Hierarchical Taxonomies using Divisive Partitioning[*]

Daniel Boley[†]

Department of Computer Science and Engineering

University of Minnesota

200 Union Street S.E., Rm 4-192

Minneapolis, MN 55455, USA

Abstract

We propose an unsupervised divisive partitioning algorithm for document data sets which enjoys many favorable properties. In particular, the algorithm shows excellent scalability to large data collections and produces high quality clusters which are competitive with other clustering methods. The algorithm yields information on the significant and distinctive words within each cluster, and these words can be inserted into the naturally occuring hierarchical structure produced by the algorithm. The result is an automatically generated hierarchical topical taxonomy of a document set. In this paper, we show how the algorithm's cost scales up linearly with the size of the data, illustrate experimentally the quality of the clusters produced, and show how the algorithm can produce a hierarchical topical taxonomy.

Keywords: unsupervised clustering, hierarchical taxonomy, scalability, text document data mining.

[†]tel: 1-612-625-3887, fax: 1-612-625-0572, e-mail: `boley@cs.umn.edu`

1 Introduction

Automated classification and knowledge discovery within large document sets has been of interest to specialized users for many years. This is particularly true of the World Wide Web (WWW), which has seen explosive growth in recent years. The WWW also is a very dynamic environment with documents begin added and replaced very rapidly. This has lead to interest in autonomous agents to explore the WWW, such as that proposed in [12, 11, 22, 26, 34, 50, 48]. Such agents must be capable of automatically classifying and/or categorizing large datasets of documents without user intervention. A user may unleash an autonomous agent to explore the WWW using certain constraints which are more general than a given fixed query, for example based on the agent's own automatic determination of the user's interests. The agent must be capable of categorizing large numbers of documents in order to present the user with a short "summary" or "taxonomy" of the documents found. Many autonomous agents, such as WebACE [26], also extract the "significant" words in order to construct new WWW search queries to find related documents.

In this paper, we present an unsupervised clustering algorithm, called Principal Direction Divisive Partitioning (PDDP), that enjoys many desireable properties. We have found that this algorithm is

- conceptually simple and straightforward in design,

- fast and scalable,

- effective in separating documents by topic,

- yields collections of "significant" and "distinctive" words,

- builds a hierarchical document classification tree.

The purpose of this paper is to explore some of these properties: specifically its performance and its use in the construction of hierarchical taxonomies. This work grew out of a larger project involving the design of a Web agent, in which several competing algorithms have been explored [26]. The PDDP algorithm has shown itself to be competitive in both cost and quality of results as compared to the other algorithms tried in this project.

As the name suggests, the method of Principal Direction Divisive Partitioning computes a *partitioning* of a given collection of documents (i.e. all the documents are assigned to a cluster). The method operates in a top-down (*divisive*) manner starting with a single cluster encompassing the entire document collection, and clusters are split based on a linear discriminant function derived from the *principal direction* (also known as

the leading principal component). The key to its speed is a fast method to obtain the principal direction. This is not the same as Latent Semantic Indexing [3, 7, 18], as discussed below in §7.

The rest of this paper consists of a description of the basic method (§2), a presentation of performance results, both with regard to cost (§3) and with regard to the quality of the computed clusters (§4), a description of how the PDDP method provides an automatic cluster labeling scheme (§5), a description of how this labeling scheme leads to a hierarchical taxonomy (§6), a summary of related work (§7), and a conclusion (§8).

2 Algorithm Description

In this section, we give a description of the basic PDDP algorithm. As a divisive algorithm, the PDDP algorithm constructs a binary tree hierarchy of clusters. Starting with a root "cluster" encompassing the entire sample set (consisting of text documents), each unsplit cluster is split into two "child" clusters until a desired number of clusters is reached. The clusters created in this way are in a hierarchy with the form of a binary tree whose root is the initial root "cluster." At any stage, the leaf nodes of this tree are the unsplit clusters. At termination, these unsplit clusters are the final set of clusters reported by this algorithm. Each stage of the algorithm consists of (a) selecting the cluster to split, and (b) actually splitting that cluster into two child clusters.

The behavior of the algorithm is controlled by (a) the method used to select the next node to be split, and (b) the method used to accomplish the splitting process. The method used to select the next node to split can be completely independent of the rest of the algorithm, depending on the specific domain. In our examples we have used a modified `scatter` value [19], defined to be the sum of squares of the distances from each document in the cluster to the overall mean (centroid) of the cluster. The computation of the `scatter` value is very straightforward. A simpler alternative, which also tends to keep the number of documents in each "leaf" cluster approximately the same, is to set the `scatter` value to just the number of documents in the cluster. A third alternative is to keep the binary tree complete by splitting all the nodes at one level before proceeding to the next level.

Once a node is selected by whatever method the user chooses, it is split, and the method used to accomplish this is the key to the computational efficiency of the entire approach. The rest of this section is devoted to a more detailed look at this process.

2

Each document is represented by a vector $\mathbf{d}$ of word frequencies, scaled to have unit length to make the results independent of document length. This scaling is called the *norm scaling*. An alternate scaling is the TFIDF scaling [44]. In the TFIDF scaling, the vector $\mathbf{d}$ representing each document is scaled as follows:

$$\text{let } \tilde{d}_i = \frac{1 + \frac{\text{TF}_i}{\max_j(\text{TF}_j)}}{\log\left(\frac{1}{\text{DF}_i}\right)}, \quad \text{then } d_i^{\text{TFIDF}} = \frac{\tilde{d}_i}{\sqrt{\sum_j(\tilde{d}_j)^2}}, \tag{1}$$

where TF_i is the raw word count for the i-th word in the document, and DF_i is the number of different documents in which the i-th word appears in the entire dataset. With this scaling, $d_i^{\text{TFIDF}} \neq 0$ for every i in every document, resulting in a completely full term frequency matrix $\mathbf{M}$. This results in much higher storage requirements and (as will be discussed in the next section) higher computation costs, without obtaining much, if any, improvement in cluster quality, at least in all the examples we have tried.

A cluster C represents a set of documents assembled into an $n \times k$ matrix, $\mathbf{M}_\mathsf{C} = \{\mathbf{d}_1, \ldots, \mathbf{d}_k\}$, where n is the number of features and k is the number of documents in the cluster. We define a linear discriminant function $g_\mathsf{C}(\mathbf{d}) = \mathbf{u}_\mathsf{C}^T(\mathbf{d} - \mathbf{w}_\mathsf{C})$, where $\mathbf{u}_\mathsf{C}$, $\mathbf{w}_\mathsf{C}$ are vectors associated with C to be determined. The linear discriminant function is used to define the splitting of the cluster: if $g_\mathsf{C}(\mathbf{d}) \leq 0$, the document $\mathbf{d}$ is placed in the new left child, otherwise $\mathbf{d}$ is placed in the new right child. Thus the behavior of the algorithm at each node in the binary tree is determined entirely by the two vectors $\mathbf{u}_\mathsf{C}$, $\mathbf{w}_\mathsf{C}$ associated with the cluster C.

