# TREC 2004 Genomics Track Experiments at IUB

Kazuhiro Seki, James C. Costello, Vasanth R. Singan, and Javed Mostafa

Laboratory for Applied Informatics Research, Indiana University Bloomington
1320 East Tenth Street, LI 011, Bloomington, Indiana 47405, USA

{kseki,jccostel,vsingan,jm}@indiana.edu

## Abstract

This paper describes the methods we developed for the three tasks of the TREC Genomics Track, i.e., ad hoc retrieval, triage, and annotation tasks. For the ad hoc retrieval task, we used the classic vector space model and studied the use of query expansion and pseudo-relevance feedback. Our submitted runs obtained a MAP of 0.183. For the triage task, we adopted a naïve Bayes classifier trained on MeSH terms and used gene names as filters to rule out false positives. The obtained normalized utility score was 0.435. For the annotation task, we focused on document representation and applied a variant of the *k*NN classifiers. One of our submitted runs produced an $F_1$ score of 0.561, ranking first out of 36 runs submitted for the annotation task.

## 1 Introduction

The volume of the biomedical literature is enormous and has been rapidly growing. For example, MEDLINE, a bibliographic database in the biomedical domain, currently contains over 13 million records and 2000 records are added daily. This rich resource of knowledge has motivated various research themes in the area of information retrieval (IR) for realizing effective information access.

To foster the IR and related research, specifically targeting biomedical text, the Text Retrieval Conference (TREC) launched the genomics track in 2003 [8], which attracted the second largest group of participants among all the tracks. This year, the genomics track had two primary tasks: *ad hoc retrieval* and *categorization* tasks. The latter mimics some parts of Mouse Genome Informatics (MGI) database curation processes that are currently carried out by human experts, and consists of two subtasks, i.e., *triage* and *annotation*, as described shortly.

The following sections report our proposed methods and the results for the ad hoc retrieval, triage, and annotation tasks in turn.

## 2 Ad Hoc Retrieval Task

### 2.1 Overview

This is a conventional ad hoc retrieval task targeting the biomedical literature. Participants were provided with 50 topics, and for each topic, they were required to retrieve a set of relevant documents sorted according to the estimated relevance. The topics were collected through interviews with biologists in order to capture the real-world information need. Each topic consisted of three fields, i.e., title, information need, and context along with topic ID. An example topic is shown in Figure 1.

```
ID:      51
TITLE:   pBR322 used as a gene vector
NEED:    Find information about base
         sequences and restriction maps
         in plasmids that are used as gene
         vectors.
CONTEXT: The researcher would like to
         manipulate the plasmid by
         removing a particular gene and
         needs the original base sequence
         or restriction map information of
         the plasmid.
```

Figure 1: An example topic for the ad hoc retrieval task.

The data set used for this task is 10 years' worth of MEDLINE data (1994–2003), containing 4,591,008 bibliographic records. Incidentally, five sample topics and the corresponding list of relevant documents (not complete) were provided for the purpose of system design and tuning.

We submitted two runs (`lga1` and `lga2`) for this task; only the difference between them is that `lga1` used pseudo-relevance feedback, while `lga2` did not.

## 2.2 Methods

**Framework** The vector space model and the TFIDF term weighting scheme [13] were adopted as the basic framework. In addition, the use of query expansion and pseudo-relevance feedback was explored. Figure 2 depicts the overview of our retrieval system.

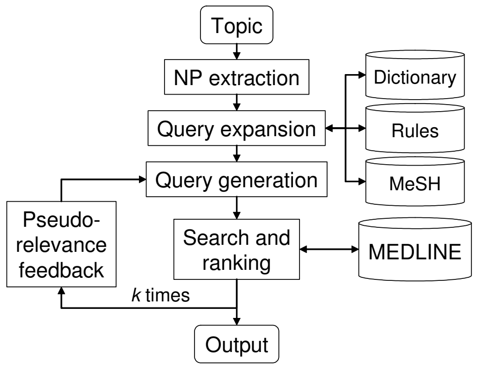

Figure 2: Framework of our IR system.

The following paragraphs describe the components of our IR system in more detail.

**NP extraction** Given a topic, the system first extracted noun phrases (NPs) from the `TITLE` and `NEED` fields as a potential query terms using a general-purpose parser [7]. The `CONTEXT` field and other categories of words (e.g., verbs) were not used since they tended to degrade the system performance on the tuning data.

**Query expansion** The extracted NPs were then expanded using three sources of information to form the final query. Firstly, a gene name dictionary was consulted to find synonyms. The dictionary was compiled from the SWISS-PROT [10] and LocusLink [12] databases and contained 493,473 records. Each record has a gene/protein name as the keyword and lists its synonyms such as aliases and symbols where no disambiguation was done for multi-sense names. Secondly, a set of hand-crafted rules was applied to generate name variants. These rules were designed to tolerate minor difference (e.g., existence/absence of symbols and spaces) between strings. Lastly, for each of the potential query terms, the Medical Subject Heading (MeSH) vocabulary was searched for the synonyms.

