Fast and Furious Text Mining

Joel D. Martin
National Research Council, Canada
joel.martin@nrc.gc.ca

Abstract

Text mining studies in biology are often limited to thousands instead of millions of Medline records or are very slow. However, with a modified search engine, many common text mining tasks can be done rapidly. In fact, some information extraction and text categorization tasks can be achieved in seconds or minutes even across tens of gigabytes of (previously indexed) text. In this paper, we present TLM, an efficient implementation of a text analysis engine that uses a highly expressive query language. With this language, users can create queries that quickly accomplish what previously required several different custom-built systems to achieve.

1 Introduction

Text mining is our only hope to find all the literature references to specific facts, such as gene or protein interactions. At present, it is still a hope and not fully a reality. Most text mining tools work for a small number of abstracts, or more rarely full-text articles (e.g., [1]). Some do work for millions of articles but are relatively slow (e.g., hours to days, [6]). Still other approaches have been designed to process millions of articles quickly, but they can apparently lose considerable accuracy compared to slower methods (e.g., [11]).

The challenge then is to build tools that permit a wide variety of very rapid text mining across millions of documents. This challenge is even more relevant when we consider that the next generation of text mining tools will be expected to handle terabytes of full-text articles, not just gigabytes of abstracts. If we cannot rapidly mine the text of Medline, how can we hope to handle the full articles?

Below, we describe a text analysis engine called TLM (Text and Language Mining) with a highly expressive query language. TLM is a principle component of our integrated suite of tools called LitMiner ([9]). TLM permits queries that can quickly accomplish what previously required several different custom-built systems to achieve.

2 TLM: A Text Analysis Engine

How is text analysis different from search? On a search engine, users compose words into queries and expect lists of documents in return. That is an important capability and many other tasks are made possible by search engines. However, our text mining tasks often require a little more and would be easier with a slightly different engine.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

A text analysis engine would let us compose queries out of words and other entities such as punctuation, tags, part of speech, phrases, sentences, etc. For example, we might want all the occurrences of **`THA`** in parentheses: **`'X(' THA ')X'`** (the *X* is a wildcard character for punctuation). Instead, we might want to refer to any occurrence of any verb: **`is <Verb> by`**, or might want to find sentences that contain two or more genes, **`<Sentence> > (<Gene> .. <Gene>)`**.

Second, a text analysis engine would return results of different types, including documents, but also sections, sentences, phrases, terms, and words. For example, in asking for sentences that contain two genes, we may not care which documents they come from. We just want the statements themselves so we can see which genes are said to inhibit each other. In asking for three words before THA, **`* * * 'X(' THA ')X'`**, we do not want to see the matching documents. We want to see the possible expansions for the acronym THA, such as Total Hip Arthroplasty(374), tetrahydroacridine(25), or Tokai High Avoider(4).[1]

Third, a text analysis engine should permit rapid statistical analysis of the text pieces that are returned. There is a wide range of possible analyses, including simple frequencies in documents or sentences and ranging to more complex distributions.

