*							1025	760	0	1026	0
3	*								1294	491	0	1302	0
4	*	*							1347	438	0	1356	0
5	*	*	*						1392	393	0	1700	0
6	*	*		*					1402	383	1	1415	1
7	*	*			*				1437	348	0	1620	0
8	*	*	*	*					1450	335	0	1764	0
9	*	*		*	*				1492	293	1	1685	1
10	*	*	*		*				1486	299	1	2047	1
11	*	*	*	*	*				1546	239	1	2134	1
12	*	*	*	*	*	*			1561	224	1	2343	1
13	*	*	*	*	*	*	*		1582	203	10	2363	12
14	*	*	*	*	*	*		*	1583	202	10	2364	12

The table presents the results from running the text extraction algorithm on the SPC data set under different settings. For the *Full MedDRA hierarchy* column, an asterisk indicates that all unique terms in the whole MedDRA hierarchy were used, whereas no asterisk indicates that MedDRA Preferred Terms were used. For the *Allow partial text matches* column, an asterisk indicates that a 60% threshold on partial match restriction was used, whereas no asterisk indicates that a 100% threshold on partial match restriction was used. For all other columns, an asterisk indicates that the particular setting is turned on. For *Levenshtein distance* and *LCS distance*, the term and word percentage cut-off values used were always 15% and 25%, respectively (see section 3.3.1).

Table 3: **Examples of non-verbatim matches in run 1**

SPC term	Matched MedDRA term
Influenza-like illness	Influenza like illness
Stevens Johnson syndrome	Stevens-Johnson syndrome
Infusion-related reaction	Infusion related reaction
Post-procedural haemorrhage	Post procedural haemorrhage

Examples of non-verbatim matches found from run 1.

Table 4: **Examples of matches in run 2**

SPC term	Matched MedDRA term
Acute renal failure	Renal failure
Oral soft tissue disorder	Soft tissue disorder
Injection site pain / discomfort	Injection site pain
Upper abdominal pain	Abdominal pain
Pleural effusion symptoms	Pleural effusion

Examples of matches that are found in run 2 due to the lowered threshold on *partial match restriction* compared to run 1 (i.e. the threshold is 60% in run 2 and 100% in run 1).

4 matched 53 additional SPC terms compared to the results of run 3. All other matches were identical to the results of run 3. All of the additional 53 matches were verbatim matches–reported as a result of lowering the threshold compared to run 3 (matching medical term was more general than the extracted SPC term).

Run 5 applied the same settings as run 4 with the addition of using *permutations* (i.e. it used *partial match restriction* with a 60% threshold and *permutations*). Run 5 gave different results for 348 extracted SPC terms compared to run 4, which were caused by the use of *permutations*. 45 extracted SPC terms reported a match that did not find any match in run 4. In all of these cases, the match was non-verbatim and was found using a permutation of the words in the term. Table 5 shows some examples.

Table 5: **Examples of permutation matches in run 5**

SPC term	Matched MedDRA term
Abnormal respiration	Respiration abnormal
Interstitial nephritis	Nephritis interstitial
Intracranial bleeding	Bleeding intracranial
Reduced visual acuity	Visual acuity reduced
Thyroxine (T4) free decreased	Free T4 decreased
Sudden unexplained death	Sudden death unexplained

Examples of matches found by using *permutations* in run 5.

In 13 cases the use of *permutations* resulted in a more specific (i.e. detailed) medical term being matched than without *permutations* (see table 6 for examples).

Run 6 applied the same settings as run 4 with the addition of using *removal of stop words* (i.e. it used *partial match restriction* with a 60% threshold and *removal of stop words*). The results differed for 66 SPC terms. For 55 SPC terms there was a match in run 6 where none were found for run 4. Most of the matches came from treating *"nec"* and *"nos"* as stop words and removing them from medical terms (there are several terms within MedDRA that contain these abbreviations, which stand for *"not elsewhere specified"* and *"not otherwise specified"* respectively). Other differences were found because *removal of stop words* from the extracted SPC terms reduced the number of words and allowed medical terms to match within the partial match restriction of 60%. Matches were also found because of *removal of stop words* in medical dictionary terms. Table 7 gives examples of all cases.

Run 6 resulted in 1 false positive (*Reduced visual acuity* → *Low visual acuity*). Even though *reduced visual acuity* may result in *low visual acuity*, the meaning of

Table 6: **More specific permutation matches in run 5**

SPC term	**Matched MedDRA term (run 4)**	**Matched MedDRA term (run 5)**
Chronic otitis media serous	Otitis media serous	Otitis media serous chronic, Chronic serous otitis media
Unstable angina pectoris	Angina pectoris, Unstable angina	Angina pectoris unstable
Increased creatine phosphokinase	Creatine phosphokinase	Creatine phosphokinase increased
Increased blood triglycerides	Blood triglycerides	Blood triglycerides increased
Increased alanine aminotransferase	Alanine aminotransferase	Alanine aminotransferase increased

Examples where more specific medical terms where matched to the SPC terms by the use of *permutations* (run 5 used *permutations* but run 4 did not).

Table 7: **Examples of matches in run 6**

SPC term	**Matched MedDRA term**
Dental disorder	Dental disorder NEC
Mood disorders	Mood disorders NEC
Appetite increased	Appetite increased NOS
Leucocytosis	Leucocytosis NOS
Renal failure/impairment	Renal failure and impairment
Local swelling at the injection site	Swelling of injection site
Tightness of the chest	Tightness in chest
Increase in heart rate	Heart rate

Examples of cases where matches were found in run 6 and none where found in run 4 (run 6 used *removal of stop words* but run 4 did not).

the terms is not exactly the same.

Run 7 applied the same settings as run 4 except for the use of *stemming* (i.e. it used *partial match restriction* with a 60% threshold and *stemming*). The results differed for 255 extracted SPC terms compared to run 4. For 90 SPC terms there were matches where there were none in run 4. In some cases differences occured because of *stemming* which resulted in several medical terms being mapped to an SPC term. *Stemming* also caused differences because more detailed medical terms were matched. Table 8 shows examples of all cases.

Run 8 used *partial match restriction* with a 60% threshold, *removal of stop words* and *permutations*. Matches were found for an additional 58 SPC terms compared to run 5 and 48 SPC terms compared to run 6. No false positive matches were generated. The results differed for 20 SPC terms compared to the results of both run 5 (using *permutations*) and run 6 (using *removal of stop words*). All of these differences were due to the combination of using *removal of stop words* and *permutations*. An investigation of all differences shows that 11 SPC terms had no match for both run 5 and 6, 3 cases had different matches for all three runs and 6 SPC terms had no match for run 5 but matches for both run 6 and 8. Table 9 shows examples of all cases where the results differ.

Run 9 used *partial match restriction* with a 60% threshold, *removal of stop word* and *stemming*. Results showed that 90 additional SPC terms were matched compared to run 6 and 55 additional SPC terms were matched compared to run 7. For 24 SPC terms, the result differs from both the result of run 6 and 7—these differences were due to the combination of using *removal of stop words* and *stemming*. In 14 of these cases, there were no matches found when using *removal of stop words* or *stemming* alone. In 4 cases there were matches in run 6 and 9 but none in run 7. In the 6 remaining cases, a new medical term match was found due to the combination. The same false positive match was reported as for run 6 (i.e. *Reduced visual acuity* $\rightarrow$ *Low visual acuity*). Table 10 shows examples of cases where the results differ.

Run 10 used *partial match restriction* with a 60% threshold, *stemming* and *permutations*. An additional 94 SPC terms were matched compared to run 5 and 49 additional SPC terms were matched compared to run 7. For 107 SPC terms, the result differs from both the result of run 5 and 7—these differences were due to the combination of using *stemming* and *permutations*. For 7 SPC terms, there were no matches found when using *stemming* or *permutations* alone. In most cases, differences were reported because new medical term matches were found due to the combination. One false positive was reported—Gastro-oesophageal re-

Table 8: **Examples of matches in run 7 compared to run 4**

SPC term	**Matched MedDRA term (run 4)**	**Matched MedDRA term (run 7)**
Burning	—	Burn
Optical neuritis	—	Optic neuritis
Allergies	—	Allergy
Thromboembolic	—	Thromboembolism
Injection site nodules and cysts	—	Injection site nodule
ECG investigations abnormal	—	Investigation abnormal
Cerebrovascular adverse reactions	—	Adverse reaction
Blood pressure fluctuations	Blood pressure	Blood pressure fluctuation
Liver function tests abnormalities	Liver function tests	Liver function tests abnormal, Liver function test abnormal
Hallucination	Hallucination	Hallucination, Hallucinations, Hallucinating
Itching	Itching	Itch, Itching
Encephalopathy	Encephalopathy	Encephalopathy, Encephalopathies
Blisters	Blisters	Blisters, Blistering, Blister

The first seven rows show examples of SPC terms where no match is found in run 4, but a match was found in run 7 due to *stemming*. Rows six and seven are examples of cases where the matching was vague (however, considering our criteria for false positives (see section 3.3) these were considered correct). The next two rows are examples of when *stemming* results in more detailed medical term matches. The last four rows are examples of when *stemming* results in more than one match for a specific SPC term.