Mathematically, the vector $\mathbf{w}_\mathsf{C} = \mathbf{M}_\mathsf{C}\mathbf{e}/k$ is the *mean* or *centroid* vector for the cluster, where $\mathbf{e}$ denotes a vector of all ones of appropriate dimension. The vector $\mathbf{u}_\mathsf{C}$ is the direction of maximal variance. This direction corresponds to the largest eigenvalue of the sample covariance matrix for the cluster. The computation of $\mathbf{u}_\mathsf{C}$ is the most expensive step in this whole process. We have used a fast Lanczos-based solver for the singular values of the matrix of documents in the cluster [23]. This algorithm is able to take full advantage of the fact the matrices are extremely sparse, often with less than 4% fill, where the fill is defined as the fraction of all matrix entries that are nonzero. A simple cost analysis shows that the cost of the whole algorithm is dominated by this singular vector computation, and the cost of the singular vector computation is proportional to the matrix fill (see §3).

The overall method can then be summarized in Table 1. As the method is "divisive" in nature, splitting each cluster into exactly two pieces at each step, the result is a binary tree whose leaf nodes are the sought-after clusters. Let n, m be the number of different words and documents, respectively. Each document is represented by a column n-vector $\mathbf{d}_i$ of word frequencies, scaled to have unit length according to the norm scaling. The columns are assembled into a single $n \times m$ matrix $\mathbf{M}$. At each stage we denote the cluster

selected for splitting with the letter C. We will use $\mathbf{e} = (1, \ldots, 1)^T$ to denote a vector of all ones of appropriate dimension, and $\|\mathbf{A}\|_F \stackrel{\text{def}}{=} \sqrt{\sum_{ij} a_{ij}^2}$ to denote the Frobenius norm of a given matrix $\mathbf{A}$ [23]. Associated with the cluster C containing k documents are:

- the left and right children of C, denoted L and R, and the parent of C, denoted P;

- the set of documents within the cluster, collected in an $n \times k$ matrix $\mathbf{M}_{\text{C}}$;

- the cluster centroid vector: $\mathbf{w}_{\text{C}} \stackrel{\text{def}}{=} \frac{1}{k} \cdot \mathbf{M}_{\text{C}} \cdot \mathbf{e}$;

- The principal direction vector (also known as the leading principal component [28]), which is the eigenvector corresponding to the largest eigenvalue of the sample covariance matrix

$$\mathbf{C} = \frac{1}{k-1} \cdot \left((\mathbf{M}_{\text{C}} - \mathbf{w}_{\text{C}}\mathbf{e}^T)(\mathbf{M}_{\text{C}} - \mathbf{w}_{\text{C}}\mathbf{e}^T)^T \right) ;$$

- The scatter value: $\texttt{scatter}_{\text{C}} = \sum_{i \in \text{C}} \|\mathbf{d}_i - \mathbf{w}_{\text{C}}\|_2^2 \equiv \|\mathbf{M}_{\text{C}} - \mathbf{w}_{\text{C}}\mathbf{e}^T\|_F^2$ (alternatively, set $\texttt{scatter}_{\text{C}} = k$ or even $\texttt{scatter}_{\text{C}} =$ the distance from root);

- The discriminent vector, used to determine which document in C goes to C's left child and which to C's right child:

$$\mathbf{v}_{\text{C}} = g_{\text{C}}(\mathbf{M}_{\text{C}}) \equiv \mathbf{u}_{\text{C}}^T(\mathbf{M}_{\text{C}} - \mathbf{w}_{\text{C}}\mathbf{e}^T); \tag{2}$$

Each of these quantities must be computed for each cluster as it is generated, except that $\mathbf{u}_{\text{C}}, \mathbf{v}_{\text{C}}$ are needed only when a cluster is actually split. The vectors $\mathbf{u}, \mathbf{v}$ are actually computed as the left and right singular vectors, respectively of the matrix $\mathbf{A} = \mathbf{M}_{\text{C}} - \mathbf{w}_{\text{C}}\mathbf{e}^T$ [23]. These two quantities also happen to be the most expensive to compute, and we discuss their computation in §3.

3   Algorithm Performance: Scalability

The key to the fast computation of the PDDP algorithm is a fast method to obtain the leading left and right singular vectors of the matrix $\mathbf{A} = \mathbf{M}_{\text{C}} - \mathbf{w}_{\text{C}}\mathbf{e}^T$, needed in step 4 of Table 1. Since this step is critical to the efficiency of the overall algorithm we sketch the main ideas behind the method we use. The computation of the scatter value used in step 3 is the *second* most expensive piece, but it is basically a norm computation which can be computed in time that is linear in the number of nonzero entries. The computation of the singular vectors for an $n \times m$ matrix, however, can easily take $O(m^3 + m^2 n)$ if a naive dense singular value

Start with $n \times m$ matrix $\mathbf{M}$ of (scaled) document vectors, and a desired number of clusters $c_{\max}$.   1. **Initialize** Binary Tree with a single Root Node.   2. **For** $c = 2, 3, \ldots, c_{\max}$ **do**   3.   **Select** leaf node C with largest `scatter` value, and L & R := left & right children of C.   4.   **Compute** $\mathbf{v}_{\mathtt{C}} = g(\mathbf{M}_{\mathtt{C}}) \equiv \mathbf{u}_{\mathtt{C}}^T(\mathbf{M}_{\mathtt{C}} - \mathbf{w}_{\mathtt{C}}\mathbf{e}^T)$ 5.   **For** $i \in \mathtt{C}$, if $v_i \leq 0$, assign document i to L, else assign it to R.   6. **Result:** A binary tree with $c_{\max}$ leaf nodes forming a partitioning of the entire data set.

Table 1: PDDP Algorithm Summary

solver is used [23]. Therefore, we take advantage of the fact that the matrix $\mathbf{M}$ in these applications tends to be extremely sparse. In our examples, the fill (fraction of all entries which are nonzero) ranged from a high of 4% to as low as 0.68%. The net result is that the matrix vector product involving the matrix $\mathbf{M}$ can be carried out extremely fast, in time proportional to the fill. We also take advantage of the fact that we seek only the leading singular value and associated left and right singular vectors.

Given the above considerations, we use a singular value solver in which the matrix enters only in the form of matrix vector products, namely the Lanczos method [23]. For the purposes of this problem, it suffices to adapt a Lanczos method for the symmetric eigenvalue problem (such as for the covariance matrix $\mathbf{A}$), since high accuracy is not required. These methods have become well established for symmetric eigenvalue problems, especially when the leading eigenvalue and associated eigenvector are sought [23]. Therefore, we will not derive this algorithm in detail. However we will make some remarks about this algorithm in this application in order to be able to say something about its cost. This algorithm is an iterative procedure in which the main step within each iteration is the matrix-vector product of the form $\mathbf{A} \cdot \mathbf{v}$, where $\mathbf{A}$ is a very sparse matrix. It is well known that the cost of this step is proportional to the number of nonzeros present in $\mathbf{A}$ [23]. The only other critical factor affecting costs is the number of iterations of the Lanczos algorithm (i.e. the number of matrix-vector products). In our Web document domain, this algorithm has never taken more than 20 iterations for each computation, due to the favorable distribution of the eigenvalues, regardless of the overall dimensionality of the document set being partitioned. Hence the overall algorithm behaves as if the number of Lanczos iterations were a constant, and the overall cost is linear in the cost of each individual matrix-vector product.

