Journal of Biomedical Informatics 42 (2009) 1046–1055

Contents lists available at ScienceDirect

Journal of Biomedical Informatics

journal homepage: www.elsevier.com/locate/yjbin

# Two learning approaches for protein name extraction

Serhan Tatar, Ilyas Cicekli *

*Department of Computer Engineering, Bilkent University, 06800 Bilkent, Ankara, Turkey*

ARTICLE INFO

*Article history:*
Received 4 March 2009
Available online 13 May 2009

*Keywords:*
Statistical learning
Bigram language model
Rule learning
Protein name extraction
Information extraction

ABSTRACT

Protein name extraction, one of the basic tasks in automatic extraction of information from biological texts, remains challenging. In this paper, we explore the use of two different machine learning techniques and present the results of the conducted experiments. In the first method, Bigram language model is used to extract protein names. In the latter, we use an automatic rule learning method that can identify protein names located in the biological texts. In both cases, we generalize protein names by using hierarchically categorized syntactic token types.

We conducted our experiments on two different datasets. Our first method based on Bigram language model achieved an *F*-score of 67.7% on the YAPEX dataset and 66.8% on the GENIA corpus. The developed rule learning method obtained 61.8% *F*-score value on the YAPEX dataset and 61.0% on the GENIA corpus. The results of the comparative experiments demonstrate that both techniques are applicable to the task of automatic protein name extraction, a prerequisite for the large-scale processing of biomedical literature.



## 1. Introduction

Biological knowledge, generated as a result of biological research, is currently stored in scientific publications which can be accessed via different knowledge sources storing vast amounts of information – Medline being a prominent example. Knowledge sources do not, however, feature a formal structure in which to access stored information, thus rendering information search, retrieval and processing especially tedious and time-consuming. This consequently results in a strong demand for automatized discovery and extraction of information.

Our research focus is on protein name extraction from unstructured texts, a prerequisite for further information processing. Though a basic task of automatic extraction of information from biological texts, it remains challenging to perform. Information extraction performed at different levels can be viewed as a layered structure, which makes the different extraction tasks at different layers interdependent. In other words, because the output of a task on one layer becomes input to the next layer, the success of a former task impacts the success of a latter one. For example, accurate location of protein names in a text will then affect the finding of interactions between proteins [1].

Parallel to the continuous growth in the biological knowledge, the number of protein names is rapidly increasing. New names, however, are not necessarily recorded in terms of standard terminology, thus complicating the information extraction process. Moreover, authors often refer to already named proteins using new variations, reflecting the fact that standards, rules or conventions for naming proteins are not necessarily well-established. Because protein names are generally depictive, they include many common words describing them, which cause difficulty in distinguishing protein name boundaries from the general language text. Abbreviations are also frequently used, and the way in which these are formed is yet another source of difficulty (e.g. "LRP6" for "Low density lipoprotein Receptor-related Protein 6", "IL5" for "Inter-Leukin 5", "Ran" for "RAs-related Nuclear protein"). Furthermore, protein names may overlap with other biomedical terms; such as genes, cell cultures or chemical compounds. To make things worse, exploiting contextual information is not very helpful in the extraction process due to the fact that protein names are usually multi-token and include symbols, common nouns, adjectives, adverbs, and even conjunctions, which makes it difficult to distinguish them from surrounding texts [2]. Zhou et al. [3] collected the reasons causing the difficulty in protein name extraction task under five main titles: descriptive naming conventions, conjunctions and disjunctions, non-standardized naming conventions, abbreviations, and cascaded constructions.

Information extraction (IE) systems can be used as a solution for automatic extraction of protein names from unstructured biological texts. IE can be defined as the identification of selected types of entities, relations, facts or events in a set of unstructured text documents written in natural language. It is the process of analyzing unstructured texts and extracting the necessary information into a structured representation, or, simply put, the process of

* Corresponding author. Fax: +90 312 266 4047.
*E-mail addresses:* statar@cs.bilkent.edu.tr (S. Tatar), ilyas@cs.bilkent.edu.tr (I. Cicekli).

1532-0464/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jbi.2009.05.004

selective information structuring [4]. Considering all the IE systems developed so far for protein name extraction, which we will summarize in the following section, systems often rely on knowledge-source based and rule-based approaches. Since both approaches require (essentially manual) careful analysis of the biomedical domain, most systems require manual development of the resources (i.e. dictionaries, rules etc.) by human experts. Furthermore, given rapid changes in biomedical knowledge, keeping the resources used by these systems up-to-date with the changes in the domain is another issue which requires constant human intervention. These issues also make it difficult to adapt the developed systems to extract other entity types.

This paper describes the use of supervised learning strategy to extract protein names from biological texts and presents the results of the conducted experiments. The principal goal is to develop IE systems that learn to extract protein names from examples and achieve acceptable extraction performance without using the resources requiring manual effort and time. Our approach is to start with a set of protein names collected from a training set, and generalize the initial set using a carefully designed strategy, while at the same time trying to minimize reduction in accuracy. In order to obtain accurate generalization and overcome the problem of data sparseness, we propose using hierarchically categorized syntactic token types. Many projects have so far used syntactic token types/patterns based on the surface features of the constituents for detecting protein names and their fragments. However, to the best of our knowledge, ours is the first study exploiting a hierarchy of syntactic token patterns. We have developed two novel methods for the extraction task. The first method follows a probabilistic approach for protein name identification. More specifically, we adapted the conceived hierarchy of syntactic token patterns concept to Bigram language model in order to address the problem of protein name extraction. In the second method, we developed an automatic rule learning algorithm to find protein names located in the biological texts. According to our rule generalization method, the learner first derives the unique match sequences, that is a sequence of similarities and differences between two strings, from the given protein names. The extraction rules are then generated by generalizing the differences in the derived unique match sequences using the hierarchical syntactic token types.

The structure of the paper is as follows. The next section summarizes the previous research done in this field. The following section describes how we applied Bigram language model to protein name extraction problem and employed rule learning for the task of protein name identification. After presenting the studied methods, Section 4 shows the results of the experimental evaluation of the study. Finally, in the last section we discuss our conclusions, pointing towards future research.

## 2. Related work

In terms of information extraction from biological texts, namely basic term identification to more complex relation extraction tasks, many IE projects have focused on dictionary-based, rule-based, machine learning, statistical, and hybrid approaches, each with its advantages and disadvantages. Krauthammer and Nenadic [5] have surveyed state-of-art approaches for addressing the protein name identification problem in the context of term identification in the biomedical literature.

There have been several attempts to develop dictionary-based techniques [6–10] for finding protein names in biological texts. Dictionary-based methods use existing protein databases in order to locate protein occurrences in text. Tsuruoka and Tsujii [8,9] reported $F = 66.6\%$ ($F$-score: harmonic average between accuracy and coverage) on GENIA corpus [11] by using approximate string searching techniques and expanding the dictionary in advance with a probabilistic variant generator. Krauthammer et al. [6] reported $F = 75\%$ (considering partial matches as correct) on a set of two papers. With a decrease on the performance score, the proposed system achieved $F = 59.8\%$ when only exact matches were considered as correct. Although their system fully matched 94.6% of the names marked by the evaluators and available in the BLAST database, it was able to fully match only 4.4% of names marked by the evaluators and not available in the database. The system attempts to perform approximate string matching after converting both dictionary entries and input texts into nucleotide sequence-like strings, which can be then compared by BLAST [7], a DNA and protein sequence comparison tool. More recently, Schuemie et al. [10] combined information from existing gene and protein databases and generated spelling variations according to rules for automatic generation of a comprehensive dictionary. Because of the name variations in referring to the same protein and the new protein names not yet found in the dictionaries, dictionary-based techniques are not totally effective on their own. Another major drawback of the dictionary-based methods is the need for regular dictionary updates.

