Using Low-Memory Approximations to Cluster Very Large Data Sets

David Littau
littau@cs.umn.edu
Daniel Boley
boleyn@cs.umn.edu

Abstract

Clustering large data sets is distinctly more difficult when the data sets become large enough so that they will not fit into memory at once. In this paper we present a variant of Principal Direction Divisive Partitioning (PDDP) which is applicable to large data bases. PDDP was designed to be applied to the entire data set at once, and for good performance required the entire data set be present in core memory. In this variant, piecemeal PDDP, the original data is broken up into sections which will fit into memory and clustered. The cluster centers are used to create approximations to the original data items. The approximations are designed to take low memory, but they are not otherwise designed to satisfy any particular goal. The piecemeal algorithm is uniquely able to take advantage of the approximations, since it accesses the data only through matrix-vector products. We evaluated the performance of piecemeal PDDP on three real data sets. The results of our experiments show that the performance of piecemeal PDDP is comparable to standard PDDP for the data examined.

keywords: clustering, large data sets, PDDP, data mining, principal directions

*This research was supported by NSF grant IIS-9811229.
1 Introduction

It is well accepted that data is being generated at a prodigious rate in science and technology research and development, on the World-wide Web, by government agencies, etc. There is too much data to be organized entirely by hand and hence automated or semi-automated methods are required to organize data, at least as a first step. Therefore, fast and effective methods for unsupervised clustering are form a critical step in the processing of data. These methods can gather the data into groupings, or clusters, which are composed of data items with similar attributes. The clusters can help with classification since, if something is known about one sample, or a few samples, of data in the cluster, the other items in the cluster will have a similar properties. Applying more precise supervised classification methods, or even simple human exploring of the data will be much easier if one knows one can focus one's attention to a few clusters.

One fast, scalable clustering algorithm is Principal Direction Divisive Partitioning (PDDP)\[3\]. We choose the PDDP method because it offers unique opportunities to take advantage of reduced order representations that can be constructed from the original data. Central to the PDDP algorithm is the calculation of the principal direction of the data. This is done using a Lanczos procedure, and a critical part of the Lanczos procedure is the calculation of a matrix-vector product. Using the approximation to the original data results in the matrix-vector product being replaced by a matrix-matrix-vector product. If this product is calculated in the correct order, no additional space (with respect to the usual matrix-vector product) is needed.

The usual clustering algorithms depend on being able to scan through the entire dataset many times. This becomes impractical when we reach the enormous sizes of datasets today. The proposed solution to this problem is to break the original data into smaller pieces, called sections, which will fit into memory. Each section is clustered using PDDP, and the centers of the clusters are extracted. The centers are then used to compute a least-squares approximation to the data comprising the section. Once this is done for each section of data, the results for all sections are collected and used to perform one more PDDP clustering operation. This final clustering will be an approximate clustering of the entire original data set. Also, since PDDP is used in each phase, the algorithm remains scalable, with the additional cost of calculating the least-squares approximation. Since the size of each section can be arbitrarily small, the cost of computing the least-squares approximation need not be significant overall.

The approximations to the original data can be made as accurate as available memory allows. The number of centroids calculated for each section of data, and the number of centroids used to approximate each original data item, are completely user-determined. Generally, the more centroids calculated for each section, and the more centers used to approximate each original data point, the more accurate the final approximation. However, more memory will be necessary to contain a more accurate approximation as compared to a cruder approximation.

Reducing the original data to approximations allows PDDP to process the dataset in a reasonable amount of time using relatively modest amounts of memory. The method examines every data item during the process of creating the classification, and will not miss or ignore the presence of outliers. It can do the most expensive work without having to examine the entire data set at once. It could be used as an incremental method by processing a set of new data as a new section without the need to examine the original old data. The amount of memory required by the approximation can be set by the user, and the only tradeoff is the accuracy of the approximation and, to a lesser extent, the accuracy of the clustering. There are no parameters which need to be selected or tuned. The ultimate objective of this work is to develop a method that will scale well to very large data sets while retaining a small memory footprint.
2 Previous Work

The problem of clustering large data sets has been addressed often in the past. Hierarchical agglomeration [12] and $k$-means [9, p201] are two popular starting points for developing scalable clustering methods. Hierarchical agglomeration assembles clusters by recursively joining the two closest clusters, where the distance is computed with respect to all points in all clusters. $K$-means starts with a fixed set of cluster centers, and goes through an iterative process of assigning the data to the closest center and then using the data in a given cluster to calculate a new center. Other methods reduce the original data set to a smaller set of representatives. Sampling may also be used to reduce the amount of time and memory a given method requires.

