

## Chapter 1

# A SCALABLE HIERARCHICAL ALGORITHM FOR UNSUPERVISED CLUSTERING

Daniel Boley

**Abstract** Top-down hierarchical clustering can be done in a scalable way. Here we describe a scalable unsupervised clustering algorithm designed for large datasets from a variety of applications. The method constructs a tree of nested clusters top-down, where each cluster in the tree is split according to the leading principal direction. We use a fast principal direction solver to achieve a fast overall method. The algorithm can be applied to any dataset whose entries can be embedded in a high dimensional Euclidean space, and takes full advantage of any sparsity present in the data. We show the performance of the method on text document data, in terms of both scalability and quality of clusters. We demonstrate the versatility of the method in different domains by showing results from text documents, human cancer gene expression data, and astrophysical data. For that last domain, we use an out of core variant of the underlying method which is capable of efficiently clustering large datasets using only a relatively small memory partition.

**Keywords:** Unsupervised Clustering, hierarchical clustering, text mining, genomics, sparse matrices, principal directions

## 1. Introduction

Explosive growth in the volume of data available electronically has created a need to be able to automatically explore large data collections. Unsupervised clustering algorithms are classical tools which have increasingly been reexamined for their applicability to data mining efforts.

Ideally, these algorithms would be fast and scalable, require little or no a-priori understanding of the data contents or attributes and need no costly graph building or association rule preprocessing. In many applications it would be useful if the algorithm could also impose a natural hierarchy on the data set, compute properties for the set as a whole, handle cases where attribute information is missing, and be independent of the order in which the data is presented. Principal Direction Divisive Partitioning (PDDP) is such an algorithm [Bol98].

Originally applied in the context of text documents retrieved from the WWW as part of the WebACE project [BGG<sup>+</sup>99b], PDDP has proven to be computationally efficient while providing high quality clusters. In addition to producing a partitioning of the data, the PDDP method

- yields weights showing which attributes were most significant in distinguishing the contents of one cluster from another,
- implements an automated stopping test based on the distribution of the data,
- allows a straightforward way to process datasets with missing attribute values,
- generates a hierarchical tree of clusters which can easily be updated locally, and
- is independent of any particular ordering of the input data, without using any randomized starting conditions.

The method has been successfully applied in a variety of application domains in addition to text documents, such as vision-based texture analysis and movie recommendation services.

## 2. The PDDP Algorithm

The PDDP algorithm employs the vector space model, where each data sample is represented by a vector of numerical attribute values. The data samples are embedded in a very high dimension Euclidean space, and the algorithm partitions this space with a collection of hyperplanes calculated to achieve good separation among the data samples. The data space is separated by a hyperplane into two half-spaces. The process continues recursively by separating each half-space with new hyperplanes computed independently. The method builds a binary tree of many polytope regions from the top down, until a stopping test is satisfied.

Since the PDDP algorithm operates directly with the collection of numerical attribute vectors, only a limited amount of preprocessing is necessary to generate the input data necessary for PDDP. This method was originally developed as part of the WebACE Project [BGG<sup>+</sup>99b] in the context of text documents where each document is represented by a scaled vector of word counts. The preprocessing consisted of removing the stop words and common word endings, and counting the number of occurrences of each word in each document. The result was an  $n$ -vector  $\mathbf{d}$  of word counts associated with each document. All these vectors were combined into a single  $n \times m$  matrix  $\mathbf{M}$  in which each column corresponded to a document and each row corresponded to a particular word. In this domain,

**Algorithm PDDP.**

0. **Start** with  $n \times m$  matrix  $\mathbf{M}$  of vectors, one for each data sample, and a desired number of clusters  $c_{\max}$ .
1. **Initialize** Binary Tree with a single Root Node.
2. **For**  $c = 2, 3, \dots, c_{\max}$  **do**
3.     **Select** leaf node  $\mathcal{C}$  with largest *ScatterValue* (1.2), and  $\mathbf{L}$  &  $\mathbf{R} :=$  left & right children of  $\mathcal{C}$  [step (a) in the text].
4.     **Compute**  $\mathbf{v}_{\mathcal{C}} = g_{\mathcal{C}}(\mathbf{M}_{\mathcal{C}}) \equiv \mathbf{u}_{\mathcal{C}}^T (\mathbf{M}_{\mathcal{C}} - \mathbf{w}_{\mathcal{C}} \mathbf{e}^T)$
5.     **For**  $i \in \mathcal{C}$ , if  $v_i \leq 0$ , assign data sample  $i$  to  $\mathbf{L}$ , else assign it to  $\mathbf{R}$  [step (b) in the text].
6. **Result:** A binary tree with  $c_{\max}$  leaf nodes forming a partitioning of the entire data set.

Figure 1.1. Summary of the method to do a full build of the PDDP tree from scratch. Here  $\mathbf{M}_{\mathcal{C}}$  is the matrix of data vectors for the data samples in cluster  $\mathcal{C}$ , and  $\mathbf{w}_{\mathcal{C}}$ ,  $\mathbf{u}_{\mathcal{C}}$  are the centroid and principal direction vectors, respectively for  $\mathcal{C}$ .

the matrix was generally very sparse, often less than 1% of the entries were nonzero. This sparsity results in a very fast and memory efficient method for carrying out the splitting process. However, this algorithm is not restricted to text domains and here we describe it in general terms.

## 2.1. Basic Algorithm Description

In the general situation, each data sample is represented by an  $n$ -vector of attribute values, and all these vectors (treated as column vectors) are assembled into an  $n \times m$  data matrix  $\mathbf{M}$ . The clustering via PDDP is a recursive process that operates directly on the matrix  $\mathbf{M}$ . PDDP starts with a single “cluster” encompassing the entire dataset, divides this cluster into subclusters recursively using a two step process. At each stage, PDDP (a) selects a cluster to split, and (b) splits that cluster into two subclusters which become children of the original cluster. The result is a binary tree hierarchy imposed on the data collection. At every stage, the leaf nodes in the tree form a partition of the entire data collection. In the process of going to the next stage, one of those leaf nodes is selected and split in two. The behavior of the algorithm is controlled by the methods used to accomplish steps (a) and (b), and these methods are independent of one another. For step (a), PDDP usually selects the cluster with the largest *scatter value* (defined in the next subsection), which is the sum of all the squared distances from each data vector to the cluster centroid  $\mathbf{w}$ , though any suitable criterion can be used.