Rule-based approaches [12–16] assume that protein names occur in texts in certain patterns and these patterns can be discovered and expressed by a meta-language. Thus, they attempt to locate new names by exploiting the determined patterns, or rules. PROPER (PROtein Proper-noun phrase Extracting Rules) system [12] achieved $F = 96.7\%$ on a set of 30 abstracts using simple lexical patterns and orthographic features. Franzen et al. [13] introduced the YAPEX system that combines lexical and syntactic knowledge, heuristic rules and a document-local dynamic dictionary. The YAPEX system reached a recall of 61.0% and a precision of 62.0% on a dataset which consists of 200 abstracts. Seki and Mostafa [16] reported $F = 63.7\%$ on the same dataset using hand-crafted rules based on surface clues reflecting the characteristics of protein names and a protein name dictionary. The PASTA (Protein Active Site Template Acquisition) system [14,15], a pipeline of processing modules, performs protein name extraction in seven steps: text preprocessing, terminological processing, syntactic and semantic analysis, discourse interpretation, and template extraction. In the experiments, the PASTA system achieved $F = 85\%$ on a set of 61 abstracts. Though rule-based systems have demonstrated remarkable performance and rules in such systems can be flexibly defined and extended as needed, rule development and management is the main issue in these systems, since manually analyzing biological texts and crafting rules require high human expertise and are often time-consuming. Moreover, domain specific rules constructed for a domain can not be easily reused for a new domain.

Machine learning techniques are also widely used in order to perform protein name extraction from texts. Bunescu et al. [17] developed and evaluated several learning systems for locating protein names in Medline abstracts. They performed comparative experiments on various systems: RAPIER [18], boosted wrapper induction (BWI) [19], memory-based learning (MBL) [20], transformation-based learning (TBL) [21], support vector machines (SVMs) [22], and maximum entropy (Max-Ent) [23]. The Max-Ent method that uses a "dictionary tagger", achieved the best result among the others with $F = 57.9\%$ on the University of Texas, Austin dataset which consists of 748 abstracts. Earlier studies [3,17,24–31] in the IE community have shown that statistical techniques can be of service in performing protein name extraction tasks. Hidden Markov Models (HMMs), one of the successful statistical learning techniques, have been applied to different IE tasks [3,24,27,30]. Several studies [25,26,28] have concentrated on the use of support vector machines (SVMs). Seki and Mostafa [31] reported $F = 63.3\%$ on the YAPEX dataset by employing a probabilistic model together with the surface clues specified for identifying protein names with

an emphasis on finding name boundaries. By avoiding deep language analysis, they reduced both processing overhead and the number of probabilistic parameters to be estimated. Machine learning techniques, using training data for learning process, generate rules via generalization of examples instead of creating patterns by hand. Statistical techniques are also relatively easy to apply as long as appropriate models and training data are available. However, in order to achieve good coverage both need a large annotated corpus (as training data) whose preparation requires domain experts and is also time-consuming.

Hybrid methods [2,28,27,32–34] combining diverse approaches and various resources are also applied to the protein name recognition task. Yu et al. [32] combined pattern-recognition and knowledge-based approaches to identify gene/protein terms in MEDLINE abstracts. Tanabe and Wilbur [2] proposed ABGene system which uses both statistical and knowledge-based strategies for finding gene and protein names. They adapted the Brill tagger [35] to protein and gene name recognition problem. NLProt [28], a system that combines a pre-processing dictionary and rule-based filtering step with several separately trained support vector machines to identify protein names in the MEDLINE abstracts, achieved $F = 75\%$ on the YAPEX dataset. SemiCRF and DictHMM, two methods developed by Kou et al. [27], achieved levels of $F = 66.1\%$ and $F = 51\%$ on the YAPEX dataset respectively.

In this paper, we propose two different learning methods. Both methods utilize supervised learning strategy. Contrary to many of the earlier studies, we did not use any resource requiring manual effort and time (e.g. dictionaries, hand-coded filtering rules and so on). Furthermore, we avoid deep syntactic/semantic analysis in order to reduce processing overhead. We focus our efforts on the pure learning performance of the extractors. The proposed methods in the paper exploit hierarchy of syntactic token patterns, a concept no previous study has examined, to generalize protein names accurately and address the problem of data sparseness. Moreover, we propose novel methods for both rule-learning and Bigram calculation.

## 3. Methods

### 3.1. Hierarchical syntactic token types

In spite of the irregularities and the lack of common standards, and all the problems stated above, protein names exhibit a degree of regularity that becomes a basis for generalization. Protein names are almost always depictive. That is, the majority of the phrases used to name proteins include words reflecting the characteristics of the named protein (function, localization, physical properties etc.). Names are also constructed using a combination or abbreviation of such characteristics and often consist of multiple words [13]. The regularities observed in the syntactic structure of the words appear in protein names are also useful for detecting them.

Two important criteria that determine the efficacy and the success of a protein name extractor are (1) the ability to recognize unseen protein names, and (2) the ability to precisely distinguish protein names from non-protein names. Both criteria require accurate generalization of the known protein names. Generalizing means to recognize the parts susceptible of being changed in new protein names, and represent them with generic placeholders [17]. In our study, we generalize protein names by using hierarchically categorized syntactic token types. In addition to accurate generalization, using hierarchically categorized syntactic token types will help overcome data sparseness problem that occurs because of the diversity of the language constructs and the insufficiency of the input data. When we consider all possible language constructs, it is not possible to observe most of the sequences during training of the language model.

We determine a two level hierarchy of useful syntactic token types for representing the protein names. Prior to assigning appropriate hierarchical token types to tokens, the input text is segmented in sentences and tokenized. We followed the standard tokenization method which uses white-space and punctuation characters as delimiters except that we removed the apostrophe symbol (') from our delimiter set as we use the symbol in our token type patterns.

Table 1 shows our hierarchy with token types and their patterns as well as the priority of the token types; the list order is the priority order of the tokens. 21 syntactic token types in Table 1 are categorized under the following five main classes:

- *Single:* The tokens in this class are usually used to diversify a protein name. Single Letter (e.g. "A", "B"), Number (e.g. "1", "2"), Roman Numeral (e.g. "I", "V") and Greek Letter (e.g. "alpha", "beta") are the token types that are classified under this class.

