Sparse Computation for Large-Scale Data Mining

Dorit S. Hochbaum and Philipp Baumann

Abstract—Leading machine learning techniques rely on inputs in the form of pairwise similarities between objects in the data set. The number of pairwise similarities grows quadratically in the size of the data set which poses a challenge in terms of scalability. One way to achieve practical efficiency for similarity-based techniques is to sparsify the similarity matrix. However, existing sparsification approaches consider the complete similarity matrix and remove some of the non-zero entries. This requires quadratic time and storage and is thus intractable for large-scale data sets. We introduce here a method called *sparse computation* that generates a sparse similarity matrix which contains only relevant similarities without computing first all pairwise similarities. The relevant similarities are identified by projecting the data onto a low-dimensional space in which groups of objects that share the same grid neighborhood are deemed of potential high similarity whereas pairs of objects that do not share a neighborhood are considered to be dissimilar and thus their similarities are not computed. The projection is performed efficiently even for massively large data sets. We apply sparse computation for the K-nearest neighbors algorithm (KNN), for graph-based machine learning techniques of supervised normalized cut and K-supervised normalized cut (SNC and KSNC) and for support vector machines with radial basis function kernels (SVM), on real-world classification problems. Our empirical results show that the approach achieves a significant reduction in the density of the similarity matrix, resulting in a substantial reduction in tuning and testing times, while having a minimal effect (and often none) on accuracy. The low-dimensional projection is of further use in massively large data sets where the grid structure allows to easily identify groups of "almost identical" objects. Such groups of objects are then replaced by representatives, thus reducing the size of the matrix. This approach is effective, as illustrated here for data sets comprising up to 8.5 million objects.

Index Terms—Big data, data mining, similarity-based machine learning, sparsification, supervised normalized cut, K-nearest neighbor algorithm, support vector machines

1 Introduction

Several leading machine learning techniques for classification and clustering such as the K-nearest neighbor algorithm [1], variants of supervised normalized cut [2], [3], [4], [5] or support vector machines with Gaussian RBF kernels [6] use as input pairwise similarities [7]. The application of similarity-based algorithms to large-scale data sets is challenging because the number of similarities grows quadratically as a function of the number of objects in the data set.

Several sparsification approaches known to date, e.g., [8], [9], [10], have been applied to reduce the number of non-zero entries in the similarity matrix with minimal effect on specific matrix properties. These approaches, however, have to generate the full set of pairwise similarities in advance and thus take at least quadratic time.

In this paper, we propose a novel methodology called *sparse computation* that overcomes the computational burden of computing all pairwise similarities between the data points by generating only the relevant similarities. Hence, not only is the resulting matrix sparse but also the computation itself is linear in the number of resulting non-zero entries. The relevant similarities are identified by projecting the data points onto a low-dimensional space in which the concept of grid neighborhoods is employed to find groups of objects with potentially high similarity. Once the relevant pairs of objects have been identified, their similarities are computed in the original space. This differentiates the method from known grid-based clustering algorithms (e.g., [11], [12], [13]) that use the grid neighborhoods to identify the clusters. With our approach, objects can belong to the same grid neighborhood while ending up in different clusters, or conversely, belong to different neighborhoods but still get clustered jointly. The grid dimensionality and grid resolution are the parameters that control the density of the generated similarity matrix.

A key aspect of sparse computation is the efficient projection of the data onto a low-dimensional space. Well-known methods such as Principal Component Analysis (PCA) or Multidimensional Scaling (MDS) require excessive running times for large and high-dimensional data sets and are thus not practical for large-scale applications. We suggest generating a low-dimensional space using 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.

The proposed sparse computation method is broadly applicable for any algorithm that requires the computation of pairwise similarities. Examples of such algorithms include classification algorithms such as the K-nearest neighbor algorithm, variants of supervised normalized cut, support vector machines with non-linear kernels, and spectral methods, as

• D.S. Hochbaum is with the Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720. E-mail: hochbaum@ieor.berkeley.edu.
• P. Baumann is with the Department of Business Administration, University of Bern, Bern CH-3012, Switzerland. E-mail: philipp.baumann@pqm.unbe.ch.

Manuscript received 28 Apr. 2015; revised 29 Apr. 2016; accepted 13 May 2016. Date of publication 6 June 2016; date of current version 29 July 2016. Recommended for acceptance by A. Bifet.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TBDATA.2016.2576470

well as clustering algorithms such as Greedy Agglomerative Clustering algorithms, K-means and other Expectation-Maximization algorithms. For graph-based 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. The machine learning task can then be performed for each of these components separately, which leads to further improvement in the efficiency of such data mining algorithms.

The new methodology is applied here to four different similarity-based machine learning techniques: the K-nearest neighbor algorithm (KNN), support vector machines (SVMs) with radial basis function kernels, and two recently devised graph-based machine learning techniques called supervised normalized cut (SNC) and K-supervised normalized cut (KSNC). A recent comparative study in [7] demonstrates that these similarity-based algorithms are superior to other leading machine learning techniques. Furthermore, in the same study, SVM with a similarity-based kernel (radial basis function) performed better than SVM with non-similarity-based kernels (linear and polynomial). The four techniques are tested and evaluated with respect to accuracy, F_1-scores and running times on ten binary classification data sets from the UCI Machine Learning Repository, the LIBSVM website and the ACM SIGKDD website. For six of the data sets, with up to 46,480 objects, we compared the performance of the techniques on 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 entries, versus the sparse computation set-up. The sparse computation set-up was tested 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 corresponding to very low matrix densities. While experiencing almost no loss in accuracy, the running time of the techniques decreases substantially with increasing grid resolution. For data sets which contain more than half a million objects, it was impossible to compute the complete similarity matrix because of limited memory. For these data sets, we tested the techniques for sparse matrices generated with increasing grid resolutions and, thus, increased sparsity. Again, we found that the accuracy of the tested techniques is affected only to a small extent by the grid resolution. For a similarity matrix with a density of 0.07 percent, it is still possible to achieve an accuracy of at least 96.48 percent with all tested techniques for a testing set in which 36.34 percent of the objects belong to the positive class. The complete input matrix would have contained more than 54 billion non-zero entries.

In addition to the impact of the grid resolution on performance and running times, we analyzed the impact of other control parameters, including the dimensionality of the low-dimensional space. It turns out that the choice of values for parameters other than the grid resolution has little impact on the performance and running time.

We also compared sparse computation to three simple sparsification approaches that we devised (SRT, URS, PRS) and an approach published by [8] (AHK). Of these, only SRT is able to retain an accuracy and F_1-scores similar to those obtained with sparse computation, but at a cost of computing the complete similarity matrix and sorting all entries of the matrix. Thus, this approach is not practical for large-scale data sets.

For massively large data sets, we make further use of the low-dimensional projection and the grid structure for data compression. The grid structure allows to easily identify groups of "almost identical" objects in the projected space and to replace such groups by representatives, thus reducing the size of the matrix. This approach is effective, as illustrated here, for data sets that comprise up to 8.5 million objects. In addition to enabling classification and clustering in massively-large data sets, the method can also be used to represent data sets compactly with minor loss of relevant information [41].

