Improving Performance of Dictionary-Based Approaches for Protein Name Recognition

Yoshimasa Tsuruoka and Jun'ichi Tsujii

Department of Computer Science, University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo 113-0033 Japan
tsuruoka@is.s.u-tokyo.ac.jp, tsujii@is.s.u-tokyo.ac.jp
Phone : +81-3-5803-1697
FAX : +81-3-5802-8872

Preprint submitted to Elsevier Science *12 January 2004*

Abstract

Dictionary-based protein name recognition is the first step for practical information extraction from biomedical documents because it can provides ID information of recognized terms. However, dictionary based approaches have two fundamental difficulties: (1) false recognitions mainly caused by short names; (2) low recall due to spelling variation. In this paper, we tackle the former problem by using a machine learning method to filter out false positives. As for the latter, we present two methods for alleviating the problem of spelling variation. One is to use an approximate string searching method, and the other is to expand the dictionary by using the probabilistic variant generator which we propose in this paper. Experimental results using the GENIA corpus show that the filtering using a naive Bayes classifier greatly improves precision with slight loss of recall. The combination of the filtering and the dictionary expansion by the variant generator achieved an F-measure of 67%.

Key words: protein name recognition, naive Bayes classifier, approximate string search, spelling variation

1 Introduction

The rapid increase of machine readable biomedical texts (e.g. MEDLINE) makes automatic information extraction from those texts much more attractive. Especially extracting information of protein-protein interactions from MEDLINE abstracts is one of the most important tasks today (1; 2; 3).

In order to extract information of proteins from a text, one has to first recognize protein names appearing in the text. This kind of problem has been extensively studied in the field of natural language processing as named entity recognition tasks. The most popular approach is to train the recognizer on an annotated corpus by using a machine learning algorithm, such as Hidden Markov Models, support vector machines (SVMs) (4), and maximum entropy models (5). In the machine learning framework, the task of the classifier is to determine the text regions corresponding to protein names.

Ohta et al. provided the GENIA corpus (6), an annotated corpus of MEDLINE abstracts, which can be used as a gold-standard for evaluating and training named entity recognition algorithms. The corpus has fostered research on machine learning techniques for recognizing biological entities in texts (7; 8; 9).

However, the drawback of those machine learning approaches is that they do not provide ID information of recognized terms. For the purpose of information extraction of proteins, ID information of recognized proteins, such as GenBank [1] ID or SwissProt [2] ID, is indispensable to integrate extracted information with relevant data in other information sources.

On the other hand, dictionary-based approaches, which we present in this paper, intrinsically provide ID information because they recognize a term by searching the most similar (or identical) one in the dictionary to the target region. This advantage makes dictionary-based approaches particularly useful as the first step for practical information extraction from biomedical documents (3).

Dictionary-based approaches, however, have two fundamental difficulties. One is a large number of false positives mainly caused by short names, which significantly degrade overall precision. Although this problem can be avoided by excluding short names from the dictionary, such a solution makes it impossible to recognize short protein names. We tackle this problem by incorporating a machine learning technique for filtering out false positive. Each protein name candidate is checked whether it is really protein name or not by a classifier trained on an annotated corpus.

[1] GenBank is one of the largest genetic sequence databases.
[2] The Swiss-Prot is an annotated protein sequence database.

The other problem in dictionary-based approaches comes from the fact that biomedical terms have many spelling variations. For example, the protein name "NF-Kappa B" has many spelling variants such as "NF Kappa B", "NF kappa B" "NF kappaB" and "NFkappaB". Exact matching techniques regard these terms as completely different terms, which results in failing to find protein names written in varied forms.

As for the problem of spelling variation, we present two solutions in this paper. One is to use approximate string searching techniques in which the surface-level similarity of strings is considered. The other is to expand the dictionary in advance by using the *probabilistic variant generator* which we propose in this paper. To show their effectiveness, the experimental results on the GENIA corpus is presented.

This paper is organized as follows. Section 2 describes the overview of our protein name recognition method. Section 3 explains the approximate string searching algorithm for alleviating the problem of spelling variation. As an alternative solution to the problem, Section 4 illustrates the probabilistic variant generator which is used for expanding the dictionary. Section 5 describes how to filter out false recognitions by a machine learning method. Section 6 presents experimental results using the GENIA corpus. Some related work is described in Section 7. Finally, Section 8 offers some concluding remarks.

2 Method Overview

Our protein name recognition method consists of two phases. The first phase is *candidate recognition phase* and the second phase is *filtering phase*.

- Candidate recognition phase

  The task of this phase is to find protein name candidates appearing in the text using a given dictionary. The protein IDs can be associated with each candidate in this phase.

  The most straightforward way to exploit a dictionary is to use exact matching algorithms. However, as mentioned earlier, the existence of many spelling variations for the same protein name makes exact matching less effective. By exact matching algorithms, one cannot find the protein names that appear in a slightly different form from the canonical names in the dictionary.