Table 9: **Examples of matches in run 8 compared to run 5 and 6**

SPC term	**Matched MedDRA term (run 5)**	**Matched MedDRA term (run 6)**	**Matched MedDRA term (run 8)**
Acute and chronic pancreatitis	Acute and chronic pancreatitis	Acute and chronic pancreatitis	Acute and chronic pancreatitis, Pancreatitis acute on chronic
Fungal infection (vaginal only)	—	Fungal infection	Vaginal infection
Reactivation of hepatitis B	—	Hepatitis B	Hepatitis B reactivation
Exacerbation of multiple sclerosis	—	Multiple sclerosis	Multiple sclerosis exacerbation
Tightness of the chest	—	Tightness in chest	Chest tightness
Impaired coordination or balance	—	—	Coordination impaired
Haemorrhage of operative wound	—	—	Operative haemorrhage
Elevated creatinine in blood	—	—	Blood creatinine
Urination abnormal	—	—	Urination abnormal NOS
Liver disorders	—	—	Other disorders of liver
Sudden sleep onset	—	—	Sudden onset of sleep

Examples of SPC terms where the results differed in run 8 compared to the result of both run 5 and 6. Run 8 used *partial match restriction* with a 60% threshold, *removal of stop words* and *permutations*. Run 5 used *partial match restriction* with a 60% threshold and *permutations*. Run 6 used *partial match restriction* with a 60% threshold and *removal of stop words*.

Table 10: **Examples of matches in run 9 compared to run 6 and 7**

SPC term	**Matched MedDRA term (run 6)**	**Matched MedDRA term (run 7)**	**Matched MedDRA term (run 9)**
Allergic and anaphylactoid reactions	—	—	Anaphylactoid reaction
Abnormal hair texture	—	—	Abnormalities of the hair
Elevation of liver enzymes	—	—	Elevated liver enzymes
Increase in serum creatinine level	—	—	Increased serum creatinine
Decrease in haemoglobin	—	—	Decreased haemoglobin
Dental disorders	Dental disorders NEC	—	Dental disorders NEC, Dental disorder NOS
Tightness of the chest	Tightness in chest	—	Tight chest
Pain in extremity	Pain in extremity	Pain in extremity	Pain of extremities, Pain in extremity
Gastrointestinal stenosis and obstructions	Gastrointestinal stenosis	Gastrointestinal stenosis and obstruction	Gastrointestinal stenosis and obstruction, Gastrointestinal stenosis and obstruction NEC

Examples of SPC terms where the results differed in run 9 compared to the result of both run 6 and 7. Run 9 used *partial match restriction* with a 60% threshold, *removal of stop words* and *stemming*. Run 6 used *partial match restriction* with a 60% threshold and *removal of stop words*. Run 7 used *partial match restriction* with a 60% threshold and *stemming*.

flux → Reflux oesophagitis. However, *Gastro-oesophageal reflux* can cause *Reflux oesophagitis*. Table 11 shows examples of cases where the results differ.

Run 11 applied *partial match restriction* with a 60% threshold, *removal of stop word*, *stemming* and *permutations*. The results were the same as for run 9, except for 447 extracted SPC terms. One false positive was reported among the results—the same as for run 10. The false positive reported in run 9 was not reported here due to the use of *permutations*. An investigation of the differences revealed that most were due to more than one medical term being matched to the SPC terms because of the addition of *permutations*. There were 54 extracted terms where a match was found but no match was established in run 9. These additional matches were found due to the additional use of *permutations*. Table 12 gives examples.

Run 12 applied *synonyms* (see Appendix C for the list of synonyms used) in addition to the settings in run 11 (i.e. it used *partial match restriction* with a 60% threshold, *removal of stop word*, *stemming*, *permutations* and *synonyms*). The result differed for 95 SPC terms compared to run 11. In 15 cases, an SPC term was matched where no match was established in run 11. Many differences were due to more reported matches for a specific extracted term due to the use of *synonyms*. Table 13 gives examples of both cases. One false positive was reported—the same as for run 10 and 11.

Run 13 applied *Levenshtein distance* (with a term and word distance threshold of 15% and 25% respectively) in addition to the settings in run 12 (i.e. it used *partial match restriction* with a 60% threshold, *removal of stop word*, *stemming*, *permutations*, *synonyms* and *Levenshtein distance*). The results differed from run 12 for 30 SPC terms. The results reported 10 false positive SPC matches. Table 14 shows examples of where the results differ.

Run 14 used the same settings as run 13 except for using *LCS distance* instead of *Levenshtein distance* as a distance measure, but with the same term and word distance thresholds (i.e. it used *partial match restriction* with a 60% threshold, *removal of stop word*, *stemming*, *permutations*, *synonyms* and *LCS distance*). The results of run 14 were the same as the results of run 13, but with one exception—the SPC term ***Hordoleum*** matched the medical term ***Hordeolum*** (in run 13 there was no match for ***Hordoleum***).

A pie chart was created (see figure 6) to illustrate the contributions of individual parts of the text extraction algorithm to the results of the SPC data set. As discussed in section 3.3.1, a specific order was used for the forward selection of algorithm parameters within groups. The chart was created by subsequently adding the parameter setting, within a group, that contributed the most to the

Table 11: **Examples of matches in run 10 compared to run 5 and 7**

SPC term	**Matched MedDRA term (run 5)**	**Matched MedDRA term (run 7)**	**Matched MedDRA term (run 10)**
Malignant neoplasms	—	—	Neoplasm malignant
Prolonged erections	—	—	Erection prolonged
Abnormal taste	—	—	Taste abnormality
Skin reactions	—	Skin reaction	Reaction skin, Skin reaction
Optical neuritis	—	Optic neuritis	Neuritis optic, Optic neuritis
Increased weight	Weight increased	—	Weight increase, Weight increased
Breast pain	Breast pain, Pain breast	Breast pain	Breast pain, Painful breasts, Pain breast
Gastro-oesophageal reflux	Oesophageal reflux	Oesophageal reflux	Oesophageal reflux, Reflux oesophagitis

Examples of SPC terms where the results differed in run 10 compared to the result of run 5 and run 7. Run 10 used *partial match restriction* with a 60% threshold, *permutations* and *stemming*. Run 5 used *partial match restriction* with a 60% threshold and *permutations* and run 7 used *partial match restriction* with a 60% threshold and *stemming*. The first three rows give examples of SPC terms where a match was only found due to the combination of *stemming* and *permutations*. The next four rows give examples where more MedDRA terms are matched to the SPC terms when both *stemming* and *permutations* were used. The final row gives example of the false positive SPC match.

Table 12: **Examples of matches in run 11 compared to run 9**

SPC term	**Matched MedDRA term (run 9)**	**Matched MedDRA term (run 11)**
Sudden unexplained death	—	Sudden death unexplained
Burning at the injection site	—	Injection site burning
Stinging at the injection site	—	Injection site stinging
Eyelid inflammation	—	Other inflammations of eyelids, Inflammation of eyelids
Skin discolouration	Skin discolouration	Skin discolouration, Discolouration skin
Suicide attempt	Suicide attempt	Attempted suicide, Suicide attempt
Reduced visual acuity	Low visual acuity	Visual acuity reduced

Examples of SPC terms where the results differ in run 11 compared to the result of run 9. The false positive reported in run 9 is correct for run 11, see last row. Run 11 used *partial match restriction* with a 60% threshold, *removal of stop words*, *stemming* and *permutations*, whereas run 9 used *partial match restriction* with a 60% threshold, *removal of stop words* and *stemming*. One false positive was reported—the same as for run 10.

Table 13: **Examples of matches in run 12 compared to run 11**

SPC term	**Matched MedDRA term (run 11)**	**Matched MedDRA term (run 12)**
Heart failures	Heart failures, Heart failure, Failure heart	Heart failures, Failure heart, Heart failure, Cardiac failure
Nephropathy toxic	Toxic nephropathy, Nephropathy toxic	Toxic nephropathy, Nephrotoxicity, Nephropathy toxic
Acute liver failure	Acute liver failure	Acute liver failure, Acute hepatic failure
Elevated triglycerides	Elevated triglycerides	Raised triglycerides, Triglyceride increased, Elevated triglycerides
Oral/Oropharyngeal bleeding	—	Oropharyngeal hemorrhage, Oropharyngeal haemorrhage
Elevations of transaminases	—	Transaminases increased
Liver enzymes decreased	—	Hepatic enzyme decreased
Elevated amylase	—	Amylase increased
Hepatic function tests abnormal	—	Abnormal liver function tests, Liver function tests abnormal, Liver function test abnormal

Examples of SPC terms where the results differed in run 12 compared to the result of run 11. Run 12 used *partial match restriction* with a 60% threshold, *removal of stop words*, *stemming*, *permutation* and *synonyms*, whereas run 11 used *partial match restriction* with a 60% threshold, *removal of stop words*, *stemming* and *permutation*.