The most important contribution of this paper is to enable the use of similarity-based techniques for large-scale machine learning by overcoming the quadratic growth of similarity computations. Our sparse computation technique achieves for the first time sparsity in a similarity matrix without computing the full matrix first. For various binary classification problems, the sparse computation technique is shown to attain the desired sparsity without diminishing classification accuracy or F_1-scores.

The remainder of the paper is structured as follows. In Section 2, we present four leading machine learning techniques that require pairwise similarities as part of the input. Two techniques are established, and two have been recently developed. Section 3 reviews the existing sparsification approaches. Section 4 provides a detailed description of the sparse computation approach. In Section 5, the experimental design is described, and the results of the empirical analysis are reported. Section 6 contains concluding remarks and directions for future research.

2 Similarity-Based Machine Learning

In this section, we present four similarity-based machine learning techniques that delivered a superior and more robust performance than other techniques such as decision trees, ensemble methods, and regression-based methods in a recent comparative analysis [7]. The four techniques are the K-nearest neighbor algorithm (cf. Section 2.1), support vector machines (cf. Section 2.2), the supervised normalized cut algorithm (cf. Section 2.3) and the K-supervised normalized cut algorithm (cf. Section 2.4). We present these techniques here as binary classification algorithms that assign a set of new objects, which we refer to here as testing objects, to either the positive or negative class based on a set of training objects.

2.1 K-Nearest Neighbor Algorithm

The K-nearest neighbor algorithm [14] uses the training objects themselves to classify new objects. It finds the K most similar training objects to the new object and then assigns to the new object the predominant class among those K neighbors. In order to find the K-nearest neighbors, each new object is compared to each training object. In the experimental analysis, we determined the nearest neighbors based on euclidean distances and considered the nearest neighbor's label for breaking ties. Parameter K was used as a tuning parameter (cf. Table 2 in Section 5.2).

2.2 Support Vector Machines

Support vector machines [15] represent objects as points in space and find the maximum-margin hyperplanes that best separate positive training objects from negative training objects. Testing objects are mapped into that same space, and their class membership is predicted based on which side of the hyperplane they fall on. In addition to performing linear classification, support vector machines can also perform non-linear classification by using kernel functions. Kernel functions are similarity functions that are computed over pairs of objects. These functions implicitly map objects into a high-dimensional space. In the empirical analysis, we test SVM with radial basis function (RBF) kernels which tend to deliver a better performance than linear or polynomial kernel functions [7]. The RBF kernel function value R_{ij} for objects i and j is computed based on the respective vectors of the attribute values x_i and x_j by

$$R_{ij} = e^{-\sigma||x_i - x_j||_2^2},$$

where σ is a scaling parameter. We used the MATLAB interface of the LIBSVM implementation (version 3.20) for support vector classification (see [16]).

2.3 Supervised Normalized Cut

The classification model of the Hochbaum's Normalized Cut (HNC) problem was used initially in the context of image segmentation. This problem was mistakenly confused with the NP-hard problem of *normalized cut* [17] and referred to by the same name in [18]. Although the two problems look similar, the normalized cut problem is intractable, whereas the HNC problem was shown to be solvable in polynomial time with combinatorial algorithms in [2] and [19]. The performance of the HNC classification model on image segmentation problems was found to be of high quality and superior to that of other techniques [5]. It was used later with similar success as a machine learning algorithm in a context related to security [4] and in evaluating the ranking of drugs according to their effectiveness [3].

The HNC problem is defined with graph formalism. We therefore introduce some essential graph notation.

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 $\bar{S} = V \setminus S$. The *capacity of a cut* $(S, \bar{S})$ is the sum of weights of edges with one endpoint in S and the other in $\bar{S}$:

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

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

$$C(A, B) = \sum_{i \in A, j \in B} w_{ij}.$$

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

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

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

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

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

The goal of the HNC problem 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 finding a non-empty set S^* strictly contained in V that minimizes the ratio of the similarity between the set and its complement, the inter-similarity, divided by the total similarity within the set S^*, the intra-similarity. The HNC problem formulation is

$$\text{HNC}(S^*) = \min_{\emptyset \subset S \subset V} \frac{C(S, \bar{S})}{C(S, S)}. \tag{1}$$

The solution set S^* is referred to here as a *source* set, and its complement is called a *sink* set. The optimization problem HNC, (1), was shown in [2] to be solvable in polynomial time using a (parametric) minimum cut procedure on the associated graph. In order to solve the ratio problem (1), one solves instead for a guessed value of a parameter λ the lambda question:

$$\min_{\emptyset \subset S \subset V} C(S, \bar{S}) - \lambda C(S, S) \leq 0? \tag{2}$$

The optimal solution for (1) is attained for λ^* selected as the smallest value for which the objective is non-positive: $\min_{\emptyset \subset S \subset V} C(S, \bar{S}) - \lambda^* C(S, S) \leq 0$. It was shown in [2] that all λ-questions can be solved with a parametric cut procedure that provides as by-product a solution to $\min_{\emptyset \subset S \subset V} C(S, \bar{S}) - \lambda C(S, S)$ for *any* value of λ.

The ratio problem (1) does not necessarily provide the best clustering solution. It may be the case that a non-optimal value of λ provides a better cluster by giving a more appropriate weighting of the two objectives. 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 for the ratio problem and to select the cluster that is the optimal solution for

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

Here, the weight value λ is one of the parameters to be tuned when implementing HNC as a classification method. Additional details on HNC, its relationship to the normalized cut problem, several extensions of the model, and the relationship to the spectral method are provided in [2] and [5].

The HNC model can be used in an unsupervised or a supervised manner. In the unsupervised case, the graph contains only unclassified nodes that 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 set S and one node to set $\bar{S}$. In the supervised case, the input graph contains classified nodes (training data) that refer to objects for which the class label (either positive or negative) is known and unclassified nodes that refer to objects for which the class label is unknown. By assigning all classified nodes with a positive label to set S and all classified nodes with a negative label to set $\bar{S}$ the HNC model can be used in a supervised manner. The goal is then to assign all unclassified nodes either to set S or set $\bar{S}$. Due to the pre-assignment of classified nodes, the graph can be compressed such that it contains only unclassified nodes. This compression can only reduce the running time of the algorithm. We refer to using the HNC model in a supervised manner as supervised normalized cut (SNC).

In this study, we test SNC with exponential similarity weights (referred to as Gaussian weights). The exponential similarity between object i and j is quantified based on the respective vectors of attribute values x_i and x_j by

$$w_{ij} = e^{-\epsilon||x_i - x_j||_2}, \tag{4}$$

where parameter ϵ represents a scaling factor. The exponential similarity function is commonly used in image segmentation and spectral clustering [20]. Two tuning parameters are used for implementing SNC: the relative weighting parameter of the two objectives, λ, and the scaling factor of the exponential weights, ϵ. Table 2 in Section 5.2 lists all tuning parameters of SNC and specifies the range of values that we tested for each parameter. The minimum cut problems were solved with the MATLAB implementation of the HPF pseudoflow algorithm version 3.23 of [21] that was presented in [22].

