CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Elsevier - Publisher Connector

Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Biomedical Informatics 37 (2004) 461–470
Journal of Biomedical Informatics
www.elsevier.com/locate/yjbin

Improving the performance of dictionary-based approaches in protein name recognition

Yoshimasa Tsuruoka*, Jun'ichi Tsujii

CREST, Japan Science and Technology (JST) Agency, Honcho 4-1-8, Kawaguchi-shi, Saitama 332-0012, Japan
Department of Computer Science, University of Tokyo, Hongo 7-3-1, Bunkyo-ku, Tokyo 113-0033, Japan

Received 22 July 2004
Available online 8 October 2004

Abstract

Dictionary-based protein name recognition is often a first step in extracting information from biomedical documents because it can provide ID information on recognized terms. However, dictionary-based approaches present two fundamental difficulties: (1) false recognition mainly caused by short names; (2) low recall due to spelling variations. In this paper, we tackle the former problem using machine learning to filter out false positives and present two alternative methods for alleviating the latter problem of spelling variations. The first is achieved by using approximate string searching, and the second by expanding the dictionary with a probabilistic variant generator, which we propose in this paper. Experimental results using the GENIA corpus revealed that filtering using a naive Bayes classifier greatly improved precision with only a slight loss of recall, resulting in 10.8% improvement in *F*-measure, and dictionary expansion with the variant generator gave further 1.6% improvement and achieved an *F*-measure of 66.6%.
© 2004 Elsevier Inc. All rights reserved.

Keywords: Protein name recognition; Naive Bayes classifier; Approximate string search; Spelling variant generator

1. Introduction

The rapid increase in machine readable biomedical texts (e.g., MEDLINE) makes automatic information extraction from these texts much more attractive. One of the most important tasks today is extracting information on protein–protein interactions from MEDLINE abstracts [1–3].

To be able to extract information about proteins from a text, one has to first recognize their names in it. This kind of problem has been extensively studied in the field of natural language processing as the named-entity recognition task. 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]. The task of the classifier in the machine learning framework is to determine the text regions corresponding to protein names.

Ohta et al. [6] provided the GENIA corpus, an annotated corpus of MEDLINE abstracts, which could 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 [7–9].

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

* Corresponding author. Fax: +81 3 5802 8872.
E-mail addresses: tsuruoka@is.s.u-tokyo.ac.jp (Y. Tsuruoka), tsujii@is.s.u-tokyo.ac.jp (J. Tsujii).

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

1532-0464/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jbi.2004.08.003

Dictionary-based approaches intrinsically provide ID information because they recognize a term by searching the one that is most similar (or identical) in the dictionary to the target region. This advantage makes dictionary-based approaches particularly useful as the first step in practical information extraction from biomedical documents [3].

Dictionary-based approaches, however, present two fundamental difficulties. The first is a large number of false positives mainly caused by short names, which significantly degrades 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 to filter out the false positives.

The other problem in dictionary-based approaches derives 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 failure to find these protein names written in various forms.

We present two alternative solutions to the problem of spelling variation in this paper. The first is using approximate string searching techniques where the surface-level similarity of strings is considered. The second is expanding the dictionary in advance with a *probabilistic variant generator*, which we propose in this paper. We present experimental results on the GENIA corpus to demonstrate their effectiveness.

This paper is organized as follows. Section 2 overviews our method of recognizing protein names. Section 3 explains the approximate string searching algorithm used to alleviate the problem of spelling variations. As an alternative solution to the problem, Section 4 describes the probabilistic variant generator that is used to expand the dictionary. Section 5 describes how false recognition is filtered out with a machine learning method. Section 6 presents experimental results obtained using the GENIA corpus. Some related work is described in Section 7. Finally, Section 8 has some concluding remarks.

2. Method overview

Our method of recognizing protein names involves the following two phases.

- Candidate recognition phase
  The task of this phase is to find protein name candidates appearing in the text using a dictionary. We propose two alternative solutions to the problem of spelling variations. The first is to use an approximate string searching algorithm instead of exact matching algorithms, which is presented in Section 3. The second is to expand the dictionary in advance with the variant generator, which is presented in Section 4.
- Filtering phase
  One of the most serious problems with dictionary-based recognition is the large number of false positives mainly caused by short entries in the dictionary. Our solution to this problem is to check whether each candidate is really a protein name or not. In other words, each protein name candidate is classified into "accepted" or "rejected" using a machine learning algorithm. The classifier uses the context of the term and the term itself as the features for classification. Only "accepted" candidates are recognized as protein names in the final output. Section 5 describes details of the classification algorithm used in this phase.