**Query generation** Since the number of the potential query terms (phrases) collected through the preceding steps can be quite large, they were filtered based on document frequency (DF) and relative frequency ratio (RFR) [3] for efficiency. Terms with high DF and/or low RFR are thought to be too general and discarded. Document frequency of term $t$, denoted as $DF(t)$, is the number of documents containing $t$ within the test collection. $RFR(t)$ is a ratio of relative frequency of $t$ in the test collection to that of $t$ in the other collection from a different domain. RFR can be used to discover terms characteristic of a corpus when compared to another corpus [3]. For this experiment, we used 5,986 abstracts from Inspec,[1] a bibliographic database for physics, engineering, computing, etc. The thresholds for DF and RFR were empirically set to 30,000 and 5, respectively, based on a preliminary look at the tuning data. Finally, terms which went through the filters were concatenated by Boolean OR operators, forming the final query.

**Search and ranking** Five textual fields (`ArticleTitle`, `AbstractText`, `DescriptorName`, `QualifierName`, and `NameOfSubstance`) in the MEDLINE data were indexed in advance. The search module exhaustively retrieved the documents which contained any terms/phrases composing the query. As for ranking the retrieved documents, TFIDF and cosine similarity were used. The four fields of the topics were not distinguished in searching and scoring in the experiments.

**Pseudo-relevance feedback** The pseudo-relevance feedback (PRF) module assumed the top $n$ ranked documents to be relevant and used significant terms in the documents to refine the query. As a measure of significance, we used TFIDF values. For each document, $m$

[1]`http://www.iee.org/Publish/INSPEC/`

terms with highest TFIDF were sent back to the query generation module to be considered for inclusion in the refined query. This process can be repeated arbitrary $k$ times. For the submitted run (`lga1`), we experimentally set $n = m = 10$ and $k = 1$.

### 2.3 Results

We submitted two sets of results, labeled as `lga1` and `lga2`; the former used pseudo-relevance feedback (PRF), while the latter did not. The mean average precision (MAP) were 0.183 for `lga1` and 0.175 for `lga2`; both were found lower than the mean 0.207 of all the 37 submitted runs.

We conducted additional experiments to examine the usefulness of each system component and the effect of some parameters. Through the experiments, we observed that:

- Using the `CONTEXT` field, PRF ($k$=3), and query expansion improved the performance in MAP by 24%, 19%, and 8%, respectively, as compared to the case where they were not used.

- As expected, terms/phrases with high DF values (more than 1000) did not contribute to the performance (at least in our framework). Thus, they may be safely removed in forming queries.

- Contrary to our expectation, thresholding on RFR degraded MAP by 9%.

## 3 Triage Task

The triage task for the TREC 2004 Genomics track asks a simple, but very important question. We are presented with a large set of articles and asked to determine which articles should be considered for further curation of genes or gene products in relation to mouse genomics research. The human eye and brain have been the data processing tools for curation in the past, but this problem can be abstracted to the computational issue of text categorization, or more specifically, binary text classification.

There are many different approaches that can be taken for this task. There has been a great deal published on techniques that implement machine learning, clustering, and filtering techniques [15], which all address different levels of the text categorization problem. Each of these techniques has been shown to have strengths and weaknesses, with some outperforming others in various tasks [1]. It has become our job as researchers and TREC participants to identify the best technique for this particular task.

### 3.1 Overview

There was a great deal of time dedicated to exploring different categorization techniques for the triage task. After experimenting with support vector machines (SVM), decision trees, various term filters, and naïve Bayesian classifiers, it was determined that the best approach for our group was to implement a naïve Bayesian classifier, in addition to a filter aimed at the genes in an article.

Our group submitted two runs to the 2004 triage task for the TREC conference and results will be discussed in section 3.3.

### 3.2 Methods

#### 3.2.1 Feature Selection

Several different techniques were used to select and test feature sets. For the naïve Bayesian classifier, there were three different sets of features tested. We looked at a document frequency (DF) set of stemmed terms, a chi-squared set of stemmed terms [18], and the set of Medical Subject Headings (MeSH) terms for each document. We received the best results, in terms of recall and precision, from the collection of all the MeSH terms within the training set, so this was the set of features used for the classifier.

The MeSH terms were collected via a web crawler, implemented in Perl, which takes a list of PubMed IDs (PMIDs) and would search the NCBI website with the assistance of NCBI's Entrez Programming Utilities (eutils). [2]

The Gene filter required a different set of features, which for each article, is defined as the set of genes contained in each article.

The first step in identifying the genes was to clean the data. A file was created for each article from the train-

[2] `http://eutils.ncbi.nlm.nih.gov/entrez/query/static/eutils_help.html`

ing and test sets, which contained only the text from the abstract and body of the article by parsing out any unnecessary data, where all of the XML tags were removed.

Next, YAGI (Yet Another Gene Identifier) [16] was applied to each article to identify all the genes within an article.

After the features for each article were properly chosen, the classifier could be trained and tested, and the filter applied.