2.4 K-Supervised Normalized Cut

The algorithm of [2] has been generalized and extended in [19] to a broader set of problems than HNC. 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. It was shown in [2] that the HNC problem (1) is equivalent to

$$\min_{\emptyset \subset S \subset V} \frac{C(S,\bar{S})}{\sum_{i \in S} d_i}. \tag{5}$$

In [19], it was demonstrated that *any* non-negative node weight q_i can be used to replace the weighted degrees of the nodes

$$\min_{\emptyset \subset S \subset V} \frac{C(S,\bar{S})}{\sum_{i \in S} q_i}. \tag{6}$$

In our set-up we solve the linearized problem for an appropriate weight parameter λ:

$$\min_{\emptyset \subset S \subset V} C(S,\bar{S}) - \lambda \sum_{i \in S} q_i. \tag{7}$$

Additional discussion on this and other variants of HNC is provided in [2] and [19]. By alternative selections of node weights, different types of machine learning techniques can be generated. In this study, we test a variant in which the node weights q_i are the average class label of the K nearest labeled objects. For example, if $K = 3$ and the three nearest objects, in terms of similarity, to i have labels 0,1, and 1, then q_i is $2/3$. This version of the problem is referred to as KSNC. The tuning parameters for KSNC are the relative weighting parameter of the two objectives λ, the scaling factor of the exponential weights, ϵ, and the integer parameter K. Table 2 in Section 5.2 includes a specification of the range of values that we tested for these parameters.

3 Existing Sparsification Approaches

The classifcation algorithms presented in the previous section, require as input pairwise similarities between the objects in the data set. The number of pairwise similarities grows quadratically in the size of the data set, which poses a challenge in terms of scalability. This challenge is shared also by a vast spectrum of clustering approaches, including greedy agglomerative clustering and expectation-maximization algorithms.

A great deal of research work has been conducted on sparsifying dense matrices. Such efforts consider input graphs or matrices that are dense and apply sparsifying algorithms that aim to preserve various matrix properties.

Arora et al. [8] describe a simple random-sampling based procedure that generates 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 at least proportional to the number of non-zero entries in the input matrix. Spielman and Teng [9] 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 a complexity that is close to being linear in the number of non-zero entries in the original Laplacian. Jhurani [10] 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.

All these sparsification approaches are based on evaluating all entries of the complete similarity matrix and determining for each entry whether or not to round it to zero. The reading of the entries of the dense similarity matrix alone requires $\Omega(n^2)$ running time for a data set of n objects. For this reason, these algorithms are not practical for large-scale data sets. By contrast, our approach determines in advance which entries of the similarity matrix are relevant and evaluates only those.

Another strategy that aims to reduce the computational burden of computing all pairwise similarities is proposed by McCallum et al. [23]. 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 the 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. [23] study the problem of reference matching in the context of bibliographic citations of research papers. The problem consists of grouping citations that reference the same paper. The approximate distance measure is based on the number of words two citations have in common, which can be computed efficiently using an inverted index. The complexity of this approach is, however, still $\Omega(n^2)$ because 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) [24]. Recently, [25] proposed an approach to optimize hash functions that are capable of preserving semantic similarities in terms of the Hamming distance metric.

4 Sparse Computation

We propose here a technique called sparse computation which achieves, for the first time, sparsity in a similarity matrix without computing the full matrix first. The technique partitions a low-dimensional projection of the data into grid blocks which are then used to identify relevant similarities. This strategy is new except for the projection method for which we use an existing procedure.

The sparse computation method works by first projecting the (high-dimensional) data set onto a low-dimensional space. In the low-dimensional space, we create grid blocks and use grid neighborhoods to select pairs of objects that are deemed to be highly similar. To obtain the projection, we use a method that we call approximate-PCA which is based on the constant time singular value decomposition of Drineas et al. [26]. That method is extremely fast as it selects a random subset of the objects (rows in the matrix) and a random subset of attributes (columns in the matrix). This method was shown in [26] to have bounds on the approximation error.

Sparse computation consists of three major steps: dimensionality reduction, grid construction and selection of matrix entries, and similarity computation. In the dimensionality reduction step, the input data $X \in \mathbb{R}^{n\times d}$ of n objects with d attributes is mapped from a d-dimensional space into a p-dimensional space, where $p \ll d$. In the grid construction and the selection of matrix entries step, the p-dimensional space is divided along each dimension into k equal intervals resulting in k^p grid blocks. We define grid neighborhoods as blocks that are adjacent either horizontally, vertically or diagonally. Since we always select $p = 3$ or 2, the number of adjacent blocks is $3^p - 1$ which is either 26 or 8. We then select pairs of objects that either reside in the same grid block or in neighboring grid blocks. In the similarity computation step, for each pair of objects identified in the previous step, we compute a similarity value which is a non-zero entry in the corresponding similarity matrix. We compute the similarity values with a gaussian similarity function (see (4)). The flowchart shown in Fig. 1 provides an overview of sparse computation.

We present our proposed techniques for accomplishing each of these steps and compare them to possible alternatives. The remainder of this section is organized as follows: In Section 4.1, we describe the technique used for the dimensionality reduction step. We also discuss alternative techniques such as random projections, which have interesting theoretical properties. In Section 4.2, we describe how we use the low-dimensional space to efficiently identify the relevant matrix entries for which the similarity value will be computed in the third step. In Section 4.3, we explain the similarity computation step. Section 4.4 introduces an extension of sparse computation to deal with massively large data sets. Finally, Section 4.5 contrasts sparse computation with multi-resolution-based algorithms.

4.1 Step 1: Dimensionality Reduction

For dimensionality reduction, we seek to employ techniques that achieve a projection from the original number of dimensions d to a small number of dimensions p, which are scalable to large-scale data sets. Some dimensionality reduction methods such as Multi-Dimensional Scaling require to first compute the complete similarity matrix. These methods defeat the purpose of sparse computation which is to compute only the relevant similarities instead of the complete similarity matrix. We propose in this section a highly-scalable sampling variant of Principal Component Analysis referred to here as approximate-PCA. Approximate-PCA is compared to a benchmark technique that is based on random projections. The two techniques are compared in this section qualitatively in terms of their properties. In addition, we conducted an empirical comparison of the two projection techniques and report results in Table 16 in Section 5.3.5. In principle, it is possible to use any dimensionality reduction method including more advanced ones such as auto-encoders or compressive sensing-based methods. Auto-encoders use artificial neural networks to learn complex non-linear encodings (see [27]). In order to obtain such an encoding quickly, one could train the neural network only for few iterations. Compressive sensing-based dimensionality reduction methods such as the one proposed by [28] use a subset of data points as a dictionary. The data points in the dictionary are mapped to a lower-dimensional space using random projections. The lower-dimensional space has around 200 dimensions and is referred to as measurement space. Other data points are then mapped to the measurement space in which they are represented as sparse linear combinations of the data points in the dictionary using regularized regression. Gao et al. [28] demonstrate the effectiveness of compressive sensing-based dimensionality reduction in several experiments. Although these alternatives are computationally more expensive than approximate-PCA, they might be useful in setups in which the data exhibits non-linear characteristics.

