Outlier Detection Using SemiDiscrete Decomposition

S. McConnell and D.B. Skillicorn
{mcconnell,skill}@cs.queensu.ca

December 2001
External Technical Report
ISSN-0836-0227-
2001-452

Department of Computing and Information Science
Queen's University
Kingston, Ontario, Canada K7L 3N6

Document prepared January 11, 2002

Abstract

Semidiscrete decomposition (SDD) is usually presented as a storage-efficient analogue of singular value decomposition. We show, however, that SDD actually works in a completely different way, and is best thought of as a bump-hunting technique; it is extremely effective at finding outlier clusters in datasets. We suggest that SDD's success in text retrieval applications such as latent semantic indexing is fortuitous, and occurs because such datasets typically contain a large number of small clusters.

Outlier Detection Using SemiDiscrete Decomposition

S. McConnell and D.B. Skillicorn
{mcconnell,skill}@cs.queensu.ca

Abstract: Semidiscrete decomposition (SDD) is usually presented as a storage-efficient analogue of singular value decomposition. We show, however, that SDD actually works in a completely different way, and is best thought of as a bump-hunting technique; it is extremely effective at finding outlier clusters in datasets. We suggest that SDD's success in text retrieval applications such as latent semantic indexing is fortuitous, and occurs because such datasets typically contain a large number of small clusters.

1 Introduction

A dataset in tabular form, with n rows representing *objects*, and m columns representing *attributes*, has a natural geometric representation. Each object can be considered as a point in the m-dimensional space spanned by the attributes. A common data mining approach is to *cluster* these points as a way to resolve structure in the dataset. For example, objects that are similar may fall in the same cluster.

When m is large, this geometrical insight is not as useful as it might be because high-dimensional spaces are hard to work with. For example, the distance from a point to its nearest and farthest neighbour tends to be almost the same. Hence, although many clustering techniques are known, they are not necessarily effective as m becomes large (say > 15).

Singular Value Decomposition (SVD) [5] is a dimension-reduction technique that can be used to generate a low-dimensional representation of a high-dimensional dataset. Clusters may be immediately apparent in this representation, or it may be used as input to other clustering techniques. We present SVD in some detail because it is, in some ways, analogous to semidiscrete decomposition.

SVD transforms an m-dimensional space into a new m-dimensional space whose axes have two useful properties: they are orthonormal, and there is an ordering of the axes such that the variation in the original data is concentrated, as far as possible, along the earlier axes. Hence ignoring dimensions associated with later axes provides a faithful representation of the original data in a lower dimensional space. SVD has been used successfully, under the name *latent semantic indexing* [1, 2], for information retrieval in text. In this application, the early dimensions capture much of the information about the correlated use of terms, and retrieval performance is very strong.

Given an $n \times m$ matrix A with $n \geq m$, the singular value decomposition of A is

$$A = U \Sigma V^T$$

where U is $n \times m$, Σ is $m \times m$, and V is $m \times m$. U and V are orthonormal, and Σ is a diagonal matrix whose elements are a set of non-negative, decreasing singular values, σ_1, σ_2, ..., σ_r (where r is the rank of A, $r \leq m$). The rows of U represent coordinates of the corresponding rows of A in a space spanned by the columns of V. Symmetrically, the rows of V represent the coordinates of the corresponding columns of A in a space spanned by the columns of U. A rank k approximation to A can be computed by multiplying the matrix consisting of the first k columns of U, the upper left $k \times k$ submatrix of Σ and the first k rows of V^T.

The *semidiscrete decomposition* (SDD) [9, 10] was developed for image compression, but has been used more often as a substitute for latent semantic indexing [6–8]. It is usually presented as a weak analogue of SVD, in which the axes of the transformed space are no longer orthonormal, and the coordinates of points in the transformed space are taken only from the set $\{\text{-}1, 0, 1\}$. The transformation itself is a mixed programming optimization problem. Its complexity is $(n+m)^3$, which makes exact solutions feasible for some data mining problems. There is also a heuristic approximation algorithm which is described below. The benefit of SDD over SVD is usually considered to be the compactness of the representation of the transformed space, since each coordinate can be stored in 2 bits.

