

Using Low-Memory Representations to Cluster Very Large Data Sets

David Littau
littau@cs.umn.edu

Daniel Boley
boley@cs.umn.edu

Abstract

Many of the algorithms designed to cluster large data sets compute representations of the data which are based on a single vector, without a unique representation of the original data items. We present an extension of Principal Direction Divisive Partitioning which creates a least-squares approximation of the data based on a small number of vectors. We show that the extension can save significant amounts of memory and cluster the data as well as the original method. We also show that in some cases using more than one vector to approximate each data item results in superior quality clusterings.

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

1 Introduction

One common solution when clustering data sets which are too large to fit into memory is to construct approximations to the original data and use the approximations to cluster the data. Usually, a given data item is approximated by a single vector, as in [4, 5, 11]. We present a variant of Principal Direction Divisive Partitioning (PDDP) [2] that approximates each data item using a small number of vectors. For each data item, the closest vectors chosen from an inexpensively produced set of vectors are used to construct a least-squares approximation to the data item. The low-memory representation of the entire original data set appears in factored form as a matrix of approximating vectors and a sparse matrix of multipliers. Using more than one vector to approximate each data item results in a unique representation for each data item, which can increase the clustering accuracy in some cases. The new method can compute clusterings with quality similar to PDDP, while being able to efficiently cluster much larger data sets.

2 Previous Work

The usual methods to approximate large data sets use one vector to approximate many original data items. BIRCH [11] constructs the approximations by examining one data item at a time. A new data item is assigned to an approximation vector based on proximity,

and the approximation vector is updated to reflect the influence of the new data item. If the new data item is too far away from the currently available approximating vectors, a new approximation vector is constructed using the data item. Scatter/Gather [5] works by creating approximations via hierarchical agglomeration, with the cluster centers being used as representatives for the original data in successive clusterings. Multiple passes of the method are used to create the final clustering of the data. The method presented in [4] is a variant of K -means. The approximations are constructed by clustering subsets of the data and using the weighted cluster centers to represent the original data. More of the original data are added to the mix, and K -means is performed on the data and the approximations. Eventually, the result is a clustering of the original data.

The method presented in this work uses more than one vector to approximate each data item. It is similar to the *concept decomposition* [6], which was originally designed as an alternative to the Singular Value Decomposition (SVD) to perform Latent Semantic Indexing (LSI) [1]. The concept decomposition involved clustering the data using a variant of K -means, and then used the cluster centers as the basis of a least-squares approximation to the data. The method we present uses PDDP to cluster the original data. We then use a small number of the cluster centers to construct an approximation to each data item, rather than using all of the available cluster centers to approximate a given data item. This technique can save a significant amount of memory with respect to the concept decomposition, yet the representation of the data is sufficient to create a reasonable clustering of the original data.

3 Method

The method we developed is a variant of PDDP. PDDP is a fast, scalable clustering algorithm, but like many other clustering algorithms it requires the data set to be small enough to fit into memory for efficient computation. Piecemeal PDDP (PMPDDP) extends PDDP to large data sets that cannot fit into memory. PMPDDP creates a low-memory representation of the

original data set that will fit into memory, and then clusters the low-memory representation using PDDP.

First, we describe the technique we use to obtain a low-memory approximation of a data set. Then, we show how we apply the technique to a data set which will not fit into memory at once.

3.1 Matrix Approximation Using Cluster Centroids Our approximation technique is based on the concept decomposition [6]. The concept decomposition uses cluster centroids to form a basis for a least-squares approximation to the original data. Our method is similar, except that we limit the number of centroids which participate in the least-squares approximation to a given data item.

Suppose we have an $n \times m$ matrix \mathbf{A} of data samples. We partition \mathbf{A} into k_c clusters and compute the k_c centroids of the clusters. These centroids are collected into a $n \times k_c$ matrix $\mathbf{C}_A = [\mathbf{c}_1 \ \mathbf{c}_2 \ \dots \ \mathbf{c}_{k_c}]$, which is the matrix of representative vectors used in the approximation. We use the PDDP method to compute the clustering of \mathbf{A} and therefore obtain \mathbf{C}_A , but in principle, any clustering method could be used here.