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

3. Candidate recognition by approximate string searching

One way to deal with the problem of spelling variations is to use a kind of "elastic" matching algorithm, by which a recognition system scans a text to find a similar term (if any) to a protein name in the dictionary. We need a similarity measure for this task. The most popular measure of similarity between two strings is the *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 [10]. Fig. 1 illustrates an example.[3] In matrix $C_{0..|x|,0..|y|}$, each cell $C_{i,j}$ keeps the minimum number of operations needed to match $x_{1..i}$ to $y_{1..j}$ and can be computed as a simple function of the surrounding cells

$$C_{i,0} = i, \tag{1}$$

$$C_{0,j} = j, \tag{2}$$

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

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

[3] For clarity of presentation, all costs have been assumed to be 1.

		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.

There are some algorithms that run faster than the dynamic programming method in 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 their uniform-cost edit distances are equal.

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

Table 1 shows the cost function we 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. Since these costs were heuristically determined by just observing a number of protein names, it is likely that we could achieve better performance by employing a systematic method of determining the cost function [12].

3.1. String searching

What we have described in the previous section is a method of calculating the similarity between two strings.

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 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

However, what we need when trying to find proteins is approximate string searching in which the recognizer scans a text to find a similar term (if any) to a term in the dictionary. The dynamic programming-based method can be easily extended for approximate string searching.

The method is illustrated in Fig. 2. In this case, the protein name to be matched is "EGR-1" and the text to be scanned is "encoded by EGR include." String searching can be done by just setting the elements corresponding to the 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.[4] When the costs for two terms are equal, the longer one is preferred due to this constant. In the case of Fig. 2, the normalized cost of the term is $(1 + 0.4)/4 = 0.35$.

To recognize a protein name in a given text, we do 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 in string searching

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

Navarro et al. [13] presented a way of reducing 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 the same prefix of many patterns. Suppose that we have just calculated the cost of the term "EGR-1" and we have to calculate next the cost of the term "EGR-2"; it is clear that we do not have to re-calculate the first four rows in the matrix (see Fig. 2). They also indicated that it is possible to determine, prior to reaching the bottom of the matrix, that the current term cannot produce any relevant

[4] α was heuristically set to 0.4 in our experiments. It would be possible to tune the value by conducting cross-validation on the training data with additional computational costs.

		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 dynamic programming matrix.

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.

We adopted Navarro's method: we first constructed a trie and filled the matrix one by one for each term, avoiding redundant calculations. The computational cost for approximate string searching was very large even with these devices for efficient computation. In our implementation, it took more than an hour to process 2000 MEDLINE abstracts on a 1.13 GHz pentium III server.

4. Expanding the dictionary with probabilistic variant generator

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

For this purpose, we propose an algorithm that can 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 need to receive a prohibitive number of unnecessary variants by setting an appropriate threshold for 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 is that it will occur. Assuming independence among operations, the generation probability of a variant can be formalized in a recursive manner

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

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.

Fig. 3 outlines 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 for 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

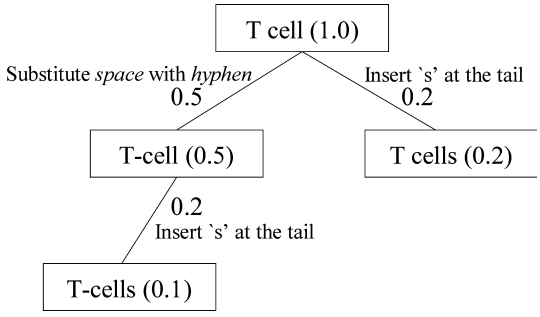

Fig. 3. Probabilistic variant generation. Numerals inside parentheses are generation probabilities, and those along the edges are operation probabilities.

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 for a substitution or a deletion, the target indicates a letter in the string. For an operation for 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 operation rules obtained from this example are listed in Table 2. They correspond to the seven types of the 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.'

