Journal of Universal Computer Science, vol. 19, no. 3 (2013), 406-427
submitted: 22/9/11, accepted: 28/1/13, appeared: 1/2/13

A Decoupled Architecture for Scalability in Text Mining Applications

Jorge Villalon
(Universidad Adolfo Ibanez, Santiago, Chile
jorge.villalon@uai.cl)

Rafael A. Calvo
(University of Sydney, Australia
rafa@ee.usyd.edu.au)

Abstract: Sophisticated Text Mining features such as visualization, summarization, and clustering are becoming increasingly common in software applications. In Text Mining, documents are processed using techniques from different areas which can be very expensive in computation cost. This poses a scalability challenge for real-life applications in which users behavior can not be entirely predicted. This paper proposes a decoupled architecture for document processing in Text Mining applications, that allows applications to be scalable for large corpora and real-time processing. It contributes a software architecture designed around these requirements and presents TML, a Text Mining Library that implements the architecture. An experimental evaluation on its scalability using a standard corpus is also presented, and empirical evidence on its performance as part of an automated feedback system for writing tasks used by real students.
Key Words: text mining, software architecture, automatic feedback
Category: M.1, L.3, D.2.11, I.7

1 Introduction

The growing availability of textual information comes with an increasing need to obtain useful information from it. Although search engine products are recognized as one of the strongest areas in the software industry, new functionalities are required to move this frontier further, allowing for more effective ways to classify, visualize, and summarize the mass of unstructured content available. The importance of this type of innovation is reflected in the vast number of projects on different aspects of Computational Linguistics and Information Retrieval.

Beyond search engines, Text Mining (TM) technologies are at the heart of many of the novel functionalities incorporated in real-world applications. For example, Fan et. al. reported the discovery of novel relationships between magnesium deficiency and depression using automatic concept extraction to link different documents with shared concepts, and by connecting more than one level of indirection [Fan et al., 2006]. Another tool by Osinski helps dealing with hundreds of search results by clustering them into tens of labeled clusters reducing information overload [Osinski and Weiss, 2005].

However, the techniques required to process documents and extract patterns can be very expensive in computational cost, affecting the possibilities of applications to scale up in real scenarios. The scalability of TM applications can be mapped in three dimensions: the size of the document corpora (going from one short document to millions of long documents), the time available to process the documents (going from an occasional execution to very high frequency real-time execution) and the computational cost of the technique required by the application. The first dimension can be found in statistical techniques that use previous knowledge like Latent Semantic Analysis [Deerwester et al., 1990], in which computational time is quadratic to the size of the corpus. The second dimension comes from real-life scenarios in which users expect an immediate response from the system and also behave in waves, demanding high usage peaks for short periods of time, like educational scenarios in which TM tools are used to provide feedback on writing assignments that have a due date. The third dimension comes from the complexity of Computational Linguistic techniques, in which complex constructs are built on top of more simple constructs, therefore the whole chain of techniques must be processed before obtaining a single pattern. An example of such constructs are 'typed dependencies' [de Marneffe et al., 2006], which require sentences to be parsed in depth, which in turn requires part of speech information from each token in the sentence.

We propose a decoupled architecture in which the processing of documents is separated according to the demands of different techniques, while at the same time is fast enough so basic patterns can be obtained immediately for user consumption. The architecture contributed in this paper is implemented in a software library called 'Text Mining Library' (TML), which integrates information retrieval, indexing, clustering, part-of-speech tagging and latent semantic analysis. We have used the area of automatic feedback for writing as a case study to demonstrate how the architecture behind TML and its Java implementation performs in this area. The use of TML within Glosser [Villalon and Calvo, 2011], a tool for automatic feedback of writing assignments is described.

Section 2 describes the TM process and outlines the architectural requirements for TM tools. Section 3 presents the design and implementation of the TML architecture and how challenges are tackled for each sub-step. Section 4 describes a case study of the architecture applied to a real application. Section 5 discusses an evaluation of this architecture and subsequent results. Section 6 concludes.

2 Background

Text Mining (TM) aims to develop methods and tools for the discovery of non trivial patterns within text. This aim overlaps with the broader field of Data

Mining (DM), but focuses the object of study on unstructured data and particularly text. TM was originally proposed as 'Knowledge Discovery from Text' by Feldman, and after more than a decade, its exact definition is still being argued among researchers, particularly with regard to a precise definition of 'non triviality' [Feldman and Dagan, 1995].

Non-triviality is the degree of novelty of the information extracted, which in turn allows humans to make valuable interpretations. A key aspect of TM tools is that they must link together extracted information to form new facts or new hypotheses, that will then be analyzed by humans in more conventional ways [Hearst, 2003]. In other words, the extracted (and linked) information is valuable when it allows humans to form new hypotheses or discover new facts. Even though some authors consider TM to be limited to the extraction of relevant information [Hotho et al., 2005], many leading researchers insist that the TM process corresponds to the whole of exploratory data analysis [Hearst, 2003].

The growing demand for TM technologies and its integration into real-life applications have posed important challenges on the scalability of such systems. Text processing techniques from Information Retrieval and Computational Linguistics can have a very high computational cost, making real-life scenarios a difficult challenge to tackle for developers from the TM area.

2.1 Previous work on architectures for TM

Only a few academic studies analyze the requirements for the architecture of their own TM tools. Holzman presented the design of a TM tool developed in C++ that implements part of the TM process, not focusing on an iterative approach but on a single execution of a model [Holzman et al., 2004]. Garcia proposed an Object Oriented architecture which defines a clear structure for the logical modules that every TM tool should have [Garcia Adeva, 2006]. In both cases the designs do not tackle the scalability issue.

Several TM tools have been described in the academic literature, while others are commercially available. All these tools have been designed for a specific purpose and do not describe their architectures. Some of them report the inclusion of parallel programming techniques for better efficiency.

OpenNLP [Baldridge et al., 2002] is a collection of open source tools for Natural Language Processing, its focus is feature extraction via the annotation of text. It is openly available and it implements a multi-threaded design to speed up text annotation. OpenNLP does not implement the whole TM process but contributes a flexible design for the integration of several other technologies.

