BioText Team Experiments for the TREC 2004 Genomics Track

PI Nakov[†], AS Schwartz[†], E Stoica[§], MA Hearst[§]
[†]Computer Science Division
[§]School of Information Management and Systems
University of California Berkeley
Berkeley, CA 94720
{nakov, sariel}@cs.berkeley.edu, {estoica,hearst}@sims.berkeley.edu

Abstract

The BioText group participated in the two main tasks of the TREC 2004 Genomics track. Our approach to the ad hoc task was similar to the one used in the 2003 Genomics track, but due to the lack of training data, we did not achieve the high scores of the previous year. The most novel aspect of our submission for the categorization task centers around our method for assigning Gene Ontology (GO) codes to articles marked for curation. This approach compares the text surrounding a target gene to text that has been found to be associated with GO codes assigned to homologous genes for organisms with genomes similar to mice (namely, humans and rats). We applied the same method to GO codes that have been assigned to MGI entries in years prior to the test set. In addition, we filtered out proposed GO codes based on their previously observed likelihood to co-occur with one another.

1 Introduction

The TREC 2004 Genomics Track consisted of an ad hoc retrieval task and a categorization task, which in turn had two sub-tasks, triage and annotation. This year, the Berkeley BioText group participated in all three tasks.

The ad hoc task was a traditional search task. The collection consisted of a ten year subset of MEDLINE (about 4.5 million documents) and 50 topics derived from information needs obtained via interviewing biomedical researchers. There was no training data, other than five sample topics and relevance judgments.

The goal of the categorization task was to mimic activities performed by curators in the Mouse Genome Informatics (MGI) project [8]. Human curators at MGI annotate genes and proteins with Gene Ontology (GO) codes based on evidence found in documents. The Gene Ontology [11] is a controlled vocabulary of terms (GO codes) describing gene product attributes. It is organized into three disjoint hierarchies: molecular functions (*MF*), biological processes (*BP*) and cellular components (*CC*). Examples of GO codes are: GO:0030246 (*carbohydrate binding*, a *MF*), GO:0007067 (*mitosis*, a *BP*) and GO:0005634 (*nucleus*, a *CC*).

In MGI, a gene is annotated with a GO code only if there is a document that contains evidence to support the annotation. The Gene Ontology defines nine evidence codes. Examples of evidence codes include: *inferred from mutant phenotype* (IMP), *inferred from direct assay* (IDA) and *inferred by curator* (IC).

Manual GO code assignment is a three step process. In the first step, all documents that talk about the mouse and its biology are identified. In the second step, a decision is made about which of the documents identified in step one have experimental evidence warranting GO code annotation. Articles that pass the second step are then manually entered into the MGI system with a tag for GO code, mapping, expression, etc. and with a target gene(s) assigned. In the third step, curators assign GO and evidence codes to the target gene(s) in each document.

The triage task corresponds to step two above, identifying which papers contain experimental evidence, while the annotation task corresponds to part of step three, annotating document-gene pairs with GO *hierarchy* codes (i.e. *MF*, *BP* and *CC*, not the codes themselves) and optionally with evidence codes.

The data for the categorization task consisted of documents from three journals: the *Journal of Biological Chemistry (JBC)*, the *Journal of Cell Biology (JCB)*, and the *Proceedings of the National Academy of Science (PNAS)*, for a total of 11,880 documents. The articles from 2002 were used for training and those from 2003 for testing.

Below we describe our approach for each task, beginning with the annotation task, which was the most novel of our approaches.

2 The Annotation Task

The requirement for the annotation task was to assign GO hierarchy codes (*MF*, *BP* and *CC*) to document-gene pairs. Optionally, we could assign the evidence codes for the annotation. There were 1,418 document-gene pairs in the training set and 877 pairs in the testing set. The evaluation used the F-measure defined as

$$F = \frac{2 \times P \times R}{P + R}, \tag{1}$$

where P is the precision and R is the recall.

The novelty of our approach centers around our methods for restricting and filtering the candidate GO codes. We use information about species whose genomes are similar to the mouse genome, and about which kinds of GO codes can logically co-occur, using a measure based on the κ coefficient [10]. This produces high recall but low precision. To improve the latter, we use an SVM classifier that was applied to three of the five submitted runs.