We use \mathbf{C}_A to compute the approximation $\mathbf{A} \approx \mathbf{C}_A \mathbf{Z}_A$, where \mathbf{Z}_A is a $k_c \times m$ matrix. Each column \mathbf{z}_i of \mathbf{Z}_A approximates the corresponding column \mathbf{a}_i of \mathbf{A} using a linear combination of the vectors in \mathbf{C}_A .

In approximating each column \mathbf{a}_i , we use only a small number (k_z) of centroid vectors, so that each \mathbf{z}_i has only k_z nonzero entries, where typically $k_z \ll k_c$. For example, to approximate \mathbf{a}_i , we choose the k_z columns in \mathbf{C}_A which are closest in Euclidean distance to \mathbf{a}_i and implicitly collect them into an $n \times k_z$ matrix \mathbf{C}_i . Then the nonzero entries in the column \mathbf{z}_i are obtained by solving for the k_z -vector $\hat{\mathbf{z}}_i$:

$$(3.1) \quad \hat{\mathbf{z}}_i = \arg \min_{\mathbf{z}} \|\mathbf{x}_i - \mathbf{C}_i \mathbf{z}\|_2.$$

Using the k_z closest centroids to approximate each \mathbf{a}_i instead of computing the approximation using all the centroids in \mathbf{C}_A creates the opportunity to save memory with respect to a concept decomposition. When $k_z = k_c$, this technique is essentially identical to the concept decomposition [6], but with $k_z \ll k_c$, this can result in significant savings in memory.

If the k_z vectors in \mathbf{C}_i are linearly independent, we use the normal equations with the Cholesky decomposition to solve the least-squares problem. Otherwise, we would have to use the more expensive SVD to get the least-squares approximation of the data item. Even though there has been no attempt to create orthogonal basis vectors, in the majority of cases the normal equations give a satisfactory approximation.

3.2 Constructing a Low-Memory Representation of a Matrix Now that we have a technique to construct a low memory representation of a data set, we need to apply it to larger data sets. The basic idea of what follows is to break the original data set into smaller pieces, construct a low-memory representation of each piece, and then assemble the individual representations into a single factored representation of the entire original data set. This single factored representation is then used to compute a clustering of the entire dataset as a single unit.

We start with an $n \times m$ matrix \mathbf{M} of data, such that \mathbf{M} will not fit into memory at once. We seek the single factored representation \mathbf{CZ} such that

$$(3.2) \quad \mathbf{M} \approx \mathbf{CZ},$$

where \mathbf{C} and \mathbf{Z} will fit into memory and can be used to cluster the data in \mathbf{M} .

\mathbf{C} and \mathbf{Z} are constructed in a piecemeal fashion. \mathbf{M} is divided into k_s disjoint *sections*

$$(3.3) \quad \mathbf{M} = [\mathbf{M}_1 \ \mathbf{M}_2 \ \dots \ \mathbf{M}_{k_s}],$$

such that each section \mathbf{M}_j of \mathbf{M} will fit into memory. This partitioning of \mathbf{M} is virtual since we assume only one section \mathbf{M}_j will be in memory at any given instance. We also assume that the ordering of the columns of \mathbf{M} is unimportant. We can now construct an approximation

$$(3.4) \quad \mathbf{M}_j \approx \mathbf{C}_j \mathbf{Z}_j$$

for each section \mathbf{M}_j of \mathbf{M} using the technique from §3.1.

After computing the approximation (3.4) for each section of data, they can be assembled into the two-matrix system

$$(3.5) \quad \begin{aligned} \mathbf{C} &= [\mathbf{C}_1 \ \mathbf{C}_2 \ \dots \ \mathbf{C}_{k_s}] \\ \mathbf{Z} &= \begin{bmatrix} \mathbf{Z}_1 & & & \mathbf{0} \\ & \mathbf{Z}_2 & & \\ & & \ddots & \\ \mathbf{0} & & & \mathbf{Z}_{k_s} \end{bmatrix}, \end{aligned}$$

where \mathbf{C} has dimension $n \times k_s k_c$ and \mathbf{Z} has dimension $k_s k_c \times n$. The parameters used to construct the low-memory representation are summarized in Table 1. The PMPDDP algorithm, shown in Figure 1, constructs the representation (3.2) as just described and then uses (3.2) to compute a final clustering of the entire dataset as a single unit.

