Sparse computation for large-scale data mining

Dorit S. Hochbaum
University of California, Berkeley
Etcheverry Hall
Berkeley, CA 94720, USA
Email: hochbaum@ieor.berkeley.edu

Philipp Baumann
University of California, Berkeley
Etcheverry Hall
Berkeley, CA 94720, USA
Email: philipp.baumann@berkeley.edu

***Abstract*—Several leading data mining and clustering algorithms rely on inputs in the form of pairwise similarities. Yet, since the number of potential pairwise similarities grows quadratically in the size of the data set, it is computationally prohibitive to apply such algorithms to large data sets. This paper addresses this challenge with a novel method of sparse computation that computes only the relevant similarities instead of the complete similarity matrix. The method employs an efficient algorithm that provides an "approximate Principal Component Analysis". In the low-dimensional space generated, the concept of grid neighborhoods is applied in order to identify groups of objects with potentially high similarity. Unlike known sparsification approaches that generate first the full set of pairwise similarities and thus take at least quadratic time, the sparse computation method generates only the relevant similarities. Sparse computation can be utilized in any data mining or clustering algorithm that requires pairwise similarities, such as the k-nearest neighbors algorithm or the spectral method. This approach is contrasted with that of grid-based clustering algorithms in that grid neighborhoods proximity is used only to determine the entries in the sparse similarity matrix, not to identify the clusters. Indeed objects can belong to the same grid neighborhood while ending up in different clusters, or conversely, belong to different neighborhoods yet get clustered jointly. The applicability of sparse computation for binary classification is demonstrated here for the recently devised supervised normalized cut (SNC). Our empirical results show that the approach achieves a significant reduction in the density of the similarity matrix, resulting in a substantial reduction in running time, while having a minimal effect (and often none) on accuracy as compared to inputs using a complete similarity matrix.**

I. Introduction

Many applications of data mining and machine learning involve the classification of large-scale data sets. These data sets are often represented by $n \times d$ matrices in which n objects are described by d attributes. Some of the leading machine learning methodologies such as the k-nearest neighbor algorithm [9], the spectral method [1] or supervised normalized cut (SNC) [13], [15], [30], [16] use as input pairwise similarities. A great advantage of these methods is that the pairwise similarities can be defined flexibly to capture the nature of affinities between objects. This flexibility, however, poses a challenge in terms of scalability, as the number of pairwise similarities grows quadratically in the size of the data set.

Existing sparsification techniques have been used in an effort to reduce the number of non-zero entries in the similarity matrix with minimal effect on the matrix properties. These approaches however require to generate, in advance, the full set of pairwise similarities and thus take at least quadratic time.

In this paper, we propose the novel methodology of *sparse computation* to generate a sparse similarity matrix. The method sidesteps the computationally expensive task of constructing the complete similarity matrix and generates only the relevant similarities. This is achieved by projecting the data onto a low-dimensional space in which the concept of grid neighborhoods, borrowed from image representation, is employed to determine efficiently which objects are potentially similar. The grid resolution is used as a parameter to control the density of the resulting similarity matrix. In general, the density of the matrix goes down with finer grid resolutions. With the proposed approach, not only is the output matrix sparse, but also the computation process generating the matrix itself is linear in the number of resulting non-zero entries. The projection onto a low-dimensional space can be attained with PCA (Principal Component Analysis), however for very large matrices applying PCA is computationally prohibitive. Instead, the low-dimensional space is generated with an algorithm referred to here as approximate-PCA . Approximate-PCA provides leading principal components that are very similar to the leading principal components of exact PCA, but requires significantly less running time than exact PCA. The projection of the data onto a low-dimensional space can therefore be accomplished efficiently. Once the non-zero entries are determined, the evaluation of the pairwise similarities for these entries is performed in the original space.

Even though the sparse computation approach uses grid neighborhoods, it is distinctly different from classical grid-based clustering algorithms, [11]. Grid-based clustering, like sparse computation, sub-divides the data space into grid blocks. But it allocates objects in the same block to the same cluster. In contrast, sparse computation is used only to determine the non-zero entries in the sparse similarity matrix, not to identify the clusters. Indeed objects can belong to the same grid neighborhood while ending up in different clusters, or conversely, belong to different neighborhoods yet get clustered jointly.

In addition to the computational speed-ups offered by sparse computation it has several advantages: It can be used for any data mining or clustering algorithm that is based on pairwise similarities; because of the projection of the data onto a low-dimensional space, sparse computation allows the visualization of a data set using the first three principal components (cf. Figure 3); for graph algorithms, such as SNC used here, an additional advantage is that sparse computation tends to break down the data set into a collection of isolated components in the graph. Each of these components is then classified as a separate data set, leading to further improvement in the efficiency of such data mining algorithms.

The new methodology is demonstrated here for Supervised Normalized cut, SNC, (also called *normalized cut prime*). This algorithm was devised in [13] as an efficient variant of

978-1-4799-5666-1/14/$31.00 ©2014 IEEE

normalized cut. It was demonstrated that SNC delivers superior performance for image segmentation problems in comparison to spectral method-based approaches that approximately solve the intractable normalized cut problem, [16]. The input to SNC is a weighted graph in which the nodes correspond to the objects in the data set and the arcs represent the similarities between the respective objects. Thus the size of the graph, and consequently the running time of the algorithm for SNC, are sensitive to the number of non-zero entries (arcs) in the similarity matrix. The running time of SNC is in practice linear in the number of arcs in the generated graph [7]. SNC was recently used as a data mining technique in [30] for enhancing the capabilities of noisy and low-resolution nuclear detectors, and in [15] for assessing the effectiveness of drugs based on HCS images of cells treated by the drugs. In these studies, SNC outperformed other machine learning methods in terms of classification accuracy. Therefore, it is of great interest to develop strategies that efficiently generate a sparse input graph with minimal loss in classification accuracy.

The study presented here applies SNC with sparse computation to seven binary classification data sets from the Machine Learning Repository from University of California at Irvine. For six of the data sets, with up to 46,480 objects, we compared the performance of SNC in the standard set-up with the complete matrix, i.e., the matrix in which the number of non-zero entries coincides with the total number of elements, to the performance of SNC for sparse matrices generated with sparse computation using different grid resolutions. Surprisingly, the accuracies obtained for the sparse matrices barely differ from the accuracies obtained for the complete matrices, even for fine grid resolutions. While experiencing almost no loss in accuracy, the running time of the SNC classifier decreases substantially with increasing grid resolution. For the seventh data set, containing more than half a million objects, it was not possible to compute the complete similarity matrix because of limited memory. For this data set, we tested SNC based on sparse matrices only. Again, it turned out that the accuracy of SNC is affected only to a small extent by the grid resolution. For a similarity matrix with density of 0.009%, SNC still delivers an accuracy of 91.14% for a data set in which 57.4% of the objects are labelled 1. The complete input graph would have contained more than 54 billion arcs.

The remainder of the paper is structured as follows. In Section II, existing sparsification approaches are reviewed. The optimization model of normalized cut prime is introduced in Section III. This section also provides a sketch description of the algorithm used to solve the classification problem based on a parametric cut technique. Section IV includes a detailed description of the sparse computation approach. In Section V, the experimental design is described and the results of the empirical analysis are reported. Section VI contains concluding remarks and directions for future research.

II. Review of existing sparsification approaches

There is a great deal of research work on sparsifying similarity matrices. Such efforts consider input graphs or matrices that are dense, and apply sparsifying algorithms with goals that focus on preserving various matrix properties.