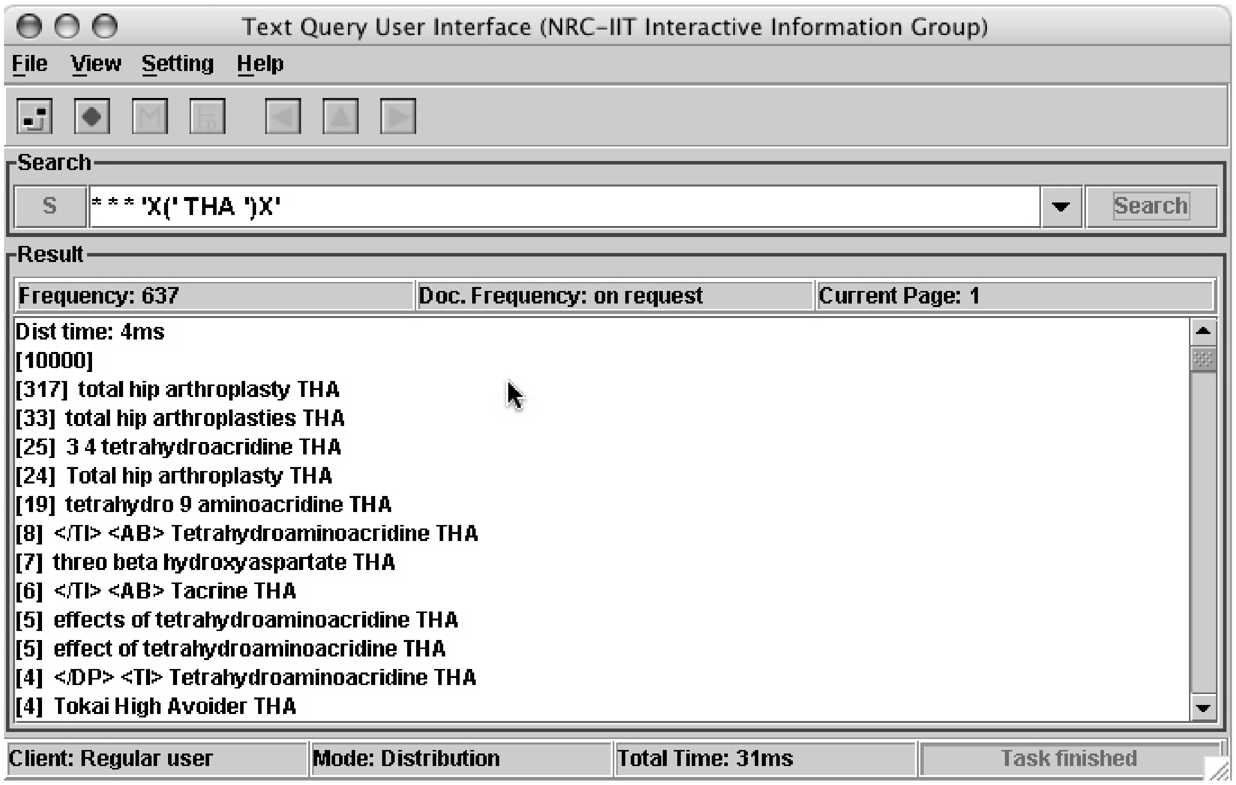
2.1 Engine Design

TLM is a relatively mature implementation of a text search and analysis engine. Figure 1 shows one client graphical user interface (GUI) that is connected to a remote installation of TLM. The figure illustrates the query **`* * * 'X(' THA ')X'`** and shows its output. TLM has many added conveniences for users and has been optimized for many types of search, but its operation can be summarized by five basic ideas that are outlined below. Although none of these ideas is completely new, some aspects are unusual or unique when compared to search engines. Furthermore, the combination of the five ideas is new. That combination is essential for supporting the above definition of text analysis.

The first and most fundamental idea behind TLM's functionality is borrowed directly from search engines. It is an inverted index of the positions of words (e.g., [2]). Uniquely in TLM, this idea is extended to include strings of spaces and punctuation as well as words. Any document or collection of documents can be described as a list of words (or punctuation) and their position of occurrence. For example, if we were indexing the current paragraph, we would assign the position 1 to the word *The*, 2 to *first*, etc. All of these words and their positions can now be organized as in a back-of-the-book index. Each word can be connected to a list of the positions in which it appears. In TLM specifically, each word or collection of adjacent punctuation (called a *separator*) is connected to a list of the positions of that term in documents. In a collection of multiple documents, the position could include the document number or could ignore it (e.g., [5]; [3]). In our collection TLMtest, the word **`tumourogenic`** appears 10 times in 9 Medline abstracts. The index stores the word position of each of those occurrences. Similarly, the separator **`',,,, '`** (four commas and a space) appears once, and that position is stored.