The amortized cost of performing a PMPDDP clustering is shown in Table 2, under the assumption that PDDP is used to both construct the matrix \mathbf{C} and to cluster the approximation (3.2).

The piecemeal clustering just computed does not constitute a good clustering of the entire dataset, since

parameter	description
m	total number of data items
n	number of attributes per data item
γ	fill fraction for the attributes
k_s	number of sections
k_d	number of data items per section
k_c	number of centroids per section
k_z	number of centroids approximating each data item
\tilde{k}_z	$\min\{k_z, \log_2(k_c)\}$
k_f	number of final clusters

Table 1: Definition of the parameters used in PMPDDP.

Operation	Cost
Obtaining \mathbf{C}	$c_2 m \gamma n \log_2(k_c)$
Computing \mathbf{Z}	$m \left(k_c k_z \gamma n + k_c \tilde{k}_z \right) + k_z^2 n + \frac{1}{3} k_z^3$
Clustering \mathbf{CZ}	$c_2 (k_f - 1) k_s k_c n + c_2 \log_2(k_f) k_z m$

Table 2: Collected costs of producing a PMPDDP clustering, with PDDP being used to obtain the columns of \mathbf{C} . See Table 1 for a definition of the parameters.

similar items in different sections are not tied together in any way. However, the low memory \mathbf{CZ} decomposition just computed can be used to compute a valid clustering of the entire dataset. The PDDP method has a unique advantage when clustering this low-memory representation. Most clustering methods (e.g. K -means) depend on computing explicit distances or similarity measures between individual data samples, and this would require that the \mathbf{C} and \mathbf{Z} be multiplied together to “reconstitute” the columns of original data matrix, either once or a few at a time repeatedly. Instead, PDDP computes its separating hyperplanes using a Lanczos-based solver, which needs only matrix-vector products of the form $\mathbf{M}\mathbf{v}$ for many vectors \mathbf{v} . With the approximation (3.2), we can compute the product as $\mathbf{M}\mathbf{v} \approx \mathbf{C}(\mathbf{Z}\mathbf{v})$ instead of $(\mathbf{CZ})\mathbf{v}$, thus avoiding ever explicitly reconstituting the columns of \mathbf{M} .

4 Experiments

We selected four real data sets to measure the performance of PMPDDP. They are summarized in Table 3.

We compared a typical PMPDDP clustering to an ordinary PDDP clustering with the same value for k_f chosen as in Table 3. The results are shown in Table 4 and illustrated for various k_z in Figure 2. In every case, the clustering quality as measured by entropy is improved when using PMPDDP with $k_z = 2$ or 3. The memory savings is striking for the datasets

Algorithm PMPDDP.

0. **Start** with a $n \times m$ matrix \mathbf{M} , where each column of \mathbf{M} is a data item, and set the values for k_s , k_c , k_z , and k_f (see Table 1).
1. **Partition** \mathbf{M} into k_s disjoint sections, $|\mathbf{M}_1 \mathbf{M}_2, \dots, \mathbf{M}_{k_s}|$.
2. **For** $j = 1, 2, \dots, k_s$ **do**
3. **Compute** the PDDP tree for the section \mathbf{M}_j with k_c clusters.
4. **Assemble** the k_c centroids from the leaf clusters into an $n \times k_c$ matrix \mathbf{C}_j .
5. **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. **Assemble** the matrices \mathbf{C} and \mathbf{Z} as in (3.5) in the text, using all the matrices \mathbf{C}_j and \mathbf{Z}_j from all passes through steps 2-5.
7. **Compute** the PDDP tree with k_f clusters for the entire system \mathbf{CZ} .
8. **Result:** A binary tree with k_f leaf nodes forming a partitioning of the entire data set.

Figure 1: PMPDDP algorithm.

which are originally dense, namely the isolet and forest cover data, and less striking for the datasets which are already sparse to begin with, such as the document datasets k1 and reuters. However, a larger document data set would probably see a higher percentage of memory savings, since the number of possible attributes in document data sets is limited by the number of words in the dictionary, which often levels off with increasing numbers of documents. As expected, PMPDDP is more expensive than PDDP. However, PMPDDP has the ability to cluster data sets which PDDP cannot cluster without the added expense of memory paging.

