

# Use of Term Clusters for Emerging Trend Detection

April Kontostathis (akontostathis@ursinus.edu)

Lars E. Holzman (leh7@lehigh.edu)

William M. Pottenger (billp@cse.lehigh.edu)

**Abstract** — We describe a system that accurately detects emerging trends in a corpus of time-stamped textual documents. Our approach uses a Singular Value Decomposition, a dimensionality reduction approach that is used for other information retrieval and text mining applications, such as Latent Semantic Indexing (LSI). We create a term by document matrix, and apply our dimensionality reduction algorithm in combination with a similarity function to identify clusters of closely associated noun phrases. Use of our approach enables the detection of 92% of the emerging trends, on average, for the five collections we tested.

## 1 INTRODUCTION

An emerging trend is a topic area that has recently appeared in the literature and is growing in interest and utility over time. An Emerging Trend Detection (ETD) algorithm takes as input a large collection of textual data and identifies topic areas that are previously unseen or are growing in importance within the corpus.

Knowledge of emerging trends is particularly important to individuals and companies who are charged with monitoring a particular field or company. For example, a market analyst specializing in biotech companies may want to review technical and news-related literature for recent trends that will impact the companies being tracked. Manual review of all the available data is simply not feasible. Human experts who are tasked with identifying emerging trends need to rely on automated systems as the amount

of information available in digital format increases.

In this work we develop clusters of related terms and use the clusters for detecting emerging trends. Our approach combines the use of a dimensionality reduction algorithm with the cosine similarity function to define the similarity between terms in the collection. This similarity is then used to create clusters of closely associated terms for each term in the collection. After we have generated our term clusters, we use the cluster size as an attribute and apply decision tree induction to construct a model for detecting terms that are emerging within a collection.

In section 2 we review the literature and describe other systems that have been used for detecting emerging trends in textual data. In section 3 we outline our approach to emerging trend detection. In section 4 we describe our experimental methodology. Our experimental results are presented in section 5. Conclusions and future work are detailed in section 6.

## **2 RELATED WORK**

The authors have recently published a survey of current research involving Emerging Trend Detection (ETD) applications [2]. A summary of the survey appears in this section. The reader is referred to the full article for a more comprehensive report on the current state of ETD systems.

Current applications in ETD fall generally into two categories: fully-automatic and semi-automatic. The fully-automatic systems take in a corpus and develop a list of emergent topics. A human reviewer would then peruse these topics and the supporting evidence found by the system to determine which are truly emerging trends. These systems often include a visual component that allows the user to track the topic in an

intuitive manner [11][25]. Semi-automatic systems rely on user input as a first step in detecting an emerging trend [18][21]. These systems then provide the user with evidence to indicate whether the input topic is truly emerging, usually in the form of user-friendly reports and screens that summarize the evidence available on the topic.

Porter and Detampel describe a semi-automatic trend detection system for technology opportunities analysis in [18]. The first step of the process is the extraction of documents from the knowledge area to be studied from a database of abstracts, such as INSPEC®. The extraction process requires the development of a list of potential keywords by a domain expert. These keywords are then combined into queries using appropriate Boolean operators to generate searches. The target databases are also identified in this phase (ex. INSPEC®, COMPENDEX®, US Patents, etc.). The queries are then input to the Technology Opportunities Analysis Knowbot (TOAK), a custom software package also referred to as TOAS (Technology Opportunities Analysis System). TOAK extracts the relevant documents (abstracts) and provides bibliometric analysis of the data. Bibliometrics uses information such as word counts, date information, word co-occurrence information, citation information and publication information to track activity in a subject area. TOAK facilitates the analysis of the data available within the documents. For example, lists of frequently occurring keywords can be quickly generated, as can lists of author affiliations, countries, or states.

CIMEL is a multi-media framework for constructive and collaborative inquiry-based learning [5]. The semi-automatic trend detection methodology described in [21] has been integrated into the CIMEL system in order to enhance computer science education. A multimedia tutorial has been developed to guide students through the

process of emerging trend detection. Through the detection of emerging trends, the students see the role that current topics play in course-related research areas.

The TimeMines system [25] is a fully automated ETD system that takes text data, with explicit date tags, and develops an overview timeline of the most important topics covered by the corpus. TimeMines relies on Information Extraction (IE) and Natural Language Processing (NLP) techniques to gather the data. The system uses hypothesis testing techniques to determine the most relevant topics in a given timeframe. Only the ‘most significant and important information’ (as determined by the program) is presented to the user.

Similar to TimeMines, ThemeRiver™ [13] summarizes the main topics in a corpus and presents a summary of the importance of each topic via a user interface. The topical changes over time are shown as a river of information. The ‘river’ is made up of multiple streams. Each stream represents a topic, and is represented by a color. Each topic also maintains its place in the river relative to other topics.

The PatentMiner system was developed to discover trends in patent data using a dynamically generated SQL query, which is based upon selection criteria given by the user [16]. The system is connected to a database containing all granted US patents. There are two major components to the system, phrase identification using sequential pattern mining [1][24] and trend detection using a querying technique developed by the authors.

[19] describes an algorithm for detecting emerging trends in text collections based on semantically determined clusters of terms. The HDDI™ system is used to extract linguistic features from a repository of textual data and to generate clusters based on the

semantic similarity of these features. The algorithm takes a snapshot of the statistical state of a collection at multiple points in time. The rate of change in the size of the clusters and in the frequency and association of features is used as input to a neural network that classifies topics as emerging or non-emerging.