  We propose two solutions to this problem. One is to use an approximate string searching algorithm instead of exact matching algorithms, which is presented in Section 3. The other is to expand the dictionary in advance by using the variant generator, which is presented in Section 4.
- Filtering phase

		G	R	-	2
	0	1	2	3	4
E	1	1	2	3	4
G	2	1	2	3	4
R	3	2	1	2	3
-	4	3	2	1	2
1	5	4	3	2	2

Fig. 1. Dynamic programming matrix.

One of the serious problems of dictionary-based recognition is a large number of false recognitions mainly caused by short entries in the dictionary. For example, the dictionary constructed from GenBank could contain the entry "NK". However, the word "NK" is often used as a part of the term "NK cells". In this case, "NK" is an abbreviation of "natural killer" and is not a protein name. Therefore this entry makes a large number of false recognitions leading to low precision.

One solution to this problem is to check each candidate whether it is really protein name or not. In other words, Each protein name candidate is classified into "accepted" or "rejected" by a machine learning algorithm. The classifier uses the context of the term and the term itself as the features for the classification. Only "accepted" candidates are recognized as protein names in the final output. Section 5 describes the detail of the classification algorithm in this phase.

In the following sections, we describe the details of the methods used in these phases.

3 Candidate Recognition by Approximate String Searching

One way to deal with the problem of spelling variation is to use a kind of 'elastic' matching algorithm, by which a recognition system scans a text to find a similar term to (if any) a protein name in the dictionary. We need a similarity measure to do such a task. The most popular measure of similarity between two strings is *edit distance*, which is the minimum number of operations on individual characters (e.g. substitutions, insertions, and deletions) required to transform one string of symbols into another. For example, the edit distance between "EGR-1" and "GR-2" is two, because one substitution (1 for 2) and one deletion (E) are required.

To calculate the edit distance between two strings, we can use a dynamic programming technique. Figure 1 illustrates an example. For clarity of presentation, all costs are assumed to be 1. The matrix $C_{0..|x|,0..|y|}$ is filled, where $C_{i,j}$ represents the minimum number of operations needed to match $x_{1..i}$ to $y_{1..j}$. This is computed as follows (10)

$$C_{i,0} = i \tag{1}$$
$$C_{0,j} = j \tag{2}$$
$$C_{i,j} = \text{if } (x_i = y_j) \text{ then } C_{i-1,j-1} \tag{3}$$
$$\text{else } 1 + \min(C_{i-1,j}, C_{i,j-1}, C_{i-1,j-1})$$

The calculation can be done by either a row-wise left-to-right traversal or a column-wise top-to-bottom traversal.

There are some algorithms that run faster than the dynamic programming method for computing uniform-cost edit distance, where the weight of each edit operation is constant within the same type (11). However, what we expect is that the distance between "EGR-1" and "EGR 1" will be smaller than that between "EGR-1" and "FGR-1", while the uniform-cost edit distances of them are equal.

The dynamic programming based method is flexible enough to allow us to define arbitrary costs for individual operations depending on a letter being operated. For example, we can make the cost of the substitution between a space and a hyphen much lower than that of the substitution between 'E' and 'F'. Therefore, we use the dynamic programming based method for our task.

Table 1 shows the cost function used in our experiments. Both insertion and deletion costs are 100 except for spaces and hyphens. Substitution costs for similar letters are 10. Substitution costs for the other different letters are 50.

3.1 String Searching

What we have described in the previous section is a method for calculating the similarity between two strings. However, what we need when trying to find proteins in texts is approximate string searching in which the recognizer scans a text to find a similar term to (if any) a term in the dictionary. The dynamic programming based method can be easily extended for approximate string searching.

The method is illustrated in Figure 2. In this case, the protein name to be matched is "EGR-1" and the text to be scanned is "encoded by EGR include".

Table 1
Cost function.

Operation	Letter	Cost
insertion	*a space* or *a hyphen*	10
	other letters	100
deletion	*a space* or *a hyphen*	10
	other letters	100
substitution	a letter for the same letter	0
	a numeral for a numeral	10
	a space for *a hyphen*	10
	a hyphen for *a space*	10
	a capital letter for the corresponding small letter	10
	a small letter for the corresponding capital letter	10
	other letters	50

		e	n	c	o	d	e	d		b	y		E	G	R		1		i	n	c	l	u	d	e
	0	1	2	3	4	5	6	7	0	1	2	0	1	2	3	0	1	0	1	2	3	4	5	6	7
E	1	1	2	3	4	5	6	7	1	1	2	1	0	1	2	1	1	1	1	2	3	4	5	6	7
G	2	2	2	3	4	5	6	7	2	2	2	2	1	0	1	2	2	2	2	2	3	4	5	6	7
R	3	3	3	3	4	5	6	7	3	3	3	3	2	1	0	1	2	3	3	3	3	4	5	6	7
-	4	4	4	4	4	5	6	7	4	4	4	4	3	2	1	1	2	3	4	4	4	4	5	6	7
1	5	5	5	5	5	5	6	7	5	5	5	5	4	3	2	2	1	2	3	4	5	5	5	6	7

Fig. 2. String searching using a dynamic programming matrix.