Given a matrix A, the semidiscrete decomposition of A of dimension k (now unrelated to the rank of A) is

$$A_k \;=\; X_k \, D_k \, Y_k$$

where the entries of X_k and Y_k are from $\{\text{-}1, 0, 1\}$, and D_k is a diagonal matrix. X is $n \times k$, D is $k \times k$, and Y is $k \times m$. We drop the subscripts if they are not important.

Let x_i be the ith column of X, d_i the ith diagonal element of D, and y_i the ith row of Y. The standard algorithm for computing the SDD generates a new column, diagonal element, and row on each step. Let A_0 by the $n \times m$ matrix of zeroes. The algorithm is, for each step i:

1. Subtract the current approximation, A_{i-1}, from A to get a residual matrix R_i.

2. Find a triple (x_i, d_i, y_i) that minimizes

$$\| \; R_i - d_i x_i y_i \; \|^2 \qquad (*)$$

   where x_i is $n \times 1$ and y_i is $1 \times m$. The standard algorithm uses the following heuristic:

   (a) Choose an initial y_i.

   (b) Solve $(*)$ for x_i and d_i using this y_i.

   (c) Solve $(*)$ for y_i and d_i using the x_i from the previous step.

   (d) Repeat until some convergence criterion is satisfied.

3. Repeat until $i = k$.

Matlab and C code for this algorithm is available from `www.cs.umd.edu/users/oleary/SDDPACK/`.

The rows of X are a description (really the coordinates) of an object in the space defined by the new axes described by Y. Therefore, if we divide the objects according to whether they have a -1, 0, or 1 in the first column of X, we have separated them into three classes. The second and subsequent columns can be used to further subdivide the objects, producing a ternary decision tree structure. Note that, unlike conventional decision trees, this is an *unsupervised* process – we have found clusters in the original data. A further step is required to examine the leaves of this tree to see, for example, whether they contain a homogeneous grouping of related objects. However, we have not discussed the order in which the axes (of Y) are chosen so, while we know that the dataset can be partitioned into subsets, we do not know which of these partitions are most significant. This turns out to be a difficult question, which we address in the next section.

2 What SDD is doing

We illustrate the way in which SDD constructs axes of the transformed space using a small matrix with an obvious structure. Suppose that

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The (first 5 columns of the) X matrix produced by SDD are

$$X = \begin{bmatrix} 1 & 0 & -1 & 0 & -1 \\ 1 & 0 & -1 & 0 & -1 \\ 1 & 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 0 & -1 \\ 1 & 0 & -1 & 0 & -1 \\ 1 & 0 & -1 & 0 & -1 \\ 1 & 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 0 & -1 \end{bmatrix}$$

and the first 5 columns of (the transposed matrix) Y are

$$Y^T = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

The values of D are

$$D = \begin{bmatrix} 1.0625 \\ 0.9375 \\ 0.058594 \\ 0.058594 \\ 0.0036621 \end{bmatrix}$$

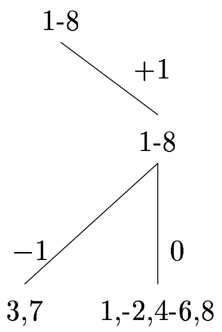

Figure 1: A decision tree based on the example showing partitioning by row number (i.e. object)

The decision tree that corresponds to the matrix X is shown in Figure 1. Unlike a conventional decision tree, this one is ternary. It also does not necessarily contain every possible branch – in other words, for some new data, the only possible classification implied by this decision tree is 'unknown'.

Consider the approximation matrices that are subtracted from A at each stage of the algorithm. On the first step, this matrix is the product $d_1x_1y_1$, where x_1 is the first column of X and y_1 is the first column of Y^T above. The product of x_1 and y_1 is

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

and d_1 is 1.0625, so the matrix that is subtracted from A is the matrix

$$\begin{bmatrix} 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 \\ 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 \\ 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 \\ 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 \\ 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 \\ 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 \\ 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 \\ 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 & 1.0625 \end{bmatrix}$$

In other words, if the matrix is represented in a side view as a histogram as in Figure 2, then the effect of the subtraction has been to remove a slice from the original matrix, leaving four positive peaks and the rest of matrix slightly negative.