Novelty detection, also referred to as first story detection, is specifically included as a subtask in the Trend Detection and Tracking (TDT) initiative. Novelty detection requires identifying those news stories that discuss an event that has not already been reported in earlier stories, and operates without a predefined query. Typically algorithms look for keywords in a news story and compare the story with earlier stories. Unlike other emerging trend detection systems, the novelty detection task in the TDT initiative requires that the input be processed sequentially in date order. Only past stories can be used for evaluation, not the entire corpus. Thus, the term ‘novelty detection’ cannot be used interchangeably with the term ‘Emerging Trend Detection.’

### **3 USING DIMENSIONALITY REDUCTION TO DEVELOP TERM CLUSTERS**

In this section we describe our term clustering algorithm in detail. Our approach relies on dimensionality reduction benefits similar to those used in Latent Semantic Indexing (LSI). LSI has been shown to improve performance for some collections in a variety of applications such as search and retrieval [10][7], filtering [8][9], and word sense disambiguation [23]. LSI derives its power from an algebraic technique known as singular value decomposition (SVD). In the next section we describe the LSI system, with particular emphasis on the use of the SVD. In section 3.2 we describe an approach to determining term similarity based on the SVD. In section 3.3 we present a sparsification algorithm that reduces the memory requirements of LSI, while maintaining

the same level of retrieval performance. In section 3.4 we present our term clustering algorithm, which combines the similarity between terms and the sparsification of the LSI matrices to identify clusters of closely related terms. In sections 4 and 5 we show that our term clustering algorithm can be used to successfully identify emerging trends in a variety of collections.

### **3.1 Latent Semantic Indexing (LSI)**

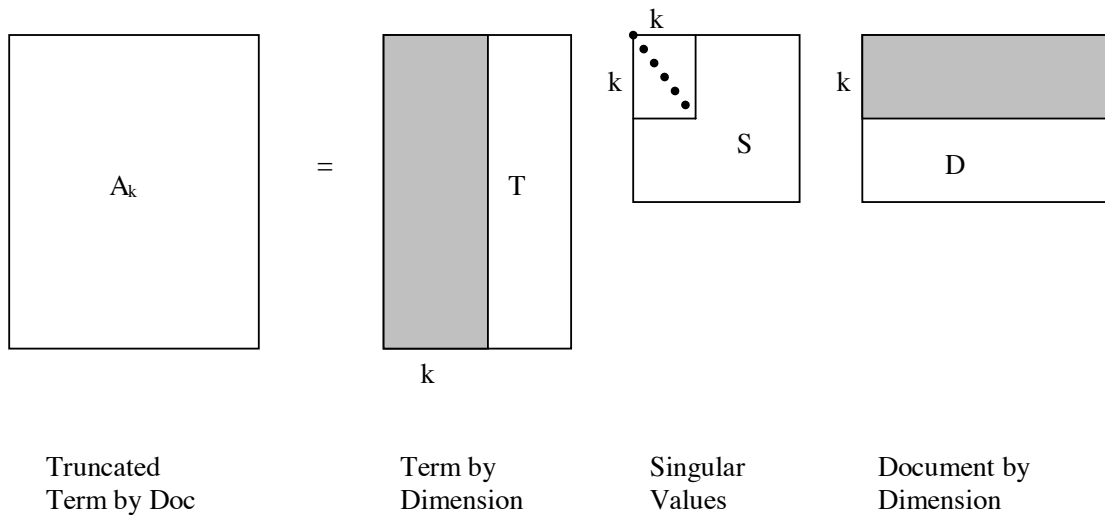
In information retrieval applications, we refer to two primary entities, terms and documents. An index term (or term) is a pre-selected word (or group of words) that can be used to refer to the content of a document. Usually, index terms are nouns or noun groups [6]. In this section we describe a commonly used approach for information retrieval, LSI. In the next section, we will apply a modified approach akin to LSI in the development of term clusters.

In traditional vector space retrieval, documents and queries are represented as vectors in  $t$ -dimensional space, where  $t$  is the number of indexed terms in the collection. Generally the document vectors are formed when the index for the collection is generated (these vectors form a matrix that is often referred to as the term by document matrix), and the query vectors are formed when a search of the collection is performed.

The SVD process employed by LSI decomposes the term by document matrix into three matrices:  $T$ , a term by dimension matrix,  $S$  a singular value matrix (dimension by dimension), and  $D$ , a document by dimension matrix. The number of dimensions is  $r$ , the rank of the term by document matrix. The original matrix can be obtained, through matrix multiplication of  $TSD^T$ .

In an LSI system, the  $T$ ,  $S$  and  $D$  matrices are truncated to  $k$  dimensions. This

process is presented graphically in figure 1 (taken from [3]). The shaded areas in the T and D matrices are kept, as are the  $k$  largest singular values, the non-shaded areas are removed. The purpose of dimensionality reduction is to reduce “noise” in the latent space, resulting in a richer word relationship structure that reveals latent semantics present in the collection. [2] discusses LSI processing in detail and provides a numerical example. LSI has been shown to improve retrieval performance for a variety of collections [10][7].



