Data Mining in e-Learning

Khaled Hammouda[1] and Mohamed Kamel[2]

[1] Department of Systems Design Engineering
[2] Department of Electrical and Computer Engineering
Pattern Analysis and Machine Intelligence (PAMI) Research Group
University of Waterloo
Waterloo ON N2L3G1, Canada
{hammouda,mkamel}@pami.uwaterloo.ca

This chapter presents an innovative approach for performing data mining on documents, which serves as a basis for knowledge extraction in e-learning environments. The approach is based on a radical model of text data that considers phrasal features paramount in documents, and employs graph theory to facilitate phrase representation and efficient matching. In the process of text mining, a grouping (clustering) approach is also employed to identify groups of documents such that each group represent a different topic in the underlying document collection. Document groups are tagged with topic labels through unsupervised keyphrase extraction from the document clusters. The approach serves in solving some of the difficult problems in e-learning where the volume of data could be overwhelming for the learner; such as automatically organizing documents and articles based on topics, and providing summaries for documents and groups of documents.

1 Introduction

Resources in learning environments are authored for the purpose of transferring knowledge to the learner. The growth of learning repositories and the ease of publishing and accessing information has created an environment where finding and making efficient use of the available information can be overwhelming. It is the job of data mining to help the learners digest large amounts of data by leveraging sophisticated techniques in data analysis, restructuring, and organization.

Learning resources are mainly found in textual form; *e.g.* text documents, web documents, articles, and papers, among other forms. Due to the unstructured and unrestricted nature of text documents, a special field in data mining was coined the term "text mining". It is the field studying the non-trivial extraction of implicit, previously unknown and potentially useful and significant information from text documents.

Text mining is generally considered more difficult than traditional data mining. This is attributed to the fact that traditional databases have fixed and known structure, while text documents are unstructured, or, as in the case of web documents, semi-structured. Thus, text mining involves a series of steps for data pre-processing and modeling in order to condition the data for structured data mining.

Text mining can help in many tasks that otherwise would require large manual effort. Common problems solved by text mining include, but not limited to, searching through documents, organizing documents, comparing documents, extracting key information, and summarizing documents. Methods in information retrieval, machine learning, information theory, and probability are employed to solve those problems.

Information extraction through text mining deals with finding particular data in text and web documents. The approaches used in this area include document parsing, analysis, and restructuring. This allows for restructuring existing learning material into current standards. Other approaches include identifying and extracting significant semi-structured information, extracting keywords and keyphrases from documents using phrase indexing and matching. These methods have high potential in e-learning due to their ability to automatically extract useful information, and tag learning objects with certain meta-data extracted from content.

Information organization through text mining provides an overview of the topics in a large set of documents without having to read the contents of individual documents. This can be achieved through data clustering and classification techniques. These techniques mainly rely on the analysis of keyword distribution in the documents. They also make use of similarity calculation through word and phrase matching. The end result is a more manageable grouping of documents tagged with topics and subjects.

While data clustering techniques are mainly used for content organization, it could be used to group learner profiles as well. In this case we can discover common interest groups of learners by judging the similarity between their profiles.

This chapter focuses on employing machine learning methods in finding relationships between text documents through phrase-based document modeling, similarity calculation, document clustering, and keyphrase extraction. Figure 1 illustrates the process of text mining in general, and refers to specific tasks as it is presented in this chapter. In particular, a set of documents is pre-processed through tokenization (identifying whole words and dropping punctuation symbols), removing stop words (very freuquent words like 'a', 'and', 'the'), and stemming (reducing different forms of a word into a single form). Then a model of the data is built using a graph-based representation of phrases in the documents. Next, pattern analysis is applied to detect similarities between the documents based on shared and significant phrases, followed by clustering the documents to form groups of documents, where each group contains only similar documents sharing the same topic. Finally, the process concludes by extracting keyphrases from the clusters and identifying the topic of each cluster.

This chapter is organized as follows. Section 2 introduces the document model used throughout the process. Section 3 presents the phrase matching capability of the model and the phrase-based similarity measure. Section 4 presents the document clustering algorithm. Section 5 presents the keyphrase extraction algorithm. Finally, section 6 provides a summary and concluding remarks.

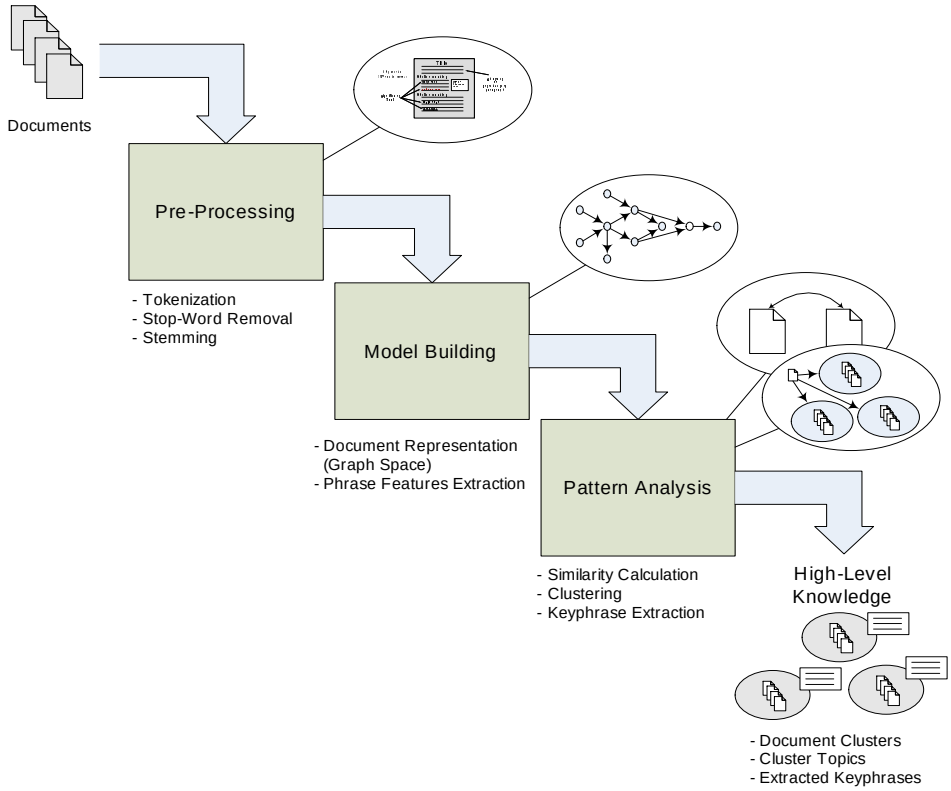


Fig. 1. Text mining process

2 Phrase-based Document Model

A process of data modeling is required to convert the input data into a form that is more suitable for processing by the data mining algorithm. In the case of text mining, input data are mainly text documents that do not necessarily obey a regular structure. The challenge is to convert the input space into feature space, where the features of the documents are expected to follow a fixed structure that can be manipulated by a text mining algorithm. The traditional document representation model, as well as the phrase-based model are introduced in this section.

2.1 Vector Space Model

By far the most common feature model in text mining is the vector space model, originally proposed by Salton *et al*in 1975 [1–3]. In this model, document fea-

tures are the words in the document collection, and feature values come from different term weighting schemes.

Each document is represented by a vector $\mathbf{d}$, in the term space, such that $\mathbf{d} = \{w_1, w_2, \ldots, w_n\}$, where w_i, $i = 1, \ldots, n$, is the weight of term i in the document. The weight of a term could be simply calculated as the *frequency* of the term in that document ($w_i = tf_i$); *i.e.* how many times it appeared in the document. A more popular term weighting scheme is TF×IDF (Term Frequency × Inverse Document Frequency), which takes into account the document frequency of a term (df_i), the number of documents in which the term appears. A typical inverse document frequency (idf) factor of this type is given by $\log(N/df_i)$. Thus the TF×IDF weight of a term is $w_i = tf_i \times \log(N/df_i)$. In other words, terms that appear more frequently in a certain document but less frequently in other documents are given higher weights in that document, since it has higher correlation with that document than others. On the other hand, terms that appear frequently in all documents are penalized in all documents since they have less discrimination power.

To represent every document with the same set of terms, we have to extract all the terms found in the documents and use them as our feature vector. To keep the feature vector dimension reasonable, sometimes only terms with the highest weights in all the documents are chosen as the features. Wong and Fu [4] showed that they could reduce the number of representative terms by choosing only the terms that have sufficient coverage over the document set.

Some algorithms [5, 4] refrain from using continuous term weights by using a binary feature vector, where each term weight is either 1 or 0, depending on whether it is present in the document or not, respectively. Wong and Fu [4] argued that the average term frequency in web documents is below 2 (based on statistical experiments), which does not indicate the actual importance of the term, thus a binary weighting scheme would be more suitable to this problem domain.