Mallet [McCallum, 2002] is a complete package for Text Mining that implements several steps of the TM process, and includes DM algorithms for applications such as document classification. Mallet was not designed as an extensible library but as a complete application, it includes several relevant algorithms including both NLP and statistical methods.

RapidMiner [Mierswa et al., 2006] is a complete application suite, it is proprietary but a community version is available for research with limitations to its performance and features.

LingPipe [Alias-I, 2011] is an open source library, that can be extended, however its license does not allow any commercial use for applications that were built with it. A common problem of open source projects that are limited by their licensing options is a lower participation from the research community, hence the incorporation of new algorithms depends on the available resources of the company that maintains it.

A different type of tools are Data Mining applications that include plugins for using text as incoming data. Weka [Holmes et al., 1994] , Matlab and R [Feinerer et al., 2008] are good examples of this approach. These applications focus on the implementation of DM algorithms, and have big communities of both developers and researchers that contribute to their code quality and new algorithms.

Only two academic studies describe general requirements for TM libraries and frameworks [Silic et al., 2007, Williams, 2003], which can be summarized as:

- Simplicity for the developer: This aspect includes characteristics such as rapid application development (the structures and the methods of the library should enable intuitive and fast integration into end-user applications), code reuse (facilitating reusing already implemented methods), and modularity (the modules should be able to change without affecting other modules).

- Rapid research cycle for researchers: Users should be able to easily integrate and test new methods within the library.

- Scalability (Computational efficiency): Design and implementation should be efficient in memory usage and CPU time.

Despite the growing number of TM tools, none of them has yet focused on the scalability problem that certain scenarios pose to real-life applications.

2.2 Writing feedback: an example application

Writing is an important activity in most professions and university graduates are expected to show proficiency across different genres. Regrettably, assessing written assignments and providing students with timely and useful feedback, are time consuming tasks. As a consequence, many researchers are building tools for automatic assessment and feedback.

Producing useful feedback for students requires the recognition, processing and visualization of textual patterns that are related to the writing quality of essays. The nature of the problem of generating automatic feedback encompasses many text mining techniques making it an apt case study for a TM architecture.

Early automated feedback systems included 'Writers Workshop', a system developed by Bell Laboratories, and 'Editor'. However, both systems had a focus on surface features of writing quality (grammar and style), which have been shown to have limited pedagogical benefits [Beals, 1998].

Modern tools that implement pedagogically sound methodologies require complex TM techniques in order to provide useful feedback. For example, Select-a-Kibitzer (SaK) [Wiemer-Hastings and Graesser, 2000], a more recent writing tutoring system is based on Flower's notion of voices that speak to the writer during the process of composition [Flower, 1994]. SaK uses avatars to give the impression of giving each voice a face and a personality. It uses Latent Semantic Analysis (LSA) described later to calculate the average distance between consecutive sentences and provide feedback on the overall coherence of the text. It can also analyze the topic of a sentence, identifying clusters of topics amongst the students so when a new topic arises the student can be asked for an explanation or reformulation. In this paper we present the case study of the TM features behind Glosser [Villalon and Calvo, 2011]. From a pedagogical standpoint, the feedback provided by it aims at triggering students' reflection on their writing.

3 A decoupled architecture for Text Mining

TM can be seen as an iterative process with five main steps: Document selection, document pre-processing, feature extraction, model creation (corpus analysis), and data manipulation (operate models) [Hearst, 1999, Hearst, 2003, Hotho et al., 2005]. Figure 1 shows the standard TM process in which the steps form a loop. First, a set of documents is selected as a corpus from a repository of documents. Second, each document is decomposed in its most basic components called tokens, and some basic statistics are computed (e.g. term frequency). Some optional sub-steps can be performed at this time, like term stemming and removing non semantic terms. The result of this stage is every document represented as a term frequency vector. Third, each document is represented as a set of common features for the whole corpus. Features can be obtained from the term frequency vectors or extracted from the text using Information Retrieval and Computational Linguistics techniques. A common feature from IR is the combination of the frequency of words in a document and their frequency in the whole corpus, while a common feature from CL is the grammatical/syntactical structure of sentences among others. Fourth, the corpus is processed using Data Mining methods which create a model of it that can be operated to extract information. One example is to create a Semantic Space using Singular Value Decomposition so semantic distances can be calculated between documents or terms. Finally, the model is operated to obtain useful information that will help the user on selecting the documents for the next iteration of the TM process. Common tasks in this step are the clustering of documents by topics, and the prediction of categories for unclassified documents.

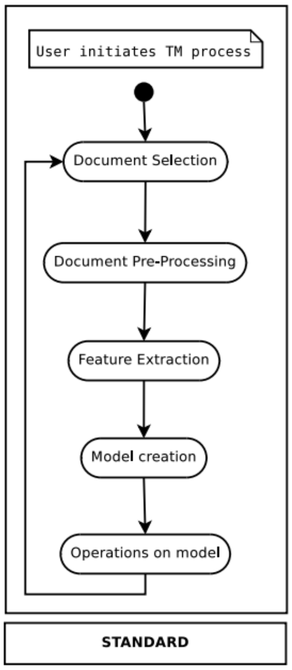

Figure 1: Text Mining process

3.1 Decoupling the TM process

An important aspect of the architecture is the decoupling of the processes, so they can be run in parallel. Figure 2 shows how the process was separated, so the initiators change. In a standard TM process, the user initiates it by selecting a corpus, in TML there are three initiators: The user selecting a corpus, adding a document and an independent process. This change allows the three processes to run in parallel, possibly in different hardware platforms, facilitating scalability.