**Figure 1: Truncation of SVD for LSI [3]**

### 3.2 LSI and Term Similarity

After we compute the SVD using the original term by document matrix, we can calculate term-term similarities. LSI provides two natural methods for describing term-term similarity. First, the term-term matrix can be created using  $T_k S_k (T_k S_k)^T$ . Second, the term by dimension ( $T_k S_k$ ) matrix can be used to compare terms using a vector similarity measure, such as cosine similarity. In this case, the cosine similarity is computed for each pair of rows in the  $T_k S_k$  matrix. The cosine similarity computation results in a value

in the range  $[-1, 1]$  for each pair of terms.

We have studied the correlations between the distribution of the term similarity values and LSI performance, and our results clearly preferred the cosine similarity metric over the term-term similarity metric [15]. From an intuitive perspective, the geometric interpretation of the cosine similarity is consistent with what other researchers have proposed for understanding the LSI term by dimension and document by dimension matrices [22].

### **3.3 Sparsification of LSI Matrices**

We also conducted a study to determine the most critical elements of the  $T_k$  and  $S_k D_k$  matrices, which are input to the query matching step in LSI [14]. We were interested in the impact, both in terms of retrieval quality and query run time performance, of removal of a large portion of the entries in these matrices.

Our sparsification algorithm focuses on the values with absolute value near zero, and we ask the question: “How many values can we remove without severely impacting retrieval performance?” Intuitively, the elements of the row vectors in the  $T_k S_k$  matrix and the column vectors in the  $S_k D_k$  matrix can be used to describe the importance of each term (document) along a given dimension. This approach stems from the premise that each dimension in the SVD represents a given concept or group of concepts [22]. Positive values indicate a similarity between the term (document) and the concept; negative values indicate dissimilarity. The algorithm we used is outlined in figure 2. We chose positive and negative threshold values that are based on the  $T_k S_k$  matrix and that result in the removal of a fixed percentage of the  $T_k$  matrix. We use these values to truncate the both the  $T_k$  and  $S_k D_k$  matrices.

In [14] we show that a significant memory savings can be achieved using our sparsification strategy to remove 70% of the values in the  $T_k$  matrix. Furthermore, this savings can be achieved without any degradation in retrieval effectiveness of the LSI system.

<p>Compute <math>T_k S_k</math> and <math>S_k D_k</math> Determine PosThres: The threshold that would result in removal of <math>x\%</math> of the positive elements of <math>T_k S_k</math> Determine NegThres: The threshold that would result in removal of <math>x\%</math> of the negative elements of <math>T_k S_k</math> For each element of <math>T_k</math>, change to zero, if the corresponding element of <math>T_k S_k</math> Falls between PosThres and NegThres For each element of <math>S_k D_k</math>, change to zero, if it falls between PosThres and NegThres</p>
---

**Figure 2: Sparsification Algorithm**

### 3.4 Term Clustering Algorithm

As noted in sections 3.2 and 3.3, we observed strong correlations between distribution of the cosine similarity values (that were computed using the LSI term by dimension matrix), and the performance of LSI [15]. We also noted that a large portion of the entries in the term by dimension and document by dimension matrices can be removed without impacting the retrieval quality of an LSI system [14].

In this section we leverage these theoretical results when developing our term clustering algorithm. In what follows, we describe our algorithm in detail. Our approach uses dimensionality reduction and sparsification to define the similarity between terms in the collection and then uses this similarity to create clusters of closely associated terms for each term in the collection.

Our term clustering algorithm is outlined in figure 3. A cluster is created for each

term,  $i$ , in the collection. In some cases, the cluster will have only one element, the term itself. Our hypothesis is that the dimensionality reduction process will identify terms that are closely related. The term clustering algorithm relies on two parameters,  $k$ , the truncation parameter used in LSI, and  $\_Threshold$ , which determines the size of the clusters. Both parameters are set globally and applied uniformly to all clusters. In section 4 we describe how these clusters, once formed, are used to detect emerging trends in a text collection.

```

Create the term by document matrix for the collection
Compute the SVD for the term by document matrix
Truncate the term by dimension,  $T$ , matrix to  $k$  dimensions
For each term  $i$ 
  For each term  $j$ 
    Compute,  $SIM$ , the similarity between term  $i$  and term  $j$ 
       $SIM = \text{Cosine Distance between row } i \text{ and row } j \text{ in the } T_k \text{ matrix}$ 
    If  $SIM > \_Threshold$ 
      Add term  $j$  to the cluster for term  $i$ 

```

**Figure 3: Term Clustering Algorithm**

## 4 EXPERIMENTAL METHODOLOGY

An Emerging Trend Detection (ETD) application takes as input a large collection of time-stamped textual data and identifies topic areas that are or are growing in importance within the corpus. In this section we describe a fully automated system for detecting emerging trends in a textual collection. Our approach combines the term clusters that we developed in the previous section, with the WEKA data mining tool [27]. In section 5, we show that we can achieve  $F_\beta$  (defined in section 5) levels from .81-.89, when we use our clusters to detect emerging trends. In the following subsections, we provide an

overview of the processing that was required to produce our list of emerging and non-emerging trends. We first discuss the input data that was used and then describe the preprocessing steps required for the data. In section 4.3 we discuss the learning algorithms we employed.