Arora et al. [3] describe a simple random-sampling based procedure for generating a sparse matrix whose eigenvectors are close to the eigenvectors of the original matrix. The algorithm considers all non-zero entries of the original matrix and uses the Chernoff-Hoeffding bounds to set some of the entries to zero. The running time of this algorithm is therefore at least proportional to the number of non-zero entries in the input matrix. Spielman and Teng [26] present a graph sparsification algorithm that produces a subgraph of the original whose Laplacian quadratic form is approximately the same as that of the original graph. Their algorithm has complexity that is close to linearly proportional to the number of edges in the original graph. Jhurani [18] recently proposed an algorithm that transforms the original matrix into a sparse matrix with minimal changes to the singular values and the singular vectors corresponding to the near null-space of the original matrix.

In all these sparsification approaches the input is the complete similarity matrix which is then sparsified by removing a large fraction of non-zero entries. Thus, the sparsification effort alone requires at least to scan the similarity matrix once, resulting in $\Omega(n^2)$ running times for a data set with n objects. For this reason, these algorithms are not applicable to massively large data sets.

McCallum et al. [20] propose a strategy to reduce the computational burden of computing all pairwise similarities. They suggest to use initially an approximate similarity measure to subdivide the objects into overlapping subsets. The exact similarities are then only computed between objects that belong to the same subset. This strategy reduces running time significantly when the computation of the exact similarity measure is expensive, e.g., when the number of attributes is large. In their paper, McCallum et al. [20] study the problem of reference matching in the context of bibliographic citations of research papers. The problem consists in grouping citations that reference the same paper. The approximate distance measure was based on the number of words two citations have in common which can be computed efficiently using an inverted index. However, the complexity of this approach is still $\Omega(n^2)$ since the approximate similarity measure must be computed for all pairs of objects.

Other approaches map the input data to compact discrete or binary codes in order to efficiently identify nearest neighbors. A good mapping ensures that similar objects in the original space are likely to get a similar code. A prominent approach that focuses on preserving a metric of the input data is locality-sensitive hashing (LSH) [17]. Recently, [22] proposed an approach to optimize hash functions that are capable of preserving semantic similarities in terms of Hamming distance metric. These methods could potentially be adapted to generate sparse similarity matrices.

III. The Variant of Normalized Cut and Supervised Normalized Cut

The classification model of normalized cut prime (NC′) was initially used in the context of image segmentation. This problem was mistakenly confused with the NP-hard problem of *normalized cut*, [25], and referred to by the same name in [24]. Although the two problems look similar, the normalized cut problem is intractable, whereas the normalized cut prime problem was shown in [13] to be polynomial time solvable with combinatorial algorithms. This finding was generalized in [14] to other ratio problems. The performance of NC′ on image segmentation problems was found to be of high quality and superior to other techniques, [16]. It was used later with similar success as a machine learning algorithm in a context related to security, [30] and in ranking of drugs according to

their effectiveness, [15]. The normalized cut problem and NC′ are defined as graph problems. Some essential graph notations and definitions are thus introduced next.

Let $G = (V, E)$ be an undirected graph with edge weights w_{ij} associated with each edge $[i, j] \in E$. A bi-partition of a graph is called a *cut*, $(S, \bar{S}) = \{[i, j] | i \in S, j \in \bar{S}\}$, where $S \subset V$ and $\bar{S} = V \setminus S$. The *capacity of a cut* $(S, \bar{S})$ is the sum of weights of edges with one endpoint in S and the other in $\bar{S}$:

$$C(S, \bar{S}) = \sum_{i \in S, j \in \bar{S}, [i,j] \in E} w_{ij}. \tag{1a}$$

More generally, for any pair of subgraphs $A, B \subseteq V$ we define

$$C(A, B) = \sum_{i \in A, j \in B, [i,j] \in E} w_{ij}. \tag{1b}$$

In particular, the *capacity of a subgraph*, $S \subset V$, is the sum of edge weights within the subgraph S:

$$C(S, S) = \sum_{i,j \in S, [i,j] \in E} w_{ij}. \tag{1c}$$

The *weighted degree* of node i is the sum of weights adjacent to i:

$$d_i = \sum_{j | [i,j] \in E} w_{ij}. \tag{1d}$$

In the context of classification the nodes of the graph correspond to objects, each of which is characterized by a feature vector. The edge weights w_{ij} quantify the similarity between the respective feature vectors associated with nodes i and j. Higher similarity is associated with higher weights.

The goal of NC′ is to find a cluster that minimizes the ratio of two criteria. One criterion is to maximize the total similarity of the objects within the cluster (the intra-similarity). The second criterion is to minimize the similarity between the cluster and its complement, or the inter-similarity. The ratio function is a way of combining the two objective functions. The problem is to find a non-empty set S^* strictly contained in V, that minimizes the ratio of the similarity between the set and its complement, inter-similarity, divided by the total similarity within the set S^*, intra-similarity. The NC′ problem formulation is:

$$\mathrm{NC}'(S^*) = \min_{\emptyset \subset S \subset V} \frac{C(S, \bar{S})}{C(S, S)}. \tag{2}$$

The solution set S^* is referred to here as a *source* set, and its complement is called a *sink* set. The optimization problem NC′ was shown in [13] to be solvable in polynomial time using a (parametric) minimum cut procedure on an associated graph. That algorithm can solve more general problems than NC′. For instance, the similarity weights used in the numerator, the inter-similarity, can be different from the similarity weights used in the denominator, the intra-similarity. Also, it was shown in [13] that the problem of NC′ (2) is equivalent to,

$$\min_{\emptyset \subset S \subset V} \frac{C(S, \bar{S})}{\sum_{i \in S} d_i}.$$

In this equivalent formulation, *any* non-negative node weights can be used to replace the weighted degrees of the nodes. Additional discussion on this and other variants of NC′ is provided in [14], [16].

The algorithm of [13] for (2) solves *parametrically* the following linear minimization problem for all parameter values λ,

$$\min_{\emptyset \subset S \subset V} C(S, \bar{S}) - \lambda C(S, S). \tag{3}$$

The optimal solution to the ratio problem NC′ corresponds to one specific optimal value of the parameter, λ^*. It is well known from [12] that for a sequence of values $\lambda_1 < \lambda_2 < \ldots < \lambda_k$, and S_i the respective source set associated with λ_i, the source sets are nested: $S_1 \subseteq S_2 \subseteq \ldots \subseteq S_k$. The value of λ^* is selected as the largest for which the objective function is non-positive: $\min_{\emptyset \subset S \subset V} C(S, \bar{S}) - \lambda^* C(S, S) \leq 0$. A bi-product of this solution method is that it finds *all* "breakpoints" of the values of λ and the associated bi-partitions, one of which is the desired optimal solution. The number of breakpoints is guaranteed to be relatively "small" [12]. Thus the minimum ratio solution corresponds to a specific linear weighting of the two criteria in the objective function. It is often the case, however, that the optimal weighting λ^* of the ratio solution is not necessarily the best in terms of the quality of the resulting cluster, and a source set associated with a non-optimal value of λ is a better cluster. After all, any arbitrary scalar multiplication of the numerator, changes the value of the optimal parameter and potentially the respective bi-partition solution. It is therefore more effective to consider a "good" weighting of the two criteria instead of solving the ratio problem. Here, this weighting value of λ is one of the parameters to be selected when implementing NC′ as a classification method.

Additional details on NC′, its relationship to the normalized cut problem, several extensions of the model, and relationship to the spectral method, are provided in [13], [14].