The next step is estimating the probability of each rule. The probability should represent how likely the operation is to occur in a given context. We estimated the operation probabilities from a large number of variant pairs with the following equation:

$$P_{\mathrm{op}} = P(\mathrm{operation}|\mathrm{context}) \approx \frac{f(\mathrm{context}, \mathrm{operation}) + 1}{f(\mathrm{context}) + 2}, \quad (6)$$

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

We acquired the actual samples of variant pairs for probability estimation from the UMLS Metathesaurus [14], which provides a huge number of biomedical terms and their concept IDs. We first obtained sets of variants by collecting protein names with the same concept ID. We then selected from each set the pairs whose edit distance is one, and used them for probability estimation.

For example, we obtained the following variant pairs from the set shown in Table 3.

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

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 represent wild cards.

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 the term and the operation to be applied to it so that the algorithm will generate a new term which has the highest possible probability.
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 can be applied, the rule that has the highest operation probability is used.

Because this algorithm generates variants in the order of their generation probability, the termination condi-

Table 3
A part of UMLS Metathesaurus

Concept 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
:	:

tion can be where that the generation probability for the generated variant is below the predefined threshold or that the number of generated variants exceeds the predefined threshold.

5. Filtering candidates by means of 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 used a naive Bayes classifier for this task of classification. The naive Bayes model is simple but effective and has been used in numerous applications of information processing including image recognition, natural language processing, and information retrieval [15–18]. Because this model assumes conditional independence among features, it is possible to estimate its parameters from a limited amount of training data.

There are some implementation variants with the naive Bayes classifier depending on their event models [19]. In this paper, we adopted the multi-variate Bernoulli event model, in which we can employ any types of binary features.

5.1. Features

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:

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

5.2. Training

The training of the classifier is carried out using an annotated corpus. We first scan the corpus for protein name candidates with 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 on protein name recognition using the GENIA corpus version 3.02 [6], which contains 2000 abstracts extracted from the MEDLINE database. These abstracts were selected from the search results with the 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 only used the protein category. When a term was recursively annotated, only the outermost (longest) annotation was used.

We conducted tenfold cross-validation for evaluating the methods. Each set of 200 abstracts was used as the test data, and the remaining 1800 abstracts were used as the training data. The results were averaged over the 10 runs.

The protein name dictionary was constructed from the training data by collecting all the terms that were annotated as proteins.[5] The average number of terms contained in the dictionary was 8055.

Each recognition was counted as correct if both boundaries of the recognized term exactly matched the boundaries of an annotation in the corpus.

6.2. Improving precision by filtering

We first conducted experiments to evaluate to what extent the filtering process improved precision. In the candidate-recognition phase, the longest matching algorithm was used for candidate recognition.

[5] We used the training data for constructing the protein name dictionary so that the recognition performance could be comparable with those of machine learning-based approaches. However, there is a chance that we have lower recognition performance if we use an external resource for constructing the dictionary.

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

$$F = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}. \quad (7)$$

The first row shows performance achieved without filtering, where all candidates identified in the candidate-recognition phase are regarded as protein names. The other rows show performance achieved with filtering using the naive Bayes classifier. In this case, only the candidates that were classified into "accepted" were regarded as protein names. Notice that the filtering significantly improved the precision (from 46.5 to 71.2%) with only a slight loss in recall. The *F*-measure was also greatly improved (from 54.3 to 65.1%).

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. To evaluate the efficacy of contexts, we conducted experiments using different feature sets.

The results in Table 4 indicate that candidate terms themselves provide strong clues for classification. However, the fact that the best performance was achieved when both feature sets were used suggests that the context of a candidate term conveys useful information about its semantic class.

6.3. Improving recall by approximate string search

We conducted experiments to evaluate how much we could further improve recognition by using the approximate string-searching method described in Section 3. Table 5 lists the results. The leftmost columns show the thresholds of normalized costs for approximate string searching. As the threshold increased, precision degraded while recall improved. The best *F*-measure was 66.1%, which is better than that of exact matching by 1.0% (see Table 4). It should be noted that approximate string-searching without filtering did not improve *F*-measure: the decline in precision impeded an improvement in recall.

6.4. Expanding the dictionary by variant generation

6.4.1. Variant generator

We used the UMLS Metathesaurus, version 2003AA to learn the operation rules. Terms having the semantic type of "Amino Acid, Peptide, or Protein" were used for learning, and they provide 36,112 pairs of spelling variants. Table 6 lists some of the rules obtained and their probabilities. The asterisks in the table represent wild cards, 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 listed in Tables 7, 8, 9, and 10. 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" listed in Table 8 are interesting. The first letters of "transcription" and "factor" were 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 listed in Table 9. It should be noted that the transformation to the British spelling variation for "tumor" appears in the seventh variant.

