APPLICATION OF AUTOMATIC MUTATION-GENE PAIR EXTRACTION TO DISEASES

by
Müge Erdoğmuş

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabancı University
Spring 2007

APPLICATION OF AUTOMATIC MUTATION-GENE PAIR EXTRACTION TO DISEASES

APPROVED BY:

Assoc. Prof. Dr. Osman Uğur Sezerman ………………………
(Dissertation Supervisor)

Assist. Prof. Dr. Esra Erdem ………………………

Assoc. Prof. Dr. Hikmet Budak ……………………....

Prof. Dr. Kemal Oflazer ………………………

Assist. Prof. Dr. Yücel Saygın ………………………

DATE OF APPROVAL: ....

To my family

&

To my fiancée Fırat

ACKNOWLEDGEMENTS

I would like to express my gratitude to my thesis advisor Osman Uğur Sezerman for his invaluable guidance and encouragement throughout this thesis. I would like to thank my thesis co-advisor Kemal Oflazer for his helpful suggestions and comments. I also would like to thank Esra Erdem, Yücel Saygın and Hikmet Budak for reading and commenting on this thesis.

I am indebted to my family for their support, encouragement and love during my studies. Last but not the least; I am grateful to my dearest, Fırat, for keeping my spirit up with his love and friendship.

ABSTRACT

APPLICATION OF AUTOMATIC MUTATION-GENE PAIR EXTRACTION TO DISEASES

Müge Erdoğmuş
Master of Science, 2007
Prof. Dr. Kemal Oflazer
Assoc. Prof. Dr. Osman Uğur Sezerman

Keywords: disease, mutation, gene, information extraction

Nowadays, it is known that several inherited genetic diseases; such as sickle cell anemia, are caused by mutations in genes. In order to find ways to prevent and even better to circumvent occurrence of these diseases, knowledge of mutations and the genes on which the mutations occur is of crucial importance.

Information on disease related mutations and genes can be accessed through publicly available databases or biomedical literature sources. However, acquiring relevant information from such resources can be problematic because of two reasons. Firstly manually created databases are usually incomplete and not up to date. Secondly reading through vast amount of publicly available biomedical documents is very time consuming. Therefore, there is a need for systems that are capable of extracting relevant information from publicly available resources in an automated fashion.

This thesis presents the design and implementation of a system, MuGeX, that automatically extracts mutation-gene pairs from MEDLINE abstracts for a given disease. MuGeX performs mainly three tasks. First task is identification of mutations, applying pattern matching in conjunction with a machine learning algorithm. The second task is identification of gene names utilizing a dictionary-based method. The final task is building relations between genes and mutations based on proximity measures.

Results of experiments indicate that MuGeX identifies 85.9% of mutations that are on experiment corpus at 95.9% precision. For mutation-gene pair extraction, we focused on Alzheimer’s disease. We observed that 88.9% of mutation-gene pairs retrieved by MuGeX for Alzheimer’s disease are correct.

ÖZET

MUTASYON-GEN ÇİFTLERİNİN OTOMATIK OLARAK TANIMLANMASININ HASTALIKLARA UYGULANMASI

Müge Erdoğmuş
Yüksek Lisans, 2007
Prof. Dr. Kemal Oflazer
Doç. Dr. Osman Uğur Sezerman

Anahtar Sözcükler: hastalık, mutasyon, gen, bilgi çıkarımı

Günümüzde, akdeniz anemisi gibi birçok kalıtsal hastalığın genlerde olan mutasyonlar sonucu ortaya çıktığı bilinmektedir. Bu hastalıkların ilerlemelerinin ve hatta ortaya çıkmalarının engellenmesini sağlayacak yöntemlerin bulunması konusunda mutasyonlar ve bu mutasyonların gerçekleştiği genlerin bilgisi büyük önem taşımaktadır.

Hastalıklara ilişkin mutasyon ve gen bilgilerine herkese açık veri bankalarından ve biyomedikal literatür kaynaklarından erişmek mümkündür. Yalnız, bu kaynaklardan ilgili bilgilerin elde edilmesi iki sebepten ötürü problemli olabilir. İlk olarak bilgilerin elle girildiği veri bankaları genellikle eksik ve güncel olmayan bilgiler içermektedirler. İkinci olarak çok büyük miktarda biyomedikal dökümanı okumak oldukça zaman almaktadır. Bu yüzden ilgili bilgileri erişime açık mevcut kaynaklardan otomatik olarak çıkartacak sistemlere ihtiyaç vardır.

Bu tezde, istenilen bir hastalık için MEDLINE özetlerinden mutasyon-gen çiftlerini otomatik olarak çıkartan MuGeX isimli sistemin tasarımı ve uygulanması sunulmaktadır. MuGeX sistemi temel olarak üç işlem gerçekleştirmektedir. İlk işlem, özetlerde geçen mutasyonların örüntü eşleştirme yönteminin bir makine öğrenimi algoritması ile birlikte kullanılması yolu ile tanımlanmasıdır. İkinci işlem, gen isimlerinin sözlük kullanımına dayanan bir metod ile tanımlanmasıdır. Sonuncu işlem ise mutasyonlar ve genler arasında yakınlık göz önünde bulundurularak ilişkilerin kurulmasıdır.

Gerçekleştirilmiş olan deneylerin sonuçları gösteriyorki MuGeX deney özetlerinde mevcut olan mutasyonların %85.9'unu %95.9 doğruluk oranı ile bulmaktadır. Mutasyon-gen çiftlerinin tanımlanması işlemi için Alzheimer hastalığına odaklandık. Gözlemlediğimiz üzere MuGeX Alzheimer hastalığına ilişkin mutasyon-gen çiflerinin getirilmesinde %88.9'luk bir doğruluk oranına sahiptir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

1.1. Sources of Information and the Need for Information Extraction

Scientific literature is the most important information source for researchers in the biomedical domain. In order to keep up with information on novel breakthroughs and new trends in other domains of biology, researchers get assistance from published scientific works. Through publicly available literature databases and online journals researchers are able to access and utilize the information found in scientific literature. MEDLINE database is one of the most commonly used information sources that contains journal articles embracing all areas of biomedical domain; and through public server of MEDLINE (PubMed), researchers can access online published literature. The number of published articles and journals in these databases increases constantly as the volume of scientific literature increases. Figure 1 shows increase in number of abstracts with respect to years (statistics are acquired by querying PubMed).

In order to find information relevant to their purposes, researchers need to read through a vast amount of publicly available biomedical articles. As a consequence of the continuing increase in amount of scientific information, finding relevant information by reading through the articles is a very time-consuming process. In such a case, the best thing that a researcher can do, is to use an accurate information retrieval system. However, even in that case the number of returned articles may be in the order of hundreds. Due to this information overload, it is difficult for researchers to keep up to date knowledge even on their own areas of specialization.

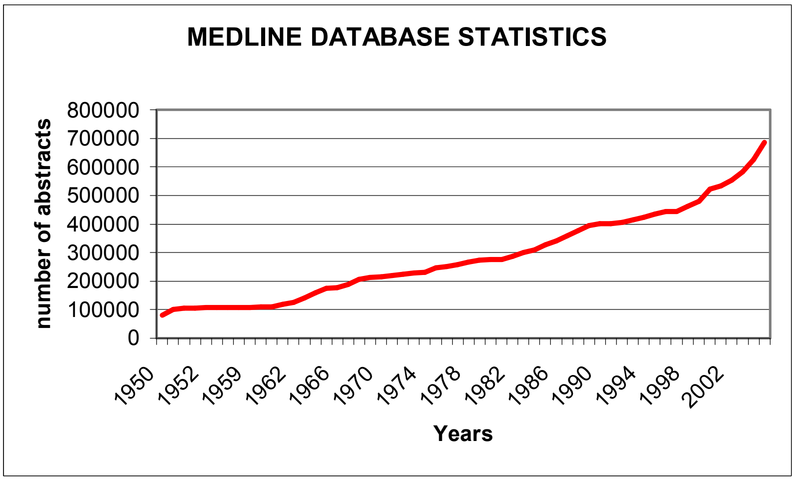


Figure 1 Increase in number of abstracts with respect to years

Other information sources for researchers are the databases that are created as results of previous works of peer researchers. These manually created comprehensive databases are commonly accessible through the Internet. However, such databases have one great drawback: they are usually incomplete and not up to date. For that reason, a researcher may not find what he or she is looking for unless the database is up to date. Furthermore, creation and maintenance of such databases require significant amount of manual labor. Werner Syndrome Mutational Database,[8] KMeyeDB,[14] and AD&FTDMDB[2] are examples of such online databases, yet we did not consider them in our study.

Because of the stated reasons application of automated methods in biomedical domain is a necessity, rather than luxury, that arises from the need to access relevant information as efficient and as timely as possible. With the advances in information extraction and knowledge management, it is possible to access relevant information from electronically available biomedical documents accurately in an automated fashion. By exploiting automatic information extraction methods the time required to access relevant information may be reduced and the dependence on manual labor may be eliminated.

1.2. Motivation

Disease specific mutation extraction is one of the sub-domains of biomedical domain that require application of automated information extraction methods. Research in this sub-domain is very important due to the fact that information acquired from these studies can help us to understand the origins of diseases. Along with mutation information, knowledge of genes on which the mutations occur is also important. Because a gene is a sequence of nucleotides that contain information about how a specific protein is to be synthesized in a cell. Moreover, proteins are large molecules and are essential components of all cells. For cells to perform properly, the proteins must carry out their functions without any error. In its most basic form, a protein can be defined as a sequence of amino acids. According to lining up of its amino acids, the protein folds and as a result of its proper folding the protein acquires its function. Everything that is essential for a protein depends on the way it folds.

Mutations that occur in the gene coding for a protein may cause the protein to function improperly by changing the amino acid sequence of that protein. In such a case, the protein cannot fold properly and the incorrectly folded protein may result in emergence of certain diseases. For instance, sickle cell anemia is known to be caused by a single mutation, change of Glutamic acid to Valine at position 6, in the amino acid sequence of hemoglobin.

Therefore, to have a better understanding of the mechanisms of disease development, the knowledge of mutations and the genes on which the mutations occur is of crucial importance. Understanding the development mechanism of a disease means being able to reason about the occurrence of that disease. Thus, scientists can find ways to prevent growth of certain diseases; and even better circumvent occurrence of certain diseases.

With this idea in mind, several experimental and computational studies have been conducted on mutations that are related to diseases – the so called disease specific mutations. Results of such studies are commonly accessible via comprehensive databases in the Internet. But, due to the reasons stated in Section 1.1, information in these databases might be inadequate and therefore utilization of such databases may give inaccurate results. So, there is a need for automatic information extraction methods. However, although there has been much prior work in the use of information extraction methods in biomedical domain, thus far no system associated extracted mutation information and their related genes to diseases. This is the reason that led us to the development of this thesis.

We have developed an automated system, called MuGeX, that is capable of extracting mutation-gene pairs from MEDLINE abstracts for a given disease name. The system is platform independent and has a web interface that allows the user to perform queries for diseases. The system utilizes natural language processing and machine learning methods with the aim of extracting and retrieving information relevant to a given disease.

1.3. Thesis Layout

The organization of the thesis as follows: Chapter 2 presents an overview of information extraction and its applications in biomedical domain. Chapter 3 presents the main natural language processing and machine learning techniques that are used in this thesis. Chapter 4 presents the followed methodology in detail. Chapter 5 discusses the experiments and results, and conclusions are given in Chapter 6.

CHAPTER 2

Information Extraction in Biological Domain

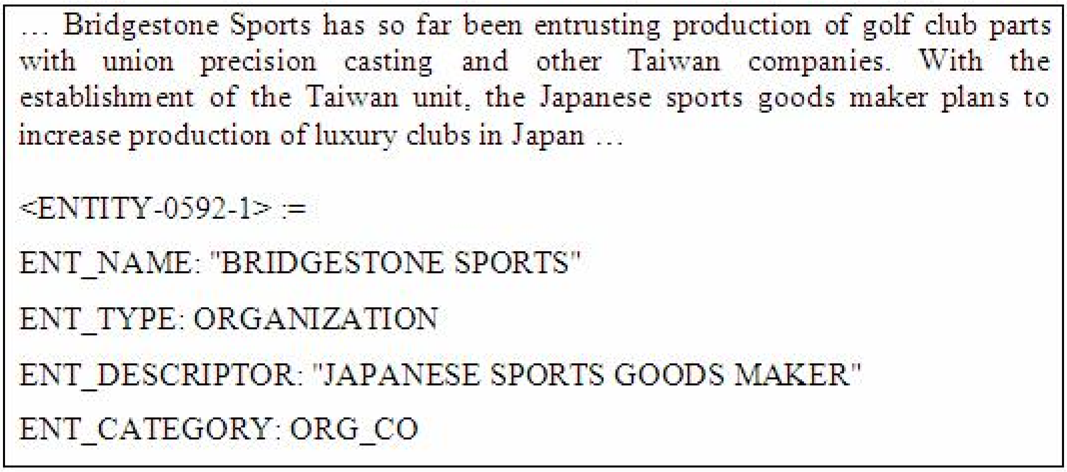
2.1. Introduction

This chapter discusses the notion of information extraction. First, a brief description of information extraction is given along with some of the commonly addressed problems. Afterwards, the applications of information extraction in biomedical domain are explained in detail.