## 4.1 Input data

Our algorithm for detecting emerging trends in text collections is based on semantically determined clusters of terms. The TMI [12] system, described in Appendix A, is used to extract terms from a repository of textual data. We then use the term clustering algorithm defined in figure 3 to generate clusters based on the semantic similarity of these terms. Our terms are maximal noun phrases that follow the regular expression that appears in equation 1, where C is a cardinal number, G is verb (gerund or present participle), P is a verb (past participle), J is an adjective, N is a noun, I is a preposition, D is a determiner, ? indicates zero or one occurrence, | indicates union, \* indicates zero or more occurrences, and + indicates one or more occurrence. This type of term was chosen because our collections contain terms of this type that have been previously identified as emerging or non-emerging trends. The list of previously classified instances is known as a truth set within the TMI framework.

$$C?(G|P|J)*N+(I*D?C?(G|P|J)*N+)* \quad (1)$$

<b>Data Set</b>	<b>Total Instances</b>	<b>Emerging</b>	<b>Non-Emerging</b>
INSPEC® 1996	74	39	35
INSPEC® 1997	77	37	40
INSPEC® 1998	129	67	62
INSPEC® 1999	113	56	57
OOSE 2000	223	50	183

**Table 1: Number of emerging/non-emerging concepts**

We used five collections to test our ETD application. These collections are summarized in table 1. The Inspec® collections were previously used in work by Zhou [28]. Each collection we studied has an associated truth set that lists emerging and non-emerging trends within that collection. We collected seven attributes for each noun phrase in each collection. The list of attributes appears in table 2 and sample data from the Inspec® 1997 collection appears in table 3. These attributes were chosen because they describe the frequency of occurrence of various terms within the collection over time (first four attributes), as well as the relationship between the candidate terms and other terms within the collection (cluster attributes). The final attribute serves as a measure of complexity of the term itself.

Number of times the term appeared in the current year
Number of times the term appeared in the year before the current year
Number of times the term appeared in the year which was two years prior to the current year
Total number of times the term appeared in all years prior to the current year
Total terms in the cluster containing the given term in the current year
Total terms in the cluster containing the given term in the year before the current year
Total number of words of length $\geq 4$ in the term

**Table 2: Attributes used in ETD studies**

Term	Occur.			Total	Cluster	Cluster	Long	Classification
	1997	1996	1995	Before	size in	Size in	words in	
association rule	0	1	0	1	0	11	2	trend
strong association rules	1	0	0	0	15	0	3	trend
classification rule	0	0	0	2	0	0	2	notrend
data warehouse	36	7	1	8	0	0	2	trend
binary decision tree	1	1	1	2	22	20	3	notrend
dynamic programming	1	2	3	10	0	0	2	notrend
sql queries	6	0	3	3	0	0	1	notrend
sql syntax	1	0	0	0	18	0	1	trend

**Table 3: Sample Data from Inspec® 97**

Once the attributes were computed for each noun phrase in our truth set, the weka.classifier.j48.J48 algorithm [27] was used to generate a decision tree. We used stratified tenfold cross-validation in our testing. For each cross-validation evaluation, we randomly selected ten datasets from our truth set with resampling<sup>1</sup>.

In our initial tests, trend and non-trend instances were selected separately and the original ratio between trend and non-trend instances was maintained (for example, a ratio of 39 trend to 30 non-trend for the Inspec® 96 data). However, we also added a replication factor, which allowed us to place a greater emphasis on the trend instances. For example, a replication factor of two would ensure that at least twice as many trend instances were used in the training sets (resulting in a ratio of 78 to 30 for the Inspec® 96 data). This process is known as weighted sampling. In our implementation of tenfold cross-validation, weighted sampling is not applied to the holdout (test) fold in each run. Our stratified tenfold cross-validation procedure was implemented using TMI [12]. In the next section we discuss the generation of the clusters for the two attributes that require cluster sizes. In section 4.3 we discuss the learning algorithm we employed. Our experimental results appear in section 5.

<sup>1</sup> Resampling means that an instance that has been selected from the original set is returned to the set and may be reselected.

## 4.2 Developing the clusters

The clustering algorithm appears in figure 3, and we developed clusters for multiple values of  $k$  and multiple sparsification levels (see section 3.3 for a description of our sparsification process). A term by document matrix was created for both the current year and the previous year, for each collection. We decomposed these matrices using PGTP [17]. Clusters were developed for values of  $k$  from 25 to 150, in increments of 25, and for sparsification levels from 0% to 90%. We used a *\_Threshold* = 1.0. Training and test sets were developed based on each set of clusters. Each set consisted of the current year and previous year clusters at a given value of  $k$  and sparsification level. These training and test sets were then used as input to the learning algorithm.

## 4.3 Learning Algorithm

In the previous sections we discussed the creation of the training data to be used in the next step of ETD model building, supervised learning. In this section we present the supervised learning algorithm used to develop the emerging trend detection model. The WEKA machine-learning library [27] was used to construct our emerging trend detection model. WEKA is an open-source collection of machine learning algorithms that can either be applied directly to a dataset or called via an Application Programming Interface (API); we employed both methods of access during our experiments.

We used the `weka.classifier.j48.J48` algorithm, which generates either a pruned or an unpruned C4.5 [20] decision tree. We used the pruned decision tree in our experiments. This classifier was chosen because the C4.5 algorithm is widely used for decision tree creation, and some previous work on ETD also employed entropy-based decision tree algorithms [28], making it easier to compare our results. In our first set of

experiments we attempted to discover the term clustering parameters that provided an optimal solution to learning trends within each collection. We used the cross-validation training and test sets described in the previous section to identify the optimal model for each collection.

Our second set of experiments focused on identifying an emerging trend detection model that would be effective across collections. For these experiments we chose the training set that was used to develop the optimal model for each individual collection and selected the test set that used the same parameters for term clustering from the other collections. The results of these experiments are discussed in the next section.