4.1.1 Approximate-PCA

The projection onto a p-dimensional space can be obtained using Principal Component Analysis. PCA projects the data points such that the p-dimensional space captures as much of the variance of the data as possible. For real-world data sets, a three-dimensional space obtained by PCA is often sufficient to account for most of the variance (e.g., 80

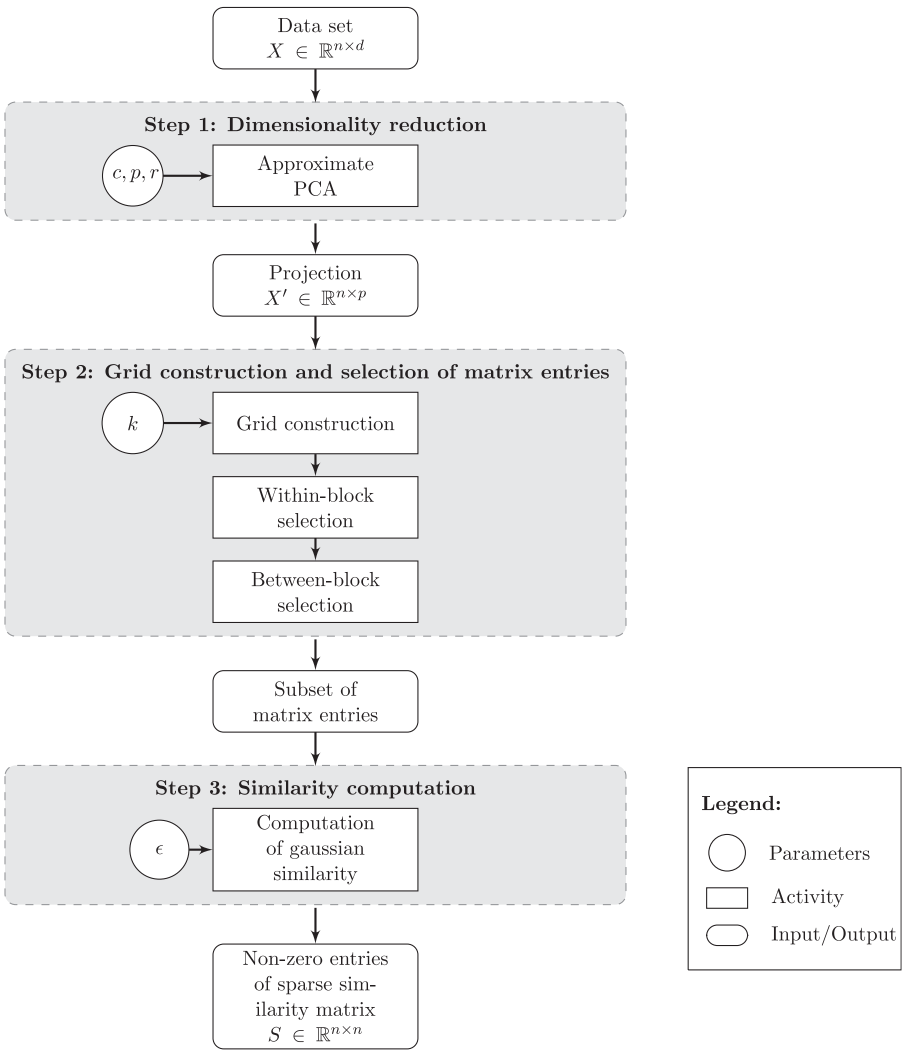

Data set
$X \in \mathbb{R}^{n\times d}$
Step 1: Dimensionality reduction
c, p, r
Approximate PCA
Projection
$X' \in \mathbb{R}^{n\times p}$
Step 2: Grid construction and selection of matrix entries
k
Grid construction
Within-block selection
Between-block selection
Subset of matrix entries
Step 3: Similarity computation
ϵ
Computation of gaussian similarity
Non-zero entries of sparse similarity matrix
$S \in \mathbb{R}^{n\times n}$
Legend:
Parameters
Activity
Input/Output

Fig. 1. Flowchart for sparse computation.

percent). Here, we are particularly interested in projections with $p \leq 3$ because the running time of the second step of sparse computation increases with increasing values of p. The main drawback of PCA is its computational complexity, which precludes its use on large-scale data sets.

We propose an algorithm referred to here as approximate-PCA that provides leading principal components very similar to exact PCA, but requires significantly less running time. The idea is to approximate the leading principal components of the input matrix X by computing exact PCA on a submatrix W of X. The number of columns c and the number of rows r of matrix W can be specified by the user. This approach was proposed by Drineas et al. [26] to approximate the singular values and corresponding singular vectors of a given matrix. For specific values of r and c, the closeness of the singular vectors of W to the singular vectors of X can be guaranteed if W is constructed as follows. First, c columns of matrix X are selected with probabilities $\{p_i\}_{i=1}^{d}$, where $p_i = |X^{(i)}|^2/\|X\|_F^2$. 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. [26] 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 X (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 for 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 $\eta = 1+\sqrt{8\log(2/\delta)}$:

$$\|X - W_\ell W_\ell^T X\|_F^2 \leq \|X - U_w U_w^T X\|_F^2 + \epsilon\|X\|_F^2. \quad (8)$$

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 $r = \Omega(\eta^2/\epsilon^4)$:

$$\|X - W_\ell W_\ell^T X\|_2^2 \leq \|X - U_w U_w^T X\|_2^2 + \epsilon\|X\|_2^2. \quad (9)$$

The computation of the probabilities requires only a linear pass over matrix X, which corresponds to $O(nd)$ time. The running time complexity of exact PCA on submatrix W is $O(rc^2 + c^3)$. Hence, the user can control the running time by selecting specific values for r and c.

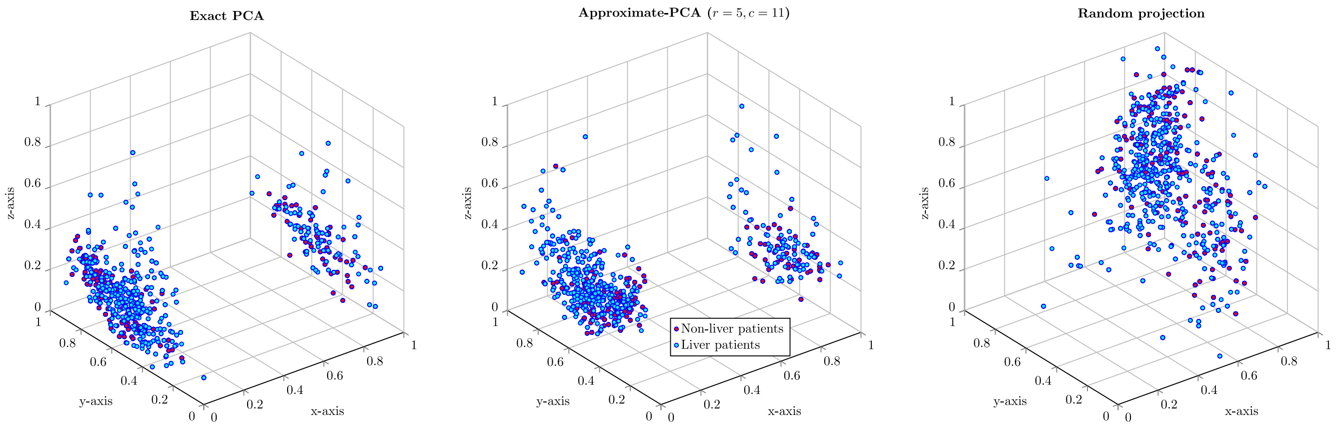


Fig. 2. Low-dimensional data set $X \in \mathbb{R}^{583\times 11}$: Comparison of exact PCA, approximate-PCA and random projections.