The cost of all the remaining steps in the PDDP algorithm also depends on the number of nonzeros, so

the net result is that the total cost of each pass of the PDDP algorithm, though dominated by the singular value computation, is still linearly proportional to the number of nonzero entries, not the dimensionality of the input data. This behavior is illustrated in Fig. 1 and is analysed at greater length in [10]. In data sets where the number of distinct words appearing in each document remains relatively invariant across the entire corpus, the number of nonzeroes, and hence the total cost of the algorithm, will scale linearly with the number of documents.

To test the scalability of the PDDP method, we have run the PDDP method on a two different datasets. The first dataset, the 'J' dataset, had modest size containing 185 documents and 10536 words consisting of the results of a World Wide Web search query using a standard search engine (Altavista). The second dataset, the 'K' dataset, was a relatively large data set with 2340 documents and 21839 words, consisting of news articles obtained from the Reuters news service via the Web in October, 1997. The documents were processed by eliminating stop words and HTML tags, stemming the remaining words using Porter's suffix-stripping algorithm [39] as implemented by [21]. etc., and then counting up the occurences of the remaining words in every document. A vector of word counts was constructed for each document, scaled to have unit length (the norm scaling), and assembled into the term frequency matrix $\mathbf{M}$. We also constructed other experiments by pruning in advance the "insignificant" words according to various a-priori strategies.

Using an improved implementation of the PDDP method, which we remark is still implemented in the MATLAB interpreter, we obtained 16 clusters in under 40 seconds (see Table 2 & Fig. 1). In Table 2 we compare the times for the divisive PDDP algorithm with the hierarchical agglomerative clustering algorithm (HAC) [19], which tends to be much more expensive [17]. We also compared this algorithm with AutoClass [13], but this turned out to take even more time than the HAC on every example in the 'J' series of experiments. The TFIDF scaling increases the fill from under 4% to 100%, yielding a proportional increase in the cost of the PDDP procedure; i.e. the PDDP procedure ran at least 25 times slower.

Fig. 1 shows how the PDDP running time depends on the number of nonzeroes in the term frequency matrix, especially for the 'K5' case with many fewer words but a much more dense matrix. We will notice in the next section that PDDP produces the best partitionings when the word lists are *not* pruned a-priori, unlike the practice common among many other proposed algorithms, and the resulting cost increase is modest. In some cases (e.g. J5, K5), pruning the word list actually fills the matrix with nonzeroes, negating the advantange of the small word list. This behavior has been consistent without exception on every data set attempted.
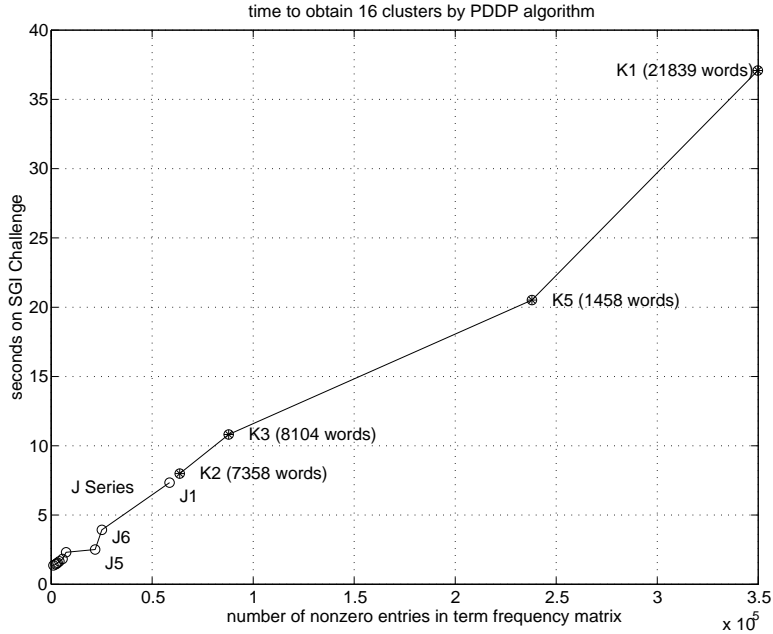
Figure 1: Times for the PDDP algorithm on an SGI versus number of nonzeros in term frequency matrix, for two different series of experiments. The datasets are listed in Table 2.

experiment	word count	PDDP entropy	time (sec) PDDP	time (sec) HAC	description of experiment
J1	10536	0.68951	7.3	−	all words (except stop words)
J2	946	1.12024	1.5	469	top words accounting for 25% of all words
J3	1763	0.85052	1.8	1014	top 20 words in each document
J4	2951	1.0988	2.3	1763	Top 5+ words + HTML-tagged words
J5	449	0.73458	2.5	218	frequent item sets from ARHP
J6	5106	0.83093	3.9	3195	all words with TF > 1
J7	1328	0.89665	1.7	695	Top 20+ with TF > 1
J8	1105	0.95650	1.6	691	Top 15+ with TF > 1
J9	805	1.07132	1.4	379	Top 10+ with TF > 1
J10	474	1.16486	1.4	231	Top 5+ with TF > 1
K1	21839	0.45802	37.1	−	all words (except stop words)
K2	7358	0.81170	8.0	−	top words accounting for 25% of all words
K3	8104	0.85794	10.8	−	top 20 words in each document
K5	1458	0.47155	20.5	−	frequent item sets from ARHP

Table 2: Sample results with 16 clusters. The J datasets have 185 documents, the K datasets have 2,340 documents. PDDP tends to produce better clusters when all words except stop words are kept in the dataset.

(A) affirmative action	(L) employee rights
(B) business capital	(M) materials processing
(C) information systems	(P) personnel management
(E) electronic commerce	(S) manufacturing systems integration
(I) intellectual property	(Z) industrial partnership

Table 3: Category lables used for entropy calculations.

4   Algorithm Performance: Quality of Computed Clusters

In addition to the speed of the method, we have measured also the quality of the resulting clusters, compared to other methods. In order to measure and compare the performance of clustering algorithms, we hand labeled each document into one of 10 topical categories shown in Table 3. These classifications were not given to the clustering algorithms at all, but used only to measure the quality of the resulting clusters. In the rightmost column of Fig. 5 we illustrate the performance of the PDDP algorithm by showing the labels of the documents in each of 25 clusters using the 'J1' dataset. However, in order to compare the performance of different algorithms in a quantitative manner, we used these labels to to compute an entropy [40] measure on the resulting partitions. The entropy of a cluster j is defined by

$$e_j = - \sum_i \left(\frac{c(i,j)}{\sum_i c(i,j)} \right) \cdot \log \left(\frac{c(i,j)}{\sum_i c(i,j)} \right),$$

where $c(i,j)$ is the number of times label i occurs in cluster j. The entropy for a cluster is zero if the labels of all the documents are the same, otherwise it is positive. The total entropy is the weighted average of the individual cluster entropies:

$$e_{\text{total}} = \frac{1}{m} \sum_j e_j \cdot (\text{number of documents in cluster } j).$$

As a consequence, the lower the entropy the better the quality, subject to the proviso that the manual assignment of the labels in Table 3 is imperfect. Indeed some documents were assigned multiple labels, and the entropy calculations were based on just one of those labels chosen arbitrarily. We compared the PDDP method to two other methods on the "J1" data set consisting of 185 documents and 10536 words. The resulting entropies are shown in Table 2 for 16 clusters and illustrated in Fig. 2 for 32 cluster partitions on the 'J' series of the experiments.