## 5 EXPERIMENTAL RESULTS

The results of the emerging trend detection studies appear in this section. Performance of an information retrieval or text mining system can be expressed in a variety of ways. In the current work, we primarily use precision and recall to measure the performance. Precision (P) and recall (R) are defined in equations (2) and (3). We also used  $F_\beta$ , a combination of precision and recall [26]. The formula for  $F_\beta$  is shown in equation 4. The  $\beta$  parameter is the ratio R/P, and allows us to place greater or lesser relative emphasis on recall vs. precision, depending on our needs. In our experiments we used  $\beta=1$ , which balances precision and recall.

$$P = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \quad (2)$$

$$R = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \quad (3)$$

$$F_\beta = \frac{(\beta^2 + 1)PR}{(\beta^2 P + R)} \quad (4)$$

Collection	k	Sparsification	Replication Factor	Recall	Precision	$F_{\beta}$
Inspec® 96	100	60%	2	.89	.87	.88
Inspec® 97	75	60%	3	.93	.82	.87
Inspec® 98	75	70%	2	.93	.71	.81
Inspec® 99	50	10%	2	.92	.79	.85
OOSE	150	40%	1	.93	.86	.89
<b>Average</b>				<b>.92</b>	<b>.81</b>	<b>.86</b>

**Table 4: Optimal intra-collection settings and results**

Collection	Recall	Precision	$F_{\beta}$
Inspec® 96	.79	.90	.84
Inspec® 97	.77	.83	.80
Inspec® 99	.84	.82	.83
OOSE	.90	.81	.85
<b>Average</b>	<b>.83</b>	<b>.84</b>	<b>.83</b>

**Table 5: sLoc Partitions Results**

## 5.1 Intra-collection experiments

In this section we discuss our intra-collection experiments. These experiments employed tenfold cross-validation on the training data from each collection. For each collection we also used a variety of replication factors to develop our model. The replication factor was incremented until the performance (as measured by  $F_{\beta}$ ) remained constant for three consecutive iterations, or until  $F_{\beta}$  decreased from the previous level. The configuration with the maximum  $F_{\beta}$  was chosen as the optimal for our intra-collection studies. As shown in table 4, our  $F_{\beta}$  performance results range from .81 - .89.

For comparison purposes, the four data sets were used with another term clustering algorithm [4] and the results of these experiments appear in table 5.  $F_{\beta}$  for our LSI-based term clustering experiments outperformed the sLoc clusters for all collections.

The optimal parameters for all collections appear in table 4, and the associated decision trees appear in figures 4 - 8. The decision trees show some areas of consistency across collections. All except OOSE involve the use of the cluster sizes and, in general,

smaller cluster sizes are used to predict the existence of emerging trends. In the next section we use the optimal configuration for each collection to predict the existence of emerging trends in other collections.

```

Occurrences_in_Year_Before_Previous_Year <= 0
| Concepts_in_Cluster_Previous_Year <= 0
| | Occurrences_in_All_Noncurrent_Years <= 2: trend (61.0/1.0)
| | Occurrences_in_All_Noncurrent_Years > 2
| | | Occurrences_in_All_Noncurrent_Years <= 6: trend (7.0/1.0)
| | | Occurrences_in_All_Noncurrent_Years > 6: notrend (2.0)
| | Concepts_in_Cluster_Previous_Year > 0: notrend (5.0)
Occurrences_in_Year_Before_Previous_Year > 0
| Long_Words_In_Feature <= 3: notrend (22.0/2.0)
| Long_Words_In_Feature > 3
| | Occurrences_in_Current_Year <= 1: notrend (2.0)
| | Occurrences_in_Current_Year > 1: trend (2.0)

Number of Leaves :    7

Size of the tree :    13

```

**Figure 4: Optimal decision tree for Inspec® 96**

```

Occurrences_in_Year_Before_Previous_Year <= 1
| Occurrences_in_All_Noncurrent_Years <= 0: trend (81.0/6.0)
| Occurrences_in_All_Noncurrent_Years > 0
| | Concepts_in_Cluster <= 0
| | | Occurrences_in_Year_Before_Previous_Year <= 0
| | | | Occurrences_in_All_Noncurrent_Years <= 1: trend (10.0/1.0)
| | | | Occurrences_in_All_Noncurrent_Years > 1: notrend (2.0)
| | | Occurrences_in_Year_Before_Previous_Year > 0: trend (3.0)
| | Concepts_in_Cluster > 0: notrend (13.0)
Occurrences_in_Year_Before_Previous_Year > 1: notrend (13.0)

Number of Leaves :    6

Size of the tree :    11

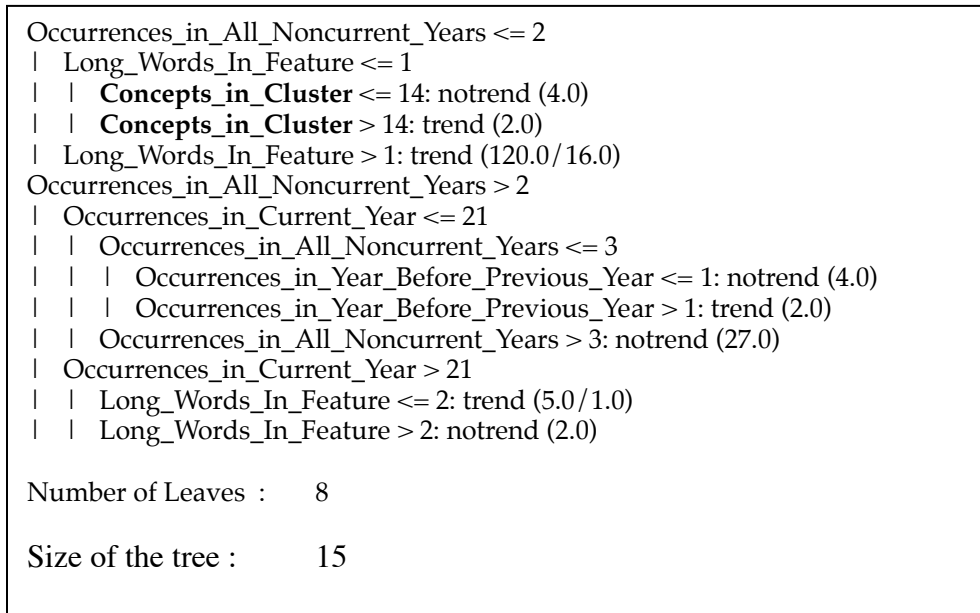
```