The simplicity of the model led to its wide adoption in the text mining literature. However, the independence between the words in the representation is one of its weaknesses. A more informed approach is to capture the phrase structure and word sequences in the document, thus providing context when comparing document features.

2.2 Graph Space Model

The model presented here for document representation is called the **Document Index Graph** (DIG). This model indexes the documents while maintaining the sentence structure in the original documents. This allows us to make use of more informative phrase matching rather than individual words matching. Moreover, DIG also captures the different levels of significance of the original sentences, thus allowing us to make use of sentence significance. Suffix trees are the closest structure to the proposed model, but they suffer from huge redundancy [6]. Apostolico [7] gives over 40 references on suffix trees, and Manber and Myers [8] add more recent ones. However, the proposed DIG model is not just an extension

or an enhancement of suffix trees, it takes a different perspective of how to match phrases efficiently, without the need for storing redundant information.

Phrasal indexing has been widely used in the information retrieval literature [9]. The work presented here takes it a step further toward an efficient way of indexing phrases with emphasis on applying phrase-based similarity as a way of clustering documents accurately.

DIG Structure Overview The DIG is a directed graph (digraph) $G = (V, E)$

where V: is a set of *nodes* $\{v_1, v_2, \ldots, v_n\}$, where each node v represents a unique word in the entire document set; and

E: is a set of *edges* $\{e_1, e_2, \ldots, e_m\}$, such that each edge e is an ordered pair of nodes (v_i, v_j). Edge (v_i, v_j) is from v_i to v_j , and v_j is adjacent to v_i. There will be an edge from v_i to v_j if, and only if, the word v_j appears successive to the word v_i in any document. A set of edges is said to be corresponding to a sentence in a document if they link the nodes corresponding to the sentence in the same order the words appeared in the sentence.

The above definition of the graph suggests that the number of nodes in the graph is the number of unique words in the document set; *i.e.* the vocabulary of the document set, since each node represents a single word in the whole document set.

Nodes in the graph carry information about the documents they appeared in, along with the sentence path information. Sentence structure is maintained by recording the edge along which each sentence continues. This essentially creates an inverted list of the documents, but with sentence information recorded in the inverted list.

Assume a sentence of m words appearing in one document consists of the following word sequence: $\{v_1, v_2, \ldots, v_m\}$. The sentence is represented in the graph by a path from v_1 to v_m, such that $(v_1, v_2)(v_2, v_3), \ldots, (v_{m-1}, v_m)$ are edges in the graph. Path information is stored in the vertices along the path to uniquely identify each sentence. Sentences that share sub-phrases will have shared parts of their paths in the graph that correspond to the shared sub-phrase.

To better illustrate the graph structure, Figure 2 presents a simple example graph that represents three documents. Each document contains a number of sentences with some overlap between the documents. As seen from the graph, an edge is created between two nodes only if the words represented by the two nodes appear successive in any document. Thus, sentences map into paths in the graph. Dotted lines represent sentences from document 1, dash-dotted lines represent sentences from document 2, and dashed lines represent sentences from document 3. If a phrase appears more than once in a document, the frequency of the individual words making up the phrase is increased, and the sentence information in the nodes reflects the multiple occurrence of such phrase. As mentioned earlier, matching phrases between documents becomes a task of finding shared paths in the graph between different documents.

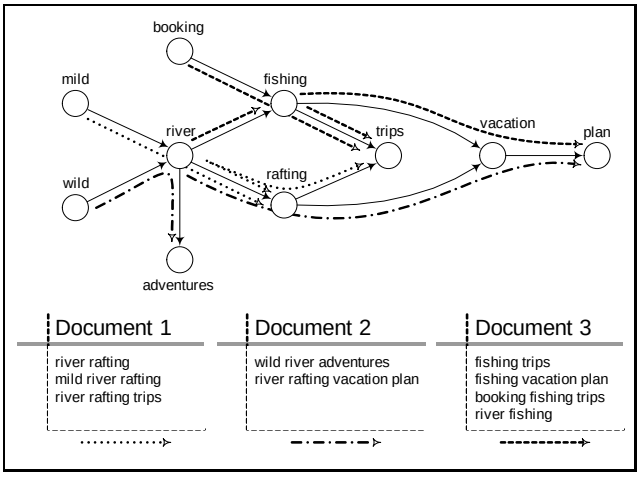


Fig. 2. Example of the Document Index Graph

DIG Construction The DIG is built incrementally by processing one document at a time. When a new document is introduced, it is scanned in sequential fashion, and the graph is updated with the new sentence information as necessary. New words are added to the graph as necessary and connected with other nodes to reflect the sentence structure. The graph building process becomes less memory demanding when no new words are introduced by a new document (or very few new words are introduced.) At this point the graph becomes more stable, and the only operation needed is to update the sentence structure in the graph to accommodate the new sentences introduced. It is very critical to note that introducing a new document will only require the inspection (or addition) of those words that appear in that document, and not every node in the graph. This is where the efficiency of the model comes from. Along with indexing the sentence structure, the level of significance of each sentence is also recorded in the graph. This allows us to recall such information when we measure the similarity with other documents.

Continuing from the example introduced earlier, the process of constructing the graph that represents the three documents is illustrated in Figure 3. The emphasis here is on the incremental construction process, where new nodes are added and new edges are created incrementally upon introducing a new document. We now define the incremental DIG construction process formally in terms of graph properties and operations.

Document Subgraph. Each document $\mathbf{d}_i$ is mapped to a subgraph g_i that represents this document in a stand-alone manner (an example is the first step in figure 3.) Each subgraph can be viewed as a *detached* subset of the DIG that represents the corresponding document in terms of the DIG properties: $g_i = \{V_i, E_i\}$, where V_i is the set of nodes corresponding

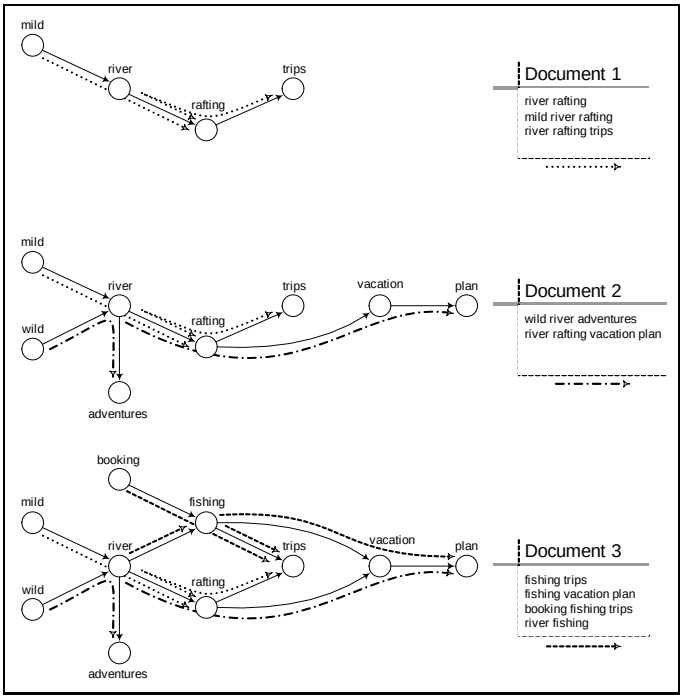


Fig. 3. Incremental construction of the Document Index Graph

to the unique words of $\mathbf{d}_i$, and E_i is the set of edges representing the sentence paths of $\mathbf{d}_i$.

Cumulative DIG. Let the DIG representation of the documents processed up to document $\mathbf{d}_{i-1}$ be G_{i-1}, and that of the documents processed up to document $\mathbf{d}_i$ be G_i. Computing G_i is done by *merging* G_{i-1} with the subgraph g_i:

$$G_i = G_{i-1} \cup g_i \tag{1}$$

G_i is said to be the *Cumulative DIG* of the documents processed up to document $\mathbf{d}_i$.

Phrase Matching. A list of matching phrases between document $\mathbf{d}_i$ and $\mathbf{d}_j$ is computed by *intersecting* the subgraphs of both documents, g_i and g_j, respectively. Let M_{ij} denote the such list, then:

$$M_{ij} = g_i \cap g_j \tag{2}$$

A list of matching phrases between document $\mathbf{d}_i$ and all previously processed documents is computed by intersecting the document subgraph g_i with the cumulative DIG G_{i-1}. Let M_i denote the such list, then:

$$M_i = g_i \cap G_{i-1} \tag{3}$$

Unlike traditional phrase matching techniques that are usually used in information retrieval literature, DIG provides complete information about full phrase matching between *every* pair of documents. While traditional phrase matching methods are aimed at searching and retrieval of documents that have matching phrases to a specific query, DIG is aimed at providing information about the degree of overlap between every pair of documents. This information will help in determining the degree of similarity between documents as will be explained in section 3.2.