Instead of the probabilistic sampling of r rows and c columns, other sampling and feature selection techniques (see [29] for a review) could be used. It is even possible to consider domain-specific knowledge if available. In a supervised context, one could select the rows such that the class labels of the training objects are balanced. However, these other techniques do not possess the theoretical guarantees of the probabilistic selection described above.

4.1.2 Random Projections

We chose random projections as a benchmark method for approximate-PCA because it has been successfully used as a dimensionality reduction technique in Locality-Sensitive Hashing for nearest neighbor searches where the objects are points in the d-dimensional euclidean space (see [30]). Random projections are simple, computationally efficient and possess theoretical guarantees on preserving pairwise distances.

The technique of random projections randomly generates p hyperplanes in the d-dimensional space. The coefficients of each hyperplane are drawn from the standard normal distribution. Let the coefficients of these p hyperplanes form the columns of a random projection matrix $R \in \mathbb{R}^{d\times p}$. The projected data $X' \in \mathbb{R}^{n\times p}$ is then the result of a simple matrix multiplication between the input matrix X and the random projection matrix R with a running time complexity of $O(dpn)$:

$$X' = XR.$$

The technique of random projections has distance preservation guarantees based on the Johnson-Lindenstrauss lemma (JLL) [31]. The JLL guarantees that there is a linear projection of the d-dimensional data onto a p-dimensional space with $p > O(\epsilon^{-2}\log(n))$ that preserves the euclidean distances between any two data points up to a factor of $(1\pm\epsilon)$, with $0 < \epsilon < 1$. Dasgupta and Gupta [32] show that a random projection matrix with i.i.d. Gaussian entries represents such a projection with probability $O(1/n^2)$. They also give tighter bounds on ϵ and p:

$$p \geq 4\left(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3}\right)\ln(n). \quad (10)$$

For cases in which n is large and p is low, the probabilistic distance preservation guarantees of random projections are rather weak. In an empirical analysis with different real-world data sets and various machine learning algorithms, Fradkin and Madigan [33] demonstrate that Principal Component Analysis consistently outperforms random projections in terms of classification accuracy. The outperformance is most distinct for low values of p.

4.1.3 Comparing Properties of Approximate-PCA and Random Projections

We contrast the properties of approximate-PCA and random projections on two real-world data sets, namely, the *Indian Liver Patient Data set* (ILPD) and a variant of the *Bag of Words* data set, referred to here as BOW1. The data can be downloaded from the Machine Learning Repository of the University of California at Irvine [34].

The ILDP data set has 583 objects with 10 attributes. We replaced the categorical attribute by two binary variables, one for each category. Hence, the input matrix X is of size 583×11. We reduced the dimensionality of this data set from $d = 11$ to $p = 3$ using exact PCA, approximate-PCA and random projections. Fig. 2 visualizes the projected data points in 3D for all three dimensionality reduction techniques. For each dimension of the p-dimensional space, the respective coordinates of the objects are scaled to lie between 0 and 1. The blue dots represent 416 liver patients, and the red dots represent 167 non-liver patients. For the exact PCA, the three leading principal columns explain 81 percent of the total variance. The x, y and z axis correspond to the first, second, and third principal component, respectively. For approximate-PCA, we set the number of selected columns c to 11 and the number of selected rows r to 5 which is less than 1 percent of the original number of rows and seems to be a small number in light of the approximation errors discussed in Section 4.1.1. 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. The two clusters that are clearly recognizable in the projections obtained by exact PCA and approximate-PCA refer to male and female patients. These two clusters are not revealed by the random projection method. Fig. 2 indicates that in spite of using a tiny sample of the feature vectors, the quality of the approximate-PCA

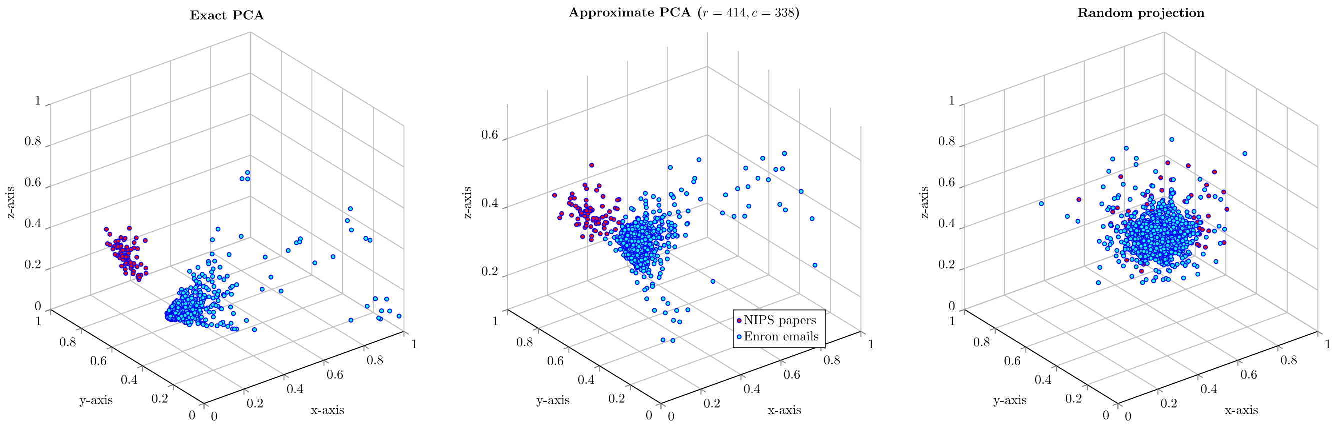


Fig. 3. High-dimensional data set $X \in \mathbb{R}^{41,361 \times 33,781}$: Comparison of exact PCA, approximate-PCA and random projections.

representation of the data is very high. In this case, one can conclude from the visualization that it makes sense to apply a machine learning technique to the set of female (male) patients separately. When using the graph-based machine learning techniques of supervised normalized cut and K-supervised normalized cut (SNC and KSNC), this will happen automatically as the two clusters will be recognized as two isolated components in the associated graphs.