**Table 1**
Hierarchically categorized syntactic token types.

| Class | Token type | Pattern | Length |
|---|---|---|---|
| Single | Roman Numeral | I\|II\|III\|IV\|V\|VI\|VII\|VIII\|IX\|X\|XI\|XII\|XIII\|XIV\|XV\|XVI\|XVII\|XVIII | No restrictions |
| Single | Number | [0–9]+ | No restrictions |
| Single | Single Letter | [a–zA–Z] | 1 |
| Single | Greek Letter | alpha\|beta\|gamma\|delta\|epsilon\|theta\|kappa\|lambda\|sigma\|mu | No restrictions |
| Abbreviation | Very Long Abbreviation | [a–zA–Z] + ([A–Z][a–z]* \| [0–9]+) ([a–zA–Z] + \| [0–9] + \| ['])* | Length > 12 |
| Abbreviation | Long Abbreviation | [a–zA–Z] + ([A–Z][a–z]* \| [0–9]+) ([a–zA–Z] + \| [0–9] + \| ['])* | Length > 7; length < 13 |
| Abbreviation | Abbreviation | [a–zA–Z] + ([A–Z][a–z]* \| [0–9]+) ([a–zA–Z] + \| [0–9] + \| ['])* | Length > 3; length < 7 |
| Abbreviation | Short Abbreviation | [a–zA–Z] + ([A–Z][a–z]* \| [0–9]+) ([a–zA–Z] + \| [0–9] + \| ['])* | Length = 3 |
| Abbreviation | Very Short Abbreviation | [a–zA–Z] + ([A–Z][a–z]* \| [0–9]+) ([a–zA–Z] + \| [0–9] + \| ['])* | Length = 2 |
| Delimiter | Frequent | [. ( ) – /] | No restrictions |
| Delimiter | Rare | [: { } < >] | No restrictions |
| Delimiter | Very Rare | [% = ; , +] | No restrictions |
| Regular | Long Frequent Type-1 | [a–zA–Z] + (ase\|gen) | Length > 8 |
| Regular | Frequent Type-1 | [a–zA–Z] + (ase\|gen) | Length < 9 |
| Regular | Frequent Type-2 | [a–zA–Z] + (in) | No restrictions |
| Regular | Frequent Type-3 | [a–zA–Z] + (al\|um\|ide) | No restrictions |
| Regular | Lower Case | [a–z][a–z']+ | No restrictions |
| Regular | Long Proper Case | [A–Z][a–z']+ | Length > 9 |
| Regular | Proper Case | [A–Z][a–z']+ | Length > 3; length < 10 |
| Regular | Short Proper Case | [A–Z][a–z']+ | Length<4 |
| Other | Other | No specific pattern | No restrictions |

- *Abbreviation:* The tokens in this class are abbreviations, which consist of both alphabetic and numeric characters. There are five token types under this class, they are classified according to their length: Very Long Abbreviation (e.g. "KKLSMYGVDLHKAKDL"), Long Abbreviation (e.g. "AcAFYHSK5OEQ"), Abbreviation (e.g. "CIN85"), Short Abbreviation (e.g. "AMP"), and Very Short Abbreviation (e.g. "AD").
- *Delimiter:* According to their frequency in the protein names, there are three types of delimiters under this class: Frequent (e.g. "-", "/"), Rare (e.g. ":", "<"), Very Rare (e.g. "%", "=").
- *Regular:* This class is the broadest class in our hierarchy and includes the tokens containing alphabetic characters only. The criteria used for classifying word types under this class are length, case and frequency. There are eight word types sorted under it: Frequent Type-1 (tokens suffixed with "ase" or "gen" and have less than nine characters, e.g. "kinase"), Long Frequent Type-1 (tokens suffixed with "ase" or "gen" and have more than eight characters, e.g. "acetyltransferase"), Frequent Type-2 (tokens suffixed with "in", e.g. "actin"), Frequent Type-3 (tokens suffixed with "al", "um" or "ide", e.g. "antiserum"), Lower Case (not of Frequent Types and consists of all lowercase letters, e.g. "chemokine"), Proper Case (not of Frequent Types, only first letter is uppercase and have length restrictions, e.g. "Academic"), Short Proper Case (not of Frequent Types, only first letter is uppercase and have length restrictions, e.g. "Ala"), and Long Proper Case (not of Frequent Types, only first letter is uppercase and have length restrictions, e.g. "Accordingly").
- *Other:* This class contains the tokens that cannot be grouped into the above classes.

### 3.2. Bigram language model for protein names

In this first method, we use a Bigram language model, a special case of N-gram which is used in various areas of statistical natural language processing, along with the hierarchically categorized syntactic word types in order to identify protein names. Statistical language models date back to Shannon's work on information theory [36]. One of the basic aims of the statistical language models is to predict the probability of the next word, given the previous word sequence: $P(w_i \mid w_1, \ldots w_{i-1})$. Given a Bigram language model, it is straightforward to compute the probability of a word sequence as follows:

$$P(w_1, w_2, \ldots, w_{i-1}, w_i) = P(w_1) \quad P(w_2 \mid w_1) \ldots P(w_i \mid w_{i-1}) \tag{1}$$

We used a modified version of Eq. (1) in order to apply the Bigram language model to the problem. Given a token $w_i$, $P_{single}(w_i)$ denotes the unigram probability of token $w_i$ being a single-word protein name, $P_{first}(w_i)$ denotes the unigram probability of token $w_i$ being the first word in a protein name, $P_{last}(w_i)$ denotes the unigram probability of token $w_i$ being the last word in a protein name, and the conditional probability $P_{in}(w_i \mid w_{i-1})$ represents the Bigram probability of the fragment $\langle w_{i-1}, w_i \rangle$ being a constituent in a protein name. After defining the unigram and Bigram probabilities, our Bigram language model calculates the likelihood of a given token sequence $\langle w_1, w_2, \ldots, w_{i-1}, w_i \rangle$ being a protein name as in Eq. (2).

$$P_{prot}(W_1, w_2, \ldots, w_{i-1}, w_i) = \begin{cases} P_{single}(w_i) & \text{if } i = 1 \\ P_{first}(w_i)\, P_{in}(w_2 \mid w_1 \ldots P_{in}(w_i \mid w_{i-1}\, P_{last}(w) & \text{if } i > 1 \end{cases} \tag{2}$$

Our first modification on Eq. (1) is calculating the unigram probabilities in three different categories according to their positions: (1) *single* for the single-word protein names, (2) *first* for the words in the first position in a multiple-word protein name, and (3) *last* for the words in the last position in a multiple-word protein name. The second modification is adding a new factor, the unigram probability of the last token being the last word in a protein name, to Eq. (1). The modification is based on the observation that the rightmost words of the protein names exhibits considerable degree of regularity and carry important information for extraction purposes. For example, the number of unique words appears in the last position in the protein names tagged in the YAPEX corpus is 110[1]. This number is 175 for the words in the first position, and 150 for the words in middle positions, 258 for the words preceding a protein name and 287 for the words following a protein name. Moreover, about half of the words appear in the last position can be grouped into *Single* class in our hierarchy.

As stated in Section 2, our system uses a hierarchy of syntactic token patterns to generalize the tokens and reduce the influence of the data sparseness problem. In the absence of a token probability ($P_{single}$, $P_{first}$, $P_{last}$, $P_{in}$), our system looks for a good substitute for the token probability to put into Eq. (2). Token type probabilities $P(t_i)$ and token class probabilities $P(c_i)$ are added to the model to substitute the token probabilities $P(w_i)$. We simply substitute the token probability by the type probability, if the type probability is available. Otherwise, we use the token class probability for the substitution. Eqs. (3)–(5) describes how our model assigns the unigram probabilities in the absence of the token probabilities.

$$P_{single}(w_i) = \begin{cases} P_{single}(t_i) & \text{if } P_{single}(t_i) \text{ available} \\ P_{single}(c_i) & \text{if } P_{single}(t_i) \text{ not available} \end{cases} \tag{3}$$

$$P_{first}(w_i) = \begin{cases} P_{first}(t_i) & \text{if } P_{first}(t_i) \text{ available} \\ P_{first}(c_i) & \text{if } P_{first}(t_i) \text{ not available} \end{cases} \tag{4}$$

$$P_{last}(w_i) = \begin{cases} P_{last}(t_i) & \text{if } P_{last}(t_i) \text{ available} \\ P_{last}(c_i) & \text{if } P_{last}(t_i) \text{ not available} \end{cases} \tag{5}$$

