Internal Report 2013-09 August 2013

Universiteit Leiden
Opleiding Informatica

A Gene Name Normalization Framework

for

a Thesis

Yi Wang

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

A Gene Name Normalization Framework

Yi Wang(s1128361)
LIACS
Leiden University
The Netherlands
csimplestring@gmail.com

Dr. Erwin M. Bakker(supervisor)
LIACS
Leiden University
The Netherlands
erwin@liacs.nl

Contents

Abstract

In Biology Natural Language Processing (BioNLP), the identification of genes or protein names in medical articles is particularly significant and challenging. The recognition and normalization of textual gene mentions make it more convenient for researchers to organize and manage their domain knowledge. However, this task is usually conducted by human experts, which is relatively expensive for economic and time reason. With more and more biological and medical articles being published, human efforts on this task seem to be impractical.

To automate the process of recognizing and normalizing of textual gene mentions, we propose a generic framework integrating various methodologies and resources, focusing on 2 main problems: 1) automatic recognition of gene names in medical articles; 2) the identification of detected gene names, assigning an EntrezGene identifier to each gene name. The contributions of this paper include: A practical and complete gene name normalization framework using public data resources and open-source software libraries. Compared with previous related work in BioCreAtIvEII, our framework consists of several functional modules and methodologies and theories that are fully explained, explored and evaluated independently. The paper provides direct insight into designing and building a practical Gene Name Normalization Framework both from an engineering and theoretical point of view. Our framework currently achieves a very promising performance: 0.8 Precision, 0.69 Recall and a F-measure score of 0.74 on the BioCreAtIvE II benchmark, which proves its validity and competency.

1. Introduction

The core of Gene Name Normalization in BioNLP is to identify an extracted gene name in biomedical text, usually by being assigned to an identifier in a biological database. This task is different from general-purpose search engine facilities or string matching in databases because of ambiguous issues in gene name entities: spelling variation of a gene name almost occurs since many authors do not apply standard rules when writing a gene name. Recognition of gene names might not be precise, ambiguous gene names within species, across species, common English words and certain medical diseases may be used. To correctly identify a gene name, related biological knowledge and linguistic methodologies are usually employed to improve performance. In general, gene name normalization can be decomposed into several subtasks:

1. *Gene Name Recognition*: the first crucial task is to extract gene names from plain text. This process is also called Name Entity Recognition (NER) in the Natural Language Processing (NLP) domain. The precision and recall of gene name recognition have a direct impact on the quality of following steps.

2. *Building A Gene Name Database*: a gene name database is used as a lexical resource for gene name retrieval. Since there is no comprehensive gene name dictionary, various gene name dictionaries are required to be compiled together. Manually curation work may be necessary to remove extremely ambiguous gene names (such as 'ORF-1') and to improve the quality of the gene dictionary.

3. *Initial Mapping*: given a recognized gene name entity, this step finds matched gene names recorded in the gene name database. These returned candidates will be used in the next steps. In most cases, naive exact string matching can not achieve a satisfactory result due to spelling variation. Additional processes might be applied to increase recall: effective tokenization strategies, aggressive searching algorithms and a variety of query types like boolean queries for example (do not consider the order of words within a gene name), etc.

4. *Disambiguation*: after the Initial Mapping Step, it is possible that a gene name shares multiple identifiers. For example, candidate 'p54' might have 4 IDs: 1656, 2289, 4841 and 42828. Disambiguation is used to select a correct identifier for this gene name. Gene profile knowledge can filter out unrelated IDs by using semantic similarity. Local context and global context can also improve the precision of disambiguation.

5. *False Positive Filter*: even after disambiguation, it is still possible that a gene name with its identifier is incorrectly mapped because the name entity refers to a protein family, protein complex or disease. A false

positive filter is used to remove such kinds of items. Common methods include building a blacklist, applying regex-pattern-based filter, etc.

The difficulties of Gene Name Normalization mainly stem from two points: 1) gene names can not completely and exactly be recognized; 2) human writers usually do not follow the standard gene naming rules, which makes it hard for program to search for the truly semantically matched gene names in a gene name database. For example, Collagenase and Collagenase inhibitor stands for two different name entities while 'ADP' can be written in the form of 'ADP precursor'. In order to tell whether the word can be ignored or not, biology knowledge needs to be considered. Furthermore, the flexibility of natural language when writing gene names sometimes makes it more difficult to identify full gene names. For example, given the text the receptor ADP and ..., the gene name entity should be receptor ADP. However, the Name Entity Recognition component only extracts ADP as a gene mention, which consequently leads to a false mapping because of the lack of modifier token receptor. As summarized, the Gene Name Normalization task is the inter-discipline of Natural Language Processing (NLP) and Biology, which requires biology background knowledge and natural language processing skills.

With a lot of previous studies on these subtasks [14, 28, 34, 38, 16, 22], we propose a generic framework for Gene Name Normalization task. Various methodologies, tools and resources are incorporated into this framework. Methods and software packages in each module are carefully configured, evaluated and experimented with. Besides considering already existing approaches and systems in Section 2, this article is organized as follows: Section 3 describes in detail the architecture and components of this framework. In Section 4, we deal with the Gene Name Recognition subtask by using dictionary-lookup, Hidden Markov Models and Conditional Random Fields. Experiments were conducted and evaluated in different modes on the BioCreAtIvE II dataset. In Section 5, the mechanism of gene name normalization is minutely explained and final results are given. Section 7 contains the conclusion and an extensive discussion of the results. In Section 8 future work is proposed and discussed.

2. Related Work

The Gene Name Normalization task, generally speaking, consists of 2 specific subtasks: automatic detection of gene names in biomedical text and assigning a biomedical database identifier to the recognized gene name. The former is called Name Entity Recognition (NER) and the latter is called Gene Normalization (GN). Many publications only deal with one specific subtask, where a few papers cover the whole task. Furthermore, many implementation, methodological and configuration issues of systems are unavailable due to proprietary reasons.

For the Name Entity Recognition (NER) problem, methodologies can be roughly categorized into 3 classes: Rule-based, Dictionary-based and Machine Learning-based approaches. Named entity recognition is particularly easy if it is possible to write a regular expression that captures the intended pattern of entities. One easy application is the detection of e-mail addresses in text. However rule-based recognition requires pre-defined regular expressions. For gene name recognition, such an approach is obviously powerless because of the spelling variation of gene names in text. Therefore, Rule-based approaches are not practical and less-used. Another approach is dictionary based lookup. In many applications, it is relatively straightforward to compile a list of names (and aliases) for entities. Given a text, the program will check whether the compiled name appears. The third approach is machine learning method. Hidden Markov Model (HMM) [30] is a classical method in sequence tagging and name entity recognition. Conditional Random Fields (CRFs) [23] also gained success in gene name recognition. Even when using the most comprehensive and up-to-date lexical resources to build gene name dictionary, it is virtually impossible to cover all gene names in text because spelling variations always occur. Machine Learning can generalize the ability of recognition of gene names, so as to improve recall. ABNER [33] is a software tool for molecular biology text analysis. ABNER's core is a statistical machine learning system using linear-chain conditional random fields (CRFs) with a variety of orthographic and contextual features. LingPipe [4] also provides NER functionalities, whose underlying implementation is based on Hidden Markov Model. MALLET [27] is a Java-based package for statistical natural language processing. MALLET includes tools for sequence tagging for applications such as named-entity extraction from text. Algorithms include Hidden Markov Models, Maximum Entropy Markov Models, and Conditional Random Fields. The highest F-score of gene name recognition in BioCreAtIvE I in 2004 is 0.82 and the highest F-score in BioCreAtIvE II in 2006 is 0.87. By far, the state of art

result is 0.87 of Gimli system [7] in 2013.

The Gene Normalization (GN) problem is divided into several steps: initial mapping, disambiguation and filtering. The order of each step can be interchanged. The goal of initial mapping is to measure the similarity of gene names located in articles and the gene name recorded in biological databases such as EntrezGene [25] or Uniprot [3]. Classical distance metrics [8], including Edit Distance, Jaro-Winkler distance and Cosine Distance, have been applied in this step but results yield are not satisfactory. If multiple gene identifiers share the same gene mention, it results in ambiguous mapping. Disambiguation is used to solve this ambiguous mapping. Currently, many systems and papers employ gene profiles to disambiguate. Since the BioCreAtIvE II [28] annotation guidelines on purpose excluded protein families, groups and complexes, it is necessary to filter them out from gene candidates. Many systems build a blacklist to filter out protein families, groups and complexes. In BioCreAtIvE II in 2006, 9 out of 20 groups had a run that achieved an F-measure of 0.75 or better, indicating a significant advance in the state of the art for gene normalization than BioCreate I. Recent work by J. Wermter [38] in 2009 and Y. Hu [17] in 2012 show an advanced in state of art results: 0.86 and 0.83, respectively. However, for intellectual property rights and further development reasons, their implementation details and data sources are not public. Their work either explored one aspect in more detail or used more high quality professional training corpus.