**Figure 5: Optimal decision tree for Inspec® 97**

## 5.2 Inter-collection experiments

In this section we describe the results when a training set from one collection is used to build a model, and the model is then evaluated using a test set from another collection.

The training set used the optimal replication factor shown in table 4; however, as previously noted for tenfold cross-validation, the trend instances were not replicated in the test data. Furthermore, the clusters used in the both the training and test data sets were developed using the same parameters. For example, when using the training set from Inspec® 96, with clusters created based on  $k=100$  and sparsification of 60%, the test sets from Inspec® 97, Inspec® 98, Inspec® 99, and OOSE use the same clustering parameters.



**Figure 6: Optimal decision tree for Inspec® 98**

The optimal decision tree for Inspec® 96 used replication factor 2,  $k=100$  and sparsification at 60%. Table 6 shows the retrieval quality when this decision tree was used to detect emerging trends in the other four collections.  $F_{\beta}$  for the four test collections degrades from the intra-collection results, dropping from an average of .86 to an average of .72.

```

Occurrences_in_Previous_Year <= 0: trend (82.0/8.0)
Occurrences_in_Previous_Year > 0
| Concepts_in_Cluster_Previous_Year <= 19
| | Concepts_in_Cluster <= 6: notrend (42.0/4.0)
| | Concepts_in_Cluster > 6
| | | Concepts_in_Cluster_Previous_Year <= 0: notrend (2.0)
| | | Concepts_in_Cluster_Previous_Year > 0: trend (7.0/1.0)
| | Concepts_in_Cluster_Previous_Year > 19: trend (4.0)

Number of Leaves :    5

Size of the tree :    9

```

**Figure 7: Optimal decision tree for Inspec® 99**

```

Occurrences_in_All_Noncurrent_Years <= 7
| Occurrences_in_All_Noncurrent_Years <= 4: trend (36.0/6.0)
| Occurrences_in_All_Noncurrent_Years > 4
| | Occurrences_in_Previous_Year <= 4
| | | Occurrences_in_Year_Before_Previous_Year <= 1: notrend (9.0)
| | | Occurrences_in_Year_Before_Previous_Year > 1
| | | | Long_Words_In_Feature <= 2: trend (4.0)
| | | | Long_Words_In_Feature > 2: notrend (2.0)
| | Occurrences_in_Previous_Year > 4: trend (6.0)
Occurrences_in_All_Noncurrent_Years > 7: notrend (122.0/1.0)

Number of Leaves :    6

Size of the tree :    11

```

**Figure 8: Optimal decision tree for OOSE**

<b>Collection</b>	<b>Recall</b>	<b>Precision</b>	<b>F-Measure</b>
Inspec® 97	.93	.63	.75
Inspec® 98	.91	.67	.77
Inspec® 99	.84	.73	.78
OOSE	.46	.76	.57
<b>Average</b>	<b>.79</b>	<b>.70</b>	<b>.72</b>

**Table 6: Cross Collection Results using Inspec® 96 Training Set**

The optimal decision tree for Inspec® 97 used replication factor 3,  $k=75$  and sparsification at 60%. Table 7 shows the precision, recall and  $F_\beta$  results when this

decision tree was used to detect emerging trends in the other four collections. In this case,  $F_{\beta}$  is much lower than the intra-collection levels.

<b>Collection</b>	<b>Recall</b>	<b>Precision</b>	<b>F-Measure</b>
Inspec® 96	.74	.77	.75
Inspec® 98	.82	.70	.76
Inspec® 99	.73	.70	.71
OOSE	.29	.44	.35
<b>Average</b>	<b>.65</b>	<b>.65</b>	<b>.64</b>

**Table 7: Cross Collection Results using Inspec® 97 Training Set**

The optimal decision tree for Inspec® 98 used replication factor 2,  $k=75$  and sparsification at 70%. Table 8 shows the results when this decision tree was used to detect emerging trends in the other four collections. In this case, recall for the Inspec® 96 collection matches the intra-collection results, and Inspec® 97 jumps to 100%. Recall for Inspec® 99 degrades slightly from 92% to 84% and OOSE is significantly lower, falling from 93% to 51%. However, precision and  $F_{\beta}$  are significantly lower when this technique is used. Precision drops to an average of 64% and average  $F_{\beta}$  drops to .70.