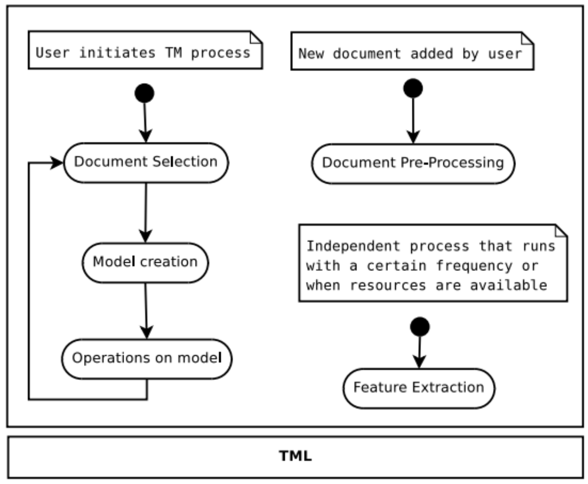

Figure 2: Text Mining process decoupled in TML's architecture

3.2 Document selection

Document selection must be very efficient and scalable as it is the first step to be performed when users interact with a system. Selecting documents can be done in two ways: by matching their content or meta-data to the user requirements, or as the result of a previous TM iteration, for example selecting all documents that belong to a particular cluster.

The first architectural decision in TML was the use of a search engine as the container for all documents used in the analysis. Search engines are optimized for managing large number of documents and retrieving them based on queries. TML is integrated with Apache's Lucene search engine, a popular open source search engine written in Java which is highly scalable.

Therefore, the most basic structure in the architecture is a Repository, which represents all available documents and their meta-data at three levels of granularity: documents, paragraphs and sentences. Repositories are implemented in TML using a Lucene index. When a new document is added it is split into paragraphs and then into sentences, each passage is then indexed so basic statistics are calculated for all of them. Basic information is stored in seven Lucene fields for each passage, table 1 shows a description for each field. The three granularity levels allow to create corpora from a single document, which can be very useful for processing books and/or analyzing essays.

Document selection is therefore performed by searching documents within a Repository. A set of documents is called a 'Corpus', and corresponds to a collection of search results obtained from running a query on the search en-

Table 1: Meta-data stored as default in a TML Repository.

Field	**Description**
External Id	An id that identifies the document uniquely
Type	If the passage is a document, paragraph or sentence
Content	The text of the passage and a vector with its calculated term frequencies
Parent	External id of the document to which the passage belongs (null if it is a document)
Reference	External id of the passage to which the passage belongs. A paragraph if it is a sentence, a document if it is a paragraph or null otherwise
Insertion Time	A time stamp indicating when the passage was created
URL	A URL indicating how to access the document (could be a path in a filesystem)
Title	Name of the file or page for the document. Paragraphs and Sentences have an automatically assigned title indicating its number and the document to which they belong.

gine. Queries are expressed using Lucene's query syntax and can use any of the fields, for example the query *'type:document AND content:English'* will find all documents containing the word 'English'.

3.3 Document preprocessing

Document Preprocessing includes several sub-steps including: Tokenization, Stop-words removal, Stemming and terms counting. It also corresponds to the second step in a common TM process, and deals with the creation of the most basic representation of text as a set of tokens. These processes are very efficient and are implemented in several TM tools and libraries, in TML these are implemented by Lucene which has a very good performance on preprocessing documents at insertion time. However, even though preprocessing a single document can be very fast with modern libraries, a large number of documents to be preprocessed simultaneously may affect the whole system performance.

In order to tackle the problem of a large number of documents to be preprocessed by the system, document preprocessing was decoupled from the TM process in the proposed architecture. Preprocessing is performed whenever a new document is added to the repository, and not as part of the TM process itself. In this way, a user can upload a new document that will be immediately preprocessed before it is inserted in the repository, and therefore will be immediately available for document selection in the process.

By decoupling this process, concurrency, a major issue for large scale systems is tackled. Concurrency problems occur when two processes are demanding the same resource, for TM the only shared resource are the documents, and the problematic scenario would be when a user is inserting a new document while at the same time another user is selecting documents to build a model. The proposed architecture solves this problem because reading and writing from the repository are separated processes, with different initiators. In TML every query performed to select documents uses a 'read-only' version of the repository which contains all the documents committed before the query was run, documents being inserted at that time will be available in subsequent queries.

As repositories in TML are implemented using Lucene, it also provides detailed configuration for the tokenization process, for example, it can be customized for different types of documents identifying term within the text, including abbreviations, URLs, email addresses and other tokens. Other customizations include the stemming rules, or replacing a word by its stem, that change across languages. Lucene also implements a range of preprocessing techniques such as Porter's stemmer and stop-word removal, and it can be extended by implementing an 'Analyzer' interface. TML benefits from Lucene's flexibility by allowing any analyzer implemented in that way to be plugged into its TM process.

3.4 Feature Extraction

Feature extraction (FE) is used to reduce a text passage to set of features. Features can be 'statistically based', where the text is represented as a number, such as the frequency of its terms, or 'linguistically based', where the features may be based on the syntactical function of its terms.

Feature extraction can be one of the most expensive processes due to the complexity of some of its algorithms. Counting terms for example can be done in a very efficient way, as text is traversed in a single direction while tokens are identified. Probabilistic grammar parsing on the other hand can be very costly because text must be traversed back and forth depending on the probabilities found as new tokens appear. In our own experiments, statistically based features can be calculated in less than a second for a short document of 500 words, while parsing its sentences may take up to one minute.

As it is impossible to predict what features would be required in a specific application, the scalability of a TM application will depend on the features required by the TM process initiated by the user.

3.4.1 Statistically based features

The most basic statistical approach is known as 'bag of words' because it represents a text passage as a vector indicating the frequency of its terms, not taking into account word order. In this way, a set of documents (corpus) can be represented in a Vector Space Model (VSM), a collection of vectors representing the documents with their dimensionality defined as the collection of terms representing the whole vocabulary used in the corpus. A VSM is represented by a $m * n$ matrix A with a_{ij} calculated as the weight of term i in document j.

Several term weighting schemes are used for specific purposes, and TML implements a set of weights by combining a local and a global component, particularly those reported in [Dumais, 1990]. In order to calculate such weights, term counting must be done at document (or passage) and corpus levels. These computations present scalability challenges when big corpora is used, however, the proposed architecture with the document preprocessing decoupled from the TM process present performance savings over traditional approaches.