We remark that in the HAC we used the TFIDF scaling, as it is generally recommended in the literature.
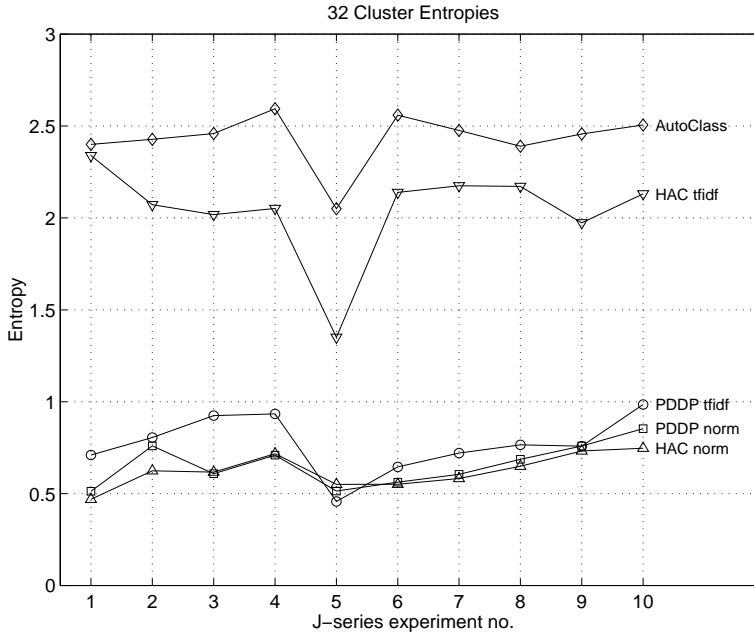
Figure 2: Comparative entropies for different methods applied to the 'J' series of experiments with 32 clusters.

In Fig. 2 we show how this scaling affects both the HAC and PDDP methods. The performance of the HAC method appears to be very sensitive to the choice of scaling, with the norm scaling yielding clusters of competitive quality according to our entropy measure. The performance of the PDDP algorithm appears to be less sensitive to the choice of scaling used, though as we remarked in the previous section, the TFIDF scaling substantially increases the cost of the PDDP algorithm.

Overall, we can conclude that both the PDDP and HAC algorithms are capable of yielding competitive clusters, as shown in Fig. 2, but the PDDP method with norm scaling is faster by at least an order of magnitude compared to HAC, as illustrated in Table 2.

We have also had a few comparisons with a recently developed association rule hypergraph clustering method (ARHP) [24, 25, 31], based on association rule hypergraphs [2, 6], but this last method is harder to compare since it eliminates some documents so that the result is not strictly speaking a partitioning of the entire document set. The resulting entropies for 33 clusters on the 'J1' document set were

$$\text{PDDP: } 0.497, \quad \text{ARHP: } 0.506, \quad \text{HAC: } 2.339, \quad \text{AutoClass: } 2.400, \tag{3}$$

where 5 documents were not clusters by the AHRP method and grouped into their own separate cluster. The entropy measure has the unfortunate property that it naturally decreases as the number of clusters increases,

9

reaching a perfect score of zero with m clusters (one document per cluster). This makes it difficult to compare results with differing numbers of clusters. However, in the experiments we have tried, the PDDP method has remained competive with the ARHP method as well as with HAC with norm scaling, and consistently better than the other more classical methods.

5   Extracting Significant Words

A desireable goal is to extract the most significant words within a cluster of documents. By "most significant words" we mean the words that are most distinctive to the cluster, or the words that distinguish the cluster from its neighboring clusters. These words should provide good indicators of the topics associated with the documents within the cluster. As part of its normal operation, the PDDP algorithm provides information on such words for each cluster.

The PDDP algorithm computes two vectors for each cluster that can be used to extract significant words. One of these vectors is the centroid vector. The largest entries in the centroid vector correspond to words appearing the most often through out the cluster C, ignoring whether or not the words also appear very often throughout the whole corpus. Hence it is possible that the leading centroid words may not be distinctive.

We show how another vector computed by the PDDP algorithm often gives much more meaningful information about the significant words in a cluster than the centroid vector. For a given cluster we examine the principal direction (computed as the left singular vector $\mathbf{u}_P$) associated with the cluster's *parent* in the computed PDDP tree. To motivate this choice, consider a cluster C and a document within C represented by a vector $\mathbf{d}$ of word counts. Let P denote the *parent* cluster for C, and assume without loss of generality that C is the *right child* of P. Let $\mathbf{w}_C$ denote the centroid vector for C, and $\mathbf{w}_P$, $\mathbf{u}_P$ denote the centroid and principal direction vector for P. Recall how $\mathbf{d}$ is placed in C. To determine whether $\mathbf{d}$ belongs in the left or right child of P, the value (2)

$$g_C(\mathbf{d}) = v = \sum_{i=1}^{n} u_i(d_i - w_i) \tag{4}$$

is computed, where d_i is the term frequency for the i-th word in the document, w_i is the average term frequency for the i-th word in cluster P, and u_i is the component of the principal direction corresponding to the i=th word. The document $\mathbf{d}$ has been placed in C because $v > 0$, otherwise it would have been placed in P's left child instead (i.e. C's sibling). Therefore the positive terms in the sum (4) must outweigh the negative terms.

We claim that the values u_i, $i = 1, \ldots, n$, are an indicator of the significance of each word for the particular cluster C. We provide a heuristic argument, since the behavior depends critically on the nature of the underlying dataset. We ask the question: why has document **d** been placed in C as opposed to C's sibling, or in other words, why is $v > 0$. The answer is to be found by examining the individual terms in the sum (4) in which the positive terms outweigh the negative terms. For each i, u_i is either positive, negative, or negligibly small. If $u_i \gg 0$, the factor $(d_i - w_i)$ is most likely greater than zero (or at least not very negative), meaning that the frequency of the i-th word in document **d** is greater than average. If $u_i \ll 0$, the factor $(d_i - w_i)$ is most likely less than or close to zero, meaning that the frequency of the i-th word in document **d** is less than average. If u_i is small, then the shifted term frequency $(d_i - w_i)$ is less important in the placement of **d** into C.

On the other hand, if **d** had been placed in C's sibling, then v would have been negative, meaning that d_i would have been greater than average for those i for which $u_i \ll 0$. Hence we can conclude that the words corresponding to large positive values u_i are most significant for C, and the words corresponding to large negative values u_i are most significant for C's sibling.

We recall that the vector $\mathbf{u}_\mathsf{P}$ itself is computed to maximize the sum of squares of the values v over all the documents in P − this is the definition of the singular vector corresponding to the leading singular value. Therefore, in most cases, the terms within the sum (4) will not all be negligible.

Figure 4 compares the performance resulting from the principal direction vector to that resulting from the centroid vector in a cluster obtained using a corpus of 494 health related news articles obtained from the World Wide Web. The leading terms in the centroid vector consist of words common to most documents throughout the corpus (many appear in a title bar on the individual web pages). The leading terms in the principal direction vector consist of terms that are significant and distinctive to at least some of the documents within the cluster. Figure 3 illustrates why this comes about. In Figure 3 we show the documents within each cluster, corresponding to each column of the matrix. The first 10 rows are the leading words from the principal direction vector $\mathbf{u}_\mathsf{P}$, and the last 10 rows are the leading words of C's centroid vector $\mathbf{w}_\mathsf{C}$, both sorted in decreasing order of "importance." Here "importance" is measured by the magnitude of the corresponding vector entry. It is seen that because the distinctive words do not each appear in more than a few documents, the centroid vector is dominated by the words appearing "across the board" which do not distinguish the cluster. The first word (dna) in $\mathbf{w}_\mathsf{C}$ that is distinctive for this cluster does not appear within the centroid $\mathbf{w}_\mathsf{C}$ until position 12.