String searching can be done by just setting the elements corresponding separators (e.g. space) in the first row to zero. After filling the whole matrix, one can find that "EGR-1" can be matched to this text at the place of "EGR 1" with cost 1 by searching for the lowest value in the bottom row and then backtracing to the top row along the lowest-cost path.

To take into account the length of a term, we use a normalized cost, which is calculated by dividing the cost by the length of the term:

$$(\text{normalized cost}) = \frac{(\text{cost}) + \alpha}{(\text{length of the term})} \tag{4}$$

where α is a constant value [3]. When the costs of two terms are the same, the longer one is preferred due to this constant.

To recognize a protein name in a given text, we perform the above calculation for every term contained in the dictionary and select the term that has the lowest normalized cost. If the cost is lower than the predefined threshold, the corresponding range in the text is recognized as a protein name candidate.

3.2 Implementation Issues for String Searching

A naive way for string searching using a dictionary is to conduct the procedure described in the previous section one by one for every term in the dictionary. However, since the size of a protein name dictionary is usually large ($\sim 10^5$), this naive way requires too much computational cost to deal with a large amount of documents.

Navarro (12) have presented a way to reduce redundant calculations by constructing a trie of the dictionary. The trie is used as a device to avoid repeating the computation of the cost against same prefix of many patterns. Suppose that we have just calculated the cost of the term "EGR-1" and next we have to calculate the cost of the term "EGR-2", it is clear that we do not have to re-calculated the first four rows in the matrix (see Figure 2). They also pointed out that it is possible to determine, prior to reaching the bottom of the matrix, that the current term cannot produce any relevant match: if all the values of the current row are larger than the threshold, then a match cannot occur since we can only increase the cost or at best keep it the same.

4 Expanding Dictionary by Probabilistic Variant Generator

An alternative way to alleviate the problem of spelling variation is to expand each entry in the dictionary in advance. For example, if you have the term "EGR-1" and the rule that spaces and hyphens are interchangeable, you can expand this entry to the two entries "EGR-1" and "EGR 1". With the expanded dictionary, you can find protein names written in varied forms simply by using exact matching algorithms.

However, developing rules for expanding terms is difficult and laborious. In addition, a lot of protein names consisting of many words are present. Suppose that we have the term "nuclear factor of activated T cells" and the rule that spaces and hyphens are interchangeable. The number of possible variants is

[3] α was set to 0.4 in our experiments.

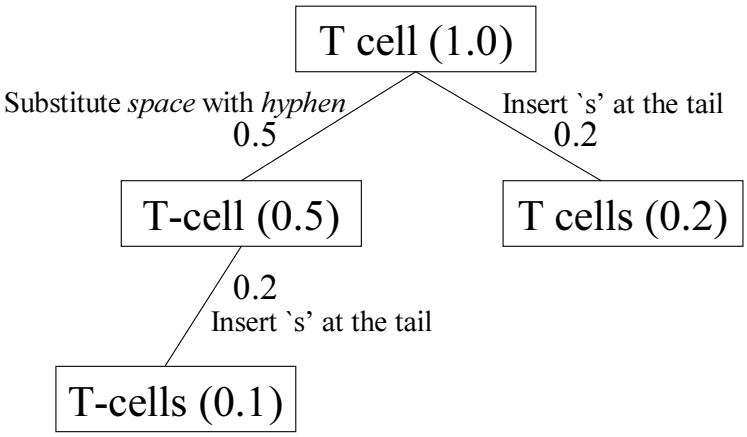

Fig. 3. Probabilistic variant generation. Numerals inside parentheses are generation probabilities, and those along the edges are operation probabilities.

32 because the term has five spaces. If we have another rule that the head of a word can be capitalized, the number of possible variants becomes prohibitively large.

To solve these problems, we propose a method to generate only "likely" spelling variants. Our method not only generates spelling variants but also gives each variant a *generation probability* that represents the plausibility of the variant. Therefore, one does not receive a prohibitive number of unnecessary variants by setting the threshold of generation probability.

4.1 Probabilistic Variant Generator

4.1.1 Generation Probability

The *generation probability* of a variant is defined as the probability that the variant can be generated through a sequence of operations. Each operation has an *operation probability* that represents how likely it will occur. Assuming independence among the operations, the generation probability of a variant can be formalized in a recursive manner,

$$P_X = P_Y \times P_{op}, \tag{5}$$

where P_X is the generation probability of variant X, P_Y is the generation probability of variant Y from which variant X is generated, and P_{op} is the probability of the operation by which Y is transformed into X.