2.1 General Procedure

In this section we present our general procedure for extracting GO codes. In a nutshell, we pre-processed the documents and the GO codes, identified the target genes in text and extracted the GO codes within a fixed size window around the target genes.

2.1.1 Gene Identification

Several gene tagging tools were made available to the participants, including YAGI [6], LingPipe [3] and AbGene [1]. While they identify most of the genes mentioned in the text, they do not tell us which of these refer to the *target gene*. For this reason we used our own gene identification tool, developed for last year's Genomics Track [7]. Our module assigns LocusLink ID(s), thus making straightforward the identification of the target gene.

However, the LocusLink gene names and synonyms had insufficient coverage, so we augmented them with gene names extracted from the MGI database, linking them to the corresponding LocusLink IDs.

2.1.2 GO Codes Identification

We built a table consisting of the text corresponding to the definition of each GO code. For each GO code's text, we eliminated stop words and punctuation characters and divided the text into tokens using space as a delimiter.

When processing the full text of a journal article, our system first identified locations of target genes, and then searched for GO codes within a window centered around the target gene. In each window, we assigned a GO code to a gene if we found a proportion of GO term tokens larger than a threshold (0.8 in our experiments).

Having found an occurrence of a GO code, we assigned to the document-gene pair the GO hierarchy code corresponding to that GO code. For example, if we found in the text the GO code *protein binding*, we assigned *MF* to the document-gene pair since *protein binding* is a molecular function.

2.2 Searching for Only a Subset of the GO codes

The GO ontology currently contains about 16,700 GO codes. Presumably, searching in text for all possible GO codes will result in a large number of false positives. Thus we decided to limit the set of GO codes that would be considered possible candidates.

For each gene we aimed to search for *only the GO codes we expect to find in text* for that gene. Given that mice, humans and rats are species with similar genomes, our assumption was that there should be an overlap between the GO codes a mouse gene is annotated with and the GO codes its homologous genes (human and rat genes with the same name as the mouse gene) annotations.

To find the expected GO codes for a gene, we used the GO annotation database (GOA [9]) from the Eu-

ropean Bioinformatics Institute (EBI) [2]. GOA contains annotations of human and rat gene products with GO codes. Thus, for a mouse gene we search in text *only for the GO codes its homologous genes are annotated with in GOA*. For example, for mouse gene *desmin*, we search for only the GO codes that human and rat genes with name *desmin* are annotated with in GOA.

For the training set we found annotations for 60% of the genes, for a total of 1,553 GO codes. Since we were missing annotations for about half of the genes, we decided to use this set of 1,553 GO codes as a pool from which to draw labels for the entire collection. We refer to this as the GOA subset.

In addition to the subset of GO codes annotated to homologous genes, we experimented with a subset of MGI codes. More precisely, since the training collection consisted of articles from 2002, our subset contained the GO codes annotated to genes in MGI before 2002. This subset had 1,791 GO codes.

2.3 Filtering GO Codes using κ Coefficient

Due to the nature of biomedical text, it is possible to find GO codes in text that are illogical. For example, if a gene is involved in *RNA transcription*, this cannot happen outside the cell, so if we find in the text both strings *RNA transcription* and *extracellular*, one has to be eliminated. To eliminate such illogical groupings, we used again the GOA human database.

For each pair of GO codes annotated to a gene (GO_1 and GO_2) we compute the κ coefficient [10, 13] as follows.

Consider the following variables

- a: the number of times the gene is annotated with both codes;
- b: the number of times the gene is annotated with GO_1 but not with GO_2;
- c: the number of times the gene is annotated with GO_2 but not with GO_1; and
- d: the number of times the gene is not annotated with GO_1 or GO_2.

The actual and the expected probabilities that GO_1 and GO_2 are annotated to the same gene are:

$$A_f = \frac{a+d}{a+b+c+d} \quad (2)$$

$$E_f = P_1 * P_2 + (1-P_1) * (1-P2) \quad (3)$$

where $P_1 = (a+b)/(a+b+c+d)$, $P_2 = (a+c)/(a+b+c+d)$.