3 Document Similarity Using Phrase Matching

Upon introducing a new document, finding matching phrases from previously seen documents becomes an easy task using DIG. Algorithm 1 describes the process of both incremental graph building and phrase matching. Instead of building document subgraphs and intersecting them with the cumulative DIG, the algorithm incrementally incorporates new documents into DIG while collecting matching phrases from previous documents at the same time.

3.1 Phrase Matching Using DIG

The procedure starts with a new document to process (line 1). Matching phrases from previous documents is done by keeping a list M that holds an entry for every previous document that shares a phrase with the current document $\mathbf{d}_i$. For each sentence (**for** loop at line 3) we process the words in the sentence sequentially, adding new words (as new nodes) to the graph, and constructing a path in the graph (by adding new edges if necessary) to represent the sentence we are processing.

As we continue along the sentence path, we update M by adding new matching phrases and their respective document identifiers, and extending phrase matches from the previous iteration (lines 14 to 16). We first consult the document table of v_{k-1} for documents that have sentences that continue along the edge e_k. Those documents share at least two terms with the current sentence under consideration. We examine the list M for any previous matching phrases (from previous iterations) to extend the current two-term phrase match (on edge e_k). This allows the extension of previous matches, and can continue for any-length phrase match. If there are no matching phrases at some point, we just update the respective nodes of the graph to reflect the new sentence path (line 19). After the whole document is processed, M will contain all the matching phrases between the current document and any previous document that shared at least one phrase with the new document. Finally we update G_i to be the current cumulative DIG, and output M as the list of documents with all the necessary information about the matching phrases, which will be used in similarity calculation later.

Algorithm 1 DIG incremental construction and phrase matching

Require: G_{i-1}: cumulative graph up to document $\mathbf{d}_{i-1}$
or G_0 if no documents were processed previously
1: $\mathbf{d}_i \leftarrow$ Next Document
2: $M \leftarrow$ Empty List {M is a list of matching phrases from previous documents}
3: **for** each sentence $\mathbf{s}_{ij}$ in $\mathbf{d}_i$ **do**
4: $v_1 \leftarrow t_{ij1}$ {first word in $\mathbf{s}_{ij}$}
5: **if** v_1 is not in G_{i-1} **then**
6: Add v_1 to G_{i-1}
7: **end if**
8: **for** each term $t_{ijk} \in s_{ij}$, $k = 2, \ldots, l_{ij}$ **do**
9: $v_k \leftarrow t_{ijk}$; $v_{k-1} \leftarrow t_{ij(k-1)}$; $e_k = (v_{k-1}, v_k)$
10: **if** v_k is not in G_{i-1} **then**
11: Add v_k to G_{i-1}
12: **end if**
13: **if** e_k is an edge in G_{i-1} **then**
14: Retrieve a list of document entries from v_{k-1} document table that have a sentence on the edge e_k
15: Extend previous matching phrases in M for phrases that continue along edge e_k
16: Add new matching phrases to M
17: **else**
18: Add edge e_k to G_{i-1}
19: **end if**
20: Update sentence path in nodes v_{k-1} and v_k
21: **end for**
22: **end for**
23: $G_i \leftarrow G_{i-1}$
24: Output matching phrases list M

The average number of the matching documents at any node tends to grow slowly. The actual performance depends on how much overlap of phrases there is in the document set; the more matching phrases the more time it takes to process the whole set, but the more accuracy we get for similarity, and vice versa. The trade-off is corpus-dependent, but in general for web documents it is typically a balance between speed and accuracy.

This efficient performance of construction/phrase-matching lends itself to online incremental processing, such as processing the results of a web search engine retrieved list of documents. The algorithm processed 2000 news group articles in as low as 44 seconds, while it processed 2340 moderate sized web documents in a little over 5 minutes.

3.2 A Phrase-based Similarity Measure

As mentioned earlier, phrases convey local context information, which is essential in determining an accurate similarity between documents. Towards this end we

devised a similarity measure based on matching phrases rather than individual terms. This measure exploits the information extracted from the previous phrase matching algorithm to better judge the similarity between the documents. This is related to the work of Isaacs *et al*[10] who used a pair-wise probabilistic document similarity measure based on Information Theory. Although they showed it could improve on traditional similarity measures, but it is still fundamentally based on the vector space model representation.

The phrase similarity between two documents is calculated based on the list of matching phrases between the two documents. From an information theoretic point of view, the similarity between two objects is regarded as how much they share in common. The cosine and the Jaccard measures are indeed of such nature, but they are essentially used as single-term based similarity measures. Lin [11] gave a formal definition for any information theoretic similarity measure in the form of:

$$\text{sim}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cap \mathbf{y}}{\mathbf{x} \cup \mathbf{y}} \tag{4}$$

The basic assumption here is that the similarity between two documents is based on the ratio of how much they overlap to their union, all in terms of phrases. This definition still coincides with the major assumption of the cosine and the Jaccard measures, and to Lin's definition as well. This phrase-based similarity measure is a function of four factors:

- The number of matching phrases P,
- The lengthes of the matching phrases ($l_i : i = 1, 2, \ldots, P$),
- The frequencies of the matching phrases in both documents (f_{1i} and f_{2i} : $i = 1, 2, \ldots, P$), and
- The levels of significance (*weight*) of the matching phrases in both document (w_{1i} and $w_{2i} : i = 1, 2, \ldots, P$).

Frequency of phrases is an important factor in the similarity measure. The more frequent the phrase appears in both documents, the more similar they tend to be. Similarly, the level of significance of the matching phrase in both documents should be taken into consideration.

The phrase similarity between two documents, $\mathbf{d_1}$ and $\mathbf{d_2}$, is calculated using the following empirical equation:

$$\text{sim}_p(\mathbf{d_1}, \mathbf{d_2}) = \frac{\sqrt{\sum_{i=1}^{P} [g(l_i) \cdot (f_{1i} w_{1i} + f_{2i} w_{2i})]^2}}{\sum_j |s_{1j}| \cdot w_{1j} + \sum_k |s_{2k}| \cdot w_{2k}} \tag{5}$$

where $g(l_i)$ is a function that scores the matching phrase length, giving higher score as the matching phrase length approaches the length of the original sentence; $|s_{1j}|$ and $|s_{2k}|$ are the original sentence lengths from document $\mathbf{d_1}$ and $\mathbf{d_2}$, respectively. The equation rewards longer phrase matches with higher level of significance, and with higher frequency in both documents. The function $g(l_i)$ in the implemented system was used as:

$$g(l_i) = (l_i/|s_i|)^\gamma \tag{6}$$

where $|s_i|$ is the original phrase length, and γ is a sentence fragmentation factor with values greater than or equal to 1. If γ is 1, two halves of a sentence could be matched independently and would be treated as a whole sentence match. However, by increasing γ we can avoid this situation, and score whole sentence matches higher than fractions of sentences. A value of 1.2 for γ was found to produce best results. he normalization by the length of the two documents in equation (5) is necessary to be able to compare the similarities from other documents.

3.3 Combining Single-term and Phrase Similarities

If the similarity between documents is based solely on matching phrases, and not single-terms at the same time, related documents could be judged as non-similar if they do not share enough phrases (a typical case.) Shared phrases provide important local context matching, but sometimes similarity based on phrases only is not sufficient. To alleviate this problem, and to produce high quality clusters, we combined single-term similarity measure with our phrase-based similarity measure. Experimental results to justify this claim is given in section 3.4. We used the *cosine correlation* similarity measure [1], with TF-IDF (Term Frequency–Inverse Document Frequency) term weights, as the single-term similarity measure. The cosine measure was chosen due to its wide use in the document clustering literature, and since it is described as being able to capture human categorization behavior well [12]. The TF-IDF weighting is also a widely used term weighting scheme [13].

Recall that the cosine measure calculates the cosine of the angle between the two document vectors. Accordingly our term-based similarity measure (sim_t) is given as:

$$\text{sim}_t(\mathbf{d_1}, \mathbf{d_2}) = \cos(\mathbf{d_1}, \mathbf{d_2}) = \frac{\mathbf{d_1} \cdot \mathbf{d_2}}{\|\mathbf{d_1}\|\|\mathbf{d_2}\|} \tag{7}$$

where the vectors $\mathbf{d_1}$ and $\mathbf{d_2}$ are represented as term weights calculated using TF-IDF weighting scheme.