The NC′ model can be used as an *unsupervised* or a *supervised* method. We refer to the latter as SNC. When the NC′ model is used as an unsupervised method, the input graph contains solely unclassified nodes which refer to objects for which the class label is unknown. To guarantee that the solution is non-empty and strictly contained in V, one has to assign, a priori, at least one node to the source set S and one node to set sink set $\bar{S}$. These nodes are referred to as *seed nodes*. In the supervised case, the input graph contains classified nodes (training data) which refer to objects for which the class label (either A or B) is known and unclassified nodes which refer to objects for which the class label is unknown. We can exploit the seed node mechanism to use the NC′ model in a supervised manner by assigning all classified nodes with label A to the source set S and all classified nodes with label B or to the sink set $\bar{S}$. The goal of the binary classification problem is then to assign all unclassified nodes either to the source set S or the sink set $\bar{S}$.

The input to the classification problem is the graph, $G = (V, E)$, defined on the set of objects V and the similarity weights associated with each pair of nodes $[i, j] \in E$. Two nodes s and t are added to the graph with an arc of infinite weight from s to each classified node with label A and from each classified node with label B to t. On this graph we seek a partition that minimizes the NC′ criterion so that $s \in S$ and $t \in \bar{S}$. Unclassified nodes which end up in S are labeled with A and unclassified nodes which end up in $\bar{S}$ are labeled with B. This process is illustrated in Figure 1. The adjustment of NC′ to a supervised context, as described above, is the *supervised* classification methodology which is used here for the data

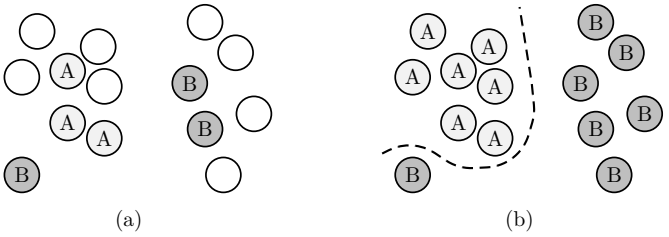

Fig. 1. (a) The input consists of the classified nodes with either label A (light-gray) or label B (gray), and the unclassified nodes with no label (white); (b) The solution consists of two sets that are separated by a cut. All nodes with label A form set S; and all nodes with label B form set $\bar{S}$, where the similarity within S and the dis-similarity between the two sets are high.

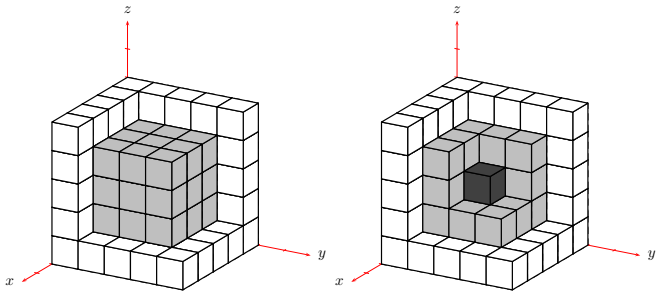

Fig. 2. Grid graph in the space of the $p = 3$ leading principal components. Here the number $k = 5$ and the length of the intervals is the same for all dimensions. The light-gray blocks are neighbors of the dark-gray block.

sets. As mentioned above, the efficiency of NC′ algorithm was established in [13]. In the context of Supervised Normalized Cut, the only additional step prior to performing the NC′ algorithm is to separate and assign training data to either the source or the sink set, which can only reduce complexity.

IV. Proposed Methodology

In this section, we present the sparse computation methodology. In Subsection IV-A, we describe the data structure that is used to partition the low-dimensional space in grid blocks and show how each feature vector can be assigned to a unique block. In Subsection IV-B, we provide the details on how to generate the low-dimensional space with the use of approximate-PCA. In Subsection IV-C, we assess, via visualization, the effectiveness of the approximate-PCA approach in comparison to the exact PCA (applied to the complete data set).

A. The block data structure for sparsification

Given a data set containing n objects in the form of feature vectors with d attributes. The first step is to map the d-dimensional space into a p dimensional space for $p \ll d$ with the use of approximate-PCA, described in the next subsection. Once all feature vectors are mapped into a p-dimensional space, the next step is to subdivide, in each dimension, the range of values of the feature vectors to a pre-specified number of equal intervals, say k. In the following, we refer to k as the grid resolution. It is possible to select a different number of intervals k_i in each dimension $i = 1, \ldots, p$. However, for the sake of simplicity, we use here uniform values of k for all dimensions which allows us to control the total number of grid blocks with a single parameter. This partitions the space of the domain of the feature vectors to k^p grid blocks. Each block is identified with a unique id and each feature vector is assigned to a single block based on the respective intervals in which its p entries fall. Identifying the block to which a vector belongs is executed efficiently, in $O(1)$ complexity per feature vector.

The determination of which pairwise similarities to compute, and the similarity weight calculation (cf. Section V-B1) is performed analogously to that of pixels in images. In 2D images, the standard rule is to consider the 4-neighbor set-up. That is, each pixel is considered adjacent to the pixels on both sides vertically and horizontally. Another neighborhood set-up is that of 8-neighbors that lets also diagonally adjacent pixels to be considered adjacent. Only similarity weights between adjacent pixels are evaluated. Here, we first consider all feature vectors in the same block to be adjacent and thus have their similarities computed in the original d-dimensional space. (If the dimension d is excessively large, we would use the approximate-PCA that samples columns as well as rows and therefore perform the similarity computations with respect to a lower dimensional space. This, however, was not necessary in any of the data sets tested here.) Then, adjacent blocks are identified and the similarities between feature vectors in those blocks are computed. Two blocks are *adjacent* if they are within one interval distance from each other in each dimension (the $\ell_{\max}$ metric). Hence, for each block, there are $3^p - 1$ adjacent blocks. This guarantees that vectors that are further away than one interval distance are not compared. The length of the intervals is considered as the threshold beyond which vectors are considered far from each other.

A finer grid resolution (higher value of k) generally leads to a lower density and thus to shorter running times to produce the similarity matrix for the SNC algorithm, or any other algorithm that uses pairwise similarities, such as k-nearest neighbors, or the spectral method. Figure 2 illustrates a grid for $p = 3$ and $k = 5$. Notice that for $k = 2$ all feature vectors are neighbors of each other, and thus we get the complete similarity matrix.

B. Approximate Principal Component Analysis

Principal Component Analysis (PCA) is a popular technique for projecting data onto a low-dimensional space. The dimensionality of the data set is reduced by selecting the few leading principal components that explain most of the variance in the data set. In real-world data sets with a high degree of correlation among attributes, typically a few leading principal components (e.g. 3) explain a large fraction (e.g. 80%) of the variance in the data sets, although this is obviously a characteristic of the data set. While the use of PCA is appealing, the computation time of PCA on an $n \times d$ matrix A is prohibitive when either n and/or d is extremely large. In fact the PCA computation dominates the computation time for generating all pairwise similarities.

Various approaches have been proposed to improve the effectiveness and efficiency of PCA in specific contexts. Xu et al. [29] propose a variant of PCA suitable for high dimensional data sets that contain arbitrarily corrupted data. Qu et al. [23] and Bai et al. [5] propose methods to compute PCA efficiently when the data is distributed across several locations. Zou et al. [31] address the issue that the principal components are usually a linear combination of all original attributes, which makes their interpretation difficult. They introduce a method called sparse principal component analysis that produces modified principal components with sparse load-

ings, i.e., most weights in the linear combination (known as loadings) are zero. Wickerhauser [27] proposes a method that computes approximate principal components with a slightly better complexity compared to the classical PCA algorithm. Although the approximation error is small, the running time is still prohibitive for large-scale data sets. Altogether the above mentioned approaches are computationally intractable when applied to massively large data sets.