<b>Collection</b>	<b>Recall</b>	<b>Precision</b>	<b>F-Measure</b>
Inspec® 96	.89	.70	.78
Inspec® 97	1.00	.58	.73
Inspec® 99	.84	.64	.73
OOSE	.51	.62	.56
<b>Average</b>	<b>.81</b>	<b>.64</b>	<b>.70</b>

**Table 8: Cross Collection Results using Inspec® 98 Training Set**

The optimal decision tree for Inspec® 99 used replication factor 2,  $k=50$  and sparsification at 10%. Table 9 shows the retrieval quality when this decision tree was used to detect emerging trends in the other four collections. In this case, recall for Inspec® 96 and 97 remains constant when compared to the intra-collection results, and Inspec® 98 increases from 93% to 95%. OOSE performance degrades from 93% to 54%, but this training set performs better than the other three sets for the OOSE data.

Precision and  $F_\beta$  show significant decreases over the intra-collection results; however, the average recall shows a smaller decrease to 83%.

<b>Collection</b>	<b>Recall</b>	<b>Precision</b>	<b>F-Measure</b>
Inspec® 96	.89	.72	.80
Inspec® 97	.93	.61	.74
Inspec® 98	.95	.65	.77
OOSE	.54	.42	.47
<b>Average</b>	<b>.83</b>	<b>.60</b>	<b>.70</b>

**Table 9: Cross Collection Results using Inspec® 99 Training Set**

The optimal decision tree for OOSE used replication factor 1,  $k=150$  and sparsification at 40%. Table 10 shows the precision and recall results when this decision tree was used to detect emerging trends in the other four collections. Recall for all collections is the same or higher with this training set, when compared to the intra-collection results. In fact, average recall increases to 94% (from 92%); however, precision degrades from 81% to 62% and  $F_\beta$  drops from an average of .86 to an average of .74.

<b>Collection</b>	<b>Recall</b>	<b>Precision</b>	<b>F-Measure</b>
Inspec® 96	.89	.65	.75
Inspec® 97	.97	.55	.70
Inspec® 98	.96	.64	.77
Inspec® 99	.93	.62	.74
<b>Average</b>	<b>.94</b>	<b>.62</b>	<b>.74</b>

**Table 10: Cross Collection Results using OOSE Training Set**

## 6 CONCLUSIONS AND FUTURE WORK

### 6.1 Discussion

In this paper we have shown that our term clustering approach can be used to detect emerging trends in a text collection. We achieved  $F_\beta$  levels from .81 to .89 on intra-collection studies, and from .35 to .80 in our inter-collection studies. These results are

much higher than the emerging trend detection application approach used by Pottenger and Yang [19], which achieved a maximum  $F_{\beta}$  of .35, and Zhou [28], which obtained  $F_{\beta}$  values in the range .09 to .29.

Interestingly, all of the decision trees produced by the optimal parameters were fairly small. The trees ranged in size from nine to fifteen nodes and had five to eight leaves. One common concern for machine learning algorithms is over-fitting<sup>2</sup>. In this case the performance of the models across collections suggests that although some over-fitting may be occurring, it is not serious. As a result, this approach may be applied to a variety of collections.

## **6.2 Future Work**

The infrastructure that we used for implementation of the ETD application provides a solid foundation for additional research in the areas of textual data mining and information retrieval. We plan to use the flexibility within this infrastructure to apply our term clustering approach to other areas of machine learning research, particularly to problems in traditional data mining, such as market basket analysis.

## **APPENDIX A – TEXT MINING INFRASTRUCTURE, COPYRIGHT 2003**

The TMI (Text Mining Infrastructure) is a system designed by the Lehigh University Parallel and Distributed Textual Data Mining lab. It is described in detail in the article [12]. Documentation and updates can be found on the WWW at: <http://hddi.cse.lehigh.edu/>. TMI is a solution to the problems posed by textual data mining (TDM) research. TDM faces many unique problems stemming from the quantity

---

<sup>2</sup> Over-fitting occurs when the solution is overly optimized to the training data (as opposed to test or other previously unseen data).

of the data present for processing and the free-form nature of this data. TDM experiments require a way of organizing this data, a way to process the organized data, and a way to discover useful patterns in the data.

To address these three areas TMI has been designed using three layers. The first layer represents the concept of a repository (or a corpus). A repository is a collection of documents (referred to as items) from a particular collection or with a common theme (i.e., abstracts from a particular journal over a range of time). The second layer stores relations between this data and performs pre-preprocessing steps. This includes the capacity to generate features (which are smaller cohesive units of text) such as noun phrases, collocations, or n-grams. This layer also stores the relations between the features and the items in the repositories. Finally, there is the machine learning layer which is responsible for discovering patterns in the data. This machine learning layer integrates all of the primary aspects of machine learning including: filtering of the data, the learning algorithms, drivers to coordinate the learning, and evaluation. An optimization framework has been added to support automated exploration of the parameter space.

TMI has been written in C++ to provide both efficiency and flexibility. To support the large research audience that exists in the field of text mining, TMI runs on Linux environments (using g++ 3.0 or higher) as well as Windows environments (using Visual Studio .NET 2003).

TMI utilizes a component framework. It is straightforward to develop a data-flow system using the provided components, and the component architecture lends itself to a rapid-prototyping approach to experiment design. The components have inputs and

outputs that support run-time checking during execution to ensure application correctness.