Here, $t_i$ denotes token type of word $w_i$, and $c_i$ denotes hierarchical class of word $w_i$. There are two alternatives for substituting the unigram probabilities: either type probabilities or class probabilities. On the other hand, there are eight alternatives for substituting the Bigram probabilities. Eq. (6) describes how our model assigns the Bigram probabilities under different conditions in the absence of the token Bigram probabilities.

$$\begin{aligned}
&\text{If } (P_{in}(t_i \mid w_{i-1}) \text{ is available}) && P_{in}(W_i \mid w_{i-1}) = P_{in}(t_i \mid w_{i-1}) \\
&\text{else If } (P_{in}(w_i \mid t_{i-1}) \text{ is available}) && P_{in}(w_i \mid w_{i-1}) = P_{in}(w_i \mid t_{i-1}) \\
&\text{else If } (P_{in}(c_i \mid w_{i-1}) \text{ is available}) && P_{in}(w_i \mid w_{i-1}) = P_{in}(c_i \mid w_{i-1}) \\
&\text{else If } (P_{in}(w_i \mid c_{i-1}) \text{ is available}) && P_{in}(w_i \mid w_{i-1}) = P_{in}(w_i \mid c_{i-1}) \\
&\text{else If } (P_{in}(t_i \mid t_{i-1}) \text{ is available}) && P_{in}(w_i \mid w_{i-1}) = P_{in}(t_i \mid t_{i-1}) \\
&\text{else If } (P_{in}(c_i \mid t_{i-1}) \text{ is available}) && P_{in}(w_i \mid w_{i-1}) = P_{in}(c_i \mid t_{i-1}) \\
&\text{else If } (P_{in}(t_i \mid c_{i-1}) \text{ is available}) && P_{in}(w_i \mid w_{i-1}) = P_{in}(t_i \mid c_{i-1}) \\
&\text{else} && P_{in}(w_i \mid w_{i-1}) = P_{in}(c_i \mid c_{i-1})
\end{aligned} \tag{6}$$

After learning the necessary model parameters, a probability estimate is produced for every possible fragment in the test data. We use sliding window technique to determine the fragments and Eq. (2) to calculate the fragment probabilities. A fragment in a sliding window is a sequence of words starting from the first word of the sliding window. Because the sliding window size parameter determines the set of candidate fragments, it has also affect on system's ability to identify protein names. In our experiments, we use the maximum protein name length in the training set as the sliding window size. Fragments whose probability estimates exceeding two threshold values are considered as possible protein names. The first threshold value ($T_1$) is used to eliminate the weak estimates. After selecting a candidate fragment available in a sliding window, our algorithm compares its probability estimate with this threshold value which is the same value for all fragments. If it passes the first threshold, it is compared with the

[1] Single protein names are not counted.

second threshold value. The second threshold value ($T_2$) is used to remove the length bias of selection. Because of the nature of Eq. (2), long fragments are more disadvantageous than short ones. We attempt to avoid this bias by adding a length (number of tokens) parameter to the equation $T_2 = \tau^{\ell}$, where $T_2$ is calculated for a candidate fragment. In the equation, $\ell$ denotes the candidate fragment length, and $\tau$ denotes a system set parameter for the cutoff value. The value of $\tau$ is greater than $T_1$, and this means that the shorter candidate fragments must satisfy more strict restrictions induced by $T_2$. If there are candidate fragments whose probability estimates exceed both of the thresholds, the longest one is selected as a protein name from the sliding window. Conversely, if none of the fragments exceeds the threshold values, the algorithm does not extract any protein name from that sliding window.

Both $T_1$ and $\tau$ parameters have influence on the performance of the extraction task. In order to obtain the optimum values for these parameters, a hold-out set is used. After the unigram and Bigram probabilities are learnt from the training set, the optimum values for $T_1$ and $\tau$ are acquired using the hold-out set. The optimum values for $T_1$ and $\tau$ are the values that produces maximum *F*-score for the hold-out set.

An example case is provided in Fig. 1 in order to make the concept clearer. In the figure, an excerpt from the YAPEX corpora [13], and the possible fragment probabilities for the window starting from token "*AIRK2*" are shown. After calculating the fragment probabilities, our algorithm starts to compare each fragment probability to the threshold values. For instance, $P_{prot}$("*AIRK2*") is compared to $T_1$(= 0.0005) and $T_2$(= $\tau$ = 0.09) values. On the other hand, $P_{prot}$("*AIRK2*", "*kinase*") is compared to $T_1$(= 0.0005) and $T_2$(= $\tau^2$ = 0.0081) values.

### 3.3. Learning rules for protein name extraction

In our second method, we developed an automatic rule learning algorithm. The success of the rule learning depends on how well it recognizes the regularities among the target elements to be extracted. One of the main performance measures in information extraction is "recall" which is the percentage of correct names that are found. Because recall is directly related to the learner's generalization ability, rule generalization is one of the critical functions in the system.

Our rule generalization method is based on specific generalization of strings as described in [37]. In order to generalize two strings, a unique match sequence of those strings is obtained, and the differences in the unique match sequence are replaced by variables to get a generalized form of the strings. A unique match sequence can be described as a sequence of similarities (substrings occurring in both strings) and differences (substrings differing between strings) between two strings. A unique match sequence can occur either once or not at all for any given two strings. To meet these criteria, the notion of unique match sequence has two necessary conditions: (1) a symbol cannot occur in any difference, if it occurs in a similarity, and (2) a symbol cannot occur in the second constituent of any difference if the same symbol is found in the first constituent of a difference. The examples provided below will clarify the unique match sequence concept.

- UMS(ε, ε) → SIM(ε)
- UMS(ab,ab) → SIM(ab)
- UMS(bc, ef) → DIFF(bc, ef)
- UMS(abcb, dbebf) → DIFF (a, d) SIM(b) DIFF (c, e) SIM(b) DIFF (ε, f)
- UMS(abb, cdb) → Ø
- UMS(ab, ba) → Ø

As evident from the examples, the unique match sequence of two empty strings is a sequence of a single similarity which is an empty string. Moreover, the unique match sequence of two identical strings is a sequence of a single similarity which is equal to that string. The unique match sequence of two totally different strings is a sequence of a single difference.

In this work, we propose the use of hierarchical syntactic token types in the generalization process of protein names using the unique match sequence concept. According to our rule generalization method, the learner first generates the unique match sequence of two given protein names. The differences in a unique match sequence are generalized using the hierarchical syntactic token types described above. A difference in a unique match sequence can be replaced with one of the following four variables by our generalization method.

- *Type variable:* A difference in a unique match sequence is generalized as a *type variable*, if the two constituents of the difference are the same token type. A *type variable* is the name of a syntactic token type. For example, the difference *DIFF*(*1,3*) is generalized as the type variable *Number* since the token types of 1 and 3 are numbers. A type variable represents all tokens of that type, and it can be replaced with any token of that type.
- *Class variable:* If the constituents of a difference are not the same token type but the same token class, the difference is generalized as a *class variable*. A *class variable* is the name of a syntactic token class. For example, the difference *DIFF*(*1,B*) is generalized as the class variable *Single* since 1 and *B* are not the same type but they are in *Single* class. A class variable represents all the tokens in that class.
- *Optional variable:* A difference in a unique match sequence is replaced with an optional variable, if one of the constituents in the difference is empty string. For example, the difference *DIFF*(*B*,ε) is generalized as *Opt*(*B*) variable which represents *B* or empty string.
- *AnyToken variable:* This operation replaces a difference in a unique match sequence with an unrestricted variable *AnyToken*, if any of the above conditions are not satisfied. Unrestricted variables are used to represent any token.