The combination of the term-based and the phrase-based similarity measures is a weighted average of the two quantities from equations (5) and (7), and is given by equation (8). The reason for separating single-terms and phrases in the similarity equation, as opposed to treating a single-term as a one-word-phrase, is to evaluate the *blending* factor between the two quantities, and see the effect of phrases in similarity as opposed to single-terms.

$$\text{sim}(\mathbf{d_1}, \mathbf{d_2}) = \alpha \cdot \text{sim}_p(\mathbf{d_1}, \mathbf{d_2}) + (1 - \alpha) \cdot \text{sim}_t(\mathbf{d_1}, \mathbf{d_2}) \tag{8}$$

where α is a value in the interval $[0, 1]$ which determines the weight of the phrase similarity measure, or, as we call it, the *Similarity Blend Factor*. According to the experimental results discussed in section 3.4 we found that a value between 0.6 and 0.8 for α results in the maximum improvement in the clustering quality.

3.4 Effect of Phrase-based Similarity on Clustering Quality

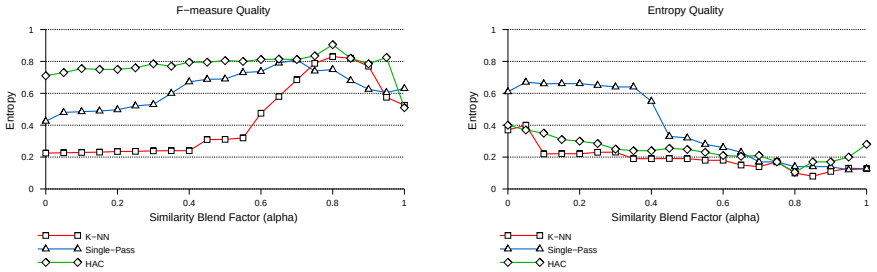
The similarities calculated by our algorithm were used to construct a similarity matrix between the documents. We elected to use three standard document clustering techniques for testing the effect of phrase similarity on clustering [14]: (1) Hierarchical Agglomerative Clustering (HAC), (2) Single Pass Clustering, and (3) K-Nearest Neighbor Clustering (k-NN). For each of the algorithms, we constructed the similarity matrix and let the algorithm cluster the documents based on the presented similarity matrix.

Table 1. Phrase-based clustering improvement

	Single-Term Similarity		Combined Similarity		Improvement
	F-measure	Entropy	F-measure	Entropy	
DS1 - UW-CAN					
HAC [3]	0.709	0.351	0.904	0.103	+19.5%F, -24.8%E
Single Pass [4]	0.427	0.613	0.817	0.151	+39.0%F, -46.2%E
k-NN [5]	0.228	0.173	0.834	0.082	+60.6%F, -9.1%E
DS2 - Yahoo! news					
HAC	0.355	0.211	0.725	0.01	+37.0%F, -20.1%E
Single Pass	0.344	0.274	0.547	0.048	+20.3%F, -22.6%E
k-NN	0.453	0.163	0.733	0.022	+28.0%F, -14.1%E
DS3 - 20-newsgroups					
HAC	0.17	0.347	0.463	0.069	+29.3%F, -27.8%E
Single Pass	0.284	0.684	0.358	0.138	+7.4%F, -54.6%E
k-NN	0.197	0.398	0.349	0.09	+15.2%F, 30.8%E

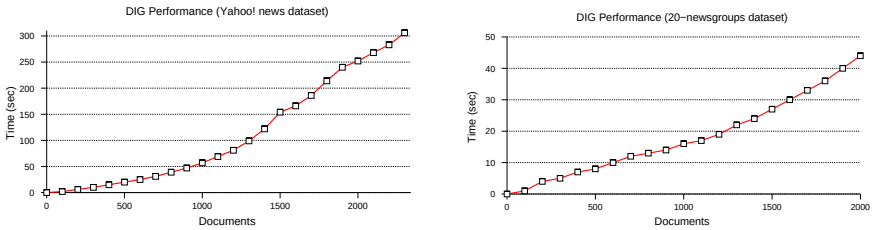
The results listed in Table 1 show the improvement in the clustering quality using the combined similarity measure. We use the F-measure and Entropy evaluation measures for judging the quality of clustering. Better clustering should have higher F-measure and lower Entropy. The improvements shown were achieved at a similarity blend factor between 70% and 80% (phrase similarity weight). The parameters chosen for the different algorithms were the ones that produced best results. The percentage of improvement ranges from 19.5% to 60.6% increase in the F-measure quality, and 9.1% to 46.2% drop in Entropy (lower is better for Entropy). It is obvious that the phrase based similarity plays an important role in accurately judging the relation between documents. It is known that Single Pass clustering is very sensitive to noise; that is why it has the worst performance. However, when the phrase similarity was introduced, the quality of clusters produced was pushed close to that produced by HAC and k-NN .

In order to better understand the effect of the phrase similarity on the clustering quality, we generated a clustering quality profile against the similarity blend factor. Figure 4 illustrates the effect of introducing the phrase similarity on the F-measure and the entropy of the resulting clusters. The alpha parameter is the similarity blend factor presented in equation 8. It is obvious that the phrase similarity enhances the quality of clustering until a certain point (around a weight of 80%) and then its effect starts bringing down the quality. As we mentioned in section 3.3 that phrases alone cannot capture all the similarity information between documents, the single-term similarity is still required, but to



(a) F-measure (b) Entropy

Fig. 4. Effect of phrase similarity on clustering quality



(a) DIG Performance using the Yahoo! news data set (b) DIG Performance using the 20-newsgroups data set

Fig. 5. DIG performance

a smaller degree. The results show that both evaluation measures are optimized in the same trend with respect to the blend factor.

The performance of the model was closely examined to make sure that the phrase matching algorithm is scalable enough for moderate to large data sets. The experiments were performed on a Pentium 4, 2.0 GHz machine with 512MB of main memory. The system was written in C++. Figure 5 shows the performance of the graph construction and phrase matching algorithm for the two different. In both cases the algorithm performed in a near-linear time. Although the two data sets contain close number of documents, the Yahoo news data set took about an order of magnitude more than the 20-newsgroup data set to build the graph and complete the phrase matching. This is attributed to two factors: (1) the Yahoo data set average words per document is almost twice that of 20-newsgroups, so we match more phrases per document, and (2) the Yahoo data set has a larger amount of shared phrases between documents on average than the 20-newsgroups data set. News group articles rarely share a large amount of phrases (except when someone quotes another post), so on average we do not need to match large number of phrases per document.

4 Document Clustering Using Similarity Histograms

In this section we present a brief overview of incremental clustering algorithms, and introduce the proposed algorithm, based on pair-wise document similarity, and employ it as part of the whole web document clustering system.

The role of a document similarity measure is to provide judgement on the *closeness* of documents to each other. However, it is up to the clustering method how to make use of such similarity calculation. Steinbach *et al*[15] give a good comparison of document clustering techniques. A large array of data clustering methods can be also found in [14]. Beil *et al* [16] proposed a clustering algorithm based on frequent terms that address the high dimensionality problem of text data sets.

The idea here is to employ an incremental clustering method that will exploit our similarity measure to produce clusters of high quality.

Incremental clustering is an essential strategy for online applications, where time is a critical factor for usability. Incremental clustering algorithms work by processing data objects one at a time, incrementally assigning data objects to their respective clusters while they progress. The process is simple enough, but faces several challenges. How to determine to which cluster the next object should be assigned? How to deal with the problem of insertion order? Once an object has been assigned to a cluster, should its assignment to the cluster be frozen or is it allowed to be re-assigned to other clusters later on?

Usually a heuristic method is employed to deal with the above challenges. A "good" incremental clustering algorithm has to find the respective cluster for each newly introduced object without significantly sacrificing the accuracy of clustering due to insertion order or fixed object-to-cluster assignment. We will briefly discuss four incremental clustering methods in the light of the above challenges, before we introduce our proposed method.

Single-Pass Clustering [17, 18]. This algorithm basically processes documents sequentially, and compares each document to all existing clusters. If the similarity between the document and any cluster is above a certain threshold, then the document is added to the closest cluster; otherwise it forms its own cluster. Usually the method for determining the similarity between a document and a cluster is done by computing the average similarity of the document to all documents in that cluster.

***K*-Nearest Neighbor Clustering** [19, 18]. Although k-NN is mostly known to be used for classification, it has also been used for clustering (example could be found in [20].) For each new document, the algorithm computes its similarity to every other document, and chooses the top k documents. The new document is assigned to the cluster where the majority of the top k documents are assigned.

Suffix Tree Clustering (STC). Introduced by Zamir *et al*[21] in 1997, the idea behind the STC algorithm is to build a tree of phrase suffixes shared between multiple documents. The documents sharing a suffix are considered as a base cluster. Base clusters are then combined together if they have a document overlap of 50% or more. The algorithm has two drawbacks. First, although the structure used is a compact tree, suffixes can appear multiple times if they are

part of larger shared suffixes. The other drawback is that the second phase of the algorithm is not incremental. Combining base clusters into final clusters has to be done in a non-incremental way. The algorithm deals properly with the insertion order problem though, since any insertion order will lead to the same result suffix tree.