## ACKNOWLEDGMENT

This research was supported in part by NSF CISE EIA grant number 0087977. The authors wish to thank the faculty and support staff at Lehigh University and Ursinus College. The authors would also like to thank the team that built the TMI infrastructure. Coauthor William M. Pottenger wishes to express his sincere gratitude to his Lord and Savior, Yeshua (Jesus) the Messiah (Christ) for his salvation.

## REFERENCES

- [1] Agrawal R., and R. Srikant. 1995. Mining Sequential Patterns. *Proceedings of the International Conference on Data Engineering (ICDE)*.
- [2] Berry, M. (Ed.). 2003. *Survey of Text Mining: Clustering, Classification, and Retrieval*. Springer-Verlag.
- [3] Berry, M. W., S. T. Dumais, and G. W. O'Brien. 1995. Using linear algebra for intelligent information retrieval. *SIAM Review*, Volume 37, No. 4. pp. 573-595.
- [4] Bouskila F., W. M. Pottenger. 2000. The Role of Semantic Locality in Hierarchical Distributed Dynamic Indexing. *In Proceedings of the 2000 International Conference on Artificial Intelligence (IC-AI 2000)*.
- [5] Blank, G. D., W. M. Pottenger, G. D. Kessler, M. Herr, H. Jaffe, S. Roy, D. Gevry, Q. Wang. 2001. CIMEL: Constructive, collaborative Inquiry-based Multimedia E-Learning. *The 6th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*.
- [6] Baeza-Yates, R. and R. N. Ribeiro-Neto. 1999. *Modern Information Retrieval*. Addison Wesley.
- [7] Dumais, S. T. 1993. LSI meets TREC: A status report. *The First Text REtrieval Conference (TREC1)*, D. Harman (Ed.), *National Institute of Standards and Technology Special Publication 500-207*, pp. 137-152.
- [8] Dumais, S. T. 1994. Latent Semantic Indexing (LSI) and TREC-2. In: D. Harman (Ed.), *The Second Text REtrieval Conference (TREC2)*, *National Institute of Standards and Technology Special Publication 500-215*, pp. 105-116
- [9] Dumais, S. T. 1995. Using LSI for information filtering: TREC-3 experiments. In: D. Harman (Ed.), *The Third Text REtrieval Conference (TREC3)* *National Institute of Standards and Technology Special Publication*.
- [10] Deerwester, S., S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman.

1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*. Vol. 41, No. 6, pp. 391-407.
- [11] Davidson, G. S., B. Hendrickson, D. K. Johnson, C. E. Meyers, and B. N. Wylie. 1998. Knowledge mining with VxInsight™: Discovery through interaction. *Journal of Intelligent Information Systems.*, Volume 11 No.3. pp. 259-285.
- [12] Holzman, L., T.A. Fisher, L.M. Galitsky, A. Kontostathis, and W. M. Pottenger. 2003. A Software Infrastructure for Research in Textual Data Mining. *Proceedings of the fifteenth annual IEEE International Conference on Tools with Artificial Intelligence.*
- [13] Havre S., E. Hetzler, P. Whitney, and L. Nowell. 2002. ThemeRiver: Visualizing Thematic Changes in Large Document Collections. *IEEE Transactions on Visualization and Computer Graphics*. Vol. 8, No. 1.
- [14] Kontostathis, April. 2003. A Term Co-occurrence Based Framework for Understanding LSI: Theory and Practice. Dissertation. Lehigh University.
- [15] Kontostathis, April and William M. Pottenger. 2003. A framework for understanding LSI Performance. *Proceedings of ACM SIGIR Workshop on Mathematical/Formal Methods in Information Retrieval-ACMSIGIR MF/IR '03.*
- [16] Lent, B., R. Agrawal, and R. Srikant. 1997. Discovering Trends in Text Databases. *Proceedings of the Third International Conference On Knowledge Discovery and Data Mining.*
- [17] Martin, D. I. and M. W. Berry. 2001. Parallel General Text Parser. Copyright 2001.
- [18] Porter, A. L., and M. J. Detampel. 1995. Technology Opportunities Analysis. *Technological Forecasting and Social Change*. Volume 49. pp. 237-255.
- [19] Pottenger, W. M. and T. Yang. 2001. Detecting Emerging Concepts in Textual Data Mining. *Computational Information Retrieval*, Michael Berry, Ed.
- [20] Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufman.
- [21] Roy S., D. Gevry, W. M. Pottenger. 2002. Methodologies for Trend Detection in Textual Data Mining. *Proceedings of the Textmine '02 Workshop, Second SIAM International Conference on Data Mining.*
- [22] Schütze, H. 1992. Dimensions of Meaning. *Proceedings of Supercomputing '92.*
- [23] Schütze, H. 1998. Automatic Word Sense Disambiguation. *Computational Linguistics*, Volume 24, No. 1.
- [24] Srikant, R., and R. Agrawal. 1996. Mining sequential patterns: Generalizations and performance improvements. *Proceedings of the Fifth International Conference on Extending Database Technology (EDBT).*
- [25] Swan, R and D. Jensen. 2000. TimeMines: Constructing Timelines with Statistical Models of Word Usage. *Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*
- [26] van Rijsbergen, C.J. 1979. *Information Retrieval*. Butterworths, London.
- [27] Witten, I. H. and E. Frank. 2000. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann. San Francisco, CA.
- [28] Zhou, L. 2001. *Machine Learning Classification for Detecting Trends in Textual Collections*. Master's Thesis. University of Illinois at Urbana-Champaign.