3. System Architecture Overview

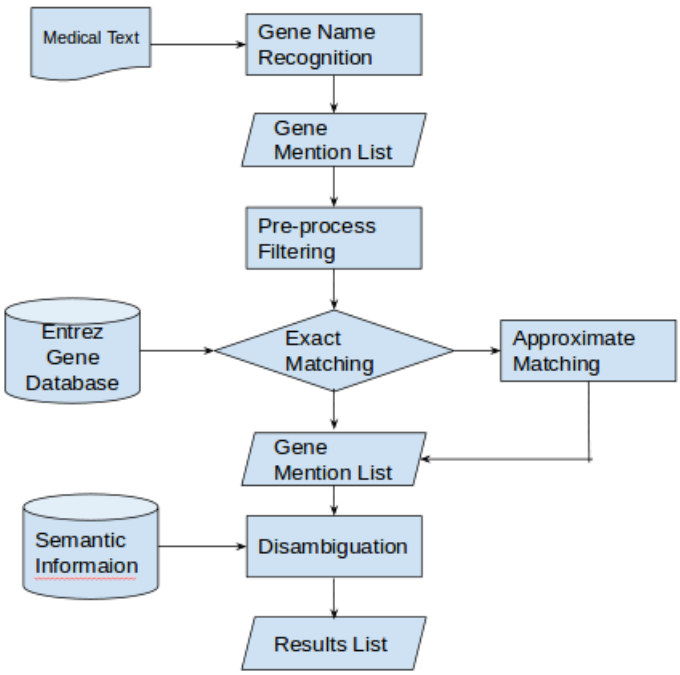

Figure 1. Pipeline of Gene Name Normalization.

In this section, we describe the architecture of our generic gene name normalization framework. Full details of our framework will be given in subsequent sections. The basic pipeline of our framework is displayed in Figure 1. For biomedical plain text as input, the Gene Name Recognition module will automatically extract a list of gene names. Then Pre-filtering module will remove false positive entities. The Initial Mapping module consists of 2 components: An exact matching component and an approximate matching component. The Initial Mapping module returns a ranked list of mapping candidates. If one gene name is assigned to multiple identifiers, disambiguation is performed. Finally, a result list that contains gene names and corresponding genes identified is returned. From a software engineering perspective, each step is designed as an individual, pluggable and configurable module. The gene name database employed is to provide lexicon resources for initial mapping.

Semantic information aims at disambiguating gene mapping pairs.

4. Gene Name Recognition

The goal of Gene Name Recognition is to extract gene names from biomedical text. As explained in the Related Work Section, the common methodologies to deal with this issue are: Dictionary-based method and Machine Learning-based method. In this section, dictionary-lookup, Hidden Markov Models and Conditional Random Fields are discussed and evaluated. Important detailed issues, including time-complexity, selection of low-level features, tagging schemes on corpora and content of training corpora are also discussed.

4.1. BioCreAtIvE II: Gene Mention Task

The Gene Mention Tagging task in BioCreAtIvE II is concerned with the named entity extraction of gene and gene product mentions in text [34]. Training data and test data are both randomly selected from MEDLINE. Those data are given in plain text form. Annotators who have biochemistry, molecular biology and genetics background identified gene mentions in those sentences, giving gold standard annotation. 20,000 sentences have been annotated.

BioCreAtIvE II has been a community-wide accepted benchmark in the BioNLP field. Therefore, all experiments conducted in the section are based on the BioCreAtIvE II benchmark, including the training data set, test data set and evaluation scripts. The evaluation metric is precision, recall and F-score, which is defined below:

$$Recall = TP/(TP + FN)$$

$$Precision = TP/(TP + FP)$$

$$F - score = 2 * Recall * Precision/(Recall + Precision)$$

where TP, FP and FN are true positives, false positives and false negatives respectively. The evaluation measures are kept consistent with those used in the BioCreative-II challenge.

4.2. Dictionary-Lookup Method

The basic idea of Dictionary-lookup is straightforward: storing a list of names (and aliases) for gene name entities, dictionary will extract those gene names that appear in text. In this experiment, LingPipe is utilized. LingPipe provides an implementation of the Aho-Corasick algorithm [1]. The beauty of the Aho-Corasick algorithm is that it finds all matches against a dictionary in linear time independent of the number of matches or size of the dictionary. The lexical resource used to build the dictionary is the official dictionary provided by BioCreAtIvE II (entrezGeneLexicon.list), with 32975 genes and 182989 (non-unique) gene names.

4.3. Machine Learning Method

Machine Learning methods for name entity recognition has achieved promising performance given a large entity tagged corpus in the gene name recognition area. Many recent research studies on machine learning-based Name Entity Recognition(NER) focus on incorporating unique properties of biomedical terminology sources into the powerful machine learning frameworks, where machine learning algorithms that can accommodate rich domain-oriented features are preferred. The availability of open source NER machine learning tools boosts the baseline performance of many NER system. The widely-used machine learning models in NER are Hidden Markov Models (HMMs) and Conditional Random Fields (CRFs). In this section, both machine learning methods are evaluated and used in experiments, using different open source software packages. HMMs are encapsulated in the LingPipe toolkit whereas CRFs are implemented in ABNER and Crf++ toolkit. The BioCreAtIvE II Gene Mention task benchmark is employed to examine the performance.

4.3.1 Hidden Markov Model

HMM [30] is generative model that proved to be very successful in a variety of sequence labeling tasks such as Speech recognition, POS tagging, chunking, NER, etc. A Hidden Markov Model H is a quintuple (S, V, π, A, B) where

1. $S = \{s_1, ..., s_N\}$ is the set of states. N is the number of states. The states of a Hidden Markov Model are hidden; we never observe them directly.
2. $V = \{v_1, ..., v_M\}$ is the vocabulary, the set of symbols that may be emitted.
3. $\pi = \{\pi_1, ..., \pi_N\}$ is the initial probability distribution on the states. It gives the probability of starting in each state, where $\sum_{i=1}^{N} \pi_i = 1$.
4. $A = a_{ij}(i \in S, j \in S)$ is the transition probability matrix. If the 'machine' is in state j, it may be in state i on the next clock tick with probability a_{ij}, where $a_{ij} \in [0, 1]$ and $\sum_{i \in S} a_{ij} = 1$ for each j.
5. $B = b_{ij}(i \in V, j \in S)$ is the emission probability matrix. if the machine is in state j, it may emit symbol i on with probability b_{ij} .

HMM is widely used in computational linguistics for part-of-speech (POS) tagging and name entity recognition (NER). Let $x = (x_1, x_2...x_n)$ be the observation sequence of length n. Let $S = (s_1, s_2...s_m)$ be a finite set each of which corresponds to a biological entity tag. Donate $s = (s_1, s_2...s_n)$ as sequence of biological entity tag for x.

HMM has two assumptions to make the statistical model tractable: Firstly, the current state s_i only depends on the previous state s_{i-1} which is called the Markov property. Secondly, it assumes that each emitted observed token x_i only depends on the current state s_i. With the two assumptions, the joint probability of the entity tag sequence s with the observation sequence x is defined as followed:

$$P(s, x) = \prod_{i=1}^{n} P(x_i|s_i)P(s_i|s_{i-1}) \tag{1}$$

It is possible to assume that the current state depends on two proceeding states(second order HMM), then the joint probability will be defined as followed:

$$P(s, x) = \prod_{i=1}^{n} P(x_i|s_i)P(s_i|s_{i-1}, s_{i-2}) \tag{2}$$

The BaumWelch algorithm [5] is used to find the unknown parameters of a hidden Markov model (HMM). It makes use of the forward-backward algorithm. The forwardbackward algorithm is an inference algorithm for hidden Markov models which computes the posterior marginals of all hidden state variables given a sequence of observations/emissions.

For name recognition task, it can be described: given a sequence of observations, find the best corresponding hidden states sequence. The Viterbi algorithm [9] is a dynamic programming algorithm for finding the most likely sequence of hidden states called the Viterbi path that results in a sequence of observed events, especially in the context of Markov information sources and hidden Markov models.

The basic theory of HMMs is also very elegant and easy to understand. This makes it easier to analyse and develop implementations for. However, it has two evident disadvantages. The first disadvantage is: HMM only captures the dependency between each state(for the first-order HMM, only the previous state and the current state) and *only* corresponding observation. In a sentence segmentation task, not only should each segmentation state depend on the current word, but also rely on other (non-local) features of the whole sentence such as indentation and number of white spaces. The second disadvantage is mismatch between the learning objective

function and prediction objective function. HMM learns a joint probability of states and observations, but in a prediction task, condition probability is preferred. To diminish above shortcomings, the Maximum Entropy Markov Model(MEMM) [6] is proposed to improve HMM. MEMM tries to model the dependency between each state and the *entire* observation sequence, rather than *only* current observation. A drawback of MEMMs is that they potentially suffer from the "label bias problem," where states with low-entropy transition distributions "effectively ignore their observations." Another source of label bias is that training is always done with respect to known previous tags, so the model struggles at test time when there is uncertainty in the previous tag. Conditional Random Fields were designed to overcome this weakness.