Table 4
Precision improvement by filtering

	Feature sets	Precision (%)	Recall (%)	*F*-measure (%)
Without filtering	N/A	46.5	65.4	54.3
With filtering	Contextual features	60.1	58.0	59.0
	Term features	68.2	59.8	63.7
	All features	71.2	60.1	65.1

Table 5
Effectiveness of approximate string search

Threshold	Without filtering			With filtering		
	Precision (%)	Recall (%)	*F*-measure (%)	Precision (%)	Recall (%)	*F*-measure (%)
4	45.8	67.4	54.5	70.6	61.8	65.9
8	42.4	68.4	52.3	69.6	63.0	66.1
12	37.4	69.3	48.5	68.3	63.9	66.0
16	30.6	70.0	42.5	66.7	64.7	65.6
20	21.0	71.4	32.4	63.5	66.1	64.7

Table 6
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 represent wild cards.

Table 7
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
:	:

The variants for the input term "T cell factor 1" are listed in Table 10. Note that the variant where a hyphen is inserted between the 'T' and "cell" is 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 on dictionary expansion using the variant generator. Expansions were carried out on terms whose lengths were equal to or longer than five characters. The maximum number of variants generated for each term was limited to 100.[6]

Table 11 shows the effectiveness of dictionary expansion. The leftmost columns show the threshold for generation probability to expand the dictionary. The recall improved as the threshold decreased. The best *F*-measure was 66.6%, which is 1.6% higher than that of the original dictionary (see Table 4).

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. [20] reported an *F*-measure of 75.9% on 100 MEDLINE abstracts using a Hidden Markov Model. Tanabe and Wilbur [21] achieved 85.7% precision and 66.7% recall using a combination of statistical and knowledge-based strategies. They used a transformation-based part-of-speech tagger to recognize single word protein names, and hand-crafted rules to filter out false positives and recover false negatives. Since the evaluation corpora used in these experiments were different from the corpus

[6] For equi-probable variants, the program continues to generate even when the number of variants exceeds the limit.

Table 8
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 9
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	Tumor 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	Tumor necrosis factors
:	:

Table 10
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
:	:

used in this paper, the results are not directly comparable.

Lee et al. [22] 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 corpora versions 3.0 and 3.02, which we used in this paper, is small, their results suggest that their methods worked better than ours regarding recognition. However, their approaches do not provide ID information on recognized terms.

Krauthammer et al. [23] proposed a dictionary-based method of gene/protein name recognition. They used BLAST for approximate string matching by mapping sequences of text characters into sequences of nucleotides that could be processed by BLAST. They achieved a recall of 78.8% and a precision of 71.1% evaluated with a partial match criterion, which was not as stringent as our criterion.

8. Conclusion

We proposed a method of two-phase protein name recognition. In the first phase, we scan texts for protein name candidates using a protein name dictionary. In the second phase, we filter the candidates via a process of machine learning. Our method is dictionary-based and can provide ID information on recognized terms, unlike machine learning approaches.

We presented two approaches to alleviate the low-recall problem caused by spelling variations. The first is to use an approximate string-searching algorithm instead of exact-matching algorithms. The second is to expand the dictionary in advance with the variant generator. We found the dictionary expansion approach to be much more attractive. The main reason was the cost of computation: the computational cost involved with approximate string searching was far greater than for exact matching. Since there are huge amounts of available biomedical documentation, processing speed is an important factor in information extraction systems.

Table 11
Effectiveness of dictionary expansion

Threshold	Without filtering			With filtering		
	Precision (%)	Recall (%)	*F*-measure (%)	Precision (%)	Recall (%)	*F*-measure (%)
2^{-1}	46.4	66.4	54.6	71.2	61.0	65.7
2^{-2}	46.5	67.1	54.9	71.6	61.7	66.2
2^{-3}	46.5	67.4	55.0	71.6	61.7	66.3
2^{-4}	46.4	67.7	55.0	71.8	62.1	66.5
2^{-5}	46.0	68.0	54.8	71.7	62.3	66.6
2^{-6}	46.0	68.1	54.8	71.7	62.3	66.6

Experimental results using the GENIA corpus revealed that filtering using a naive Bayes classifier greatly improved precision with only a slight loss of recall, resulting in 10.8% improvement in *F*-measure, and dictionary expansion with the variant generator gave further 1.6% improvement and achieved an *F*-measure of 66.6%.