On the second step, the matrix that is subtracted is given by the product of x_2 and y_2,

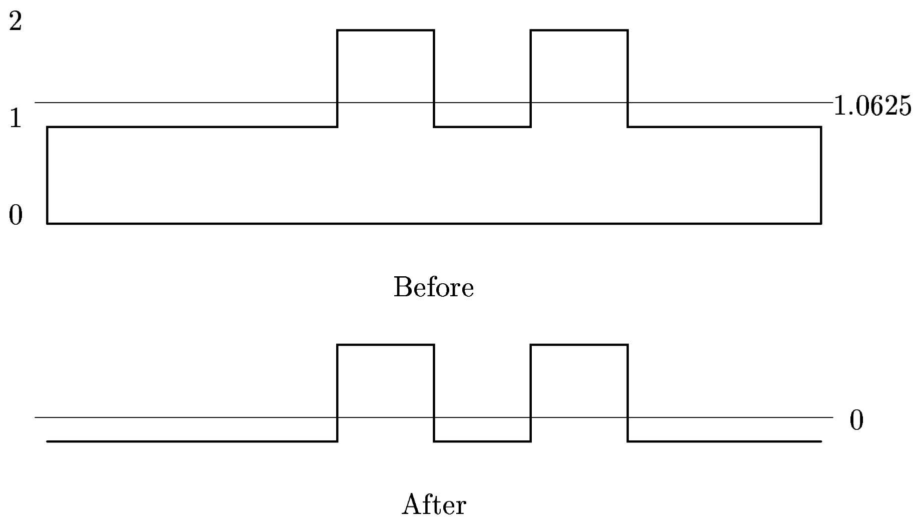

Figure 2: A side view of A showing the effect of the first subtraction

which is

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In other words, this product picks out the shape of the remaining peaks. Note also that the value of d_2 is 0.9575, which is exactly the height of the peaks. The effect of the subtraction of the second matrix is to leave a matrix that is almost filled with small negative values.

We can see from this example that the general effect of SDD is to find regions of the matrix in which the magnitude of the values is relatively large. In fact, it is the *volume*, in the sense of Figure 2, that determines which region is selected, since both the magnitude of the values and the size of the region over which they occur are used. SDD is a form of *bump hunting* – on each step, it finds a region of the dataset that sticks out above (below) the rest of the data when values are considered as heights. The product of x_i and y_i selects the position of the region, while d_i selects the height of the bump. (This is slightly simplified because SDD can also remove bumps with *negative* correlation with a selected pattern of rows and columns.) The use of the 2-norm in the objective function being maximized means that height has more effect on bump selection than the size of the dataset covered by the bump. This means that the algorithm tends to first remove high, but local, bumps in preference to flatter but more widespread bumps. In other words, it tends to find outlier clusters in the data. This explains the order in which axes are chosen by SDD.

Given this discussion, it seems odd that the first bump removed in the example above is a large flat one, rather than the more localized bump that is removed second. The reason is that the localized bump is not very high, so the number of locations of the large flat bump overwhelm the height of the small number of locations in the smaller bump. When the height of the peaks are increased, then the localized bump is removed first. For example, if we start with this matrix:

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 9 & 1 & 9 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 9 & 1 & 9 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

then the X matrix becomes

$$X = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & -1 & 1 & -1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & -1 & 1 & -1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

and Y becomes

$$Y = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

with the values of D

$$D = \begin{bmatrix} 9 \\ 0.9375 \\ 0.9375 \\ 0.058594 \\ 0.058594 \\ 0.0036621 \end{bmatrix}$$

It is clear that now the peaks are removed first, and the large flat slice from the bottom second. It is important to note that the order in which slices are removed matters – if the flat slice had been removed first, its corresponding d value would have been 2 (= 128/64, the average value of the matrix) and the second d value would have been 7.