Once we know the positions of each word and separator, we can ask how often two particular words occur near each other. We do this simply by comparing the lists of positions and checking certain conditions. For example, we might want to find all the (up to) four word phrases that contain *cancers* and *tumours*. Our search engine can retrieve the lists of word positions for *cancers* and for *tumours* and can iterate through those lists looking for two, three, or four word phrases that contain both. For example, suppose *cancers* appears as the 10th and 45th word of a document and *tumours* appears as the 20th and 43rd word of the same document. Scanning

[1]All example queries described in this paper were run against the TLMTest collection. For these examples, TLM was running on a 2.4 GHz AMD Opteron. The TLMTest collection is a set of 15,176,580 Medline records. A collection of important fields was included (eg., title, abstract, MeSH terms, etc.) resulting in approximately 22 Gigabytes of text. This text was indexed by TLM in approximately 18 hours. An additional 24.5 hours was used to create a list of potentially useful tags such as **`<Sentence>, <Noun>, <ContainsDigit>`**. The algorithms used for division into sentences and the part-of-speech tagging are very simple and will be replaced in future uses of TLMTest.


Text Query User Interface (NRC-IIT Interactive Information Group)
File View Setting Help
Search
S
Search
Result

```
* * * 'X(' THA ')X'

Frequency: 637    Doc. Frequency: on request    Current Page: 1

Dist time: 4ms
[10000]
[317]  total hip arthroplasty THA
[33]  total hip arthroplasties THA
[25]  3 4 tetrahydroacridine THA
[24]  Total hip arthroplasty THA
[19]  tetrahydro 9 aminoacridine THA
[8]  </TI> <AB> Tetrahydroaminoacridine THA
[7]  threo beta hydroxyaspartate THA
[6]  </TI> <AB> Tacrine THA
[5]  effects of tetrahydroaminoacridine THA
[5]  effect of tetrahydroaminoacridine THA
[4]  </DP> <TI> Tetrahydroaminoacridine THA
[4]  Tokai High Avoider THA

Client: Regular user    Mode: Distribution    Total Time: 31ms    Task finished
```

Figure 1: A screenshot of a GUI client to TLM. The figure show the overall frequency for the query, the overall number of documents that contain the query, and the time to run the query. As well, in the center frame, the results are organized in a frequency distribution.

those word position lists reveals that the range of positions from the 43rd word to the 45th word describes a phrase containing both *cancers* and *tumours*. In the TLMTest collection, we find 128 occurrences in 120 documents. One of the resulting phrases is *tumours, including cancers*, which appears twice.

The second idea is that, unlike most search engines, TLM returns parts of documents. Usually, we don't want to see the whole document and only want a snippet, like Google's several word summary that appears under every result, or a passage (e.g., [8]). TLM does this by returning a range of two positions, say the position of the word *'tumours'* and the position of the nearby word *'cancers'* (cf., [3]). If they appear as the 43rd and 45th words respectively, the range is from 43 to 45. All query operations process these ranges of word positions.

The use of ranges leads directly to the third idea, that lists of ranges can be given a tag name. This idea is similar to macros or variables in some search engines, but its simple syntax is unique to TLM. For example, all the titles in a collection of documents can be described as a list of word position ranges. If that list is given the name **`<Title>`** or **`<TI>`**, the user can easily refer to the list of ranges in later queries. These special tags could be based on XML tags that actually appear in the text or can be defined by the user during the creation of queries. TLM stores these tags in an inverted index of ranges, just like the inverted index of word and separator positions.

The fourth fundamental idea is that the result of a query can be statistical information rather than just a simple list of discovered documents. This is a common text mining activity that is unique as part of a search engine. The simplest form of such statistical information is the frequency and document frequency of a query. For example, the query **`mdm2/i | hMDM2 | mouse double minute 2`** matches 7,811 words or phrases in the TLMTest. These matches occur in 6,248 sentences, and these sentences occur in 1,769 documents. A second useful form of statistical output is a frequency distribution of the distinct matches to a query. As an example, the query, **`blood near growth near vessel*`**, is a request for phrases that have the word *blood* near (within 10 words of) the word *growth* which together are near any word beginning with the six letters *vessel*. This would tell us that there are 280 occurrences of *blood vessel growth*, 90 occurrences of *growth of new blood vessels*, 52 of *growth of blood vessels*, 6 of *growth of new capillary blood vessels*, etc.