• 1. Enzyme Triggers Emphysema in Smokers	• 2. Pregnancy Factor Protects Fetus
• 3. Drug May Offer New Way to Beat Colds	• 4. New Doping Agent for Athletes Reported
• 5. Lens Coat Cuts Cataract Cost	• 6. Dengue Fever Strikes "Dr. Quinn"
• 7. Gold Probe Detects DNA	• 8. Gold Probe Detects DNA
• 9. 'Dry Brushing' Beats Dental Plaque	•10. FDA Moves to Ban Laxative Ingredient
•11. Dengue Fever Strikes "Dr. Quinn"	•12. Muscles Adapt to Exercise Lifestyle

<div align="center">The 12 Document Headlines.</div>

vector	leading words
centroid	help entertai health new polit sport tech bize index
principal dir.	dengue fever dna gene gold probe diseas gubler guttman

Table 4: Sample cluster from the "K1" document set, showing the document headlines and the leading words extracted from the centroid and principal direction vectors. Notice the principal direction words are better indicators of the document topics, compared to the centroid words. Some articles were repeated on later dates with slightly different contents. The document numbers correspond to the columns of Fig. 3.

We interpret the possible match between the leading words from the principal direction and the centroid as follows. It turns out that in most clusters the leading words obtained from the principal direction tend to agree with those obtained from the centroid vector (see e.g. Fig. 5). This indicates that in most such clusters, the most common words within the cluster are also the most distinctive. Such clusters are more homogeneous, with the distinctive words relevant to most documents within the cluster. A mismatch between the two sets of leading words usually indicates that, though the principal direction words are distinctive to the cluster, they may not apply to many documents within the cluster, leading to the conclusion that the cluster is less homogeneous.

6   Hierarchical Taxonomy

The PDDP algorithm constructs a tree providing a hierarchical structure to the entire document set. In order to provide a hierarchical taxonomy, one must not only provide the hierarchy, but one must also provide the taxonomy in the form of informative labels for each substructure. In our case here, we propose to use the PDDP tree to provide the hierarchy and the principal direction vectors as in §5 to provide the taxonomy. If we followed the binary tree structure in a simplistic way, we would have a very tall hierarchy with very few substructures at each level (i.e. branching factor of 2). For a usable taxonomy, it would be more meaningful
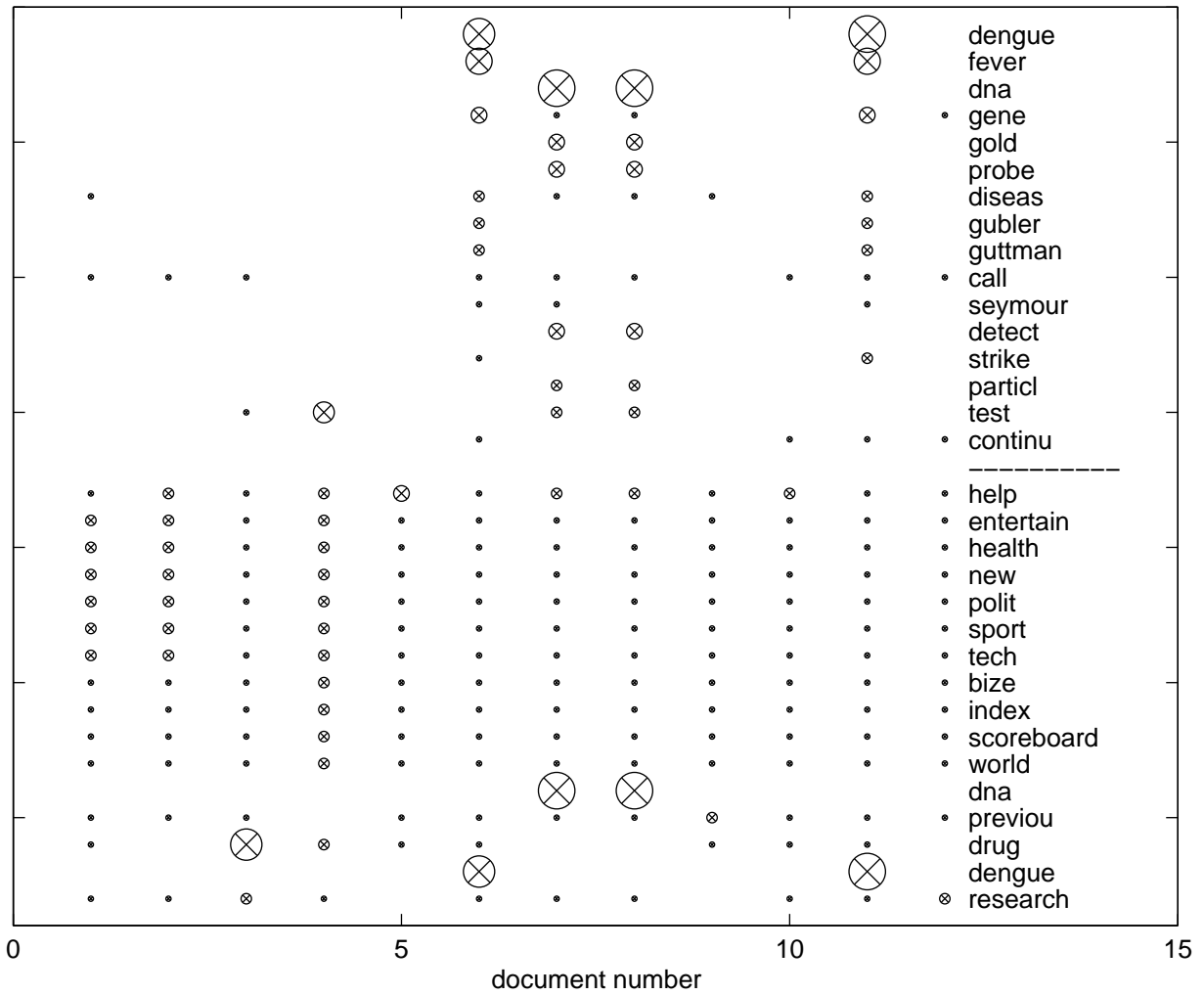
Figure 3: Word frequencies for the 12 documents shown in Fig. 4 for the leading words in the principal direction vector (top 10 rows) and in the centroid (bottom 10 rows), both in decreasing order of importance. Importance is measured in terms of the magnitude of the corresponding matrix entry. The size of the marker ⊗ represents the magnitude of the matrix entry.

to use a branching factor greater than 2. In order to accomplish this with the PDDP algorithm, we can follow the following approach.

Let b be the desired branching factor. Use the PDDP algorithm to create b clusters. For each cluster we have a set of leading significant words obtained from its parent's principal direction vector. Instead of treating the entire corpus as a unit, we now apply the PDDP algorithm separately to each of the b individual clusters, creating b subclusters for each. This process can be repeated on each of the b^2 subclusters to as many levels as desired. The result will be a hierarchical structure with a branching factor of b. For each subcluster at every level, we have a corresponding principal direction vector yielding the significant words. Following the discussion in the previous section, these significant words are not those appearing very often within the entire corpus or even within a wide subset documents, but are those that distinguish each document subcluster from the other subclusters. Hence these words will serve to highlight what is distinctive about each subcluster. They are not intended as a way to categorize the entire corpus within a global category schema.