Although SDD was originally developed as a storage-efficient representation approximating SVD, these examples make it clear that it is actually computing something quite different. When the natural structure of a dataset is many small clusters then SVD and SDD will tend

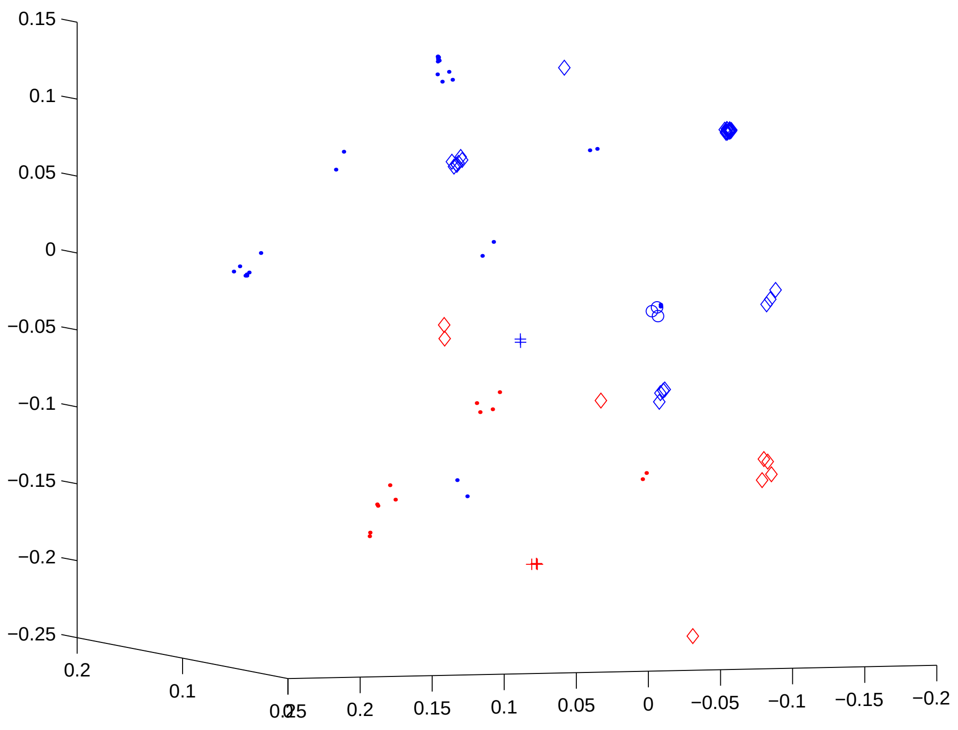

Figure 3: Plot of sparse clusters, position from SVD, shape (most significant) and colour from SDD

to produce similar results: SVD because it captures a faithful low-dimensional representation of the cluster structure of the data, but SDD because each cluster looks like a small bump. This is shown in Figure 3 which uses a dataset typical of text retrieval applications – many zeroes, the remaining values small positive integers.. Here the position of points (in 3 dimensions) is determined by SVD but the shape and colour used to render them is determined by SDD. It is clear that each cluster (in the SVD sense) is homogeneously labelled by SDD. Hence the two techniques agree.

The critical fact is that any congruence between outcomes is a form of coincidence, since each technique works in a completely different way. This explains both the success of SDD as a substitute for SVD in latent semantic indexing (because document-text matrices tend to contain many small clusters) and the lack of success of SDD we had experienced in other data mining applications. SDD is an *outlier detector* and so will tend to emphasize the most unusual patterns in a dataset – using it as if it were an analogue of SVD leads to frustration.

The heuristic embedded in the basic SDD step leads to two problems, one of them solvable, the other not. The algorithm is quite sensitive to the initial choice of y_i. This means that it does not always find the largest possible slice to remove from the matrix at each step. Hence later steps can find a large slice that was missed on previous steps. It is therefore possible that the values of D are not always decreasing.

We apply the following modification to the algorithm. After the X, Y, and D matrices have been computed, we do the following:

1. Form the product of the d_is with the number of non-zero entries in the corresponding columns of Y.

2. Sort the columns of X, elements of D and rows of Y into decreasing order of the products from the first step.

3. Form a decision tree using the new arrangement of the columns of X.

This has the effect of reordering the slices so that those with the largest volume appear first, in other words, the strongest outliers appear closest to the top of the decision tree.