A fifth fundamental idea is that query speed is more important than conserving disk space. The availability of low cost massive storage, allows us to store multiple indexes that each accelerate different types of queries. Many search engines have indexes based on compression technology (e.g., [13]) and advertise that they require disk space that is only a small percentage of the original text size. TLM was designed to make many aspects of text mining fast. For example, there is a case-sensitive inverted index and a case-insensitive conversion index. These two indexes allow the user to specify specific capitalization patterns. The query **`hMDM2`** is a request for matches to exactly that term. In contrast, the query **`hmdm2/i`**, is a request for matches to *hMDM2*, *HMDM2*, *hmdm2*, *hMdm2*, etc. In addition, many common queries are pre-computed with results stored in a file. As a result of all this, the indexing file system can be four times the size of the original text (or more). If any part of that is removed, some type of common query would be slower to calculate.

All that follows and all that we have tried as part of LitMiner should be possible given a text analysis engine implementing these five ideas. The engine should have an inverted index of words, separators, and tag ranges. It should return parts of documents as ranges of word positions that match the query and should permit statistical post processing before giving the user the answer. Finally, it should prioritize fast text mining over conserving disk space.

2.2 A summary of the query language.

We have already presented a few example queries, with only a simple definition of the query language. More example queries are shown in Table 1. From these examples, it is obvious that the enhanced expressive power of TLM is in exchange for increased complexity. Most internet users would prefer Google's simple syntax to these complex queries. However, in many cases, the simplicity can be restored without losing the power, by using interfaces such as LitMiner that bury the query complexity behind GUI buttons.

In general, a TLM query is composed of words, tags, or separators connected by pairwise operators. All operators describe transformations of two lists of word position ranges into a resulting list of word position ranges. There are four major operators in TLM, as well as a syntax for tag definitions.

`interact* <Adverb>`	All adverbs that appear immediately after the word stem *'interact'*.
`<NounPhrase> > (<TI> > cancer)`	All noun phrases that appear in titles that contain the word *'cancer'*.
`interact* near protein*`	All passages that have the word stem interact near (within 10 words) of the word stem protein.

Table 1: Some example queries for TLM. See section 2.2 for an interpretation of the query language's operators.

The first major operator, and the one with the highest precedence is adjacency. When two words are separated by a space in a query, **`open heart`**, it forms a request for phrases that contain the first word followed immediately by the second word.

The second major operator is the 'or' operator. It simply merges two lists of word ranges. For example, **`mdm2 | MDM2`** is a request for all the word position ranges that contain just the word *mdm2* and all the word ranges that contain just the word *MDM2*. Then it merges those lists of word ranges.

The third major operator restricts answers to have two nearby parts. There are actually two forms of this operator, **`near`** and **`..`**. A query like **`word1 near word2`** is a request for all the word position ranges in which the two words appear within 10 words of each other. Similarly, a query could request that the two words be nearby and in order, **`word1 .. word2`**. It is a request for all the ranges in which the two words appear within 10 words of each other and *word1* appears first.

Both the **`near`** and **`..`** operators can be modified with specified distances. The simplest modification is to add '/' followed by a number. The modified operation **`near/2`** means that two ranges must be near, within two words. Similarly, **`../100`** means that the two ranges must be in order and within 100 words. The distance can be further modified by specifying a minimum distance as well. For example, **`near`**{**`4,10`**} means that the two ranges must be at least four words apart and up to 10 words apart.

The fourth major operator tests containment and was inspired by [3]. Considering two word position ranges, it is possible for them to overlap, for one to contain the other, or for them to be non-overlapping. In TLM, queries can force all answers to contain at least one example of another range. For example, the query **`<TI> > geopolitical`** is a request for word ranges that are whole titles, but only the ones that contain the word **`geopolitical`**. This query could be reversed and be a request for **`geopolitical < <TI>`**, ranges of length 1 with the word 'geopolitical', but only those occurrences inside titles.

The common search engine operators **`and`** and **`not`** were purposely omitted from this description, because they are not flexible enough for text analysis. In most query languages, **`and`** is a request for documents containing both of two words (or boolean expressions). In TLM, a query such as **`<DOC> > protein > gene`** is also a request for documents that contain both words. This approach is more flexible than the operator **`and`**, because it also applies to smaller document segments such as abstracts, or sentences, or phrases. For example, **`<TI> > protein > gene`**. Similarly, **`not`** typically is a request for documents that do not contain a particular word. In TLM, a query such as **`<DOC> /> protein`** would have the same effect, while also permitting **`<Sentence> /> protein`**.