Table 14: **Examples of matches in run 13 and 14 compared to run 12**

SPC term	**Matched MedDRA term (run 12)**	**Matched MedDRA term (run 13 & 14)**
Lymphopaenia	—	Lymphopenia
Leukopaenia	—	Leukopenia
Pollakisuria	—	Pollakiuria
Leucocyturia	—	Leukocyturia
Dysgueusia	—	Dysgeusia
Hyperhydrosis	—	Hyperhidrosis
Loss of appetite	—	Appetite lost
Hypophosphotemia	—	Hypophosphatemia
Neutropaenia	—	Neutropenias, Neutropenia
Fatal cerebral haemorrhage	Cerebral hemorrhage, Cerebral haemorrhage, Haemorrhage cerebral, Hemorrhage cerebral	Cerebral hemorrhage fetal
Fatal bleeding	—	Fetal hemorrhage
Infusion-associated reactions	—	Reaction asocial, Asocial reaction
Local paraesthesia	—	Local anaesthesia, Anaesthesia local
Transient blurred vision	Blurred vision, Vision blurred	Blue vision transient
Blood creatinine phosphokinase increased	—	Blood creatine phosphokinase increasedavanza

Examples of SPC terms where the results differed in run 13 and 14 compared to the result of run 12. Run 13 and 14 both used *partial match restriction* with a 60% threshold, *removal of stop words*, *stemming*, *permutation*, *synonyms* and *Levenshtein distance* or *LCS distance* respectively. Run 12 used the same settings except for *Levenshtein distance* and *LCS distance*. The first 9 rows give examples where the use of a distance measure generated a match. The last 6 rows give examples of false positive matches.

results at the given point.

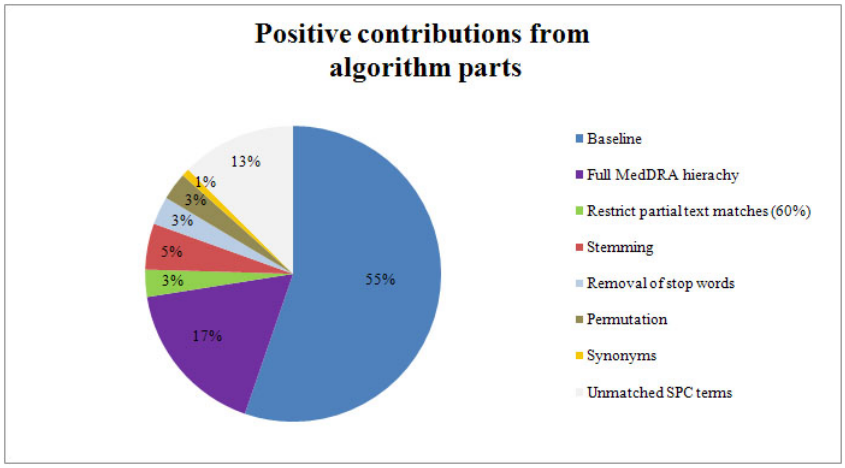

Figure 6: **SPC results**. A pie chart that shows the positive contribution to the SPC results (i.e. number of SPC terms matched) when individual algorithm parts are subsequently added to the algorithm. The baseline consists of run 1 (using MedDRA Preferred Terms as term list and a 100% threshold on partial text matches). The percentage values given indicate the additional portion of SPC terms matched when the specific parameter is added. As discussed in section 3.3.1, we used a specific order for the forward selection of algorithm parameters within groups. Note that the chart could have potentially looked different if algorithm parameters were selected using complete forward selection.

4.1.2 Martindale

Martindale holds information of a total of 5564 substances where 1716 of these have ADR texts [23]. A random sample of ADR texts were drawn from one text extraction run that used *removal of stop words*, *permutations*, *stemming* and WHO-ART Preferred Terms as term list (see section 3.3.2). Table 15 shows the results of the analysis of the text extraction algorithm output from 9 ADR texts in the random sample (one ADR text was omitted because the text was strongly overlapping with another text within the sample, see section 3.3.2). The results from each ADR text were analyzed by a domain expert to assess the number of correctly matched medical terms, false positives and unmatched medical terms. Example 5 illustrates the results from the clinical evaluation. It uses text parts extracted from the adverse effects text of *Meloxicam*, *Gatifloxacin* and *Glipizide*.

Table 15: **Results from random sample of texts in Martindale**

Substance	**Matched Terms**	**Unmatched Terms**	**Algorithmic F.P.** [(i)]	**Semantic F.P.** [(ii)]
Ceftazidime	23	12	0	1
Fosfomycin	9	5	0	0
Gatifloxacin	76	35	4	2
Glipizide	31	15	1	2
Meloxicam	67	37	1	12
Methadone hydrochloride	44	20	0	5
Procaine Benzylpenicillin	31	20	1	2
Telithromycin	27	14	0	1
Vancomycin	26	13	0	2
TOTAL	**334**	**171**	**7**	**27**

[(i)] The number of false positives that were caused by errors in the matching due to the algorithm.

[(ii)] The number of false positives that were caused by matching medical terms not describing ADRs.

The results from analyzing the output of the text extraction algorithm on 9 substances (i.e. ADR texts) in Martindale. For each substance, the number of correctly matched medical terms (i.e. the number of matched terms subtracted with the number of false positives), false positives and unmatched medical terms were determined by a domain expert. For each false positive, we distinguish between two types—those caused by faulty matching in the text extraction algorithm and those caused by correctly matching medical terms, but where the medical term is not describing an ADR in the text.

Example 5 *An example to show how the results of the clinical evaluation of the 9 random sample adverse effects texts looked like. To illustrate, we use 5 short text parts, taken from the adverse effects text of Meloxicam, Gatifloxacin and Glipizide. In the text parts, correctly matched medical terms, false positives and unmatched medical terms are highlighted differently—correctly matched medical terms are marked in bold, unmatched medical terms are underlined and false positives are both underlined and marked in bold. The first two texts parts are taken from*

Meloxicam, the next two from Gatifloxacin and the last text part from Glipizide. The results are summarized below each respective text part.

"...CNS-related adverse effects include ***headache****,* ***vertigo****,* ***dizziness****,* ***nervousness****,* ***tinnitus****,* ***depression****, drowsiness, and* ***insomnia****. Hypersensitivity reactions may occur occasionally and include* ***fever****,* ***angioedema****,* ***bronchospasm****, and* ***rashes****. Hepatotoxicity and aseptic* ***meningitis****, which occur rarely, may also be hypersensitivity reactions. Some patients may experience visual disturbances."*

Correct: headache, vertigo, dizziness, nervousness, tinnitus, depression, insomnia, fever, angioedema, bronchospasm, rash, meningitis
False positive: meningism
Unmatched: drowsiness, hypersensitivity reactions, hepatotoxicity, visual disturbances

"...Other risk factors for renal impairment with NSAIDs include ***dehydration****, cirrhosis, surgery,* ***sepsis****, and a history of* ***gout*** *or* ***hyperuricaemia****."*

Correct: -
False positive: dehydration, sepsis, gout, hyperuricaemia
Unmatched: -

"...Severe life-threatening events, including hyperosmolar nonketotic hyperglycaemic ***coma****,* ***diabetic*** *ketoacidosis,* ***hypoglycaemic coma****,* ***convulsions****, and mental status changes have been reported very rarely..."*

Correct: hypoglycaemic coma, convulsions
False positive: coma diabetic
Unmatched: hyperglycaemic coma, diabetic ketoacidosis

"...Reports have included anaphylaxis (which has sometimes been fatal, and may occur after the first dose), ***serum sickness****,* ***Stevens-Johnson syndrome****, toxic* ***epidermal necrolysis*** *(sometimes fatal),* ***laryngeal oedema****, and* ***vasculitis****.*

Correct: serum sickness, Stevens Johnson syndrome, epidermal necrolysis, oedema, vasculitis
False positive: laryngitis
Unmatched: anaphylaxis

"...reported an increased incidence in mortality from cardiovascular ***complications in diabetic*** *patients given tolbutamide..."*

Correct: -
False positive: diabetic complication
Unmatched: cardiovascular complications

The results from the other 4 text extraction runs performed on all 1716 ADR texts (see section 3.3.2) are shown in table 16. All runs used *removal of stop words*, *permutations*, *stemming*, *synonyms* (see Appendix C for list of synonyms used). For each run, it displays the min, max, median and mean number of matched medical terms per text.

Table 16: **Results from Martindale runs**

Run	Term dictionary	No. of terms	Total no. matched terms	Mean matched terms per text	Min matched terms per text	Max matched terms per text	Median matched terms per text	Running time
1	WHO-ART PT	2175	54348	31.7	0	144	26	4 h, 2 min, 4 s
2	WHO-ART HY	5824	75609	44.1	0	208	34	9 h, 1 min, 8 s
3	MedDRA PT	18786	81995	47.8	0	266	37	26 h, 55 min, 53 s
4	MedDRA HY	70328	164109	95.6	0	555	71	105 h, 20 min, 4 s

The results from running the text extraction algorithm on the Martindale data set. All runs used *removal of stop words*, *permutations*, *stemming*, *synonyms* (see Appendix C for list of synonyms used). All runs were performed using two threads on a computer with a 3.16GHz Intel Core 2 Duo E8500 (Dual core) CPU, 2 GB RAM and Windows 7 Professional 64-bit Operating system.