To illustrate the effect of varying one of the PMPDDP parameters, we varied k_z , and fixed all the other parameters to the values given in Table 3. The parameter values used are representative, and were not chosen to give optimal results. The results for the entropies for a PDDP clustering of the different low-memory representations are shown in Figure 2. The results are normalized with respect to a PDDP clustering of the original data set. The entries in the graph corresponding to 0 centroids are a PDDP clustering of a low-memory representation using the centroid in \mathbf{C} closest to a data item to approximate that data item.

For the data examined, the results are better if the

dataset	isolet	k1	reuters	forest
m	7997	2340	9494	581012
n	617	21839	19968	54
categories	26	20	66	7
γ	dense	0.68%	0.20%	dense
k_s	5	5	5	5
k_c	150	50	100	500
k_z	5	5	5	1
k_f	150	50	100	500

Table 3: Datasets and parameter values used for PMPDDP experiments. The isolet data is from Murphy and Aha [9], the k1 document data is from [3], the reuters data is the standard reuters data set [8] selecting the items which had only one topic assigned to them, and the forest data is from [7] with the binary attributes converted to continuous attributes.

dataset	isolet	k1	reuters	forest
PDDP entropy	.814	.982	.799	.689
PMPDDP entropy	.810	.960	.787	.672
PDDP time	34.41	14.19	17.63	196.01
PMPDDP time	66.20	70.62	137.99	476.83
CZ memory % of M	10.9	57.2	68.4	4.13

Table 4: Comparison of PDDP and PMPDDP clusterings for the data sets and parameter values shown in Table 3.

closest centroid is used to represent the original data item, rather than computing a least-squares approximation with $k_z = 1$. For the k1, isolet and reuters data, using more than one centroid to approximate each data item gives better clustering results than using just one centroid, but no advantage was found for using more than 6 centroids, except for the k1 data, where using 9 centroids gave the lowest entropy. In contrast, the forest cover data is best served by using the closest centroid to approximate the data. The forest cover data set has only 7 categories, and most of the attributes were converted from discrete values. Many of the original data items are very likely to be quite close together, so using the closest centroid probably results in a very accurate representation of the original data items.

4.1 Application with K -means The low-memory representation was designed to be clustered using PDDP, since PDDP does not require that the data be explicitly reconstructed to cluster the data. However, if it is feasible to reconstruct a small amount of data

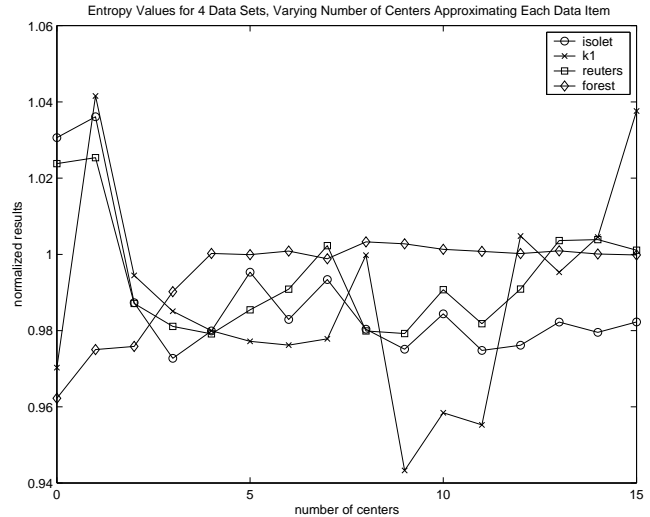


Figure 2: Entropy values for the data sets shown in Table 3 with varying k_z (see Table 1), fixing the other PMPDDP parameter values to those shown in Table 3. The entries shown for the “fictional” value $k_z = 0$ are for a PMPDDP clustering in which each data item is represented simply by the nearest centroid in \mathbf{C} . The results are normalized with respect to an ordinary PDDP clustering of the original data set with the same k_f .

at any given time (at the cost of having to repeat this multiple times), it is possible to use the low-memory representation in other clustering algorithms. K -means is an ideal candidate.