2.2. Information Extraction

Information extraction (IE) is the task of extracting information relevant to a question in hand from unstructured data, i.e., natural language text. Owing to *Message Understanding Conferences* (MUCs), information extraction gained much prevalence. Besides, these conferences assisted in shaping the constituents of information extraction. Several tasks are dealt with in scope of information extraction. Some of the highly addressed tasks may be summarized as follows:

- *Named Entity Recognition Task* aims to extract names of entities from unstructured text; such as names of people, places, and brands. The extracted names are generally tagged for further processing.
- *Relation Extraction Task* aims to build relations between previously identified entities; such as finding out which brand belongs to which company in an article.
- *Template Filling Task* aims to extract relevant information from unstructured text and map the extracted information into a previously defined structure, which may be designated as a template. Figure 2 shows a template entity for "Bridgestone Sports Company". Information to fill the template is extracted from the text written above.[19]



… Bridgestone Sports has so far been entrusting production of golf club parts with union precision casting and other Taiwan companies. With the establishment of the Taiwan unit, the Japanese sports goods maker plans to increase production of luxury clubs in Japan …

<ENTITY-0592-1> :=

ENT_NAME: "BRIDGESTONE SPORTS"

ENT_TYPE: ORGANIZATION

ENT_DESCRIPTOR: "JAPANESE SPORTS GOODS MAKER"

ENT_CATEGORY: ORG_CO

Figure 2 An example of Template Filling Task

2.3. Background Work in Biomedical Domain

One of the most widely addressed information extraction tasks in biomedical domain is named entity recognition. The aim of named entity recognition in this domain is to extract names of genes, proteins and other cellular substances. Named entity recognition encompasses two main problems: term identification and term classification. While term identification deals with identification of words that indicate existence of domain specific information in an article, term classification as the name implies, tries to classify the identified terms into biological categories, i.e., gene, protein.

On the surface named entity recognition seems to be trivial; however due to a number of problems it is one of the most challenging tasks. The major difficulty in named entity recognition arises from absence of a standard naming convention. While many entities have more than one name, a name may refer to different entities depending upon the context. For instance, both PTEN and MMAC1 are used to refer to same gene; CD4 may refer to a cell or a protein depending upon its context.

Approaches to the problem of named entity recognition may be grouped into three categories: dictionary-based approaches, rule-based approaches, and machine learning - based approaches. Main idea in dictionary-based approaches is to match the entries in the dictionary against text. Due to previously mentioned problems of ambiguity in naming and the absence of a naming convention, exact matching yields very low recall values. In order to overcome low recall, instead of performing exact matching on original dictionary entries, Hanisch et al,[10] extended the dictionary by generating typical variants of each entry and then applied exact matching on extended dictionary in ProMiner system. Another solution to low recall problem has been proposed by Tsuruoka and Tsujii[26] where they used approximate string matching in place of exact matching.

The underlying idea of rule-based approaches is to build a set of rules that describes common naming structures for desired entity category. In construction of these rules, surface clues, such as symbols and capitalization, morpho-syntactic features such as singularity-plurality, dictionary of affixes and certain name constituents are utilized. For instance, the rule, “words that contains capital letters, digits, and non-alphabetical characters” can be used to extract candidate gene names. One of the successful systems that adopt a rule-based approach is developed by Wilbur and Tanabe, named AbGene.[25] As first step, by using a modified version of Brill POS tagger[6], which is adapted to tag also gene and protein names, AbGene tags all sentences. Then in order to eliminate falsely identified gene and protein names, a set of manually created rules are applied to each sentence.

In addition to dictionary-based and rule-based approaches, several studies are carried out that employs numerous supervised machine learning techniques. For instance, Chang et al.[1] built a system called GAPSCORE and compared performances of Naïve Bayes classifiers, Support Vector Machines and Maximum Entropy models on the named entity recognition task. The GAPSCORE system constructs a numerical feature vector for each word by analyzing the word's surface form, context and morphology. Then with the use of machine learning algorithms each word is given a score; and according to their scores, words are classified as denoting gene or not. Another system is developed by Collier et al.[7] that employs use of a Hidden Markov Model. This model, which exploits lexical and orthographic features of words, is trained with word bi-grams. For a specified word sequence, the system attempts to find the most likely sequence of entity categories.

The current trend in biomedical named entity recognition systems is to use hybrid approaches that combine machine learning algorithms with rule-based and/or dictionary-based approaches. In such systems, a given text is initially processed via several natural language processing methods, such as part-of-speech tagging. Afterward, several machine learning algorithms are run and for further refinement results of machine learning algorithms are filtered with use of pre-defined rules.

Researchers working in biology domain also make use of information extraction methods to extract relationships between biological entities. Some aim to extract the interactions between proteins and detect molecular pathways (e.g. Krauthammer et al.[15]). Current systems for relation extraction may be categorized into two: co-occurrence based systems and systems that make use of natural language processing. In co-occurrence-based systems two entities that are frequently observed together in abstracts or sentences have high probability of being actually related. However, with this approach the type of relation between the entities cannot be inferred. In natural language processing based systems the relation extraction process usually starts with basic operations like tokenization, part-of-speech tagging, and sentence splitting. Then for each sentence, a syntax tree is constructed to analyze the relationships between noun phrases of the sentence. In order to mark biological entities in sentences, different named entity recognition methods are used. Afterwards, a set of hand crafted rules are used to extract relationships between biological entities in the light of constructed syntax trees.

Recently the interest of researchers in biomedical information extraction domain have shifted to new problems as several initial problems like named entity recognition are considered to be solved. New initiatives have focused on the automated extraction of proteins and the sequence mutations related to them from biomedical literature.

Rebholz-Schuhmann et al.[22] developed an information extraction system called MEMA that scans Medline abstracts for mutations and extracts mutation-gene pairs.

MEMA system is capable of extracting both nucleotide and protein sequence mutations. It consists of three processes: identification of gene names, identification of mutations and disambiguation of gene names. For mutation identification a set of hand-crafted mutation patterns are formed and compiled into a finite state automaton (FSA). For gene name identification another set of regular expressions that are based on a gene name dictionary are compiled into another FSA. When more than one gene name is identified in a sentence or in an abstract in order to decide to which gene the identified mutation is to be linked, MEMA uses syntactical rules and a proximity based distance measure.

Horn et al.[12] developed MuteXt that extracts single point mutations from abstracts and full-text articles, whenever available. MuteXt follows an approach similar to that of MEMA to identify protein names and mutations. However, it extracts only protein sequence mutations for the GPCR and nuclear hormone receptor super-families. So as to identify mutations and protein names MuteXt utilizes regular expression matching. As it was the case with MEMA, regular expression to identify protein name makes use of a protein name dictionary and a list of synonyms.

Another system is developed by Baker et al.,[3,4] named Mutation Miner, that retrieves mutation annotations from full-text documents and associates them to “protein structure visualizations”. Mutation Miner is a four-tier system, where tier 1 is a web client that allows users to interact with the system, tier 2 is a web server that receives the user query and notices the tier 3, which performs information retrieval and analysis, and tier 4 is the layer that handles the resources. Tier 3 contains natural language processing subsystem that performs named entity recognition, sentence splitting, part-of-speech tagging, noun phrase chunking. Mutation Miner makes use of finite state transducers to extract named entities from the documents. It extracts protein citations and mutation expressions at named entity recognition stage. Following natural language processing operations, relations are built between extracted proteins and mutations. Proteins are related to mutations on the basis of the assumption that a protein cited in the same sentence with a mutation must be the protein that has been mutated. For extraction of relations between proteins and host organisms, Mutation Miner utilizes a template-based approach that investigates NP-NP patterns.

Cohen et al.[16] developed Mutation GraB that extracts and validates amino acid level point mutations from biomedical literature. Mutation GraB utilizes a rule-based method for protein name identification. For this purpose they created a dictionary of protein names and synonyms, and searched the documents for the dictionary entries via regular expressions. Similar to previous methods, mutations are identified by applying regular expressions. For mutation-protein association Mutation GraB uses a novel graph-based method. This method investigates significance of word bi-grams and then applies a shortest path distance search in order to associate a mutation with a protein.

From all of the mentioned points above, it is seen that biomedical information extraction is a very active and promising research area. While some of the problems are answered there are still lots of problems that are waiting to be solved.

2.4. Conclusions

In this chapter we answered the questions of what the notion of information extraction means, what are the most commonly addressed problems in information extraction, and what kind of works are carried in biomedical domain that benefit from information extraction methods. Next chapter discusses techniques that are used in the development of this thesis.

CHAPTER 3

Background Information

3.1. Introduction

This chapter presents the main natural language processing and machine learning techniques that are used in the context of the development of this thesis. First a brief introduction to regular expressions is given along with its application areas. Afterwards, lexical analysis is described succinctly. Following lexical analysis, two machine learning algorithms that are used in scope of this thesis are explained.

3.2. Regular Expressions

Before starting to explain what regular expressions are, it will be better to give some basic notions of automata theory; namely alphabets, strings and languages. An alphabet is defined as a finite set of symbols and is denoted with $\sum$.[11] A string is described as concatenation of a set of characters that are selected from $\sum$. Finally, a language is defined as a subset of all possible strings that can be derived from a specific finite alphabet. It should be noted that the string that consists of no symbols is denoted as *empty string* and it is denoted with symbol $\in$. Likewise, a language that contains no strings is named as *empty language*, and is denoted with symbol $\varnothing$.

To give an example, let the alphabet be defined as $\Sigma = \{0,1\}$. We can define strings over this alphabet by concatenating any combination of 0s and 1s. For instance, 010 and 11010 are possible strings over alphabet Σ. A language that can be defined over the alphabet Σ is the set of all strings that consist of alternating 0s and 1s.

As it is defined by Hopcroft, Motwani and Ullman[11] regular expressions are patterns that are “capable of defining all and only the regular languages”. Regular languages are the most restricted languages in the Chomsky hierarchy. Informally one can define a regular language as the one that can be accepted by a finite state automaton.

Before giving formal definition of what is meant by the notion of regular expression, it will be convenient to define the following three operations on languages: concatenation, union and closure. Concatenation of two languages L1 and L2 is accomplished by concatenating every string in language L1 with each string in language L2. In a similar manner we can define union of two languages as the set of strings all of which are in at least one of the two languages. Closure operation on a language L is described as concatenation of “any number of strings” that are taken from a language L.[11]

Now, we are equipped with all the information that is necessary to define regular expressions. One can define a regular expression over an alphabet Σ as follows:[11,24]

- $\varnothing$ is a regular expression that defines the empty language, $\varnothing$.
- $\in$ is a regular expression that defines the language, which consists of the empty string, $\{\in\}$.
- For any symbol *a* that belongs to the alphabet Σ, there is a regular expression that defines the language, which consists of the string *a*, {a}.
- For any two regular expressions E and F over the same alphabet Σ, E+F is a regular expression over Σ that denotes the union of the two languages defined by the two regular expressions E and F.
- For any two regular expressions E and F over the same alphabet Σ, EF is a regular expression over Σ that symbolizes the concatenation of the two languages defined by the two regular expressions E and F.

- For a regular expression E over an alphabet $\sum$, E* is a regular expression over the same alphabet that symbolizes the closure of the language that is defined by E.
- Finally, for a regular expression E over an alphabet $\sum$, (E) is a regular expression that symbolizes the same language that is defined by E.

Informally one may define regular expressions as patterns that are used to specify certain classes of strings in a text document. Regular expressions are supported by several programming languages; and the syntax used to define the regular expressions slightly changes depending on the programming language. One of the most commonly used syntax is the Perl syntax. Perl regular expressions allow us to write character patterns to represent large sets of characters as briefly as possible. Some of the most commonly used regular expression rules are given below.[27]

- The symbol *dot* (.) is used to represent any character.
- A sequence of characters given inside square braces is used to represent union of characters, i.e., [0123456789] denotes "any single digit".
- Instead of enumerating all characters, one can specify a range of characters using *x-y* format inside square braces. This rule is used to represent disjunction of all characters between and including *x* and *y*. For instance, instead of enumerating all characters inside the square braces, one can write a rule that signifies "any digit" using [0-9].
- The symbol *bar* (|) is used to represent union of characters.
- The symbol *star* (*) means zero or more occurrence of the character or the regular expressions that precedes it. For instance, [0-9]* means zero or more occurrence of a digit.
- The symbol *question mark* (?) means zero or one occurrence of the character or the regular expressions that precedes it. For instance, [0-9]? means zero or one occurrence of a digit.
- The symbol *plus* (+) means one or more occurrence of the character or the regular expressions that precedes it. For instance, [0-9]+ means one or more occurrence of a digit.