3.4.2 Linguistically based features

Linguistic features are those obtained from a linguistic analysis of the text. The purpose of Linguistic Analysis is to understand a piece of text, in order to transform it into a new form which is useful for some purpose. An example is word disambiguation, as many words and phrases are ambiguous, a deeper knowledge is required in order to properly identify its meaning.

From a linguistic perspective language is seen as a layered structure in which larger textual units are understood as a combination of smaller ones. Linguistic Analysis aims to understand each layer, process that is usually subdivided in two subtasks: Syntax and semantics. Syntax describes how the different elements of a textual unit can be combined, while semantics describes how to calculate the 'meaning' and is represented using grammars.

The most basic level in Linguistic Analysis is the identification of words' Parts of Speech (POS). POS are linguistic categories defined by the syntactic and morphological behavior of words, such as 'nouns' and 'verbs'. For example, the sentence 'John loves Mary' contains three words, its POS annotated equivalent is 'John/NNP loves/VPZ Mary/NNP' which indicates that both John and Mary are nouns, particularly proper nouns, and loves is a verb in its third person singular present form.

Recent studies in LA are obtaining more semantic information from grammatical trees. An example of this are 'typed dependencies' which identify how words are related to each other from the argumentative perspective. The grammatical tree is parsed using rules that determine the argumentative roles of each word in a sentence [de Marneffe et al., 2006].

The hierarchical layered structure of linguistic features forces the calculation of lower levels in order to obtain the higher ones, making the process computationally expensive. For example, a short document of 500 words can take over a minute to be grammatically parsed at the sentence level, if an application has to wait all this time before presenting an interaction with the user it will not meet minimum usability standards.

In TML, scalability has been addressed by hierarchically decoupling extraction processes according to their complexity (see Figure 2). In this way, easier and less expensive processes are run immediately, while more expensive processes can be set to run offline according to computer power availability.

The core concept in FE are 'Annotations', which correspond to the extracted information of a text through NLP means. Annotations in TML are textual data obtained by running an Annotator on a text passage, for example given the text passage 'John Loves Mary', its corresponding POS Annotation will be 'John/NNP loves/VPZ Mary/NNP' that can be obtained running a POS parser Annotator. Annotators in TML can be easily extended by implementing the 'Annotator' interface, which requires two levels of information for basic and more complex annotations. All annotators require a name (e.g. 'PennTree') and to implement a method to obtain the string with the annotated version of a string passed as parameter. An optional initialization method will be called once when TML is firstly executed in case the annotator requires it. If the annotator requires structured info, the returned string will correspond to XML, and an extra method allows the author to provide a schema to validate and interpret the data. Finally, an easy method to extract pieces of text from the passage given a label is also provided, this allows a simple way to extract for example all nouns from a sentence.

TML also separates the process of reading and writing annotations in order to improve scalability as shown in Figure 3. Reading annotations happen when an application requires annotated text. At this point it will use the Annotator's interface to get the annotated text, if it is available in TML's annotations database it will be returned immediately, if it is not TML will throw an exception. This architecture allows modern applications to use asynchronous calls to TML that would never freeze while performing heavy calculations, allowing programmers to create highly responsive applications. Writing annotations on the other hand must happen when computer power is available, and as the processes are decoupled, it could run in a completely different hardware (or in many). Both processes are linked through a relational database TML uses to store annotations. As databases manage concurrency very efficiently, several applications (or instances of one application) can be reading from the same source, while the slower processes make new annotations available.

Writing annotations is performed by an Annotator Manager that runs according to a configuration file, indicating what annotators to use. Whenever a new document is added to a Repository, the new document is also added to the annotations database, then when the manager runs it will look for all new documents that have no annotations yet and run all the annotators for each

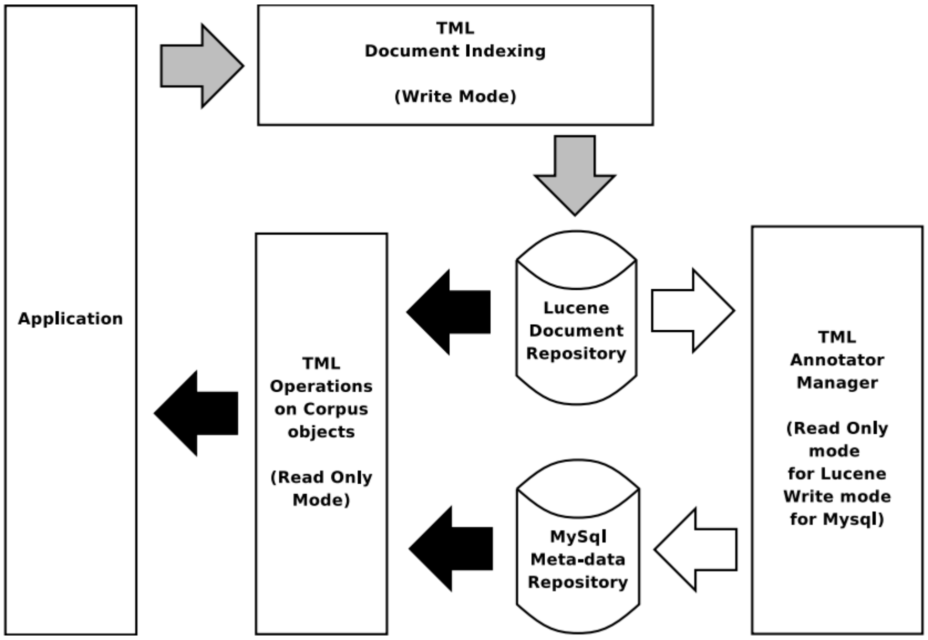

Figure 3: TML's asynchronous reading and writing operations for annotations

document. An atomic locking at the package level managed by the database allows many instances of the annotator manager to run simultaneously, facilitating horizontal scalability.