DC-tree Clustering. The DC-tree incremental algorithm was introduced by Wong *et al*[4] in 2000. The algorithm is based on the B^+-tree structure. Unlike the STC algorithm, this algorithm is based on vector space representation of the documents. Most of the algorithm operations are borrowed from the B^+-tree operations. Each node in the tree is a representation of a cluster, where a cluster is represented by the combined feature vectors of its individual documents. Inserting a new document involves comparison of the document feature vector with the cluster vectors at one level of the tree, and descending to the most similar cluster. The algorithm defines several parameters and thresholds for the various operations. The algorithm suffers from two problems though. Once a document is assigned to a cluster it is not allowed to be re-assigned later to a newly created cluster. Second, which is a consequence of the first drawback, clusters are not allowed to overlap; *i.e.* a document can belong to only one cluster.

4.1 Similarity Histogram-based Incremental Clustering

The clustering approach proposed here is an incremental dynamic method of building the clusters. We adopt an overlapped cluster model. The key concept for the similarity histogram-based clustering method (referred to as **SHC** hereafter) is to keep each cluster at a high degree of coherency at any time. We represent the coherency of a cluster with a new concept called **Cluster Similarity Histogram**.

> **Cluster Similarity Histogram**. A concise statistical representation of the set of pair-wise document similarities distribution in the cluster. A number of *bins* in the histogram correspond to fixed similarity value intervals. Each bin contains the count of pair-wise document similarities in the corresponding interval.

Our objective is to keep each cluster as coherent as possible. In terms of the similarity histogram concept this translates to maximizing the number of similarities in the high similarity intervals. To achieve this goal in an incremental fashion, we judge the effect of adding a new document to a certain cluster. If the document is going to *degrade* the distribution of the similarities in the clusters very much, it should not be added, otherwise it is added. A much stricter strategy would be to add documents that will *enhance* the similarity distribution. However, this could create a problem with perfect clusters. The document will be rejected by the cluster even if it has high similarity to most of the documents to the cluster (because it is perfect).

We judge the quality of a similarity histogram (cluster cohesiveness) by calculating the ratio of the count of similarities above a certain similarity threshold

S_T to the total count of similarities. The higher this ratio, the more coherent is the cluster.

Let n_c be the number of the documents in a cluster. The number of pair-wise similarities in the cluster is $m_c = n_c(n_c + 1)/2$. Let $S = \{s_i : i = 1, \ldots, m_c\}$ be the set of similarities in the cluster. The histogram of the similarities in the cluster is represented as:

$$H = \{h_i : i = 1, \ldots, B\} \tag{9a}$$

$$h_i = \text{count}(s_k) \qquad s_{li} \leq s_k < s_{ui} \tag{9b}$$

where B: the number of histogram bins,
h_i: the count of similarities in bin i,
s_{li}: the lower similarity bound of bin i, and
s_{ui}: the upper similarity bound of bin i.

The histogram ratio (HR) of a cluster is the measure of cohesiveness of the cluster as described above, and is calculated as:

$$HR_c = \frac{\sum_{i=T}^{B} h_i}{\sum_{j=1}^{B} h_j} \tag{10a}$$

$$T = \lfloor S_T \cdot B \rfloor \tag{10b}$$

where HR_c: the histogram ratio of cluster c,
S_T: the similarity threshold, and
T: the bin number corresponding to the similarity threshold.

Basically we would like to keep the histogram ratio of each cluster high. However, since we allow documents that can degrade the histogram ratio to be added, this could result in a chain effect of degrading the ratio to zero eventually. To prevent this, we set a *minimum* histogram ratio $HR_{\min}$ that clusters should maintain. We also do not allow adding a document that will bring down the histogram ratio significantly (even if still above $HR_{\min}$). This is to prevent a bad document from severely bringing down cluster quality by one single document addition.

We now present the incremental clustering algorithm based on the above framework (Algorithm 2). The algorithm works incrementally by receiving a new document, and for each cluster calculates the cluster histogram before and after simulating the addition of the document (lines 4-6). The old and new histogram ratios are compared and if the new ratio is greater than or equal to the old one, the document is added to the cluster. If the new ratio is less than the old one by no more than ε and still above $HR_{\min}$, it is added (lines 7-9). Otherwise it is not added. If after checking all clusters the document was not assigned to any cluster, a new cluster is created and the document is added to it (lines 11-15).

In comparison with the criteria of single-pass clustering and k-NN clustering, the similarity histogram ratio as a coherency measure provides a more representative measure of the tightness of the documents in the cluster, and how the

Algorithm 2 Similarity Histogram-based Incremental Document Clustering

```
1: L ← Empty List {Cluster List}
2: for each document d do
3:    for each cluster c in L do
4:       HR_old = HR_c
5:       Simulate adding d to c
6:       HR_new = HR_c
7:       if (HR_new ≥ HR_old) OR ((HR_new > HR_min) AND (HR_old − HR_new < ε)) then
8:          Add d to c
9:       end if
10:   end for
11:   if d was not added to any cluster then
12:      Create a new cluster c
13:      ADD d to c
14:      ADD c to L
15:   end if
16: end for
```

external document would affect such tightness. On the other hand, single-pass compares the external document to the *average* of the similarities in the cluster, while the k-NN method takes into consideration only a few similarities that might be outliers, and that is why we sometimes need to increase the value of the parameter k to get better results from k-NN . This was the main reason for devising such a concise cluster coherency measure and employing it in assessing the effect of external documents on each cluster.

4.2 Similarity Histogram-based Clustering Evaluation

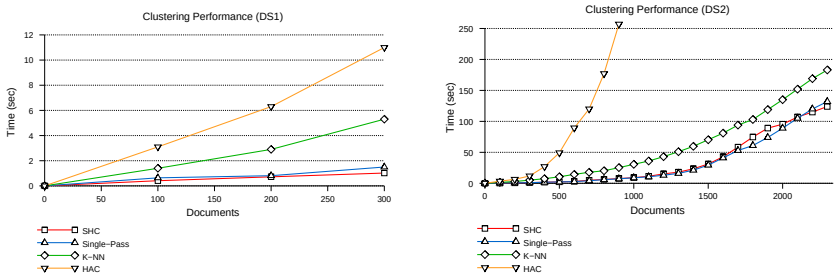
The SHC method was evaluated using two document sets (DS1 and DS2). We relied on the same evaluation measures F-measure and Entropy.

Table 2 shows the result of SHC against HAC, Single-Pass, and k-NN clustering. For the first data set, the improvement was very significant, reaching over 20% improvement over k-NN (in terms of F-measure), 3% improvement over HAC, and 29% improvement over Single-Pass. For the second data set an improvement between 10% to 18% was achieved over the other methods. However, the absolute F-measure was not really high compared to the first data set. The parameters chosen for the different algorithms were the ones that produced best results.

By examining the actual documents in DS2 and their classification it turns out that the documents do not have enough overlap in each individual class, which makes it difficult to have an accurate similarity calculation between the documents. However, we were able to push the quality of clustering further by relying on accurate and robust phrase matching similarity calculation, and achieve higher clustering quality.

Table 2. SHC Improvement

	DS1			DS2		
	F-measure	Entropy	S[6]	F-measure	Entropy	S
SHC	0.931	0.119	0.504	0.682	0.156	0.497
HAC[7]	0.901	0.211	0.455	0.584	0.281	0.398
Single-Pass[8]	0.641	0.313	0.385	0.502	0.250	0.311
k-NN [9]	0.727	0.173	0.367	0.522	0.161	0.452



(a) Clustering Performance - DS1 (b) Clustering Performance - DS2

Fig. 6. SHC Clustering Performance

The time performance comparison of the different clustering algorithms is illustrated in figure 6, showing the performance for both data sets. The performance of SHC is comparable to single-pass and k-NN , while being much better than HAC. The reason for the gain in performance over HAC is because HAC spends so much time in recalculating the similarities between the newly merged cluster and all other clusters during every iteration, which brings its performance down significantly. On the other hand, SHC, single-pass, and k-NN share the same general strategy for processing documents, without having to recalculate similarities at each step. Thus, while the SHC algorithm generates better quality clustering, it still exhibits the same, or better, performance as other incremental algorithms in its class.

5 Keyphrase Extraction from Document Clusters

Document clusters are often represented as a membership matrix, where on one dimension are the document identifiers and on the other dimension are the cluster identifiers. An element in the membership matrix determines whether the document belongs to a cluster or not (if binary membership is used), or the degree of membership of the document to the cluster (if fuzzy membership is used).