The second problem occurs because the height of a slice removed depends on the current contents of the matrix, which depends on the order in which previous slices were removed. Reordering at the end cannot therefore reproduce exactly the effect of having chosen a different removal order during the algorithm's execution. The problem occurs because the height of a slice is determined by the *average* height of the locations that will be removed. In the second example above, the peaks of height 9 are completely removed at the first step. However, if the matrix is changed slightly like this:

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 9 & 1 & 8 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 9 & 1 & 9 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

then the slices are removed in exactly the same places and orders, but the values of D are now

$$D = \begin{bmatrix} 8.75 \\ 0.9375 \\ 1.6875 \\ 0.51562 \\ 0.51562 \end{bmatrix}$$

in other words, slightly less of the peaks is removed. In some fundamental sense, the way in which the d_i are computed prevents the original matrix from being partitioned as cleanly as it might otherwise be. There seems to be no simple solution to this problem.

3 Examples of outliers discovered by SDD

We now give some examples illustrating how SDD finds outliers, and how its clusters relate to those produced by SVD.

In a geochemical dataset containing 1670 sample points measuring the concentration of 33 elements, SDD produced the decision tree shown in Figure 4. The outlier clusters that appear near the top of the tree are:

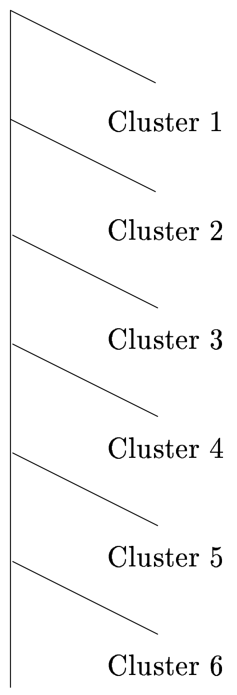

Figure 4: The decision tree for a geochemical dataset

Cluster 1: a cluster of size 3 containing high concentrations of Ce, Hf, La, Sr, Th, and Zr – these appear to be samples from pegmatites, which are of interest because they may contain gemstones;
Cluster 2: a cluster of size 2 containing high concentrations of Co, Cr, and Ni – these suggest mafic rocks and perhaps the presence of a nickel or platinum ore body;
Cluster 3: a single sample containing high concentrations of Cd and Sb – this probably represents a measurement problem since concentrations of cadmium are notoriously hard to measure accurately;
Cluster 4: two samples containing high concentrations of Sn;
Cluster 5: a cluster of size 8 containing high concentrations of Cu, Pb, and Zn – this represents typical sulphide mineralization;
Cluster 6: a cluster of size 3 containing all the samples in the dataset with high concentrations of gold;

These clusters appear to have some inherent significance. In experiments using a number of other techniques (neural networks, genetic algorithms, decision trees, Autoclass, k-means, and support vector machines) no other technique was able to find outlier clusters such as these in such a complex and highly correlated dataset.

A dataset containing observed properties of galaxies, with 8 attributes and 460 rows is shown in Figure 5. Here the positions of the points are from an SVD and the colour-coding from SDD. The resulting SDD tree is:

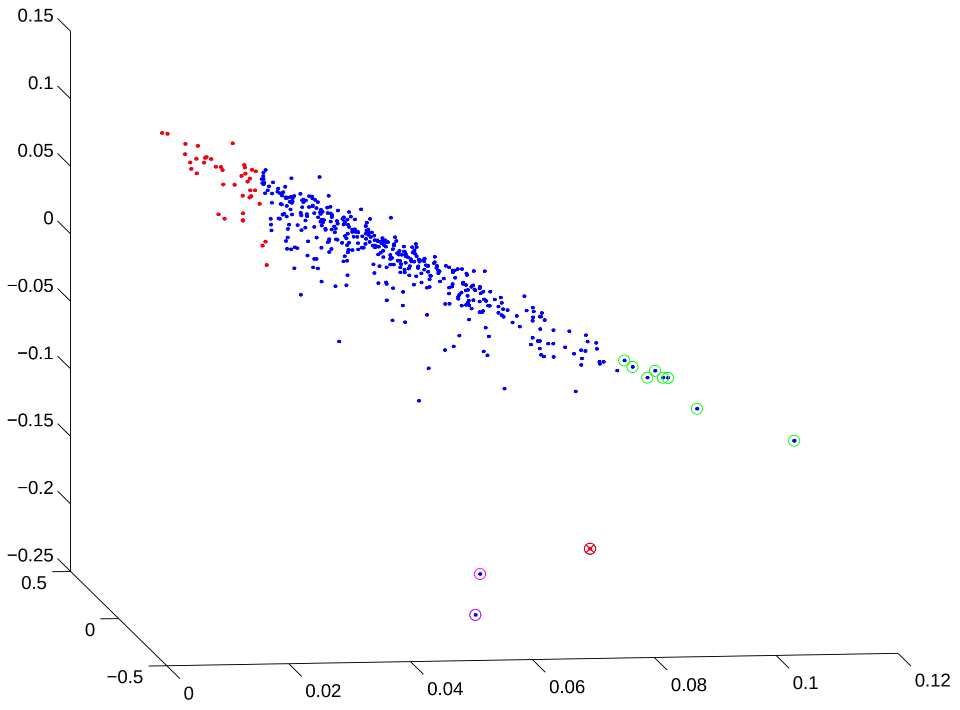

Figure 5: Top-level outlier classes in a galactic dataset

0				41 red points
1	1			9 green points
	0	1	0	103 blue points
			1	1 red cross point
		0	1	162 blue points
			−1	2 magenta points
		−1	1	142 blue points

It is clear that the visible outliers in the plot have been captured by the SDD decision tree. More strikingly, the SDD decision tree has partitioned the single cluster generated by SVD into a set of subclusters reflecting finer detail. Figure 5 makes the distinction between SVD and SDD clear – points near a colour boundary will be interpreted as very similar by any technique based on SVD but as completely dissimilar using SDD.

It is difficult to validate the results given by either technique in astrophysical terms. However, at least some of the noticeable outliers are in fact unusual galaxies.

4 Related work

Friedman and Fisher [4] introduced PRIM, a technique for bump hunting that seems to be quite effective. However, it is a top-down technique that finds bumps by removing slices in one dimension at a time. In contrast, SDD finds bumps using a heuristic approximation to a mixed programming problem.

Rule-based techniques can also be considered as finding bumps (regions of the dataset with high confidence but small support). Some of these techniques (i.e. association rules) use exhaustive search, and therefore do not necessarily scale well. Others use various heuristics (e.g. beam search) to reduce the computational cost.

There are also a class of methods based on SVD that result in decision trees, notable Boley's *principal direction divisive partitioning* [3], which separates a dataset based on its variation in the direction of the first singular value axis, and then repeats this process on each partition of the dataset independently.

We conclude by comparing the classifications produced by SVD, SDD, and PDDP on a small dataset with an obvious structure:

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

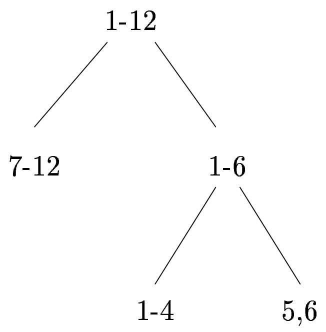

Figure 6: The decision tree produced by PDDP, partitioned by rows

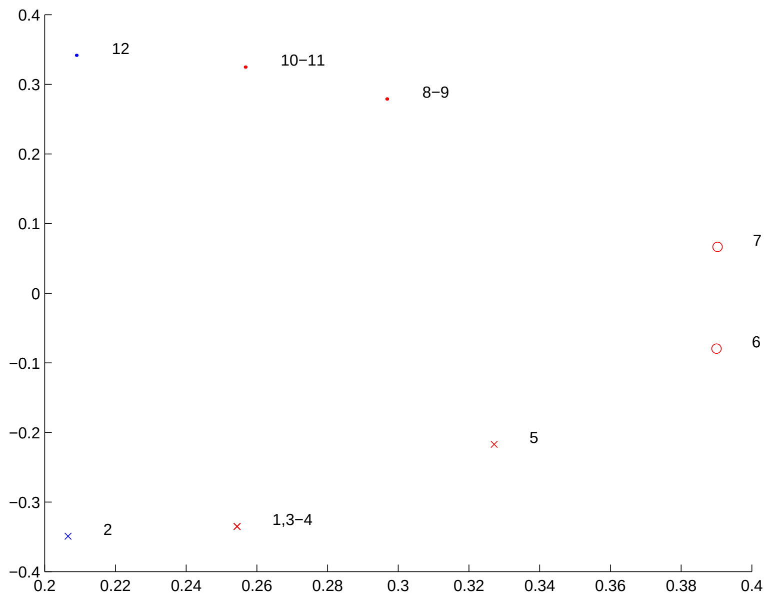

Figure 7: 2-D plot of dataset, positions from SVD, shape and colour from SDD

The tree produced by PDDP is shown in Figure 6. Figure 7 shows the position of points in the first two dimensions produced by SVD (with the points colour coded using the SDD classification which is shown as a decision tree in Figure 8). The decision tree produced by SDD is more discriminating than the tree produced by PDDP. It also agrees better with the apparent distinctions between points captured by the SVD representation.