Regular expressions are mostly used to find patterns in natural language documents. This process is called matching. If a certain series of characters matches a regular expression then that series of characters belongs to the set that is defined by that specific regular expression; and the matching results in success. One may start with a rather general regular expression and then modify it so as to describe the pattern more accurately. Regular expressions are not only used for pattern extraction purposes. Another common usage of regular expressions is seen in lexical analyzers.

3.3. Lexical Analysis

Lexical analyzers are frequently used in compilers for scanning a given program code and recognizing all tokens of the input program.[11] A token is defined as a sequence of characters, such as names and literals. In identification of tokens, lexical analyzers make use of regular expressions. In the most basic sense a lexical analyzer consists of

- a set of regular expressions to identify each token
- a bracketed code section for each regular expression that describe what should the lexical analyzer do when it identifies a token corresponding to that specific regular expression.

Lex command in UNIX or *flex* command in GNU allows users to create lexical analyzers. Below you may see a simple *lex* input that scans the text and identifies tokens that consist of only digits and tokens that consist of only uppercase and lowercase letters.

```
%%
[0-9]+          {
                        printf(“A number is identified\n”);
                }
[A-Za-z]+       {
                        printf(“A word is identified\n”);
                }
```

The first line after “%%” symbol handles the tokens that consist of only digits. When lexical analyzer identifies a token that matches this regular expression, it displays a message on the screen. In a similar manner, whenever the lexical analyzer identifies a token that consists of only uppercase and lowercase letters, it informs the programmer that a word is seen in the input text.

3.4. Machine Learning Algorithms

Machine learning is a subfield of artificial intelligence. It deals with development of algorithms that automatically enhances their performance through learning and experience.[18] Machine learning has wide application areas, including data mining, bioinformatics, speech recognition, pattern recognition, robotics.

In scope of this thesis we concentrated on application of machine learning algorithms to the problem of document classification. In the most basic sense, aim of document classification task can be described as categorization of documents into previously defined categories on the basis of their contents. The two most commonly used algorithms for document classification are Rocchio algorithm and Naïve Bayes algorithm. Below we give brief descriptions of these algorithms.

3.4.1. Rocchio Algorithm with TF-IDF Weighting

Rocchio learning algorithm is based on vector space model.[23] In vector space model each document is represented with a vector of form $\vec{d} = (\omega_1, \omega_2, ..., \omega_{|N|})$, where each ω_i represents a distinct word in the document and $|N|$ is the number of unique words in the document. Weight of each ω_i is calculated using TF-IDF weighting scheme. TF-IDF stands for *term frequency-inverse document* frequency. In TF-IDF weighting a word is important to a document if it occurs frequently in that document and infrequently in other documents of the corpus. Term frequency (TF) is simply the number of times a given word ω_i appears in a given document d and is calculated with equation (3.1).

$$TF(\omega_i, \vec{d}) = \frac{n_i}{\sum_k n_k} \qquad (3.1)$$

where n_i is the number of occurrences of word ω_i. Inverse document frequency (IDF) is calculated using equation (3.2).

$$IDF(\omega_i) = \log \frac{|D|}{|\omega_i \in \vec{d}|} \tag{3.2}$$

In this formula $|D|$ is the number of documents in corpus and $|\omega_i \in \vec{d}|$ is the number of documents that contain word ω_i, where $\vec{d} \in D$ As can be understood from the formula, IDF produces low values for words that occur frequently, and high values for words that occur rarely across corpus. Therefore, by combining term frequency with inverse document frequency, we incorporate the knowledge of how discriminative a word is across whole documents in corpus into our weight calculation. TF-IDF weight of a word in a document is calculated with equation (3.3).

$$TF - IDF(\omega_i) = TF(\omega_i, \vec{d}) * IDF(\omega_i) \tag{3.3}$$

For classification purposes, document vectors of each class are combined, and consequently one prototype vector is constructed for each class. These vectors serve as the learned model, and used for classifying new documents. A new document is classified into one of the classes using Cosine similarity measure. According to cosine similarity measure, similarity between two vectors is captured by the cosine of the angle between them. Therefore, in order to assign new document to one of the classes the cosines of class vectors and the new document are calculated. The new document is assigned to the class with which its document vector has the highest cosine.

3.4.2. Naïve Bayes Algorithm

Naïve Bayes algorithm is one of the most widely used algorithms for document classification. As the name implies, Naïve Bayes algorithm is based on the Bayes theorem with strong independence assumptions. By taking a probabilistic approach, Naïve Bayes algorithm attempts to assign a new document to the most probable class, i.e., class of documents that contain amino acid level mutations, given the feature values.[18] As with Rocchio learning algorithm, the features of a document are the words of that document; and values of these features are the probabilities that are assigned based on Bayes' theorem. According to Bayes' theorem the probability of assigning a document with feature vector X to a class c is computed with equation (3.4).

$$P(c \mid X) = \frac{P(X \mid c) * P(c)}{P(X)} \tag{3.4}$$

In this formula $P(X)$ is probability of observing the feature vector X and $P(c)$ is the probability of assigning any document to class c. As mentioned before, Naïve Bayes algorithm assumes that all features of a document occur independently from each other. By taking this assumption into consideration Naïve Bayes algorithm assigns a document with feature vector $X = (x_1, x_2, ..., x_n)$ to a class c using equation (3.5).

$$P(c \mid X) = P(c) * \prod_i P(x_i \mid c) \tag{3.5}$$

where x_i is a feature of document's feature vector. For each class $P(c \mid X)$ is calculated and a new document is assigned to the class with maximum probability.

3.5. Conclusions

In this chapter we presented to give information on the main natural language processing and machine learning techniques that are used in the context of the development of this thesis. Next chapter presents how these techniques are combined together so as to form the underlying structure of MuGeX system.

CHAPTER 4

Methodology

4.1. Introduction

This chapter discusses the methodology that is followed throughout the development of this thesis. First a brief description of the system is given. Then functioning and working principles of each module is described in detail.

4.2. System Overview

The document corpus that is used throughout the thesis consists of abstracts that are published on MEDLINE before January 1st, 2005 and that contain either word “mutation” or “polymorphism”. There were 376211 abstracts that matched the search criteria. Figure 3 shows the basic components of the MuGeX system. As seen from figure, the system is composed of two main stages: pre-processing stage and analysis stage. In pre-processing stage the downloaded abstracts are first parsed. Then tokenization and sentence splitting operations are performed on resulting structures.

Following pre-processing stage, analysis of abstracts begins with extraction of mutations via regular expression matching. Abstracts that contain potentially ambiguous mutations are passed through an operation called disambiguation, where incorrectly identified mutations are filtered out via a machine learning algorithm. After mutation extraction process is completed, abstracts that contain mutations are scanned for gene names. Finally each identified mutation is associated with a gene according to proximity measures.

MuGeX is a platform independent system that has a web interface. Users are able to query the system to find mutation-gene pairs related to a disease by entering the name of the disease. As result of a query the information relevant to that specific disease is retrieved from the system and displayed on the web.

In scope of this thesis, we focused on extraction of mutation-gene pairs for Alzheimer's disease since there is great amount of information and easily accessible databases. However the system is designed in such a way so as to respond to any query for any disease name without any modification. Still, in case of demand, modular nature of the system makes it easier for developers to modify or extend the system's capabilities.

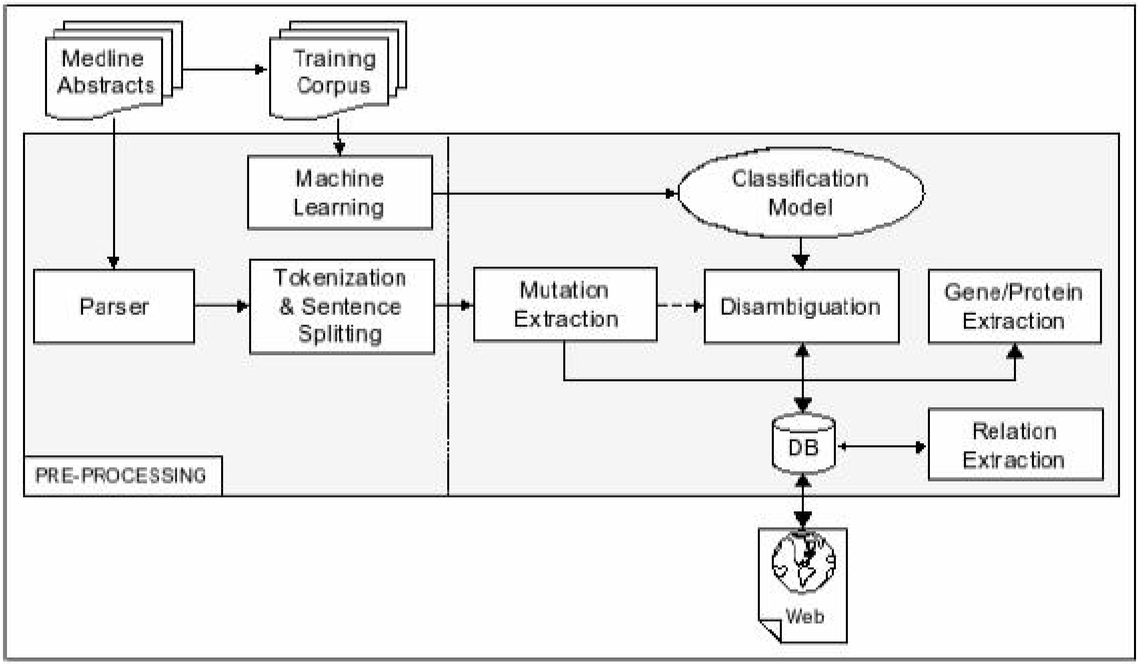


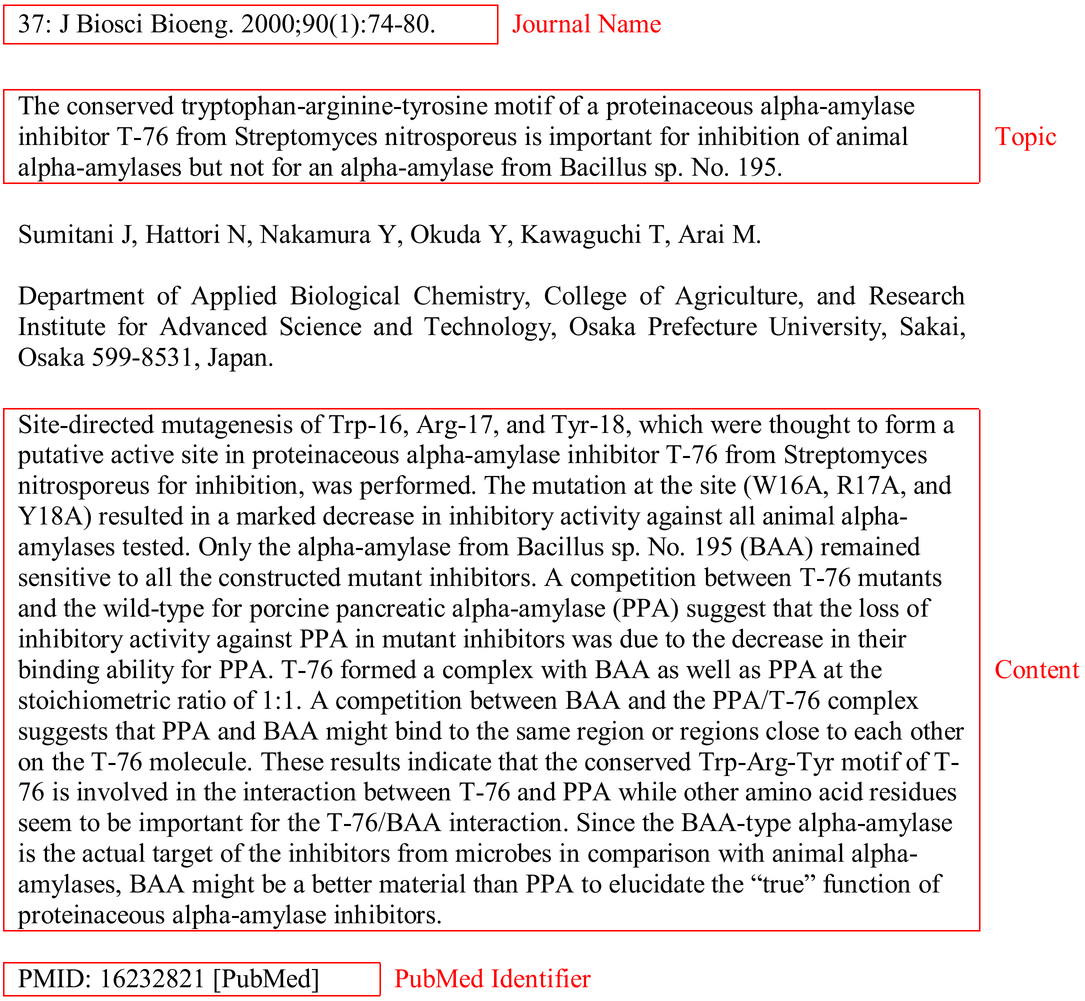
Figure 3 A schematic illustration of MuGeX system

4.3. Pre-Processing Stage

4.3.1. Parser Module

Abstracts downloaded from public server of MEDLINE consist of not only information relevant to our task, but also some unnecessary information; such as address of authors, names of authors. A parser module extracts the information that is important for our purpose – name of the journal, title of the abstract, content of the abstract and its unique PubMed identifier – from the downloaded abstracts. It utilizes a lexical analyzer in connection with a parser generator. The lexical analyzer was developed using Flex[9] and parser generator was developed using Bison[5]. While the lexical analyzer breaks a given text into tokens, parser generator translates a grammar specification into an executable program which then can parse that specific grammar. When lexical analyzers and parser generators are used in connection each token that is identified by the lexical analyzer is passed to parser generator, which reconstructs the text on the basis of its grammar rules using these tokens from lexical analyzer.