Once selected in step (a), the node is split in step (b), and this splitting process is the single most expensive step in the whole computation. The key to the

computational efficiency of the entire approach is the efficient computation of the vectors needed in this step. Suppose PDDP were to split cluster  $\mathcal{C}$  consisting of  $k$  data samples of attribute values. It places each data sample  $\mathbf{d}$  in the left or right child of cluster  $\mathcal{C}$  according to the sign of the linear discriminant function

$$g_{\mathcal{C}}(\mathbf{d}) = \mathbf{u}_{\mathcal{C}}^T(\mathbf{d} - \mathbf{w}_{\mathcal{C}}) = \sum_{i \in \mathcal{C}} u_i(d_i - w_i), \quad (1.1)$$

where  $\mathbf{u}_{\mathcal{C}}$ ,  $\mathbf{w}_{\mathcal{C}}$  are vectors associated with  $\mathcal{C}$  to be determined. If  $g_{\mathcal{C}}(\mathbf{d}) \leq 0$ , the data sample  $\mathbf{d}$  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  $\mathbf{u}_{\mathcal{C}}$ ,  $\mathbf{w}_{\mathcal{C}}$  associated with the cluster  $\mathcal{C}$ .

The vector  $\mathbf{w}_{\mathcal{C}} \stackrel{\text{def}}{=} (1/k) \sum_{j \in \mathcal{C}} \mathbf{d}_j$  is the *mean* or *centroid* vector. The vector  $\mathbf{u}_{\mathcal{C}}$  is the direction of maximal variance, also known as the leading left singular vector for the matrix  $\mathbf{M}_{\mathcal{C}} - \mathbf{w}_{\mathcal{C}}\mathbf{e}^T$ . This direction corresponds to the largest eigenvalue of the sample covariance matrix for the cluster. Here  $\mathbf{M}_{\mathcal{C}}$  is the matrix of columns of data samples in cluster  $\mathcal{C}$ . The computation of  $\mathbf{u}_{\mathcal{C}}$  is the most costly part of this step. It can be performed quickly using a Lanczos-based solver for the singular values of the data matrix. This algorithm is very efficient, especially since low accuracy is all that is required, and can take full advantage of any sparsity present in the data.

The overall method can be summarized in Figure 1.1. 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.

We use the classical “iris” data collection (see [DH73, p218] and references therein) to give a simple description of the structures produced by the PDDP algorithm. This data collection consists of 150 flowers: numbers 1-50 are of type *setosa*, numbers 51-100 *versicolor*, and number 101-150 *virginica*. To illustrate the binary tree on a simple case, data from 6 flowers in the set were chosen : 1, 2, 51, 52, 101, 102 with attributes shown in Fig. 1.2.

Fig. 1.3 shows the binary tree that results when the PDDP algorithm is used to split this collection of six flowers into 3 clusters. The top box in Fig. 1.3 represents the root: it contains the indices of all six flowers. The column headed *centroid* is the centroid vector for all six flowers, and the column headed *direction* is the principal direction vector for all six flowers. The root has two children, one of which is a leaf node. In the leaf nodes, the principal direction vectors are not computed because they are not needed. For the non-leaf nodes, shown are the centroid and principal direction vector for the four flowers 51, 52, 101, 102. In this simple case, the PDDP algorithm partitioned the flowers consistently with their types. At each stage, the scatter value was used to select the next node to split, and an automatic stopping test employed using this scatter value (see §2.2 below). Though the scatter values are not shown in Fig. 1.3, the interior non-leaf node had a higher scatter than its “sibling” leaf node.

flower type	1	2	51	52	101	102
sepal length	5.10	4.90	7.00	6.40	6.30	5.80
sepal width	3.50	3.00	3.20	3.20	3.30	2.70
petal length	1.40	1.40	4.70	4.50	6.00	5.10
petal width	0.20	0.20	1.40	1.50	2.50	1.90

Figure 1.2. Raw attributes for six flowers selected to illustrate the PDDP algorithm in Fig. 1.3.

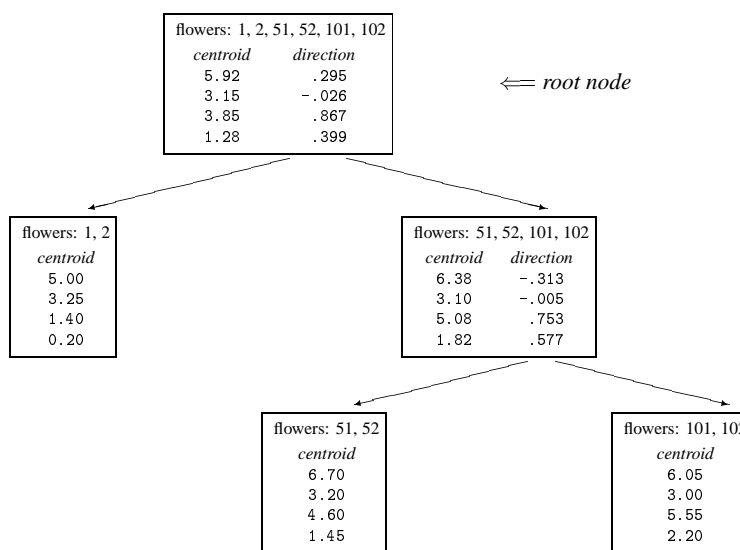


Figure 1.3. PDDP binary tree generated from the six irises shown in Fig. 1.2. Each box represents a node, listing the indices of the flowers represented by that node together with the average values of the attributes over all flowers in that node (“centroid vector”). In the two non-leaf nodes, the “principal direction vector” is shown, which is not computed at all for the leaf nodes.