We illustrate the performance of this technique in Fig. 5 using a braching factor $b = 5$. We have extracted the three leading words based on the principal direction vector. The three words are used to form an automatically generated topic label for each cluster. This was also repeated using the centroid vector. In this example, once the corpus and the numerical branching factor were chosen, the entire process described above was carried out completely automatically to yield the indicated results.

The performance shown in Fig. 5 appears to be typical over the several different datasets we have tried. In particular, in many of the clusters there is a high degree of agreement between the words selected by the centroid vector and the words selected by the principal direction vector. However, the centroid vector measures how common a word is within a cluster, and a cluster can easily appear in which the centroid vector is dominated by undistinctive words common to all the documents within an outer cluster. An example are the clusters shown in Fig. 5 lines 4–6, in which the two most common words within each cluster, `affirm` & `action`, are also the two most common words throughout the entire upper-level cluster in line 1. This effect was discussed in the previous section. In these cases, the principal direction vector still avoids the common words, but picks out the words that distinguish each subcluster from another. A heirarchical taxonomy is more useful if the words in the lower-level labels are not merely repeats of those in the upper-level labels. For example, in lines 4–6 in Fig. 5, repeating the words `affirm` and `action` (as done in the centroid labels in the middle column) conveys no additional information about the contents of the documents in those subclusters,

whereas the principal direction vector (left column) is able to pick out other significant and distinctive words.

The result is that many labels generated from the principal direction vectors will be "more unique" with less repetition of the same common words. The labels of subclusters will tend to involve new words giving more information about their contents. It is easier to assign a unique topic label to each cluster at each level with less topic overlap.

7 Related Work

Existing approaches to document clustering are generally based on either probabilistic methods, or distance and similarity measures (see [22]). Distance-based methods such as k-means analysis, hierarchical clustering [30] and nearest-neighbor clustering [33] use a selected set of words (features) appearing in different documents as the dimensions. Each such feature vector, representing a document, can be viewed as a point in this multi-dimensional space. Most of these methods tend to be agglomerative (bottom-up) in contrast to our divisive (top-down) method.

Agglomerative methods can suffer from several difficulties. The cost of standard agglomerative nearest-neighbor methods can easily reach $O(m^2 n)$, where m, n are the number of documents and words, respectively. A modified version of an agglomerative approach was used in [17] as part of a document classification and exploration process. In their approach the entire set was partitioned into smaller subsets on each of which the agglomerative algorithm was applied. The resulting clusters could then be further agglomerated together at a cost much less than that of agglomerating all the individual documents at once. The result is a logarithmic cost. However, as with all agglomerative algorithms, the resulting clusters depend on the order in which the individual documents are presented, and varying the order can affect the performance of the overall process.

The iterative approach is a variation of a hierarchical clustering method in which a seed hierarchy (tree) is updated incrementally by adjusting the clusters based on a "quality" measure. In [20], they describe a novel method for updating a whole subtree of the hierarchy which would be applicable to our top-down tree as well as to their bottom-up tree. They also propose a category index (similar in form to the Gini Index [36, 49]) as a measure of cluster "quality," which could be adapted in the PDDP algorithm as a substitute for our `scatter` value. Many researchers have studied this approach either as a stand-alone method or as a way to refine an initial hierarchy constructed by another method [19, 35, 9, 8]. As a refining method, it would be applicable to the PDDP algorithm.

```
     PRINCIPAL DIRECTION TAXONOMY      :CENTROID VECTOR TAXONOMY          :HAND LABELS
 1. affirm.action.minor              :affirm.action.employ             :
 2.     applic.minor.percent         :    applic.minor.action          :AAAAP
 3.     employ.employe.innsmouth     :    employ.employe.action        :AALL
 4.     raza.public.preferenti       :    affirm.action.public         :AAA
 5.     action.affirm.social         :    affirm.action.people         :AAAAAAA
 6.     black.white.women            :    affirm.action.black          :AAAA
 7. system.manufactur.engin          :system.technologi.manufactur     :
 8.     heate.process.materi         :    materi.process.engin         :MMMMMMMMMMM
 9.     system.model.integr          :    system.manufactur.inform     :CCSSSSSSSSSSSM
10.     technologi.industri.program  :    technologi.manufactur.develop:ZZZZSSSSM
11.     research.system.design       :    research.system.design       :CCCZSSM
12.     industri.partner.partnership :    industri.engin.comput        :ZZZZZ
13. busi.internet.electron           :busi.inform.industri             :
14.     inform.system.manag          :    inform.system.manag          :CCCCCCCCCCCCS
15.     busi.loan.capit              :    busi.capit.develop           :BBBBBBBBBBZ
16.     industri.partnership.inform  :    industri.partnership.inform  :CZZZZZZZM
17.     electron.busi.phillip        :    electron.busi.commerc        :EEEEEEEE
18.     oracl.server.applic          :    server.internet.web          :EEEEEEEC
19. personnel.manag.employe          :personnel.employe.manag          :
20.     busi.capit.financi           :    busi.capit.financ           :BBBBBBBBBPM
21.     personnel.manag.servic       :    personnel.manag.servic       :PPPPPPPPPPPP
22.     public.comment.certif        :    public.manag.personnel       :EPPPPZ
23.     employ.retir.public          :    employe.employ.right         :LLLLLLLLP
24.     union.disciplin.labor        :    union.employe.labor          :LLLLL
25. patent.intellectu.properti       :patent.copyright.intellectu      :
26.     web.internet.commerc         :    web.internet.commerc         :EEC
27.     editor.ip.intellectu         :    right.advertis.administr     :IIIIIELM
28.     copyright.copi.softwar       :    copyright.copi.softwar       :II
29.     intellectu.properti.inform   :    intellectu.properti.patent   :IIIIII
30.     patent.applic.fil            :    patent.applic.properti       :IIIIII
```

Table 5: Three-word cluster labels for the 'J1' dataset using a branching factor of 3. The left column labels were created using the principal direction data, and the middle using centroid vector data. The right column shows the document labels assigned by hand for each individual document based on the categories in Table 3 with one letter per document.

AutoClass [13] is a method using Bayesian analysis based on the probabilistic mixture modeling [47]. Given a data set it finds maximum parameter values for a specific probability distribution functions of the clusters. In our experiments, we have found that the performance of this algorithm suffers when applied to very large datasets as in our application. The cost was on the same order of magnitude as an agglomerative method, and the quality of the resulting clusters was not competitive with the other methods (see §3).

Probabilistic methods such as Bayesian classification used in AutoClass [13] do not perform well when the size of the feature space is much larger than the size of the sample set and the features are not statistically independent, as is typical of document categorization applications on the Web. It is possible to reduce the dimensionality by selecting only frequent words from each document, or to use some other method to extract the salient features of each document. However, the number of features collected using these methods still tends to be very large, and determining which features can be discarded without affecting the quality of clusters is difficult.