In order to make the rule generalization concept clearer, an example rule generation is given in Fig. 2. In the example, a generalized rule is learnt from two protein name instances: "*presenilin-1*" and "*galectin-3*". First, the unique match sequence of these two

*… and AIRK2 kinase bind one another…*

**Possible Fragment Probabilities**
**(*Sliding-Window Size = 4, $T_1$ = 0.0005, $\tau$ = 0.09*):**
- $P_{prot}$("AIRK2") = Psingle("AIRK2")
- $P_{prot}$("AIRK2", "kinase") = $P_{first}$("AIRK2") $P_{in}$("kinase" | "AIRK2") $P_{last}$("kinase")
- $P_{prot}$("AIRK2" , "kinase" , " bind") = $P_{first}$("AIRK2") $P_{in}$("kinase" | "AIRK2") $P_{in}$("bind" | "kinase") $P_{last}$("bind")
- $P_{prot}$("AIRK2" , "kinase" , "bind" , "one") = $P_{first}$("AIRK2") $P_{in}$("kinase" | "AIRK2") $P_{in}$("bind" | " kinase") $P_{in}$("one" | " bind") $P_{last}$("one")

**Fig. 1.** An example case for filtering.

protein names is found. In order to obtain the generalized rule, our learner starts to perform the generalization operations on the differences in the order of their specificity. Because the constituents of the first difference are of the same token type, the learner replaces the difference element in the unique match sequence with the token type variable of type *Frequent Type-2*. After generalizing the first difference, the learner substitutes the second difference element in the unique match sequence with the token type variable *Number*. In the example, the generalized rule states that a token with type *Frequent Type-2* followed by the token "–" and a token with type *Number* forms a protein name. In the example, some of the new protein names recognizable by the generalized rule are also shown: "*caveolin-1*", "*thrombospondin-1*", "*interleukin-3*", and "*interleukin-12*".

We validate each generalized rule on the training set in order to give their confidence factors. At the end of the confidence evaluation, each rule is assigned a confidence factor. The confidence factor of a given rule shows how well the rule classifies positive and negative instances. Our main resource for assigning a confidence factor to a rule is the result of applying that rule to the training set. Rule confidence factor is the degree, which the rule gives the correct result across individual instances (both positive and negative). In other words, the confidence factor of a rule is the percentage of correctly extracted names as a result of applying that rule to the training set. If the confidence factor of the generalized rule exceeds a certain threshold value, the learner puts the generalized rule into the final rule-set. Otherwise, the rule learnt by generalization is discarded. Furthermore, confidence factors are also used to determine the rule priorities. The generated rules are ordered according to their confidence factors and applied in that order during the test.

In order to make a full use of the training data and improve the algorithm's extraction performance by further rule refinement, positive examples (i.e. protein names) that are uncovered by the learnt rule-set are added to the rule-set and each rule in the rule-set is associated with a set of exceptions. Normally, the resulting rule-set at the end of the rule generation step covers all of the positive instances in the training data. However, there may be some positive instances in the training data that are uncovered by the resulting rule-set after rule filtering step because the rules below the threshold are eliminated after this step. The problem in this particular case is that the extractor would miss a protein name in the test data, even if the missed protein name occurs in the training data, because the final rule-set generated at the end of the training does not cover that protein name. We address this issue by adding the uncovered positive examples to the final rule-set. The second problem is that of efficient utilization of the negative examples (i.e. non-protein names) in the training data, though they are used in confidence factor calculation. If it is not a 100% confident rule, a rule in the final rule-set may cover some negative instances in the training data. This leads to recognition of an incorrect protein name during the test, even if that name is marked as a non-protein name in the training data. This issue is solved by associating each rule in the final rule-set with a set of exceptions. During confidence factor calculation, every negative instance recognized by the candidate rule is put into that rule's exception set. If any of the names in a rule's exception set are recognized by the test, the recognized names are just ignored and not extracted.

- ***Seed Protein Name Instances:***
  *"presenilin – 1", "galectin – 3"*
- ***Unique Match Sequence:*** DIFF*(presenilin, galectin)* SIM*(-)* DIFF*(1,3)*
- ***Generalize:*** DIFF*(presenilin, galectin)* ➔ *Frequent Type-2*
- ***Generalize:*** DIFF*(1, 3)* ➔ *Number*
- ***Generalized Rule:*** Frequent Type-2 SIM*(-)* Number
- ***Recognizable Protein Names:***
  *"caveolin – 1", "thrombospondin – 1", "interleukin – 3", "interleukin – 12"*

**Fig. 2.** An example rule generation.

## 4. Experimental evaluation

### 4.1. Corpora and methodology

In order to evaluate the performance and the behavior of the proposed methods under different conditions, we conducted a set of experiments. The main objective of the experimentation is to analyze the performance and behavior of our methods on different datasets, and with different threshold values. We also compare our methods to several other studies. Moreover, we investigated the impact of using hierarchical token types and the other novelty we proposed on the extraction process.

We conducted our experiments on two different datasets. The first one, the YAPEX corpora [13], contains two mutually exclusive collections of biological abstracts. The training (reference) collection contains 1745 annotated protein names, and the total number of the annotated protein names in the test set is 1966. We recorded performance scores on this dataset both with and without cross validation for comparison purposes. The second dataset used for evaluation of the proposed methods is the GENIA corpus [11] which contains 2000 labeled abstracts. We performed 10-fold cross validation on the GENIA dataset to evaluate the developed methods.

We measured precision, recall, and *F*-score; as is commonly done in the Message Understanding Conference (MUC) evaluations. Precision is the fraction of correct outcomes divided by the number of all outcomes. For instance, precision value for the protein name extraction task is the percentage of extracted protein names that are correct. On the other hand, recall is analogous to sensitivity in binary classification. Recall can be defined as the fraction of correct outcomes divided by the total number of possible correct answers. The *F*-score, harmonic mean of precision and recall, provides a method for combining precision and recall scores into a single value.

Franzen et al. [13] defines several notions of correctness. The most common two are "strict mode" and "sloppy mode". In strict mode, two protein names are not considered a match unless they consist of the same character sequence in the same position in the text. In the latter approach, partial matches are also considered correct. In order not to over-estimate the performance, we used the strict mode which is the most conservative approach to determine the truth-value of the matching in our experiments.

### 4.2. Results and discussion

#### 4.2.1. Comparison of methods

Table 2 shows the quantitative results of the experiments performed. Our Bigram method reached $F = 67.7\%$ on the YAPEX dataset and $F = 66.8\%$ on the GENIA dataset without relying on hand-crafted rules, dictionaries, and deep language processing. With a lower *F*-score performance, the developed rule learning algorithm reached $F = 61.8\%$ on the YAPEX dataset and $F = 61.0\%$ on the GENIA dataset. Compared to the YAPEX system, both of the developed methods produced better results in terms of *F*-score.