Two methods based on hierarchical agglomeration are Scatter/Gather [7] and CURE [13]. Scatter/Gather reduces the running time by only applying agglomeration to smaller sets of the data, and then by agglomerating the centers resulting from the previous operation. The method is $O(p^2)$, where $p$ is the number of data points agglomerated at any given time, and careful choice of parameters is necessary to insure that the overall running time does not change from rectangular to quadratic. CURE reduces the running time by measuring the inter-cluster distance using a small fixed number of well-scattered points, and by sampling the data. This reduces the overall running time to $O(p^2 \log p)$, where $p$ is the total number of well-scattered points chosen to represent the data.

CLARANS [16] is a $k$-centers algorithm which finds a set of $k$ cluster medoids, which are like the centroids in $k$-means except they are members of the data set. Various clusterings are examined by replacing one existing medoid by a randomly chosen medoid. Only a small number of alternate clusterings are examined in order to keep the running time rectangular.

The problem of $k$-means requiring re-starts due to convergence to local optima is addressed in [6]. This method is designed to find good initial centers for large data sets, such that the $k$-means algorithm will not have to be repeated. It does this by clustering independent samples of the data, and then clustering the resulting centroids. The final set of centroids becomes the starting centers for $k$-means. The applicability to large data sets can be further enhanced by the method outlined in [5]. The method applies $k$-means to as much data as can fit into memory and uses the cluster centers to represent the data. The cluster centroids are used to represent the data which has been clustered, new data is loaded into memory, and the process continues. Both solutions to scaling $k$-means to large data sets retain the same $O(kn)$ running time that $k$-means needs.

BIRCH [18] is a method which is used to pre-cluster large data sets. BIRCH incrementally groups the data as tightly as possible based on similarity in their attributes, and in the process constructs as many representatives of the original data as the available memory will contain. If the amount of space taken by the data runs out during the BIRCH process, the tightness of the groupings is relaxed, the groupings are internally re-assigned, and the processing of the data continues. Since BIRCH examines every data point, it runs in $O(n)$ time. Another algorithm must be applied to obtain the actual clustering of the data.

Using representatives and/or sampling appears to be a common method of clustering large data sets. BIRCH [18] was designed to produce representatives which are like cluster centroids along with the number of data points represented by the vector. The representatives in [5] perform a similar task. The latter method used sampling to avoid memory limitations, but would still allow exposure to the entire data set, as did BIRCH. CURE [13], on the other hand, made all of its clustering decisions using one fixed sample of data. This would be true of any technique which uses sampling.

In the method outlined in this work, we allow more than one representative for each data point,
and the relationship between each data point and its representative(s) is uniquely represented. We also examine every data point when constructing our approximation. We believe this method may have some advantages when compared with the clustering algorithms previously proposed for large data sets.

3 Clustering

The word cluster is defined differently depending on the domain of interest. In this work, a cluster is defined as a group of data items that have similar values for a certain subset of their attributes. It follows that clustering is defined as the creation of clusters, with the goal being that every item in a given cluster is more similar to other items in that cluster than they are to the items in any other cluster.

Since clustering is done in this case using the vector space model, the vector space model is described first. Then, a more formal mathematical definition of clustering is given, followed by the description of a few measurements appropriate to clustering.

3.1 Vector Space Model

The vector space model is very common in clustering and data mining applications. In the vector space model, an individual data item is represented by a column vector of its attributes. The interpretation of the attributes is dependent on the type of data involved. In the case of document data, the document vector $\mathbf{x}$ can be a count of word occurrences or frequencies. If word $j$ appears in the $i$-th document $l$ times, then $x_{ij} = l$ (before it is scaled). For other kinds of data a given attribute is different. For example, an attribute of astronomical data could be position, brightness, the red-shift value, etc. The important point is that each attribute, be it a word or measurement, has a unique row assigned to it in the column vector representing the data item, and that each attribute has a numerical value.

The entire data set can be thought of as a matrix of attribute vectors. If we have a data set containing the attribute vectors $\mathbf{x}_i$, then we can define

$$\mathbf{M} \overset{\text{def}}{=} [\mathbf{x}_1 \mathbf{x}_2 \ldots \mathbf{x}_n],$$

where $\mathbf{M}$ is an $m \times n$ matrix, $m$ is the number of attributes in each vector $\mathbf{x}_i$, and $n$ is the number of items in the data set. Since the clustering and approximation methods outlined in this proposal use linear algebraic techniques, this representation is very convenient.