3.5 Model creation

In DM a model correspond to a representation of the 'knowledge' extracted from a data set, for TM it corresponds to the knowledge obtained from a corpus. Models can be created using DM methods such as neural networks, bayesian networks and support vector machines, or IR methods such as Semantic Spaces and Topic models.

Regardless of their type, models are computationally expensive to create because most algorithms are highly complex and the amounts of data are of a considerable size, therefore the biggest challenge for model creation is scalability.

3.5.1 Data Mining models using Weka and Matlab

In order to create models based on DM methods, TML can integrate in its process methods from two of the most important DM packages available today: Matlab and Weka. Matlab is a popular commercial tool used for DM both in the industry and in academic environments, Weka is also a very popular package that is open source and it is also integrated in other tools such as Rapid-Miner. TML can use Matlab methods in two ways: Using TML from Matlab or vice versa. For the former, a simple class serves as an interface from Matlab that returns any dataset from TML obtained with previous steps. The latter is implemented through the Java interface for Matlab, with which any Java program can invoke the Matlab runtime. TML can also use Weka methods in two ways: Using TML exported data from Weka or using Weka from TML. The former can be easily done executing all previous processes using TML and then exporting the data set into the Weka format. The latter is used by using Weka's Java interface from a new TML functionality.

An integration with Weka from TML was made using the last method mentioned. Every set of documents and their corresponding features obtained with TML can be represented as a set of Weka instances, and therefore any of its method can be invoked to build a model.

3.5.2 Semantic Spaces

A second type of models are *Semantic Spaces* (SS), which are derived from the VSM using matrix factorization techniques. SS models were initially proposed in the statistical technique known as Latent Semantic Analysis, developed by Deerwester et. al. to improve document retrieval performance [Deerwester et al., 1990] . By deriving a SS, two linguistic problems in IR were overcome: synonymy and polysemy. The former occurs when two words share the same meaning and the latter when a single word has more than one meaning. In a SS these meaningful relations can be inferred from a corpus.

A SS is created from a VSM's term-document matrix which is then factorized using linear algebra techniques such as Singular Value Decomposition (SVD) and Non Negative Matrix Factorization (NMF). After the factorization terms and documents that are semantically related are clustered together [Deerwester et al., 1990]. When used for IR, user queries are *projected* in the semantic space as *pseudo document*s, hence are placed close to all other conceptually related documents.

Mathematically the creation of a semantic space comprises three basic steps: First, it starts from the VSM of a corpus, this is a term by text passage matrix A with values $a_{i,j}$ indicating the weight of the ith term for the jth text passage. Second, the matrix is decomposed using a factorization technique (e.g. SVD) in three other matrices, as shown in equation 1, which represent an orthogonal space where the vectors are sorted by the amount of information they provide [1]. The columns from matrices U and V are orthogonal, and form a new base for

[1] NMF decomposes the matrix A into two other matrices, however it can be written in the same way as SVD taking Σ as the identity.

the space. Finally the dimensions of the space are reduced, which is the key step in this technique. This is done by keeping only k values in matrix Σ and setting the rest to 0, so when multiplying back an approximation of the original matrix is obtained as shown in equation 2.

$$A = U * \Sigma * V^t \tag{1}$$

$$A_k = U_k * \Sigma_k * V_k^t \tag{2}$$

The resultant model is again a VSM but in which terms and documents were projected only on the k most important dimensions.

Factorization algorithms are computationally expensive, and such cost grows exponentially with the size of the corpus. For example, given a $m * n$ matrix, SVD's complexity is $O(mn^2 + m^2n + n^3)$, while NMF's is $O(nmk)$ per iteration until it converges.

3.5.3 Model caching for scalability

Regardless of the model type (IR or DM), its computation is generally expensive, however, in real-life scenarios such models can be reused. For example, if several essays are going to be projected on a semantic space to calculate distances between their paragraphs and prescribed readings, the space containing the prescribed readings should be calculated only once for all students in the same course.

In TML, whenever a model is required, a unique label based on its corpus (selected documents) and model parameters is calculated, then a file system based cache is checked for its existence and if the model was already calculated it is read from the file system, otherwise it is calculated and the saved on the file system cache.

3.6 Operations on models

The execution of a model correspond to exploit the information obtained in the model for a particular application. This is the most common activity for programmers that want to use TM techniques in their applications. For example, a Semantic Space created with all the sentences from a document can be used to rank the sentences by their importance in the space for summarization, or to calculate the semantic distances between consecutive sentences in order to calculate how cohesive a paragraph is.

4 Case study: Glosser

The flow of information in Glosser and how it interacts with TML is as follows: Students upload their document (or commit a new version in an online writing environment) and ask for feedback immediately. After the students reflect on the feedback, they react to it by revising their document and producing a new version, and starting a new iteration by asking for feedback again. Such a scenario must cope with the immediacy expected by the users and the high load that occurs when the assignment's due date approaches.

This feedback is in the form of descriptive information about different aspects of the document. For example, by analysing the words contained in each paragraph, it can measure how 'close' two adjoining paragraphs are. If the paragraphs are too far this can be a sign of a lack of flow or what is called lexical in cohesiveness and Glosser flags a small warning sign.

Glosser provides feedback on several aspects of the writing: structure, coherence, topics, keywords. It also has a section that analyzes participation of individuals in a group. Each of these areas is identified by a tab on the homepage of the web application. The Questions section contains triggers to prompt student reflection on a particular set of features of the composition, and can be customized for each course or activity. The instructions section describes how to interpret the feedback, which is provided as "gloss" on the lower half of the page. The feedback is an alternate type of visual or textual representation of features that have been automatically sourced from the text. The history of versions provides a record of the writing process , which includes information on the time of each revision and the author responsible for it.

Glosser uses TML to implement all its tabs, and as its process is decoupled, it is separated in two parts: Indexing and Operating. Indexing is the process of retrieving the documents so they can be glossed, it has two steps. Firstly, Glosser harvest all versions of documents from Google Docs through its API, and inserts them in a database. This happens on demand, every time a user clicks on a document to gloss it. Secondly, an independent and parallel process which runs every 15 seconds inserts the documents in a TML Repository. Operating corresponds to the selection of a corpus, creating a model and operating it.