Tsuruoka and Tsujii [8,9] reported $F = 66.6\%$ on GENIA corpus by using approximate string searching techniques for discovery, a probabilistic variant generator for dictionary expansion and a naive Bayes classifier for filtering. Using less resource, Bigram produced better results on the same dataset. Krauthammer et al. [6] reported $F = 75\%$ (considering partial matches as correct) on a set

**Table 2**
Quantitative results of the experiments performed.

| | Corpus | Cross validation | Precision (%) | Recall (%) | *F*-score (%) |
|---|---|---|---|---|---|
| Bigram | YAPEX | Yes | 67.5 | 67.9 | 67.7 |
| Bigram | YAPEX | No | 63.3 | 71.8 | 67.3 |
| Bigram | GENIA | Yes | 61.4 | 73.2 | 66.8 |
| Rule | YAPEX | Yes | 64.4 | 59.4 | 61.8 |
| Rule | YAPEX | No | 63.1 | 60.1 | 61.6 |
| Rule | GENIA | Yes | 60.0 | 62.1 | 61.0 |
| YAPEX | YAPEX | No | 62.0 | 59.9 | 61.0 |

"Bigram" denotes our first method based on Bigram language model, and "Rule" denotes the developed rule learning method. The last row contains the performance values for the YAPEX project and cited from the project homepage (Protein halt i text, http://www.sics.se/humle/project/prothalt). The performance scores on the YAPEX dataset are recorded both with and without 10-fold cross validation, whereas the performance scores on the GENIA dataset are recorded only with 10-fold cross validation. We use the standard formula for precision, recall, and *F*-score calculation: precision = (true positives)/(true positives + false positives); recall = (true positives)/(true positives + false negatives); *F*-score = (2 ∗ precision ∗ recall)/(precision + recall).

of two papers. When only exact matches were considered as correct, their system achieved $F = 59.8\%$. Moreover, the reported performance scores are based on a very small dataset. In terms of the amount of the resources used the most similar previous work is the method developed by Seki and Mostafa [31]. They reported $F = 63.3\%$ on the YAPEX dataset by employing a probabilistic model together with the surface clues specified for identifying protein names with an emphasis on finding name boundaries. Though they used a number of basic heuristic rules, their method does not rely on dictionaries, part-of-speech (POS) taggers and/or syntactic parsers. The two other methods that we can compare on the same datasets are SemiCRF and DictHMM, two learning methods for protein name recognition proposed by Kou et al. [27]. Using dictionaries with CRF-like learning methods and additionally utilizing POS tags, SemiCRF achieved $F = 66.1\%$ on the YAPEX dataset and $F = 72.3\%$ on the GENIA dataset. While SemiCRF produced better results on the GENIA dataset, Bigram achieved better performance on the YAPEX dataset with respect to *F*-score. Moreover, Bigram's high recall performance is noteworthy. On both datasets, Bigram has higher recall score. Our rule learning algorithm, on the other hand, produced lower scores on both datasets. Their novel system, DictHMM, reached $F = 51\%$ on the YAPEX dataset and $F = 54.7\%$ on the GENIA dataset. DictHMM combines a dictionary with an HMM to perform name matching. Compared to DictHMM, the developed Bigram and rule learning methods produced better results in terms of *F*-score. Though DictHMM has low *F*-score performance, it emphasizes recall over precision. However, Bigram has a higher recall score on both datasets. The best published result on the YAPEX dataset belongs to the NLProt system [28]. NLProt reached $F = 75\%$ on the YAPEX dataset and $F = 71\%$ on the GENIA dataset. NLProt combines a pre-processing dictionary and rule-based filtering step with several separately trained support vector machines. The utilization of several dictionaries (dictionary of protein names, dictionary of chemical compounds, etc.) and hand-tailored filtering rules has an impact on the performance difference between NLProt and our methods. According to their results, leaving out only the dictionary yields 3% loss on NLProt's *F*-score.

Exploiting only hierarchical token types, our Bigram method achieved a better performance than many other techniques with respect to *F*-score. The primary advantage of using less resource is the decrease in the processing overhead and, therefore, faster extraction speed, which is especially important for the extraction of protein names from large number of biomedical documents. Furthermore, the method does not suffer from the drawbacks typical for dictionary-based systems and systems using hand-crafted rules. For instance, there is no need for regular dictionary updates. The methods adaptability to the other bio-entity extraction tasks is also easier for not using hand-crafted rules.

The developed rule learning method is also effective for protein name extraction and achieved better performance than some of the previous work. However, its performance is behind the developed Bigram method in terms of *F*-score. Particularly, the difference in the recall rates is very obvious, though both methods use the same concept for generalization. This is due to the small number of variables defined for generalization. The granularity level of the learnt rules affects the performance of the rule learning systems. While over-specific rules may cause low recall, over-general rules cause low precision. This issue could be addressed by adding more comprehensive variables. The increase in the number and the flexibility of the variables used for the rule construction would improve the expressiveness of the rules, and thus result in better performance.

#### 4.2.2. Hierarchy usage

We investigated the effects of using hierarchical token types for generalization by testing the Bigram method on the YAPEX dataset with different token type sets. The performance values obtained for different token type sets are shown in Table 3. The first set contains 11 base token types (Single Letter, Number, Roman Numeral, Greek Letter, Short Abbreviation, Long Abbreviation, Delimiter, Regular-Frequent Type, Regular-Lowercase, Regular-Proper, Other) from our original set and does not have a hierarchy. The second set contains five core classes in addition to the same token types in the first set and has a hierarchical relation between them as in our original set. The third set contains all 21 token types available in the original set but does not have a hierarchy; and the last set is the original set with a hierarchy. As seen from Table 3, hierarchy usage has significantly contributed to the recall rate. The *F*-score value improved from 21.2% to 62.3% on the sets that consist of base token types. Moreover, we observed even more performance improvement on the sets that consists of all token types introduced in Section 3.1. The obtained improvement is 45.2% in the *F*-score values. Another notable impact is the relatively small decrease in the precision rates. The decrease is inversely proportional to the broadness of the token type set. It is more considerable when the number of token types is small. For instance, the precision decrease is only 1.4% when we add a hierarchy level to original type set. On the other hand, the decrease in the precision becomes more significant for the base type set. Overall, the results show that hierarchy usage improved the performance of the extraction process.

#### 4.2.3. Smoothing

We employed two novel ideas to achieve better performance in the Bigram method: (1) modification of the standard Bigram calculation method, and (2) the use of hierarchically categorized

**Table 3**
Performence values for Bigram method with different token type sets ($T_1 - 0.0005$, $\tau = 0.09$).

| Set | Number of token/class types | Hierarchy | Precision (%) | Recall (%) | *F*-score (%) |
|---|---|---|---|---|---|
| Base types | 11 Token types | No | 63.8 | 12.7 | 21.2 |
| Base types with hierarchy | 11 Token types + 5 token classes | Yes | 56.8 | 68.9 | 62.3 |
| All types | 21 Token types | No | 64.7 | 13.3 | 22.1 |
| All types with hierarchy | 21 Token types + 5 token classes | Yes | 63.3 | 71.8 | 67.3 |

**Table 4**
Performence values for Bigram method with different calculation methods and different smoothing schemes ($T_1 - 0.0005$, $\tau = 0.09$).

| Bigram calculation method | Smoothing | Precision (%) | Recall (%) | *F*-score (%) |
|---|---|---|---|---|
| Eq. (1) | No smoothing | 17.3 | 15.3 | 16.2 |
| Eq. (1) | Good-turing | 16.9 | 16.5 | 16.7 |
| Eq. (1) | Hierarchical token types | 31.8 | 48.8 | 38.5 |
| Eq. (2) | No smoothing | 64.7 | 13.3 | 22.1 |
| Eq. (2) | Good-turing | 62.7 | 15.7 | 25.1 |
| Eq. (2) | Hierarchical token types | 63.3 | 71.8 | 67.3 |

The first three rows contain the performance values obtained by using standard Bigram calculation method described in Eq. (1) and the last three shows the performance values obtained by using modified calculation method described in Eq. (2). We investigated the schemes where no smoothing performed on the observed probabilities ("No smoothing"), good-turing [38] method employed for probability estimation ("Good-Turing"), and hierarchically categorized syntactic token types used ("Hierarchical Token Types").

syntactic token types to overcome the data sparseness problem. In order to evaluate the contribution of the former, we compared our modified version to the standard Bigram calculation method described in Eq. (1). Moreover, we performed comparative experiments on different smoothing schemes to see the effect of hierarchy usage to remedy the data sparseness problem. Table 4 shows the results of the comparison. Our smoothing scheme based on the hierarchical token types outperformed the others. Moreover, the modified Bigram version proposed in this paper obviously improve over the standard version.