The TLM query language also permits the definition of variables to hold partial query results. Multiple variable assignments can appear in a single query and the variable value is available even inside the same query but to the right of the first appearance. For example, the query **`($det = (the|a|an)) .. <$det>`** is a request for two determiners that appear near each other. As in this example, a variable name, which always begins with a $, is assigned the results of a query using an **`=`** operator. That variable then becomes a tag name for future queries by simply enclosing the variable name within **`< >`**.

3 TLM for Text Mining

TLM is a step closer to what users need. TLM queries have greater expressive power compared to most search engines, because a wider range of textual patterns can be specified. In exchange for much more complex queries, this greater expressive power allows queries to better correspond to real world entities. In biological (or any) text mining, there is a gap between a referent, such as a gene, and how we refer to that entity, i.e., the gene. In some sense, all queries are 100% accurate because they return exactly what they are supposed to do. Practically, though, they rarely find all and only what we want them to. TLM is not perfect, but it is a step beyond many search engines.

In this section, we will consider a few examples of how TLM can be useful for biological text mining. In none of these illustrations do we prove that TLM results are more accurate than previous results, only that they are similar. The point of this exercise is that TLM can do relevant text mining and can do it rapidly. We will leave it to future work to discover the best ways to use TLM to produce the highest accuracy, precision, and recall.

3.1 MedMiner

The goals of using TLM for mining the biological literature match many of those for MedMiner ([12]). MedMiner was designed to access ‘extrinsic’ information about genes. It was composed of three key components: internet-based querying of multiple databases, text filtering, and a carefully designed user interface. TLM could address the querying and text filtering. Our LitMiner system is our attempt to create a carefully designed interface.

In illustrating the value of their system, the authors considered a specific biological relationship (inhibition) between two genes, MDM2 and P53. They argued for their system on the basis of the completeness of the result, the amount of irrelevant information presented, the query complexity, and the running time.

3.1.1 More complete and fewer irrelevant sentences

TLM can be used to further increase the completeness of the results. As the authors noted, MedMiner will “miss relevant concepts if they are not represented in the keywords”. The interactive use of TLM with frequency distributions can partially address this problem.

The gene, MDM2, could be represented by any number of synonyms. A simple string of queries on TLM can tell us new terms to add. Each of the following queries has results that suggest new synonyms. The first query is a request for four words followed by MDM2. This query suggests that the two most frequent expansions of MDM2 are *murine double minute 2* and *mouse double minute 2*. The third column shows the accepted suggestions.

Query	time	Suggested synonyms
`* * * * MDM2`	330 ms	`murine double minute 2 | mouse double minute 2`
`MDM2*/i | MDM*/i (2 | ii/i)`	1120 ms	`MDM2 | mdm2 | Mdm2 | mdm 2 | MDM 2 | Mdm 2`
`hMDM2*/i | hMDM*/i 2`	200 ms	`hMDM2 | hmdm2 | hMdm2`

In a few seconds, we have a better query than simply **`MDM2`**. If we include the synonyms from Entrez Gene [10] and truncate important words, we produce a more complex query for MDM2 (Table 2). This query took about 30 seconds to create and about 9.7 s to run.

These queries can yield more complete results. In addition, like MedMiner, TLM’s results for inhibition displays the phrase or the sentence that indicates the relationship rather than merely identifying the document. It is also possible to highlight the gene names and inhibition phrases, because TLM returns the positions of matches.

3.1.2 Query complexity and running time

As shown in Table 2, the TLM queries created for MDM2 and P53 are rather complex as is the query for identifying some sort of relationship between genes. However, TLM provides user defined tags which greatly simplifies later queries. After the first three complex queries in the table have been submitted, the very simple fourth query can be submitted to ask for all phrases in Medline where MDM2 and P53 are said to interact.