One example is the 'Coherence' tab. In this tab, a semantic space is created using the paragraphs of the essay, the semantic distances between each consecutive paragraph are then calculated. The paragraphs are shown on the screen, and whenever the distance between two consecutive paragraphs is above a threshold, a warning signal between the paragraphs will be shown and both will be highlighted as shown in Figure 4.

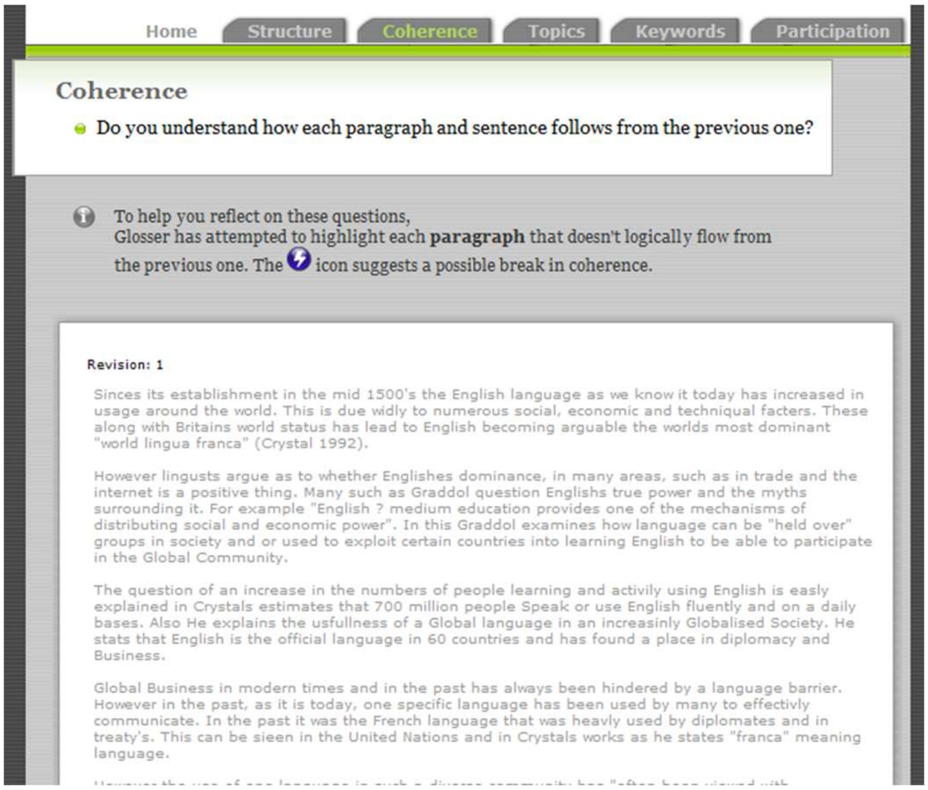

Figure 4: Glosser coherence tab

5 Evaluation

Scalability has a standard evaluation method, which is to measure the time it takes the system to perform a single but complete task. The task is executed using a growing number of documents so its growth can be analyzed. Each execution must be repeated to avoid external influences from the operating system such as memory management or I/O problems.

To evaluate the architecture's design, we compared TML with a standalone Weka [Holmes et al., 1994] implementation as a baseline. Weka has a built-in text mining feature that produces a VSM representation of a Set. Although the processing of text in the two architectures follows a similar sequence, the differences on storage and indexing strategies produce significant performance differences. Next is a discussion of the sequence and the computational complexity of each step and results from experiments using two different textual corpora are used to evaluate the implementation.

5.1 TM process time

We can summarize the TM process time separated in its sub-steps in the following equation:

$$T_{total} = T_{sel} + T_{pre} + T_{fe} + T_{mod} + T_{op} \tag{3}$$

Where T_{sel} is the time for obtaining the Set, T_{vsm} is the time to create the VSM and T_{mod} the time to compute the model and store it if the algorithms allows, T_{use} is the time of using the model and using it within the business logic of a real system. Where T_{sel} is the time for selecting the corpus, T_{pre} is the time to pre-process the documents, T_{fe} is the time to extract the features and T_{mod} the time to compute the model and store it if the algorithms allows, T_{op} is the time of operating the model and using it within the business logic of a real system.

In the present analysis we compare two architectures, Weka and TML with their respective times T^w and T^t. The Weka architecture consists of the Weka application connected directly to the content repository used by TML.

Document selection can be performed in Weka in three ways, reading the content from the file system, from a pre-built arff (Weka native format) file and from a database, the best performance is achieved using a relational database so we used this option in our experiments. For this task Weka uses a JDBC connection to a relational database. Most modern databases implement full text searching using inverted files indexing, the time for this task is $O(log(N))$ with N the total number of documents in the corpus. TML uses the Lucene index searching capabilities that also implements an inverted file index for searching, for this we expect that the time to obtain the contents of a Set to be the same in both architectures, this is shown in equation 4.

$$T_{sel}^{weka} \approx T_{sel}^{tml} \tag{4}$$

Document pre-processing and **Feature extraction** are implemented in Weka as a filter, which is applied once the content from the documents has been loaded on the previous step. As this two steps are the ones decoupled in TML, and run when documents are added or in parallel, this is where time savings are maximum, equation 5 shows the expected relation.

$$T_{pre}^{weka} + T_{fe}^{weka} > T_{pre}^{tml} + T_{fe}^{tml} \tag{5}$$

Creating the model and **Operating the model** have no differences between the architectures. Both use similar or identical implementations for any particular algorithm. But some techniques (e.g. KNN classification) don't need to train a model so it all happens at runtime. With others (e.g. Neural Network classification) a model is built and this task is highly time consuming. In both architectures this time is exactly the same, and at the date of submission both share the same method to store a model, serializing the data structure as a binary object. Since in both architectures the time requirements are the same we expect the relations shown in equations 6 and 7.