3.2 Clustering Model

There are many instances in which data must be classified, but the classes are initially undefined, meaning that the data is not labeled. This is the usual case when an unsupervised clustering method is applied. The clustering is a re-arrangement of the original data such that the data is grouped using some measure of similarity. Let $\mathcal{M} = \{\mathbf{x}_1 \mathbf{x}_2 \ldots \mathbf{x}_n\}$, which is just the set of attribute vectors from the matrix $\mathbf{M}$ in (1). Then clustering can be interpreted as the result of the partition of the set $\mathcal{M}$ into subsets such that

$$\mathcal{M} = \bigcup_{i=1}^{K} \mathcal{M}_i, \quad \mathcal{M}_i \cap \mathcal{M}_j = \emptyset, i \neq j$$

(2)
where $K$ is the total number of clusters. The partitioning takes place using only internal information and without any prior knowledge of the data. The goal of a clustering method is to maximize the similarity among the elements of each subset $M_i$ and minimize the similarity of the elements in $M_i$ with respect to any other subset $M_j$. The vectors in the set $M_C$ can be used to define the matrix $M_i$ such that

$$M_i \equiv [x_j], \; x_j \in M_i. \quad (3)$$

In this work, a cluster $C$ will usually be referred to using its matrix of attribute vectors $M_C$.

### 3.3 Measurements

There are a few standard measurements associated with clusters and clustering algorithms. One common measurement when evaluating scalar data is the mean value. The mean value describes the data in a compact way. In the case of a cluster, where the data items have multiple attributes, the mean is called the centroid. The centroid $w_C$ of a cluster $M_C$ is defined as:

$$w_C \equiv \frac{1}{k_C} \sum_{j \in C} x_j \quad (4)$$

where $k_C$ is the number of items in cluster $M_C$ and $x_j$ is the $j^{th}$ column of $M_C$.

The most commonly used intrinsic measure of the quality of a cluster is the ScatterValue. The ScatterValue is a measure of the cohesiveness of the data items in a cluster with respect to the centroid of the cluster, and is analogous to the variance of scalar data. The ScatterValue of a cluster $M_C$ is defined as:

$$\text{ScatterValue}_C \equiv \sum_{j \in C} (x_j - w_C)^2 = \|M_C - w_C e^T\|^2 \quad (5)$$

where $e$ is the $m$-dimensional vector $[1 \; 1 \; \ldots \; 1]^T$ and $\| \cdot \|_F$ is the Frobenius norm. The Frobenius norm is the square-root of the sum of the squares of every entry in the matrix. A low ScatterValue indicates good cluster quality, but it should be noted that this is a relative measure of quality, since it is dependent on the number of items in the cluster. Normalizing with respect to the number of items in the given cluster would fix this problem.

Another useful clustering performance measure is entropy. The entropy measures the coherence of a cluster with respect to how a cluster is labelled. An entropy calculation assumes that the labelling is perfect, which is not a good assumption in every case since any labelling performed by a human can be subjective. Since the data must be labelled, entropy cannot be used when the data have not been classified prior to clustering.

The entropy of cluster $j$ is defined by:

$$e_j \equiv - \sum_i \left( \frac{c(i,j)}{\sum_i c(i,j)} \right) \cdot \log \left( \frac{c(i,j)}{\sum_i c(i,j)} \right), \quad (6)$$

where $c(i,j)$ is the number of times label $i$ occurs in cluster $j$. If all of the labels of the items in a given cluster are the same, then the entropy of that cluster is zero. Otherwise, the entropy is positive. The total entropy for the PDDP tree is the weighted average of the cluster entropies:

$$e_{total} \equiv \frac{1}{m} \sum_i e_i \cdot k_i \quad (7)$$

The lower the entropy, the better the quality of the clustering.
4 The PDDP Algorithm

PDDP [3] is a clustering algorithm developed using techniques from numerical linear algebra. PDDP is a top-down method which recursively divides the data into smaller and smaller clusters, assembling all the clusters into a binary tree. Starting with the root node representing the entire dataset, PDDP computes the hyperplane which best divides the data. All the data on one side of the hyperplane is associated with one branch, and the data on the other side of the hyperplane is associated with the other branch. The process continues on each branch in turn until some stopping criteria is met, which can be based on intrinsic measures of the data in the nodes or on a final desired number of leaf nodes (clusters). This method was originally developed as part of the WebACE Project [4] in the context of text documents where each document is represented by a scaled vector of word counts. However, this algorithm is not restricted to text domains and here it is described in general terms.