## 2.2. Stopping Test

To make a working implementation, it is necessary to choose a stopping test, and in many domains, also a way to handle missing data.

Unless there is some other underlying reason to choose a particular number of clusters, the following stopping test can be used. It has been very successful on text documents, giving a number of clusters approximately equal to that computed by other methods tried. There are two components to the general stopping test, (a) a measure of the scatter for each individual cluster and (b) a measure of the relative separation between the clusters. In a top-down algorithm, the former should decrease while the latter should increase. Hence these values lead to a stopping test based on the ratio between these two values. Specifically, the method stops when the ratio of the individual cluster scatter

(a) to the cluster separation measure (b) decreases below a given threshold. It only remains to determine how to compute these two quantities.

For cluster scatter (a), we use a quantity already computed to decide which cluster should be split at each stage (step 3 in Fig. 1.1). This scatter value is defined by sum of squares of the distances from the individual data sample within a cluster to the cluster centroid:

$$ScatterValue_c \stackrel{\text{def}}{=} \sum_{j \in \mathcal{C}} (\mathbf{d}_j - \mathbf{w}_c)^2 = \|\mathbf{M}_c - \mathbf{w}_c \mathbf{e}^T\|_F^2, \quad (1.2)$$

which can be efficiently computed as the Frobenius norm of the cluster matrix.

For the cluster separation measure (b), we use an approximation which can be computed efficiently, even if it is less effective than other measures. The approximation is computed by collecting the centroid vectors for all the individual clusters at each stage and computing their mutual scatter value. That is, let  $\mathbf{W} \stackrel{\text{def}}{=} (\mathbf{w}_{c_1} \ \cdots \ \mathbf{w}_{c_c})$  be the  $n \times c$  matrix of all the collected centers of the  $c$  individual clusters existing at stage  $c$ . Then their mutual scatter value is

$$CentroidScatter \stackrel{\text{def}}{=} \sum_{j=1}^c (\mathbf{w}_{c_j} - \tilde{\mathbf{w}})^2 = \|\mathbf{W} - \tilde{\mathbf{w}} \mathbf{e}^T\|_F^2, \quad (1.3)$$

where  $\tilde{\mathbf{w}} = \frac{1}{c} \mathbf{W} \mathbf{e}$  is the centroid of the collected centroids.

Then the stopping test adopted for the PDDP method is the ratio

$$\frac{\max_{j=1}^c ScatterValue_{c_j}}{CentroidScatter} \leq \text{some fixed threshold value}. \quad (1.4)$$

The *threshold value* is usually set to 1, but it can be set to larger or smaller values to obtain coarser or finer clusters, respectively.

### 2.3. Missing Values

Missing values often appear in datasets representing data that is unknown as opposed to “zero.” It is desirable to avoid having these values affect the placement of data samples into clusters. In the presence of missing data, it is a simple matter to compute the centroid vector  $\mathbf{w}_c$  by averaging each attribute value only over the non-missing values. Once the centroid vector is computed, we are able to replace each missing value with the corresponding average value for that attribute. With this replacement, the corresponding entries in  $\mathbf{M}_c - \mathbf{w}_c \mathbf{e}^T$  are zero and hence have no contribution to the principal direction  $\mathbf{u}_c$  or the linear discriminant value (1.1). This replacement is temporary, and once the splitting process is completed each missing entry is reset to its original ‘missing’ value before proceeding to the next split. In this fashion missing values never push samples into the left or right child of a cluster, but the choice of which child cluster receives a given sample is based only on the non-missing attribute values.

<i>cluster:</i>	1	2	3
setosa	50	0	0
versicolor	0	46	4
virginica	0	0	50

Figure 1.4. PDDP Clustering result (confusion matrix) on the entire dataset of 150 irises, using norm scaling on the input data and the scatter-based automated stopping test (1.4) [with a threshold value of 2]. Each entry in the confusion matrix is a count showing the number of flowers of that type in the computed PDDP cluster. When the threshold value is set to 1, cluster 3 is split in two.

### 3. Performance

Performance testing has demonstrated PDDP provides high quality clusters at a relatively low computational cost. Most of the experience has been on text documents, but the algorithm has also been successful on datasets of movie ratings, texture images, toxicity databases, etc. In this section we summarize the main performance results from the experience on text documents. For example, using the PDDP algorithm with the stopping test on the entire “iris” dataset, the algorithm yielded the clusters shown in Fig. 1.4.

#### 3.1. Speed

The cost of the PDDP method depends almost entirely on the cost of its most expensive step, which is the computation of the principal direction vector. The computation of this vector is carried out with a Lanczos-based eigensolver [GV96], whose cost is proportional to the number of nonzeros in the data matrix. Thus the PDDP method scales linearly with the size of the data matrix. This behavior is shown in Fig. 1.5, where it is seen that the cost depends more on the number of nonzeros (the horizontal axis) than the actual number of documents or words. For example, Fig. 1.5 shows that the time to obtain the clusters shown in Fig. 1.6 was approximately 37 seconds on an SGI Challenge workstation.

To compare the cost of PDDP with that of more classical methods, we applied PDDP, Hypergraph [HKKM98], K-means - LSI [BDO95], Agglomeration [DH73], and AutoClass [CS96] methods to a text document collection of 185 documents with 10538 word dictionary. The first three methods all took under 2 minutes a 185MHz Sun workstation, but Agglomeration and Autoclass each took at least 30 minutes. It is difficult to compare PDDP with the more classical methods on large examples because, unlike PDDP, most classical methods do not scale linearly with the size of the problem. In our experiments on larger datasets, unmodified Agglomeration and Autoclass became prohibitively expensive. We remark that Agglomeration is a classical algorithm which generates a tree bottom-up that is very similar to that produced top-down by PDDP.