### 3.2.2 Naïve Bayesian Classifier

The naïve Bayesian classifier is a simple classification model that takes into account conditional probabilities based on Bayes' Theorem. This model uses a supervised training data set to calculate the necessary probabilities and can then be applied to a test data set to classify documents.

Conceptually, we can think of the classification task as the following conditional probability:

$$p(C|F_1...F_n) \tag{1}$$

where, $C$ is a class or category and $(F_1...F_n)$ is a set of features.

The classes for the triage task are simply the set $\{pass, fail\}$, where the document passes to be further curated, or fails and is not considered for curation. The features represent the MeSH terms.

We can now apply Bayes' Theorem to the conditional probability (Equation (1)) and derive the following:

$$p(C|F_1...F_n) = \frac{p(C)p(F_1...F_n|C)}{p(F_1...F_n)} \tag{2}$$

Ultimately, we are only interested in the numerator of equation 2, which is the joint probability. We can ignore the denominator of the equation, as $C$ is not a parameter, and over the entire set of documents, the $p(F_1...F_n)$ can be thought of as a constant. We can then rewrite the joint probability, as:

$$p(C, F_1...F_n) \tag{3}$$

With the joint probability, we are still carrying along the conditional probability for set $\{F_1...F_n\}$, so by repeatedly applying the definition of the conditional probability, we derive a series of equations as follows:

$$\begin{aligned} &p(C, F_1...F_n) \\ &\quad = p(C)p(F_1...F_n|C) \\ &\quad = p(C)p(F_1|C)p(F_2...F_n|C, F_1) \\ &\quad = p(C)p(F_1|C)p(F_2|C, F_1)p(F_3...F_n|C, F_1, F_2) \\ &\quad = ... \end{aligned} \tag{4}$$

So far the idea of "naïve" from the naïve Bayesian classifier has not come into play, however, mathematically this is an important aspect of the classifier and will now be applied. "naïve" refers to the assumption that all features within the set $\{F_1...F_n\}$ are independent, where $F_i \neq F_j$. When this is true for each feature, the probability of each feature does not depend on the presence or absence of any other feature.

$$p(F_i|C, F_j) = p(F_i|C) \tag{5}$$

After applying this concept to Equation (4), we can rewrite the joint probability as [5]:

$$\begin{aligned} p(C, F_1...F_n) &= p(C)p(F_1|C)p(F_2|C)...p(F_n|C) \\ &\quad OR \\ p(C, F_1...F_n) &= p(C)\prod_{i=1}^{n} p(F_i|C) \end{aligned} \tag{6}$$

The above explanation of a naïve Bayesian classifier has been implemented in the Perl module *Algorithm:NaiveBayes*.[3] This module was used in conjunction with Perl code to train the classifier using the articles in the training set and their associated MeSH terms. The trained classifier was then applied to the test data set, which returned a set of data that will be passed on to the Gene Filter.

### 3.2.3 Gene Filter

The gene filter was used solely to eliminate false positives. The naïve Bayesian classifier returned a large number of documents with high recall, but with low precision.

Often times, there are a large number of genes, and/or gene products, that are mentioned within an article related to molecular biology. Also, research articles tend to describe a very specific area of research, where a specific group of genes, proteins, DNA, or RNA are

[3]`http://search.cpan.org/~kwilliams/Algorithm-NaiveBayes-0.03/lib/Algorithm/NaiveBayes.pm`

mentioned. This phenomena can be exploited, by looking for correlations between the genes that are mentioned in the articles with curation information, as compared to the genes that are mentioned in the articles that are not passed on for curation. Trends in genes and gene products that are present in articles passed on for curation gives us another measure for the triage task.

The gene filter counts the gene products and genes, which will be referred to simply as genes, and are identified by the inserted YAGI tags, `<GENE></GENE>`. This is done for all articles within the training set. The set of genes collected from the entire set of articles are then separated based on their triage classification. The set of genes present in the articles deemed worthy of further curation are noted $G_p$, while the set of genes present in the articles which are not taken for further curation are noted $G_n$. The genes that are in union with set $G_p$ and $G_n$ are eliminated from each set. This gives us a unique set of genes that only appear in the set of articles that will be used for further curation, noted $G_{pu}$, and a set of unique genes that only appear in the set of articles that are not used for further curation, noted $G_{nu}$.

Set $G_{nu}$ was then ranked, and if a gene occurred in more than a certain percentage of the negatively curatable articles, it was added to the final gene set, noted $G_f$. The percentage used in determining if a gene should be included in set $G_f$ was adjusted for our two submitted runs. Set $G_f$ was used to eliminate false positive articles from the MeSH filtered set of articles. An article was eliminated from the final set of articles if a gene from set $G_f$ was present in the article.

Overall, after applying the naïve Bayesian classifier and the gene filter, we derive a set of articles that were classified as being of interest to further curate by MeSH terms and genes.