4.3.2 Conditional Random Fields

Conditional Random Fields (CRFs) [23, 35] is a rather novel approach that has already become very popular for a great amount of NLP tasks due to its remarkable characteristics. CRFs are indirected graphical models which belong to the discriminative class of models. The principal difference of this approach with respect to HMM is that it maximizes a conditional probability of labels given an observation sequence. This conditional assumption makes it easy to represent any additional feature that a researcher could consider useful. This [29] leads to a great diminution of a number of possible combinations between observation word features and their labels and, therefore, it makes it possible to represent much additional knowledge in the model. In this approach the conditional probability distribution is represented as a multiplication of feature functions exponents.

Feature Function in CRF Taking the POS labeling as an example, in a CRF, each feature function is a function that takes the following factors as input:

1. a sentence x.
2. the position i of a word in the sentence.
3. the label s_i of the current word.
4. the label s_{i-1} of the previous word.

The output of a feature function a real-valued number (or 0 and 1 as an indicator function). So the feature function has the following format:$f(x, i, s_i, s_{i-1})$. The feature function can be defined in any form, for instance:

$$f(x, i, s_i, s_{i-1}) = \begin{cases} 0 & \text{if } x_i \text{ is 'beta' and } s_{i-1} \text{ is 'B' and } s_i \text{ is 'I'.} \\ 1 & \text{otherwise} \end{cases}$$

$$f(x, i, s_i, s_{i-1}) = \begin{cases} 1 & \text{if } x_i \text{ is a number and } s_{i-1} \text{ is 'O' and } s_i \text{ is 'B'.} \\ 0 & \text{otherwise} \end{cases}$$

Although, in general, feature functions can belong to any family of functions, implementation of many software packages often employs the simplest case of binary functions.

Feature to Probability After the feature functions are defined, each feature is assigned a weight w_i to reflect how important this feature is when labeling. These weights can be learned during training phase. Therefore, given a sentence, labels s is scored by adding up weighted features of all words in this sentence:

$$score(s|x) = \sum_{j=1}^{m} \sum_{i=1}^{n} w_j f_j(x, i, s_i, s_{i-1}) \tag{3}$$

Since the score is only a number, transformation from score to probability is performed by exponentiating and normalizing. Because the value of feature is real-value, exponentiating makes score to be positive and normalization makes it to be between 0 and 1. Therefore the probability of label sequence s conditioned on sentence x is:

$$P(s|x) = \frac{\exp score(s|x)}{\sum_{s'} \exp score(s'|x)} \tag{4}$$

Linear-Chain CRF Note that in the definition of the feature function, it is assumed that the feature only depends on the current label and the previous label, not arbitrary labels in a sentence. A CRF with such a restriction is called a 'linear-chain CRF'. In this paper, when talking about CRF, it refers to linear-chain CRFs by default. It is possible to let the feature depend on arbitrary labels in one sentence, but this often leads to a very high time complexity to train. Most of the available CRFs packages only implement linear-chain CRFs.

4.3.3 HMMs vs CRFs

Comparing HMMs and CRFs, there is a strong connection between them: the weighted features in CRFs can be viewed as the transition probabilities and emission probabilities in HMMs. But CRFs outperform both MEMMs and HMMs on a number of real-world tasks in many different fields [10], including bioinformatics, computational linguistics and speech recognition. In contrast to HMMs, the advantages in CRFs are:

1. CRFs use a rich set of features. Since the local nature in HMMs, each labeling only considers the current word itself. While features functions introduce many useful information behind the current word: token shape, orthographic and morphological structure, etc. These features are incorporated into model.
2. CRFs over hidden Markov models is their conditional nature, resulting in the relaxation of the independence assumptions required by HMMs in order to ensure tractable inference.
3. CRFs avoid the label bias problem, a weakness exhibited by maximum entropy Markov models (MEMMs) and other conditional Markov models based on directed graphical models.

4.4. Gene Name Normalization: Experiments and Results

In this section, we described the setups and results of gene name recognition task. These experiments are categorized into two classes: dictionary-lookup and machine learning methods. Specifically, HMMs and CRFs methods are used in the machine learning experiments. Three open source software packages LingPipe[4], ABNER[33] and Crf++[21] are used, which are widely used in academia and industry. Results on BioCreAtIvE II benchmark and analysis are included.

4.4.1 Experiment Setup

For the dictionary-lookup experiment, the first step is building a gene name dictionary based on Lexical Resources. Then given test data, gene names in text will be extracted. For machine learning experiments, the first step is training the statistical model. The model is trained on tagged corpora (training corpora) with different tagging schemes. After the training model is completed, gene names will be automatically recognized. The extracted gene names will be evaluated by using scoring script. If one gene name is correctly recognized, it is considered to be a true positive, otherwise, it is a false positive. The following experiment setup is used:

1. Tools: LingPipe, ABNER and Crf++.
2. Lexical Resource: This is for dictionary-lookup method. The lexical resource used to build dictionary is the official dictionary provided by BioCreAtIvE II(entrezGeneLexicon.list), with 32975 genes and 182989 (non-unique) gene names.

3. Training Corpus: This is for machine learning method to train model. Training corpus are from BioCreAtIvE II, 20,000 annotated sentences selected from MEDLINE randomly are released as training data.

4. Tagging Scheme: Three software packages in experiment follow different tagging schemes, which will be explained in their own experiment section.

5. Evaluation: System performance will be scored automatically by how well the generated gene/gene product list corresponds to one generated by human annotators. Acceptable alternatives to the gold standard names, also generated by human annotators, will count as true positives. The evaluation script provided by BioCreAtIvE II is used.

4.4.2 LingPipe

LingPipe [4] is a toolkit for processing text using computational linguistics. LingPipe has been used to do tasks like: finding the names of people, organizations or locations in news, automatically classify Twitter search results into categories, suggesting correct spellings of queries. LingPipe provides a Java API for developers who want to integrate LingPipe utilities into systems. LingPipe only needs to annotate actual gene mentions. LingPipe provides two interfaces for dictionary-lookup and the first-order HMM, respectively.

Dictionary-lookup LingPipe gives an interface to operate dictionary-lookup method. The first step is to build a dictionary. LingPipe reads lexical resource file's line by line and stores gene names internally to construct a dictionary. When testing, LingPipe reads one sentence at a time and extracts gene names in sentence if a recorded gene name is matched.

HMM Training LingPipe also provides a functional interface for the HMM. Training the HMM in LingPipe is also straightforward. Given a large tagged corpus, LingPipe automatically computes the initial state probability distribution, transition probability matrix and emission probability matrix of the HMM. A tagging scheme in LingPipe is illustrated as below:

"*Input: Comparison with alkaline phosphatases and 5-nucleotidase.*"
"*Tags: Comparison_TAG with_TAG alkaline_GENE1 phosphatases_GENE1 and_TAG 5-nucleotidase_GENE2.*"

The first line is the original sentence while the second line is the tagged sentence. Each token is followed by '*_TAG*', '*_GENE1*' and '*_GENE2*' where *_TAG* means it is not a gene name, *_GENE1* and *_GENE2* means it is a gene entity. The index *1* and *2* just works as indicators to distinguish two different but adjacent gene names.

4.4.3 ABNER

ABNER [33] is a software tool for molecular biology text analysis. It began as a user-friendly interface for a system developed as part of the NLPBA/BioNLP 2004 Shared Task challenge. At ABNERs core is a statistical machine learning system using linear-chain conditional random fields (CRFs) with a variety of orthographic and contextual features. Version 1.5 includes two models trained on the NLPBA and BioCreative corpora, for which performance is competent (roughly 0.8 F-score on). The new version also includes a Java API allowing users to incorporate ABNER into their systems, as well as train and use models for other data.

Training CRF It is easy to train CRF model, merely the tagged corpus is required. ABNER will read the corpus, tokenize sentence and automatically extract low-level features, constructing training feature template, then training CRF model. ABNER follows the BIO tagging scheme. B means the beginning of a gene, I refers to inside a gene and O stands for outside a gene. An example tagged sentence used by ABNER is shown as below:baldwin2003lingpipe

"*Input: Comparison with alkaline phosphatases and 5-nucleotidase.*"
"*Tags: Comparison|O with|O alkaline|B phosphatases|I and|O 5-nucleotidase|B.*"

The first line is the original sentence and the second line is the tagged sentence. Each token is followed by a tag, separated by a vertical line |. There are three kinds of tags: B, I and O. The features in CRFs are important to improve performance during training. However, feature selection is always a challenging task. Fortunately, ABNER can generate feature sets for users. ABNERs default feature set comprises orthographic and contextual features, mostly based on regular expressions and neighboring tokens. The feature set is slightly modified from previous work (Settles, 2004) for improved performance, and can be viewed/modified in the source code distribution. Note that ABNER currently does not use syntactic or semantic features. Research indicates that such features can improve performance slightly, but presently they are not dynamically generated by ABNER.