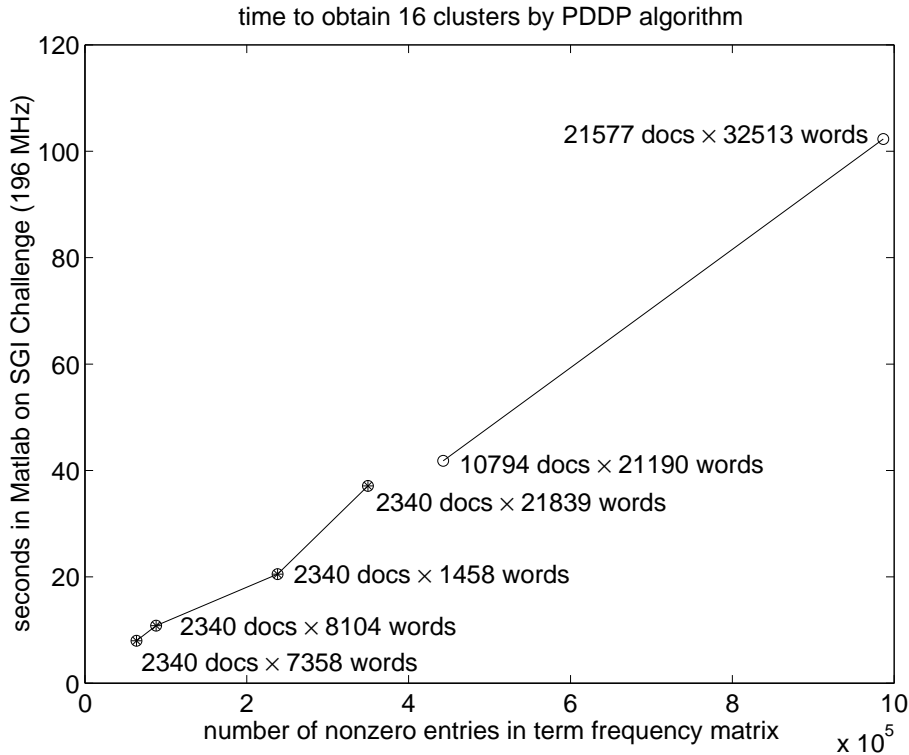


Figure 1.5. Time to compute 16 clusters for various datasets using interpreted MATLAB code on an SGI challenge workstation (196 MHz), against the number of nonzeros in the data matrix.

Agglomeration is well known to be very effective, but it depends on the choice of “inter-cluster” separation, and it begins by computing the separation between every pair of data samples (documents), treating them as “singlet clusters.” This is an  $O(m^2)$  process, where  $m$  is the number of data samples. By randomly splitting the samples into  $\sqrt{m}$  initial clusters, applying Agglomeration to each cluster, and then agglomerating the resulting clusters, one can reduce the total asymptotic running time to  $O(m)$  (see e.g. [CKPT92]), but the results depend on the random initial splitting which leads to variability in the final clustering.

Other algorithms also exist which can also lead to high quality results (e.g. k-nearest neighbor or k-means) [GJJ96], but most suffer from high cost (e.g. nearest neighbor) or depend critically on a good starting point which is often chosen randomly (e.g. k-means). BIRCH [ZRL96] incrementally builds a tree during a single pass over the data, and sweeping over the data in a different order could change the result.



<i>cluster:</i>	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
business	90	0	0	0	7	0	5	12	0	6	0	1	18	3	0	0
health	0	150	166	171	3	0	1	1	0	0	0	0	0	2	0	0
politics	2	0	0	0	100	1	2	0	0	1	0	2	1	5	0	0
sports	0	0	0	0	1	62	35	0	0	1	0	0	0	42	0	0
technology	8	0	0	0	0	1	14	24	0	8	0	1	4	0	0	0
entertain.	24	0	0	4	11	4	22	61	135	131	148	159	143	137	204	206

Figure 1.6. Confusion Matrix showing how documents were distributed to 16 different clusters by topic labels. This data set has 2340 documents, and was based on a dictionary of 21839 words. Further subdivisions are necessary to classify the large collection of entertainment documents in this set.








<i>method</i>	<i>entropy</i>
Agglomeration - norm scaling [DH73]	0.684 
PDDP - norm scaling	0.689 
Hypergraph [HKKM98]	0.787 
K-means - LSI [BDO95]	0.837 
PDDP - TFIDF scaling [SB88]	1.057 
AutoClass [CS96]	2.049 
Agglomeration - TFIDF scaling	2.339 

Figure 1.7. Entropies by various methods on a set of 185 documents with a 10538 word dictionary.

### 3.2. Quality of Clusters

There is no absolute scale to measure the quality of clusters, but one can see the so-called *confusion matrix* for a dataset consisting of 2,340 text documents using a dictionary of 21839 words in Fig. 1.6. The confusion matrix shows how the different types of documents in six broad categories were distributed among 16 clusters.

To be able to compare the quality of the clusters produced by PDDP with that arising from other methods, it is necessary to quantify the measure of quality using, for example, an entropy measure as in [Bol98, BGG<sup>+</sup>99a]. This entropy measure requires that the documents be given category labels in advance by hand, and hence it is difficult to apply this measure to very large datasets. However, for the dataset of 185 documents used to compare the costs of PDDP with other methods mentioned above, it was relatively easy to assign topic labels to the documents. These labels were not used by any of the classification methods discussed in this paper or for any of the performances tests. The labels were used only to measure the entropy of the resulting clusters as a measure of quality. The entropy of a given cluster  $\mathcal{C}$  is defined by

$$e_{\mathcal{C}} = - \sum_i \left( \frac{c(i, \mathcal{C})}{\sum_i c(i, \mathcal{C})} \right) \cdot \log \left( \frac{c(i, \mathcal{C})}{\sum_i c(i, \mathcal{C})} \right),$$

where  $c(i, \mathcal{C})$  is the number of times label  $i$  occurs in cluster  $\mathcal{C}$ . A cluster has zero entropy if the all the documents in the cluster have the same label, otherwise it has a positive entropy. The total entropy is the weighted average of the individual cluster entropies:

$$e_{\text{total}} = \frac{1}{m} \sum_{\mathcal{C}} e_{\mathcal{C}} \cdot [\text{number of documents in cluster } \mathcal{C}].$$

As a consequence, the lower the entropy the better the quality.