Then

$$\kappa(GO_1, GO_2) = \frac{A_f - E_f}{1-E_f} \quad (4)$$

We compute the κ coefficient between every pair of GO codes annotated to a gene in the GOA human database. The κ coefficient takes values between -1 and 1. GO codes that are both frequently annotated to a gene have positive κ coefficients. GO codes that are not annotated together to genes have 0 or negative κ coefficients.

We use the κ coefficients to eliminate illogical GO codes as follows. In particular, we eliminate a GO code that does not have positive κ coefficients with more than 75% of the rest of GO codes found in the text for that gene.

2.4 Filtering GO Codes with an SVM Classifier

Experimental results showed that our technique for identifying GO hierarchy codes achieves high recall, but low precision. For example, on training data using the MGI subset (which in this case contains GO codes annotated before 2002) and a window of 10, we obtained a recall of 0.75, but a precision of only 0.24.

To improve precision, we filter the GO codes using an SVM classifier provided as part of the open source machine learning package WEKA (Waikato Environment for Knowledge Analysis [5, 14]) from the University of Waikato, New Zealand.

There are three GO hierarchy codes *MF*, *BP* and *CC*, so we can train three separate binary classifiers, one for each hierarchy code. Since we obtain a sufficiently high recall by just extracting GO codes from text, we only resort to the classifier when the GO hierarchy code the classifier is responsible for has been assigned in the previous step. Otherwise, the hierarchy code is not assigned.

We experimented with the following features.

- GO codes extracted from text: (e.g. *mitosis*);

Run	Classifier	F	Precision	Recall
biotext21	N	0.4041	0.2708	0.7960
biotext22	N	0.4008	0.2658	0.8141
biotext23	Y	0.3299	0.4437	0.2626
biotext24	Y	0.3367	0.4452	0.2707
biotext25	Y	0.3149	0.4181	0.2525

Table 1: Annotation Task Submission Results.

- GO hierarchy codes corresponding to the GO codes extracted from text: (e.g. *BP*);
- document section: e.g. title, introduction, results, figure etc.;
- number of previous assignments in MGI (before 2003) of a particular GO code to the target gene;
- number of previous assignments in MGI (before 2003) of a particular GO code to a gene from the same family as the target gene;
- number of previous assignments in LocusLink (before 2003) of a particular GO code to a human homolog of the target gene;
- number of previous assignments in LocusLink (before 2003) of a particular GO code to a human homolog of a gene from the family of the target gene;
- MeSH tree codes, cut at level 3 (e.g. F02.463.425);
- evidence codes keywords from a manually constructed list.

Most of the features were extracted from a window around the target gene (feature-dependent size).

We extracted the document sections from the SGML files (we do not allow our windows to cross the section boundaries). The reason to include the document sections as features is based on the observation that GO codes annotated to genes seem to be found primarily in titles, figure captions, and some particular sections (e.g. results and discussions).

The idea to look at the previous assignments to gene/family comes from the observation that the more often a particular GO code has been assigned in the past, the higher the likelihood that it will be assigned again. Similarly, homologous genes in mouse and human are likely to share the same functions, processes and components, i.e. to be assigned the same GO hierarchy codes. MeSH terms are generally among the best features for text classification tasks in the biomedical domain and proved to be useful in this case too. A description of our MeSH term identification module can be found in [7].

Finally, for each evidence code, we manually constructed a list of relevant keywords. These were derived from the evidence codes definition found at the GO Web site[1] and have been further augmented with related terms by a person with biomedical background. We also tried words/stems but these did not help and were not used in the final submission.

While gene families are already available online from MGI[2], they had limited coverage and were further restricted to mouse only. Instead, we extracted family information from LocusLink by searching for gene names and synonyms containing one of the strings: *family*, *superfamily/super-family* and *subfamily/sub-family*. For example, the gene *olfactory receptor, family 5, subfamily V, member 1* is a member of *subfamily V* of the *olfactory receptor family*. The naming regularities in LocusLink allowed us to design a simple set of rules and to extract 13,456 different genes grouped into 3,575 families/subfamilies/superfamilies.