#### 4.2.4. Threshold factor

The threshold factors also have an impact on the performance of our extraction methods. The Bigram method automatically finds threshold values ($T_1$ and $\tau$) by testing the obtained model on a hold-out set. However, it is also possible to manually set these values and adjust the performance tradeoff between precision and recall. Fig. 3 shows the performance of the Bigram method on the YAPEX dataset as the threshold parameter "$\tau$" changes and "$T_1$" stays constant. $T_1$ value is set to 0.0 in Fig. 3(a), while it is set to 0.0005 in Fig. 3(b). As expected, the recall value decreases with the increase in threshold in both graphs. In contrast, the precision value is directly proportional with the threshold value; the precision increases parallel to the increase in the threshold value. We observe a drastic change in the performance between $\tau = 0.0$ and $\tau = 0.1$ values in Fig. 3(a). Normally, one would expect a high recall value and a low precision value where $\tau$ value is set to 0.0. However, both precision and recall scores are very low at $\tau = 0.0$ in Fig. 3(a). This is due to the fact that the longest fragment, among the all candidate fragments whose probability estimate exceed the threshold value, is selected as a protein name. Because, $T_1$ is constantly set to 0.0 in Fig. 3(a), the longest fragments in the sliding windows were selected as protein names without any restriction where $\tau = 0.0$, which caused a lot of errors. In a different scheme, Fig. 3(b), where $\tau$ is set to a non-zero value, the extractor exhibits the normal behavior (high recall, low precision at $\tau = 0.0$).

Fig. 4 shows the performance of the Bigram method on the YAPEX dataset as the threshold parameter "$T_1$" changes and "$\tau$" stays constant. $\tau$ value is set to 0.0 in Fig. 4 (a) and 0.09 in Fig. 4(b). A similar trend as in Fig. 3 is also observed in this figure: the recall value decreases with the increase in threshold and the precision increases parallel to the increase in the threshold value. Another similarity is the very low precision and recall scores obtained at $T_1 = 0.0$ in Fig. 4(a). Moreover, we observe a drastic change in the performance between $T_1 = 0.0$ and $T_1 = 0.0005$ values in Fig. 4(a). The reason for such behavior is the same as discussed in Fig. 3.

On the other hand, automatic rule learning method has a user-set threshold parameter. Fig. 5 shows the performance of automatic rule learning method on the YAPEX dataset as the threshold parameter changes. The optimum value for the threshold parameter is found to be 0.48, where the acquired *F*-score value is maximized, through experimentation. We observe the same trend in Fig. 5; the recall value decreases and the precision value increases with the increase in threshold. Another notable observation is the drop in the recall rate where the threshold parameter is 0.49. This behavior is caused by the elimination of a general rule whose true positive (TP) returns are more than its false positive (FP) returns. More importantly, the number of TPs recognized by this rule has a prominent share in the total number of TPs recognized by the rule-set. Therefore the elimination of this rule leads to a considerable decrease in the total number of TPs, which also causes the drop in the recall rate. This situation is a result of the small number of variables defined for generalization. The transition would be smoother if the learnt rules were fine-grained enough.

## 5. Conclusions and future work

In this paper, we present two different learning approaches for identifying protein names in biomedical texts. Our first method follows a probabilistic approach and uses Bigram language model. The second method is an automatic rule learning method for protein name extraction. This method aims to learn rules automatically to recognize the protein name patterns and generalize these patterns by processing similarities and differences between them.

Compared to previous work, the proposed methods exploit a two-level hierarchy of useful syntactic token types for representing the protein names instead of using a simple list of token types. In

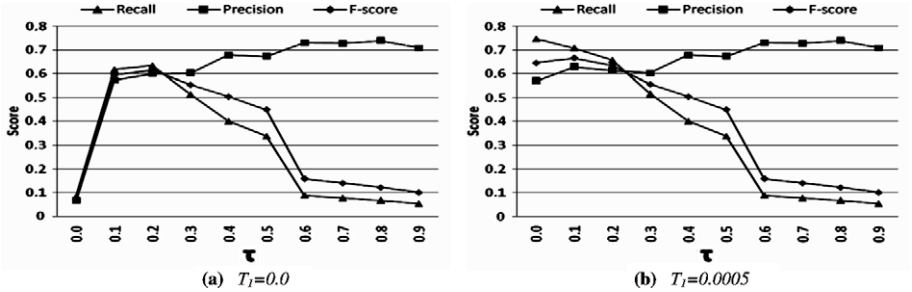


**Fig. 3.** The performance of Bigram method on the YAPEX dataset as the threshold parameter "$\tau$" changes.

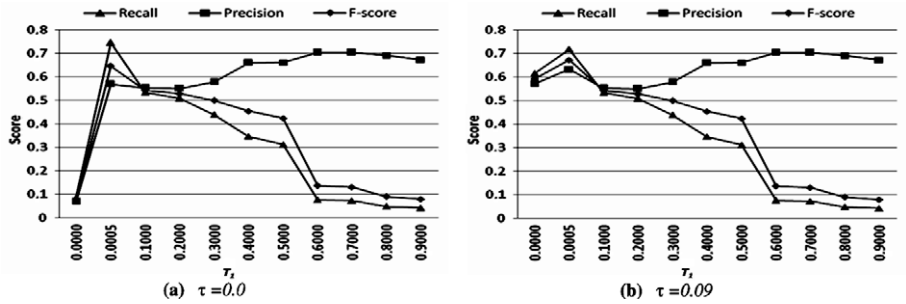


**Fig. 4.** The performance of Bigram method on the YAPEX dataset as the threshold parameter "$T_1$" changes.

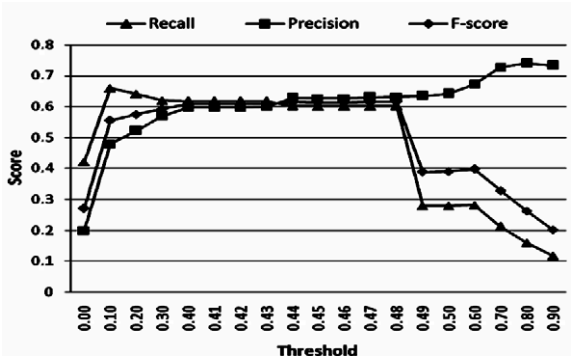


**Fig. 5.** The performance of automatic rule learning method on the YAPEX dataset as the threshold parameter changes.

the first approach, we use the hierarchy to generalize the tokens and reduce the influence of the data sparseness problem. In the latter one, we use the hierarchy for rule generalization. The results show that using a hierarchy improved the performance of the extraction process and the impact of the hierarchy usage is directly proportional to the broadness of the token type set.

We performed several experiments on different datasets to evaluate the performance of our methods. We compared the performance of the proposed approaches with previous methods. The comparison results indicate that our suggested methods can be used for protein name identification task effectively. Avoiding deep syntactic/semantic analysis, the proposed methods reduce processing overhead. They also achieve adaptability which is another major advantage. Thus, the proposed techniques can be applied to other bio-entities such as gene names.