Drineas et al. [10] propose two algorithms for computing approximations to the singular value decomposition (SVD) of a matrix. The running time of the first algorithm, called LinearTimeSVD, is linear in the size of the input matrix and the running time of the second algorithm, called ConstantTimeSVD, is constant, independent of the size of the input matrix. For both algorithms they proved bounds on the approximation error. Both algorithms are extremely fast and easy to implement. The ConstantTimeSVD algorithm is particularly appealing because its speed can be fully controlled. Here, we utilized the ConstantTimeSVD algorithm of [10] to compute approximate principal components.

We first describe how exact PCA works within our approach, and then explain the modifications required for approximate-PCA. PCA can be used to reduce the dimensionality of an $n \times d$ matrix A in which n objects are described by d attributes. The reduction of dimensionality is achieved by replacing the original d attributes by a smaller number of p principal components, which are linear combinations of the original attributes. By using the leading p principal components, the original matrix A is transformed into an $n \times p$ matrix T_p in which the first column has the largest possible variance and each succeeding column has the largest variance subject to being uncorrelated to the preceding column. The transformation reads as follows:

$$T_p = AB_p, \tag{4}$$

where B_p is a $d \times p$ matrix whose columns are the p leading principal components. Principal components are eigenvectors of $A^T A$, but they can also be computed using singular value decomposition (SVD) of the form,

$$A = U\Sigma V^T, \tag{5}$$

where $\Sigma \in \mathbb{R}^{n \times d}$ is a diagonal matrix $\mathbf{diag}(\sigma_1, \ldots, \sigma_\rho)$, for $\rho = \min\{n, d\}$, that contains the singular values $\sigma_i(A)$ of A. The columns of $U \in \mathbb{R}^{n \times n}$ and $V \in \mathbb{R}^{d \times d}$ are the corresponding left and the right singular vectors, respectively. The right singular vectors coincide with the principal components [10]. Hence, matrix T_p can also be derived as follows:

$$T_p = AV_p, \tag{6}$$

where V_p denotes the $d \times p$ matrix consisting of the first p columns of V.

The ConstantTimeSVD algorithm of [10] approximates the singular values and the corresponding singular vectors of an original matrix A by computing SVD on $W^T W$, where $W \in \mathbb{R}^{r \times c}$ is a submatrix of A. The number of columns c and the number of rows r of matrix W can be specified by the user, and the work in [10] links those choices to the closeness between the true, or exact, SVD and the approximate one. The running time of the ConstantTimeSVD algorithm is therefore independent of n and d and can be arbitrarily small.

For specific values of r and c, the closeness of the singular vectors of W to the singular vectors of A can be guaranteed, if W is constructed as follows. First, c columns of matrix A are selected with probabilities $\{p_i\}_{i=1}^{d}$ where $p_i = |A^{(i)}|^2 / \|A\|_F^2$. The selected columns are each rescaled by an appropriate factor to form a matrix $C \in \mathbb{R}^{n \times c}$. Then, r rows of C are selected with probabilities $\{q_j\}_{j=1}^{n}$, where $q_j = |C_{(j)}|^2 / \|C\|_F^2$. The selected rows form matrix $W \in \mathbb{R}^{r \times c}$.

Drineas et al. [10] prove bounds for the size of the difference between the leading ℓ left singular vectors of W (W_ℓ) and the leading w left singular vectors of A (U_w) with respect to the Frobenius norm and the spectral norm, where w is a user-specified integer value and parameter $\ell = \min\{w, \max\{t : \sigma_t^2(W) \geq \gamma \|W\|_F^2\}\}$. Parameter $\sigma_t^2(W)$ denotes the tth squared singular value of matrix W and $\gamma = \epsilon / 100w$ for error parameter $\epsilon > 0$. The approximation error ϵ on the Frobenius norm holds with probability at least $1 - \delta$ when $c = \Omega(w^2\eta^2/\epsilon^4)$, and $r = \Omega(w^2\eta^2/\epsilon^4)$, where $\eta = 1 + \sqrt{8 \log(2/\delta)}$:

$$\|A - W_\ell W_\ell^T A\|_F^2 \leq \|A - U_w U_w^T A\|_F^2 + \epsilon \|A\|_F^2. \tag{7}$$

The approximation error ϵ for the spectral norm holds with probability at least $1 - \delta$ for $\gamma = \epsilon/100$, $c = \Omega(\eta^2/\epsilon^4)$, and $r = \Omega(\eta^2/\epsilon^4)$:

$$\|A - W_\ell W_\ell^T A\|_2^2 \leq \|A - U_k U_k^T A\|_2^2 + \epsilon \|A\|_2^2. \tag{8}$$

In this paper, we compute PCA based on a small subset of rows (feature vectors), selected with the probability q_j. This is called here approximate-PCA. In all data sets considered here, the number of attributes is small, so all columns are preserved. Yet, the technique can be extended to very high-dimensional data sets in which PCA is computed on a small subset of rows (feature vectors) as well as a small subset of columns (attributes).

C. Effectiveness of Approximate-PCA

The usefulness of sparse computation with approximate-PCA to detect similar objects is demonstrated for a data set collected from north east of Andhra Pradesh in India, which can be downloaded from the Machine Learning Repository of the University of California at Irvine [4]. The data set has 583 objects and 10 attributes. The three leading principal components derived by exact PCA and approximate-PCA respectively, are used to visualize the data in 3D in Figure 3. The blue dots represent 416 liver patients and the green dots represent 167 non-liver patients. Both plots visualize the objects in the respective coordinate system defined by the first three principal components. For the exact PCA, the three leading principal columns explain 81% of the total variance. For the approximate-PCA the number of selected rows r was set to 5 which is less than 1% of the original number of rows and seems to be a small number in light of the approximation errors discussed in Section IV-B. However, it turns out that in practice even such a small fraction of the feature vectors is sufficient to discern the main structure of the data (cf. Figure 3). The two clusters that are clearly recognizable in both plots refer to male and female patients. This visualization reveals that in spite of using a tiny sample of the feature vectors, the quality of the approximate-PCA representation of the data is very high. In this case, one would conclude from the visualization that it makes sense to apply a machine learning technique to the

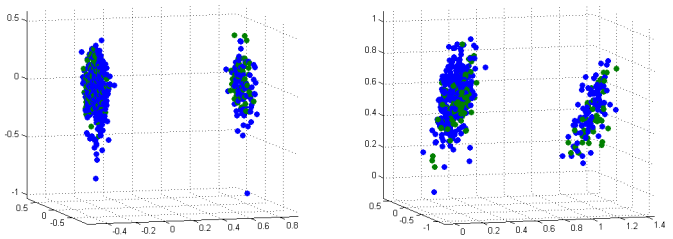

Fig. 3. Left: Visualization of exact PCA; Right: Visualization of approximate-PCA with $r = 5$ rows. Blue represents liver patients and green represents non-liver patients.

set of male patients separately from that of female patients. In our algorithm, this will happen automatically as the two clusters will be recognized as two isolated components in the respective graph.

Figure 3 also manifests the distinction between the use of sparse computation and grid-based clustering. Classical grid-based clustering algorithms, such as described in chapter 12 of [11], would determine, based on the PCA results, that the clusters are those sets of objects that are within close proximity in the p-dimensional space, which in this case are the clusters of male and female patients. This result will miss out on the goal of differentiating the clusters of liver and non-liver patients that are not evident, or separated, in the p dimensional space. In contrast, our algorithm will use sparse computation as a tool to set up the similarity graph in which male and female grouping form separate components. In each of these components SNC will perform the clustering task that will separate liver versus non-liver patients.

V. Empirical Analysis

In this section, we evaluate the effectiveness of the proposed sparse computation method based on seven real-world data sets. In Section V-A, we describe the characteristics of the data sets. In Section V-B, we explain the design of the experimental analysis and in Section V-C, we report the numerical results.