As with many natural language texts, the file that contains downloaded abstracts has a grammar. In the most basic form its "language" can be defined as a series of abstracts that are separated by two empty lines. Going one step further we can define the "abstract" as consisting of a series of sections an abstract has. For instance, journal name is followed by topic of the abstract, which in turn is followed by content of the abstract, which is consecutively followed by a unique PubMed identifier. It should be noted that each section of an abstract is separated with a single empty line (see Figure 4). Again going one step further, we can construct a grammar rule for a section as a series of tokens separated by white spaces.



37: J Biosci Bioeng. 2000;90(1):74-80.

The conserved tryptophan-arginine-tyrosine motif of a proteinaceous alpha-amylase inhibitor T-76 from Streptomyces nitrosporeus is important for inhibition of animal alpha-amylases but not for an alpha-amylase from Bacillus sp. No. 195.

Sumitani J, Hattori N, Nakamura Y, Okuda Y, Kawaguchi T, Arai M.

Department of Applied Biological Chemistry, College of Agriculture, and Research Institute for Advanced Science and Technology, Osaka Prefecture University, Sakai, Osaka 599-8531, Japan.

Site-directed mutagenesis of Trp-16, Arg-17, and Tyr-18, which were thought to form a putative active site in proteinaceous alpha-amylase inhibitor T-76 from Streptomyces nitrosporeus for inhibition, was performed. The mutation at the site (W16A, R17A, and Y18A) resulted in a marked decrease in inhibitory activity against all animal alpha-amylases tested. Only the alpha-amylase from Bacillus sp. No. 195 (BAA) remained sensitive to all the constructed mutant inhibitors. A competition between T-76 mutants and the wild-type for porcine pancreatic alpha-amylase (PPA) suggest that the loss of inhibitory activity against PPA in mutant inhibitors was due to the decrease in their binding ability for PPA. T-76 formed a complex with BAA as well as PPA at the stoichiometric ratio of 1:1. A competition between BAA and the PPA/T-76 complex suggests that PPA and BAA might bind to the same region or regions close to each other on the T-76 molecule. These results indicate that the conserved Trp-Arg-Tyr motif of T-76 is involved in the interaction between T-76 and PPA while other amino acid residues seem to be important for the T-76/BAA interaction. Since the BAA-type alpha-amylase is the actual target of the inhibitors from microbes in comparison with animal alpha-amylases, BAA might be a better material than PPA to elucidate the "true" function of proteinaceous alpha-amylase inhibitors.

PMID: 16232821 [PubMed]

Figure 4 Sections of a MEDLINE abstract

As mentioned before, the lexical analyzer breaks a given abstract into tokens and sends each identified token to parser generator. It also identifies the sections of the abstract by searching for specific signs. To give an example of what is implied by "sign" take the name of the journal in which the abstract is published. The journal name is always specified with a number followed by a colon, which is in turn followed by the name of the journal (see Figure 4). So, whenever lexical analyzer identifies this sign it sends a signal to parser generator specifying the beginning of a journal section and continues to send the tokens it identifies. Since each section ends with two carriage returns, whenever lexical analyzer sees two carriage returns it sends another signal to parser generator specifying end of a section, in this case end of journal section. Consequently, parser generator recreates the journal section of the given abstract.

Thus, in order to reconstruct an abstract with desired sections, a set of regular expressions is defined and given as an input to the lexical analyzer to identify beginning and ending of each section. Each section is parsed in a manner similar to the parsing of the journal section, which is described above, and brought together by the parser generator.

As result of this whole parsing process relevant information from each abstract is parsed and combined together in an XML-like format (see Figure 5). This formatting makes it easier to process the downloaded abstracts in the following stages.

```
<entry>
        <id>37</id>
        <journal>37: J Biosci Bioeng. 2000;90(1):74-80.</journal>
        <topic>The conserved tryptophan-arginine-tyrosine motif of a proteinaceous alpha-
amylase inhibitor T-76 from Streptomyces nitrosporeus is important for inhibition of
animal alpha-amylases but not for an alpha-amylase from Bacillus sp. No. 195.</topic>
        <content>Site-directed mutagenesis of Trp-16, Arg-17, and Tyr-18, which were
thought to form a putative active site in proteinaceous alpha-amylase inhibitor T-76 from
Streptomyces nitrosporeus for inhibition, was performed. The mutation at the site (W16A,
R17A, and Y18A) resulted in a marked decrease in inhibitory activity against all animal
alpha-amylases tested. Only the alpha-amylase from Bacillus sp. No. 195 (BAA) remained
sensitive to all the constructed mutant inhibitors. A competition between T-76 mutants and
the wild-type for porcine pancreatic alpha-amylase (PPA) suggest that the loss of inhibitory
activity against PPA in mutant inhibitors was due to the decrease in their binding ability for
PPA. T-76 formed a complex with BAA as well as PPA at the stoichiometric ratio of 1:1. A
competition between BAA and the PPA/T-76 complex suggests that PPA and BAA might
bind to the same region or regions close to each other on the T-76 molecule. These results
indicate that the conserved Trp-Arg-Tyr motif of T-76 is involved in the interaction
between T-76 and PPA while other amino acid residues seem to be important for the T-
76/BAA interaction. Since the BAA-type alpha-amylase is the actual target of the inhibitors
from microbes in comparison with animal alpha-amylases, BAA might be a better material
than PPA to elucidate the “true” function of proteinaceous alpha-amylase
inhibitors.</content>
        <pmid>PMID: 16232821 [PubMed]</pmid>
</entry>
```

Figure 5 Parsed MEDLINE abstract in XML-like format

4.3.2. Tokenization and Sentence Splitting Module

The second step in pre-processing is composed of tokenization and sentence splitting operations. At this step, non-alphanumeric characters, except hyphens, are removed from content and topic of each abstract. Hyphens are not removed since they are occasionally used to denote mutations, i.e., V113-->A. Also each sentence is labeled with <sentence> and </sentence> tags.

As it is the case with parser module, this module utilizes a lexical analyzer in connection with a parser generator in order to split the content and topic of each abstract into sentences. Again the lexical analyzer is developed using Flex[9] and parser generator is developed using Bison[5]. The grammar used by the parser generator for this module is an extended version of the grammar for parser module. A section is assumed to consist of a series of sentences and a sentence is described as a series of tokens separated by white spaces and ending with a period.

Yet again the lexical analyzer breaks a given abstract into tokens and sends each identified token to the parser generator. The so called "signs" that designate the beginning and ending of sections is redefined for the lexical analyzer.

Once more take the journal name as example. As seen from Figure 5, journal section is bounded by <journal> and </journal> tags. Thus, each time lexical analyzer identifies the token <journal>, it sends a signal to parser generator specifying the beginning of a journal section and continues to send the tokens it identifies. In order not to give rise to wrong sentence boundaries, which is generally caused by use of abbreviations, a list of commonly used abbreviations is used (see Appendix A for details of the list). Each time a period, which is not preceded by an abbreviation, is encountered an end of sentence signal is sent to the parser generator, so that each sentence is enclosed with <sentence> and </sentence> tags. Since journal section ends with token </journal>, when lexical analyzer encounters this token it sends a signal specifying the end of journal section.

As mentioned before, tokenization encompasses removal of non-alphanumeric characters excluding hyphens. Tokenization is performed via regular expression matching. For this purpose a set of regular expressions are created and each identified section is matched against this set of rules. Whenever a match occurs, the matched portion of the text is simply deleted. A regular expression library is utilized to implement this operation.

Figure 6 illustrates format of the abstract entry at the end of this step. As seen in figure, original versions of topic and content are also preserved for they might be necessary in the future. Another thing to mention is that a new tag, named "ambiguous", is introduced. This tag implies that the abstract contains an "ambiguous" mutation. What is meant by ambiguous mutation will be made clear in next section. For now, it is sufficient to note that a special rule is defined to identify certain mutation-like tokens in lexical analysis phase. If there is at least one such token then value of ambiguous tag is set as 1; otherwise it is set as 0.

```
<entry>
        <id>37</id>
        <ambiguous>1</ambiguous>
        <journal>37: J Biosci Bioeng. 2000;90(1):74-80.</journal>
        <topic>
               <sentence>The conserved tryptophan-arginine-tyrosine motif of a proteinaceous
        alpha-amylase inhibitor T-76 from Streptomyces nitrosporeus is important for inhibition of
        animal alpha-amylases but not for an alpha-amylase from Bacillus sp. No. 195.</sentence>
        </topic>
        <tok_topic>
               <sentence>The conserved tryptophan-arginine-tyrosine motif of a proteinaceous
        alpha-amylase inhibitor T-76 from Streptomyces nitrosporeus is important for inhibition of
        animal alpha-amylases but not for an alpha-amylase from Bacillus sp. No. 195</sentence>
        </tok_topic>
        <content>
               <sentence>Site-directed mutagenesis of Trp-16, Arg-17, and Tyr-18, which were
        thought to form a putative active site in proteinaceous alpha-amylase inhibitor T-76 from
        Streptomyces nitrosporeus for inhibition, was performed.</sentence>
               <sentence> The mutation at the site (W16A, R17A, and Y18A) resulted in a marked
        decrease in inhibitory activity against all animal alpha-amylases tested.</sentence>
               .............
               <sentence> Since the BAA-type alpha-amylase is the actual target of the inhibitors
        from microbes in comparison with animal alpha-amylases, BAA might be a better material
        than PPA to elucidate the “true” function of proteinaceous alpha-amylase
        inhibitors.</sentence>
        </content>
        <tok_content>
               <sentence>Site-directed mutagenesis of Trp-16 Arg-17 and Tyr-18 which were
        thought to form a putative active site in proteinaceous alpha-amylase inhibitor T-76 from
        Streptomyces nitrosporeus for inhibition was performed</sentence>
               <sentence>The mutation at the site  W16A R17A and Y18A resulted in a marked
        decrease in inhibitory activity against all animal alpha-amylases tested</sentence>
               ............
               <sentence> Since the BAA-type alpha-amylase is the actual target of the inhibitors
        from microbes in comparison with animal alpha-amylases BAA might be a better material
        than PPA to elucidate the true function of proteinaceous alpha-amylase
        inhibitors</sentence>
        </tok_content>
        <pmid>PMID: 16232821 [PubMed]</pmid>
</entry>
```

Figure 6 Illustration of abstract entry at the end of pre-processing stage

4.4. Analysis Stage

4.4.1. Mutation Extraction Module

The mutations that are extracted by MuGeX are amino acid level protein mutations, meaning that the mutations cited using amino acid terminology. Mutation extraction is performed by regular expression matching. For this purpose a set of 20 patterns is formed from phrases that describe protein mutations. A subset of example mutation citations that are extracted by MuGeX is shown in Table 1.

W16A
Ile-15-Thr
Arg506 to Gln
Ala399→Asp
Substitution of Methionine for valine at position 30
glycine 264 is replaced with a serine
Ala 231 to Val substitution

Table 1 A subset of protein mutation examples

In the most basic sense, a mutation is described with a wild-type[1] amino acid followed by position of mutation on the amino acid sequence of the protein, which is in turn followed by a mutant amino acid. Amino acids can be designated using single-letter codes (i.e., R), three-letter codes (i.e., Arg), or full names (i.e., arginine). While conventionally three-letter codes start with an uppercase letter, there is no such convention for full names. Therefore full names are matched in all lowercase letters; and lowercase letters with a leading uppercase letter (i.e., [A|a]rginine).