4.4.4 CRF++

CRF++ [21] is a simple, customizable, and open source implementation of Conditional Random Fields (CRFs) for segmenting/labeling sequential data. CRF++ is designed for generic purpose and will be applied to a variety of NLP tasks, such as Named Entity Recognition, Information Extraction and Text Chunking. As the name suggests, it is written in C++.

Training CRF It is a little complicate to use CRF++ when training the model. Unlike ABNER, users should define the features by themselves. That is to say, users have to define features on the tagged corpus before training. Both the training file and the test file need to be in a particular format for CRF++ to work properly. Generally speaking, training and test file must consist of multiple tokens. In addition, a token consists of fixed-number of columns. The definition of tokens depends on tasks, however, typically, they correspond to words. Each token must be represented in one line, with the columns separated by white space (spaces or tabular characters). A sequence of tokens becomes a sentence. To identify the boundary between sentences, an empty line is put. An example of training file format for CRF++ is:

Input: "*Comparison with alkaline phosphatases and 5-nucleotidase.*". Feature Set is below:

Comparison	comparison	NN	SingleCap	O
with	with	IN	NoCap	O
alkaline	alkaline	NN	NoCap	B
phosphatases	phosphatas	NNS	NoCap	I
and	and	CC	NoCap	O
5	5	CD	NoCap	B
-	-	HYPH	NoCap	I
nucleotidase	nucleotidase	NN	NoCap	I

The first line is original sentence. Below are tokenized sentence with features. As described, the first column is the original token, from the second column to the fourth column, they are extracted features for this token. The last column is the tag. The tagging scheme is BIO, which is described in ABNER. The usage of CRF++ is summarized as follow:

1. Tokenize sentences. In our experiment, Gennia Tagger [37] is applied as the tokenizer. Gennia Tagger is a successful tool for biomedical text analysis. Besides tokenization, it also generates part-of-speech and stemmed word for each token. These can be used in the 'Extracting Feature' step.

template	expanded feature
%x[0,0]	Comparison
%x[0,1]	comparison
%x[0,2]	NN
%x[0,3]	SingleCap

Table 1. Feature Template file explanation: when the current location is 0, namely, the first token 'Comparison', Row specfies the relative position from the current focusing token and col specifies the absolute position of the column.

2. Extract Features. Feature plays an important role in CRF. Based on the previous work by [26, 22, 16], common orthographic features: stemmed word, word length, vowel in word etc are extracted. Biological domain related features are also included: names of nucleic acids, nucleosides, nucleotides, residues of amino acids etc. The complete feature set is explained in Table 8.

3. Preparing Feature Template. The purpose of template file in CRF++ is to determine which features are used in training and test. In each template, special macro $\%x[row, col]$ will be used to specify a token in the input data. Row specfies the relative position from the current focusing token and col specifies the absolute position of the column. The purpose of template file is to derive new features at a higher level, generating the combination of low-level features at different locations within one sentences. An example of feature template file is in Table 1.

4.4.5 Results and Discussion

Tool	Statistical Model	Training Set	TP	FP	FN	Precision	Recall	F-measure
Dictionary-based	-	BioCreAtIvEII	2498	1322	3833	0.65	0.39	0.49
LingPipe	HMM	BioCreAtIvEII	4624	1550	1707	0.74	0.73	0.73
ABNER	CRF	BioCreAtIvEII	4725	848	1606	0.84	0.74	0.79
Crf++	CRF	BioCreAtIvEII	5219	553	1112	0.90	0.82	0.86

Table 2. Gene Name Recognition results by LingPipe, ABNER and Crf++ on BioCreAtIvE II benchmark.

Tool	Statistical Model	Training Set	TP	FP	FN	Precision	Recall	F-measure
ABNER	CRF	GeneTag	3968	697	2363	0.85	0.62	0.72
ABNER	CRF	BioCreAtIvEII	4531	1825	1800	0.71	0.71	0.71
LingPipe	HMM	GeneTag	5074	2696	1257	0.65	0.8	0.71
LingPipe	HMM	BioCreAtIvEII	5074	2664	1257	0.65	0.8	0.72

Table 3. Gene Name Recognition Results using different corpora.

Combination	Training Set	TP	FP	FN	Precision	Recall	F-measure
CRF++ ∩ LingPipe ∩ ABNER	BioCreAtIvEII	3117	144	3214	0.95	0.49	0.64
CRF++ ∪ LingPipe ∪ ABNER	BioCreAtIvEII	5904	4058	427	0.59	0.93	0.72
(CRF++ ∩ LingPipe) ∪ ABNER	BioCreAtIvEII	5232	990	1099	0.84	0.82	0.83
CRF++ ∪ (LingPipe ∩ ABNER)	BioCreAtIvEII	5444	710	887	0.88	0.85	0.87
(CRF++ ∩ ABNER) ∪ LingPipe	BioCreAtIvEII	5564	3368	767	0.62	0.87	0.72

Table 4. Different Combination of Gene Name Recognition Results

The benchmark results are displayed in Table 2. From Table 2, it is shown that one disadvantage of the dictionary-lookup method for gene name recognition is the low recall. The reason for low recall is that spelling variation often occurs in biomedical text. For example, changing number '1' to Roman numeral 'I'. Even though a comprehensive and up-to-date dictionary is used, it can not cover all variation of gene names. Therefore, the way to boost performance in dictionary-lookup is to improve the quality of dictionary: more gene name, more synonyms and more spelling variation of gene names. Manually curation is also required to remove the extreme ambiguous gene names, for examples, 'hypothesis protein', 'ORF-1'.

Compared with the result of dictionary-lookup, the machine learning methods show a better performance for generalization in recognizing gene name. The HMM method boost precision and recall significantly from 0.65 to 0.74, 0.39 to 0.73, respectively.

The CRF method is the winner in the conducted experiments. The best F-measure score is 0.86, which is almost the best score for Gene Name Recognition in BioCreAtIvEII[34] and state of art result [7]. Even though ABNER and CRF++ both implement CRF model, the results are still not the same. This is because ABNER uses its default feature set while CRF++ allows customizable features. This difference implies that a high dimensional feature set usually result in a better performance.

The difference in training time between HMM and CRF method is manifest: it takes 33 minutes to train CRF mode both in ABNER and CRF++ while training the HMM only takes 10 seconds. The time complexity of training CRF is relatively high, which is a primary drawback of CRF.

To further improve the performance for the Gene Name Recognition, additional experiments are conducted. We assume that the more corpus provided, the more precise model will be. With more corpora, transition matrix and emission matrix in HMM should be more precise, and the feature weights in CRF also should be accurate. The additional corpus used is GeneTag [36] by NCBI. The results are in Table 3.

In ABNER, the F-measure score is increased by 1%. More specific, the precision is boosted from 0.71 to 0.85 while the recall is decreased to 0.62. Conversely, the F-measure score in LingPipe is reduced by 1% because false positive increased a slightly. Comparing GeneTag with BioCreAtIvEII corpus, GeneTag provides the whole labeled abstract as training unit while BioCreAtIvEII is using one stand alone labeled sentence as training unit. This shows that CRF indeed makes more use of global features to improve precision. However, the different structures and contents in corpus have influenced CRF's feature weights to a certain extent. Furthermore, different machine learning models capture features from different perspectives.

We also conducted a series of combinations of result sources, to test whether it can generate an optimized result. These combinations are displayed in Table 4. Combination indeed slightly increases F-score by from 0.86 to 0.87.

5. Gene Name Normalization

Normalization here means identification of an extracted gene name in plain text by being assigned to a gene identifier in biological database. Compared with general-purpose search engine facilities, this task is a challenge because of widespread gene name ambiguities within species, across species, using common English words and other confusing medical terms. Related biological domain knowledge, as well as linguistic methodologies, should be incorporated into this process in order to obtain satisfactory results. These subtasks are listed:

1. *Building A Gene Name Database*: the gene name database is used as lexical resource for gene name retrieval. Since there is no comprehensive gene name dictionary, various gene name dictionaries are required to be compiled together. Manually curation work might be done such as removing extremely ambiguous gene names ('ORF-1'), to improve the quality of gene name dictionary.