## 3.3 Results and Discussion

Overall, the use of a naïve Bayesian classifier and gene filter performed relatively well, as compared to the other 57 triage task submissions by TREC participants. Table 1 shows the best, median, and worst performances of the entire group of 59 submissions, as well as our submissions. The main measure for this task was the normalized utility score, which takes into account the true and false positive articles in a submitted run. Our first run performed well, and our normalized utility was 0.434, where the best performance was a score of 0.651 and the median performance was 0.342. Our second run did not perform as well as the first, but still managed to perform at the median with a score of 0.342.

Table 1: The top 3 rows represent the Best, Median, and Worst results from the 59 triage runs that were submitted for 2004. The bottom two rows represent our two submitted results for the triage task.

| Run | Precision | Recall | $F$-score | Norm. Utility |
|---|---|---|---|---|
| Best | 0.230 | 0.988 | 0.284 | 0.651 |
| Median | 0.136 | 0.557 | 0.183 | 0.342 |
| Worst | 0.071 | 0.014 | 0.026 | 0.011 |
| Run 1 | 0.111 | 0.721 | 0.193 | 0.434 |
| Run 2 | 0.108 | 0.581 | 0.183 | 0.342 |

There was only one difference between the first and the second run, which deals with the gene filter. Both runs were classified through the naïve Bayesian classifier using the same set of MeSH terms and set of training articles. However, the gene filter for each run was adjusted, in regards to the percentage of genes that were included in set $G_f$ (Section 3.2.3). Set $G_f$ was used to eliminate false positive articles from the filtered set of articles from the naïve Bayesian classifier and the percentage of genes used in the first run was much stricter than the percentage of genes used in the second run. For the first run, a gene needed to occur in at least 10% of the negatively curatable articles to be included in set $G_f$, while a gene needed to be present in only 5% of the negatively curatable articles to be included in set $G_f$ for run 2.

By increasing the amount of genes that are used to eliminate an article from the final run, we are increasing the amount of true positive articles that are eliminated, which is what happened in our second run. The first run had an appropriate amount of genes to eliminate false positives, while not eliminating true positives.

Overall, we feel that our solution of using a naïve Bayesian classifier and gene filter to the 2004 Genomics Triage Task was a success. Both of our submitted runs normalized utility scores were above the median of the 57 TREC run submissions, higlighted by our first run score, which was well above the median score.

## 4 Annotation Task

### 4.1 Overview

The goal of the annotation task is to assign GO hierarchy and evidence codes to a given ⟨article, gene⟩ pair. There are three hierarchy codes and 12 evidence codes possibly assigned.[4] For this task, we were allowed to submit results only for hierarchy code assignment and for hierarchy and evidence code assignment. Hereafter, we call the former *Task1* and the latter *Task2*.

For the data sets, 504 and 378 full-text articles in SGML format were provided as the training and test data, respectively. Before use, they were automatically converted to XML format by a UNIX command `SGML2XML`. In addition, the character entities representing Greek alphabets (e.g., `&agr;`) were converted to the corresponding English spellings (e.g., `alpha`).

### 4.2 Methods

**Framework** As in the triage task, the annotation task is a classification problem, where GO hierarchy codes or combinations of hierarchy and evidence codes are regarded as classes. For this task, we adopted *k*NN classifiers which have been reported as one of the best classifiers for text categorization [17]. To apply the classifiers, each ⟨article, gene⟩ pair was first represented by a term weight vector. Since an entire article is not necessarily relevant to the particular gene, we used only the paragraphs mentioning the gene name. Figure 3 depicts the flow of the processes.

**Finding synonyms** First, aliases and abbreviations of the given gene name were searched for in the article itself. Specifically, we looked at `<KEYWORD>` and `<GLOSSARY>` fields that may explicitly define a pair of original name and corresponding abbreviation.[5] We also examined the use of body text because gene name abbreviations often appear immediately following the original names [14]. However, it slightly degraded the classification performance and thus was disabled for our submitted runs. Additionally, the gene name dictionary used in the ad hoc retrieval task was consulted for finding synonyms. Hereafter, "gene names" refers to the original names, aliases, and abbreviations for short.

[4]`http://www.geneontology.org`
[5]`http://highwire.stanford.edu/about/dtd/`

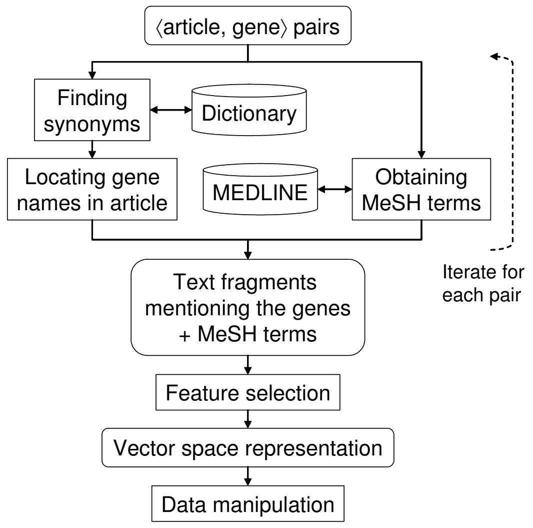

Figure 3: A data processing flow to represent input by term weight vectors.