Algorithm PDDP.
0. Start with $n \times m$ matrix $M$ of vectors, one for each data sample, and a desired number of clusters $k_f$.
1. Initialize Binary Tree with a single Root Node.
2. For $c = 2, 3, \ldots, k_f$ do
3. Select leaf node $C$ with largest ScatterValue (5),
   and $L \& R :=$ left & right children of $C$ [step (a) in the text].
4. Compute $v_C = g_C(M) \equiv u_C^T(M_C - w_C e_C^T)$
5. For $i \in C$, if $v_i \leq 0$, then assign data sample $i$ to $L$,
   else assign it to $R$ [step (b) in the text].
6. Result: A binary tree with $k_f$ leaf nodes forming a partitioning of the entire data set.

Figure 1: PDDP. $M_C$ is the matrix of data vectors for the data samples in cluster $C$, and $w_C$, $u_C$ are the centroid and principal direction vectors, respectively for $C$.

The clustering via PDDP is a recursive process that operates directly on the matrix $M$. PDDP starts with a single “cluster” encompassing the entire data set and 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 ScatterValue, 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 $C$ consisting of $k$ data samples of attribute values. It places each data sample $x$ in the left or right child of cluster $C$ according to the sign of the linear discriminant function

$$g_C(x) = u_C^T(x - w_C) = \sum_{i \in C} u_i(x_i - w_i),$$

(8)
where \( \mathbf{u}_C, \mathbf{w}_C \) are vectors associated with \( C \) to be determined. If \( g_C(\mathbf{x}) \leq 0 \), the data sample \( \mathbf{x} \) is placed in the new left child, otherwise \( \mathbf{x} \) 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 \( \mathbf{u}_C, \mathbf{w}_C \) associated with the cluster \( C \).

The vector \( \mathbf{w}_C \) is the mean or centroid vector, as defined in (4). The vector \( \mathbf{u}_C \) is the direction of maximal variance, also known as the leading left singular vector for the matrix \( \mathbf{M}_C - \mathbf{w}_C \mathbf{e}^T \). This direction corresponds to the largest eigenvalue of the sample covariance matrix for the cluster. Here \( \mathbf{M}_C \) is the matrix of columns of data samples in cluster \( C \). The computation of \( \mathbf{u}_C \) 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 (see [3]). 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 is summarized in Fig. 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.

5 Matrix Approximation Using Cluster Centers

An approximation to a matrix \( \mathbf{A} \) can be thought of as a system which captures the most “important” information in \( \mathbf{A} \). Obtaining the approximation usually involves a tradeoff among accuracy, the memory space occupied by the approximation, and the time required to obtain the approximation. These three aspects of the approximation are usually in conflict, and must be tailored to fit the desired application. Unlike the recent low-rank approximation methods in [1, 2, 14, 19], this approximation is not used to extract directly the essential concepts that pervade an entire dataset, in which each new “concept vector” is forced to be independent (orthogonal in some cases) of the preceding “concept vectors.” Rather, we are interested in using it only as a low-memory approximation which can be used in place of the original data in the matrix vector products needed by the PDDP algorithm. Hence we can afford to use a faster method to generate the approximation.

One basis for approximation which has proven to be useful is the centroids resulting from a clustering operation. The following description was taken from [8] and has been slightly modified. Say that you have a clustering of a data set with \( k \) clusters. The centroids of the clusters can be gathered into a \( n \times k \) matrix \( \mathbf{C} \) such that:

\[
\mathbf{C} = [\mathbf{c}_1 \mathbf{c}_2 \ldots \mathbf{c}_k].
\] (9)

Given the original matrix \( \mathbf{M} \) representing the data, it is possible to construct an approximation to \( \mathbf{M} \):

\[
\mathbf{M} \approx \mathbf{CZ},
\] (10)

where \( \mathbf{Z} \) is a \( k \times m \) matrix such that, for a given column \( \mathbf{z}_i \) of \( \mathbf{Z} \),

\[
\mathbf{z}_i = \arg \min_{\mathbf{z}} \| \mathbf{x}_i - \mathbf{Cz} \|_2,
\] (11)

where \( \mathbf{x}_i \) is the \( i^{\text{th}} \) column of \( \mathbf{M} \). The direct way to solve for each \( \mathbf{z}_i \) is to use a least-squares approximation. In the context of [8] \( \mathbf{Z} \) was dense, and this was called a concept decomposition. The concept decomposition was designed to be a low-cost alternative to the singular value decomposition.

The amount of memory occupied by the approximation is variable. Assuming a reasonably small number of centroids in \( \mathbf{C} \) and a dense \( \mathbf{M} \), the bulk of the memory will be taken up by \( \mathbf{Z} \). If \( \mathbf{Z} \) is allowed to be dense, then the memory savings will be minimal. The solution is to enforce
a sparsity condition on $Z$, using a small, fixed number of the centroids in $C$ to approximate each original column of $M$.