Figure 3 shows an example of the generation process, which can be represented as a tree. Each node represents a generated variant and its probability. Each edge represents an operation and its probability. The root node corresponds to

the input term and the generation probability of the root node is 1 by definition. We can obtain the variants of an input term in order of their generation probabilities by growing a tree in a best-first manner.

4.1.2 Operation Probability

To calculate the generation probabilities in our formalization, we need the probability of each operation.

We used three types of operations for the generation mechanism:

- Substitution
  Replace a character with another character.
- Deletion
  Delete a character.
- Insertion
  Insert a character.

These types of operations are motivated by the ones used in approximate string matching. We consider character-level contexts in which an operation occurs, and we estimate the probability of the operation from a large number of pairs of spelling variations.

An operation probability is defined as the probability that the operation will occur in a given context. First, we represent contexts using the neighboring characters of the operation. The following seven types of contexts are used in this paper. They differ in relative position to the target and in how much the context is specified:

- the target letter and the preceding two letters.
- the target letter and the preceding letter.
- the target letter and the following letter.
- the target letter and the following two letters.
- the target letter, the preceding letter and the following letter.
- the target letter, the preceding two letters and the following two letters.
- the target letter only.

For an operation of a substitution or a deletion, the target indicates a letter in the string. For an operation of an insertion, the target indicates a gap between two letters. For example, if the original string is "c-Rel" and the variant is "c-rel". The operation is a substitution of 'R' with 'r'. The rules obtained from this example are shown in Table 2. They correspond to the seven types of aforementioned context. The first rule indicates that if the letter 'R' is preceded by the string "c-", then one can replace the letter with 'r'.

Table 2
Example of operation rules.

Left Context	Target	Right Context	Operation
c-	R	*	Replace the target with 'r'
-	R	*	Replace the target with 'r'
*	R	e	Replace the target with 'r'
*	R	el	Replace the target with 'r'
-	R	e	Replace the target with 'r'
c-	R	el	Replace the target with 'r'
*	R	*	Replace the target with 'r'

Asterisks indicate a wild card.

The next step is estimating the probability of each rule. The probability should represent how likely the operation will occur in a given context.

We estimate the probabilities from a large number of pairs of spelling variants. In this paper, a pair of spelling variants is defined as follows.

- The two strings convey the same meaning.
- The edit distance between the two strings is 1. In other words, One string must be able to transform to the other in one operation, and vice versa.

"c-Rel" and "c-rel" is an example of a pair of spelling variants. This example contains two operations, the substitution of 'R' with 'r' and the substitution of 'r' with 'R'.

We acquire such kind of variant pairs from UMLS Metathesaurus (13). The thesaurus provides a huge number of biomedical terms and their semantic IDs. We can obtain variant pairs by collecting protein name pairs having the same semantic ID. Examples of the entries in the UMLS Metathesaurus are shown in Table 3. From the protein names in the table, we can obtain the following variant pairs:

{"gp140 v fms", "gp140 v-fms"}

{"v-fms Protein, "v fms Protein"}

Once we have obtained a large set of variant pairs, we can estimate the operation probabilities by using the following equation.

$$P_{op} = P(operation|context) \tag{6}$$

Table 3
A part of UMLS Metathesaurus.

Semantic ID	Protein name
:	:
C0079930	Oncogene Protein gp140(v-fms)
C0079930	Oncogene protein GP140, V-FMS
C0079930	fms Oncogene Product gp140
C0079930	fms Oncogene Protein gp140
C0079930	gp140(v-fms)
C0079930	gp140 v fms
C0079930	gp140 v-fms
C0079930	v-fms, gp140
C0079930	v-fms Protein
C0079930	V-FMS protein
C0079930	v fms Protein
C0079930	Oncogene protein V-FMS
C0079930	GP140 V-FMS protein
:	:

$$\approx \frac{f(context, operation) + 1}{f(context) + 2}, \tag{7}$$

where $f(context)$ is the frequency of the occurrence of the context, and $f(context, operation)$ is the frequency of the simultaneous occurrence of the context and operation in the set of variant pairs. We adopted Laplace's smoothing (adding 1 to the numerator and 2 to the denominator).

4.1.3 Generation Algorithm

Once the rules and their probabilities are learned, we can generate variants from an input term using those rules.

The whole algorithm for variant generation is given below. Note that V represents the set of generated terms.

(1) Initialization
    Add the input term to V.
(2) Selection
    Select a term and an operation to be applied to it so that the generation

probability of the generated term will be the highest possible.

(3) Generation
Generate a new term using the term and the operation selected in Step 2. Then, add the generated term to V.

(4) Repeat
Go back to Step 2 until the termination condition is satisfied.

In the generation step, the system applies the rule whose context matches any part of the string. If multiple rules are applicable, the rule that has the highest operation probability is used.