All of our regular expressions are based on one basic expression, which is used to detect mutations similar to W16A. That basic expression is [Amino][0-9]+[Amino],[2] where Amino is a list of single-letter amino acid codes. Remaining mutation patterns are found by making modifications to this expression. All regular expressions are compiled with the help of a C++ regular expression library.

[1] Typical form of an organism, strain, gene, or characteristic as it occurs in nature.

[2] [Amino]: (A|V|L|I|G|F|W|M|P|S|T|Y|N|Q|C|D|E|K|R|H).

As pre-processing operations are completed for an abstract, pattern matching is applied sentence by sentence to tokenized versions of topic and content of that abstract in order to identify mutations. Whenever a mutation is identified, it is enclosed with <mutation> and </mutation> tags for further processing.

At the end of this step, all information about each abstract is inserted into a database table, named Tbl:Medline (see Appendix C for detailed information on major database tables). While each entry of the database table represents an abstract, each field of the database table represents each tag, excluding sentence tag, seen in Figure 6. Further processing on an abstract will be made by querying the database for the abstract in question; and any modification on the abstract will be reflected to its corresponding database entry.

The main challenge in the mutation extraction process is to distinguish actual protein mutations from mutation like terms. For instance, due to ambiguity in naming, one may misinterpret a nucleotide mutation as a protein mutation. A14C is one example of such mutations. A14C can refer either to mutation of alanine to cysteine at position 14 or mutation of adenine to cytosine at position 14. This naming ambiguity exists also between mutations and other biological entities. The name of a strain or a cell line may easily be misinterpreted as a protein mutation, i.e., T47D is name of a cell line. As can be understood from the given examples, this ambiguity emanates when mutations are expressed in [Amino][0-9]+[Amino][2] format using single-letter amino acid codes. From now on, the term "potentially ambiguous mutations" will be used to refer to the tokens in this format. Potentially ambiguous mutations are identified during lexical analysis via regular expression matching, and the abstracts that contain these mutations are tagged for further processing. In order to resolve disambiguities MuGeX utilizes a machine learning technique. The next section discusses this disambiguation process.

4.4.2. Disambiguation Module

As mentioned in previous section, nucleotide mutations and names of some biological entities cited using single-letter codes are prone to confusion with protein mutations. The disambiguation module aims to classify the abstracts that are previously labeled as containing potentially ambiguous mutation. The classification is performed by applying a supervised machine learning algorithm on topic and content of the labeled abstracts. We focused on two learning algorithms: The Naïve Bayes algorithm and the Rocchio algorithm with Term Frequency-Inverse Document Frequency (TF-IDF) weighting. In order to decide which algorithm to use and which processing options to apply, a set of experiments are carried out.

In order to test the performance of the of Rocchio algorithm and Naïve Bayes algorithm we designed a training benchmark that consists of randomly selected 3600 Medline abstracts. This value corresponds to 10% of abstracts that contain potentially ambiguous mutations in the whole corpus. 2771 of those abstracts contain amino acid level mutations, 768 contain DNA level mutations, and the remaining abstracts contain biological entities that are cited with mutation-like terms. An abstract is labeled as "protein", if it contains at least one amino acid level mutation, and it is labeled as "not protein" if it does not contain any amino acid level mutations. For classification, we have used Rainbow[17], which is one of the front-ends of Bow library[17] designed for document classification. Rainbow not only provides functions to process documents; but also provides several classification algorithms including Naïve Bayes and Rocchio with TF-IDF weighting.

Since both learning algorithms adopt a bag-of-words approach, first, feature vectors for each abstract are constructed. As feature values, the frequency of words are used instead of just a binary presence or absence of words. In order to remove common words, a stop-word[3] list is constructed (see Appendix B for details of the list). The list is a combination of most common English words and a list of stop-words for Medline Database.

In order to decide which algorithm to use the classification performance of the two algorithms with respect to different processing options were investigated. The effects of tokenization, stemming and use of word n-grams on performance of classifiers are analyzed. For this purpose eight different classifier models are constructed. The results for this dataset are averaged over a number of random train/test splits; and in each experiment 30% of data is used for testing. Results are given in Table 2.

Before going into detail of test results, it should be noted that stemming is performed utilizing Porter stemmer.[20] Porter stemmer removes commonly observed

[3] Common words that have no meaning by themselves.

morphological and inflectional suffixes from the English words. Another thing to be mentioned is that two types of tokenization procedure are used. The first one, which is named by McCallum[17] as white space tokenizer, delimits the tokens in an input text by only white spaces ignoring tokens that contain characters other than uppercase and lowercase letters. The second tokenizer is named alphanumeric tokenizer; and it delimits tokens in an input text by non-alphanumeric characters. It should be mentioned that the tokens that consist of only digits are not included in the dictionary. Apart from these, in dictionary construction we used word n-grams. We tested the effect of using uni-grams and bi-grams.

The disambiguation performance is measured in terms of precision, recall and accuracy. While accuracy measures how close the result of an experiment to the actual value, precision measures how close the results of an experiment to each other. Equations (4.1), (4.2), (4.3) gives the formulas for precision, recall and accuracy respectively.

$$precision = \frac{TP}{TP+FP} \tag{4.1}$$

$$recall = \frac{TP}{TP+FN} \tag{4.2}$$

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{4.3}$$

In these equations TP denotes true positive, FP denotes false positive, TN denotes true negative and FN denotes false negative. True positive is used to indicate that an abstract that contains amino acid level mutation is classified as containing amino acid level mutation. In a similar manner, true negative is used to indicate that an abstract that does not contain any amino acid level mutation is classified as not containing amino acid level mutation. False positive is used to indicate that an abstract that does not contain any amino acid level mutation is classified as containing amino acid level mutation. In the same way, false negative is used to signify that an abstract that contains amino acid level mutation is classified as not containing any amino acid level mutation.

While precision values for both algorithms are almost same in all experiments, due to low recall values Rocchio learning algorithm performed worse than Naive Bayes learning algorithm. However both algorithms respond to the changing processing options in a more or less similar manner.

	Rocchio			Naïve Bayes		
Model	%Precision	%Recall	%Accuracy	%Precision	%Recall	%Accuracy
1	89.8	77.0	76.0	90.2	86.6	82.7
2	90.2	82.4	79.9	90.6	87.7	83.8
3	89.8	75.9	75.3	90.3	85.2	81.9
4	89.3	77.3	80.7	82.4	97.2	82.2
5	90.5	76.1	75.9	90.8	87.1	83.6
6	89.8	88.7	83.8	84.7	97.0	84.6
7	89.4	83.0	79.6	82.9	96.7	82.5
8	90.1	85.8	82.1	85.1	96.6	84.7

Note: 1: white space tokenizer, no stemming, unigram. 2: alphanumeric tokenizer, no stemming, unigram. 3: white space tokenizer, stemming, unigram, 4: white space tokenizer, no stemming, bi-gram. 5: alphanumeric tokenizer, stemming, unigram. 6: alphanumeric tokenizer, no stemming, bi-gram. 7: white space tokenizer, stemming, bi-gram. 8: alphanumeric tokenizer, stemming, bi-gram.

Note: Model 1 constitutes the basis case.

Table 2 Comparison of Rocchio and Naive Bayes algorithms

When abstracts are processed using only Porter stemming algorithm (Model 3), accuracy of both classifiers show no significant difference with respect to Model 1. It is observed that while frequent words tend to have more inflected forms, infrequent words, such as gene names, tend to have less inflected forms in the vocabulary. Therefore, when words are stemmed, frequently observed words become more frequent while frequency of infrequent words increase relatively less. As a result, weights of words stay nearly same after stemming; and hence performances of classifiers show no significant difference.

When tokenization is performed considering not only words that consist of letters but also words that consist of alphanumeric characters (Model 2), accuracy values of classifiers improve. This is to be expected because use of words that consist of alphanumeric characters is highly common in medical literature. Incorporating that information into classification process improves performance in terms of precision, recall and accuracy. When alphanumeric tokenization is applied along with stemming (Model 5), because of the reasons stated above, the accuracy values do not change significantly for Naïve Bayes algorithm. However, decrease in recall for Rocchio algorithm is rather unexpected. Even though performance of Rocchio on this model (Model 5) is increased with respect to its performance on the model which processes data using only stemming (Model 3), it would be reasonable to apply Rocchio on a dataset which is processed only with alphanumerical tokenization.

Looking at word bi-grams instead of unigrams (Model 4) caused small amount of decrease in precision and increase in recall values of Rocchio algorithm, which led to a considerable increase in accuracy. On the other hand, while recall of the Naïve Bayes algorithm increases significantly, due to decrease in precision, the accuracy of Naïve Bayes decreases slightly. When bi-grams are used, due to not having enough instances of word pairs in the corpus classification accuracy drops.

We believe that if size of the training set increases, the precision of Naïve Bayes algorithm will increase. On the other hand, since frequency of word pairs in the corpus is small, weights of such pairs is high according to TF-IDF weighting. By taking these high weighted word pairs into consideration, the accuracy of Rocchio algorithm increases. Furthermore, when bi-grams are used in accordance with stemming (Model 7) the frequency of word pairs increases since all inflected forms of the words are mapped to single representation. Due to increase in frequency of word pairs, the accuracy of Naïve Bayes algorithm slightly increases while accuracy of Rocchio algorithm slightly decreases. However, still the changes in accuracy values are not significant.

When bi-grams are used in accordance with alphanumeric tokenization (Model 6) precision, accuracy and recall values of both classifiers increases. Because now the information that comes from words that contains alphanumeric characters is incorporated into the classifier models.

Highest classifier accuracy is obtained using Naïve Bayes learning algorithm and processing the data by performing stemming, alphanumeric tokenization and considering word bi-grams (Model 8). For disambiguation purposes, we want classification results to be as accurate as possible while conserving high precision value.

In order to improve classification accuracy feature reduction is performed on previously constructed models. For this purpose, first N unique words that have the highest information gain are selected and remaining words are removed from vocabulary, where N ranges between 1000 and 150000[4]. Performance graphic of classifiers with respect to changing values of N are seen in Figure 7 and Figure 8.

[4] Size of vocabulary increases to 150000 when word bi-grams are used.