A. Data sets

All the data sets used in this analysis are available on the Machine Learning Repository of the University of California at Irvine [4]. The selected data sets cover areas related to life sciences, engineering, social sciences and business. Our interest was in focusing on large data sets that include thousands of objects. Some of the data sets contain categorial attribute values, which are replaced here by a set of boolean attributes (one boolean attribute per category). In the following, we briefly describe each data set and mention further modifications that we made. The characteristics of the modified data sets are summarized in Table I. The imbalance ratio is defined as number of majority labels divided by number of minority labels.

The data set *Adult* (ADU) stems from the census bureau database and each feature vector represents a person. The goal is to predict whether a person's annual income exceeds 50,000 USD. Persons whose annual income exceeds 50,000 USD are labeled 1. In the original data set there is a categorial and a continuous attribute to capture the educational level of the persons. To avoid double use, we removed the categorial attribute. In addition, we removed all feature vectors that contain missing values.

The data set *Bank Marketing* (BAN) contains information of direct marketing campaigns that were conducted by a Portuguese banking institution. The goal is to predict whether a client opens a long-term deposit after being targeted by such a marketing campaign. Clients who opened the deposit are labeled 1 [21].

The data set *Covertype* (CO2) contains 581,012 feature vectors that describe cartographic characteristics of forest cells in northern Colorado. The goal is to predict the forest cover type. There are seven different cover types which are labeled 1 to 7. By relabeling the types 2 to 7 to 0, we converted the original problem into a binary classification problem. Due to the large number of feature vectors, we have to apply the proposed approach with a sufficiently high grid resolution. In order to analyze low grid resolutions we derived a smaller data set (CO1) from the original data set. The smaller data set was obtained by sampling 8% of the feature vectors of each cover type.

The data set *Cardiotocography* (CAR) consists of feature vectors obtained from fetal cardiotocograms. The goal is to assign a feature vector to one of three fetal states (normal, suspect, pathologic). To obtain a binary classification task, we labeled the states suspect and pathologic as 1 and normal as 0.

The data set *Letter Recognition* comprises 20,000 feature vectors describing numerical attributes of letter images. The goal is to identify a feature vector as one of the 26 capital letters of the English alphabet. Analogously to [6], we converted this data set into two binary classification instances. In data set *LE1*, we labeled the letter "O" as 1 and all other letters as 0. This labeling obviously results in a class imbalance. In data set *LE2*, we labeled the letters "A"–"M" as 1 and all other letters as 0, which results in a well-balanced class distribution.

B. Experimental Design

Each data set is randomly partitioned into two sets, a tuning set (20%) and a testing set (80%). In Subsection V-B1, we describe how we tune the parameters of the SNC model based on the tuning sets and in Subsection V-B2, we describe how we test the SNC model together with the sparse computation approach for different grid resolutions based on the testing sets. The experimental analysis is implemented in Matlab and C. We used the implementation of the pseudoflow algorithm of [12] that was presented in [8] to solve the SNC model. All computations were performed on a standard workstation with two Intel Xeon CPUs (model E5-2687W) with clock speed 3.10 GHz and 128 GB of RAM.

1) Tuning: As noted in Section II, we need to define a similarity measure and specify a value for parameter λ, for which the cluster solution solving,

$$\min_{\emptyset \subset S \subset V} C(S, \bar{S}) - \lambda C(S, S) \qquad \text{cf. (3)}$$

is best. We quantify the similarity between two feature vectors v_i and v_j by the Gaussian similarity function:

$$e^{-\epsilon ||v_i - v_j||_2}$$

where parameter ϵ represents a scaling factor. The Gaussian similarity function is commonly used in image segmentation and spectral clustering [19]. We use it here as a default similarity function as we assume no prior knowledge on the domain of the data. We apply grid search [28] to find a

TABLE I. DATA SETS (AFTER MODIFICATIONS)

Abbr	Domain	Attribute types	# Feature vectors	# Attributes	# 1-Labels	# 0-Labels	Imbalance ratio
CAR	Cardiotocography	Real	2,126	21	471	1,655	3.51
LE1	Letter recognition	Integer	20,000	16	753	19,247	25.56
LE2	Letter recognition	Integer	20,000	16	9,940	10,060	1.01
BAN	Bank marketing	Binary, Real	45,211	51	5,289	39,922	7.55
ADU	Income prediction	Binary, Integer	45,222	88	11,208	34,014	3.03
CO1	Forest cover types	Binary, Integer	46,480	54	16,947	29,533	1.74
CO2	Forest cover types	Binary, Integer	581,012	54	211,840	369,172	1.74

TABLE II. TUNING RESULTS

Abbr	Normalization	ϵ	λ	Tune-Acc [%]
CAR	1	14	1E-3	90.38
LE1	1	30	1E-5	98.30
LE2	0	3	1E-5	92.84
BAN	1	4	1E-5	88.11
ADU	1	3	1E-1	81.49
CO1	1	30	1E-5	79.65

promising combination of values for the parameters ϵ and λ. Thereby, we select ϵ from the set $\epsilon \in \{1, 2, \ldots, 30\}$ and λ from the set $\{10^{-5}, 10^{-4}, \ldots, 10^{-1}\}$. For each combination of tuning parameter values, we try to improve the accuracy by normalizing the tuning set prior to computing the similarities. We normalize all attribute values to lie between zero and one, by computing

$$a_f = \frac{v_f - \min\ v_f}{\max\ v_f - \min\ v_f}$$

where v_f is the actual value of feature f and the maximum and minimum are taken over all feature vectors in the tuning set [28].

For each value of ϵ, we first compute a similarity matrix for all feature vectors in the tuning set. This similarity matrix can be denser than the one used for the testing, and for all data sets presented here, except for CO2, the tuning similarity matrix is complete. If the tuning set is large, such as the case here for CO2, one could use sparse computation to generate a sparse similarity matrix.

For a given similarity matrix we perform ten runs, in each of which the tuning set is randomly partitioned into a set of classified objects (50%) and a set of unclassified objects (50%) (cf. Figure 4). For each such partition, and each similarity matrix (computed as a function of ϵ), and a selected value of λ, we run the SNC algorithm.

Table II lists for each data set the combination of tuning parameter values that yields the highest average accuracy for the ten runs. The average accuracy (Tune-Acc) reported in Table II was achieved based on the complete similarity matrix of the tuning set. The second column of Table II shows that for all data sets except LE2, the average accuracy can be improved by normalizing the input data. For data set CO2, we could not use the complete similarity matrix for tuning because of its excessive size. Therefore, we used the same tuning parameter values as those computed for set CO1. Alternatively, we could have performed tuning on the sparse similarity matrix.