Because this algorithm generates variants in order of their generation probabilities, the termination condition can be that the generation probability of the generated variant is below the predefined threshold or that the number of generated variants exceeds the predefined threshold.

5 Filtering Candidates by a Naive Bayes Classifier

In the filtering phase, we use a classifier trained on an annotated corpus to suppress false recognition. The objective of this phase is to improve precision without the loss of recall.

We conduct binary classification ("accept" or "reject") on each candidate. The candidates that are classified into "rejected" are filtered out. In other words, only the candidates that are classified into "accepted" are recognized as protein names in the final output.

In this paper, we use a naive Bayes classifier for this classification task.

5.1 Naive Bayes classifier

The naive Bayes classifier is a simple but effective classifier which has been used in numerous applications of information processing including image recognition, natural language processing and information retrieval (14; 15; 16; 17).

Here we briefly review the naive Bayes model. Let $\vec{x}$ be a vector we want to classify, and c_k be a possible class. What we want to know is the probability that the vector $\vec{x}$ belongs to the class c_k. We first transform the probability $P(c_k|\vec{x})$ using Bayes' rule,

$$P(c_k|\vec{x}) = P(c_k) \times \frac{P(\vec{x}|c_k)}{P(\vec{x})} \tag{8}$$

Class probability $P(c_k)$ can be estimated from training data. However, direct estimation of $P(c_k|\vec{x})$ is impossible in most cases because of the sparseness of training data.

By assuming the conditional independence among the elements of a vector, $P(\vec{x}|c_k)$ is decomposed as follows,

$$P(\vec{x}|c_k) = \prod_{j=1}^{d} P(x_j|c_k), \tag{9}$$

where x_j is the jth element of vector $\vec{x}$. Then Equation 8 becomes

$$P(c_k|\vec{x}) = P(c_k) \times \frac{\prod_{j=1}^{d} P(x_j|c_k)}{P(\vec{x})} \tag{10}$$

By this equation, we can calculate $P(c_k|\vec{x})$ and classify $\vec{x}$ into the class with the highest $P(c_k|\vec{x})$.

There are some implementation variants of the naive Bayes classifier depending on their event models (18). In this paper, we adopt the multi-variate Bernoulli event model, in which all features are binary.

5.2 Features

As the input of the classifier, the features of the target must be represented in the form of a vector. We use a binary feature vector which contains only the values of 0 or 1 for each element.

In this paper, we use the local context surrounding a candidate term and the words contained in the term as the features. We call the former *contextual features* and the latter *term features*.

The features used in our experiments are given below.

- Contextual Features
  W_{-1} : the preceding word.
  W_{+1} : the following word.
- Term Features
  W_{begin} : the first word of the term.
  W_{end} : the last word of the term.
  W_{middle} : the other words of the term without positional information (bag-of-words).

Suppose the candidate term is "putative zinc finger protein" and the sentence is

... encoding a putative zinc finger protein was found to derepress beta- galactosidase ...

We obtain the following active features for this example.

$\{W_{-1}$ a$\}$, $\{W_{+1}$ was$\}$, $\{W_{begin}$ putative$\}$, $\{W_{end}$ protein$\}$, $\{W_{middle}$ zinc$\}$, $\{W_{middle}$ finger$\}$.

5.3 Training

The training of the classifier is done on an annotated corpus. We first scan the corpus for protein name candidates by the dictionary matching method described in Section 3 or 4. If a recognized candidate is annotated as a protein name, this candidate and its context are used as a positive ("accepted") example for training. Otherwise, it is used as a negative ("rejected") example.

6 Experiment

6.1 Corpus and Dictionary

We conducted experiments of protein name recognition using the GENIA corpus version 3.02 (6). The GENIA corpus is an annotated corpus, which contains 2,000 abstracts extracted from MEDLINE database. These abstracts are selected from the search results with MeSH terms *Human*, *Blood Cells*, and *Transcription Factors*.

The biological entities in the corpus are annotated according to the GENIA ontology. Although the corpus has many categories such as protein, DNA, RNA, cell line and tissue, we used only the protein category. When a term was recursively annotated, only the outermost (longest) annotation was used.

The first 200 abstracts of the corpus were used as the test data. The remaining 1,800 abstracts were used as the training data. The protein name dictionary was constructed from the training data by collecting all the terms that were annotated as proteins.

Each recognition was counted as correct if the both boundaries of the recog-

Table 4
Precision improvement by filtering.

	Precision	Recall	F-measure
without filtering	45.2%	66.4%	53.8%
with filtering	71.7%	60.8%	65.8%

nized term exactly matched the boundaries of an annotation in the corpus.

6.2 Improving Precision by Filtering

We first conducted experiments to evaluate how much precision is improved by the filtering process. In the candidate recognition phase, the longest matching algorithm was used for candidate recognition.