2.5 Results

We submitted five runs for the annotation task (see Table 1). To each document-gene pair we assigned GO hierarchy codes only. Three runs used an SVM classifier. The two runs without a classifier, biotext21 and biotext22, obtained F = 0.4041 (P = 0.2708, R = 0.7960) and respectively F = 0.4008 (P = 0.2658, R = 0.8141). The first run was obtained with the MGI subset and a window of 10 (2,700 GO codes). The second run was obtained with the GOA subset (1,004 GO codes) and a window of 20.

Tables 2 and 3 show the F-measure obtained without a classifier, on training and respectively testing data for window sizes of 10 and 20. MGI and GOA denote the two subsets we experimented with (the MGI

[1] http://www.geneontology.org/GO.evidence.html
[2] http://www.informatics.jax.org/mgihome/nomen/genefamilies/index.shtml

Subset of GO codes	**Win = 10**	**Win =20**
MGI	0.3725 (0.06)	0.3704 (0.05)
MGI + κ	0.3507 (0.06)	0.3726 (0.06)
MGI + MeSH	0.3863 (0.07)	0.3981 (0.07)
MGI + κ + MeSH	0.3851 (0.08)	0.4008 (0.07)
GOA	0.3967 (0.06)	0.3903 (0.05)
GOA + κ	0.3988 (0.06)	0.3904 (0.06)
GOA + MeSH	0.3791 (0.07)	0.4006 (0.07)
GOA + κ + MeSH	0.3769 (0.08)	0.4006 (0.08)
All GO	0.3803 (0.04)	0.3733 (0.03)
All GO + κ	0.3810 (0.05)	0.3730 (0.04)
All GO + MeSH	0.3990 (0.05)	0.3883 (0.06)
All GO + κ + MeSH	0.3970 (0.05)	0.3883 (0.06)

Table 2: F-measure on *training* data for assigning GO hierarchies. Numbers in parentheses show F-measures for computing exact GO codes (not part of the TREC task).

Subset of GO codes	**Win = 10**	**Win =20**
MGI	0.4041 (0.06)	0.4019 (0.04)
MGI + κ	0.4067 (0.06)	0.4065 (0.05)
MGI + MeSH	0.4626 (0.08)	0.4674 (0.07)
MGI + κ + MeSH	0.4666 (0.08)	0.4694 (0.07)
GOA	0.3989 (0.06)	0.4021 (0.04)
GOA + κ	0.3991 (0.06)	0.4008 (0.05)
GOA + MeSH	0.4450 (0.08)	0.4644 (0.07)
GOA + κ + MeSH	0.4431 (0.08)	0.4608 (0.07)
All GO	0.3002 (0.02)	0.3923 (0.03)
All GO + κ	0.2996 (0.02)	0.3946 (0.04)
All GO + MeSH	0.1793 (0.02)	0.4628 (0.05)
All GO + κ + MeSH	0.1719 (0.02)	0.4634 (0.06)

Table 3: F-measure on *testing* data for assigning GO hierarchies. Numbers in parentheses show F-measures for computing exact GO codes (not part of the TREC task).

GO codes annotated before 2003 and respectively the GO codes annotated to homologous genes in GOA database), while All GO denotes the set of all GO codes. In parenthesis, we show the F-measure computed with the exact GO codes, assuming the target goal was to assign the correct GO code for a gene, not just the *MF*, *CC* or *BP* code.

Several points are worth noting. First, we see a dramatic difference between the F-measure computed in terms of *MF*, *CC*, *BP* and the F-measure computed in terms of exact GO codes. This shows that while our system can assign a correct hierarchy code, it is generally not able to assign the correct GO code. Second, using a reduced set of codes yields a marginally better performance. Third, from the point of view of running time, it is also about twenty times faster to search for a reduced set of codes than to search for all the codes.

The performance on testing data was better than the performance on training data. A possible explanation for this could be the fact that the testing set was smaller (877 document-gene pairs compared to 1423 document-gene pairs in the training set). Filtering with κ generally helps though not significantly. Also, the results obtained with the MGI subset and with the GOA subset are comparable.