2. *Initial Mapping*: given a recognized gene name entity, this step finds matched gene names recorded in gene name database. These returned candidates will be used in the next steps. In most cases, naive exact string matching can not achieve a satisfactory result due to spelling variation. Additional processes might be applied to increase recall: effective tokenization strategies, aggressive searching algorithms and a variety of query type like boolean query (do not consider the order of word within a gene name) etc.

3. *Disambiguation*: after Initial Mapping step, it is possible that a gene name shares multiple identifiers. For example, candidate 'p54' might have 4 IDs: 1656, 2289, 4841 and 42828. Disambiguation is used to select a correct identifier for this gene name. Gene profile knowledge can filter out unrelated IDs by using semantic similarity. Local global context can also improve the precision of disambiguation.

4. *False Positive Filter*: even after disambiguation, it is still possible that this gene name with its identifier is an incorrect map because the name entity refers to a protein family, protein complex or disease. The False Positive Filter removes such kind of items. Common methods include building blacklist, regular pattern filter, etc.

5.1. Integrating Various Gene Name Sources

The result of the Gene Name Recognition task is very significant to the following steps. Since a wide range of methods and software packages are utilized in the Gene Name Recognition task, a hybrid combination method is proposed, to boost recall and precision. Besides CRF++, ABNER and LingPipe tools, a dictionary based gene name recognition component is tested in the Gene Name Recognition task. The final result set is combined as followed: any gene mention appears above (including) twice among CRF++, ABNER, LingPipe and dictionary based result sets, is voted into the final result. The final Gene Mention List = (CRF++ $\cap$ ABNER) $\cup$ (CRF++ $\cap$ LingPipe) $\cup$ (LingPipe $\cap$ ABNER) $\cup$ (CRF++ $\cap$ Dictionary) $\cup$ (ABNER $\cap$ Dictionary) $\cup$ (LingPipe $\cap$ Dictionary). Here, intersection set means those gene names that appear both in two result sets. Union set refers to collection of these intersection result sets.

5.2. Building Gene Name Database

A Gene Name database is a fundamental component in Gene Normalization (GN) task. It returns all matched gene name candidates. The quality of a gene name database determines the performance of GN task to a large extent. The gene name lexicon resource used is the official EntrezGene lexicon list provided by BioCreAtIvE II. Two important issues must be taken into account when building the gene name database:

1) the indexing and retrieval system used to store gene names.

2) the tokenization strategy when compiling gene name resources.

5.2.1 Retrieval Facility

There are many popular and open source relational database such as MySQL and PostgreSQL. They are widely used today in academia and industry. However, these RDBMS are not very suitable for the GN task because full text indexing in those RDBMS is not well supported. The 'LIKE' keyword in an SQL query is rather inefficient when applying to approximate searching. In our framework, Lucene [15] is deployed as gene name retrieval component.

Lucene Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform. Lucene supports a wide range of query types: term query, boolean query and fuzzy query etc. All tokenized gene mentions and its gene identifier are stored in Lucene as a document. The Gene profile score for each gene name is also computed by employing Lucene facilities. Besides fast indexing and searching, one advantage of using Lucene is that it provides Term Frequency / Inverse Document Frequency (TF-IDF) score-ranked result. The ranked score can be used for further analysis. Each gene mention found in text will be queried against Lucene and a ranked list will be returned as results. Since the actual word order within the gene name might affect query result (for instance, 'IL-2 receptor' can also be written as 'receptor of IL-2'), a boolean query in Lucene is used. In boolean query, Lucene only checks the meaningful overlapped tokens, stop words and word order do not play any role.

5.2.2 Tokenization Strategy

Inspired by [38], each gene name is stored in a customized tokenized format in Lucene. This means 'ILBeta-2' is transformed into 'il beta 2' when storing into Lucene. When Lucene is searching relevant gene names, overlap of tokens within a gene name is a significant factor to measure similarity. Actually, Lucene has its own tokenization strategy: a document is tokenized by white spaces and other punctuation marks. But in our application scenario, each gene name as a document is relatively short. For example, there is no any punctuation or white space in 'ILBeta-2', which actually consists of three meaningful tokens: 'IL', 'Beta' and '2'. Therefore a customized tokenization strategy is applied. After tokenization, each biological token is separated by white space. Then Lucene's default tokenization scheme will be enforced to apply, building indexes on each token rather than only the entire word. Experiments show this customized tokenization strategy indeed boosts recall. Another noticeable factor is the 'stop word' in a gene name. In the gene name 'receptor of the ADP', the stop words are 'of' and 'the'. These stop words are usually meaningless in a string and semantic level. Removing stop words can help limit and narrow down the returned results, which makes searching drastically faster. The customized tokenization rules inspired by [38] to be applied include:

1. remove special characters: dash, underscore etc.
2. remove stop words in English
3. if a lower case letter is followed by an upper case letter, separate them. For example: 'sIL' becomes 2 tokens: 's' and 'IL'.
4. if a word is followed by a digit, a greek letter(such as 'alpha') or a roman numeral(like 'VII'), separate them. For example, 'ILbeta' becomes 2 tokens: 'IL' and 'beta'.
5. roman numerals are converted into digits. (For example, 'II' becomes '2').
6. each token is converted to be lower case.

5.3. Pre-process Filter

Before searching relevant gene names, a pre-process filtering is performed to check the validity of recognized gene names. This is because erroneously identified or extremely ambiguous gene names always introduce noise in the initial mapping procedure, consequently adding more false positive mappings. The goal of pre-process filtering is to remove these 'troublesome' gene mentions.

Rule-based Filter Here 5 filtering rules are performed:

1. Expand abbreviation: 'HAP2/4' becomes 'HAP2' and 'HAP4'.
2. If the length of gene mention is less than 2, remove it.
3. If the gene mention is in protein complex BlackList, remove it.
4. If the gene mention is a number or a common English word, remove it.

Regular Expression Pattern Filter Sometimes the recognized gene name is too ambiguous and nonsensical, it is better to remove them in this step. For example, 'hypothetical protein precursor' is recognized as a gene name. In a literal meaning, we can know that the writer just assumes it is a hypothetical gene. There is no need to identify this expression further. This tool is using summarized regular expressions [24] to filter out a list of highly ambiguous and nonsensical terms. The regular expressions representing those literally nonsensical gene names are included as below. In these regular expressions: '.' matches any single character; '[]' matches a single character that is contained within the brackets; '()' defines a marked subexpression; '*' matches the preceding element zero or more times; '+' matches the preceding element one or more times; '?' matches the preceding element zero or one time.

1. $[0-9.]$+k?aa\s*long hypothetical proteins? (For example: '202aa long hypothetical protein')
2. $[0-9.]$+k proteins?
3. $[0-9.]$+k?
4. hypothetical protein precursors?
5. unnamed protein products?
6. (conserved|conserved hypothetical|expressed|hypothetical| hypothetical conserved|novel|predicted|putative|putative exported| unknown)(polyproteins?| proteins? | orfs?)? (For example: 'conserved proteins')

Local Context Filtering Besides these above rules, a Local Context Filtering method [13] is used. Local Context Filter method makes use of the contextual environment (sentence in which the gene mention is founded or its neighbor words) to filter false positive recognized gene mentions. A context environment usually refers to the neighboring words of a gene mention. For instance, A gene mention 'p65' is detected by the Name Entity Recognizing component, if the word before 'p65' is 'cell', this mention more likely refers to a cell line or a cell type; if the word prior to 'p65' is 'mouse', it indicates that it is a mouse gene. Furthermore, we can extend gene mentions to include its next word and form a new gene name. This new gene name will be looked up in Lucene: if it exists, the original gene tends to be a false positive one, while the extended gene name is more likely to be correct. This method can reduce false positive and increase true positive.

5.4. Initial Mapping

The goal of entity mapping is to measure the similarity of the gene name located in its context and the term recorded in the biological database. Classic methods for similarity computation based on word overlap such as edit distance, Jaro-Winkler distance and TF-IDF yield acceptable results. In this test, each gene mention is queried and the top 10 most similar gene names are returned as candidates. For each candidate, we apply the exact string matching and approximate matching to select candidates.

5.4.1 Exact matching

If the detected gene name is identical to the gene name in bio-database, they are considered to be exactly matched. However, such a restricted matching usually leads to a low recall. Traditional exact matching measures two strings at surface character level. In this particular scenario, gene name matching often requires the semantically matched. As discussed above, it does not make sense to take stop words within a gene name into consideration when performing matching. Only those meaningful tokens represent the true content of a gene name. Therefore, it is reasonable to choose meaningful tokens, dropping unrelated tokens, when performing exact matching. In the gene mapping phase, we defined 'exact matching' at token level. Here are rules [17] to redefine 'exact matching' in mapping phase:

1. If a lower case letter is followed by an upper case letter, separate them.
2. If a word is followed by a digit, a greek letter(such as 'alpha') or a roman numeral(like 'VII'), separate them.
3. The matching is case insensitive, any punctuation marks(e.g., space, hyphen), any English stop word(e.g., a, the) is removed.