$$T_{mod}^{weka} = T_{mod}^{tml} \tag{6}$$

$$T_{op}^{weka} = T_{op}^{tml} \tag{7}$$

Finally, the total savings of time given the architecture can be calculated as the difference between T^{weka} and T^{tml} , simplified using all previous equations as shown in equation 8.

$$\Delta T = \Delta T_{sel} + \Delta T_{pre} + \Delta T_{fe} \tag{8}$$

5.2 Theoretical analysis of savings

In order to analyze if the savings will be relevant, the computational complexity of the pre-processing and feature extraction steps were calculated. The first step is to pre-process the documents, assuming N documents, each with an average length in characters of L and an average length in words of W, the complexity for processing each document is the sum of the complexity for tokenizing, stemming and removing stop-words. Given that tokenizing complexity is $O(L_d)$, stemming the words complexity is $O(W)$ (the size of the stemming dictionary is constant) and removing stop words complexity is a constant, the complexity for processing all the documents is shown in equation 9.

$$T_{pre}\ is\ O(N * W + N * L) \tag{9}$$

The time saved depends on the length of the documents, both in characters and in words. It is known that for every language there is a relation like $W * \alpha \approx L, \alpha \geq 1$ with α the average word length in characters for the language. For the experiments, the same Set of documents for each architecture will be compared, therefore the average length of the document is a constant. As a consequence the asymptotic behavior of the difference of time between architectures can be expected to b e linear against the number of documents as shown in equation 10.

$$T_{fe}\ is\ O(N) \tag{10}$$

Another question that raises is if the time saved is relevant within the whole time, for this we analyzed the remaining steps. The second step is building the dictionary, this is the process of identifying all the different terms that exist in the Set and building a common group, usually the terms has to be sorted within each document according to the same order the words appear in the dictionary. The complexity of checking every term and sorting them is $O(W * log(W))$ per document. At the end of this process the dictionary contains a reduced set of terms. Finally the weighting process affects only the resultant terms but they have to be updated for each document so its complexity is $O(D * W)$.

The relevance of the difference can be calculated then as the ratio R_Δ shown in equation 11.

$$R_\Delta = \frac{1+\alpha}{2+\alpha+log(W)} \tag{11}$$

Given that α and W are constant for the corpus, measured for the Uppsala collection we have $\alpha = 7.54$ and $W = 196$ so we could expect a rough approximation of the theoretical improvement of 49.78% in running time.

5.2.1 Performance

Overall time was measured for document selection, document preprocessing and feature extraction using two corpora: Reuters21578 [Hayes and Weinstein, 1990] and USE - Uppsala Student English [Axelsson and Berglund, 1999] corpora. Reuters21578 [Yang and Pedersen, 1997] is a corpus widely used by text mining researchers to explore performance of different classification algorithms, parameters, etc. For the experiment the training set included 7,769 documents with an average document length of 770 characters (short news stories). The Uppsala corpus contains 1,489 essays written in English by swedish students at the Uppsala University in Sweden. The documents have an average length of 4,746 characters.

We measured the computation time for each of the two corpus. The time it takes to create the VSM was calculated by repeating an experiment 10 times for each architecture. In each run the number of documents was processed in batches of 100, after which time was measured.

Figure 5 shows the graph with the results from the experiment with the Uppsala corpus, we can see clearly how both architectures show a linear behavior when the number of documents increase. This is what we expected from our theoretical analysis, we also can see that the overall time for TML is always lower than the time for Weka, averaging an improvement in time of 57%. The results for Reuters21578, where similar with TML being around 50% faster.

6 Conclusion

Text mining applications are becoming ubiquitous. Document summarization, clustering and visualization are some of the areas in which commercial developers and researchers are increasingly working. However, most of the required

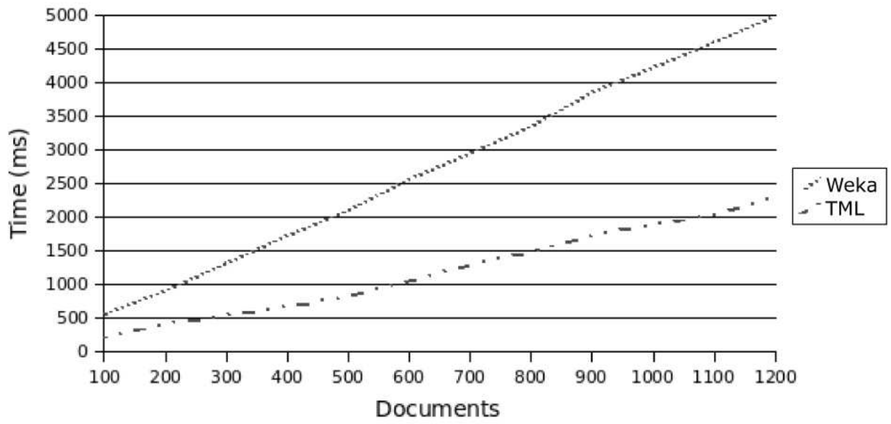

Figure 5: TML vs Weka on USE corpus

techniques can be computationally expensive posing scalability challenges for real-life applications.

This paper proposes a decoupled architecture for the TM process that allows scalability issues to be tackled as several of the computation costly sub-processes can be run in parallel. It also introduces TML a Text Mining library that implements the proposed architecture. A process description of whole TM process has been used to describe the scalability challenges and the solutions used for each sub-process.

An important aspect of the TML implementation is to bring together state of the art tools from different data processing and computational linguistics areas. TML integrates a search engine (i.e. Apache Lucene), using it for pre processing, handling the corpus and selecting documents. It also integrates linguistic features for operations that require them such as named entity recognition or typed dependencies.

A case study of applications for automatic feedback of writing was used to describe how the TML architecture can be used. Automatic feedback of academic writing is a challenging area that encompasses many of the text mining domains.

Acknowledgements

The authors would like to thank Stephen O'Rourke for his contribution to this project. This project has been funded by an Australian Research Discovery Grant DP0665064 and a Norman I Prize scholarship.