A data analysis performed after the TREC submission showed that we can improve the F-measure using the MeSH terms. Particularly, after finding a GO code in text, we eliminate it if we cannot find 50% of its tokens in the MeSH terms annotated to the document. Our assumption here is that the gene function has to be a major topic in the document. MeSH terms represent the major topics in the document, hence we eliminate the GO codes not found in the MeSH terms. Using the MeSH terms we improve our performance on the testing collection from 0.4041 to 0.4694 (obtained with the MGI subset and a window of 20).

We submitted three runs with the SVM classifier (see Table 1). Biotext23 and biotext25 used the GOA subset, while biotext24 used all.

For biotext23 and biotext24 we searched for features in a window of 200 words centered around the gene, while for biotext25 we searched in a window of 100 words. While we gain in precision (for example biotext24 achieves a precision of 0.4452 compared to biotext21 which obtains a precision of 0.2708), we lose a lot on recall and these runs are worse compared to the ones without the classifier. The best F obtained with a classifier is only 0.3367, whereas the best F obtained without the classifier is 0.4041.

3 The Ad Hoc Retrieval Task

For the ad hoc retrieval task we used a database-centered approach similar to what we submitted last year for TREC [7]. We experimented with several techniques for gene recognition and different MeSH [4] sub-hierarchies.

In MeSH, each concept is assigned a unique identifier and one or more alphanumeric tree numbers (corresponding to particular positions in the hierarchy). For example, A (*Anatomy*), A01 *Body Regions*, A01.456 *Head*. In addition, each MeSH concept is assigned a semantic type, e.g., *Enzyme*, *Mammal*, *Chemicals*. For the ad hoc task we used the MeSH terms annotated to documents and the MeSH chemicals sematic type.

We processed the queries and extracted a variety of features including:

- `GENE_LOCUS`: genes found by approximate matching to the LocusLink genes and synonyms;
- `GENE_YAGI`: genes found using Yagi [6];
- `GENE`: words looking like a gene/protein/chemical name found in the text (e.g. SLC40A1);
- `FAMILY`: family name inferred from the query text (using a pattern matching);
- `FAMILY_LOCUS`: family names for the genes (automatically generated by processing the synonym names in LocusLink, as described above);
- `MeSH`: MeSH terms;
- `ORGANISM_MeSH`: the organism name (inferred from MeSH).

For every type of feature we generated a complex sub-query, as follows[3]. Let G be the various forms of the feature and let LG be other lower-confidence rules for normalizing the feature names that have a higher rate of false positives (implemented only for LocusLink genes). The score of a document R given feature type f is computed as follows:

[3]See [7] for a more detailed description of our database system.

Score(R, f) = the aggregated SUM over the result of the UNION operator GROUP BY document identifier of:

(a) J * (G compared to terms in titles)
(b) J * (LG compared to terms in abstracts)
(c) K * (LG compared to terms in titles)
(d) K * (LG compared to terms in abstracts)
(e) L * (MeSH concepts compared to MeSH terms assigned to documents)

where $J = 1, K = 0.015$, and $L = 1.4$ (determined experimentally). The final score of a document, is calculated as $\sum_f Score(R, f) * tf(f) * weight(f)$, where $tf(f)$ is the term frequency of the feature in the topic, and $weight(f)$ is the weight assigned to feature type f.

Because of the lack of training topics with reliable reference judgement, we could not fine tune the feature type weights. The following weights were used in our official run:

GENE_LOCUS	=	1
GENE	=	0.6
GENE_YAGI	=	0.8
MeSH	=	2
ORGANISM_MeSH	=	4
FAMILY	=	3
FAMILY_LOCUS	=	3

3.1 Results

We submitted one official run for the ad-hoc task. The mean average precision (MAP) over the 50 topics was 0.1384, which is quite low. We did not produce additional unofficial runs, but we believe that the system suffered from the lack of proper training data for tuning the different weights. Fine-tuning the weights on the training data was one of the main reasons our system achieved high performance last year. Another limitation of the system was that we did not have a sub-query for terms that were not mapped to gene-names or MeSH terms.