The BOW1 data set has 41,361 objects with 33,781 attributes. The objects are text documents from two different sources, and the attributes are vocabulary words that occurred at least 10 times. The two sources are Enron emails and NIPS papers. A document is represented as a so-called bag of words, i.e., a set of vocabulary words. We used a simplified representation that disregards word order and multiplicity of words. For each document, a binary vector indicates which words occur in the document. Again, we reduced the dimensionality of the data set to $p = 3$ dimensions using exact PCA, approximate-PCA and random projections. Fig. 3 visualizes the projections obtained by the three techniques. For the visualization, we randomly sampled 5 percent of the projected Enron emails and 5 percent of the projected NIPS papers. Exact PCA requires 10,192 seconds on a standard workstation with two Intel Xeon CPUs (model E5-2687W v3) with a clock speed of 3.10 GHz and 256 GB of RAM. For approximate-PCA, we set the number of selected columns c to 338 and the number of selected rows r to 414, which corresponds to 1 percent of the total number of columns/rows. Approximate-PCA with $c = 338$ and $r = 414$ requires only 0.18 seconds on the same machine, which corresponds to a speed up of $55,818$. The projection obtained by approximate-PCA is very similar to the one obtained by exact PCA. In the low-dimensional space obtained by the two PCA-based approaches, the Enron emails are nicely separated from the NIPS papers. The random projection method has the lowest running time with 0.03 seconds, but it captures only little of the structure of the data set and does not separate the two classes in the low-dimensional space.

An empirical study comparing the relative effectiveness of the two techniques is conducted in Section 5.3.5. It turned out that approximate-PCA is more effective in terms of accuracy and F$_1$-scores.

4.2 Step 2: Grid Construction and Selection of Entries in the Similarity Matrix

The goal of this step is to identify the relevant entries in the similarity matrix without computing them first. Each entry corresponds to a pair of objects and our strategy is to identify pairs of objects that are of potential high similarity. The strategy is based on using grid neighborhoods in the p-dimensional space to identify pairs for which the similarity will be computed.

Once all objects are mapped into the p-dimensional space, we subdivide, in each dimension, the corresponding coordinates of the objects into a pre-specified number of k equal intervals. 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 $i = 1, \dots, p$. 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 p-dimensional space into k^p grid blocks. Each object is assigned to a single block based on the respective intervals in which its p coordinates fall. Identifying the block to which an object belongs is executed efficiently, with $O(1)$ complexity per object. As we only identify neighboring blocks of non-empty blocks, the complexity of identifying the neighboring blocks is $O(\min(n, k^p))$ with $p \in \{2, 3\}$.

The selection of pairs is performed in two steps. First, we consider all objects in the same block to be similar and thus select all pairs of objects that belong to the same block, *within-block selection*. Second, we consider objects in adjacent blocks to be similar and select all pairs of objects that belong to adjacent blocks, *between-block selection*. Two blocks are adjacent if they are within a one-interval distance from each other in each dimension (the L_{max} metric). Hence, for each block, there are up to $3^p - 1$ adjacent blocks (see Figs. 4 and 5). This generalizes the neighborhood concept used in image segmentation for two dimensions. For example the four-neighbors set-up has each pixel considered adjacent to the pixels on both sides, vertically and horizontally. Another rule is the eight-neighbors set-up that also considers diagonally neighboring pixels to be adjacent.

The total number of selected entries depends on the grid resolution k. The finer the grid resolution, the smaller the blocks and hence the smaller the set of objects that fall in a block and its neighborhood. Therefore, a finer grid resolution (higher value of k) generally leads to a lower number of

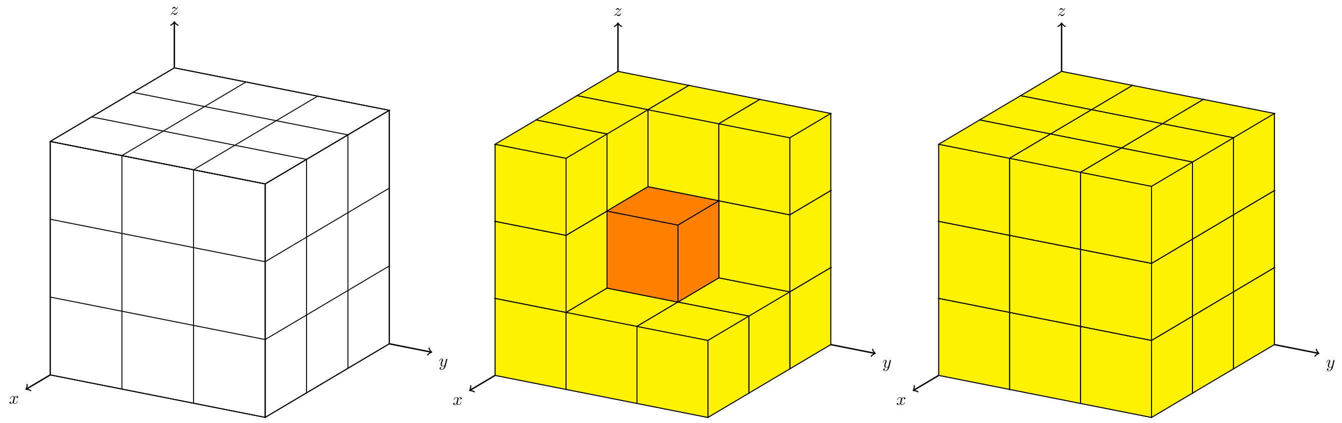


Fig. 4. Grid blocks in a $p = 3$-dimensional space. Here the grid resolution k is set to 3. The yellow blocks are adjacent to the orange block.

selected entries in the similarity matrix and therefore sparser matrices. Notice that for $k = 2$, all objects are neighbors of each other, and thus the case of $k = 2$ corresponds to having the complete matrix.

4.3 Step 3: Similarity Computation

In this step, a similarity value is computed for each matrix entry (pair of objects) selected in step 2. It is possible to use any similarity function to quantify the similarity value between two objects. In our computational analysis, we compute the similarity based on the original d-dimensional input space using the gaussian similarity function introduced in Section 2.3 with ϵ as a scaling parameter (see (4)). For very high-dimensional data sets, such as BOW1, the similarities could also be computed based on the low p-dimensional space.

4.4 Grouping of δ-Identical Objects

For some data sets, it is possible that the density of objects within a block is high and consequently a large number of similarity matrix entries are computed as compared to a balanced density. High density in blocks causes a larger number of pairwise comparisons to be evaluated and increases the density of the similarity matrix. To address this, we introduce the concept of δ-identical objects with respect to a specific grid neighborhood. The idea of δ-identical grouping is to consolidate objects that are within δ-distance into few representatives. Therefore, the presence of blocks with high density turns out to be advantageous as with δ-identical object consolidation it will lead to a substantial reduction in the number of objects.

Objects are considered to be δ-identical if they are of the same type (negative training objects, positive training objects and testing objects), belong to the same block, and are within $\delta\ L_{\max}$ distance. The value of δ is selected as a fraction of the length of a grid block. This fraction is considered a parameter to be determined. Sets of δ-identical objects are represented by a single representative, which is the center of gravity of the set. Note that sets could be singletons. The representatives are assigned a multiplicity weight, which is the number of objects in the set that they represent. The similarities are then computed between representatives only, whereby each similarity value is multiplied by the product of the weights of the two corresponding representatives. Hence, the number of representatives determine the size of the similarity matrix. In our empirical analysis, we consider two extremes for the value of δ. For data sets with up to a million objects, we set δ to zero. For data sets with more than a million objects, we set δ to 1, i.e., in each grid block, we replace all positive training objects by a single representative, all negative objects by a single representative and all testing objects by a single representative. In general, however, δ would be set to some value between these two extremes. Indeed, we are currently investigating how to set the δ parameter as a function of the properties of the data set.