These rules could improve the coverage of matching. For example, the term 'NF-Kappa B' and its spelling variations: 'NF Kappa B', 'NF kappa B' and 'NF kappaB' are all considered to be exactly matched.

5.4.2 Approximate matching

However, exact matching alone is not able to cover all of the gene mention variants. Spelling variations always occur in articles. The first question is how to measure the similarity between two strings. Several classic methods for similarity computation are examined.

Jaro-Winkler distance Jaro-Winkler distance [19], an extension of Jaro-distance, was firstly used to measure the similarity between peoples name. Jaro-Winkler distance uses a prefix scale p which gives more favourable ratings to strings that match from the beginning for a set prefix length. 0 means two strings are totally different and 1 means identical. The Jaro distance d_j of two given strings s_1 and s_2 is :

$$d_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases} \tag{5}$$

where m is the number of matching characters and t is the half number of transpositions. Two characters from s_1 and s_2 respectively, are considered matching only if they are the same and not farther than

$$\lfloor \frac{max(|s_1|, |s_2|)}{2} \rfloor \tag{6}$$

Each character of s_1 is compared with all its matching characters in s_2. The number of matching (but different sequence order) characters divided by 2 defines the number of transpositions. For example. in comparing 'CRATE' with 'TRACE', only 'R' 'A' 'E' are the matching characters, i.e. m=3. Although 'C', 'T' appear in both strings, they are farther than 1, i.e., floor(5/2)-1=1. Therefore, t=0 . In 'DwAyNE' versus 'DuANE' the matching letters are already in the same order D-A-N-E, so no transpositions are needed. Jaro-Winkler distance uses a prefix scale p which gives more favourable ratings to strings that match from the beginning for a set prefix length ℓ. Given two strings s_1 and s_2, their JaroWinkler distance d_w is:

$$d_w = d_j + (\ell p(1 - d_j)) \tag{7}$$

where

1. d_j is the Jaro distance for strings s_1 and s_2

2. ℓ is the length of common prefix at the start of the string up to a maximum of 4 characters

3. p is a constant scaling factor for how much the score is adjusted upwards for having common prefixes. p should not exceed 0.25, otherwise the distance can become larger than 1. The standard value for this constant in Winkler's work is p = 0.1

Edit distance The Edit distance is also a metric to measure the similarity between two strings. The edit distance between two words is the minimum number of single-character edits (insertion, deletion, substitution) required to change one word into the other. The phrase edit distance is often used to refer specifically to Levenshtein distance [11]. Mathematically, the Levenshtein distance between two strings a, b is given by $lev_{a,b}(|a|, |b|)$ where

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0 \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + [a_i \neq b_j] \end{cases} & \text{otherwise} \end{cases} \tag{8}$$

Note that the first element in the minimum corresponds to deletion (from a to b), the second to insertion and the third to match or mismatch, depending on whether the respective symbols are the same.

Cosine distance The Cosine similarity [18] is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0 is 1, and it is less than 1 for any other angle. Cosine similarity then gives a useful measure of how similar two documents are likely to be in terms of their subject matter. The cosine of two vectors can be derived by using the Euclidean dot product formula. Given two vectors of attributes, A and B, the cosine similarity, $cos(\theta)$, is represented using a dot product and magnitude as

$$cos(A,B) = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \tag{9}$$

Note that cosine distance measures the similarity between two vectors, not two strings. Therefore tokenization is performed to convert a gene name into a vector, then start computing cosine distance.

Token Metric Matching We propose an approximate matching based on the Token Metric. The basic idea is that the gene pair only differs in one or two tokens, some substituted tokens can be ignored while other substituted tokens results in different semantic meaning. For instance, 'ABC' and 'ABC precursor' are the same name entity because many writers tend to omit 'precursor', just for simplicity. However, 'collagenase' and 'collagenase inhibitor' represent completely two distinguishing meaning. If the true positive probability of those substituted tokens can be calculated, we can measure the gene candidate pair in a quantitative way.

1. Training Model. The training data provided by BioCreAtIvEII are used to build model. We firstly run our NER component and extract gene names from training set articles, and employing Lucene to return a list of candidate pairs. If one candidate pair is correct, it is *positive*, otherwise, it is *negative*.

2. Calculate Metric. Each pair contains < detected-gene-name, dictionary-gene-name >. Alignment operation is performed. For example, <'AC', 'ADC'> will be aligned as <'A_C', 'ADC'>(here '_' represents a place holder). After alignment, we calculate the true positive probability of such a substitution(from '_' to 'D').

3. Scoring. If the substitution probability is beyond one threshold (e.g. 0.5), this pair is considered to be *true*.

	EMPTY	precursor	inhibitor	alpha	A
EMPTY	1	1	0.1	0.2	0.2
precursor	1	1	0	0	0
inhibitor	0.1	0	1	0	0
alpha	0.2	0	0	1	0.7
A	0.2	0	0	0.7	1
.		.	.	.	.
.		.	.	.	.
.		.	.	.	.

Figure 2. Illustration of Token Metric.

The core of Token Metric idea is to tell whether the different words appearing in a gene name are important or not. For example, 'ADP' and 'ADP receptor' are two completely different name entities while 'AB' and 'AB precursor' are exchangeable. This difference usually depends on human writer's habit or some conventions. If a comprehensive Token Metric can be constructed, it can tell whether two gene names are semantically identical by judging the different words are exchangeable or not. Currently, we construct the Token Metric on the BioCreAtIvE II official training corpus.

5.5. Disambiguation

After the procedure of initial mapping, if one unique identifier is found for a gene, this identifier is automatically assigned. If a gene name is mapped to multiple gene identifiers, disambiguation is required to filter out false positive identifiers at semantic level.

Gene Profile Each gene has a corresponding profile to represent its background knowledge, distinguishing from others. A gene profile usually consists of its gene summary (from EntrezGene[25]), chromosome location, GO term, key words and protein functionality (from UniProt[3]) etc. Compiling this knowledge from various gene databases is often required. After a gene profile is constructed, the whole text in which that gene mention occurs will be queried against its candidate's gene profile in Lucene, candidates with the highest TF-IDF score (if exceeds a certain threshold) will be elected as the winner.

Profile Compilation As mentioned above, a gene profile consists of several features: gene summary, chromosome location, GO term, key words and protein functionalities. However, features can not be obtained from a single resource, e.g., EntrezGene database. In this paper, gene profile features are collected from a wide range of sources: besides from EntrezGene, GO terms are collected from Gene Ontology [2], key words and protein functionalities and comments on this gene are assembled from UniProt. These features are represented as 'bag-of-words' in natural language [38] form and stored in Lucene.

TF-IDF Score Once the gene profile is constructed, the text in which gene mention occurs is queried against the profile. TF-IDF score is returned as the semantic similarity score. The Term Frequency-Inverse Document Frequency (TF-IDF), is a numerical statistic which reflects how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval and text mining. The TF-IDF value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to compensate for the fact that some words are generally more common than others.

TFIDF [31] is the product of two statistics, term frequency and inverse document frequency. Various ways for determining the exact values of both statistics exist. In the case of the term frequency tf(t,d), the simplest choice is to use the raw frequency of a term in a document, i.e. the number of times that term t occurs in document d. If we denote the raw frequency of t by f(t,d), then the simple tf scheme is tf(t,d) = f(t,d).

1. boolean "frequencies": tf(t,d) = 1 if t occurs in d and 0 otherwise;
2. logarithmically scaled frequency: tf(t,d) = log (f(t,d) + 1);
3. augmented frequency, to prevent a bias towards longer documents, e.g. raw frequency divided by the maximum raw frequency of any term in the document:

$$tf(t,d) = 0.5 + \frac{0.5 \times f(t,d)}{max\{f(w,d) : w \in d\}} \tag{10}$$

The inverse document frequency is a measure of whether the term is common or rare across all documents. It is obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient.

$$idf(t,D) = log\frac{|D|}{|d \in D : t \in d|} \tag{11}$$

where

1. $|D|$: cardinality of D, or the total number of documents in the corpus
2. $|\{d \in D : t \in d\}|$: number of documents where the term t appears (i.e., tf(t,d) $\neq$ 0). If the term is not in the corpus, this will lead to a division-by-zero. It is therefore common to adjust the formula to 1 + $|\{d \in D : t \in d\}|$.

Mathematically the base of the log function does not matter and constitutes a constant multiplicative factor towards the overall result. Then tfidf is calculated as

$$tfidf(t,d,D) = tf(t,d) \times idf(t,D) \tag{12}$$

A high weight in TF-IDF is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents; the weights hence tend to filter out common terms. Since the ratio inside the IDF's log function is always greater than or equal to 1, the value of IDF (and TF-IDF) is greater than or equal to 0. As a term appears in more documents, the ratio inside the logarithm approaches 1, bringing the IDF and TF-IDF closer to 0.