The MedMiner time for a similar inhibition query was approximately 60 s and the equivalent PubMed query was 30 s when that paper was first written. It is not easy to compare these times with TLM. As a preparatory step, TLM requires between 2 and 60 s to perform each of the individual gene queries like those shown in Table 2. In addition, it requires approximately 6 minutes to process the interaction verb query in row 3 of Table 2. However, after that preparation, requests between arbitrary pairs of genes require an average of 4.8 s. This

Run time	Frequency	Query
9.7 s	11,010	`$mdm2 = MDM2/i | MDM/i 2 | HDM2/i | HDM/i 2 | MGC71221/i | P53/i bind*/i protei*/i | Mouse/i double*/i minute*/i 2 | murine/i double*/i minute*/i 2 | hMDM2/i`
1.4 s	195,159	`$p53 = tp53/i | tp/i 53 | Cys51Stop/i | TRP/i 53 | TRP53/i | p53/i | tp53s/i | Cys51Stops/i | TRP53s/i | p53s/i`
372 s	15,664,040	`$Iverb = (inhibit*/i | block*/i | reduc*/i | decreas*/i | acetylat*/i | activat*/i | target*/i | suppress*/i | stabiliz*/i | regulat*/i | phosphorylat*/i | modulat*/i | is/i ../2 conjugat*/i ../2 to/i | interact*/ i| inhibit*/i | destabiliz*/i | bind*/i | bound/i | associate*/i ../2 with/i)`
4.8 s	719	`<$mdm2> n/5 <$p53> > <$Iverb>`

Table 2: The queries (and times) needed to find the passages describing the interaction between P53 and MDM2)

suggests a scheme where gene queries and interaction verb queries are updated nightly, allowing users to get more complete pairwise responses in only a few seconds.

Overall, TLM meets many of the same goals as MedMiner but also provides improved performance (assuming some pre-processing) and a fast interactive solution to the problem of missing relevant concepts.

3.2 Finding interactions between sets of proteins

Blaschke et al. ([1]) went beyond a single pair of genes and described a text mining system that scanned 6728 abstracts looking for the pattern `<Protein> .. <InteractionVerb> .. <Protein>`, that is two proteins separated by a verb (or nominalization) that means some form of interaction. In their first example, they scanned for six different proteins separated by several different interaction patterns. The six proteins were, *pell, dorsal, toll, tube, spatzle,* and *cactus*.

Their scan of abstracts rediscovered nine known pairwise interactions between the proteins. The authors noted that frequency of the mention of a relationship can help determine which interactions to predict.

As an illustration, we attacked this same problem with TLM. Table 3 shows the queries created to represent parts of this task and their time to run. Each query was assigned to a variable for later use. The variable called `$Protein` is a list of capitalized and lowercase protein names. That query was combined with `$InteractionVerb` to find patterns of the type sought in the original paper, `protein .. verb .. protein`. In a total time of about 75 seconds, 15 million abstracts were searched and TLM rediscovered the interactions discovered in the original paper. The time for each component query is shown in Table 3.

The query in the fourth row resulted in 57 total phrases, 55 of which were unique. Of all fifteen automatically detected interactions reported in [1], the 57 results contain at least one example interaction for each. Six of the results identified the same relationship verb. TLM did not find exactly the same results, because it was searching all of Medline, it permitted matches across sentence boundaries, and it was only looking for results of length five words or fewer.

Using TLM to follow Blaschke et al.'s example required a few minutes and returned similar results with very few irrelevant phrases. In addition, these results included suggestions of the two known interactions between Pelle and Cactus and between Dorsal and Cactus that the earlier technique missed (*"Cactus inhibits Dorsal"*, *"Pelle proteins Phosphorylation of Cactus"*). The same 57 TLM results also reveal that there is a protein called

Run time	Frequency	Query
1.8 s	233,899	**`$Proteins = Pelle | Dorsal | Toll | Tube | Spatzle | Cactus | pelle | dorsal | toll | tube | spatzle | cactus`**
66.7 s	11,671,613	**`$InteractionVerb = acetylat*/i | activat*/i | target*/i | suppress*/i | stabiliz*/i | regulat*/i | phosphorylat*/i | modulat*/i | is/i ../2 conjugat*/i ../2 to/i | interact*/i | inhibit*/i | destabiliz*/i | bind*/i | bound/i | associate*/i ../2 with/i`**
5.8 s	57	**`<$Proteins> ../5 <$Proteins> > <$InteractionVerb>`**

Table 3: The queries (and times) needed to find interactions among Pelle, Dorsal, Toll, Tube, Spatzle, and Cactus.