**Locating gene names in the given article** Next step is to find text fragments mentioning the gene in question within the paired article, since the entire article may not be necessarily relevant to the particular gene. As gene names generally have many variants due to the inconsistent use of symbols, spaces, etc. [6], both gene names and text were preprocessed as follows, so as to tolerate such minor difference:

- Replace symbols with spaces
- Insert space between different character types, such as alphabets and numerals (e.g., `Diet1` → `Diet 1`)
- Insert space between Greek alphabets and other words (e.g., `NF-kappaB` → `NF kappa B`)
- Lowercase all characters

Then, each paragraph in the article was checked if it was likely to contain any of the gene names associated with the gene. Note that section titles were appended to each paragraph within the sections since they were often found to be descriptive. In addition, if the paragraph referred to figures and/or tables for the first time, the captions were also appended to it. For each gene name and each context mentioning any word of the gene name, the word-overlap score defined below was computed.

$$Overlap(gene, context) = \frac{M - \alpha \cdot U}{N + \beta} \tag{7}$$

where $M$ and $U$ denote the numbers of matching and unmatching words, respectively; $\alpha$ is a penalty for unmatching words (set to 0.3); $N$ is the number of words composing the gene name; and $\beta$ penalizes shorter gene names (set to 2). If any of the scores associated with a paragraph exceeded a predefined threshold (set to 0.3), all terms in the paragraph were added to the feature set representing the ⟨article, gene⟩ pair. The values of the parameters were determined based on our preliminary experiments on the training data.

**Obtaining MeSH terms** For each input article, all the associated MeSH terms were obtained from MEDLINE using Entrez Utilities,[6] irrespective of the genes that were paired with the article. These MeSH terms were also added to the feature set representing the ⟨article, gene⟩ pairs. However, a special symbol `MESH+` was concatenated to each MeSH term in order to distinguish MeSH from other terms.

**Feature selection** The features (terms) extracted in the previous steps were preprocessed by stopword removal based on the PubMed stopword list[7] and stemming. We used Lovins stemmer [9] for our official runs but also examined Porter stemmer [11] and non-stemming. For feature selection, we used chi-square statistic [18] defined as:

$$\chi^2(t, c) = \frac{N(AD - CB)^2}{(A + C)(B + D)(A + B)(C + D)} \tag{8}$$

where $A$ is the number of documents containing term $t$ in class $c$, $B$ is the number of documents containing $t$ in classes other than $c$, $C$ is the number of documents not containing $t$ in $c$, $D$ is the number of documents not containing $t$ in classes other than $c$, and $N$ is the total number of documents. For a given term $t$, the chi-square statistic was computed for each class (either a hierarchy code or a combination of hierarchy and evidence codes depending on the task). The maximum score was taken as the chi-square statistic for term $t$, that is, $\chi^2(t) = \max_i \chi^2(t, c_i)$. Only the top $n$ terms with higher chi-square statistics were used for the following processes. For the official runs, $n$ was set to 2000 and 3000 for *Task1* and *Task2*, respectively, based on the preliminary experiments.

[6]`http://www.ncbi.nlm.nih.gov/entrez/query/static/eutils_help.html`

[7]`http://www.ncbi.nlm.nih.gov/entrez/query/static/help/pmhelp.html`

**Data manipulation** Each ⟨article, gene⟩ pair was represented by a term vector using the term set obtained through the preceding steps. As term weighting schemes, we compared TFIDF and TFCHI as well as raw term frequency (TF). The following shows the definitions of TFIDF and TFCHI of term $t$ in document $d$.

$$\begin{aligned} \mathrm{TFIDF}(t, d) &= (1 + \log(\mathrm{TF}(t, d)) \cdot \frac{N}{\mathrm{DF}(t)} \\ \mathrm{TFCHI}(t, d) &= (1 + \log(\mathrm{TF}(t, d)) \cdot \log(\chi^2(t)) \end{aligned} \tag{9}$$

The latter scheme, TFCHI, is an instance of *supervised term weighting* schemes proposed by Debole and Sebastiani [4]. It takes into account pre-labeled class information in the training data and re-uses statistics computed in the feature selection step (chi-square statistics in this case) in place of IDF. Since TFCHI produced slightly better $F$-scores than the others in our preliminary experiments, it was used for our official runs. In addition, singular value decomposition (SVD) [2] was applied to compensate for the small size of the training data. The number of singular values used for reconstructing the matrix (i.e., the number of dimensions after reduction) was empirically set to 50 and 70 for *Task1* and *Task2*, respectively.