The results are shown in Table 4. F-measure is defined as the harmonic mean of precision and recall:

$$F = \frac{2 \times precision \times recall}{precision + recall} \tag{11}$$

The first row shows the performances achieved without filtering. In this case, all the candidates identified in the candidate recognition phase are regarded as protein names. The second row shows the performance achieved with filtering by the naive Bayes classifier. In this case, only the candidates that are classified into "accepted" are regarded as protein names. Notice that the filtering significantly improved the precision (from 45.2% to 71.7%) with slight loss of the recall. The F-measure was also greatly improved (from 53.8% to 65.8%).

6.2.1 Efficacy of Contextual Features

The advantage of using a machine learning technique is that we can exploit the context of a candidate. In order to evaluate the efficacy of contexts, we conducted experiments using different feature sets.

Table 5 shows the results. The first row shows the performances achieved by using only contextual features. The second row shows those achieved by using only term features. The performances achieved by using both feature sets are shown in the third row.

The results indicate that candidate terms themselves are strong cues for classification. However, the fact that the best performance was achieved when both

Table 5
Performance on different feature sets.

Feature Set	Precision	Recall	F-measure
contextual features	59.6%	59.1%	59.4%
term features	68.9%	61.0%	64.7%
all features	71.7%	60.8%	65.8%

Table 6
Effectiveness of approximate string search.

Threshold	Precision	Recall	F-measure
2	71.9%	59.8%	65.3%
4	71.6%	62.0%	66.4%
6	71.2%	62.2%	66.4%
8	70.3%	62.9%	66.4%
10	69.7%	63.6%	66.5%
12	69.0%	64.3%	66.6%
14	68.5%	65.4%	66.9%
16	68.0%	65.7%	66.8%
18	67.3%	66.0%	66.6%
20	64.7%	67.0%	65.8%
22	62.9%	67.2%	65.0%

feature sets were used suggests that the context of a candidate conveys useful information about the semantic class of the candidate.

6.3 Improving Recall by Approximate String Search

We conducted experiments to evaluate how much we can further improve the recognition performance by using the approximate string searching method described in Section 3. Table 6 shows the results. The leftmost columns show the thresholds of the normalized costs for approximate string searching. As the threshold increased, the precision degraded while the recall improved. The best F-measure was 66.9%, which is better than that of exact matching by 1.1% (see Table 4).

6.4 Expanding Dictionary by Variant Generation

6.4.1 Variant Generator

We used UMLS Metathesaurus version 2003AA for learning operation rules. The terms having the semantic type of "Amino Acid, Peptide, or Protein" were used for the learning. Table 7 shows a part of the obtained rules and their probabilities. Asterisks in the table is a wild card, meaning that one can ignore the context of that position. Notice that there are many rules for replacing spaces and hyphens. This suggests that spaces and hyphens are often used interchangeably in protein names.

The variants of some biomedical terms generated by our algorithm are shown in Tables 8 to 11. The first two input terms "NF-kappa B" and "transcription factor" are the two most frequent protein names in the corpus.

The generated variants of "transcription factor" shown in Table 9 are interesting. The first letter of "transcription" and "factor" is capitalized in the second and third variant respectively. This reflects the fact that the first letter of a word is often capitalized in biomedical terms. Notice that the plural form of "factor" is generated in the first variant.

The variants for the input term "tumor necrosis factor" are shown in Table 10. It should be noted that transformation to the British spelling variation of "tumor" appears in the seventh variant.

The variants for the input term "T cell factor 1" are shown in Table 11. Notice that the variant in which a hyphens is inserted between 'T' and "cell" was ranked at the top. The eighth variant "T cell factor I" is also interesting, where the numeral '1' is replaced with the letter 'I'.

6.4.2 Dictionary Expansion

We conducted experiments of dictionary expansion using the varinat generator. Expansions were done on the terms whose length was equal to or longer than five characters. The maximum number of variants generated for each term was limited to 100.

Table 12 shows the effectiveness of dictionary expansion. The leftmost columns show the threshold of generation probability for expanding the dictionary. The recall improves as the threshold decreases. The best F-measure is 67.0%, which is roughly equal to the performance of approximate string search.

Table 7
Operation rules and their probabilities.