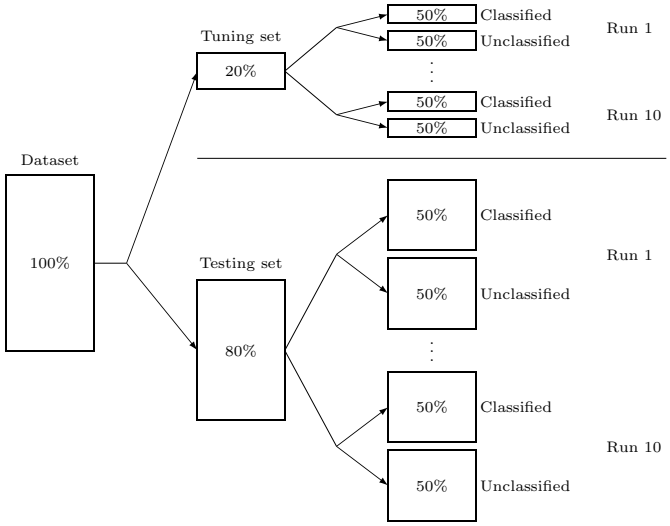

Fig. 4. Visualization of experimental design

2) Testing: We apply the proposed sparse computation approach to each testing set with different values of grid resolutions determined by the number of intervals k. For the testing sets CAR, LE1, LE2, BAN, ADU, and CO1, we tested $k = \{2, \ldots, 20\}$. For the value of $k = 2$ the similarity matrix is complete, and our purpose is to compare the sparse computation approach accuracy and running time to that of using the complete similarity matrix. This comparison is impossible for the largest testing set CO2 as the respective complete matrix cannot be stored in memory. We therefore tested $k = \{100, 200, 300, 400, 500\}$ for the testing set CO2. We choose these large values for k because the marginal change in density generally decreases with increasing grid resolution.

We normalize all testing sets prior to applying approximate-PCA to reduce the effect of outliers [28]. The normalization in general leads to a more balanced distribution of the feature vectors in the projected space and thus to a sparser similarity matrix. We set the number of randomly selected rows for the approximate-PCA to 1000 for CO2 and 30 to all other data sets. To construct the grid, we always used the top three principal components, which turned out to be a good compromise between the amount of variance explained and the resulting number of blocks in the grid. For the computation of the similarities, we normalize the testing sets only if normalization increased the accuracy for the tuning set. The remaining parameters, namely ϵ and λ are also chosen according to the tuning results (cf. Table II).

Analogously to the tuning, we evaluate the accuracy for each grid resolution k by performing ten runs of random subsampling validation.

C. Numerical results

Table III and Figure 5 present the testing results for all data sets except CO2. Table III lists for each of the tested grid resolutions k, the average accuracy (Acc) for the ten runs, the average density (Den) of the ten input graphs, and the average running time required for solving the SNC problem, which is equivalent to the running time required for computing one minimum cut (T_{cut}). The results show that the density of the input graph in general decreases with increasing grid resolution. Theoretically, it is possible that a small increase in the grid resolution results in an increase in density. For example in the one-dimensional case where an interval of length 12 is equally split into 3 subintervals of length 4, then two objects that are located at point 3.5 and point 8.5 do not have their distance evaluated because they belong to non-adjacent intervals. However, if the grid resolution is increased to 4, i.e., the interval of length 12 is equally split into 4 subintervals of length 3, then these two objects fall into adjacent intervals and have their distance evaluated. In our experiments, this happened only once (cf. ADU, increase from $k = 18$ to $k = 19$).

The most surprising result of our study is that across all data sets the accuracy achieved with the sparse similarity matrices barely changes with increasing grid resolution, whereas the running time decreases substantially; for the data set LE1, the accuracy even increases. Also, the accuracies reported in Table III are similar or considerably higher than the accuracies that were obtained based on the complete similarity matrix of the corresponding tuning sets (cf. Table II). The time required to perform the cut is roughly proportional to the number of arcs in the input graph, as the pseudoflow algorithm used in SNC runs in practice in linear time [7]. Thus the decreased density causes a proportional decrease in the running time. Figure 5 visualizes the effect of the grid resolution on the average accuracy and the average density of the input graphs. The plot for testing set CO2 is shown separately in Figure 6.

In Table IV, we report for each grid resolution k the CPU time that is required to compute the grid (T_{grid}). The computation of the grid includes the assignment of all feature vectors of the testing set to the respective boxes in the grid, and the identification of pairs of neighboring non-empty boxes. We also indicate the CPU time required to compute the similarities (T_{sim}) between pairs of feature vectors that lie in the same or in neighboring boxes. Our experiments show that a substantial reduction in CPU time can be achieved by increasing the grid resolution. The time required to construct the grid is relatively small, but increases with increasing grid resolution. The time required to compute the similarities in general decreases with increasing grid resolution. However, beyond a certain number of boxes, the time required to compute the similarities starts to increase (cf. LE1, LE2). This is because the Matlab function that we use to compute the distances within a box (*pdist*) is more efficient than the Matlab function that we use to compute the distances between objects in neighboring boxes (*pdist2*).

Table V, presents all the numerical results for the CO2 testing set. With a density of only 0.009%, it is still possible to achieve an accuracy of 91.14%. Note that the complete input graph for the CO2 testing set contains over 54 billion arcs.

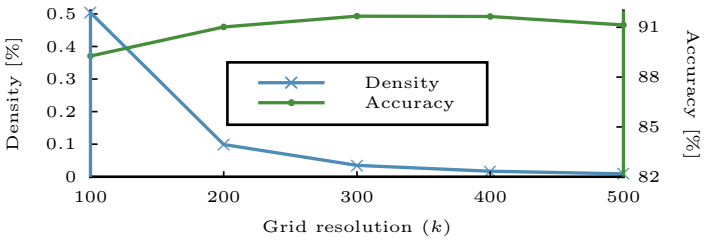

Fig. 6. Effect of grid resolution (k) on average density of validation graph and average accuracy for set CO2

TABLE V. TESTING RESULTS FOR DATA SET CO2

k	Acc [%]	Den [%]	T_{cut} [s]	T_{grid} [s]	T_{sim} [s]
100	89.27	0.504	23.05	129.93	74.61
200	91.02	0.099	3.99	289.47	62.05
300	91.67	0.035	1.29	571.62	79.45
400	91.65	0.017	0.56	949.98	92.99
500	91.14	0.009	0.27	1318.68	92.30

VI. CONCLUSIONS

In this paper, we propose a novel method of sparse computation to efficiently generate a sparse similarity matrix for massively large data sets to be used as input to similarity based classification algorithms. A key feature of the method is that it identifies, without calculating all pairwise similarities, those pairs of objects that are highly similar. This is accomplished by using an algorithm that we refer to as "approximate Principal Component Analysis" that projects the data onto a low-dimensional space. That low-dimensional space is then partitioned into a finite number of grid blocks which allow the identification of groups of similar objects using the concept of grid neighborhoods borrowed from image representation. The resulting sparse similarity matrix is then used in the classification algorithm of Supervised Normalized Cut, the complexity of which is proportional to the number of non-zero entries in the matrix. The effectiveness of the approach is demonstrated for large data sets from the UCI repository. The approach significantly improves running times with minimal loss in accuracy.

The success of the approach proposed here points out several promising directions for future research. These include the utilization of the sparse computation method for other machine learning methods where similarities are a required input. In particular, we plan to use sparse computation in combination with the k-nearest neighbor algorithm and with the spectral method. We also plan to test the approximate-PCA idea for very high dimensional data sets in which the number of attributes is extremely large. Other planned future investigations include the investigation of the effect of using a different number of intervals in each dimension of the grid instead of using the same number for all dimensions and to evaluate the performance of tuning with the sparse similarity matrix instead of the complete similarity matrix. Finally, we plan to compare sparse computation to other dimensionality reduction approaches such as Locality-Sensitive Hashing, [2].