This approximation is designed to take up as little memory space as possible while reproducing the columns of $M$ with enough accuracy to allow good clustering. It will probably not satisfy any requirements which might normally be imposed on a matrix approximation used in a traditional linear algebraic application, such as the linear independence of the basis vectors.

### 6 Clustering Large Data Sets with Approximations

Recall that our problem is to cluster data sets that are too large to fit into memory. One solution to this problem is to divide the original data set into smaller pieces which will fit into memory, cluster them, and obtain an approximation to each piece of data. Once this is done, the approximations can be gathered into one system and then clustered. A more formal description of the method follows.

A matrix $M$ can be divided into $k_s$ disjoint sections such that:

$$M = [M_1 \ M_2 \ldots \ M_{k_s}].$$

(12)

The partitioning of $M$ is assumed to be virtual or arbitrary (e.g. only the data for one section is in memory at a given time), and the ordering of the columns of $M$ is assumed to be unimportant. Once a section $M_j$ is available, an approximation to $M_j$ can be constructed:

$$M_j \approx C_j Z_j,$$

(13)

where $C_j$ is an $n \times k_c$ matrix of the form (9) and $Z_j$ has the form (10, 11) with $k_c$ rows. Each column of $Z_j$ has at most $k_z$ nonzeros. The centroids in each $C_j$ are obtained through some kind of clustering algorithm.

Once an approximation is available for each section of data, they can be assembled into an approximation of the entire data set $M$:

$$M \approx C_M Z_M,$$

(14)

where

$$C_M = [C_1 \ C_2 \ldots \ C_{k_s}] \quad \text{(an } n \times k_s k_c \text{ matrix)}$$

(15)

and

$$Z_M = \begin{bmatrix}
Z_1 \\
Z_2 \\
\vdots \\
Z_{k_s}
\end{bmatrix},$$

(16)

(a $k_s k_c \times m$ matrix with $k_z$ nonzeros per column). Now that an approximation to $M$ is available, and will fit into memory (because it was designed to fit into memory through judicious choice of the total number of centroids in each $C_j$ and the number of nonzero elements in each $Z_j$), it can be used to cluster the entire original data set. The algorithm for the method described in (12-16) is shown in Fig. 2.

PDDP is an ideal choice of algorithm to cluster the data using this approximation. Central to PDDP is the calculation of the principal direction of the data to determine the splitting hyperplane. The principal direction is calculated using the iterative procedure developed by Lanczos, and this
Algorithm Piecemeal PDDP.

0. Start with a \( n \times m \) matrix \( \mathbf{M} \) of vectors, one vector for each data sample, and a desired number of final clusters \( k_f \). Set the values for \( k_s \) (the desired number of sections), \( k_c \) (the number of clusters produced for each section), and \( k_z \) (the number of centroids used to approximate each data point).
1. Partition \( \mathbf{M} \) into \( k_s \) disjoint sections, \( \mathbf{M}_1 \mathbf{M}_2, \ldots, \mathbf{M}_{k_s} \).
2. For \( j = 1, 2, \ldots, k_s \) do
3. \textbf{Compute} the PDDP tree for the section \( \mathbf{M}_j \) with \( k_c \) clusters.
4. \textbf{Assemble} the \( k_c \) centroids from the leaf clusters into an \( n \times k_c \) matrix \( \mathbf{C}_j \).
5. \textbf{Compute} the \( k_c \times m \) matrix \( \mathbf{Z}_j \) minimizing the quantity \( \| \mathbf{M}_j - \mathbf{C}_j \mathbf{Z}_j \|_F \)
subject to the constraint on the number of nonzero elements \( k_z \) in each column of \( \mathbf{Z}_j \).
6. \textbf{Assemble} the matrices \( \mathbf{C}_M \) and \( \mathbf{Z}_M \) as in (15, 16) in the text, using all the matrices \( \mathbf{C}_j \) and \( \mathbf{Z}_j \) from all passes through steps 2-5.
7. \textbf{Compute} the PDDP tree for the system \( \mathbf{C}_M \mathbf{Z}_M \) with \( k_f \) clusters.
8. \textbf{Result}: A binary tree with \( k_f \) leaf nodes forming a partitioning of the entire data set.