5 Conclusion

We have shown that SDD works by finding regions of a dataset with anomalously large (positive or negative) values and extracting them. It is therefore best regarded as a bump hunting technique. SDD removes bumps with the largest 'volume', a quantity that is based both on the magnitude of values and the number of positions in the dataset where they occur. This has the general effect of selecting outlier clusters early in the process. We have

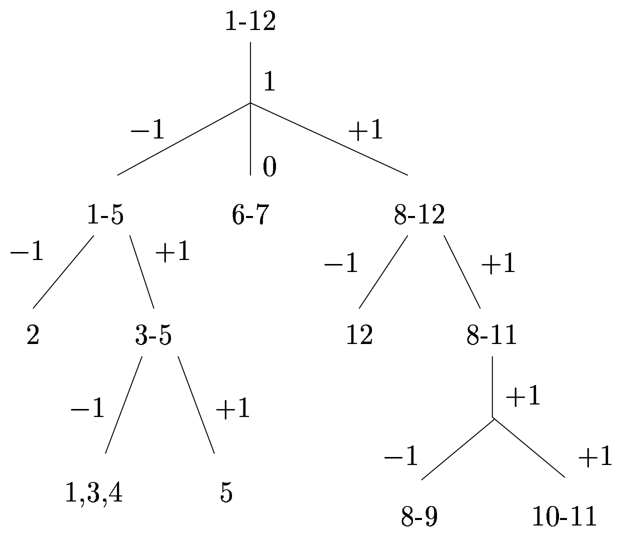

Figure 8: The decision tree produced by SDD, partitioned by rows

demonstrated how this happens in some typical (small) datasets. We have also suggested a modification to the basic algorithm designed to make outliers even more obvious.

The SDD technique works in marked contrast to singular value decomposition, which is a space-transforming technique that attempts to model decreasing amounts of variation. SDD and SVD agree on datasets that contain many small clusters, but do not agree when datasets contain a few large clusters. This explains the observation that SDD can be used quite effectively in latent semantic indexing.

Acknowledgements: We would like to thank D.R. Cohen, T.K. Kyser and H.E. Jamieson for assistance with interpreting results using the geochemical dataset.

References

[1] Michael W. Berry and Ricardo D. Fierro. Low-rank orthogonal decompositions for information retrieval applications. *Numerical linear algebra with applications*, 3(4):301–327, 1996.

[2] M.W. Berry, S.T. Dumais, and G.W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37(4):573–595, 1995.

[3] D.L. Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998.

[4] J. Friedman and N. Fisher. Bump hunting on high-dimensional data. *Statistics and Computation*, 1997.

[5] G.H. Golub and C.F. van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

[6] G. Kolda and D.P. O'Leary. A semi-discrete matrix decomposition for latent semantic indexing in information retrieval. *ACM Transactions on Information Systems*, 16:322–346.

[7] T.G. Kolda and D.P. O'Leary. *Latent Semantic Indexing Via A Semi-Discrete Matrix Decomposition*, volume 107 of *IMA Volumes in Mathematics and Its Applications*, pages 73–80. Springer Verlag.

[8] T.G. Kolda and D.P. O'Leary. Computation and uses of the semidiscrete matrix decomposition. *ACM Transactions of Information Processing*, 1999.

[9] D.P. O'Leary and S. Peleg. Digital image compression by outer product expansion. *IEEE Transactions on Communications*, 31:441–444, 1983.

[10] S. Zyto, A. Grama, and W. Szpankowski. Semi-discrete matrix transforms (sdd) for image and video compression. Technical report, Department of Computer Science, Purdue University, 2000.