Various clustering methods were applied to this dataset, and the results are shown in Fig. 1.7. Each method was forced to produce the same number of clusters in order to make the results consistent.

## 4. Scientific Data Set Examples

We illustrate how this clustering method performs on scientific data sets by using two examples, one drawn from gene expression data and one drawn from an astronomical sky survey. The aim is to show the kinds of results that can be obtained in an automated way from the PDDP clustering algorithm. The goal is to allow easier exploration of large datasets, facilitating the extraction of new patterns and novel discoveries from the data. The description of novel discoveries themselves is beyond the scope of this paper.

### 4.1. Gene Expression Data

Recent years has seen the explosion of genetic data available electronically. Much genetic data is structural, such as the genetic sequences of base pairs making up the DNA in living cells. Recently, however, there has been much interest in functional genetic data, such as showing which genes are active (i.e. *expressed*) under a variety of external conditions. Such information can be used to discover the purpose of many genes as well as to identify which genes are prone to “misbehavior,” thereby causing disease. For example, it is well known that cancer is often caused by genes behaving improperly giving rise to uncontrolled growth in cells. It has been found recently that some cancers have a distinctive gene signature which can be used to both refine diagnoses and to focus treatments. We use the data from one such study on the gene signature for a certain class of lymphomas [AED<sup>+</sup>00] to illustrate the kinds of analysis that the PDDP produces in an automated fashion.

The gene expression data in [AED<sup>+</sup>00] consists of microarray assays of 4026 genes on 96 different tissue samples obtained from patients with a certain class of non-Hodgkins lymphoma. The goal was to identify which genes were important for different kinds of cancer, both for the purpose of distinguishing between cancers and for the purpose of designing therapies tailored for each particular cancer. The data consists of a  $4026 \times 96$  matrix of attribute values

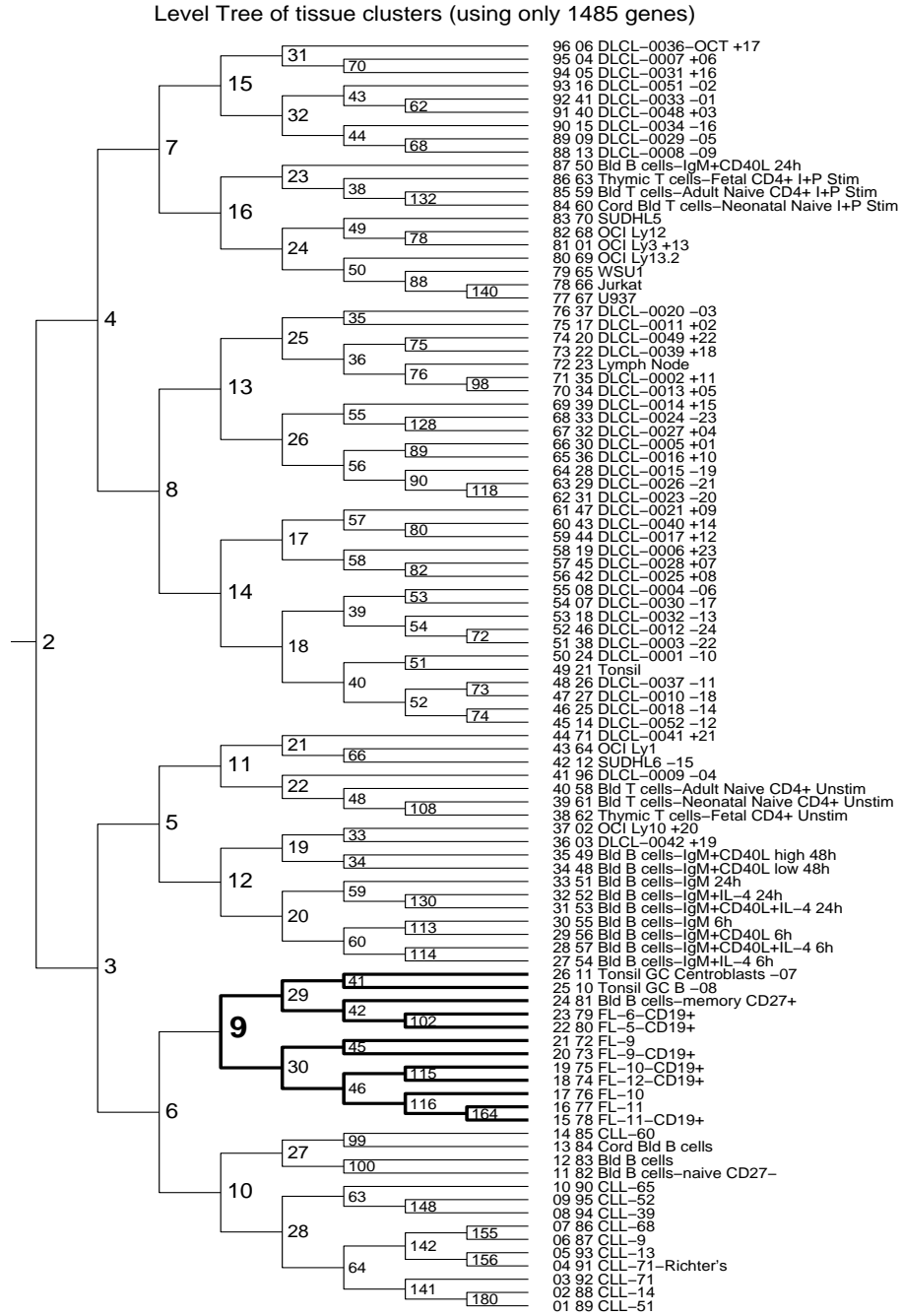


Figure 1.8. PDDP tree using the 1485 most distinctive genes from the 4026 × 96 gene expression matrix from [AED+00]. Node 9 is illustrated in Fig. 1.9.