VII. ACKNOWLEDGMENTS

The work of the first author is supported in part by NSF award No. CMMI-1200592. The work of the second author was supported by the Swiss National Science Foundation (Project No. IZK0Z2 151046). The authors acknowledge the help of Prof. Fishbain and Prof. Trautmann who provided access to their workstations. The authors further express their

TABLE III. TESTING RESULTS FOR ALL DATA SETS EXCEPT CO2

		CAR			LE1			LE2			BAN			ADU			CO1		
	k	Acc [%]	Den [%]	T_{cut} [s]	Acc [%]	Den [%]	T_{cut} [s]	Acc [%]	Den [%]	T_{cut} [s]	Acc [%]	Den [%]	T_{cut} [s]	Acc [%]	Den [%]	T_{cut} [s]	Acc [%]	Den [%]	T_{cut} [s]
complete matrix	2	91.37	100.00	<0.01	99.30	100.00	0.35	97.44	100.00	3.84	88.43	1.00	0.59	81.43	100.00	14.10	85.54	100.00	15.44
sparse matrices	3	91.41	79.96	<0.01	99.30	87.76	0.35	97.44	87.76	4.59	88.43	0.74	0.54	81.47	68.47	11.82	85.54	52.76	9.90
	4	91.41	57.81	<0.01	99.30	69.94	0.29	97.44	69.94	3.66	88.41	0.50	0.47	81.61	43.14	6.84	85.54	39.71	6.99
	5	91.49	42.96	<0.01	99.30	51.61	0.23	97.44	51.61	2.69	88.40	0.32	0.32	81.01	31.23	6.39	85.54	31.82	4.98
	6	91.52	29.76	<0.01	99.30	39.27	0.17	97.44	39.27	2.00	88.39	0.22	0.27	81.42	24.46	3.90	85.54	22.74	3.63
	7	91.45	21.27	<0.01	99.31	28.80	0.13	97.44	28.80	1.49	88.37	0.16	0.18	81.05	20.54	4.24	85.54	20.97	3.34
	8	91.57	15.95	<0.01	99.32	22.30	0.10	97.44	22.30	1.14	88.34	0.12	0.18	81.00	17.09	3.15	85.54	18.86	2.98
	9	91.28	12.35	<0.01	99.32	17.08	0.08	97.42	17.08	0.83	88.36	0.09	0.14	80.75	14.58	3.39	85.54	16.96	2.56
	10	91.44	9.62	<0.01	99.33	13.55	0.07	97.44	13.55	0.68	88.34	0.07	0.11	80.78	12.62	2.68	85.54	15.74	2.38
	11	91.25	7.57	<0.01	99.36	10.79	0.05	97.42	10.79	0.52	88.32	0.06	0.09	80.66	11.75	2.55	85.54	14.21	2.13
	12	91.10	6.26	<0.01	99.34	8.75	0.04	97.43	8.75	0.44	88.33	0.05	0.07	80.71	11.05	2.45	85.54	13.20	2.02
	13	91.29	5.19	<0.01	99.36	7.18	0.03	97.42	7.18	0.32	88.33	0.04	0.06	80.63	9.63	2.22	85.54	12.45	1.87
	14	91.31	4.32	<0.01	99.42	5.97	0.03	97.37	5.97	0.26	88.37	0.03	0.05	80.64	9.62	2.17	85.54	11.51	1.75
	15	90.88	3.60	<0.01	99.36	5.00	0.02	97.29	5.00	0.23	88.30	0.03	0.04	80.70	8.25	1.96	85.54	10.86	1.67
	16	90.97	3.07	<0.01	99.42	4.23	0.02	97.22	4.23	0.18	88.33	0.03	0.04	80.69	8.90	2.03	85.53	10.31	1.58
	17	90.94	2.69	<0.01	99.48	3.61	0.02	97.20	3.61	0.15	88.29	0.02	0.03	80.82	7.38	1.62	85.53	9.73	1.48
	18	90.63	2.34	<0.01	99.40	3.09	0.02	97.10	3.09	0.13	88.29	0.02	0.03	80.68	7.03	1.63	85.53	9.17	1.47
	19	90.62	2.04	<0.01	99.41	2.69	0.01	96.99	2.69	0.10	88.24	0.02	0.03	80.65	7.23	1.60	85.51	8.57	1.34
	20	90.84	1.81	<0.01	99.42	2.34	0.01	96.97	2.34	0.09	88.24	0.02	0.02	80.74	5.80	1.41	85.54	8.14	1.29

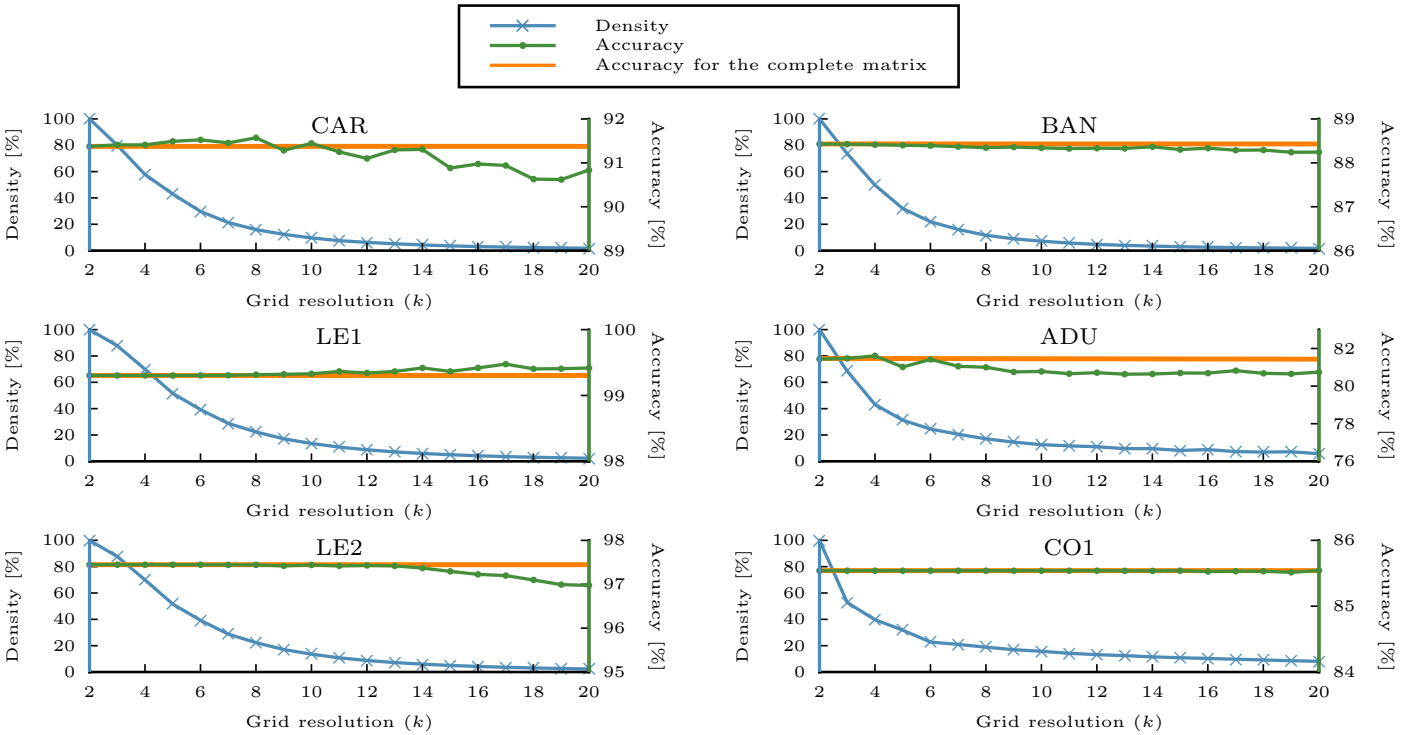

Fig. 5. Effect of grid resolution (k) on average density of validation graph and average accuracy for all sets except CO2

gratitude to the reviewers (in particular Reviewer 3) whose detailed remarks were very helpful in improving the paper.

REFERENCES

[1] C. Alpert and S.-Z. Yao. Spectral partitioning: the more eigenvectors, the better. In *32nd ACM/IEEE Design Automation Conference*, pages 195–200, 1995.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51:117–122, 2008.