4.2 Technical Solution - TextMiner Application

TextMiner is the name given to the application implemented to fulfill the requirements set up in the methods of the project. TextMiner is built on the .NET platform using Visual Studio 2008 and .NET framework 3.5. (The section is divided into paragraphs describing the functionality, architecture, data model, parallelization and graphical user interface of the system).

4.2.1 Functionality

The TextMiner application has functionality to execute the text extraction algorithm and view the results. Below is a list of supported functionality:

- Load a term dictionary into the application (from a text file or CSV-file)
- Load a medical text source into the application (from a text file, CSV-file or XML-file (Martindale))
- Delete a loaded term dictionary or medical text source
- Set up parameters and start a text extraction run (see Appendix B for description of all the parameters)
- Stop an ongoing text extraction run
- View results from finished and aborted runs
- Search results of finished and aborted runs for specific drug substances and ADR terms

4.2.2 Architecture

For smaller applications, classes provide a sufficient unit of organization for structuring the code. On a higher level, packages are used to accomplish the task of organizing code. Packages in .NET are also called assemblies or dynamically linked libraries (DLL). By structuring the application into well-defined packages the design becomes more clean, the application becomes easier to maintain and it facilitates code reuse [22].

The TextMiner application has been partitioned into four packages. Figure 7 shows a graph of the packages and their dependencies. The arrows point in the direction of a dependency. This means that the *TextMiner* package is dependent on the *TextMinerCore* and *TextMinerGUI* packages, *TextMinerGUI* is dependent on *TextMinerCore* and *TextMinerCore* is dependent on *TextMiningTools*. The

TextMiningTools package thus has no dependencies and can therefore easily be reused as a single package in other .NET implementations.

The *TextMiningTools* package includes all classes that provide functionality for the text extraction algorithm. The package includes classes for Levenshtein distance, longest common subsequence, Soundex, Porter stemming, permutation, removal of stop words, handling of synonyms and for running the text extraction algorithm and retrieving results. The *TextMinerGUI* package contains all classes needed for the GUI. *TextMinerCore* consists of classes needed to send user parameters from the GUI, to consume the functionality within the *TextMiningTools* package and to store and retrieve data from a database instance.

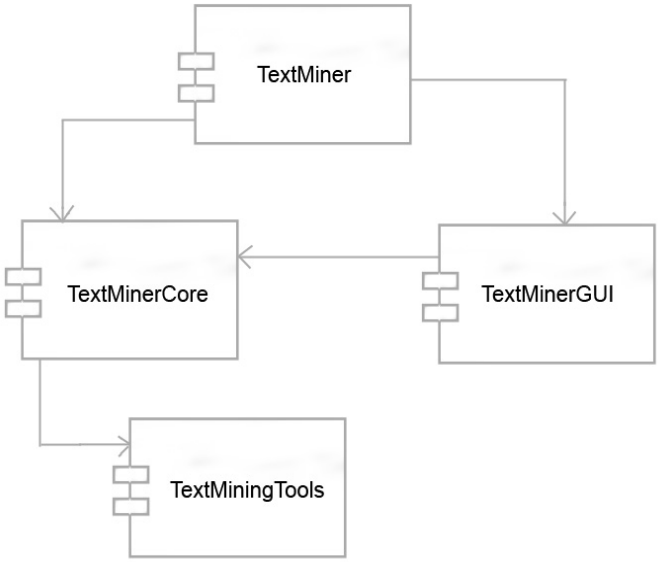

Figure 7: **TextMiner application packages**. A package dependency graph for the TextMiner application. Note that the graph is a directed acyclic graph (DAG).

4.2.3 Data Model

TextMiner used a relational data model implemented in the SQL Server 2008 database management system (DBMS) to store information. The database schema can be seen in figure 8. Medical text sources loaded into the application were stored in the *LexiconSource* and *LexiconEntry* tables. Loaded term dictionaries were stored in the *TermSource* and *Term* tables. When a new text extraction run was started by the user, information about it was stored in the *TextMiningRun* table. Information about which medical text source and term dictionary that were used for the run was provided by the foreign key relationships to the *LexiconSource* and

TermSource tables respectively. During a text extraction run, new results were saved continuously when extraction of a single medical ADR text was finished. The results were saved in the *TextMiningTermHit* and *TextMiningHit* tables. By continuously saving results, even runs aborted by the user stored results up to the point of abortion.

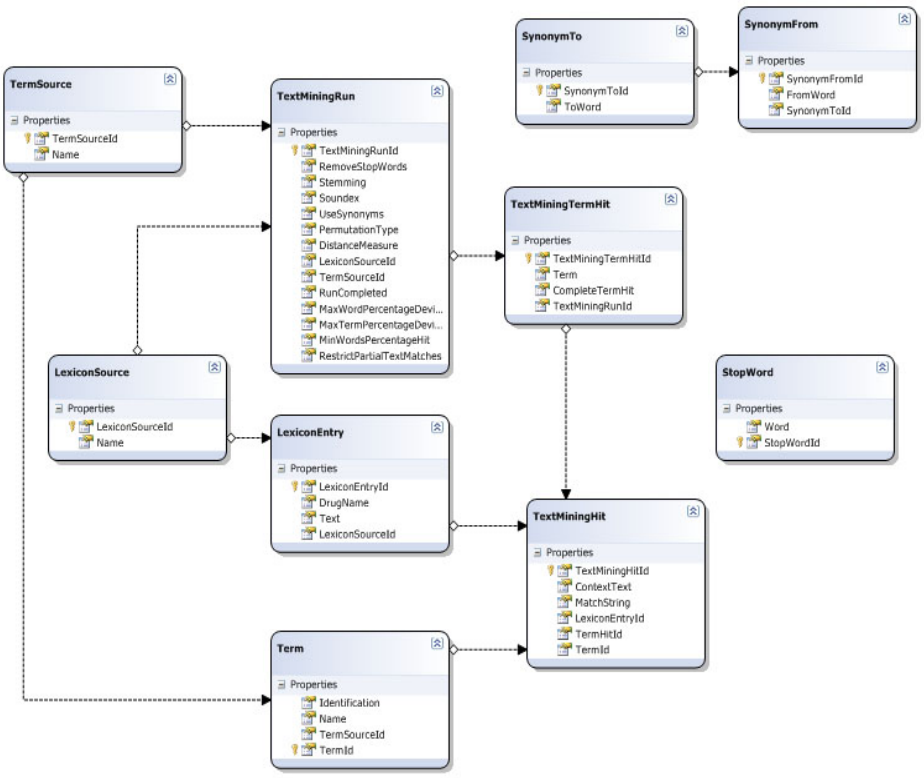

Figure 8: **TextMiner database schema**. An overview of the database schema. Shows all the tables and the columns for each table in the TextMiner database. It shows what type of data that can be stored. Foreign key relationships between tables are also indicated.

4.2.4 Parallelization

Support for parallelization has been implemented for the text extraction algorithm. The application programming interface (API) of the compiled TextMiningTools dynamically linked library exposes a public parameter called *NumberOfThreads* where the user can set the number of threads to be used by the algorithm (see Appendices A and D for further details). Two steps in the algorithm have been parallelized:

- The preprocessing of all terms in the term dictionary as outlined in section 3.2
- The search of all preprocessed terms in a preprocessed adverse effects text as described in section 3.2

Both steps are carried out on all terms in the term dictionary. The parallelization makes use of this and implements the same idea for parallelization—to divide the list of terms into one sublist for each thread. Each thread then carries out the task for the sublist of terms and at the end all results from individual threads are merged.

4.2.5 Graphical User Interface

A GUI was developed using Microsoft Windows Forms technology. The GUI was built on top of the text extraction system to allow for easy administration of text extraction algorithm runs. The user can easily load new term dictionaries and medical text sources. It is simple to set up the properties and variables for the text extraction algorithm. All finished and aborted runs can be viewed. Single runs can be searched by the user to see if particular ADR-terms were extracted from a specific medical text. The user can also easily switch medical text source and medical terminology source for a search. Results from runs can be viewed directly and exported to spreadsheet software for further processing. Appendix D provides screenshots of the graphical user interface.

5 Discussion

A text extraction tool was developed on the .NET platform to identify ADR information in adverse effects texts—it includes a GUI for user interaction, a data layer for storage and retrieval of data and a text extraction algorithm with functionality for preprocessing text (*removal of stop words*, *Porter stemming* and use of *synonyms*) and searching medical terms using permutations of words and spelling variations (*Soundex*, *Levenshtein distance* and *LCS distance*). Many parts of the algorithm gave a positive contribution to the results. For the SPC data set the best parameter settings—using *removal of stop words*, *stemming*, *permutations* and *synonyms*—reached a precision of 99.96% with a recall of 87% (a verbatim identification had a precision of 100% and a recall of 72%). In Martindale, the precision was 98% (accounting for semantic false positives; the precision not accounting for semantic false positives was 90%) with a recall of 64%. The use of a distance measure (i.e. *Levenshtein distance* or *LCS distance*) or *Soundex* did not give a positive contribution to the results—their use generated too many false

positive matches. The most substantial contribution to performance was from the use of full MedDRA hierarchy. Sophisticated text algorithms together contributed roughly equal performance impact.