This kind of cluster representation is useful for testing the accuracy of clustering, but not very useful for humans. A more easier representation for clusters is to put labels to the clusters so that the end user can spot interesting clusters

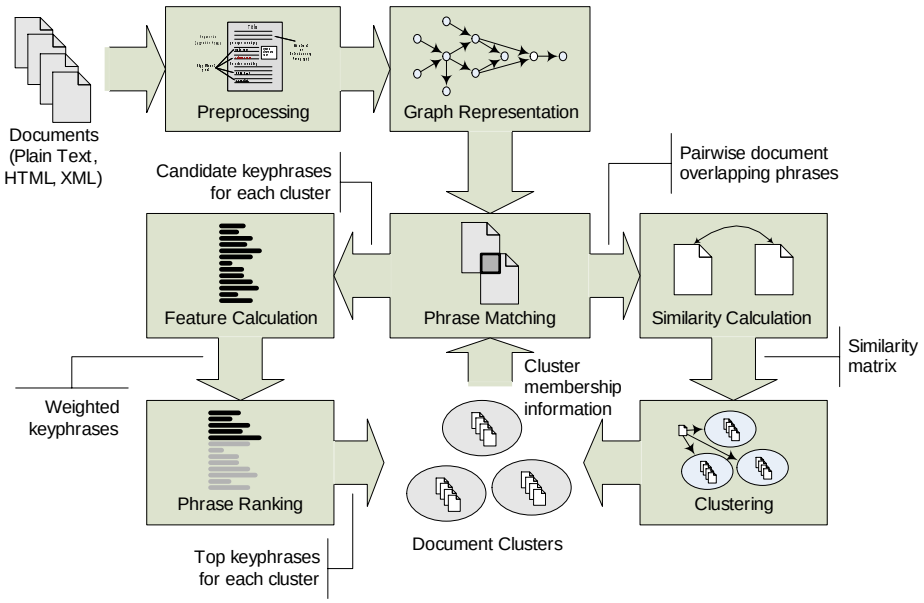


Fig. 7. CorePhrase Keyphrase Extraction System

without having to look at individual documents in each cluster. That is where keyphrase extraction comes into play.

In this section we present a highly accurate method for extracting keyphrases from document clusters, with no prior knowledge about the documents; *i.e.* it is domain-independent. The algorithm is called **CorePhrase**, and is based on finding a set of core phrases that best describe a document cluster.

The algorithm leverages the DIG structure presented earlier to intersect every pair of documents to extract their shared phrases. A list of candidate keyphrases for the cluster is then generated by consolidating all shared phrases in a cluster. The extracted candidate keyphrases are then analyzed for frequency, span over the document set, and other features. Each phrase is assigned a score based on its features, then the list is ranked and the top phrases are output as the descriptive topic of the document cluster. Four scoring method variants are employed and their performance is analyzed. Figure 7 illustrates the different components of the keyphrase extraction system.

5.1 Extraction of Candidate Keyphrases

A candidate keyphrase that has the power to represent a set of documents in a cluster (rather than a single document) would naturally lie at the *intersection* of those documents. The CorePhrase algorithm works by first finding all possible keyphrase candidates through matching document pairs together, extracting all matching phrases between document pairs. A master list of candidate phrases for

the document cluster is then constructed from the pairwise document matching lists by consolidating the individual lists to remove duplicates. The resulting list contains all phrases that are shared by at least two documents.

This process of matching every pair of documents is inherently $O(n^2)$. However, by using a proven method of document phrase indexing graph structure, known as the Document Index Graph (DIG), the algorithm can achieve this goal in near-linear time [22]. In DIG, phrase matching is done in an incremental fashion; all documents up to document $\mathbf{d}_i$ are represented by a graph structure, and, upon introducing a new document $\mathbf{d}_{i+1}$, the new document is matched to the graph to extract matching phrases with all previous documents. The new document is then added to the graph. This process produces complete phrase-matching output between every pair of documents in near-linear time, with arbitrary length phrases.

Figure 8 illustrates the process of phrase matching between two documents. In the figure, the two subgraphs of two documents are matched to get the list of phrases shared between them.

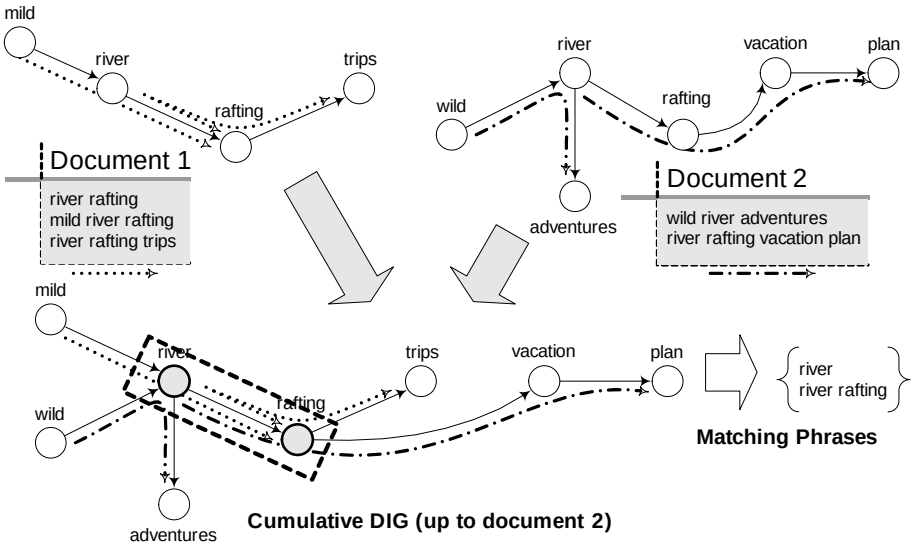


Fig. 8. Phrase Matching Using Document Index Graph

When a matching phrase, p_{ij}, is found between documents $\mathbf{d}_i$ and $\mathbf{d}_j$, we calculate its features with respect to each document, $\mathbf{p}_i$ and $\mathbf{p}_j$, respectively, according to section 5.2.

Since this method outputs matching phrases for each new document, it is essential to keep a *master list*, M, of unique matched phrases, which will be used as the list of candidate keyphrases. The following simple procedure keeps this list updated:

Algorithm 3 Candidate Keyphrase Extraction

{calculate M_i for document $\mathbf{d}_i$ using (3)}
$M_{ij} = \{p_{ij}: 1 < j < i\}$: matching phrases between $\mathbf{d}_i$ and $\mathbf{d}_j$
$M_i = \{M_{ij}\}$: matching phrases of $\mathbf{d}_i$
for each phrase p_{ij} in M_i **do**
  if phrase p_{ij} is in master list M **then**
    add feature vector $\mathbf{p}_i$ to p_{ij} in M
    add feature vector $\mathbf{p}_j$ to p_{ij} in M if not present
  else
    add p_{ij} to M
    add feature vectors $\mathbf{p}_i$ and $\mathbf{p}_j$ to p_{ij} in M
  end if
end for
for each *unique* phrase p_k in M **do**
  calculate averages of feature vectors associated with p_k
end for

The set of matching phrases from all documents forms a pool of candidate keyphrases. Each phrase in this pool is guaranteed to have been shared by at least two documents.

It should be noted that using the matching phrases from multi-document sets as candidate keyphrases saves us from problems often faced by single-document keyphrase extraction, namely that of having to identify possible candidates using heuristic techniques, such as the case in the Kea [23] and Extractor [24] algorithms.

5.2 Phrase Features

In order to judge the quality of the candidate keyphrases, we need to differentiate between them based on quantitative features. Each candidate keyphrase p is assigned the following features:

df: **document frequency**; the number of documents in which the phrase appeared, normalized by the total number of documents.

$$df = \frac{\mid \text{documents containing } p \mid}{\mid \text{all documents} \mid}$$

w: average **weight**; the average weight of the phrase over all documents. The weight of a phrase in a document is calculated using structural text cues. Examples: title phrases have maximum weight, section headings are weighted less, while body text is weighted lowest.

pf: average **phrase frequency**; the average number of times this phrase has appeared in one document, normalized by the length of the document in words.

$$pf = \arg avg \left[\frac{\mid \text{occurrences of } p \mid}{\mid \text{words in document} \mid} \right]$$

d: average phrase **depth**; the location of the first occurrence of the phrase in the document.

$$d = \arg avg \left[1 - \frac{\mid \text{words before first occurrence} \mid}{\mid \text{words in document} \mid} \right]$$

Those features will be used to rank the candidate phrases. In particular, we want phrases that appear in more documents (high *df*), have higher weights (high w), higher frequencies (high pf), and shallow depth[10].