”Twist” that interacts with Dorsal and another related protein called *”Kra”* (*”Dorsal-interaction proteins (Twist and Cactus)”*, *”Kra associates with Pelle and Tube”*).

We repeated this exercise for the authors’ larger protein list for cell cycle control in Drosophila. We constructed a single query (**`$CellCycleProtein`**) for the 91 proteins included in ([1]), using case insensitive searches. This created many irrelevant matches where both proteins were the same. In addition, many pairs of proteins were not matched because of intervening matches. To address these problems, we created one query for each of the protein names. This meant finding, for example, ranges containing *Myb* followed by an interaction verb, then by a cell cycle protein other than Myb.

For this second exercise, we reused the definition for interaction verbs that must occur between each pair. The TLM queries, including the redefinition of the variables CellCycleProtein and InteractionVerb, took a total of 6 minutes, 32 seconds.

The original paper ([1]) rediscovered 28 well-known interactions, 20 possible interactions, and missed one well-known interaction. In the list of 610 resulting phrases from TLM, we also found evidence for 27 of the 28 known interactions and all but five of the possible interactions. The main interaction missed by TLM was between *cdc2* and *twine*. However, TLM did detect the interaction between *cdk* and *p21* that the original paper missed.

TLM clearly supports the extraction of significant facts from large text collections. Specific entities can be identified and relationships between those entities can be correctly discovered. TLM can achieve these and similar tasks in minutes. This is fast enough to allow a tolerable interaction between the user and the text.

4 Text categorization

Another important text mining application is text categorization. Researchers have applied text categorization to label Medline abstracts as relevant or not to some task (e.g., [6]; [11]). For example, in PreBIND, text categorization was used to select papers about protein-protein interactions for later human curation.

In principle, this is similar to search engine retrieval. However, text categorization uses additional computation (slower) to improve the precision and recall (and accuracy) as compared to the results of search. A search on Google might return 100 results with only 10 of them being relevant. In that search, the precision would be 10%. If the search results completely missed 190 other relevant documents, the recall of that search would be 5%. In contrast, text categorization often results in 65% recall and precision ([11]) or even 90% recall and precision ([6]).

The other difference with Google, besides precision and recall, is the time necessary to produce the results.

Google often reports millisecond response time whereas Donaldson et al. [6] quotes a time in days to apply a text categorization model to 12 million Medline records. With other techniques ([11]) text categorization like levels of accuracy can be achieved much more quickly. Even in that second case, though, the authors suggest using a cluster of several processors to achieve fast learning and application of that learning.

TLM can be used to achieve high recall and precision without requiring days or multiple processors. To illustrate this potential for categorization, we recreated the experiment described in [6]. For this experiment, we used the following technique. From the training examples of protein-protein interaction abstracts, we extracted two general types of features: *"A appears in the document"*, *"A near/5 B"* In those features, A and B refer to one or two word phrases. Among those thousands of possible features, we selected the 5000 that individually were most diagnostic in determining whether a document was a positive example or a negative one. As in the previous study, we used information gain to select those features. Then we applied Ripper ([4]) to learn a boolean expression of the features that would select the positive documents while excluding the negative ones. These boolean expressions were translated into acceptable TLM queries allowing rapid application across all of Medline.

We divided the development set into 10 folds and performed cross-validation, each time training on 90% and testing on the remaining 10%. As a result, we found a precision of 89% and a recall of 86%. Both of these numbers are lower than, but similar to, the results reported in [6]. From past studies, we can expect this new technique to always under-perform Support Vector Machines (e.g., [7]; [14]). However, we expect the new technique to always outperform techniques such as those in [11] again based on performance in those same past studies.

In addition to the high precision and recall, TLM plus Ripper was fast. In our illustration, a single query that resulted from applying Ripper required an average of 85.25 s when submitted to TLM. This is much shorter than the hours necessary to apply an SVM. As well, it is much faster than would be possible with any non-index based technique. In fact, in contrast to the suggestions in [11], we are able to achieve reasonable performance with a single CPU and several users.