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 limited computational resources and performs well. There is a chance, however, of improving 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 were considered in this paper for the mechanism responsible for generating variants. There can, however, be other types of operations, such as word-insertion and word-replacement. Our future work should encompass these types of operation to improve recall for long protein names.
- Applying the variant generator to other classes
  We conducted experiments only on protein names in this paper. Since the variant generator can be easily applied to other classes, it would be interesting to investigate the performance gain on other classes. We expect to achieve comparable performance gain on at least gene and RNA names.

Acknowledgments

This work was partially supported by Grant-in-Aid for Scientific Research on Priority Areas (C) ''Genome Information Science'' from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

[1] Marcotte EM, Xenarios I, Eisenberg D. Mining literature for protein–protein interactions. Bioinformatics 2001;17(4):359–363.
[2] Thomas J, Milward D, Ouzounis C, Pulman S, Carroll M, Automatic extraction of protein interactions from scientific abstracts. In: Proceedings of the pacific symposium on biocomputing (PSB2000), vol. 5. 2000. p. 502–13.
[3] Ono T, Hishigaki H, Tanigami A, Takagi T. Automated extraction of information on protein-protein interactions from the biological literature. Bioinformatics 2001;17(2):155–61.
[4] Vapnik VN. The nature of statistical learning theory. New York: Springer; 1995.
[5] Berger AL, Pietra SAD, Pietra VJD. A maximum entropy approach to natural language processing. Comput Linguist 1996;22(1):39–71.
[6] Ohta T, Tateisi Y, Kim J-D, Tsujii J. Genia corpus: an annotated research abstract corpus in molecular biology domain. In: Proceedings of the human language technology conference (HLT 2002); 2002.
[7] Takeuchi K, Collier N. Use of support vector machines in extended named entity recognition. In: Proceedings of the 6th conference on natural language learning 2002 (CoNLL-2002); 2002. p. 119–25.
[8] Kim JD, Tsujii J. Corpus-based approach to biological entity recognition. In: Text data mining SIG (ISMB2002); 2002.
[9] 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; 2002. p. 1–8.
[10] Jurafsky D, Martin JH. Speech and language processing. Prentice-Hall; 2000.
[11] Navarro G. A guided tour to approximate string matching. ACM Comput Surveys 2001;33(1):31–88.
[12] Ristad ES, Yianilos PN. Learning string-edit distance. IEEE Trans Pattern Anal Mach Intell 1998;20(5):522–32.
[13] Navarro G, Baeza-Yates R, Arcoverde J, Matchsimile: a flexible approximate matching tool for personal names searching. In: Proceedings of the XVI brazilian symposium on databases (SBBD'2001); 2001. p. 228–42.
[14] Humphreys BL, Lindberg DAB. Building the unified medical language system. In: Proceedings of the 13th SCAMC; 1989. p. 475–80.
[15] Lewis DD. Naive Bayes at forty: the independence assumption in information retrieval. In: Proceedings of ECML-98, 10th European conference on machine learning, no. 1398. 1998. p. 4–15.
[16] Escudero G, Marquez L, Rigau G. Naive Bayes and exemplar-based approaches to word sense disambiguation revisited. In: Proceedings of the 14th european conference on artificial intelligence; 2000. p. 421–25.
[17] Pedersen T, 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. p. 63–9.
[18] Nigam K, Ghani R. Analyzing the effectiveness and applicability of co-training. In: CIKM; 2000. p. 86–93.
[19] McCallum A, Nigam K. A comparison of event models for naive Bayes text classification. In: AAAI-98 workshop on learning for text categorization; 1998.
[20] Collier N, Nobata C, Tsujii J. Automatic acquisition and classification of molecular biology terminology using a tagged corpus. J Terminol 2001;7(2):239–58.
[21] Tanabe L, Wilbur WJ. Tagging gene and protein names in biomedical text. Bioinformatics 2002;18(8):1124–32.
[22] Lee K-J, Hwang Y-S, Rim H-C. Two-phase biomedical NE recognition based on SVMs. In: Proceedings of the ACL 2003 workshop on natural language processing in biomedicine; 2003. p. 33–40.
[23] Krauthammer M, Rzhetsky A, Morozov P, Friedman C. Using BLAST for identifying gene and protein names in journal articles. Gene 2000;259:245–52.