Figure 2: Piecemeal PDDP algorithm.

iterative procedure is based on the formation of the matrix-vector product \( \mathbf{M} \mathbf{v} \). For any given split, the matrix \( \mathbf{M} \) can be replaced by the quantity \( \mathbf{C}_M \mathbf{Z}_M \), resulting in the formation of the product \( \mathbf{C}_M (\mathbf{Z}_M \mathbf{v}) \). Note that by calculating the matrix-matrix-vector product in this order, the product of \( \mathbf{C}_M \mathbf{Z}_M \) never needs to be formed explicitly. The increase in the computational cost shouldn’t be severe, and there won’t be any increase in the memory requirements with respect to standard PDDP.

Many other clustering algorithms would not be able to take full advantage of the memory savings given by \( \mathbf{C}_M \mathbf{Z}_M \). Any method which requires a similarity measure between two data points in order to make a clustering decision would require either that \( \mathbf{C}_M \mathbf{Z}_M \) be formed explicitly (most likely negating much of the memory savings), or that individual columns of \( \mathbf{C}_M \mathbf{Z}_M \) be calculated every time they are needed, which would result in a large increase in computational cost. Hierarchical agglomeration and \( k \)-means are two algorithms which would not be suitable for this approximation, and any derivative algorithms of the two would most likely not be appropriate either.

7 Data

There are three real data sets which will be used to measure the performance of piecemeal PDDP. Two of them are scientific data sets, and one of them is a document data set.

The astronomical data was derived from the Minnesota Automated Plate Scan (APS) [17]. This is a project to 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 sample used consists of 212089 galactic objects, each with 26 attribute values. An attribute might be coordinate information, color, brightness, etc. The attributes were generated by analyzing the blobs in the images, and the values were normalized so they lie on the interval \([0,1]\).

The ISOLET (Isolated Letter Speech Recognition) data was generated by having 150 subjects speak the name of each letter of the alphabet twice. The attributes were extracted from recordings of the speakers, and include contour features, sonorant features, pre-sonorant features, and post-
sonorant features. A total of 617 attributes were extracted from the pronunciation of each letter, and were scaled so they all lie on the interval $[-1.0, 1.0]$. There are a total of 7797 items available when the training and test sets are combined. The data first appeared in [10], and is in the UCI repository [15].

The k1 data set [4] consists of text documents selected from 20 news categories from the YAHOO web site. This data set has been included to demonstrate the effectiveness of the algorithms on document collections that might typically be retrieved from the World-wide Web. The data set consists of 2340 documents spanning 21839 words. The words were stemmed using Porter’s suffix stripping algorithm [11], and the stop words were removed. The document vectors were scaled to unit length, but no other scaling was performed.

<table>
<thead>
<tr>
<th>dataset</th>
<th>astro</th>
<th>islet</th>
<th>k1</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of samples</td>
<td>$m$</td>
<td>212089</td>
<td>7997</td>
</tr>
<tr>
<td>number of attributes per sample</td>
<td>$n$</td>
<td>26</td>
<td>617</td>
</tr>
<tr>
<td>sparsity (percentage nonzeroes)</td>
<td>dense</td>
<td>dense</td>
<td>0.68%</td>
</tr>
<tr>
<td>number of columns in assembled $C_M$ (15)</td>
<td>$k_s k_c$</td>
<td>10000</td>
<td>750</td>
</tr>
<tr>
<td>number of centers in final clustering</td>
<td>$k_f$</td>
<td>2000</td>
<td>150</td>
</tr>
</tbody>
</table>

Figure 3: Datasets and parameter values used for experiments.

8 Experimental Results

The method was evaluated through a series of experiments. The experiments were designed to demonstrate that piecemeal PDPP will give a clustering which is as good as standard PDPP while allowing more data to be clustered due to the reduced memory requirements. In every case, the least-squares problem (11) was solved using the normal equations applied to the $k_z$ closest (in terms of Euclidean distance) centers.

For the first experiments, the number of sections $k_s$ (see Fig. 2) was varied over a range of 1 to 20. The product $k_s k_c$ was fixed, and the value of $k_z$ was 5. Given the different sizes for the datasets, we used different values for the various parameters. These are summarized in Fig. 3. The results shown in the figures were normalized with respect to standard PDPP (Fig. 1) using the same value for $k_f$ as was used in piecemeal PDPP. Recall that each data item is being approximated by $k_z$ centers chosen out of a pool of $k_c$ centers from its own section. As the number of sections grows, the product $k_s k_c$ remains fixed, hence fewer centers are available to approximate each data item in step 5 of Fig. 2. As a result, the quality of the approximations would be expected to go down as the number of sections increases. There are no results for the astronomical data for $k_s = 1$, $k_c = 10000$, since there was not enough memory available to produce 10000 clusters.