The *df* feature can be regarded as the *support* of the keyphrase; *i.e.* from a frequent-set analysis point of view, *df* tells how many items (documents) support the keyphrase. Since we are extracting keyphrases that are shared by at least two documents, the minimum support is accordingly two. Although this may seem unnecessarily low support value, when the keyphrases are ranked (as described in the next section), the top ranking phrases usually exhibit high support.

5.3 Phrase Ranking

In single-document keyphrase extraction setting, the above phrase features will be used as input vectors to a machine learning algorithm for training. The model is then applied to unseen documents to extract the keyphrases. However, in our case we are looking at discovering "good" keyphrases from multi-document data sets or clusters. Thus, we will use the features to calculate a *score* for each phrase, rank the phrases by score, and select a number of the top phrases as the ones describing the topic of the cluster.

There are two phrase scoring formulas used, as well as two methods of assigning the score to the candidate phrases, for a total of four variants of the CorePhrase algorithm.

First Scoring Formula. The score of each phrase p is calculated using the following empirical formula:

$$\text{score}(p) = (w \cdot pf) \times -\log(1 - df) \tag{11}$$

The equation is derived from the tf×idf term weighting measure; however, we are rewarding phrases that appear in more documents (high *df*) rather than punishing those phrases. Notice also that the first scoring formula does not take the *depth* feature into account. We will refer to the variant of the algorithm that uses this formula as CorePhrase-1.

Second Scoring Formula. By examining the distribution of the values of each feature in a typical corpus, it was found that the *weight* and *frequency* features usually have low values compared to the *depth* feature. To take this fact into account, it was necessary to "expand" the *weight* and *frequency* features by

[10] It might seem counter-intuitive to look for phrases with high *df* to readers familiar with the tf-idf term weighting scheme. Remember that we are not scoring the phrase with respect to a particular document, but rather with respect to the whole document set. So the more common a phrase is across all documents the better.

taking their square root, and to "compact" the *depth* by squaring it. This helps even out the feature distributions and prevents one feature from dominating the score equation. The formula is given in equation 12.

$$\text{score}(p) = (\sqrt{w \cdot pf} \cdot d^2) \times -\log(1 - df) \tag{12}$$

We will refer to the variant of the algorithm that uses this formula as CorePhrase-2.

Word weight-based score assignment. A *modified* score assignment scheme based on word weights is also used:

- First, assign *initial* scores to each phrase based on phrase scoring formulas given above.
- Construct a list of unique individual words out of the candidate phrases.
- For each word: add up all the scores of the phrases in which this word appeared to create a word weight.
- For each phrase: assign the *final* phrase score by adding the individual word weights of the constituent words and average them.

We will refer to the variants of the algorithm that use this method as CorePhrase-1M and CorePhrase-2M, based on the equation that was used to assign the initial phrase scores.

5.4 Keyphrase Extraction Evaluation

For the evaluation of keyphrase extraction, in addition to subjective evaluation of the extracted keyphrases, we relied on two other extrinsic evaluation measures that quantitatively assess how well the extracted keyphrases relate to the topic of the original class or cluster. The name of each class represents the reference topic name against which the extracted keyphrases are compared for evaluation.

The first measure is called *overlap*, which measures the similarity between each extracted keyphrase to the predefined topic phrase of the cluster. The similarity is based on how many terms are shared between the two phrases. The overlap between an extracted keyphrase p_i and the topic phrase p_t is defined as:

$$\text{overlap}(p_i, p_t) = \frac{\mid p_i \cap p_t \mid}{\mid p_i \cup p_t \mid} \tag{13}$$

Evaluating each extracted keyphrase alone might not give a good idea of how the whole set of top k phrases fit the topic. To evaluate the top k keyphrases as a set, we take the average overlap of the whole set. This measure is essentially telling us how well the top keyphrases, as a set, *fit* the reference topic.

The second evaluation measure is called *precision*[11], which gives an indication of how high the single keyphrase that best fits the topic is ranked. The best keyphrase is defined as the first keyphrase, in the top k, that has maximum

[11] This is not the same as the precision measure usually used in the information retrieval literature.

overlap with the reference topic. Thus, the precision for the set of top k phrases ($\mathbf{p}^k$) with respect to the reference topic p_t is defined as:

$$\text{precision}(\mathbf{p}^k, p_t) = \text{overlap}(p_{\max}, p_t) \cdot \left[1 - \frac{\text{rank}(p_{\max}) - 1}{k}\right] \tag{14}$$

where $p_{\max} \in \mathbf{p}^k$ is the first phrase with maximum overlap in the top k phrases; and $\text{rank}(p_{\max})$ is its rank in the top k. In other words, precision tells us how *high* in the ranking the best phrase appears. For example, if we get a perfect overlap in the first rank, precision is maximum. The lower the best phrase comes in the ranking, the lower the precision.

5.5 Keyphrase Extraction Results

We have applied the CorePhrase algorithm on ten clusters produced from two data sets. The documents in each cluster were processed by the four variants of the CorePhrase algorithm. The extracted keyphrases are ranked in descending order according to their score, and the top 10 keyphrases were selected for output by the algorithm. In addition, a keyword-based extraction algorithm was used as a baseline for comparison. The algorithm extracts the centroid vector of a cluster represented as a set of keywords and selects the top frequent keywords in the cluster. This method is considered representative of most cluster labeling methods.

Table 3 shows the results of keyphrase extraction by the CorePhrase algorithm variants for three of the classes (two classes from the first data subset, and one classe from the second subset.) The phrases in the results are shown in stemmed form, with stop words removed. In a real system the output of the algorithm would have to be in the original unstemmed form for presentation to the end user.

The keyphrases extracted by the variants of the CorePhrase[12] algorithm are very close to the reference topic, which is a subjective verification of the algorithm correctness. We leave it to the reader to judge the quality of the keyphrases.

A more concrete evaluation based on the quantitative measures, overlap and precision, is illustrated in figure 5.5 (only CorePhrase-2 and CorePhrase-2M are shown). For each of the four variants of the CorePhrase algorithm, in addition to the baseline keyword centroid algorithm, we report the overlap and precision. The average overlap is taken over the top 10 keyphrases/keywords of each cluster.

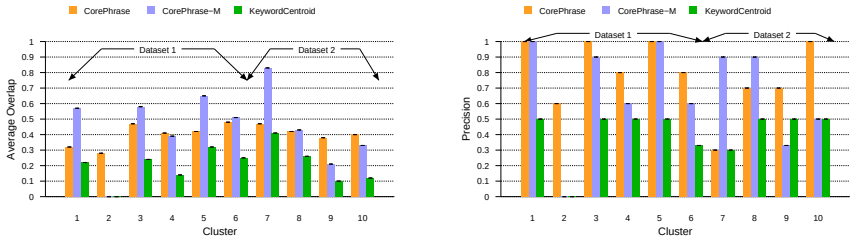
The first observation is that CorePhrase performs consistently better than the keyword centroid method. This is attributed to the keyphrases being in greater overlap with the reference topic than the naturally-shorter keywords. An interesting observation also is that CorePhrase-M, which is based on weighted words for phrase-scoring, and the keyword centroid follow the same trend. This is due to the link between the phrase scores and their constituent word scores.

[12] Throughout this discussion the name CorePhrase will refer to both CorePhrase-1 and CorePhrase-2, while CorePhrase-M will refer to both CorePhrase-1M and CorePhrase-2M; unless otherwise specified.

Table 3. Keyphrase Extraction Results – Top 10 Keyphrases

	CorePhrase-1	CorePhrase-2	CorePhrase-1M	CorePhrase-2M
			canada transporation	
1	canada transport	canada transport	transport canada	canada transport
2	panel recommend	canada transport act	canada transport	transport canada
3	transport associ	transport act	road transport	transport act
4	transport associ canada	transport associ	transport issu	transport issu
5	associ canada	panel recommend	govern transport	recommend transport
6	canada transport act	unit state	surfac transport	transport polici canada transport
7	transport act	transport associ canada tac	public transport	canadian transport
8	road transport	associ canada tac	transport public	transport public
9	transport infrastructur	canada tac	transport infrastructur	public transport
10	transport associ canada tac	public privat sector	transport passeng	transport infrastructur
			winter weather canada	
1	winter storm	sever weather	new hampshir new	environment assess environment
2	winter weather	winter weather	new jersei new	program legisl
3	environ canada	winter storm	new mexico new	program hunt
4	sever weather	weather warn	new hampshir new jersei new	fund program
5	weather warn	sever winter	new jersei new mexico new	environment link fund program
6	freez rain	sever weather warn	new hampshir new jersei new mexico	environment assess environment link fund
7	new brunswick	sever winter weather	new hampshir	environment link
8	heavi snowfal	new brunswick	hampshir new	environment assess environment link
9	winter weather warn	environ canada	carolina new hampshir new	assess environment
10	warn issu	cold winter	carolina new	environment assess
			campus network	
1	campu network	campu network	network network	network network
2	uw campu network	uw campu network	network uw network	network level network
3	uw campu	uw campu	network level network	network uw network
4	roger watt	network connect	uw network	network subscrib network
5	roger watt ist	level network	network uw	level network level network
6	watt ist	high speed	network subscrib network	level network
7	ip address	uw resnet	network assign network	network level
8	ip network	connect uw	network uw campu network	network assign network
9	high speed	area campu network	network level	extern network level network level network
10	request registr	switch rout	level network level network	network level network rout