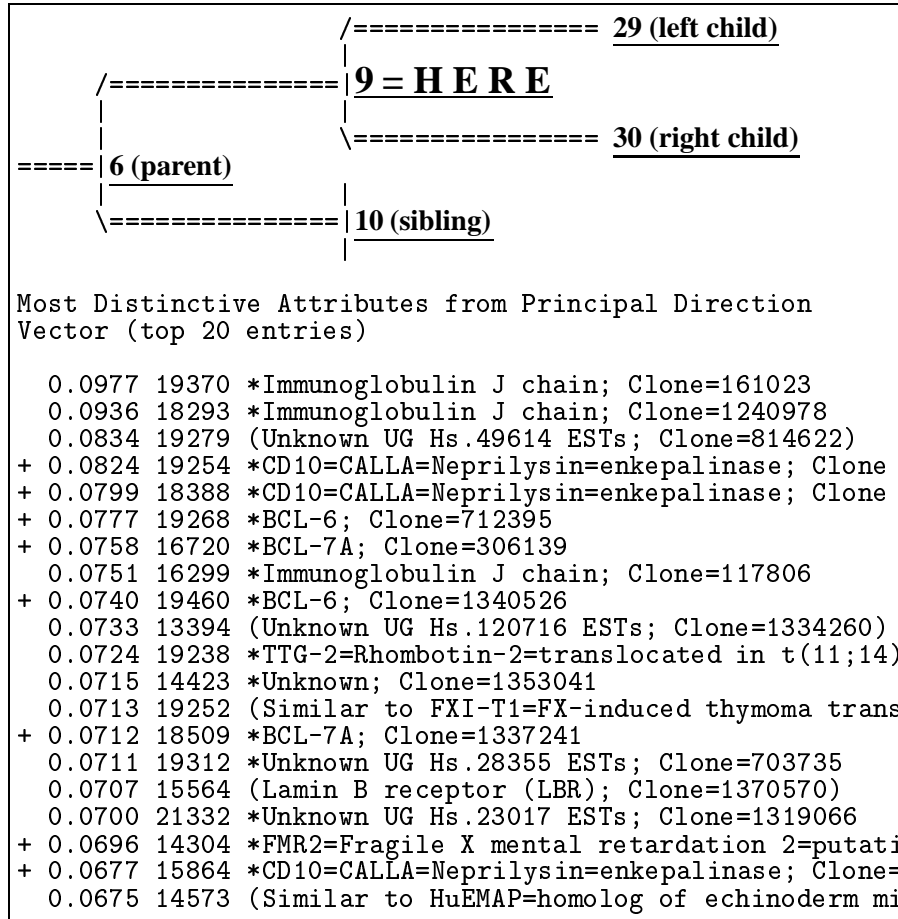


Figure 1.9. Excerpt from web browser display of Node 9 in the tree of Fig. 1.8 showing the genes distinguishing this Node from its sibling node. The symbol “+” has been added to identify the germinal centre B cell genes, (known to differ in FL compared to CLL).

in the range  $[-2, 2]$  representing logarithms of the activity of each gene within each tissue. Approximately 5% of the data is missing because of faults during the collection of the experimental data. The authors of [AED<sup>+</sup>00] applied hierarchical agglomeration to obtain a tree structure on the 96 tissue samples, and then manually examined the clusters within that tree to identify significant genes. PDDP also produces a very similar tree, but top-down instead of bottom-up.

The PDDP tree can be used in at least two ways on this dataset. The simplest is to use the principal direction information to identify which genes are significant in splitting *anywhere* in the tree. One may remove any gene for which the

corresponding entry (weight) in any principal direction vector is small in absolute value. Such genes are never significant in splitting a cluster into subclusters using the inner product (1.1). Therefore the genes that remain are more useful in distinguishing among the tissues samples. In this experiment, we kept the genes whose weight in any principal direction vector exceeds a given threshold value. This value was chosen by noting that if all the entries in a principal direction vector had equal weight, the weight would have to be  $1/\sqrt{n}$  because the vector is normalized to make the sum of squares equal to 1. To extract the significant genes, a value of 3 times this neutral value was chosen, namely  $3/\sqrt{n} = .078$ . This reduced the number of genes from 4062 down to 1485, and PDDP was then applied to this reduced set (96 tissues by 1485 genes). The PDDP trees produced using all 4026 genes and with only 1485 genes were very similar, and we show the latter tree in Fig. 1.8.