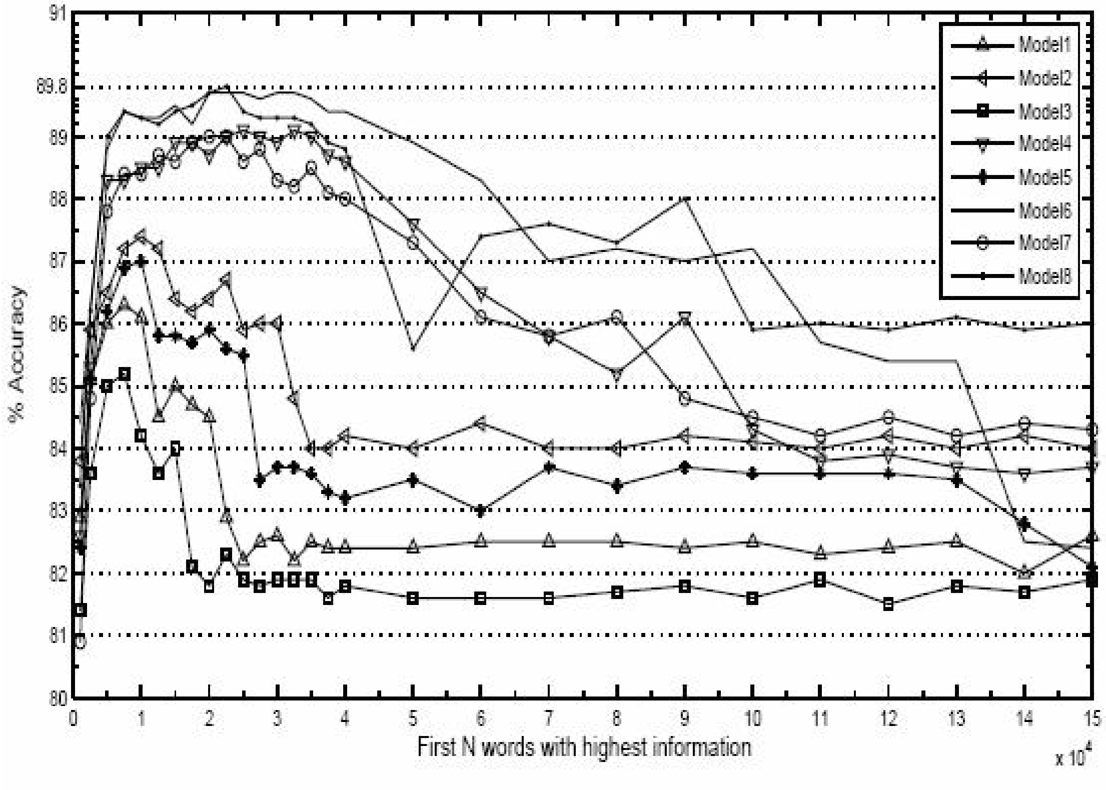


Figure 7 Performance of Naïve Bayes Algorithm

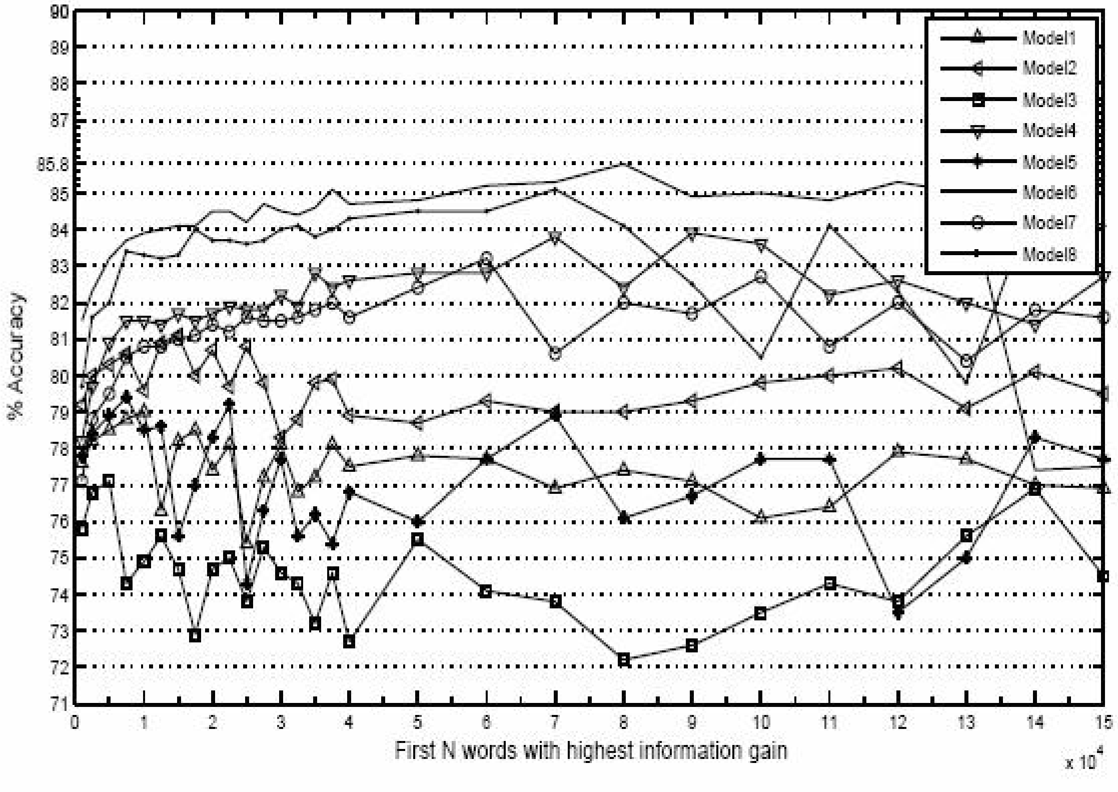


Figure 8 Performance of Rocchio Algorithm

The performance of Naïve Bayes algorithm for all models changes substantially with respect to the size of vocabulary. As expected accuracy first increases up to some point and then decreases; finally reaching to a value closer to the accuracy value when vocabulary pruning is not performed. As seen in Figure 7, best performance is obtained using Model 8 on a vocabulary consisting of 22500 words with highest information gain. Unlike Naïve Bayes algorithm, accuracy values for Rocchio do not improve very much, and tend to oscillate inside a value range. The highest accuracy value for Rocchio is obtained using Model 6. Compared to accuracy values of previous experiments, application of feature reduction caused a considerable improvement in performance of both classifiers; but still Naïve Bayes algorithm outperforms Rocchio algorithm. Therefore, the Naïve Bayes algorithm with feature reduction performed on Model 8 is used.

After deciding on which algorithm and processing options to use, the selected model is trained and incorporated into MuGeX system. For disambiguation purposes, the content and topic of each abstract that contain potentially ambiguous mutation are retrieved from the medline database table via querying the database table against value of the field named “ambiguous”. Afterwards, topic and content of each retrieved abstract is classified using Naïve Bayes algorithm that is trained with the selected model. If the abstract is classified as not containing protein mutation then <mutation> tags around all potentially ambiguous mutations are removed; and the tokenized topic and tokenized content fields of the database table entry corresponding to that abstract is updated. Otherwise tokenized versions of topic and content of the abstract is remained unchanged.

4.4.3. Gene/Protein Extraction Module

For gene name identification, a dictionary is constructed from HUGO database.[13] The dictionary consists of approved symbols, approved names, previous symbols and possible aliases of genes. Currently there are 82.408 entries in the dictionary.

When dictionary-based approaches are employed the problem of the absence of a standard naming convention arises. The naming variations are used inconsistently most of the time. Therefore, prior to gene name extraction, the gene dictionary has to be processed in order to normalize gene names. The dictionary is processed in the following manner. For gene symbols all hyphens are replaced with white spaces, parenthesized materials are removed; and for gene names parenthesized materials are removed, all punctuation characters are replaced with white spaces and all uppercase characters are converted into lowercase characters. Table 3 shows some examples of gene name normalization.

HUGO Entry	Normalized Form
alpha-1-B glycoprotein	alpha 1 b glycoprotein
alpha 1,3-galactosyltransferase 2 (isoglobotriaosylceramide synthase)	alpha 1 3 galactosyltransferase 2
CD2 (cytoplasmic tail) binding protein 2	cd2 binding protein 2
AGT-1	AGT 1

Table 3 Examples of Gene Name Normalization

A database table, named gene_synonyms, is constructed from normalized form of gene symbols and names. This table contains two fields: approved gene symbol and gene synonym. For each approved gene symbol, there are as much entries in the table as sum of the number of approved names, previous symbols and possible aliases for that gene. This table is used to map identified gene names into its approved symbol; so that all identified genes will be unified at the final stage.

This module employs exact matching method to identify gene names. Gene identification is performed with the help of a lexical analyzer that is capable of identifying the entries in the dictionary. The lexical analyzer is developed using Flex.

Gene extraction module operates in the following way. As first step, tokenized versions of topic and content of each abstract that contain at least one mutation is retrieved from medline database table. Afterwards all hyphens and parenthesized materials are removed from both topic and content of each abstract; and they are scanned to find gene symbols by the lexical analyzer. Subsequently, both content and topic are converted into lowercase letters and are scanned to find gene names. Each time a gene mention is identified by the lexical analyzer, it is encapsulated with <gene>, </gene> tags for further processing. After all genes of an abstract are identified, these genes are unified via a simple database query on gene_synonyms database table. At the final step, if at least one gene is found in an abstract, then the tokenized topic and/or tokenized content fields of the database table entry corresponding to that abstract is updated.

4.4.4. Relation Extraction Module

The final step of the analysis stage is relation extraction. In order to associate a mutation with a gene, an approach similar to that of Rebholz-Schuhmann et al.[22] is employed. The relation extraction module first checks whether a given abstract mentions a single gene or not. If it does, then any mutation in the abstract is associated directly with that gene. Otherwise, the sentence that contains the mutation is scanned for existence of a gene. If sentence does not contain any gene mention then first sentence of the abstract is analyzed, since authors usually tend to mention name of the gene on which they conduct an experiment in the first sentence. However, if there is a gene in the same sentence with a mutation, then the mutation is associated with that gene. On the other hand, if there exist more than one gene in the same sentence, then the mutation is associated to one of the genes according to proximity rules; meaning that the mutation is associated to the most closest gene.

A database table, named Tbl:Mutation, is constructed from identified mutation-gene pairs (see Appendix C for detailed information on major database tables). This table will be useful when a user queries the system in order to find mutation-gene pairs related to a disease. In such a case, without any need to re-extract relations for a specific disease, relevant mutation-gene pairs will be retrieved directly from this database table. In construction of this database table, all previously identified mutations are unified so as to be represented in the same format. The format is defined to be “wild-position-mutant”. In this format “wild” denotes the wild type amino acid, “position” denotes the position of mutation on the amino acid sequence, and “mutant” denotes the mutant amino acid.

4.5. User Interface

Users can interact with MuGeX via web interface. Users are able to query the system for any disease name. Subsequent to a query, information retrieved from MuGeX is displayed on the browser. Furthermore, MuGeX is equipped with a caching mechanism. Results of at most 10 queries are stored in separate database tables; however system can easily be modified to store more than 10 query results. When there are 10 database tables and user performs a query, the table that is not accessed for the longest time is deleted; and a new table is created for the queried disease. With this caching mechanism, after querying the system for a specific disease, which takes about 1 minutes depending on the length of the query string, in repeating queries for the same disease the results are displayed at constant time.

Retrieved information comprises mutation-gene pairs, PubMed identifiers and topics of the abstracts that contain these mutation-gene pairs. Each PubMed identifier is linked out to public server of Medline. Thus, whenever user clicks on an identifier, public server of Medline displaying corresponding abstract is opened in a new window. Besides, a button is placed next to each identifier that defines an abstract that contains potentially ambiguous mutation. This button allows the user to review the abstract and label it as containing protein level mutation, DNA level mutation or cell line/strain. In the next complete run of MuGeX, this information is incorporated into training corpus that is used to train the machine learning algorithm. Thus, by incorporating user contribution we aim to improve accuracy of disambiguation step.

4.6. Conclusions

In this chapter we tried to give the methodology that is followed throughout the development of this thesis by mentioning the function of each module. The question of how the techniques given in Chapter 3 are combined together to form MuGeX system is answered. In the next chapter we explain the experiments conducted and their results.

CHAPTER 5

Experiments and Results

5.1. Introduction

This chapter presents the conducted experiments and results for MuGeX system. We first describe the test to measure performance of mutation extraction and then present resulting precision and recall values along with a discussion. Then we report performance of mutation-gene relation extraction in detail.

5.2. Performance of Mutation Extraction

To estimate recall and precision of mutation extraction, a test corpus that consists of 231 randomly selected Medline abstracts, which contain either word “mutation” or “polymorphism”, was created. It contains 472 unique mutations 280 of which are ambiguous mutations.

In order to overcome the bias caused by the occurrence of same mutation mentioned more than once in an abstract, performance of MuGeX is evaluated with respect to two metrics, namely cited mutation and contained mutation. While several occurrences of same mutation mention is considered as one individual result for contained mutation metric, in cited mutation metric, each correctly identified mutation mention is considered as individual result. While a term that is correctly identified as a mutation with or without means of disambiguation is considered as true positive, a term that is incorrectly identified as a mutation with or without means of disambiguation is considered as false positive. A mutation that is not extracted, whereas it should be, is considered as false negative. Performance of MuGeX is tested with and without the disambiguation module. The resulting recall and precision values are given in Table 4 and Table 5.

	% Precision	% Recall	% F-Measure
Cited mutation	85.8	89.0	87.4
Contained mutation	88.3	84.6	86.4

Table 4 Recall and precision values for mutation extraction (without disambiguation)

	% Precision	% Recall	% F-Measure
Cited mutation	95.9	85.9	90.6
Contained mutation	96.7	82.0	88.7

Table 5 Recall and precision values for mutation extraction (with disambiguation)

As seen in Table 4, for cited mutation extraction when disambiguation is not performed MuGeX system has 85.8% precision and 89.0% recall. When disambiguation is performed MuGeX system has 95.9% precision and 85.9% recall. So, when disambiguation is incorporated into mutation extraction process, the precision of MuGeX is increased by 10 points (11.7% relative) in return for 3.1 points (3.5% relative) decrease in recall. Besides when disambiguation is performed, F-Measure is increased by 3.2%. High precision values imply that mutations retrieved by MuGeX system are considerably relevant. Looking at recall values it may be said that MuGeX system is capable of retrieving a large portion of available mutation information from biomedical documents. Thus, the results obtained by analyzing abstracts via MuGeX system are very encouraging.

To have a better insight in mutation extraction capability of MuGeX, its performance on contained mutation extraction on the test corpus is given in Table 6.

Unidentified Mutations				Incorrect Mutations	Correct Mutations
Grammatical Mutations	Unrecognized Typing	No Regex to match	False Negative		
37	11	23	14	13	387

Table 6 Result of contained mutation extraction in numbers

As seen from Table6, MuGeX extracted 387 of 472 existing mutations. Furthermore, while MuGeX retrieved 13 incorrect mutations, a total of 85 mutations could not be identified. When the results are analyzed, it is seen that 11 of the 13 incorrect mutations are ambiguous mutations that pass the disambiguation step as false positives. The remaining incorrect mutations are irrelevant patterns that happened to be matched by some regular expression.