The results for the normalized scatter values are shown in Fig. 4(a). In some cases, applying piecemeal PDPP to the entire data set together ($k_s = 1$) unexpectedly improved the quality of the clustering. This might be the result of the data being somewhat “smoothed” by the averaging operation which results from using the centroids to approximate the data. The results are worse as the number of sections increases, which is most likely the result of having fewer centers to choose from when selecting the five closest centers while obtaining the least-squares approximation (step 5 of Fig. 2).

The results for the normalized entropy values are shown in Fig. 4(b). Note that since the astronomical data is not labeled, no entropy values can be calculated. The entropy values again
Figure 4: Results for the scatter (a) and entropy (b) values for a varying number of sections \( k_s, k_z = 5 \) using the parameters in Fig. 3 normalized with respect to standard PDDP (Fig. 1) using the same value for \( k_f \). The astronomical data is unlabeled, so no entropy can be calculated.

indicate a reduction in quality as the number of sections increases, but for the isoelet data the entropy is better than standard PDDP until the number of sections is 5 or more. The results for the k1 document data are not as favorable, but are still comparable with the standard PDDP results. It could be that the method does not perform as well for document data, or it could be that the labeling of the document data was not accurate.

The results for the normalized time values are shown in Fig. 5. The time cost is the one category in which piecemeal PDDP suffers compared with standard PDDP. The expectation from [3] is that PDDP should be approximately linear in the number of samples being clustered, depending on the convergence of the singular value solver. Since the piecemeal algorithm must compute a large collection of intermediate centers, we would expect a corresponding increase in the time cost. The code for standard PDDP has been highly optimized, while the code for piecemeal PDDP is not as mature. There is the additional expense in obtaining more singular values than in standard PDDP, obtaining the least-squares approximations, and in the additional matrix multiplication required for every Lanczos iteration when the final clustering is obtained. These comments apply when the data set is small enough to be processed by the standard algorithm. The piecemeal method can be applied to data sets that are too large to be processed by the standard algorithm, and in these cases the piecemeal method would become competitive with any “out-of-core” variant of the standard method.

In the case of the astronomical data, piecemeal PDDP varies from ten times to fifty-two times slower than standard PDDP. The relative time required for piecemeal PDDP on the other two data sets is not as high, with the k1 data set taking a relatively steady six times longer, and the isoelet data taking from three to at most five times longer.

Since the product \( k_s k_c \) was a fixed value, the size of both \( C_M \) and \( Z_M \) remained constant throughout these experiments. The memory used by the approximation for the astronomical data was about 32 percent of the memory used by the original data. The size of the approximation of the isoelet data was roughly 11 percent of the size of the original data, and the approximation to the k1 data was about 58 percent of the size of the original data. These results seem to indicate
that dense data will have the most memory savings when using this technique, and that the higher
the attribute dimension of a dense data set, the more significant the memory savings.

For the experiments in Figs. 6–7, the number of centroids \( k_z \) used to approximate each data point
was varied from 1 to 26, and the values of the other variables were fixed at \( k_c = 1000, k_f = 2000, k_z = 10 \)
for the astronomical data, \( k_c = 150, k_f = 150, k_z = 5 \) for the isolet data, and \( k_c = 50, k_f = 50, k_z = 5 \)
for the \( k_1 \) data. The values reported in the figures were normalized with respect to standard PDDP
using the same value for \( k_f \) as was used in piecemeal PDDP.

The results for the scatter values are shown in Fig. 6(a). Using more centers to approximate
each data item improved the result for the astronomical and isolet data sets, while the \( k_1 \) data did
not seem to be sensitive to the number of centers used. The most dramatic improvement in scatter
value was for moving from using 1 center to using 2 centers to approximate each data item in the
astronomical data set.

The results for the entropy values are shown in Fig. 6(b). The entropy values vary quite a
bit over the range of the number of centers shown, but are always comparable to those from the
standard PDDP algorithm. In some cases, they are even better. The most dramatic improvement
seems to occur when going from using 1 center to approximate each data item to using 2 centers.

The results for the time taken by the algorithm are shown in Fig. 7(a). As would be expected,
increasing the value of \( k_z \) increases the amount of time needed to compute the approximation and
the final clustering. The time values for the astronomical data were compiled from different runs,
and the different operating conditions, such as available fast memory and user load, influenced the
results. We would expect that under ideal conditions the curve would be more flat between the
data points for \( k_z = 15 \) and \( k_z = 23 \). They indicate a very high time penalty for the astronomical
data as \( k_z \) increases beyond 8 or so. The increase appears to be approximately linear for the isolet
and \( k_1 \) data sets. The isolet data set had the least increase in computation time and a fairly flat
curve. The \( k_1 \) data falls in between the two, requiring more time than the isolet data but still
having a linear increase in cost.