4 The Triage Task

For the triage task we analyzed the PubMed citations only; we did not use the full text, although we think doing so could have improved the results. We trained an SVM classifier using SVM-light [12] with the following features:

- the words from the title, weighted by tf.idf for the entire collection, and normalized so that the weight vector sums to one (for use by the SVM classifier);
- the MeSH labels assigned to the citation, cut off at tree level 3;
- a nominal feature indicating if the article had been assigned a genre other than "Journal Article" (such as "Survey"), which can sometimes indicate negative evidence for curation;
- the DataBankName feature assigned to some PubMed entries, since those articles that have been assigned this feature often discuss a topic relevant to genomics and often indicate a positive curation decision.

In order to optimize the Normalized Utility (NU) measure used to assess the triage task, and because there were many more negative instances than positive, it was necessary to include documents with negative SVM scores in the result set. Experimentation on the training set indicated that this cutoff should be set to -0.9. However, when applied to the test set, the number of documents retrieved at this threshold seemed too large, so we decided to submit at larger cutoff scores of -0.7, -0.6, -0.5, -0.4, and -0.3. After receiving the test labels it was evident that we should have used the -0.9 cutoff determined experimentally, as that would have brought our NU score to 0.344, which is close to the track average. Our best official NU score was 0.32 using a cutoff of -0.7.

5 Conclusions

For the annotation task we experimented with reduced sets of GO codes and filtered the GO codes using the κ coefficient. To improve precision we further filtered the GO codes using an SVM classifier. We obtained a marginally better performance with the reduced sets of GO codes versus the performance obtained with all the GO codes. Also, searching in text for only a subset of GO codes is about 20 times faster. Using the κ coefficient to eliminate codes that should not co-occur seems to improve performance slightly.

Unfortunately, while clearly helpful on the training data, the SVM classifier achieved a lower performance on the testing data. A possible explanation for this performance could be that the testing documents contained mostly new genes for which little is

known. For a lot of them we did not have any synonyms available, i.e. we found far less instances of the target gene in text, which means far less features available to the classifier. In addition, some of our most important features were eliminated altogether: if the gene is new then we do not have previous assignments to it, there are no known homologous genes and we do not know its family. All these make the testing data look different from the training, which can be disastrous for a maximum margin classifier like SVM. Moreover, fewer windows means fewer possible GO categories proposed and thus less utility from filtering them.

6 Acknowledgements

We would like to thank Janice Hamer for her help in generating keywords corresponding to evidence codes. This research was supported by NSF grant DBI-0317510 and a gift from Genentech.

References

[1] Abgene. `ftp.ncbi.nlm.nih.gov/pub/tanabe/AbGene/`.

[2] European bioinformatics institute (ebi). http://www.ebi.ac.uk/.

[3] Lingpipe. http://alias-i.com/lingpipe/.

[4] Medical subject heading (mesh). http://www.nlm.nih.gov/mesh/.

[5] Waikato environment for knowledge engineering. http://www.cs.waikato.ac.nz/ml/weka/.

[6] (yagi) yet another gene identifier. http://www.cs.wisc.edu/ bsettles/yagi/.

[7] Bhalotia, G., Nakov, P., Schwartz, A., and Hearst, M. Biotext team report for the trec 2003 genomic track. In *Proceedings of the 12th Text REtrieval Conference* (2003), pp. 612–621.

[8] Blake, J., Richardson, J., Bult, C., Kadin, J., Eppig, J., and the members of the Mouse Genome Database Group. Mgd: The mouse genome database. In *Nucleic Acids Res* (2003), vol. 31, pp. 193–195.

[9] Camron, E., Barrell, D., Lee, V., Dimmer, E., and Apweiler, R. The gene ontology annotation (goa) database - an integrated resource of go annotations to the uniprot knowledgebase. In *Silico Biology* (2004), vol. 4(1), pp. 5–6.

[10] Cohen, J. A coefficient for agreement for nominal scales, 1960.

[11] Consortium, T. G. O. Gene ontology: tool for the unification of biology. In *Nature Genet.* (2000), vol. 25, pp. 25–29.

[12] Joachims, T. *Learning to Classify Text Using Support Vector Machines.* Kluwer, 2002.

[13] Kraemer, H. Kappa coefficient. In *S. Kotz and N. L. Johnson (Eds.), Encyclopedia of Statistical Sciences.* (1982), John Wiley & Sons.

[14] Witten, I., and Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, 1999.