Most of the unidentified mutations are grammatical mutations, meaning that the mutations that are described using natural language. You may see examples of unidentified grammatical mutations in Table 7.

PubMed Id	Mutation
1541680	… *replacement of a highly conserved leucine residue by proline at position 207* in the alpha-spectrin chain ...
15288791	... the mutant version of aspartate transcarbamoylase in which *Glu50 in the catalytic chains was replaced by Ala* destabilizes ...
8452538	*... the tryptophan residues 388 and 412 in the glucose transporter GLUT1 were altered to leucine ...*
8026500	... three mutants in which *Gly156 and/or Asn157 was replaced by Phe ...*

Table 7 Examples of grammatical mutations that cannot be identified by MuGeX

The last example of Table 7 is rather interesting. While MuGeX identifies the mutation of Asn157 to Phe, it cannot identify mutation of Gly156 to Phe in this sentence. Besides these grammatical mutations, 23 mutation citations could not be identified since they could not be matched with current regular expression set of MuGeX. Examples of such mutation citations are show in Table 8.

PubMed Id	Mutation
11578065	34Asn (AAT)-to-Ser (AGT)
10896920	(15)glycine (GGT) to alanine (GCT)
1985702	Val----Glu

Table 8 Some mutation patterns that cannot be identified by MuGeX

These unidentified grammatical and unmatched mutations can be retrieved and thus recall can be increased by introducing new regular expressions or relaxing the existing ones. Furthermore, 11 mutations could not be identified due to typing errors and nonstandard citations, i.e., citing three-letter coded amino acids in all uppercase letters. Yet again, by relaxing existing regular expressions these unidentified mutations can be identified. Apart from these mutations, the loss of relevant information at disambiguation step is another cause of decrease in recall. However, this loss of information is negligible when high increase in precision is considered.

To better judge the disambiguation performance of MuGeX, we enumerated the number of correctly disambiguated and incorrectly disambiguated contained mutations. An amino acid level ambiguous mutation is judged to be correctly disambiguated if the abstract in which it is cited is classified as containing amino acid level mutation; otherwise it is judged to be incorrectly disambiguated. Same reasoning is followed for correctly and incorrectly disambiguated protein mutation-like terms. Table 9 displays results corresponding to correctly and incorrectly disambiguated mutations.

Correctly Disambiguated		Incorrectly Disambiguated	
Amino Acid Level Mut.	Other	Amino Acid Level Mut.	Other
214	41	14	11

Table 9 Disambiguation performance in numbers

As mentioned before, the test corpus contains 472 unique mutations 280 of which are ambiguous mutations. While 228 of these ambiguous mutations are amino acid level mutations, 52 of them are amino acid level mutation-like terms. Looking at Table 9, it may be said that MuGeX is capable of differentiating between amino acid level mutations and mutation-like terms with a precision of 95.1% and a recall of 93.8%. It can be concluded that utilized disambiguation method is very successful in differentiating between amino acid level mutation citations and non-amino acid level mutation citations.

However, disambiguation performance can be improved by utilizing more sophisticated machine learning algorithms; however it is important to emphasize that even with a naïve algorithm one is able to acquire accurate results. Another approach to improve performance might be to filter all mutation-like terms prior to classification; such as cell line, strain name, using a comprehensive dictionary. However, if this process is not handled carefully this methodology would produce several false negatives.

Finally, we compared performance of MuGeX with that of MEMA[22] system. The result of the comparison is given in Table 10. Recall and precision values for MEMA system are taken from the original publication. It is said in the publication that MEMA's recall and precision values were estimated over a set of 100 abstracts, each of which contain at least one mutation and this set is retrieved by querying public server of MEDLINE with keywords “mutation” and “polymorphism”. Similarly, MuGeX system is tested on a set of 231 abstracts, which are collected via querying public server of MEDLINE with the same keywords. Like MuGeX, MEMA is capable of extracting mutations that are described in natural language. Difference between these two systems arises from the type of mutations they identify. While MEMA extracts also nucleotide level mutations, MuGeX is designed to extract only amino acid level mutations.

	MEMA			MuGeX		
	%Precision	%Recall	%F-Measure	%Precision	%Recall	%F-Measure
Cited	98.6	74.7	85.0	95.9	85.9	90.6
Contained	97.9	75.3	85.1	96.7	82.0	88.7

Table 10 Comparison of MuGeX with MEMA

As seen in Table 10, MuGeX performs better than MEMA in terms of recall. Considering that MuGeX does not extract nucleotide level mutations while MEMA does, this result is promising. When performance of both systems is compared in terms of F-Measure, MuGeX system outperforms MEMA due to its ability to retrieve higher amount of existing relevant information while maintaining high precision. It is not noted how MEMA differentiates between nucleotide level mutations and amino acid level mutations. However with its disambiguation approach MuGeX is able to differentiate between not only nucleotide level mutations and amino acid level mutations, but also between point mutation like biological terms (i.e., cell lines) and amino acid level mutations successfully.

5.3. Performance of Mutation-Gene Association for Alzheimer's Disease

To measure the performance of mutation-gene association, MuGeX system was queried with the keyword "alzheimer". As a result of this query MuGeX retrieved a set of mutation gene pairs from abstracts that contain word "alzheimer". If no gene was associated with a mutation, then only mutation was retrieved. Subsequently, MuGeX retrieved 505 abstracts that contain 873 contained mutations and 808 contained mutation-gene pairs. Performance of mutation-gene relation extraction is given in numbers in Table 11.

Incorrect Relations				Unknown Relations	Correct Relations
Incorrect Association	Unidentified Gene	Incorrect Mutation	Other		
28	43	18	1	65	718

Table 11 Results of mutation-gene relation extraction

MuGeX extracted 718 correct relations along with a total of 90 incorrect relations and 65 unidentified relations. As seen from Table 11, main problem in relation extraction is low performance of gene name identification module. This is caused mainly by the fact that gene names cited in several abstracts do not exist in the gene name dictionary downloaded from HUGO Database. Also, MuGeX normalizes gene names by considering only their orthographic variants. Therefore, mutations in other variant forms; such as morphological variants, cannot be identified by exact matching technique of MuGeX. Due to unidentified gene names in context of a mutation 43 incorrect relations are built. In addition to this, again due to unidentified gene names, 65 mutations could not be associated with any gene, even though there is at least one gene citation in each abstract. 18 of 90 incorrect relations result from the wrongly identified ambiguous mutations that pass disambiguation step as false positives. Also, 28 of the 90 incorrect relations result from associating the mutation with a wrong gene. Since MuGeX utilizes only a notion of proximity, most of these 28 incorrect relations are observed when more than one gene is cited in the context of a mutation. Still, the performance of MuGeX is promising. 88.9% of mutation-gene pairs retrieved by MuGeX are correct.

We also compared our results with information found in Alzheimer Disease & Frontotemporal Dementia Mutation Database (AD&FTDMDB), which is a locus-specific database of Human Genome Variation Society.[2] This database not only contains mutations that are collected from literature and presentations at scientific meetings; but also links to scientific documents that contain those mutations.

We retrieved 183 substitution type mutations from AD&FTDMDB by querying the database with respect to phenotype. 25 of these mutations are observed on APP gene, 147 mutations are observed on PSEN1 gene and 11 mutations are observed on PSEN2 gene. Mutation-gene pairs obtained by querying MuGeX system by keyword "alzheimer" is compared with the entries retrieved from AD&FTDMDB database. Comparison results are illustrated in Table 12.

Gene	Identified	Unidentified					Total
		MEE	ABE	ADE	MDE	NIC	
APP	21	0	0	0	0	4	25
PSEN1	84	2	1	4	33	23	147
PSEN2	9	0	0	1	1	0	11
Cumulative	114	2	1	5	34	27	183

Note: MEE: Mutation Extraction Error. ABE: Association Building Error. ADE: Reference abstract does not contain disease name. MDE: Reference abstract does not mention the mutation. NIC: Reference abstract is not in our corpus.

Table 12 Comparison of MuGeX results with data extracted from AD&FTDMDB

Due to reasons that are not related to performance of MuGeX system, 66 mutation-gene pairs could not be identified. While documents that cite five of these 66 mutation-gene pairs do not mention disease name in their abstracts, documents that discuss 34 mutation-gene pairs do not mention the cited mutation in their abstracts. Besides these, documents that cite 27 pairs do not exist in our corpus since they are dated after January 1, 2005. However, MuGeX is very good at extracting remaining mutation-gene pairs. It extracts 97.4% of remaining 117 mutation-gene pairs correctly.

We believe that if the corpus is extended so as to contain abstracts of documents that are published thus far, number of identified mutation-gene pairs will increase. However, it should be noted that retrieval of mutation-gene pairs is limited by the information given in the abstract. And usually the information contained in the abstracts is insufficient. For instance, in this case, most of the unidentified mutation-gene pairs result from incited mutations. Therefore, to improve recall, it might be a better idea to analyze full text articles instead of abstracts. However, processing full text articles brings about extra overhead and different processing.

5.4. Conclusions

In this chapter we reported the results of our experiments. The performance of MuGeX is measured in two aspects: its performance in mutation extraction and its performance in relation extraction. We tried to discuss the outcome of experiments and addressed the question of why certain erroneous extractions are observed in detail.

Our results indicate that MuGeX is successful at mutation extraction. It is observed that main problem that influences performance of mutation extraction in terms of recall is insufficiency of regular expression set of MuGeX. To overcome this problem the existing regular expressions may be relaxed or new regular expressions may be created. Moreover, it is seen that utilized disambiguation method is very successful in differentiating between amino acid level mutations and non-amino acid level mutations. However, further improvements may be achieved with utilization of more sophisticated machine learning algorithms.

The relation extraction performance of MuGeX is also encouraging. The principal factor that affects relation extraction performance of MuGeX is the identification of gene mentions. Performance of the system in terms of both recall and precision can be improved by utilization of a more comprehensive gene name dictionary or a combination of several gene name dictionaries. Further improvements will be achieved by extending the gene name normalization procedure so as to cover more variation types.

CHAPTER 6

Conclusions

In this thesis, we presented design and implementation of a system that is capable of automatically extracting mutation-gene pairs from MEDLINE abstracts for a given disease name.

In mutation extraction, performance of MuGeX is satisfactory. MuGeX retrieves large amount of existing relevant mutation information while maintaining accuracy of extracted mutations at a high level. MuGeX differs from existing systems of mutation extraction in the way it handles ambiguous mutation citations. By looking at context of the mutation-like entities and following a supervised approach, MuGeX is capable of differentiating between not only nucleotide level mutations and amino acid level mutations, but also between point mutation like biological terms and amino acid level mutations successfully.

In relation extraction, performance of MuGeX is highly encouraging. It is observed that most of the extracted incorrect and unidentified relations are caused by unidentified gene names. However, for abstracts where genes are identified correctly, it is seen that the relations that are built by MuGeX are correct with high probability. For Alzheimer's disease, MuGeX retrieves a large portion of the previously known and approved mutation-gene pairs. Thus, we may draw the conclusion that MuGeX retrieves highly relevant information. Besides these approved mutation-gene pairs, MuGeX retrieves additional pairs that are in some way related to Alzheimer's disease. For instance, pairs that consist of mutations, which are synthetically produced as a result of mutagenesis experiments are also retrieved in context of Alzheimer's disease. In extreme case, where all these additional pairs are considered to be irrelevant, MuGeX will still be regarded as a versatile system that lessens the search space of researchers with the information it retrieves, given that it assures that a portion of the retrieved information is highly relevant.

One of the advantages of MuGeX is that its memory requirements are very modest compared to similar systems. It is fully functional on a machine with 100-150 MB free main memory. Also, the complete run of MuGeX with a corpus of 376000 abstracts on the same machine takes about 16 hours, which is considerably short compared to similar systems. Moreover as long as the MEDLINE corpus that constitutes basis of the system is kept up to date the time required to access relevant information is very short. Querying the system for a disease name takes about 30 seconds, depending on the length of the query term. However, this advantage is also a handicap since it requires the system to be executed from beginning to end. Still, since the complete run of MuGeX takes a reasonable time, running the system once in a month does not seem to be a major concern. Nevertheless as future work, development of a runtime update mechanism is also planned that avoids the execution of the system from beginning to end when the basis corpus is renewed.

Currently, MuGeX does not give semantic meaning to the extracted mutation-gene relations. It is just known that the extracted mutation and its associated gene are related somehow; but it is not known whether the relation is positive or negative; meaning that the mutation is eventuated on that gene or not. Same vagueness exists between the disease name and the mutation-gene pairs that are associated to it. With utilization of natural language processing techniques semantic meanings can be given to extracted relations in the future.