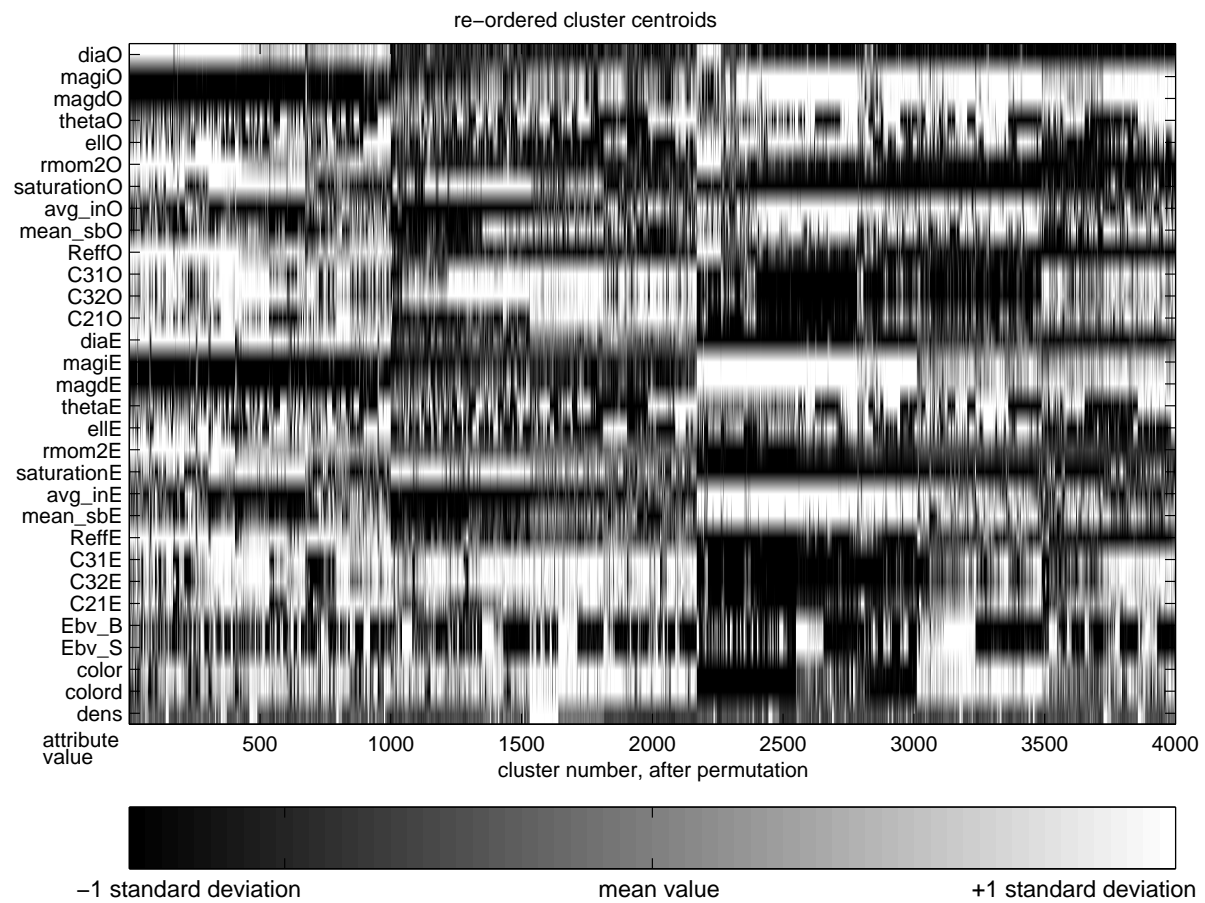
To allow a user to explore the tree and identify these distinctive genes, the software converts the tree from the internal representation into an HTML document which can be browsed using any standard web browser. As an example, an excerpt from a web browser display is reproduced in Fig. 1.9, showing the distinctive genes for the cluster shown in bold in Fig. 1.8. Of course, there is no intention to claim that the genes listed in Fig. 1.9 are all significant for this cluster or any other cluster, but we do remark that many of the listed genes were specifically identified in [AED<sup>+</sup>00] as germinal centre B cells, known to distinguish the FL tissue samples in this cluster from the CLL tissue samples in the sibling cluster.

## 4.2. Astronomical Sky Survey

The advent of high-powered computational facilities has given rise to automated surveys of the sky with a view of cataloging and/or classifying the enormous number of objects that can be observed from Earth. A further goal of such surveys is to allow automated or semi-automated exploration to permit one to find “interesting” objects or clusters of objects for further study. Automated clustering methods would permit the exploration of this enormous dataset in many dimensions, allowing astronomers to identify distinctive objects and relationships between objects. In [PM99], *kd*-trees were proposed as a way to speed up a k-means clustering of Sloan Digital Sky Survey (a more recent automated sky survey) data. Though computed using different algorithms, *kd*-trees can be thought of as a special case of PDDP trees in which each separating hyperplane is normal to a coordinate axis. Hence PDDP trees share many of the same favorable properties, including yielding good seeds for the k-means method. This is an area which requires further research.

Preliminary experiments were carried out with data derived from the Minnesota Automated Plate Scan (APS) [PHO<sup>+</sup>93, Web98]. This is a project to

Figure 1.10. Individual centroid vectors for 4000 clusters computed from a sample APS dataset.



digitize the contents of photographic plates of the heavens from the Palomar Observatory Sky Survey (POSS I) originally produced in the 1950s, before the advent of artificial satellites.

The size of this data set spurred the development of an out-of-core version of the PDDP method, since the data set is too large to be loaded into memory at one time. This out-of-core version of the PDDP method has been designed for the situation where there is a large number of data samples, but relatively few attributes. Suppose the number of attributes  $n$  is much less than the number of data samples  $m$ , so that the data matrix  $\mathbf{M}$  is very wide but not tall. In the following we use  $\mathbf{e} \stackrel{\text{def}}{=} (1 \ \cdots \ 1)^T$  to denote a column vector of all ones of appropriate size. To compute the centroid vector  $\mathbf{w}_C = \frac{1}{k} \mathbf{M}_C \mathbf{e}$  (where  $k$  is the size of the cluster  $C$ ), it is necessary to make a pass over the entire matrix  $\mathbf{M}_C$ . To compute the principal direction vector  $\mathbf{u}_C$ , to the low accuracy needed for this algorithm, it suffices to accumulate the  $n \times n$  matrix

$$\mathbf{A} \stackrel{\text{def}}{=} (\mathbf{M}_C - \mathbf{w}_C \mathbf{e}^T)(\mathbf{M}_C - \mathbf{w}_C \mathbf{e}^T)^T = \mathbf{M}_C \mathbf{M}_C^T - \mathbf{Z} - \mathbf{Z}^T + k \mathbf{w}_C \mathbf{w}_C^T,$$

where  $\mathbf{Z} \stackrel{\text{def}}{=} \mathbf{M}_C \mathbf{e} \mathbf{w}_C^T = k \mathbf{w}_C \mathbf{w}_C^T$ . Thus the matrix  $\mathbf{A}$  is

$$\mathbf{A} = \mathbf{M}_C \mathbf{M}_C^T - k \mathbf{w}_C \mathbf{w}_C^T$$

With one pass through the data, column by column, we can accumulate  $\mathbf{M}_C \mathbf{M}_C^T = \sum_{j \in C} \mathbf{d}_j \mathbf{d}_j^T$ , and  $\mathbf{w}_C \stackrel{\text{def}}{=} (1/k) \sum_{j \in C} \mathbf{d}_j$ , and use the result to form  $\mathbf{A}$  without any further access to the data. By saving the columns associated with each cluster in a separate scratch file as each cluster is formed, the passes over the data matrix can be made very fast.