The second observation is that the variants of the algorithm that use the depth feature (CorePhrase-2 and CorePhrase-2M) are consistently better than those that do not use the depth feature (CorePhrase-1 and CorePhrase-1M) in terms of both overlap and precision. This is attributed to the fact that some common phrases usually appear at the end of each document (such as "last updated", "copyright", the name of the web site maintainer). If depth information is ignored, these phrases make their way up the rank (*e.g.* the phrase "roger watt" in **campus network** cluster, which is the name of the network maintainer that appears at the end of each document.) If depth information is taken into consideration, these phrases are penalized due to their appearance at the end of the document.



(a) Average Overlap (b) Precision

Fig. 9. CorePhrase Accuracy Comparison

Another observation is that the four variants of the algorithm were able to discover the topic of the cluster and rank it in the top 10 keyphrases, which can be deduced from the maximum overlap value. CorePhrase is somewhat better than its word-weighted counterpart (CorePhrase-M) in extracting the best phrase and ranking it among the top 10, where it achieves 97% overlap on average for the best phrase. The word-weighted variant achieves 83% maximum overlap on average for the best phrase.

However, if we look at the set of the top 10 extracted phrases as a whole and not just the best phrase, the word-weighted variant achieves better performance in terms of average overlap (45% for CorePhrase-M against 40% for CorePhrase). This is attributed to the fact that keyphrases extracted by the word-weighted version will always contain heavily weighted words, which often overlap with the reference topic.This means that CorePhrase-M will consistently extract phrases containing words found in the reference topic, but which do not necessarily constitute the best descriptive keyphrases. This drawback manifests itself when there are few words which occur very frequently throughout the candidate phrases, but are not part of the reference topic. In this case the algorithm will rank up irrelevant phrases which contain those words due to their heavy weight. (An example is the **winter weather canada** cluster.)

A final observation is that CorePhrase consistently achieves better precision than CorePhrase-M (79% for CorePhrase against 67% for CorePhrase-M.) This means that CorePhrase does not only find the best keyphrase, but ranks it higher than CorePhrase-M.

To summarize these findings: (a) CorePhrase is more accurate than keyword-based algorithms; (b) using phrase depth information achieves better performance; (c) using word-weights to rank phrases usually produces a better *set* of top phrases; however, ignoring the word-weights usually produces the best descriptive phrase and ranks it higher; and (d) in most cases, CorePhrase is able to identify the reference topic in the top few keyphrases.

6 Summary and Conclusion

This chapter presented a framework for text mining based on a phrase graph model of the underlying documents. The level of document representation and manipulation is shifted to its constituent phrases rather than individual words. Phrasal analysis of documents opened the door for more accurate representation, similarity calculation, and eventually higher clustering quality. Achieving the same results using traditional vector-space methods would be impractical.

The clustering framework is comprised of four components. The first component is the DIG data structure; an efficient graph structure for representing and indexing phrases in documents. This structure is the underlying foundation upon which other components function.

The second component is the near-linear phrase matching algorithm, which is capable of generating all matching phrases between every pair of documents in

near-linear time, with arbitrary-length phrases. The matching phrases are used to construct a complete similarity matrix for use by various clustering algorithms.

The third component is an incremental clustering algorithm based on similarity histogram distribution. The algorithm maintains tight clusters incrementally by keeping the similarity distribution in each cluster coherent.

Finally, the fourth component is the CorePhrase keyphrase extraction algorithm for labeling the generated clusters with keyphrases. The algorithm accurately extracts the phrases that best describe each cluster using the DIG structure to extract the candidate keyphrases, then rank the top representative phrases.

This framework is coherent, robust, and efficient, as demonstrated by experimental results. The underlying model is flexible and could be extended or enhanced to accommodate other phrase-based tasks for text mining.

The application of the model in e-learning environments provides a way to automatically group learning resources based on content, which can be overwhelming in very large learning repositories. Text mining can help reduce the load on the learner by offering a digest of the data that is accurate enough to acquire the desired information.

References

1. Salton, G., Wong, A., Yang, C.: A vector space model for automatic indexing. Communications of the ACM **18** (1975) 613–620
2. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill computer science series. McGraw-Hill, New York (1983)
3. Salton, G.: Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison Wesley, Reading, MA (1989)
4. Wong, W., Fu, A.: Incremental document clustering for web page classification. In: 2000 International Conference on Information Society in the 21^{th} Century: Emerging Technologies and New challenges (IS2000), Japan (2000)
5. Jiang, Z., Joshi, A., Krishnapuram, R., Yi, L.: Retriever: Improving web search engine results using clustering. Technical report, CSEE Department, UMBC (2000)
6. K., S.: Reducing the space requirement of suffix trees. Software — Practice and Experience **29** (1999) 1149–1171
7. Apostolico, A.: The myriad virtues of subword trees. In Apostolico, A., Galil, Z., eds.: Combinatorial Algorithms on Words. NATO ISI Series. Springer-Verlag (1985) 85–96
8. Manber, U., Myers, G.: Suffix arrays: A new method for on-line string searches. SIAM Journal on Computing **22** (1993) 935–948
9. Caropreso, M.F., Matwin, S., Sebastiani, F.: Statistical phrases in automated text categorization. Technical Report IEI-B4-07-2000, Pisa, IT (2000)
10. Isaacs, J.D., Aslam, J.A.: Investigating measures for pairwise document similarity. Technical Report PCS-TR99-357, Dartmouth College, Computer Science, Hanover, NH (1999)
11. Lin, D.: An information-theoretic definition of similarity. In: Proc. 15^{th} International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA (1998) 296–304

12. Strehl, A., Ghosh, J., Mooney, R.: Impact of similarity measures on web-page clustering. In: Proceedings of the 17^{th} National Conference on Artificial Intelligence: Workshop of Artificial Intelligence for Web Search (AAAI 2000), Austin, TX, AAAI (2000) 58–64
13. Yang, Y., Pedersen, J.P.: A comparative study on feature selection in text categorization. In: Proceedings of the 14^{th} International Conference on Machine Learning (ICML'97), Nashville, TN (1997) 412–420
14. Jain, A.K., Dubes, R.C.: Algorithms for Clustering Data. Prentice Hall, Englewood Cliffs, N.J. (1988)
15. Steinbach, M., Karypis, G., Kumar, V.: A comparison of document clustering techniques. KDD-2000 Workshop on TextMining (2000)
16. Beil, F., Ester, M., Xu, X.: Frequent term-based text clustering. In: Proceedings of the 8^{th} International Conference on Knowledge Discovery and Data Mining (KDD 2002), Edmonton, Alberta, Canada (2002) 436–442
17. Hill, D.R.: A vector clustering technique. In Samuelson, ed.: Mechanized Information Storage, Retrieval and Dissemination. North-Holland, Amsterdam (1968)
18. Cios, K., Pedrycs, W., Swiniarski, R.: Data Mining Methods for Knowledge Discovery. Kluwer Academic Publishers, Boston (1998)
19. Dasarathy, B.V.: Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques. McGraw-Hill Computer Science Series. IEEE Computer Society Press, Las Alamitos, California (1991)
20. Lu, S.Y., Fu, K.S.: A sentence-to-sentence clustering procedure for pattern analysis. IEEE Transactions on Systems, Man, and Cybernetics **8** (1978) 381–389
21. Zamir, O., Etzioni, O., Madanim, O., Karp, R.M.: Fast and intuitive clustering of web documents. In: Proceedings of the 3^{rd} International Conference on Knowledge Discovery and Data Mining, Newport Beach, CA, AAAI (1997) 287–290
22. Hammouda, K., Kamel, M.: Document similarity using a phrase indexing graph model. Knowledge and Information Systems **6** (2004) 710–727
23. Frank, E., Paynter, G.W., Witten, I.H., Gutwin, C., Nevill-Manning, C.G.: Domain-specific keyphrase extraction. In: IJCAI. (1999) 668–673
24. Turney, P.D.: Learning algorithms for keyphrase extraction. Information Retrieval **2** (2000) 303–336