When δ is 1, the total number of representatives is bounded by the number of grid blocks and can thus be controlled by the parameters k (grid resolution) and p (number of dimensions). For each of the k^p grid blocks, no more than three representatives are generated which bounds the total number of representatives by $3k^p$. A conservative bound on the total number of pairwise comparisons between representatives (total number of non-zeros in the similarity matrix of the representatives) is $(3 + \frac{9(3^p-1)}{2})k^p$: For each of the k^p blocks, there are at most three within-block comparisons and nine between-blocks comparisons for each adjacent block. Since a block has at most $3^p - 1$ adjacent blocks, the total number of between-blocks comparisons is bounded by $9(3^p - 1)$. This number of between-blocks comparisons is divided by 2, because each adjacent block is counted twice. The bound is conservative as it assumes that each block contains three representatives, whereas the actual number can be lesser or zero.

4.5 Contrasting Sparse Computation with Multi-Resolution Based Algorithms

Existing multi-resolution and grid-based clustering approaches partition the data space into a number of (hyper-rectangular) grid blocks and achieve a significant reduction of the computational complexity by focusing on the blocks rather than on the data points.

Schikuta [12] proposes the GRIDCLUS algorithm, which partitions the d-dimensional space into grid blocks such that each block contains up to a predefined maximum number of points. Blocks with the highest density (number of objects per spatial volume) become cluster centers, and the remaining blocks are then assigned iteratively to the cluster

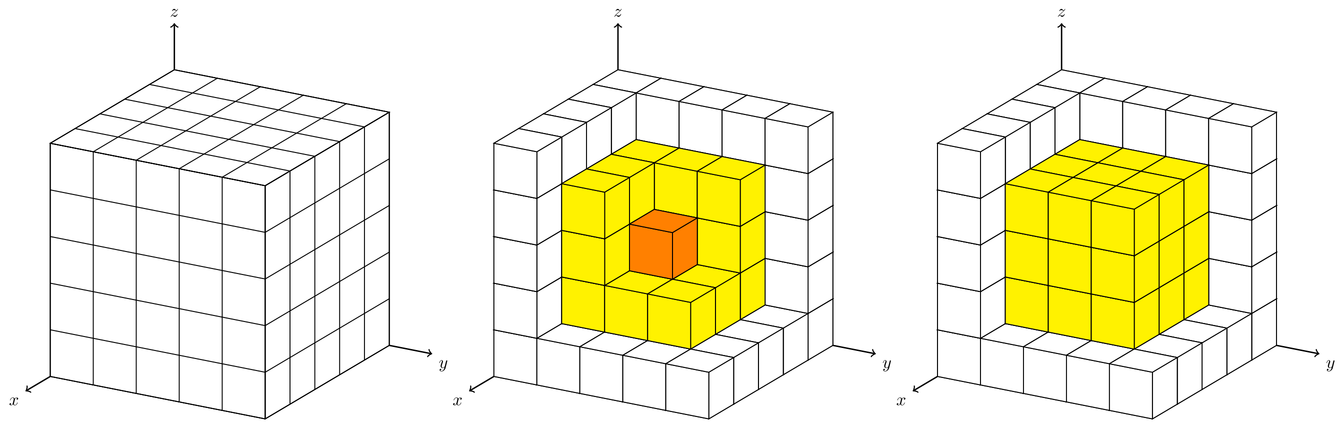


Fig. 5. Grid blocks in a $p = 3$-dimensional space. Here the grid resolution k is set to 5. The yellow blocks are adjacent to the orange block.

centers. During this process, new cluster centers are built and existing clusters are merged. Wang et al. [11] propose STING, which is a multi-resolution clustering algorithm that recursively partitions the d-dimensional data space into grid blocks. The root block, which corresponds to the whole spatial area, is subdivided into children blocks. The subdivision is recursively repeated for the children blocks until the average number of objects per block is within a prescribed range. For each block, summary statistics such as the mean, variance, and minimum and maximum values are computed for each attribute (dimension) based on the objects that lie in the block. Due to the hierarchical structure among the blocks and the summary statistics, spatial data queries can be processed efficiently. Sheikholeslami et al. [13] propose WaveCluster, which partitions the original d-dimensional space into $M = \prod_{i=1}^{d} m_i$ grid blocks by dividing each dimension $i = 1, \ldots, d$ into m_i intervals. Each object is assigned to a single block. Wavelet transforms are then used to convert the feature space of the non-empty blocks into the frequency domain, where the clusters can be distinguished more easily. The GDILC clustering algorithm of Zhao and Song [35] partitions the original d-dimensional space into $M = \prod_{i=1}^{d} m_i$ grid blocks, analogous to WaveCluster. The grid structure is used to compute a density value for each object by counting the number of objects in adjacent blocks that are within a predefined distance threshold. Objects with a density value above a given threshold form a cluster. Based on the grid data structure, the remaining objects are assigned to these clusters and clusters that are close together are merged.

Sparse computation differs from the above described approaches in the following ways:

- Unlike other grid-based clustering algorithms, which construct the grid in the original d-dimensional space, sparse computation constructs the grid in a p-dimensional space with $p \ll d$. This allows to control the number of grid blocks through the two parameters p and k. For algorithms that construct the grid in the d-dimensional space, the number of blocks depends on the dimensionality of the data set and grows exponentially in the number of dimensions. Hence, these algorithms are impractical for high-dimensional data sets. Furthermore, the dimensionality reduction step tends to remove noise in the data set which may lead to higher classification accuracy.
- With the exception of GDILC, the mentioned grid-based clustering algorithms treat data points that fall into the same block as a single data point. Instead, sparse computation performs the clustering task for each data point individually. The grid structure is only used for selecting pairs of objects deemed to be similar. Pairs of objects that fall in the same block can still be assigned to different clusters. Fig. 2 manifests this difference between sparse computation and grid-based clustering. Grid-based clustering algorithms 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, sparse computation will compute similarities among male and female patients in the original space which enables the machine learning algorithm to separate liver versus non-liver patients.
- Sparse computation is robust against distortion induced by the dimensionality reduction step. The error of projecting two objects that are far from each other in the original space to the same block has little impact on the classification result because the similarity is computed in the original space where it is close to zero. The error of determining that two objects that are close to each other in the original space are dissimilar is unlikely due to the transitivity of similarities. In order to make such an error it is necessary that on all possible paths with intermediate objects between this pair of objects, the same type of error, in terms of separating objects on the path and deeming them dissimilar, has to occur. This is highly unlikely as the group of similar objects form a clique, with a number of paths between each pair that is proportional to the size of the clique.