The memory used by the approximations \( C_M Z_M \) normalized with respect to \( M \) is shown
in Fig. 7(b). The values for the \( k_1 \) data are the actual number of nonzeros returned by MATLAB,
while the values for the astronomical and isolet data were calculated experimentally. In all cases,
the amount of memory used by the approximation increases linearly with the number of centers used to approximate each data item. This is expected. The results for the astronomical data show that using more than 17 centers to approximate each data item is a break-even proposition as far as memory is concerned. This is not surprising since the astronomical data has only 26 attributes. Recall that the sparse array containing the matrix $Z_M$ has more overhead per entry than a dense matrix. The islet data saves a significant amount of memory when it is approximated. The k1 data saves some memory, but the effect is not as striking because the original dataset is already very sparse. However, it is interesting to note that even with the sparsity of the original data, it is still possible to obtain entropies as good as the original with less memory.

To test the sensitivity of the method to the ordering of the data, the experiment with $k_s = 10$, $k_c = 1000$, $k_f = 2000$, $k_z = 5$ for the astronomical data, $k_s = 5$, $k_c = 150$, $k_f = 150$, $k_z = 5$ for the islet data, and $k_s = 5$, $k_c = 50$, $k_f = 50$, $k_z = 5$ for the k1 data, was repeated 10 times with a different random ordering of the data set. The results for the scatter values, normalized with respect to standard PDDP using the same $k_f$, are shown in Figure 8(a). The graph seems to indicate that while there is some sensitivity to data ordering, it is not severe. It also seems to indicate that the data already had some ordering present.

A confusion matrix was constructed from a sample run for each data set. A confusion matrix for the astronomical data set with $k_z = 5$, $k_s = 10$, $k_c = 1000$, and $k_f = 2000$, using standard PDDP with $k_f = 2000$ as the ground truth, is shown in Fig. 8(b). A confusion matrix for the islet data with $k_z = 5$, $k_s = 5$, $k_c = 150$, and $k_f = 150$, using standard PDDP with $k_f = 150$ as the ground truth, is shown in Fig. 9(a), and a confusion matrix for the k1 data with $k_z = 5$, $k_s = 5$, $k_c = 50$, and $k_f = 50$, using standard PDDP with $k_f = 50$ as the ground truth, is shown in Fig. 9(b). The rows of the confusion matrix were permuted so that the most similar clusterings are located on the diagonal. As can be seen, the clusterings computed by the two methods are similar.
9 Conclusions

The experiments demonstrate that piecemeal PDDP works for the data sets examined. The performance indicated by the scatter and entropy values does not suffer significantly with respect to standard PDDP when the approximations to the data are used to cluster the original data. The amount of memory taken by the approximation can be varied to suit the application. In general, it appears that as much memory as possible should be used to contain the approximation, since the accuracy of the clustering generally increases with the number of centroids used to approximate each data item.

Piecemeal PDDP appears to be able to compute clusterings of a quality almost as good as that from the standard PDDP, and in some cases better, while using significantly less memory. Piecemeal PDDP does suffer from an increased time cost due to the many intermediate clusters to be computed and the intermediate least squares problems encountered. So the use of the piecemeal method is a trade-off between the memory available and the size of the dataset.

Most of the methods for large data sets involve some kind of sampling of the data. Piecemeal PDDP examines every data point and creates an approximation to every data point. Birch [18] examines every data point as well, but it still uses one vector to represent each data point. Piecemeal PDDP is more flexible in that it can use as many centroids to represent each data point as memory allows. The result is a method which will cluster large data sets to good accuracy in a reasonable amount of time.

References


Figure 8: Part (a) shows the results for the scatter for \(k_z = 5\) with the same parameters as in Fig. 6, normalized with respect to standard PDDP using the same \(k_f\). Part (b) shows the confusion matrix for the astronomical data set for \(k_z = 5, k_s = 10, k_c = 1000,\) and \(k_f = 2000,\) comparing piecemeal PDDP with standard PDDP with \(k_f = 2000.\)


Figure 9: Confusion matrices comparing Piecemeal PDDP with standard PDDP with the same $k_f$. Part (a) shows the confusion matrix for the isolet data with $k_z = 5$, $k_s = 5$, $k_e = 150$, and $k_f = 150$. Part (b) shows the confusion matrix for the k1 data with $k_z = 5$, $k_s = 5$, $k_e = 50$, and $k_f = 50$.