**Classification** Using the matrix created by the procedure above, each input (a pair of article and gene) was classified into one or more predefined classes by a variant of $k$NN classifiers. First, the input ⟨article, gene⟩ was represented by a term weight vector following the same procedure illustrated in Figure 3 except for feature selection and data manipulation. The features previously identified on the training data were used as filters and the other terms not found in the feature set were discarded. Then, the remaining terms were weighted by the same scheme as applied to the training data, but global term weights (either IDF or CHI) were obtained from the training data as we were not allowed to use those statistics from the test data. After that, the input term vector was transformed into lower dimensions (50 for *Task1* or 70 for *Task2*) using the matrix of singular vectors computed earlier on the training data. Given an input vector, the (original) *k*NN algorithm finds *k* neighbors most similar to the input among the training data, and aggregates similarity scores associated with the *k* neighbors for each class. If these scores are greater than per-class thresholds, the corresponding classes are assigned to the input. We further weighted the aggregated scores by the number of the *k* neighbors having class *c*. Equation (10) shows the scoring scheme given class *c* and input vector *v*.

$$Score(c, v) = \sum_i sim(v, n_{c,i}) \times |n_c| - t_c \qquad (10)$$

where $n_c$ is the *k* nearest neighbors having class *c*, $|n_c|$ is the number of $n_c$, $t_c$ is a per-class threshold, and $sim(\cdot)$ returns the cosine similarity between the arguments. Threshold $t_c$ was optimized so as to maximize the micro-averaged *F*-score over all the training data. The use of $|n_c|$ intends to give higher weights to classes more frequent in the *k* neighbors.

## 4.3 Results and discussions

We submitted two sets of results for each of the subtasks. They were denoted as `lgcad` for *Task1* (hierarchy code assignment) and `lgcab` for *Task2* (hierarchy and evidence code assignment). The only difference is that `lgcad1` and `lgcab1` used SVD for feature space dimensionality reduction, while `lgcad2` and `lgcab2` did not. Table 2 summarizes the results for the submitted runs. After submission, however, we found a few bugs related to approximate word matching and term weighting in our codes. Table 3 presents the results produced by the corrected codes.

Table 2: The official results for the annotation task. Mean is the mean $F_1$ score over all the submissions.

| | Runtag | Prec | Recall | $F_1$ score | Mean |
|---|---|---|---|---|---|
| *Task1* | `lgcad1` | 0.441 | 0.769 | 0.561 | 0.382 |
| | `lgcad2` | 0.427 | 0.785 | 0.553 | |
| *Task2* | `lgcab1` | 0.341 | 0.492 | 0.403 | – |
| | `lgcab2` | 0.323 | 0.607 | 0.422 | |

Additional experiments were conducted to investigate what factors contributed to the performance.

Table 3: The corrected results (not official) for the annotation task. Mean is the mean $F_1$ score over all the submissions.

| | Runtag | Prec | Recall | $F_1$ score | Mean |
|---|---|---|---|---|---|
| *Task1* | `lgcad1` | 0.514 | 0.729 | 0.603 | 0.382 |
| | `lgcad2` | 0.509 | 0.719 | 0.596 | |
| *Task2* | `lgcab1` | 0.403 | 0.519 | 0.454 | – |
| | `lgcab2` | 0.382 | 0.498 | 0.433 | |

Specifically, we examined the effects of the following factors:

- Threshold for approximate word matching in identifying relevant paragraphs (see Equation (10)): We tested 0 (meaning to use all the paragraphs irrespective of gene name occurrences), 0.3 (our default), 0.5 (more restrictive). In addition, exact word matching (denoted as "Exact") was tested instead of approximate word matching.
- Use of the gene name dictionary: Our system was tested with/without the gene name dictionary.
- Use of section titles: Same as above.
- Use of MeSH terms: Same as above.
- Stemming algorithms: Porter's and Lovins' algorithms were compared as well as non-stemming.
- Term weighting schemes: Raw term frequencies (TF), TFIDF, and TFCHI were compared.
- *k*NN scoring: The original and our proposed variant were compared.
- Use of SVD for dimensionality reduction: Our system was tested with/without SVD.

In the experiments, only the factor under consideration was changed with other features remaining the same as `lgcad1` or `lgcab1`. Notice that, however, the per-class threshold $t_c$ for *k*NN was optimized for each setting in order to compare the best possible $F_1$ scores under different settings. Tables 4 and 5 shows the results, where asterisks (*) indicate the default settings used to produce `lgcad1` or `lgcab1`.

The following summarizes the empirical observations from Tables 4 and 5.

Table 4: Results for GO hierarchy code assignment (*Task1*). Numbers in parentheses are percent increase/decrease relative to the default setting indicated by an asterisk.