Appendix A – List of Commonly Used Abbreviations

PEOPLE	jr, mr, mrs, ms, dr, prof, sr, gov, supt, det, rev
ARMY	col, gen, lt, cmdr, adm, capt, sgt, cpl, maj
INSTITUTES	dept, univ, assn, bros
COMPANIES	inc, ltd, co, corp
PLACES	arc, al, ave, cl, ct, cres, dr, dist, mt, ft, la, pl, plz, rd, st, tce, Ariz, Ark, Cal, Calif, Col, Colo, Conn, Del, Fed, Ga, Ida, Id, Ill, Ind, Ia, Kan, Kans, Ken, Ky, La, Me, Md, Is, Mich, Minn, Miss, Mo, Mont, Neb, Nebr, Nev, Mex, Okla, Ok, Ore, Penna, Penn, Pa, Dak, Tenn, Tex, Ut, Vt, Va, Wash, Wis, Wisc, Wy, USAFA, Alta, Man, Ont, Que, Sask, Tuk
MONTHS	jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec
MISCELLENCOUS	vs, etc, no, esp, Gen, Genet, Mol, resp, ca, i.e., pv, etc, ed, p, sp

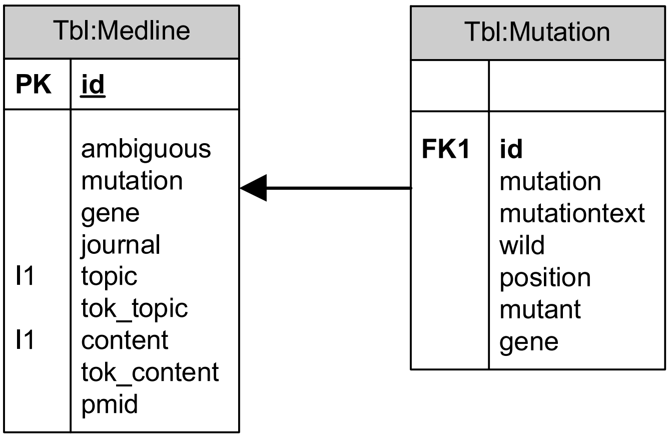
Appendix B – Stopword List

a
able
about
above
according
accordingly
across
actually
affected
affecting
affects
after
afterwards
again
against
all
allow
allows
almost
alone
along
already
also
although
always
am
among
amongst
an
and
another
any
anybody
anyhow
anyone
anything
anyway
anywhere
apart
apparently
appear
appreciate
appropriate
are
arise
around
as
aside
ask
associated
at
available
away
awfully
be
became
because
become
been
before
beforehand
behind
being
believe
below
beside
best
better
between
beyond
boil
both
brief
briefly
but
by
cal
came
can
cannot
cant
cause
certain
change
chem
clearly
co
com
come
concerning
consequently
consider
contain
containing
contains
copyright
corresponding
could
course
currently
definitely
described
despite
did
different
do
does
doing
done
down
downwards
dr
due
during
each
edu
effect
eg
either
else
elsewhere
enough
entirely
especially
et
etc
even
ever
every
everybody
everyone
everything
everywhere
ex
exactly
example
except
furthermore
few
followed
following
for
former
formerly
forth
found
from
further
far
gave
get
gets
give
given
gives
giving
go
goes
going
gone
got
gotten
had
happens
hardly
has
have
having
he
help
hence
her
here
hereafter
hereby
herein
hereupon
hers
herself
him
himself
his
hither
how
howbeit
however
ie
if
ignored
immediate
importance
important
inc
indeed
indicate
indicated
indicates
inner
instead
into
inward
is
it
its
itself
just
keep
keeps
kept
kg
kcal
kma
km
know
knowledge
known
knows
largely
last
lately
later
latter
least
less
lest
let
like
likely
little
look
looking
looks
ltd
made
mainly
make
many
may
maybe
me
mean
meanwhile
merely
mg
might
ml
mm
more
moreover
most
mostly
much
must
my
myself
name
namely
near
nearly
necessarily
necessary
need
needs
neither
never
new
next
no
nobody
non
none
nor
normally
not
nothing
novel

now
nowhere
obtain
obtained
obviously
of
off
often
old
on
once
ones
only
onto
or
other
others
otherwise
ought
our
ours
ourselves
out
outside
over
overall
owing
own
particular
particularly
past
per
perhaps
plus
poorly
possible
possibly
potentially
present
presumably
previously
primarily
probably
quickly
quite
rather
rd
readily
really
reasonably
recently
refs
regarding
regardless
regards
relatively
respectively
resulted
resulting
results
right
said
same
say
saying
says
second
secondly
see
seeing
seem
seemed
seems
seen
self
selves
sensible
sent
several
shall
she
should
show
showed
shown
shows
significantly
similar
similarly
since
slightly
so
some
somebody
somehow
someone
something
sometime
sometimes
somewhat
somewhere
soon
specifically
specified
specify
specifying
state
states
still
strongly
sub
substantially
successfully
such
sufficiently
sup
sure
take
taken
tell
than
that
that’s
the
their
theirs
them
themselves
then
there
thereafter
thereby
therefore
therein
thereupon
these
they
think
this
thorough
thoroughly
those
though
through
throughout
thus
to
together
Too
took
toward
towards
tried
tries
truly
try
trying
under
un
unfortunately
unless
unlikely
until
unto
up
upon
us
use
used
useful
usefully
uses
using
usually
value
various
very
via
viz
vs
want
wants
was
way
we
Well
went
were
what
whatever
when
whenever
where
whereas
whereby
wherein
whereupon
wherever
whether
which
while
who
whoever
whole
whom
whose
why
widely
will
willing
wish
with
within
without
wonder
would
yes
yet
you
your
yourself
yourselves

Appendix C – Major Database Tables

In construction of database and database tables we made use of MySQL. The database contains two major tables, namely medline and mutation. Structures of these tables are illustrated below.



Tbl:Medline	
PK	**id**
	ambiguous
	mutation
	gene
	journal
I1	topic
	tok_topic
I1	content
	tok_content
	pmid

Tbl:Mutation	
FK1	**id**
	mutation
	mutationtext
	wild
	position
	mutant
	gene

“Tbl:Medline” stores information on the abstracts that are retrieved from public server of Medline. Primary key of medline table is the unique identifier given to each abstract sequentially. The field named “ambiguous” takes on binary values. If an abstract contains potentially ambiguous mutation then value of this field is set to be 1; otherwise it is set as 0. Similarly “mutation” and “gene” fields of the table take on binary values. Values of these fields are set according to the existence of mutations and genes. “journal”, “topic”, “content” and “pmid” fields of the table store information on journal, topic, content and pubmed id sections of an abstract respectively. What is meant by content of an abstract is actually the body of the abstract. Finally, while “tok_topic” field keeps tokenized version of the topic section, “tok_content” keeps tokenized version of the content section of an abstract. It is important to note that Medline table contains a full-text index, named as I1, on topic and content fields, which provides the system with fast text search capability on these fields. This feature is extremely useful when querying MuGeX system against a disease name.

“Tbl:Mutation” is used for storing information on mutations that are extracted from the abstracts in the corpus. Mutation table does not have any primary key, yet it is related to mutation table via the field named “id”. “mutation” field of the table contains the mutation in wild-position-mutant format, where wild denotes wild type amino acid, position denotes the position of mutation on the amino acid sequence and mutant denotes the mutant amino acid. “mutationtext” field contains the mutation citation as it is matched in the abstract. For instance, let the following sentence be part of an abstract: “It is observed that in several patients Valine at position 114 is replaced with Isoleucine”. Then for this abstract an entry will be inserted into mutation table. While “mutation” field for this entry will be “V-114-I”, “mutationtext” field for this entry will be “Valine at position 114 is replaced with Isoleucine”. Besides these, in order to store wild type amino acid, position of mutation and mutant amino acid the table contains three fields, namely “wild”, “position” and “mutant” respectively. Finally, the field named as “gene” stores the name of the gene, which is associated with the mutation.

REFERENCES

1. Altman R. B., Chang J. T., Schütze H. GAPSCORE: finding gene and protein names one word at a time, *BMC Bioinformatics*. Vol. 20, pp. 216-225, 2004.
2. Alzheimer Disease & Frontotemporal Dementia Mutation Database. http://www.molgen.ua.ac.be/ADMutations/. 26-April-2007.
3. Baker J.O., Witte R. Combining Biological Databases and Text Mining to Support New Bioinformatics Applications, *10th International Conference on Applications of Natural Language to Information Systems (NLDB)*, Springer LNCS. pp. 310-321, 2005.
4. Baker J.O., Witte R. Enriching Protein Structure Visualizations with Mutation Annotations Obtained by Text Mining Protein Engineering Literature, in: *The Third Canadian Working Conference on Computational Biology (CCCB'04)*, Markham, Ontario, Canada, 2004.
5. Bison – GNU Parser Generator. http://www.gnu.org/software/bison/. 21-June-2007.
6. Brill E. A simple rule-based part-of-speech tagger, *Proceedings of {ANLP}-92, 3rd Conference on Applied Natural Language Processing*, pp. 152-155, 1992.
7. Collier N., Nobata C., Tsujii J. Extracting the Names of Genes and Gene Products with a Hidden Markov Model, *Proceedings of COLING-2000*, 2000.
8. Database of WS-associated WRN mutations. http://www.pathology.washington.edu/research/werner/ws_wrn.html. 25-July-2007.
9. Flex: The Fast Lexical Analyzer. http://flex.sourceforge.net/. 21-June-2007.
10. Hanisch D., Fundel K., Mevissen H.T., Zimmer R., Fluck J. ProMiner: rule-based protein and gene entity recognition, *BMC Bioinformatics*. Vol. 6, S14, 2005.
11. Hopcroft J.E., Motwani R., Ullman J.D. *Introduction to Automata Theory, Languages, and Computation*, 2nd Edition, Addison-Wesley, pp. 83-113, 2001.
12. Horn F., Lau A.L., Cohen F.E. Automated extraction of mutation data from the literature: Application of MuteXt to G protein-coupled receptors and nuclear hormone receptors, *BMC Bioinformatics*. Vol. 20, pp. 557–568, 2004.
13. HUGO Gene Nomenclature Committee. http://www.gene.ucl.ac.uk/nomenclature/. 26-April-2007.

14. Keio Mutation Database for eye disease genes (KMeyeDB). http://mutview.dmb.med.keio.ac.jp/MutationView/jsp/index.jsp. 25-July-2007.
15. Krauthammer M., Kra P., Iossifov I., Gomez S.M., Hripcsak G., Hatzivassiloglou V., Friedman C., Rzhetsky A. Of truth and pathways: chasing bits of information through myriads of articles, *BMC Bioinformatics*. Vol. 18, supplement pp. 249–57, 2002.
16. Lee L.C., Horn F., Cohen F.E. Automatic Extraction of Protein Point Mutations Using a Graph Bigram Association, *PLoS Comput Biol*. 3(2): e16, 2007.
17. McCallum A.K. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. http://www.cs.cmu.edu/~mccallum/bow. 1996.
18. Mitchell T. *Machine Learning*, McGraw Hill, 1997.
19. MUC-7 Information Extraction Task Definition. http://www-nlpir.nist.gov/related_projects/muc/proceedings/ie_task.html. 13-June-2007.
20. Porter M. F., An algorithm for suffix stripping, *Program*, **14** (3), pp 130−137, 1980.
21. Princeton University Biology Library Medline Database – Stop Words. http://biolib.princeton.edu/instruct/MedSW.html. 10-April-2007.
22. Rebholz-Schuhmann D., Mercel S., Albert S., Tolle R., Casari G., Kirsch H. Automatic extraction of mutations from Medline and cross-validation with OMIM, *Nucleic Acids Research*. Vol. 32, pp. 135–142, 2004.
23. Rocchio J.J. Relevance feedback in information retrieval, in Salton, *The SMART Retrieval System Experiments in Automatic Document Processing*, Prentice-Hall, pp.313–323, 1971.
24. Sipser M. *Introduction to the Theory of Computation*, PWS Publishing Company, Boston, pp. 63-77, 1997.
25. Tanabe L., Wilbur W. Tagging gene and protein names in biomedical text, *BMC Bioinformatics*. Vol. 18, pp. 1124-1132, 2002.
26. Tsuruoka Y., Tsujii J. Boosting precision and recall of dictionary-based protein name recognition, *Proceedings of the ACL-03 Workshop on Natural Language Processing in Biomedicine*. pp. 41-48, 2003.
27. Wall L., Christiansen T., Orwant J. *Programming Perl*, 3rd Edition, O'Reilly, California, Chapter 5, 2000.