The PDDP algorithm is based on the representation of the data in a multi-dimensional space. But unlike most methods using such representation, the PDDP algorithm does not depend so critically on a distance measure. In fact, the use of a distance measure is limited to the selection of the next cluster to split, and this choice could be made by any user-supplied method. Techniques such as TFIDF [44] have been proposed precisely to deal with several issues, including the presence of words appearing in all the documents. With the simple norm scaling used for the PDDP algorithm, the leading singular vectors (principal directions) automatically pick out the most discriminating words (the words most critical in separating one set of documents from another in a split) without further scaling. In fact, preliminary experiments have indicated that TFIDF, when used with the PDDP algorithm, did not improve the results, but did substantially increase the cost.

Distance-based schemes generally require the calculation of the mean of document clusters. If the dimensionality is high, then the calculated mean values may not differ significantly from one cluster to the next, especially if the initial set of clusters are poor. Hence the distances between the cluster means may not provide a good measure of separation between clusters. This issue must be addressed in any agglomerative algorithm, but is much less critical in a divisive algorithm.

The PDDP method differs from that of Latent Semantic Indexing (LSI) [3, 7, 18] in many ways. Latent Semantic Indexing (LSI) was originally proposed for a different purpose, namely as a method to aid query-based document retrieval. LSI is useful when noise is present in data sets of very high dimensionality.

LSI first reduces the dimensionality by orthogonal projection. A low rank (say, rank $k << \min\{m, n\}$) approximation to the document matrix $\mathbf{M}$ is computed using the leading k singular values and vectors of the matrix, where k can easily be on the order of 10, where k can easily be on the order of 100 or more. This has the effect of removing the noise, while representing each document by means of a set of k "generalized features." But unlike the PDDP algorithm, the SVD is applied to $\mathbf{M}$, not $\mathbf{A} = (\mathbf{M} - \mathbf{we}^T)$, and is used only to preprocess the data to reduce the dimensionality of its representation. In addition, computing 100 leading singular values and vectors is considerably more complicated and expensive than computing just one as in the PDDP Algorithm. It is well known from the Kaniel-Paige theory that the largest eigenvalue or singular value is among the first to converge in the Lanczos method [23], and also fewer temporary Lanczos vectors must be generated saving much memory as well. In addition, though the PDDP method repeats this singular value computation at every step, it is applied to the entire dataset only once. Thereafter, the singular value computation is applied only to subclusters, resulting in a much lower cost.

Another method related to LSI is that of Linear Least Squares Fit (LLSF) [15]. This method is also intended as an aid in query-based document retrieval and is also based on the use of the singular value decomposition, but it adds an extra layer with a list of concepts between the search words and the documents, enhancing the quality of query search results.

The formula (2) is an example of a linear discriminant function. Linear discriminant functions have been used extensively to partition samples in a test set into two classes. An example is the Fisher linear discriminant function (see e.g. [37]), typically used with a training set of samples with known "correct" classifications. As such it is usually used as a tool in "supervised learning," in which a training set with previously known class designations are used. An "optimal" direction is chosen on which to project all the samples, and a cut-point is selected using a one-dimensional Bayes rule based on the known mean and covariance matrices for the individual classes. Generally, all these methods operate under the assumption that the individual classes can be approximated by well separated, convex "clouds" in the high-dimensional feature space, or at a minimum that each class occupies a connected region. As an unsupervised clustering method, the PDDP algorithm is not affected by such assumptions, but will simply deposit each sample into different clusters.

Previous algorithms for unsupervised clustering based on the use of *one-dimensional projection* Bayesian analysis or linear discriminants is very limited, at least to the knowledge of this author. A hint in this direction appears on [37, p500], where the repeated use of a Fisher-style linear discriminant is suggested,

resulting in a hierarchical classification. It is even suggested that the Karhunen Loeve transformation might lead to good directions. However no algorithm is given and no discussion of the specific structure that might result from such an algorithm is discussed. The PDDP algorithm itself can be thought of as a limited use of the Karhunen Loeve transformation, though we are extracting only one leading principal direction at each stage, and we are not using the principal directions as a basis to represent the original sample feature vectors, but only to define hyperplanes in the feature space.

A motivation of this work is its application in a Web agent [26]. A number of Web agents use various information retrieval techniques and characteristics of open hypertext Web documents to automatically retrieve, filter, and categorize these documents [12, 11, 22, 34, 50, 48]. For example, HyPursuit [48] uses semantic information embedded in link structures as well as document content to create cluster hierarchies of hypertext documents and structure an information space. Pattern recognition methods and word clustering using the Hartigan's K-means partitional clustering algorithm are used in [50] to discover salient HTML document features (words) that can be used in finding similar HTML documents on the Web. Some other Web agents designed to aid the user navigate through the Web have been proposed in [42] using a graph distance matrix, in [14, 29] using a separate layer of thesaurus terms or predefined categories. In [43], they extract the context of the words to refine the matches and use a nearest neighbor-like clustering approach. The broad category of statistical approaches are compared to knowledge-based or semantic approaches in [16].

Many engines have been proposed that attempt to record documents of interest to the user. The number of times a user visits a document and the total amount of time a user spends viewing a document are just a few methods for determining user interest [1, 4, 5]. Syskill & Webert [38, 1] represents an HTML page with a Boolean feature vector, and then uses simple Bayesian classification to find Web pages that are similar, but for only a given single user profile. Also, Balabanovic [5] presents a system that uses a single well-defined profile to find similar Web documents for a user. Candidate Web pages are located using best-first search, comparing their word vectors against a user profile vector, and returning the highest -scoring pages. A TFIDF scheme [44] is used to calculate the word weights, normalized for document length. The system needs to keep a large dictionary and is limited to one user.

The Web agent "WebMate" proposed in [45] also uses TFIDF scaling, and is based on an incremental nearest neighbor approach [46] in which each new document vector is appended to a list of vectors representing topic centers, and then the two closest vectors are agglomerated. A "trigger word model" is used to form

word groups with which search queries may be refined.

The Kohonen Self-Organizing Feature Map [27, 32] is a neural-network-based Web agent that projects high dimensional input data into a feature map of a smaller dimension such that the proximity relationships among input data are preserved. On data sets of very large dimensionality such as those discussed here, the word set was preprocessed by removing stop words and infrequent words and then by using a self-organizing semantic map [41] to coalesce the remaining words down to a few hundred word clusters before the self-organizing algorithm was applied.

8   Conclusions

In this paper we have described an unsupervised divisive partitioning algorithm which exhibits the property of scalability while yielding good quality clusters. The key to its speed is the method used to compute the principal direction, and the key to the quality of the clusters is the simplicity of the overall approach. By taking advantage of all the vectors computed during the course of the algorithm, we have shown a possible way to gather information about the topics associated with the documents within each cluster by extracting "significant" words. The hierarchical nature of the data structure generated by the method naturally leads to a hierarchical classification or taxonomy of the given document set.

In order to make a practical algorithm that can be incorporated into an autonomous web agent, it is necessary to automate the decisions regarding how many clusters to compute, how many leading words to extract from each cluster, and how to update the tree as new documents arrive. Some of these aspects were addressed in [10], but they deserve further study. Also the effect of different scalings and the algorithm behavior on more homogeneous datasets are still open questions.