| Feature | | Training data | | | Test data | | |
|---|---|---|---|---|---|---|---|
| | | Precision | Recall | $F_1$ score | Precision | Recall | $F_1$ score |
| Threshold for gene name match | 0 (output all) | 0.367 | 0.713 | 0.484 (−11.5%) | 0.479 | 0.802 | 0.600 (−4.9%) |
| | 0.3* | 0.477 | 0.642 | 0.547 | 0.548 | 0.746 | 0.631 |
| | 0.5 | 0.385 | 0.346 | 0.365 (−33.3%) | 0.413 | 0.434 | 0.423 (−33.0%) |
| | Exact | 0.434 | 0.297 | 0.353 (−35.5%) | 0.543 | 0.331 | 0.412 (−34.7%) |
| Dictionary | Not used | 0.361 | 0.593 | 0.449 (−17.9%) | 0.420 | 0.543 | 0.474 (−24.9%) |
| | Used* | 0.477 | 0.642 | 0.547 | 0.548 | 0.746 | 0.631 |
| Section titles | Not used | 0.479 | 0.628 | 0.543 (−0.7%) | 0.545 | 0.764 | 0.636 (+0.8%) |
| | Used | 0.477 | 0.642 | 0.547 | 0.548 | 0.746 | 0.631 |
| MeSH terms | Not used | 0.479 | 0.632 | 0.545 (−0.3%) | 0.567 | 0.731 | 0.638 (+1.1%) |
| | Used* | 0.477 | 0.642 | 0.547 | 0.548 | 0.746 | 0.631 |
| Stemmer | None | 0.479 | 0.633 | 0.546 (−0.2%) | 0.554 | 0.739 | 0.633 (−0.3%) |
| | Porter | 0.472 | 0.632 | 0.540 (−1.3%) | 0.529 | 0.772 | 0.628 (−0.5%) |
| | Lovins* | 0.477 | 0.642 | 0.547 | 0.548 | 0.746 | 0.631 |
| Term weighting | TF | 0.379 | 0.706 | 0.493 (−9.9%) | 0.430 | 0.721 | 0.539 (−14.6%) |
| | TFIDF | 0.435 | 0.756 | 0.552 (+0.9%) | 0.531 | 0.731 | 0.615 (−2.5%) |
| | TFCHI* | 0.477 | 0.642 | 0.547 | 0.548 | 0.746 | 0.631 |
| *k*NN scoring | Original | 0.465 | 0.652 | 0.543 (−0.7%) | 0.556 | 0.750 | 0.639 (+1.3%) |
| | Variant* | 0.477 | 0.642 | 0.547 | 0.548 | 0.746 | 0.631 |
| SVD | Not used | 0.447 | 0.664 | 0.535 (−2.2%) | 0.506 | 0.812 | 0.623 (−1.3%) |
| | Used* | 0.477 | 0.642 | 0.547 | 0.548 | 0.746 | 0.631 |

Table 5: Results for GO hierarchy and evidence code assignment (*Task2*). Numbers in parentheses are percent increase/decrease relative to the default setting indicated by an asterisk.

| Feature | | Training data | | | Test data | | |
|---|---|---|---|---|---|---|---|
| | | Precision | Recall | $F_1$ score | Precision | Recall | $F_1$ score |
| Threshold for gene name match | 0 (output all) | 0.249 | 0.152 | 0.188 (−53.3%) | 0.356 | 0.607 | 0.449 (−3.2%) |
| | 0.3* | 0.369 | 0.444 | 0.403 | 0.370 | 0.623 | 0.464 |
| | 0.5 | 0.263 | 0.213 | 0.235 (−41.7%) | 0.228 | 0.399 | 0.290 (−37.5%) |
| | Exact | 0.278 | 0.266 | 0.272 (−32.5%) | 0.321 | 0.259 | 0.287 (−38.1%) |
| Dictionary | Not used | 0.288 | 0.358 | 0.319 (−20.8%) | 0.280 | 0.510 | 0.361 (−22.2%) |
| | Used* | 0.369 | 0.444 | 0.403 | 0.370 | 0.623 | 0.464 |
| Section titles | Not used | 0.371 | 0.458 | 0.410 (+1.7%) | 0.383 | 0.590 | 0.464 (0%) |
| | Used | 0.369 | 0.444 | 0.403 | 0.370 | 0.623 | 0.464 |
| MeSH terms | Not used | 0.368 | 0.463 | 0.410 (+1.7%) | 0.395 | 0.561 | 0.464 (0%) |
| | Used* | 0.369 | 0.444 | 0.403 | 0.370 | 0.623 | 0.464 |
| Stemmer | Not used | 0.371 | 0.483 | 0.420 (+4.2%) | 0.397 | 0.544 | 0.459 (−1.1%) |
| | Porter | 0.381 | 0.413 | 0.396 (−1.7%) | 0.413 | 0.529 | 0.463 (−0.2%) |
| | Lovins* | 0.369 | 0.444 | 0.403 | 0.370 | 0.623 | 0.464 |
| Term weighting | TF | 0.353 | 0.363 | 0.357 (−11.4%) | 0.213 | 0.736 | 0.330 (−28.9%) |
| | TFIDF | 0.313 | 0.506 | 0.387 (−4.0%) | 0.295 | 0.667 | 0.409 (−11.9%) |
| | TFCHI* | 0.369 | 0.444 | 0.403 | 0.370 | 0.623 | 0.464 |
| *k*NN scoring | Original | 0.336 | 0.494 | 0.400 (−0.7%) | 0.364 | 0.621 | 0.459 (−1.1%) |
| | Variant* | 0.369 | 0.444 | 0.403 | 0.370 | 0.623 | 0.464 |
| SVD | Not used | 0.338 | 0.463 | 0.391 (−3.0%) | 0.374 | 0.552 | 0.446 (−3.9%) |
| | Used* | 0.369 | 0.444 | 0.403 | 0.370 | 0.623 | 0.464 |

- For the threshold for approximate word matching, our default (threshold=0.3) outperformed others in most cases, indicating the effectiveness of our framework to use only paragraphs containing the target gene names and the importance of approximate term matching as well as the choice of the threshold.