5.6. False Positive Filter

It is still possible that the candidate mappings after disambiguation are incorrect. Specifically, the BioCreAtIvE II benchmark annotation guidelines [28] on purpose excluded protein families, groups and complexes. A Blacklist Filter is constructed to discard such unwarranted gene mentions.

Building A BlackList There does not exist a single comprehensive and up-to-date resource to include all protein families, groups and complexes. Therefore this list is compiled out of several publicly available on-line resources. Previous work [38, 17] employed this method to remove false positives. All terms under 'Multiprotein Complexes' are selected. Secondly, the protein family and complexes list are augmented by using the GENIA corpus [20]. The primary annotated resource created in the GENIA Project is the GENIA corpus, which consists of multiple layers of annotation, encompassing both syntactic and semantic annotation. The GENIA technical term annotation covers the identification of physical biological entities as well as other important terms. The corpus annotation covers the full 1,999 abstracts of the primary GENIA corpus. Specifically, terms tagged as 'protein_family_or_group' and "protein_complex' are parsed and extracted.

Protein Family Auto-Filter From the Peregrine system [32], it is noted that some gene synonyms in the gene database or extracted from text are actually family names and therefore should be removed [32]. In the Peregrine system, for a term, it searches the whole dictionary to check whether this term is followed by a number, a roman numeral or a greek letter. For instance, 'Zinc finger protein' is also detected as a substring in 'Zinc finger protein 51', and is therefore removed.

Since the dictionary employed in Peregrine is manually curated, in our framework, the search scope is narrowed down. It only searches in all the recognized gene names of this text. In one text, if there are several extracted gene names share the same prefix term and this prefix term is also recognized as a gene name, this prefix term is considered as protein family. It finds the potential protein family automatically and efficiently.

6. Results and Analysis

All above functional components are employed in our framework. In the initial mapping phase, exact matching and appropriate match are experimented, respectively. In the disambiguation step, different matching profile thresholds are tested. Experiments of classic methods and Token Metric method are also included. The best final combined results are also displayed.

The results of exact matching are displayed in Table 5. All rules defined in exact matching are applied. The threshold for profile matching is tested under 0.1, 0.3 and 0.9. Results show that 0.3 is good trade off between precision and recall.

Methods	TP	FP	Precision	Recall
exact match(Rule1,2,3) + profile(>0.1)	535	133	0.8	0.68
exact match(Rule1,2,3) + profile(>0.3)	506	111	0.82	0.64
exact match(Rule1,2,3) + profile(>0.9)	479	101	0.82	0.61

Table 5. Exact Match with Gene Profile Score.

Several classic methods are discussed in appropriate matching section. Therefore, it is interesting to examine their performance. These results are shown in Table 6. Jaro-Winkler and Edit distance metric are not suitable for measuring the similarity between gene names. because they compare gene names at surface string level, by character to character. As [8] pointed that the use of such string distances is most useful for matching problems with little prior knowledge. Therefore, comparing gene names at semantic level, by token to token, can yield acceptable results. The Token Metric indeed increases recall and precision, compared with other methodologies.

Methods	TP	FP	Precision	Recall
approximate match(Cosine Distance>0.8) + profile(>0.1)	40	81	0.33	0.05
approximate match(Cosine Distance>0.9) + profile(>0.1)	13	9	0.59	0.01
approximate match(Cosine Distance>0.95) + profile(>0.1)	10	4	0.71	0.01
approximate match(Edit Distance>0.8) + profile(>0.1)	10	43	0.18	0.01
approximate match(Edit Distance>0.9) + profile(>0.1)	3	7	0.30	0.00
approximate match(Edit Distance>0.95) + profile(>0.1)	0	0	0.00	0.00
approximate match(Jaro-Winkler Distance>0.8) + profile(>0.1)	35	162	0.17	0.04
approximate match(Jaro-Winkler Distance>0.9) + profile(>0.1)	14	91	0.13	0.01
approximate match(Jaro-Winkler Distance>0.95) + profile(>0.1)	9	57	0.13	0.01
approximate match(Token Metric confidence>0.7) + profile(>0.1)	30	18	0.62	0.04
approximate match(Token Metric confidence>0.8) + profile(>0.1)	10	8	0.55	0.01
approximate match(Token Metric confidence>0.9) + profile(>0.1)	3	1	0.75	0.01

Table 6. Approximate Match with Gene Profile Score.

The final result, shown in Table 7, consists of exact matching (applying rule 1,2,3), appropriate matching (using Token Metric (threshold 0.7) or Cosine Distance (threshold 0.95)), gene profile disambiguation (threshold is 0.3). False Positive Filter is also turned on.

Methods	TP	FP	Precision	Recall
exact match(Rule1,2,3) + approximate match(cosine distance>0.95) + profile(>0.3)	543	136	0.80	0.69
exact match(Rule1,2,3) + approximate match(Token Metric confidence>0.7) + profile(>0.3)	549	150	0.78	0.70

Table 7. Final Result on Gene Name Normalization.

During the experiment, we also extended the gene name database by adding gene names and other synonyms from UniProt. We believe that a larger gene name dictionary should boost recall, consequently promoting overall performance. However, an interesting result we got is that recall is increased by merely 2%, whereas precision is decreased by 5%. We checked the UniProt data and found that the records in UniProt contain too many irregular, extremely ambiguous names and synonyms. Too many identifiers share one name which introduces more ambiguous mapping pairs.

7. Conclusion and Discussion

The final best F-measure score we got in our framework is 0.74, whereas the best F-measure score in BioCreAtIvE II workshop is 0.81. This is a clear indication of the quality of our framework. The whole framework we designed and implemented is built on open source libraries, pluggable and configurable.

In Gene Name Recognition task, so far, the state of art result is 0.87 [7] in 2013. The gene name recognition component in our framework achieves 0.86 by making a combination of various tools, augmenting more related biological features and modifying feature template in CRF, which is comparable to the state of art.

In Gene Normalization task, the best score in BioCreative-II is 0.81, while our final result is 0.74, ranking the 9th position among 20 teams. Recent work [38] in 2009 and [17] in 2012 shows an advanced in state of art results: 0.86 and 0.83, respectively. Their work either explored one aspect in more detail or used more high quality professional training corpus.

Biocreative-III on 2010 also contains Gene Normalization task, however, with some different task requirements and description. Its evaluation method employs TAP (Threshold Average Precision), taking the result ranking score into consideration. Moreover, input document is the whole biomedical article, rather than merely plain abstract text. Parsing the structure of an article increases difficulty in GN task. Therefore, BioCreate-III is not considered in our work.

There are many possibilities to improve the overall performance of our framework. For the gene name recognition task, the experiments show that exploring more biological domain features, using specified domain knowledge can achieve a satisfactory performance. Parsing direction during training CRF model might also affect the recognition capacity for complicate gene name. The parsing direction during training CRF model may have an influence on the recognition capacity for complicate gene name. For example, given the text 'the receptor ADP and ...', the gene name entity should be 'receptor ADP'. However, the NER component only extracts 'ADP' as a gene mention, which consequently leads to a false mapping because of the lack of modifier token 'receptor'. Checking the training corpus, it is found that in most cases the 'receptor' appears in the last position of a gene name. Therefore, bi-directional parsing on the training corpus i.e., forward and backward parsing might correct this issue. For some special cases, only linear-chain CRF can not correctly identify the gene names in a complete form. For example, 'DEFCAP-L and -S', only 'DEFCAP-L' is extracted. 'DEFCAP-S' is missed.

For gene name normalization task, a wide range of rules and methodologies are applied. The Token Metric similarity we proposed indeed increases recall. Currently, our Token Metric only captures the substitution probability between tokens by using the official training corpus provided by BioCreAtIvEII. If more training pairs are available, our Token Metric could be more precise. A study of writing style in human writers should be researched further, e.g., summarizing and incorporating more accurate rules to tell whether two words is exchangeable in two gene names. As pointed out, In [38], to achieve a high recall and precision, there are more than 8 sources to be compiled in order to achieve a comprehensive blacklist. Compiling and curation of such a large volume data requires a lot of time, usually human effort is involved. More high quality and professional training corpus could increase performance. Gene name normalization is a domain-centric task in natural language processing, background knowledge in biology should be incorporated into it. Furthermore, a more specified language model should be proposed. This specified language model should consider many potential factors of context, not only just one previous or next neighbor words, but also the structure of one particular biological phrase. Moreover, exploration of more machine learning methods could give another insight into this task.