[3] S. Arora, E. Hazan, and S. Kale. A fast random sampling algorithm for sparsifying matrices. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 272–279. Springer Berlin, 2006.

[4] A. Asuncion and D.J. Newman. UCI Machine Learning Repository, 2007.

[5] Z.-J. Bai, R.H. Chan, and F.T. Luk. Principal component analysis for distributed data sets with updating. In J. Cao, W. Nejdl, and M. Xu, editors, *Proceedings of International Workshop on Advanced Parallel Processing Technologies*, pages 471–483. Springer, 2005.

[6] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 161–168, Pittsburgh, PA, 2006.

[7] B. G. Chandran and D. S. Hochbaum. A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem. *Operations Research*, 57(2):358–376, 2009.

[8] B. G. Chandran and D. S. Hochbaum. *HPF: Pseudoflow parametric maximum flow solver version 1.0. http://riot.ieor.berkeley.edu/Applications /Pseudoflow/parametric.html*, 2012, Last updated on Aug, 2012.

[9] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Trans. on Information Theory*, 13:21–27, 1967.

TABLE IV. RUNNING TIMES OF SPARSE COMPUTATION IN SECONDS

		CAR		LE1		LE2		BAN		ADU		CO1	
	k	T_{grid}	T_{sim}	T_{grid}	T_{sim}	T_{grid}	T_{sim}	T_{grid}	T_{sim}	T_{grid}	T_{sim}	T_{grid}	T_{sim}
complete matrix	2	0.02	0.15	<0.01	13.19	<0.01	12.80	0.02	67.52	0.02	73.32	0.03	76.53
sparse matrices	3	<0.01	0.14	0.01	10.59	<0.01	10.84	0.02	49.43	0.02	49.58	0.03	39.76
	4	<0.01	0.13	0.02	8.39	0.02	8.37	0.04	32.34	0.07	33.85	0.03	29.23
	5	<0.01	0.13	0.05	6.16	0.05	6.09	0.09	20.19	0.20	22.87	0.05	23.84
	6	<0.01	0.13	0.09	4.62	0.09	4.58	0.19	13.76	0.32	19.76	0.11	17.00
	7	<0.01	0.16	0.12	3.47	0.12	3.38	0.30	10.27	0.39	15.84	0.15	15.73
	8	<0.01	0.19	0.15	2.78	0.15	2.71	0.36	7.70	0.48	13.88	0.24	13.92
	9	<0.01	0.17	0.17	2.24	0.17	2.27	0.46	6.32	0.53	11.80	0.31	12.42
	10	<0.01	0.19	0.18	2.10	0.18	1.99	0.53	5.95	0.54	11.18	0.33	11.67
	11	0.01	0.23	0.20	1.81	0.20	1.80	0.59	4.91	0.62	10.49	0.37	10.67
	12	0.01	0.25	0.21	1.69	0.21	1.69	0.65	4.58	0.68	10.12	0.43	9.76
	13	0.02	0.27	0.22	1.59	0.23	1.58	0.70	4.26	0.76	8.74	0.45	9.26
	14	0.02	0.29	0.24	1.55	0.24	1.56	0.76	4.12	0.76	8.90	0.50	8.52
	15	0.02	0.33	0.26	1.52	0.26	1.53	0.81	3.92	0.85	7.95	0.54	8.04
	16	0.02	0.35	0.28	1.52	0.28	1.53	0.87	3.85	0.85	8.38	0.59	7.72
	17	0.04	0.35	0.30	1.54	0.30	1.56	0.94	3.78	0.91	7.21	0.61	7.38
	18	0.03	0.33	0.32	1.54	0.33	1.54	1.00	3.73	0.92	6.93	0.66	6.88
	19	0.03	0.33	0.35	1.56	0.35	1.58	1.08	3.78	0.97	7.61	0.68	6.49
	20	0.03	0.36	0.39	1.62	0.39	1.67	1.15	3.77	1.01	6.64	0.71	6.27

[10] P. Drineas, R. Kannan, and M.W. Mahoney. Fast monte carlo algorithms for matrices II: computing a low-rank approximation to a matrix. *SIAM J. Computing*, 36:158–183, 2006.

[11] G. Gan, C. Ma, and J. Wu. *Data clustering: theory, algorithms, and applications*. SIAM, 2007.

[12] D.S. Hochbaum. The pseudoflow algorithm: a new algorithm for the maximum-flow problem. *Operations Research*, 56:992–1009, 2008.

[13] D.S. Hochbaum. Polynomial time algorithms for ratio regions and a variant of normalized cut. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 32:889–898, 2010.

[14] D.S. Hochbaum. A polynomial time algorithm for rayleigh ratio on discrete variables: Replacing Spectral Techniques for Expander Ratio, Normalized Cut and Cheeger Constant. *Operations Research*, 61:184–198, 2013.

[15] D.S. Hochbaum, C.-N. Hsu, and Y.T. Yang. Ranking of multidimensional drug profiling data by fractional-adjusted bi-partitional scores. *Bioinformatics*, 28:i106–i114, 2012.

[16] D.S. Hochbaum, C. Lu, and E. Bertelli. Evaluating performance of image segmentation criteria and techniques. *EURO Journal on Computational Optimization*, 1:155–180, 2013.

[17] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 604–613. ACM, 1998.

[18] C. Jhurani. Subspace-preserving sparsification of matrices with minimal perturbation to the near null-space. Part I: basics. 2013. arXiv:1304.7049 [math.NA].

[19] U. V. Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17:395–416, 2007.

[20] A. McCallum, K. Nigam, and L.H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 169–178, Boston, MA, 2000.

[21] S. Moro, R. Laureano, and P. Cortez. Using data mining for bank direct marketing: an application of the CRISP-DM methodology. In *Proceedings of the European Simulation and Modelling Conference - ESM*, pages 117–121. EUROSIS Guimaraes, Portugal, 2011.

[22] M. Norouzi, D.J. Fleet, and R. Salakhutdinov. Hamming distance metric leanring. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1061–1069. Curran Associates, Inc., 2012.

[23] Y. Qu, G. Ostrouchov, N. Samatova, and A. Geist. Principal component analysis for dimension reduction in massive distributed data sets. In S. Parthasarathy, H. Kargupta, V. Kumar, D. Skillicorn, and M. Zaki, editors, *Workshop on High Performance Data Mining at the Second SIAM International Conference on Data Mining*, pages 7–18. Arlington, VA, 2002.

[24] E. Sharon, M. Galun, D. Sharon, R. Basri, and A. Brandt. Hierarchy and adaptivity in segmenting visual scenes. *Nature*, 442:810–813, 2006.

[25] J.B. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22:888–905, 2000.

[26] D.A. Spielman and S.-H. Teng. Spectral sparsification of graphs. *SIAM J. Computing*, 40:981–1025, 2011.

[27] M.V. Wickerhauser. Large-rank approximate principal component analysis with wavelets for signal feature discrimination and the inversion of complicated maps. *Journal of Chemical Information and Computer Sciences*, 34:1036–1046, 1994.

[28] I.H.. Witten and E. Frank. *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd ed. edition, 2005.

[29] H. Xu, C. Caramanis, and S. Mannor. Principal component analysis with contaminated data: the high dimensional case. 2010. arXiv:1002.4658 [stat.ML].

[30] Y.T. Yang, B. Fishbain, D.S. Hochbaum, E.B. Norman, and E. Swanberg. The supervised normalized cut method for detecting, classifying, and identifying special nuclear materials. *INFORMS Journal on Computing*, 2013.

[31] H. Zou, T. Hastie, and R. Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15:265–286, 2006.