The pie chart of the results of the SPC data set (see figure 6 in section 4.1.1) shows that the largest contribution to the results was from the use of full MedDRA hierarchy—accounting for an increase of matched SPC terms with 17%. The sophisticated text extraction algorithms (i.e. *restriction of partial text matches*, *stemming*, *removal of stop words*, *permutations* and *synonyms*) together contributed roughly equally to performance—together they contributed to a 15% increase of matched SPC terms.

The results showed that the text extraction algorithm generated more false positive matches on Martindale than on the SPC data set. The main reason was that the free adverse effects texts in Martindale posed an additional challenge compared to the extracted SPC terms; text semantics. 79% of the false positive matches were caused by text semantics, i.e. matching medical terms not describing ADRs. If the text extraction algorithm was able to distinguish ADRs from other contexts (i.e. indications, precautions, interactions etc.), the results on free unstructured adverse effects text could have been improved. We suggest further research to implement context based medical term extraction—one example would be to use sentence structure analysis, which has successfully been used as a step to extract gene-protein biological functions from free texts [1].

Accounting for semantic false positives, the precision was still slightly lower for Martindale than for the SPC data set. In the SPC data set, the SPC terms are isolated from each other, whereas for Martindale the medical terms can be found anywhere within the ADR texts. This poses a subtle problem for the text extraction algorithm when using Martindale—some false positives are generated from matching adjacent words in the text not belonging to ADR terms or belonging to different ADR terms, e.g. for the text *"...mortality from cardiovascular complications in diabetic patients..."* the algorithm matches *diabetic complication*, for the text *"...including hyperosmolar nonketotic hyperglycaemic coma, diabetic ketoacidosis, hypoglycaemic coma,..."* the algorithm matches *coma diabetic*. The recall for Martindale was lower than for the SPC data set (64% compared to 87%)—a reason for missing more medical terms in Martindale is the use of WHO-ART Preferred Terms (2175 unique terms) as term list compared to MedDRA hierarchy (70328 unique terms).

For the SPC data set, the use of string distance measures (i.e. *Levenshtein distance* and *LCS distance*) did not give a positive contribution to the results—slightly more than 20 additional SPC terms were matched at the expense of generating 9 false positives. We used a 15% cut-off value on term distance and 25% cut-off value on word distance (see section 3.3.1). The word distance cut-off needs to be set less

strict than the term distance cut-off (otherwise we potentially lose word matches that might had been possible to use to build term matches). The word distance cut-off was set to 25% —it allows many word matches but still limits the number of matches for performance reasons. An analysis of the false positive matches shows that often the match originates from 1 or 2 character differences, e.g. after preprocessing, *fatal* matches *fetal* (leading to the false positive match *fatal cerebral haemorrhage* → *Cerebral hemorrhage fetal*), *oper* matches *open* (leading to the false positive match *Haemorrhage of operative wound* → *Open wound*) and *blur* matches *blue* (leading to the false positive match *Transient blurred vision* → *Blue vision transient*). For the future, it would be interesting to try a combined *maximum number of character differences* and *percentage* cut-off—potentially it could decrease the number of false positives.

Stemming allows matching where the suffixes of words are different but conflated to the same stem. The use of *Levenshtein distance* or *LCS distance* allow matches where the words are similar enough (i.e. within a distance cut-off value). Hence, *stemming* and distance measures can achieve the same matching effects. However, for *stemming* the string differences are confined to the suffixes, whereas for distance measures the differences can occur anywhere in the strings. Using the two together can be problematic—seen in the false positives generated from run 13 and 14 on the SPC data set (discussed above). It would be interesting to evaluate a run that uses a distance measure but not *stemming*.

Soundex did not provide a positive contribution in our application. The results from the SPC data set demonstrate that *Soundex* generates many false positives and in that sense is inferior to string distance measures (*Levenshtein distance* and *LCS distance*) in approximate string matching in our implementation. The simplified Soundex algorithm encodes strings to four character codes always starting from the beginning of the strings. Hence, strings that have equal characters in the beginning but differs at the end are often encoded as identical Soundex codes. The approximate string matching with *Soundex* therefore proved to be too unspecific generating too many false positive matches, e.g. *hypothermia*, *hypoadrenalism*, *hypotropia*, *hypothrombinaemia* and *hypothromboplastinaemia* all have identical Soundex codes (H136).

The text extraction algorithm uses a *best* function to determine the *best* match for overlapping matches in the text (i.e. matches that originate from positions in the original text that overlap with each other) and only returns those matched terms. Another possibility would have been to return all medical term matches found. The reason for using a *best* function was to limit the returned matches, e.g. if the text contains *"...common adverse effect is abdominal pain..."* we only return *abdominal pain* instead of both *abdominal pain* and *pain*. Which approach is to be the preferred may vary between users and applications.

The results from Martindale show that the use of a more comprehensive term dictionary results in more medical terms being extracted per ADR text—mean, median and maximum matched medical terms per text all increase. However, the increase in matches is small when changing from WHO-ART hierarchy to MedDRA Preferred Terms considering the difference in size of these dictionaries—a likely cause is that many MedDRA Preferred Terms are not representing ADRs and would thus not be found in the ADR texts. Many terms within the Lowest Level Terms (LLT) of the MedDRA hierarchy are variations of spelling and permutations of words corresponding to Preferred Terms (PT). We will match all permutation variations of a term in the term dictionary as a result of using the MedDRA hierarchy in combination with *permutations*—an explanation for the large increase of matches in Martindale when using MedDRA hierarchy instead of MedDRA Preferred Terms with the same algorithm settings. As an example, the SPC term *Blood glucose increased* will be matched to the medical term *Blood glucose increased* when MedDRA Preferred Terms are used and *Glucose blood increased* and *Blood glucose increased* when MedDRA hierarchy is used as term dictionary. Thus, increased number of matched terms may not correspond to increased number of matches in the original text.

The Martindale results show that the execution time of the text extraction algorithm increases approximately linearly with the size of the medical term dictionary. Parallelization was implemented to improve the efficiency. However, an optimized search data structure for the preprocessed text would likely improve efficiency further. An example is to use a hash table to store the preprocessed text—the key could be the word and the values could be objects containing the word together with its position within the text. Each search for a medical term within the text would become more effective, but there would be an initial cost to create the data structure. However, when using string distance measures (i.e. *Levenshtein distance* or *LCS distance*) all keys in the hash table would be needed to be searched.

During the preprocessing step, the original text is transformed into a preprocessed form. All text extraction is performed on the preprocessed text. After a medical term match is found, the section of the original text where the match was found is extracted. However, there is at the moment no easy way of going from a position in the preprocessed text to the corresponding position in the original text. The section of original text where the match originated from is identified by counting the number of words before and after the match in the preprocessed text and isolate the original text between these word numbers. Since the preprocessing step always results in a text that has less or equally many words as the original text this method will perform a correct extraction. However, often the extracted original text will be unnecessary long, containing several additional sentences or

even paragraphs in addition to where the match originated from. A great improvement would be to allow for a simple way to go from positions in the preprocessed text to positions in the original text. This would allow for more accurate and precise extraction of match origin in the original text. In addition, it would simplify when determining if a matched medical term originates from a verbatim match or not—there would be no need for an additional search of the original medical text using the sorted original terms.

As the project progressed, a number of technical difficulties were encountered mainly related to the vast amount of data that needed to be processed—one difficulty was that the Martindale XML-file exceeded 250 MB in size. A first solution to parsing the XML-file was to use an existing .NET XML-parsing API to traverse the nodes. However, it required all nodes in the XML-file to be read into primary memory as a hierarchical tree and then traversing the in-memory tree to extract the ADR texts. It worked well for small XML-files, but did not prove feasible for the Martindale XML-file. The in-memory tree never got constructed correctly due to its large size and consequently no data could be extracted. A second solution was developed based on opening and reading the XML-file sequentially line by line until the end of the file was reached. For each read line the text was scanned for opening and closing of XML-tags of certain types that contain information to be extracted. This *"streaming"* solution proved to be successful for extraction of information from Martindale and can be used as a general approach when processing very large XML-files.

Another technical difficulty was that the output from one text extraction run could be several GBs in size. The text extraction algorithm was first developed to return an object containing all the search results of all input texts. No results were returned until the algorithm finished searching all the medical texts. For small and medium sized input data sets where the number of medical texts and the text lengths were limited the solution was successful. However, it was not working for larger data sets, such as the medical adverse effects texts in Martindale. The search result object grew steadily in size as the algorithm progressed, resulting in continuously increased memory allocation, which eventually resulted in an out of memory problem. To overcome the memory issues another mechanism for returning results from the algorithm was needed. A new solution was implemented based on the idea of returning results of single medical texts continuously as the algorithm progressed using an event mechanism. Hence, there was no build up of internal memory allocation by the algorithm and results were continuously provided for single ADR texts.

The implemented text extraction tool includes a limited set of functionalities and it would be possible to extend the text extraction functionalities so that additional preprocessing and approximate string matching algorithms could be used.

However, due to time limits, such extensions were not possible to include in the project.

Besides extending the text extraction functionalities, an interesting area for future research would be to use the output (i.e. the matched medical terms) from the text extraction algorithm to generate feature vectors for substances. The feature vectors could be viewed as finger prints of each substance's adverse reaction profile. For example, the vectors could be used to group substances based on their similarity of adverse reactions using either hierarchical or flat clustering (i.e. hierarchical clustering produces nested partitions whereas flat clustering produces a single partition with objects in disjoint groups) [9].