References

[1] M. Ackerman, D. Billsus, S. Gaffney, S. Hettich, G. Khoo, D. Kim, R. Klefstad, C. Lowe, A. Ludeman, J. Muramatsu, K. Omori, M. Pazzani, D. Semler, B. Starr, and P. Yap. Learning probabilistic user profiles. *AI Magazine*, 18(2):47–56, 1997.

[2] A. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo. Fast discovery of association rules. In U. Fayyad, G. Piatetsky-Shapiro, P. Smith, and R. Uthurusamy, editors, *Advances in Knowledge*

Discovery and Data Mining, pages 307–328. AAAI/MIT Press, 1996.

[3] T. Anderson. On estimation of parameters in latent structure analysis. *Psychometrika*, 19:1–10, 1954.

[4] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. WebWatcher: A learning apprentice for the World Wide Web. In *Proc. AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*. AAAI Press, 1995.

[5] M. Balabanovic, Y. Shoham, and Y. Yun. An adaptive agent for automated Web browsing. *Journal of Visual Communication and Image Representation*, 6(4), 1995.

[6] C. Berge. *Graphs and Hypergraphs*. American Elsevier, 1976.

[7] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37:573–595, 1995.

[8] G. Biswas, L. Weinberg, and C. Li. ITERATE: A conceptual clustering method for knowledge discovery in databases. In B. Braunschweig and R. Day, editors, *Innovative Applications of Artificial Intelligence in the Oil and Gas Industry*, 1994.

[9] G. Biswas, L. Weinberg, Q. Yang, and G. Koller. Conceptual clustering and exploratory data analysis. In *Proc. 8th Int'l. Machine Learning Workshop*, pages 591–595, 1991.

[10] D. Boley. Principal Direction Divisive Partitioning. Technical Report TR-97-056, Department of Computer Science, University of Minnesota, Minneapolis, 1997.

[11] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the Web. In *Proc. of 6th International World Wide Web Conference*, 1997.

[12] C. Chang and C. Hsu. Customizable multi-engine search tool with clustering. In *Proc. of 6th International World Wide Web Conference*, 1997.

[13] P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In U. Fayyad, G. Piatetsky-Shapiro, P. Smith, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 153–180. AAAI/MIT Press, 1996.

[14] H. Chen, K. Lynch, K. Basu, and T. D. Ng. Generating, integrating, and activating thesauri for concept-based document retrieval. *IEEE Expert*, 8(2):25–34, April 1993.

21

[15] C. Chute and Y. Yang. An overview of statistical methods for the classification and retrieval of patient events. *Meth. Inform. Med.*, 34:104–110, 1995.

[16] W. Croft. Knowledge-based and statistical approaches to text retrieval. *IEEE Expert*, 8(2):8–12, April 1993.

[17] D. Cutting, D. Karger, J. Pedersen, and J. Tukey. Scatter/gather: a cluster-based approach to browsing large document collections. In *15th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'92)*, pages 318–329, 1992.

[18] S. Deerwester, S. Dumais, G. Furnas, L. T.K., and R. Harshman. Indexing by latent semantic analysis. *J. Amer. Soc. Inform. Sci.*, 41:41, 1990.

[19] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, 1973.

[20] D. Fisher. Iterative optimization and simplification of hierarchical clusterings. *J. Art. Intell. Res.*, 4:147–179, 1996.

[21] W. B. Frakes. Stemming algorithms. In W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval Data Structures and Algorithms*, pages 131–160. Prentice Hall, 1992.

[22] W. B. Frakes and R. Baeza-Yates. *Information Retrieval Data Structures and Algorithms.* Prentice Hall, Englewood Cliffs, NJ, 1992.

[23] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins Univ. Press, 3rd edition, 1996.

[24] E. Han, G. Karypis, V. Kumar, and B. Mobasher. Clustering based on association rule hypergraphs (position paper). In *Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 9–13, Tucson, Arizona, 1997.

[25] E. Han, G. Karypis, V. Kumar, and B. Mobasher. Clustering in a high-dimensional space using hypergraph models. Technical Report TR-97-063, Department of Computer Science, University of Minnesota, Minneapolis, 1997.

[26] S. Han, D. Boley, M. Gini, R. Gross, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. WebACE: A web agent for document categorization and exploration. In *Autonomous Agents'98 Conf.*, 1998.

[27] T. Honkela, S. Kaski, K. Lagus, and T. Kohonen. Newsgroup exploration with WEBSOM method and browsing interface. Technical Report A32, Helsinki University of Technology, Laboratory of Computer and Information Science, Espoo, Finland, 1996.

[28] J. E. Jackson. *A User's Guide To Principal Components*. John Wiley & Sons, 1991.

[29] P. Jacobs. Using statistical methods to improve knowledge-based news categorization. *IEEE Expert*, 8(2):13–24, April 1993.

[30] A. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[31] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proceedings ACM/IEEE Design Automation Conference*, 1997.

[32] T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, 2nd edition, 1997.

[33] S. Lu and K. Fu. A sentence-to-sentence clustering procedure for pattern analysis. *IEEE Transactions on Systems, Man and Cybernetics*, 8:381–389, 1978.

[34] Y. S. Maarek and I. B. Shaul. Automatically organizing bookmarks per content. In *Proc. of 5th International World Wide Web Conference*, 1996.

[35] R. Michalski and R. Stepp. Automated construction of classifications: Conceptual clustering versus numerical taxonomy. *IEEE Trans. Patt. Anal. Mach. Intell.*, 5:219–243, 1993.

[36] J. Mingers. Am empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3:319–342, 1989.

[37] M. Nadler and E. P. Smith. *Pattern Recognition Engineering*. Wiley, 1993.

[38] M. Pazzani, J. Muramatsu, and D. Billsus. Syskill & Webert: Identifying interesting Web sites. In *National Conference on Artificial Intelligence*, pages 54–61, Aug. 1996.

[39] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[40] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[41] H. Ritter and T. Kohonen. Self-organizing semantic maps. *Biological Cyvernetics*, 61:241–254, 1989.

[42] E. Rivlin, R. Botafogo, and B. Shneiderman. Navigating in hyperspace: Designing a structure-based toolbox. *Comm. ACM*, 37(2):87–96, Feb. 1994.

[43] G. Salton, J. Allan, and C. Buckley. Automatic structuring and retrieval of large text files. *Comm. A.C.M*, 37(2):97–108, Feb. 1994.

[44] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[45] K. Sycara and L. Chen. WebMate: A personal agent for World-Wide Web browsing and searching. In *Autonomous Agents'98 Conf.*, 1998.

[46] K. Sycara and A. Pannu. A learning personal agent for text filtering and notification. In *Proc. Int'l. Conf. Knowledge Based Systems (KBCS 96)*, 1996.

[47] D. Titterington, A. Smith, and U. Makov. *Statistical Analysis of Finite Mixture Distributions*. John Wiley & Sons, 1985.

[48] R. Weiss, B. Velez, M. A. Sheldon, C. Nemprempre, P. Szilagyi, A. Duda, and D. K. Gifford. HyPursuit: A hierarchical network search engine that exploits content-link hypertext clustering. In *Seventh ACM Conference on Hypertext*, Mar. 1996.

[49] S. Weiss and C. Kulikowski. *Computer Systems that Learn*. Morgan Kaufman, 1991.

[50] M. R. Wulfekuhler and W. F. Punch. Finding salient features for personal Web page categories. In *Proc. of 6th International World Wide Web Conference*, Apr. 1997.