We implemented a version of K -means applicable to the low-memory approximation, such that only 50 columns of the original data were re-constructed at any given time. In the interest of saving time, the algorithm was terminated after 40 iterations for all data sets considered. The smaller data sets usually converged before 40 iterations, but the larger data sets did not. In order to minimize the effect of the local-optimality quality of K -means, 30 random start trials were performed, and the results were averaged. Table 5 shows the results for a K -means clustering of the original data \mathbf{M} and the low-memory representation \mathbf{CZ} constructed using the parameters shown in Table 3. As can be seen, the cost of K -means rose at a higher rate than PMPDDP, but provided a better clustering for the data sets examined except for the k1 document data. K -means usually performed better when using the original data as opposed to the low-memory representation when clustering. However, standard K -means has the limitation of requiring all the data to fit into memory for efficient clustering, so using the low-memory

dataset	isolet	kl	reuters	forest
K -means on \mathbf{M} entropy	.523	1.01	.668	.656
K -means on \mathbf{CZ} entropy	.653	.949	.758	.659
K -means on \mathbf{M} time	91.31	34.65	92.17	5156.22
K -means on \mathbf{CZ} time	89.61	356.73	1171.26	4436.21

Table 5: Comparison of K -means clusterings on the original data \mathbf{M} and on the approximation \mathbf{CZ} , where \mathbf{CZ} was constructed using the parameters shown in Table 3. K -means was run for 40 iterations, and the results were averaged over 30 trials.

representation in the context of K -means is a viable option.

Previous results [10] have indicated that using a PDDP clustering to initialize K -means can give a good K -means clustering without requiring restarts. We did some additional experiments and confirmed that using a PMPDDP clustering to initialize a K -means clustering of the low-memory approximation will give better than average clusterings in a much shorter time than random-start K -means.

5 Conclusion

Approximations to the data are often constructed as an aid when clustering large data sets. The usual solution is to create vectors which resemble cluster centroids, and then assign each original data item to the closest vector. These approximations are either created during a clustering process, or used to cluster the data once they are constructed.

We presented PMPDDP, which is a clustering algorithm which uses more than one vector to approximate each data item. The data were divided into small pieces, and an approximation to each piece was constructed using the centroids from a clustering of the piece of data. Each data item is examined during the process which computes the low-memory representation, and no data items are removed from consideration. The approximations were then gathered and clustered at once to produce a clustering of the entire original data set.

PMPDDP was able to cluster the data to greater accuracy than PDDP for the data sets examined. The low-memory representation of the original data saved a significant amount of memory while still providing a unique representation for each data item. In most cases, using the low-memory representation of the data resulted in a better clustering than using the closest

centroid to approximate each data item.

We also demonstrated that the low-memory representation can be used to find a K -means clustering of the data. Given sufficient time and random restarts, K -means found a better clustering than PMPDDP. However, the combination using PMPDDP clusters as the starting point for K -means can produce clusterings either equal to or superior than K -means on its own in less time and without requiring random restarts.

6 Acknowledgments

This research was partially supported by NSF grant IIS-0208621.

References

- [1] M. W. Berry, S. T. Dumais, and Gavin W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37:573–595, 1995.
- [2] D.L. Boley. Principal Direction Divisive Partitioning. *Data Mining and Knowledge Discovery*, 2:325–344, 1998.
- [3] 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.
- [4] P. S. Bradley, Usama M. Fayyad, and Cory Reina. Scaling clustering algorithms to large databases. In *Knowledge Discovery and Data Mining*, pages 9–15, 1998.
- [5] Douglass R. Cutting, Jan O. Pedersen, David Karger, and John W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 318–329, 1992.
- [6] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143–175, 2001.
- [7] S. Hettich and S. D. Bay. The UCI KDD archive, 1999. kdd.ics.uci.edu/.
- [8] D. Lewis. Reuters-21578. <http://www.research.att.com/~lewis>, 1997.
- [9] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases, 1994. www.ics.uci.edu/~mllearn/MLRepository.html.
- [10] S. M. Savaresi and D. L. Boley. Bisecting k -means and PDDP: a comparative analysis. Technical Report 00-048, University of Minnesota Department of Computer Science, Minneapolis, MN, 2000.
- [11] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.