6 Conclusion

The results indicate that sophisticated text extraction can markedly improve the identification of ADR information from adverse effects texts compared to a verbatim extraction.

In a collection of semi-structured summary of product characteristics texts, the text extraction tool was able to increase recall to 87% compared to a verbatim identification with a recall at 72% while maintaining a high precision (99.96% for sophisticated extraction). Results from Martindale, a medical reference for information about drugs and medicines, show a high precision of 98% (90% if not accounting for semantic false positives). The majority of false positive matches (73%) where caused by extracting medical terms not describing ADRs. The most substantial contribution to performance was from the use of full MedDRA hierarchy. Sophisticated text algorithms together contributed roughly equally to performance. String distance measures (i.e. *Levenshtein distance* and *LCS distance*) and *Soundex* did not contribute positively to the results in our implementation.

7 Acknowledgements

This work would not have been possible without the help of others. First of all, I would like to thank my supervisors Tomas Bergvall and Niklas Norén for all valuable help, support and encouragement. Further, Johanna Strandell has been an excellent pharmacology resource, who contributed with checking the output from the text extraction algorithm on the Martindale data set.

My friend Andreas Zetterström for all our interesting discussions and for proofreading of the report. My wife, Annica, for sharing my journey and always believing in me and supporting me all the way.

I also would like to express my gratitude to all staff at the UMC research department for providing a friendly working environment and contributing in making my thesis work a great experience. Last but not least, I would also like to thank my scientific reviewer Mats Gustafsson for accepting to review the project.

References

[1] Asako, K., Yoshiki, N. and Toshihisa T., *Automatic extraction of gene/protein biological functions from biomedical text*, Bioinformatics, Oxford University Press, Vol. **21**, No. **7**, (2005), pp. 1227–1236.

[2] Azzopardi L., Porter Stemming algorithm implementation, University of Paisley, 2002, *http://tartarus.org/∼martin/PorterStemmer/csharp2.txt*, cited 2010-09-06.

[3] Bate, A., Lindquist, M., Edwards, I. R., Olsson, S., Orre, R., Lansner, A. and De Freitas, R. M., *A Bayesian neural network method for adverse drug reaction signal generation*, European Journal of Clinical Pharmacology, **54**, (1998), pp. 315–321.

[4] BlackWasp, The Soundex Algorithm, *http://www.blackwasp.co.uk/Soundex.aspx*, cited 2010-09-14.

[5] Brown E. G., Wood L. and Wood S., *The Medical Dictionary for Regulatory Activities (MedDRA)*, Drug Safety, **20**, (1999). pp. 109–117.

[6] Caster O., Automatic extraction of adverse drug reaction terms from medical free text, *http://www.iscb2009.info/RSystem/Soubory/Prez%20Wednesday/S33.1%20Caster.pdf*, cited 2010-09-28.

[7] Cohen, A. M. and Hersh, W. R., *A survey of current work in biomedical text mining*, Briefings in Bioinformatics, **6** (1), (2005), pp. 57–71.

[8] Delamothe T., *Reporting adverse drug reactions*, British Medical Journal, (1992), 304:465.

[9] Feldman R and Sanger J., *The text mining handbook*, Cambridge University Press, 32 Avenue of the Americas, NY, USA, 2007.

[10] Hall, P. A. V. and Dowling G. R., *Approximate string matching*, Computing Surveys, **12**, (1980).

[11] Hatzivassiloglou, V., Duboué, P. A. and Rzhetsky, A., *Disambiguating proteins, genes and RNA in text: a machine learning approach*, Bioinformatics, Oxford University Press, Vol. **17**, Suppl. **1**, (2001), pp. 97–106.

[12] Hyunchul, J., Jaesoo, L., Joon-Ho, L., Soo-Jun, P., Kyu-Chul, L. and Seon-Hee, P., *Finding the evidence for protein-protein interactions from PubMed abstracts*, Bioinformatics, Oxford University Press, Vol. **22**, No. **14**, (2006), pp. 220–226.

[13] ISO 1087, *Terminology and vocabulary*, Geneva: International Organization of Standardization, (1990).

[14] Knuth D. E., *The Art of Computer Programming*, Addison-Wesley, Vol. 3, (1973).

[15] Knuth D. E., *The Art of Computer Programming—Generating All Tuples and Permutations*, Addison-Wesley, Vol. 4, Fascicle 2 (2005).

[16] Konchady M., *Text mining application programming*, Charles River Media, Boston, MA, USA, 2006.

[17] Levenshtein, V. I., *Binary codes capable of co/retting deletions, insertions, and reversals*, Sov. Phys. Dokl., **10**, (1966), pp. 707–710.

[18] Lew, A. and Mausch H., *Dynamic Programming—A computational tool*, Studies in Computational Intelligence (SCI), **38**, (2007), pp. 45–100.

[19] Lindquist M., *VigiBase, the WHO Global ICSR Database System: Basic Facts*, Drug Information Journal, Vol. **42**, (2008), pp. 409–419.

[20] Lindquist M. and Edwards I. R., *The WHO Programme for International Drug Monitoring, its database, and the technical support of the Uppsala Monitoring Center*, The Journal of Rheumatology., **28**, (2001), pp. 1180–1187.

[21] Lowrance, S. and Wagner R., *An extension of the string-to-string correction problem*, J. ACM, **22**, (1975), pp. 177–183.

[22] Martin R. C. and Martin M. , *Agile Principles, Patterns, and Practices in C#*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006.

[23] Martindale (xml). Including data up to 2009-10-19, Royal Pharmaceutical Society.

[24] Navarro G., *A guided tour to approximate string matching*, ACM Computing Surveys, Vol. **33**, No. **1**, (2001).

[25] Norén G. N., Bate, A., Orre, R. and Edwards, I. R., *Extending the methods used to screen the WHO drug safety database towards analysis of complex associations and improved accuracy for rare events*, Statistics in medicine., **25**, (2006), pp. 3740–3757.

[26] 0dell, M. K. and Russell R.C., *U,S. Patent nos. 1,261,167 (**1918**) and 1,435,683 (**1922**).*

[27] Porter, M. F., *An algorithm for suffix stripping*, Program, **14**(3), (1980), pp. 130–137.

[28] Ståhl, M., Lindqvist, M, Edwards, I. R. and Brown, E. G., *Introducing triage logic as a new strategy for the detection of signals in the WHO Drug Monitoring Database*, Pharmacoepidemiology and Drug Safety, **13**, (2004), pp. 355–363.

[29] Temkin, J. M. and Gilder, M. R., *Extraction of protein interaction information from unstructured text using a context-free grammar*, Bioinformatics, Oxford University Press, Vol. **19**, No. **16**, (2003), pp. 2046–2053.

[30] Walsh, G., *Biopharmaceuticals: Biochemistry and Biotechnology*, John Wiley & Sons Ltd, Second Edition, (2003), pp. 94.

[31] Zobel, J. and Dart P., *Finding approximate matches in large lexicons*, Software-Practice and Experience, **25**(3), (1995), pp. 331–345.

Appendix A: Classical Permutation Algorithm

The text extraction algorithm implements a classical algorithm to generate all possible permutations for a given set or multiset of n elements a_1, a_2, ..., a_n. The algorithm will generate the possible permutations in lexicographical order. It has its roots in 14th century India [15]. Below is an outline of the steps involved [15]:

1. Sort the individual elements in the set in ascending order (i.e. $a_1 \leq a_2 \leq \ldots \leq a_n$) and let it be the first permutation.

2. Start from the last element in the permutation. Find the element just before the longest tail (i.e. sequence of elements at the end ordered in descending order).

3. Find the element in the tail that is the smallest larger than the element just before the tail (found in the previous step). Swap the elements found in step 2 and 3.

4. Sort the elements in the tail in ascending order. This is the next permutation in lexicographical order. Continue from step 2 using this permutation.

5. Terminate when all elements are sorted in decreasing order (this occurs when step 2 does not find an element).

Below follows two examples that show how the algorithm finds all possible permutations in lexicographical order.

Example 6 *We want to find all permutations of {2, 1, 3}. First we sort the elements in ascending order {1, 2, 3}, which is the first permutation. The longest tail consists of the element '3'. The element just before the tail is '2'. Therefore the '2' and '3' are swapped. Next permutation becomes {1, 3, 2}. The procedure is repeated. The longest tail now consists of elements '3' and '2'. The element just before the tail is now '1' and the smallest element larger than '1' in the tail is '2'. These elements are swapped and the two elements in the tail are sorted in ascending order to generate the next permutation {2, 1, 3}. Repeating the procedure will generate the permutations {2, 3, 1}, {3, 1, 2} and {3, 2, 1}. All permutations generated in lexicographical order are thus:*
{1, 2, 3}, {1, 3, 2}, {2, 1, 3}, {2, 3, 1}, {3, 1, 2} and {3, 2, 1}.