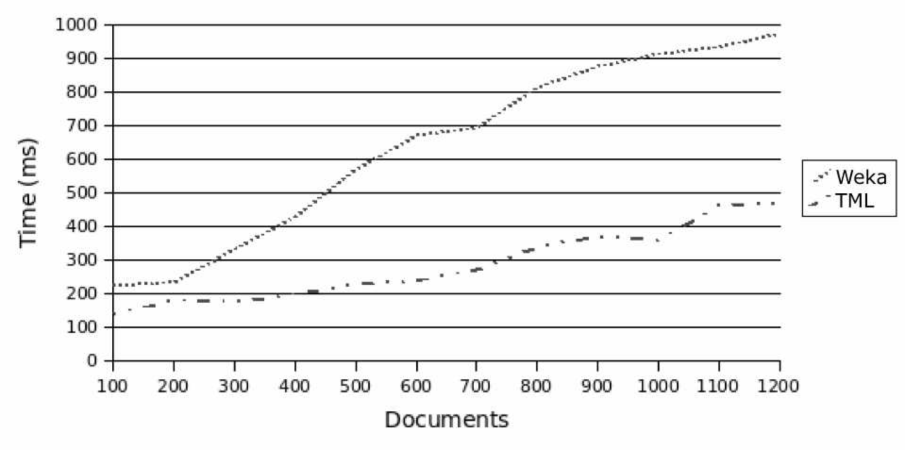

Figure 6: TML vs Weka on Reuters21578 corpus

References

[Alias-I, 2011] Alias-I (2011). Lingpipe 4. http://alias-i.com/lingpipe (accessed September, 2011).

[Axelsson and Berglund, 1999] Axelsson, M. W. and Berglund, Y. (1999). Use = uppsala student english corpus. Department of English, Uppsala University.

[Baldridge et al., 2002] Baldridge, J., Morton, T., and Bierner, G. (2002). The opennlp maximum entropy package. Technical report.

[Beals, 1998] Beals, T. (1998). Between teachers and computers: Does text-checking software really improve student writing? *English Journal*, 87(1):67–72.

[de Marneffe et al., 2006] de Marneffe, M.-C., MacCartney, B., and Manning, C. D. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of the fifth international conference on Language Resources and Evaluation*, pages 449–454, Genoa, Italy.

[Deerwester et al., 1990] Deerwester, S., Dumais, S., Furnas, G., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

[Dumais, 1990] Dumais, S. (1990). Enhancing performance in latent semantic indexing. Technical report, Bellcore.

[Fan et al., 2006] Fan, W., Wallace, L., Rich, S., and Zhang, Z. (2006). Tapping the power of text mining. *Communications of the ACM*, 49(9):76–82.

[Feinerer et al., 2008] Feinerer, I., Hornik, K., and Meyer, D. (2008). Text mining infrastructure in r. *Journal of Statistical Software*, 25:1–54.

[Feldman and Dagan, 1995] Feldman, R. and Dagan, I. (1995). Knowledge discovery in textual databases (kdt). In *Proceedings of the First international conference on Knowledge Discovery and Data Mining*, pages 112–117, Montreal, Canada.

[Flower, 1994] Flower, L. (1994). *The construction of negotiated meaning: a social cognitive theory of writing*. Southern Illinois University Press.

[Garcia Adeva, 2006] Garcia Adeva, J. J. (2006). Serving text-mining functionalities with the software architecture plato. In *Computational Intelligence for Modelling,* *Control and Automation, 2006 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*.

[Hayes and Weinstein, 1990] Hayes, P. and Weinstein, S. P. (1990). Construe/tis: a system for content-based indexing of a database of new stories. In *Proceedings of the Second Annual Conference on Innovative Applications of Artificial Intelligence*, Washington DC, USA.

[Hearst, 1999] Hearst, M. A. (1999). Untangling text data mining. In *Proceedings of ACL'99: the 37th Annual Meeting of the Association for Computational Linguistics*, Maryland, USA.

[Hearst, 2003] Hearst, M. A. (2003). What is text mining? Avaliable online on http://people.ischool.berkeley.edu/ hearst/text-mining.html.

[Holmes et al., 1994] Holmes, G., Donkin, A., and Witten, I. H. (1994). Weka: a machine learning workbench. Technical report, University of Waikato, Department of Computer Science.

[Holzman et al., 2004] Holzman, L. E., Fisher, T. A., Galitsky, L. M., Kontostathis, A., and Pottenger, W. M. (2004). A software infrastructure for research in textual data mining. *International Journal on Artificial Intelligence Tools*, 14:829–849.

[Hotho et al., 2005] Hotho, A., Nurnberger, A., and PaaB, G. (2005). A brief survey of text mining. *GLDV Journal for Computational Linguistics and Language Technology*, 20:19–62.

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

[Mierswa et al., 2006] Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., and Euler, T. (2006). Yale: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD International Conference on Know ledge Discovery and Data Mining*.

[Osinski and Weiss, 2005] Osinski, S. and Weiss, D. (2005). A concept-driven algorithm for clustering search results. *IEEE Intelligent Systems*, 20(3):48–54.

[Silic et al., 2007] Silic, A., Saric, F., Basic, B. D., and Snajder, J. (2007). Tmt: Object-oriented text classification library. In *Proceedings of the 29th International Conference on Information Technology Interfaces*, pages 559–564, Cavtat, Croatia.

[Villalon and Calvo, 2011] Villalon, J. and Calvo, R. A. (2011). Concept maps as cognitive visualizations of writing assignments. *International Journal of Educational Technology and Society*, 14(3):16–27.

[Wiemer-Hastings and Graesser, 2000] Wiemer-Hastings, P. and Graesser, A. C. (2000). Select-a-kibitzer: A computer tool that gives meaningful feedback on student compositions. *Interactive Learning Environments*, 8:149–169.

[Williams, 2003] Williams, K. (2003). A framework for text categorization. Master's thesis, School of Electrical & Information Engineering, University of Sydney.

[Yang and Pedersen, 1997] Yang, Y. and Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 412–420, Nashville, USA.