Our future work will be carried out in several directions. First, we believe that extending the token type set would provide further increase in the generalization capability, since the results show that using hierarchically categorized syntactic word types positively affects the extraction performance. Moreover, defining new generalization variables would have a positive impact on the generalization performance of our rule learning method. Currently, we generate the base rules and generalize them just one step and scan the entire rule space. Adding some more steps to the generalization process and employing a search mechanism during rule generation would improve the efficiency of rule construction.

## References

[1] Zhou D, He Y. Extracting interactions between proteins from the literature. J Biomed Inform 2008;41(2):393–407.
[2] Tanabe L, Wilbur WJ. Tagging gene and protein names in biomedical text. Bioinformatics 2002;18(8):1124–32.
[3] Zhou G, Zhang J, Su J, Shen D, Tan C. Recognizing names in biomedical texts: a machine learning approach. Bioinformatics 2004;20(7):1178–90.
[4] Grishman R. Information extraction: techniques and challenges. In: Pazienza MT, editor. International summer school on information extraction: a multidisciplinary approach to an emerging information technology, lecture notes in computer science, vol. 1299. London: Springer-Verlag; 1997. p. 10–27.
[5] Krauthammer M, Nenadic G. Term identification in the biomedical literature. J Biomed Inform 2004;37(6):512–26.
[6] Krauthammer M, Rzhetsky A, Morozov P, Friedman C. Using BLAST for identifying gene and protein names in journal articles. Gene 2000;259(1–2):245–52.
[7] Altschul S, Madden T, Schaffer A, Zhang J, Zhang Z, Miller W, et al. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res 1997;25(17):3389–33402.
[8] Tsuruoka Y, Tsujii J. Improving the performance of dictionary-based approaches in protein name recognition. J Biomed Inform 2004;37(6):461–70.
[9] Tsuruoka Y, Tsujii J. Boosting precision and recall of dictionary-based protein name recognition. In: Proceedings of the ACL 2003 workshop on natural language processing in biomedicine, vol. 13. Association for Computational Linguistics; 2003. p. 41–8.
[10] Schuemie MJ, Mons B, Weeber M, Kors JA. Evaluation of techniques for increasing recall in a dictionary approach to gene and protein name identification. J Biomed Inform 2007;40(3):316–24.
[11] Kim J-D, Ohta T, Tateisi Y, Tsujii J. Genia corpus: a semantically annotated corpus for bio-textmining. Bioinformatics 2003;19(Suppl. 1):180–2.
[12] Fukuda K, Tsunoda T, Tamura A, Takagi T. Toward information extraction: identifying protein names from biological papers. Pac Symp Biocomput 1998:707–18.
[13] Franzen K, Eriksson G, Olsson F, Asker L, Liden P, Cöster J. Protein names and how to find them. Int J Med Inform 2002;67(1–3):49–61.
[14] Demetriou G, Gaizauskas R, Artymiuk P, Willett P. Protein structures and information extraction from biological texts: the PASTA system. Bioinformatics 2003;19(1):135–43.
[15] Humphreys K, Demetriou G, Gaizauskas R. Two applications of information extraction to biological science journal articles: enzyme interactions and protein structures. Pac Symp Biocomput 2000:502–13.
[16] Seki K, Mostafa J. An approach to protein name extraction using heuristics and a dictionary. Am Soc Inf Sci Technol Annu Conf 2003:71–7.
[17] Bunescu R, Ge R, Kate RJ, Marcotte EM, Mooney RJ, Ramani AK, Wong YW. Comparative experiments on learning information extractors for proteins and their interactions. Artif Intell Med 2005;33(2):139–55.
[18] Califf ME, Mooney RJ. Relational learning of pattern-match rules for information extraction. In: Proceedings of the sixteenth national conference on artificial intelligence and the eleventh innovative applications of artificial intelligence conference innovative applications of artificial intelligence. American Association for Artificial Intelligence; 1999. p. 328–34.
[19] Freitag D, Kushmerick N. Boosted wrapper induction, In: Proceedings of the seventeenth national conference on artificial intelligence and twelfth conference on innovative applications of artificial intelligence. AAAI Press/The MIT Press; 2000. p. 577–83.
[20] Duda RO, Hart PE. Pattern classification and scene analysis. New York: John Wiley and Sons; 1973.
[21] Brill E. Transformation-based error-driven learning and natural language processing: a case study in part-of-speech tagging. Comput Linguist 1995;21(4):543–65.
[22] Vapnik VN. The nature of statistical learning theory. Berlin: Springer; 1995.
[23] Berger AL, Della Pietra SA, Della Pietra VJ. A maximum entropy approach to natural language processing. Comput Linguist 1996;22(1):39–71.
[24] Collier N, Nobata C, Tsujii J. Extracting the names of genes and gene products with a Hidden Markov Model. In: Proceedings of the 18th international conference on computational linguistics, vol. 1. Association for Computational Linguistics; 2000. p. 201–7.
[25] Kazama J, Makino T, Ohta Y, Tsujii J. Tuning support vector machines for biomedical named entity recognition. In: Proceedings of the Acl-02 workshop on natural language processing in the biomedical domain, vol.3. Association for Computational Linguistics; 2002. p. 1–8.
[26] Takeuchi K, Collier N. Bio-medical entity extraction using support vector machines. Artif Intell Med 2005;33(2):125–37.

[27] Kou Z, Cohen WW, Murphy RF. High-recall protein entity recognition using a dictionary. Bioinformatics 2005;21(1):266–73.
[28] Mika S, Rost B. Protein names precisely peeled off free text. Bioinformatics 2004;20(1):241–7.
[29] Hou W, Chen H. Enhancing performance of protein and gene name recognizers with filtering and integration strategies. J Biomed Inform 2004;37(6):448–60.
[30] Ray S, Craven M. Representing sentence structure in hidden Markov models for information extraction. In: Proceedings of the 17th international joint conference on artificial intelligence. Morgan Kaufmann; 2001. p. 1273–79.
[31] Seki K, Mostafa J. A Probabilistic Model for Identifying Protein Names and their Name Boundaries. In: Proceedings of the IEEE computer society conference on bioinformatics. IEEE Computer Society; 2003. p. 251–58.
[32] Yu H, Hatzivassiloglou V, Rzhetsky A, Wilbur WJ. Automatically identifying gene/protein terms in MEDLINE abstracts. J Biomed Inform 2002;35(5–6): 322–30.
[33] Seki K, Mostafa J. A hybrid approach to protein name identification in biomedical texts. Inform Proc Manag 2005;41(4):723–43.
[34] Yamamoto K, Kudo T, Konagaya A, Matsumoto Y. Protein name tagging for biomedical annotation in text. In: Proceedings of the ACL 2003 workshop on natural language processing in biomedicine, vol. 13. Association for Computational Linguistics; 2003. p. 65–72.
[35] Brill E. Some advances in transformation-based part of speech tagging. In: Proceedings of the twelfth national conference on artificial intelligence, vol. 1. American Association for Artificial Intelligence; 1994. p. 722–27.
[36] Shannon CE. A mathematical theory of communication. Bell Sys Tech J 1948;27:379–423 [and 623–56].
[37] Cicekli I, Cicekli NK. Generalizing predicates with string arguments. Appl Intell 2006;25(1):23–36.
[38] Gale W, Sampson G. Good-turing smoothing without tears. J Quant Linguist 1995;2:217–37.