Example 7 *Let us say we have the term {***duodenal ulcer hemorrhagic***}. It consists of 3 words and we want to find all term permutations (i.e. all permutations of the words). There should be* $3! = 6$ *possible permutations. The algorithm starts by finding the lexicographically smallest permutation, which is {***duodenal**

hemorrhagic ulcer}. *The next permutation in lexicographical order is* {**duodenal ulcer hemorrhagic**}. *Next permutation will be* {**hemorrhagic duodenal ulcer**}. *The next permutations will be* {**hemorrhagic ulcer duodenal**}, {**ulcer duodenal hemorrhagic**} *and* {**ulcer hemorrhagic duodenal**} *generated in this order. After the sixth permutation the algorithm will terminate because there is no next permutation. All possible permutations have been generated.*

Appendix B: Algorithm Parameters

Table 17 shows the parameters that can be set on the text extraction algorithm. Each parameter is described and its possible values are given.

Table 17: **Text extraction algorithm parameters**

Parameter Name	**Description**	**Possible values**
LexiconEntries	A list of lexicon entries (i.e. the ADR texts to search by the algorithm)	Can be any list of lexicon entries. Each lexicon entry contains an ADR text and a drug name.
Terms	A list of medical terms to search for in the ADR texts.	Can be any list of terms.
NumberOf-Threads	The number of threads to use to run the algorithm. More threads will increase the speed of the algorihm in a multicore processor environment.	An integer in the interval [1, 10].
StopWords	A list of stop words. Stop words are words that are treated as words with no meaning and will be removed in the extraction process.	Any list of stop words.
RemoveStop-Words	A boolean value indicating if *removal of stop words* should be used in the preprocessing step.	True or false.
Stemming	A boolean value indicating if the *Porter stemming* algorithm should be used in the preprocessing step.	True or false.
Synonyms	A list of synonyms to use. Synonyms are treated as equivalent by the algorithm.	Any list of synonyms.
UseSynonyms	A boolean value indicating if *synonyms* should be used in the preprocessing step.	True or false.
Soundex	A boolean value indicating if the *Soundex* algorithm should be used in the preprocessing step.	True or false.

Table 17: **Text extraction algorithm parameters**

Parameter Name	**Description**	**Possible values**
PermutationType	The type of permutation to use.	Can be set to either ***none***, ***term permutation*** or ***word position permutation***.
DistanceMeasure	The type of distance measure to use.	Can be set to either ***none***, ***Levenshtein distance*** or ***LCS distance***.
MaxWordPercentageDeviation	The maximum allowed percentage deviation (i.e. percentage calculation based on character differences) for a single word match when using string distance measures. Applicable when DistanceMeasure is set to either ***Levenshtein distance*** or ***LCS distance***.	A percentage value in the interval [0, 100].
MaxTermPercentageDeviation	The maximum allowed percentage deviation (i.e. percentage calculation based on character differences) for a complete term match when using string distance measures. Applicable when DistanceMeasure is set to either ***Levenshtein distance*** or ***LCS distance***	A percentage value in the interval [0, 100].
RestrictPartialTextMatches	Imposes a threshold on minimum percentage of the words in the text that need to be matched by a term to result in a match.	True or false.
MinWordsPercentageHit	The minimum percentage threshold value of the words in the text that need to be matched by a term to result in a match. Applicable when RestrictPartialTextMatches is set to true.	A percentage value in the interval [0, 100].

Table 17: **Text extraction algorithm parameters**

Parameter Name	**Description**	**Possible values**
HitComparer-Method	A delegate method that is called by the algorithm when there are overlapping hits in the text. It is used to determine which of two overlapping hits in the text is *the best*.	Any method that takes two hits as parameters and returns an integer.

Appendix C: Used Stop Words and Synonyms

The text extraction algorithm requires a list of stop words and a list of synonyms when *removal of stop words* and *synonyms* are used respectively. In the text extraction performance analysis, all the runs that used *removal of stop words* used the list of stop words below. The list is a customized version of the universal stop words list given by Konchady [16] (p. 96–97). Customization consists of addition of *nec* and *nos* and removal of *no* and *not*. *Nec* and *nos* are abbreviations of *"not elsewhere specified"* and *"not otherwise specified"* respectively–two frequently occurring words within MedDRA terms. Our stop words list is:

about, add, ago, after, all, also, an, and, another, any, are, as, at, be, because, been, before, being, between, big, both, but, by, came, could, did, do, does, due, each, else, end, far, few, for, from, get, got, had, has, have, he, her, here, him, himself, his, how, if, in, into, is, it, its, just, let, lie, like, low, make, many, me, might, more, most, much, must, my, nec, never, nos, nor, now, of, off, old, on, only, or, other, our, out, over, per, pre, put, re, said, same, see, she, should, since, so, some, still, such, take, than, that, the, their, them, then, there, these, they, this, those, through, to, too, under, up, use, very, via, want, was, way, we, well, were, what, when, where, which, while, who, will, with, would, yes, yet, you, your.

All the runs in the text extraction performance analysis that used *synonyms* used the following list of synonyms:

qt → qtc
ferritin → iron
heart → cardiac
gi → gastrointestinal
convulsion → seizure
convulsions → seizures
anti platelet → antiplatelet
international normalised ratio → inr
nephrotoxicity, nephrotoxic → nephropathy toxic
hepatic → liver
estrogen → oestrogen
hepatotoxicity, hepatotoxic → hepatotoxic effect
potentiated, raised, elevated → increased
decreased → lowered
decrease → lower
haemorrhage, hemorrhage, haemorrage → bleeding
acute → immediate

Appendix D: TextMiner - Graphical User Interface

This appendix presents the GUI of the TextMiner application. Figure 9 shows a screenshot of the *Text Sources*-tab. This is where all the term sources and lexicon sources that have been loaded into the application are displayed.

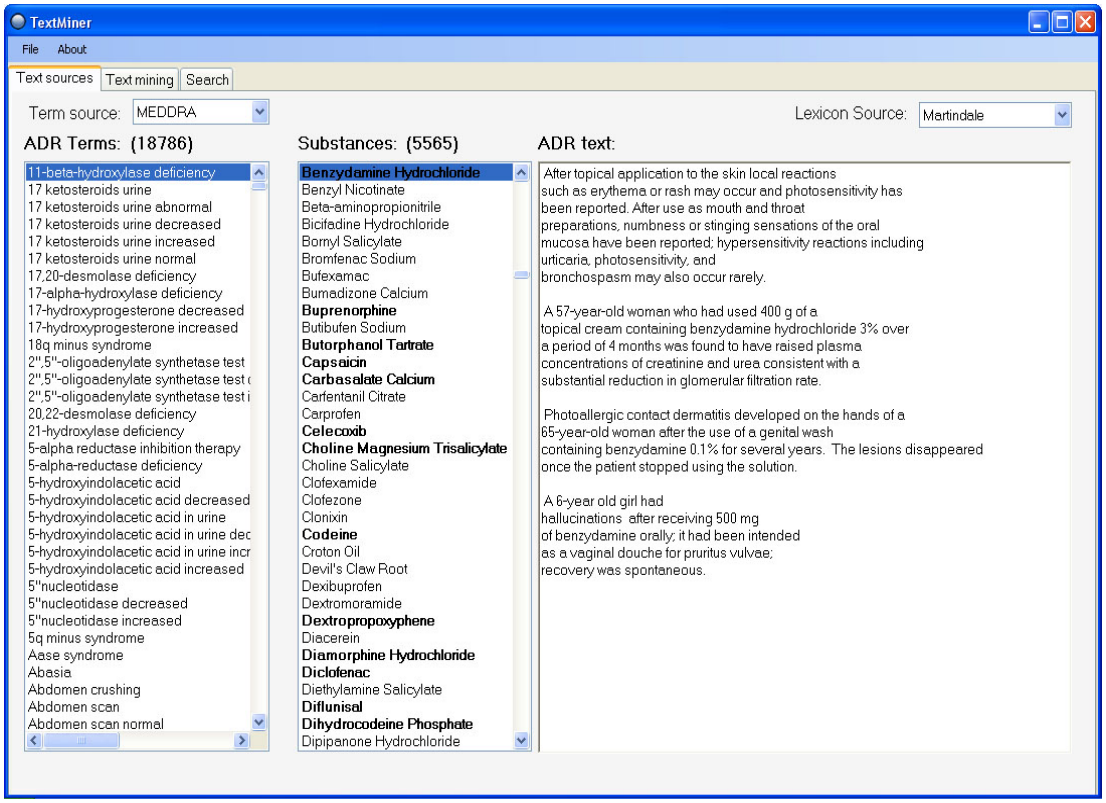

***Figure 9: Text Sources*-tab**. Screenshot of the *Text Sources*-tab. The term source drop down box on the left is populated with all term sources that have been loaded into TextMiner. The term source list below the drop down box shows all terms for the selected term source. On the right, the lexicon source drop down box contains all lexicon sources that have been loaded into the application. The selected lexicon source is displayed.

Figure 10 shows a screenshot of the *Text Mining*-tab. This is where the user can start a new text extraction run. The user can easily set the parameters for the run. All finished runs can be seen along with their respective run information.