8. Future Work

There are many aspects in our framework that should be considered in order to improve the performance. Here we list some inherent directions and changes:

1. Incorporating more training corpora into CRF model. Currently, only 2 kinds of corpora are tested on Gene Name Recognition. One interesting observation is that when merging GeneTag and BioCreAtIvE II as a large corpus to train CRF model, the F-measure score decreased by 2% unexpectedly. Such a merge interferes with dominating feature weights, in return, the feature score in CRF can not 'confidently' identify a gene. More detailed research is needed.

2. Gene name recognition is similar to speech recognition in in many aspects, testing other CRF models such as high-order or non-linear chain CRF [10]could improve overall performance. Additionally, modifying more combination on low-level features in feature template file when using CRF++ might also help extract full complex gene name.

3. Using new toolkit. Existing toolkits and libraries embedded into our framework are only responsible for specific tasks. GNAT[12] now provides a complete functionalities ranging from gene name identification to gene normalization released in Java library. This normalization library could provide more convenience when developing gene normalization system.

9. Acknowledgement

I would like to acknowledge and express my heartfelt gratitude to the following persons who have made the completion of this article. I would like to thank my supervisor, Dr. Erwin M. Bakker, who guided me a with clear research direction through weekly discussion and feedback. I have learned a lot from this project. I would like to thank my friends: Lingpeng Kong, Hao Wang who give me inspiration and suggestion in this thesis. Finally, I want to thank my parents who always supported me sincerely.

References

[1] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[2] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, et al. Gene ontology: tool for the unification of biology. *Nature genetics*, 25(1):25–29, 2000.

[3] A. Bairoch, R. Apweiler, C. H. Wu, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, et al. The universal protein resource (uniprot). *Nucleic acids research*, 33(suppl 1):D154–D159, 2005.

[4] B. Baldwin and B. Carpenter. Lingpipe. *Available from World Wide Web: http://alias-i. com/lingpipe*, 2003.

[5] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171, 1970.

[6] A. Borthwick. *A maximum entropy approach to named entity recognition*. PhD thesis, New York University, 1999.

[7] D. Campos, S. Matos, and J. L. Oliveira. Gimli: open source and high-performance biomedical name recognition. *BMC bioinformatics*, 14(1):54, 2013.

[8] W. W. Cohen, P. Ravikumar, S. E. Fienberg, et al. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03)*, volume 47, 2003.

[9] G. D. Forney Jr. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[10] E. Fosler-Lussier, Y. He, P. Jyothi, and R. Prabhavalkar. Conditional random fields in speech, audio, and language processing. *Proceedings of the IEEE*, 101(5):1054–1075, 2013.

[11] M. Gilleland. Levenshtein distance. *Three Flavors. http://www. merriampark. com/ld. htm*, 2005.

[12] J. Hakenberg, M. Gerner, M. Haeussler, I. Solt, C. Plake, M. Schroeder, G. Gonzalez, G. Nenadic, and C. M. Bergman. The gnat library for local and remote gene mention normalization. *Bioinformatics*, 27(19):2769–2771, 2011.

[13] J. Hakenberg, L. Royer, C. Plake, H. Strobelt, and M. Schroeder. Me and my friends: gene mention normalization with background knowledge. In *Proc 2nd BioCreative Challenge Evaluation Workshop*, pages 1–4, 2007.

[14] D. Hanisch, K. Fundel, H.-T. Mevissen, R. Zimmer, and J. Fluck. Prominer: rule-based protein and gene entity recognition. *BMC bioinformatics*, 6(Suppl 1):S14, 2005.

[15] E. Hatcher, O. Gospodnetic, and M. McCandless. Lucene in action, 2004.

[16] C.-N. Hsu, Y.-M. Chang, C.-J. Kuo, Y.-S. Lin, H.-S. Huang, and I.-F. Chung. Integrating high dimensional bi-directional parsing models for gene mention tagging. *Bioinformatics*, 24(13):i286–i294, 2008.

[17] Y. Hu, Y. Li, H. Lin, Z. Yang, and L. Cheng. Integrating various resources for gene name normalization. *PloS one*, 7(9):e43558, 2012.

[18] A. Islam and D. Inkpen. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2(2):10, 2008.

[19] M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989.

[20] J.-D. Kim, T. Ohta, Y. Tateisi, and J. Tsujii. Genia corpusa semantically annotated corpus for bio-textmining. *Bioinformatics*, 19(suppl 1):i180–i182, 2003.

[21] T. Kudo. Crf++: Yet another crf toolkit. *Software available at http://crfpp. sourceforge. net*, 2005.

[22] C.-J. Kuo, Y.-M. Chang, H.-S. Huang, K.-T. Lin, B.-H. Yang, Y.-S. Lin, C.-N. Hsu, and I.-F. Chung. Rich feature set, unification of bidirectional parsing and dictionary filtering for high f-score gene mention tagging. In *Proceedings of the second BioCreative challenge evaluation workshop*, volume 23, pages 105–107, 2007.

[23] J. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[24] H. Liu, Z.-Z. Hu, J. Zhang, and C. Wu. Biothesaurus: a web-based thesaurus of protein and gene names. *Bioinformatics*, 22(1):103–105, 2006.

[25] D. Maglott, J. Ostell, K. D. Pruitt, and T. Tatusova. Entrez gene: gene-centered information at ncbi. *Nucleic acids research*, 35(suppl 1):D26–D31, 2007.

[26] A. McCallum and W. Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 188–191. Association for Computational Linguistics, 2003.

[27] A. K. McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[28] A. Morgan, Z. Lu, X. Wang, A. Cohen, J. Fluck, P. Ruch, A. Divoli, K. Fundel, R. Leaman, J. Hakenberg, et al. Overview of biocreative ii gene normalization. *Genome biology*, 9(Suppl 2):S3, 2008.

[29] N. Ponomareva, P. Rosso, F. Pla, and A. Molina. Conditional random fields vs. hidden markov models in a biomedical named entity recognition task. In *Proc. of Int. Conf. Recent Advances in Natural Language Processing, RANLP*, pages 479–483, 2007.

[30] L. Rabiner and B. Juang. An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3(1):4–16, 1986.

[31] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[32] M. J. Schuemie, R. Jelier, and J. A. Kors. Peregrine: Lightweight gene name normalization by dictionary lookup. In *Proc of the Second BioCreative Challenge Evaluation Workshop*, pages 131–133, 2007.

[33] B. Settles. Abner: an open source tool for automatically tagging genes, proteins and other entity names in text. *Bioinformatics*, 21(14):3191–3192, 2005.

[34] L. Smith, L. Tanabe, R. Ando, C.-J. Kuo, I.-F. Chung, C.-N. Hsu, Y.-S. Lin, R. Klinger, C. Friedrich, K. Ganchev, et al. Overview of biocreative ii gene mention recognition. *Genome Biology*, 9(Suppl 2):S2, 2008.

[35] C. Sutton. An introduction to conditional random fields. *Foundations and Trends® in Machine Learning*, 4(4):267–373, 2012.

[36] L. Tanabe, N. Xie, L. H. Thom, W. Matten, and W. J. Wilbur. Genetag: a tagged corpus for gene/protein named entity recognition. *BMC bioinformatics*, 6(Suppl 1):S3, 2005.

[37] Y. Tsuruoka. Genia tagger: Part-of-speech tagging, shallow parsing, and named entity recognition for biomedical text. *Available at: www-tsujii. is. su-tokyo. ac. jp/GENIA/tagger*, 2006.

[38] J. Wermter, K. Tomanek, and U. Hahn. High-performance gene name normalization with geno. *Bioinformatics*, 25(6):815–821, 2009.

A. Appendix

Feature	explanation	example
feature 1	word itself	'alkaline' : 'alkaline'
feature 2	stemmed word	'phosphatases' : 'phosphatas'
feature 3	part-of-speech	'alkaline' : NN
feature 4	capital information	'Comparison' : InitialCaps
feature 5	case of letter	'alkaline' : lowercase
feature 6	digit count	'5-nucleotidas' : SingleDigit
feature 7	word length	'alkaline' : len6+(the word length is longer than 6)
feature 8	vowel characters	'alkaline' : 'a–a—e'
feature 9	is amino acid in short form?	'Ser'
feature10	is amino acid in long form?	'tyrosine'
feature11	is ATCG sequence?	ATCGU
feature12	is Greek letter?	alpha
feature13	is Roman Numeral?	I,II, VII
feature14	is nucleoside?	Thymine
feature15	is nucleotide?	ATP
feature16	is nucleic acid?	cDNA
feature17	digit sequence	CO2 : CO*
feature18	upper case and lower case sequence	CO2: AA1
feature19	compaction of feature18	CO2: A1
feature20	the first 1-7 letters of a token	alkaline: a, al, alk, alka, alkal, alkali, alkalin
feature21	reverse order of feature20	alkaline: alkalin, alkali, alkal, alka, alk, al, a

Table 8. Feature set defined in Crf++: the second column is the explanation of feature, the third column gives a concrete example.