As for the case of identifying specific interactions, we have only shown that TLM can be used to achieve similar results quickly. More work has to be done to devise and evaluate a scheme to create consistently high recall and precision while still requiring only a few minutes.

5 Discussion

A text analysis engine is a necessary tool for the future of text mining in biology and other fields. In contrast to search engines, in a text analysis engine, queries are composed of not just words, the results are not just documents, and the final answer is not just a list. Queries can contain punctuation, tags, variables, etc. Results can be documents, sections, topic-based passages, paragraphs, sentences, phrases, etc. The final answers can be a list or could be multiple levels of frequency counts or a frequency distribution.

One example of such a text analysis engine is TLM. It has an inverted index of words, separators, and tag ranges. It returns parts of documents represented by ranges of word positions that match the query and permits statistical processing of the results. As well, it favours speed over conserving disk space.

In our illustrations, we have taken classic examples of text mining in biology and shown that TLM can match the reported performance and can do so very quickly. We have not shown TLM's results to be conclusively better or worse than earlier results, only that they are similar and fast.

CONTRIBUTIONS & ACKNOWLEDGEMENTS

All the code for TLM, except for a public domain sdbm implementation (by J. Chapweske), was written at NRC (engine by the author; GUI by Chengbi Dai). Chengbi Dai's client GUI is shown in Figure 1 above. All the examples in this paper were created by the author. The idea to use RIPPER for query based text categorization arose in discussion with Berry de Bruijn.

I would also like to thank all LitMiner team members for the motivation to create TLM, both on the computer side (Berry de Bruijn, Lynn Wei, Darrell Ferguson, Norm Vinson, and Jeff Demaine), and on the biology side (Hung Fang, Annie Law, Qing Liu, Maria Moreno, Brandon Smith, and Roy Walker).

References

[1] C. Blaschke, M. Andrade, C. Ouzounis, and A. Valencia. Automatic extraction of biological information from scientific text: Protein-protein interactions. In *Intelligent Systems for Molecular Biology*, pages 60–67, 1999.

[2] James P. Callan, W. Bruce Croft, and John Broglio. TREC and Tipster experiments with Inquery. *Information Processing and Management*, 31(3):327–343, 1995.

[3] Charles L. A. Clarke and Gordon V. Cormack. Shortest substring retrieval and ranking. *ACM Transactions on Information Systems*, 18(1):44–78, 2000.

[4] William W. Cohen. Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference*, 1995.

[5] O. de Kretser and A. Moffat. Effective document presentation with a locality-based similarity heuristic. In *Proceedings of the Twenty Second International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 113–120. ACM Press, 1999.

[6] I. Donaldson, J. Martin, B. de Bruijn, C. Wolting, V. Lay, B. Tuekam, S. Zhang, B. Baskin, G.D. Bader, K. Michalickova, T. Pawson, and C.W. Hogue. Prebind and Textomy–mining the biomedical literature for protein-protein interactions using a support vector machine. *BMC Bioinformatics*, 4(11), 2003.

[7] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *ECML-98, Tenth European Conference on Machine Learning*, 1998.

[8] M. Kaszkiel and J. Zobel. Effective ranking with arbitrary passages. *Journal of the American Society For Information Science and Technology*, 52(4):344–364, 2001.

[9] J. Martin and B. de Bruijn. Litminer. www.litminer.ca, 2003.

[10] U.S. National Library of Medicine. Entrez gene. www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=gene.

[11] B.P. Suomela and M.A. Andrade. Ranking the whole medline database according to a large training set using text indexing. *BMC Bioinformatics*, 6(75), 2005.

[12] L. Tanabe, U. Scherf, L. Smith, J. Lee, L. Hunter, and J. Weinstein. Medminer: an internet text-mining tool for biomedical information, with application to gene expression profiling. *BioTechniques*, 37:1210–1217, 1999.

[13] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Morgan-Kaufmann Publishers, 2nd edition, 1999.

[14] Yiming Yang and X. Liu. A re-examination of text categorization methods. In *Proceedings of SIGIR-99, 22nd ACM International Conference on Research and Development in Information Retrieval*, 1999.