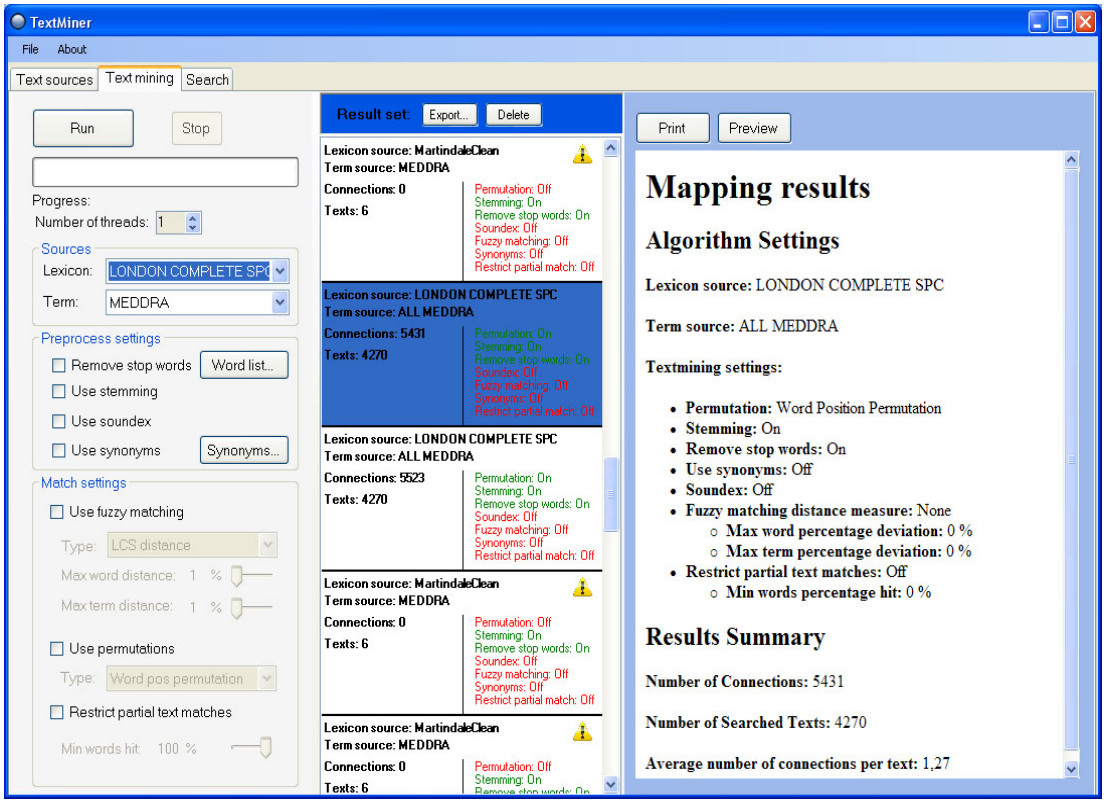

***Figure 10: Text Mining*-tab**. Screenshot of the *Text Mining*-tab. On the left, the user can set up algorithm parameters, start a new text extraction run, view progress and abort an ongoing run. All finished runs are stored in the list in the middle. Run information is displayed for the selected run.

Figure 11 shows a screenshot of the *Search*-tab. This is where the user can search results of finished and aborted runs for specific drug substances and ADR terms.

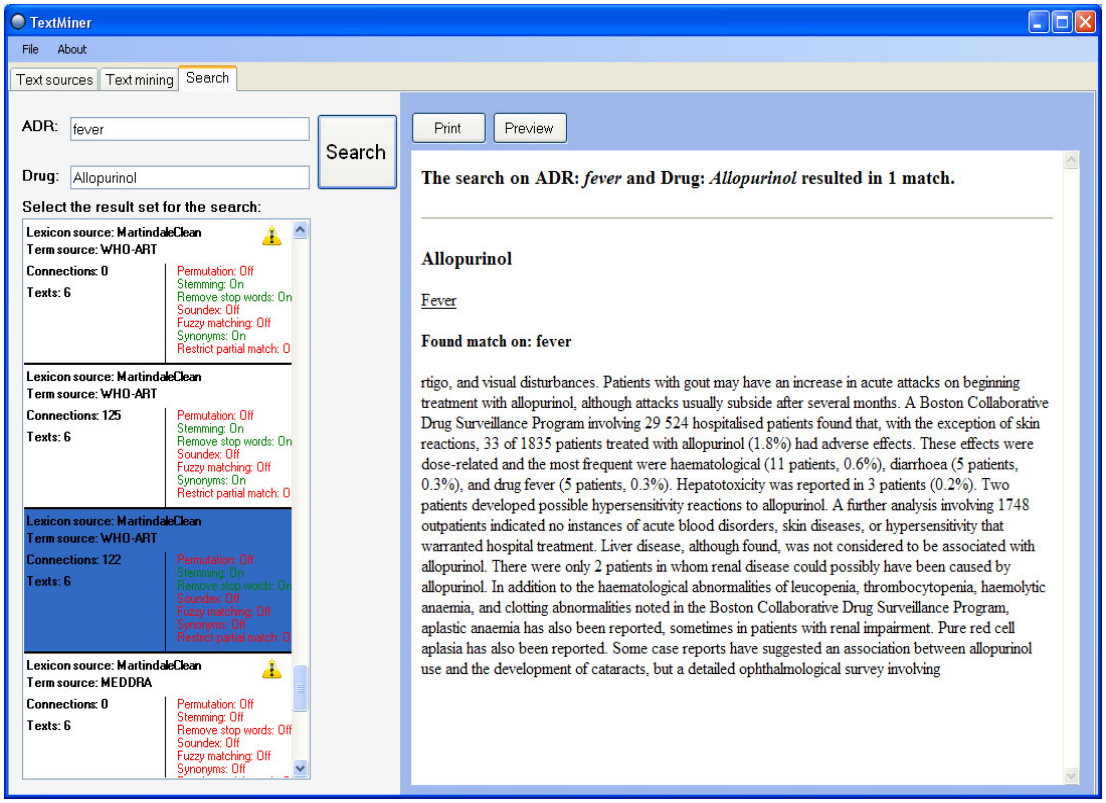

***Figure 11: Search*-tab**. Screenshot of the *Search*-tab. On the left, the user can select the text extraction run to search and enter a drug substance and ADR term to search for. The results of the search are displayed on the right side.

Appendix E: Code Example Using TextMiningTools.dll

This appendix consists of a code example to illustrate how to set up the algorithm parameters, listen to algorithm events and start a text extraction run by code.

```
1
2  // An example method that shows how to set up the text extraction
3  // algorithm parameters, set up event handlers for algorithm events
4  // and start a text extraction run.
5  public void RunExtractionAlgorithm(LexiconSource lexSource, TermSource termSource)
6  {
7          //************************************************************
8          // Get a list of ILexiconEntry objects and a list of terms from
9          // a database
10         //************************************************************
11         IEnumerable<ILexiconEntry> lexiconEntries =
12                 Database.GetAllLexiconEntries(lexSource);
13
14         IEnumerable<string> termNames = Database.GetAllTerms(termSource);
15
16         //************************************************************
17         // Set up the text extraction algorithm parameters
18         //************************************************************
19
20         // Set algorithm lexicon entries
21         // to the lexicon entries fetched from the database
22         algorithm.LexiconEntries = lexiconEntries;
23
24         // Set algorithm ADR terms
25         // to the terms fetched from the database
26         algorithm.Terms = termNames;
27
28         // Set the number of threads to use (parallelization)
29         // (1-10 threads are possible)
30         algorithm.NumberOfThreads = 2;
31
32         // Set preprocessing parameters
33         algorithm.Stemming = true;
34         algorithm.Soundex = false;
35         algorithm.RemoveStopWords = true;
36         algorithm.UseSynonyms = true;
37
38         // Set the permutation type
39         algorithm.PermutationType = PermutationType.WordPositionPermutation;
40
41         // Set the distance measure to use
42         // and cut-off values in percent for how much deviation to allow
43         // during the matching
44         algorithm.DistanceMeasure = DistanceMeasure.LCSDistance;
45         algorithm.MaxTermPercentageDeviation = 15;
46         algorithm.MaxWordPercentageDeviation = 20;
47
48         // Set the restriction of partial text matches
49         algorithm.RestrictPartialTextMatches = false;
50         algorithm.MinWordsPercentageHit = 0;
51
52         // Set the stop words for the algorithm
53         algorithm.StopWords = Database.GetStopWordList();
54
55         // Set the synonyms for the algorithm
56         algorithm.Synonyms = Database.GetSynonymList();
```

```
57
58          // Set up event handlers for algorithm events
59          algorithm.algorithmRunResult +=
60                  new AlgorithmRunResult(algorithm_algorithmRunResult);
61          algorithm.algorithmRunStarted +=
62                  new AlgorithmRunStarted(algorithm_algorithmRunStarted);
63          algorithm.algorithmException +=
64                  new AlgorithmException(algorithm_algorithmException);
65          algorithm.algorithmRunFinished +=
66                  new EventHandler(algorithm_algorithmRunFinished);
67          algorithm.algorithmProgressChanged +=
68                  new AlgorithmProgress(algorithm_algorithmProgressChanged);
69
70          // Start the text extraction run
71          algorithm.Run();
72  }
```