- The use of gene name dictionary drastically improved the performance (around 20% in $F_1$ score). It proves the widespread use of gene synonyms and their importance in this domain.

- The use of section titles and MeSH terms had little effect for both tasks.

- The different stemming algorithms made little difference except for the case where stemmer was not used on the training data for *Task2*, which showed 4.3% increase relative to the case where Lovins stemmer was used (our default).

- For term weighting schemes, TFCHI and TFIDF consistently worked better than TF. In addition, TFCHI worked notably better than TFIDF for *Task2*, partly supporting the idea of the supervised term weighting schemes [4] that class-based term weights (e.g., chi-square statistics as used for this work) is more appropriate for classification.

- The different *k*NN scoring schemes had little effect for both tasks.

- The use of SVD marginally improved classification in $F_1$, especially for *Task2*.

## 5 Summary

We took part in the three tasks of the Genomics Track. For the ad hoc retrieval task, we adopted the vector space model and applied several existing IR techniques including noun phrase extraction, query expansion, and pseudo-relevance feedback. Our submitted run using all the features obtained 0.183 MAP. For the triage task, we applied the naïve Bayes classifier trained solely on MeSH terms and obtained a normalized utility score of 0.434. Additionally, we examined the use of gene names as a filter to rule out false positives, which, however, deteriorated the performance to 0.342 contrary to our expectation. Lastly, for the annotation task, we represented each input (i.e., a pair of article and gene) by a set of terms extracted from text fragments mentioning the gene in question, where we paid special attention to dealing with various forms of gene synonyms and variants. Then, we applied a *k*NN classifier to assign hierarchy codes and optionally evidence codes. Our submitted runs achieved an $F_1$ of 0.561 for hierarchy code annotation and 0.422 for hierarchy plus evidence code annotation, which were found to be the best scores among all the participants. Further experiments showed that approximate gene name matching and the gene name dictionary contributed the most to the performance, followed by the use of SVD and the term weighting schemes.

## Acknowledgment

This project is partially supported by the NSF grant ENABLE #0333623.

## References

[1] Kjersti Aas and Line Eikvil. Text categorization: A survey. Technical report, Norwegian Computing Center, 1999.

[2] Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37(4):573–595, 1995.

[3] Fred J. Damerau. Generating and evaluating domain-oriented multi-word terms from texts. *Information Processing & Management*, 29(4):433–447, 1993.

[4] Franca Debole and Fabrizio Sebastiani. Supervised term weighting for automated text categorization. In *Proceedings of SAC-03, 18th ACM Symposium on Applied Computing*, pages 784–788, 2003.

[5] P. Domingos and M. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2–3):103–130, 1997.

[6] Sergei Egorov, Anton Yuryev, and Nikolai Daraselia. A simple and practical dictionary-based approach for identification of proteins in MEDLINE abstracts. *Journal of the American Medical Informatics Association*, 11(3):174–178, 2004.

[7] Dennis Grinberg, John Lafferty, and Daniel Sleator. A robust parsing algorithm for link grammars. Technical Report CMU-CS-95-125, Carnegie Mellon University, 1995.

[8] William Hersh. Report on TREC 2003 genomics track first-year results and future plans. *SIGIR Forum*, 38(1):69–72, 2004.

[9] Julie Beth Lovins. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11:22–31, 1968.

[10] Claire O'Donovan, Maria Jesus Martin, Alexandre Gattiker, Elisabeth Gasteiger, Amos Bairoch, and Rolf Apweiler. High-quality protein knowledge resource: SWISS-PROT and TrEMBL. *Brief Bioinform*, 3(3):275–284, 2002.

[11] Martin Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[12] Kim D. Pruitt and Donna R. Maglott. RefSeq and LocusLink: NCBI gene-centered resources. *Nucleic Acids Research*, 29(1):137–140, 2001.

[13] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., 1986.

[14] Ariel S. Schwartz and Marti A. Hearst. A simple algorithm for identifying abbreviation definitions in biomedical text. In *Proceedings of the Pacific Symposium on Biocomputing (PSB)*, volume 8, pages 451–462, 2003.

[15] Fabrizio Sebastiani. Machine learning in automated text retrieval. *ACM Computing Surveys*, 34(1):1–47, March 2002.

[16] Burr Settles. Biomedical named entity recognition using conditional random fields and rich feature sets. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA)*, 2004.

[17] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 42–49, 1999.

[18] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the 14th International Conference on Machine Learning*, pages 412–420, 1997.