Operation Probability	Left Context	Target	Right Context	Operation
0.971	*	^	*	delete the target
0.958	*	o	ea	delete the target
0.958	rh		ea	insert 'o'
0.952	e	*hyphen*	R	replace the target with *space*
0.950	or	*space*	3,	replace the target with *hyphen*
0.950	or	*hyphen*	3,	replace the target with *space*
0.947	TH		(1	insert *space*
0.947	or	s	_a	delete the target
0.945	*	*space*	tR	replace the target with *hyphen*
0.938	_T	*space*	Ce	replace the target with *hyphen*
0.938	L	*space*	I	replace the target with *hyphen*
0.938	in	*space*	bi	replace the target with *hyphen*
0.938	ne	*space*	tR	replace the target with *hyphen*
0.938	3	*space*	*	replace the target with *hyphen*
0.938	r	*hyphen*	3	replace the target with *space*
0.938	E	*space*	1	replace the target with *hyphen*
0.938	V	*space*	I	replace the target with *hyphen*
0.938	ne	s	_R	delete the target
0.938	*	s	_2	delete the target
0.929	*start of term*	l	o	replace the target with 'L'
0.929	NA	*space*	DE	replace the target with *hyphen*
0.929	NA	*hyphen*	DE	replace the target with *space*
0.929	in	*hyphen*	A	replace the target with *space*
0.929	rg	*hyphen*	*	replace the target with *space*
0.923	k	a	e	delete the target
0.923	x	*hyphen*	1	delete the target
:	:	:	:	:

Asterisks indicate a wild card.

Table 8
Generated variants for “NF-kappa B”.

Generation Probability	Generated String
1.000	NF-kappa B
0.466	NF kappa B
0.317	NF-kappa-B
0.286	NF-Kappa B
0.233	NFkappa B
0.211	NP-kappa B
0.199	NP kappa B
0.190	NF Kappa B
0.150	NF-kappaB
0.148	NF kappa-B
0.090	NF-Kappa-B
0.081	NP Kappa B
:	:

Table 9
Generated variants for “transcription factor”.

Generation Probability	Generated String
1.000	transcription factor
0.571	transcription factors
0.356	Transcription factor
0.219	transcription Factor
0.206	trancription factor
0.203	Transcription factors
0.137	transcription-factor
0.125	transcription Factors
0.117	trancription factors
0.107	transcription factorss
0.078	transcription-factors
0.073	Trancription factor
:	:

Table 10
Generated variants for "Tumor necrosis factor".

Generation Probability	Generated String
1.000	Tumor necrosis factor
0.571	Tumor necrosis factors
0.218	Tumor necrosi factor
0.188	Tumors necrosis factor
0.176	tumor necrosis factor
0.139	Tumor-necrosis factor
0.137	Tumor necrosis-factor
0.125	Tumour necrosis factor
0.124	Tumor necrosis Factor
0.124	Tumor necrosi factors
0.107	Tumors necrosis factors
:	:

Table 11
Generated variants for "T cell factor 1".

Generation Probability	Generated String
1.000	T cell factor 1
0.604	T-cell factor 1
0.498	T cell factor-1
0.301	T-cell factor-1
0.196	T cell factors 1
0.139	T cell factor1
0.137	T cell-factor 1
0.135	t cell factor 1
0.129	T cell factor I
0.124	T cell Factor 1
0.118	T-cell factors 1
:	:

Table 12
Effectiveness of Dictionary Expansion

Threshold	Precision	Recall	F-measure
2^{-1}	71.8%	61.5%	66.2%
2^{-2}	72.0%	61.9%	66.6%
2^{-3}	72.1%	61.6%	66.4%
2^{-4}	72.4%	61.9%	66.7%
2^{-5}	72.5%	62.3%	67.0%
2^{-6}	72.5%	62.1%	66.9%

7 Related Work

Kazama et al. (9) reported an F-measure of 56.5% on the GENIA corpus version 1.1 using SVMs. Collier et al. (19) reported an F-measure of 75.9% on 100 MEDLINE abstracts by using a Hidden Markov Model. Since the evaluation corpora used in their experiments are different from the corpus used in this paper, the results are not directly comparable.

Lee et al. (20) reported an F-measure of 69.2% on the GENIA corpus version 3.0 using SVMs. Shen et al. achieved an F-measure of 70.8% on the same corpus by incorporating various features into a Hidden Markov Model. Since the difference between the GENIA corpus version 3.0 and the GENIA corpus version 3.02, which we used in this paper, is small, their results suggest that their methods work better than our method regarding recognition performance. However, their approaches do not provide ID information of recognized terms.

Krauthammer et al. (21) proposed a dictionary-based gene/protein name recognition method. They used BLAST for approximate string matching by mapping sequences of text characters into sequences of nucleotides that can be processed by BLAST. They achieved a recall of 78.8% and a precision of 71.1% evaluated by a partial match criterion, which is less strict than our criterion.

8 Conclusion

In this paper we propose a two-phase protein name recognition method. In the first phase, we scan texts for protein name candidates using a protein name dictionary. In the second phase, we filter the candidates using a machine learning technique. Our method is dictionary-based and can provide ID

information of recognized terms unlike machine learning approaches.

Experimental results using the GENIA corpus show that the filtering using a naive Bayes classifier greatly improves precision with slight loss of recall. We achieved an F-measure of 67.0% for protein name recognition on the GENIA corpus.

We presented two approaches for alleviating the low-recall problem caused by spelling variation. The first one is to use an approximate string searching algorithm instead of exact matching algorithms. The other one is to expand the dictionary in advance by using the variant generator.