As a small test case to demonstrate the out-of-core method, we used a small sample from the APS consisting of 212089 galactic objects, each with 26 attribute values. Using a 195 MHz SGI challenge machine (relatively old machine chosen to match that used to produce the timings in Fig. 1.5) and a Matlab implementation of the out-of-core algorithm, a PDDP tree with 4000 clusters can be computed in under 10 minutes. Of course it needs enough scratch disk space to store two copies of the data set, but the access to each copy of the data is sequential and hence reasonably efficient. Of course with enough memory and a compiled version, this can be computed much more efficiently with a standard in-core algorithm, but this experiment serves to demonstrate the practicality of the out-of-core algorithm. To visualize the results, we collected all the centroids for the 4000 clusters and displayed them side by side using color-coding to display the magnitude of each attribute value. The result is shown in Fig. 1.10, where it is seen that certain attributes appear to vary together over a portion of the data. The interpretation of these results is still an active area of research, especially an analysis of the utility of the encodings used in this digitization of the APS data set.

## 5. Conclusion

This brief introduction to the method of Principal Direction Divisive Partitioning is intended to show that this algorithm is a useful tool in situations which call for unsupervised clustering of large data collections which can be represented using very high dimensional numerical vectors. The PDDP method operates on the entire data collection as a unit, producing clusters of a quality comparable with those of the best alternative methods, but often at less cost. This brief description should be sufficient to show that the PDDP algorithm can be adapted to a wide variety of datasets and different situations, such as the presence of missing data, as well as providing useful information on the factors that were most significant in yielding the clusters that are eventually computed. The examples include some hints on the kinds of information which can be extracted from the PDDP tree, both to explain the results and to extract the most significant parts of the dataset.

Of course, there are limitations to the PDDP algorithm. The main limitations are PDDP's sensitivity to scaling of the data and the need to have all the data available at once. The sensitivity to scaling is a natural consequence of the way the data is represented with vectors of real numbers, and any algorithm using such a representation will suffer from the same limitation. The second limitation is the object of current research, where we have the goal of allowing the tree to be computed using a fraction of the data and updating the tree as new data becomes available.

## Acknowledgments

This work was partially supported by NSF grant IIS-9811229.

## References

- [AED<sup>+</sup>00] A. A. Alizadeh, M. B. Eisen, R. E. Davis, C. Ma, I. S. Lossos, A. Rosenwald, J. C. Boldrick, H. Sabet, T. Tran, X. Yu, J. I. Powell, L. Yang, G. E. Marti, T. Moore, J. Hudson Jr, L. Lu, D. B. Lewis, R. Tibshirani, G. Sherlock, W. C. Chan, T. C. Greiner, D. D. Weisenburger, J. O. Armitage, R. Warnke, R. Levy, W. Wilson, M. R. Grever, J. C. Byrd, D. Botstein, P. O. Brown, and L. M. Staudt. Distinct types of diffuse large b-cell lymphoma identified by gene expression profiling. *Nature*, 403:503–511, 2000.
- [BDO95] M. W. Berry, S. T. Dumais, and Gavin W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37:573–595, 1995.
- [BGG<sup>+</sup>99a] D. Boley, M. Gini, R. Gross, S. Han, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. Partitioning-based cluster-



- ing for web document categorization. *Decision Support Systems*, 27(3):329–341, 1999.
- [BGG<sup>+</sup>99b] D.L. Boley, M. Gini, R. Gross, E-H Han, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. Document categorization and query generation on the World Wide Web using WebACE. *AI Review*, 13(5-6):365–391, 1999.
- [Bol98] D. L. Boley. Principal Direction Divisive Partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998.
- [CKPT92] D.R. Cutting, D.R. Karger, J.O. Pedersen, and J.W. 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.
- [CS96] P. Cheeseman and J. Stutz. Bayesian classification (Autoclass): Theory and results. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 153–180. AAAI/MIT Press, 1996.
- [DH73] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.
- [GJJ96] Earl Gose, Richard Johnsonbaugh, and Steve Jost. *Pattern Recognition and Image Analysis*. Prentice Hall, 1996.
- [GV96] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins Univ. Press, 3rd edition, 1996.
- [HKKM98] E.H. Han, G. Karypis, V. Kumar, and B. Mobasher. Hypergraph based clustering in high-dimensional data sets: A summary of results. *Bulletin of the Technical Committee on Data Engineering*, 21(1), 1998.
- [PHO<sup>+</sup>93] R.L. Pennington, R.M. Humphreys, S.C. Odewahn, W. Zumach, and P.M. Thurmes. The automated plate scanner catalog of the palomar sky survey – scanning parameters and procedures. *P. A. S. P.*, 105:521ff, 1993.
- [PM99] Dan Pelleg and Andrew Moore. Accelerating exact k-means algorithms with geometric reasoning conference on knowledge discovery in databases. In *Proc. of the Fifth Int'l Conference on Knowledge Discovery and Data Mining*, 1999.
- [SB88] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

- [Web98] Web Site. The automated plate scanner, 1998. <http://aps.umn.edu>, supported by NASA.
- [ZRL96] T. Zhang, R. Ramakrishnan, and M. Linvy. Birch: an efficient data clustering method for large databases. In *Proc. of 1996 ACM-SIGMOD Int. Conf. on Management of Data*, Montreal, Quebec, 1996.