Although the experimental results showed that the performances of the two approaches are roughly the same, we found the dictionary expansion approach much more attractive. The main reason is the cost of computation: the computational cost of approximate string searching is far bigger than exact matching. Since the amount of available biomedical documents is huge, the processing speed is an important factor of information extraction systems.

The future direction of this research involves:

- Use of state-of-the-art classifiers
  We have used a naive Bayes classifier in our experiments because it requires a small computational resource and exhibits good performance. There is a chance, however, to improve performance by using state-of-the-art machine learning techniques including maximum entropy models and support vector machines.
- Extending the algorithm for variant generation
  Three types of operations are considered in this paper for the mechanism of variant generation. There can be, however, other types of operations, such as word-insertion and word-replacement. Our future work should encompass those types of operations to improve the recall for long protein names.

References

[1] E. M. Marcotte, I. Xenarios, D. Eisenberg, Mining literature for protein-protein interactions, BIOINFORMATICS 17 (4) (2001) 359–363.

[2] J. Thomas, D. Milward, C. Ouzounis, S. Pulman, M. Carroll, Automatic extraction of protein interactions from scientific abstracts, in: Proceedings of the Pacific Symposium on Biocomputing (PSB2000), Vol. 5, 2000, pp. 502–513.

[3] T. Ono, H. Hishigaki, A. Tanigami, T. Takagi, Automated extraction of information on protein-protein interactions from the biological literature, BIOINFORMATICS 17 (2) (2001) 155–161.

[4] V. N. Vapnik, The Nature of Statistical Learning Theory, Springer, New York, 1995.
[5] A. L. Berger, S. A. D. Pietra, V. J. D. Pietra, A maximum entropy approach to natural language processing, Computational Linguistics 22 (1) (1996) 39–71.
[6] T. Ohta, Y. Tateisi, J.-D. Kim, J. Tsujii, Genia corpus: an annotated research abstract corpus in molecular biology domain, in: Proceedings of the Human Language Technology Conference (HLT 2002), 2002.
[7] K. Takeuchi, N. Collier, Use of support vector machines in extended named entity recognition, in: Proceedings of the 6th Conference on Natural Language Learning 2002 (CoNLL-2002), 2002, pp. 119–125.
[8] J. D. Kim, J. Tsujii, Corpus-based approach to biological entity recognition, in: Text Data Mining SIG (ISMB2002), 2002.
[9] J. Kazama, T. Makino, Y. Ohta, J. Tsujii, Tuning support vector machines for biomedical named entity recognition, in: Proceedings of the ACL-02 Workshop on Natural Language Processing in the Biomedical Domain, 2002.
[10] G. Navarro, Approximate text searching, Ph.D. thesis, Dept. of Computer Science, Univ. of Chile (1998).
[11] G. Navarro, A guided tour to approximate string matching, ACM Computing Surveys 33 (1) (2001) 31–88.
[12] G. Navarro, R. Baeza-Yates, J. Arcoverde, Matchsimile: A flexible approximate matching tool for personal names searching, in: Proceedings of the XVI Brazilian Symposium on Databases (SBBD'2001), 2001, pp. 228–242.
[13] B. L. Humphreys, D. A. B. Lindberg, Building the unified medical language system, in: Proceedings of the 13th SCAMC, 1989, pp. 475–480.
[14] D. D. Lewis, Naive Bayes at forty: The independence assumption in information retrieval., in: Proceedings of ECML-98, 10th European Conference on Machine Learning, no. 1398, 1998, pp. 4–15.
[15] G. Escudero, L. arquez, G. Rigau, Naive bayes and exemplar-based approaches to word sense disambiguation revisited, in: Proceedings of the 14th European Conference on Artificial Intelligence, 2000.
[16] T. Pedersen, A simple approach to building ensembles of naive bayesian classifiers for word sense disambiguation, in: Proceedings of the First Annual Meeting of the North American Chapter of the Association for Computational Linguistics, 2000, pp. 63–69.
[17] K. Nigam, R. Ghani, Analyzing the effectiveness and applicability of co-training, in: CIKM, 2000, pp. 86–93.
[18] A. McCallum, K. Nigam, A comparison of event models for naive bayes text classification, In AAAI-98 Workshop on Learning for Text Categorization.
[19] N. Collier, C. Nobata, J. Tsujii, Automatic acquisition and classification of molecular biology terminology using a tagged corpus, Journal of Terminology 7 (2) (2001) 239–258.

[20] K.-J. Lee, Y.-S. Hwang, H.-C. Rim, Two-phase biomedical ne recognition based on svms, in: Proceedings of the ACL 2003 Workshop on Natural Language Processing in Biomedicine, 2003, pp. 33–40.

[21] M. Krauthammer, A. Rzhetsky, P. Morozov, C. Friedman, Using BLAST for identifying gene and protein names in journal articles, Gene 259 (2000) 245–252.