Overall, the novelty of sparse computation lies in decoupling the task of identifying groups of similar objects from the similarity computation task and the classification task. This makes sparse computation a general, efficient and

TABLE 1
Datasets (After Modifications)

Abbr	Source	Domain	Attribute types	# Objects	# Attributes	# Positives	# Negatives	$\frac{\text{\# Positives}}{\text{\# Negatives}}$
CAR	UCI	Cardiotocography	Real	2,126	21	471	1,655	0.285
SPL	LIBSVM	Molecular biology	Integer	3,175	60	1,648	1,527	1.079
LE1	UCI	Letter recognition	Integer	20,000	16	753	19,247	0.039
LE2	UCI	Letter recognition	Integer	20,000	16	9,940	10,060	0.988
BAN	UCI	Bank marketing	Binary, Real	45,211	51	5,289	39,922	0.132
ADU	UCI	Income prediction	Binary, Integer	45,222	88	11,208	34,014	0.330
COV	UCI	Forest cover types	Binary, Integer	581,012	54	211,840	369,172	0.574
KDD	UCI	Network intrusion detection	Binary, Integer	4,898,431	122	3,925,650	972,781	4.035
RLC	SIGKDD	Record linkage comparison	Binary, Real	5,749,132	16	20,931	5,728,201	0.004
BOW2	UCI	Bag of words	Integer	8,499,752	234,151	299,752	8,200,000	0.037

robust methodology that can be combined with supervised and unsupervised machine learning techniques as well as any similarity measure.

5 Empirical Analysis

This section describes how sparse computation affects the classification performance and running times of different similarity-based classifiers. In Section 5.1, we present ten real-world data sets on which the classifiers are tested. In Section 5.2, we describe how the classifiers are tuned and tested on the data sets. Finally, Section 5.3 reports the numerical results.

5.1 Data Sets

We use eight data sets from the UCI Machine Learning Repository [34], one data set from the LIBSVM website [16], and one data set from the ACM SIGKDD website [36]. The selected data sets represent a variety of fields including life sciences, engineering, social sciences, computer sciences and business. The data sets differ in the number of objects, the number of attributes, and the distribution of class labels. The collection comprises binary and multiclass classification problems. For the binary classification problems the class of interest is the positive class, and the complement is the negative class. We converted the multiclass problems into binary classification problems by treating one of the classes as the positive class and all other classes as the negative class. Some data sets have missing values. For those sets, we removed the objects that contain missing values. Categorical attribute values were replaced by a set of boolean attributes (one boolean attribute per category). In the following, we provide a short description of each data set. A summary of the characteristics can be found in Table 1. The last column of the table lists the ratio of the number of objects in the positive class to the number of objects in the negative class.

The data set *Cardiotocography* (CAR) consists of fetal cardiotocograms. The states "suspect" and "pathologic" were consolidated into the positive class, whereas the normal states were treated as negatives.

The data set *Splice* (SPL) contains a set of DNA sequences. The positive class are sequences that contain an exon/intron or an intron/exon splice junction. DNA sequences that contain neither junction belong to the negative class. We downloaded the LIBSVM version of this data set in which the conversion of strings to numbers and the labeling of the objects have already been performed.

The data set *Letter Recognition* comprises numerical attributes of letter images. Similar to [37], we converted this data set into two binary classification instances. In data set *LE1*, only letter "O" is treated as positive. This labeling obviously results in a high class imbalance. In data set *LE2*, letters {"A", "B",..., "M"} were treated as positives, which results in a well-balanced class distribution.

The data set *Adult* (ADU) contains census data on adults. The task is to predict whether an adult's annual income exceeds $50,000$ USD. Adults whose annual income exceeds $50,000$ USD form the positive class. In the original data set, a categorical and a continuous attribute capture the educational level of the adult. To avoid double use, we removed the categorical attribute.

The data set *Bank Marketing* (BAN) contains information on direct marketing calls that were conducted by a Portuguese banking institution. Clients who opened up a long-term deposit after being called are treated as positives [38].

The data set *Covertype* (COV) contains cartographic characteristics of forest cells in northern Colorado. There are seven different cover types, which are labeled 1 to 7. We treated type 1 as the positive class and types 2 to 7 as the negative class.

The data set *KDDCup99* (KDD) is the full data set from the KDD Cup 1999 which contains close to 5 million records of connections to a computer network. Each connection is labeled as either normal, or as an attack. We treated attacks as the positive class.

The data set *Record Linkage Comparison Patterns* (RLC) comprises 5.7 million comparison patterns of pairs of patient records from a German cancer registry. The patterns describe the similarity of the two corresponding patient records. The attributes refer to personal information such as for example first name, family name, sex, and date of birth. The attribute values quantify the agreement of the underlying information. The agreement of name components is quantified by a real number in the interval [0,1], where 1 indicates complete agreement and 0 indicates maximal disagreement. For all other attributes only the value 1 (equal) and 0 (not equal) are used. For each attribute that has missing values, we substitute the missing values with value 0 and introduce an additional binary attribute to indicate missing values. The goal is to classify the comparison patterns as a match (corresponding records refer to same patient) or a mismatch (corresponding records refer to different patients). The matches form the positive class.

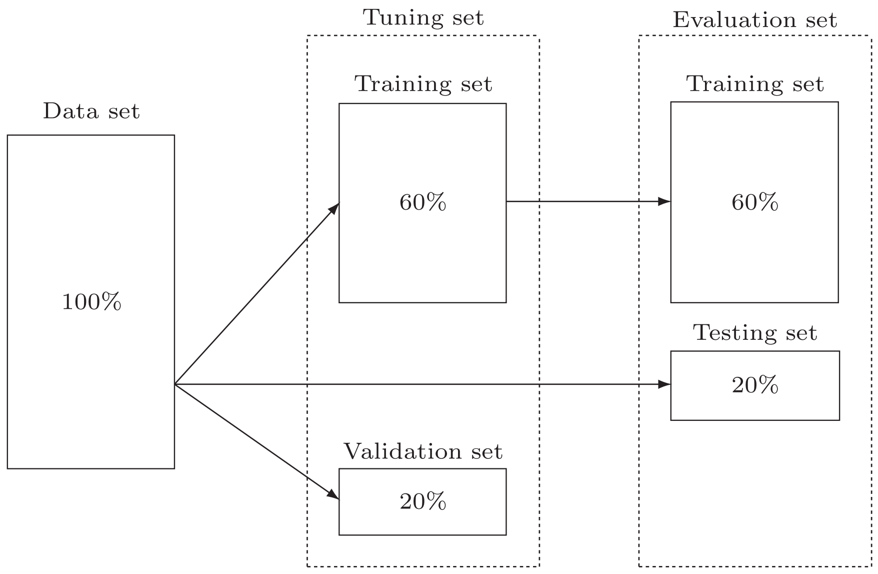


Fig. 6. Partitioning of data sets.

The data set *Bag of Words* (BOW2) comprises over 8.5 million text documents from two different sources (New York Times articles and PubMed abstracts). The attributes are vocabulary words that occurred at least 10 times. A document is represented as a so-called bag of words, i.e., a set of vocabulary words. We used here a simplified representation which disregards word order and multiplicity of words. For each document, a binary vector indicates which words occur in the document. We treated New York Times articles as positives.

5.2 Experimental Design

Sparse computation is evaluated in combination with the four similarity-based classifiers KNN, KSNC, SNC, and SVM introduced in Sections 2.1-2.4. We first introduce the performance metrics in Section 5.2.1 and then describe the tuning and testing of the classifiers in Sections 5.2.2 and 5.2.3, respectively. The experimental analysis is implemented in MATLAB R2015a, and the computations were performed on a workstation with two Intel Xeon CPUs (model E5-2687W v3) with a clock speed of 3.10 GHz and 256 GB of RAM.

5.2.1 Performance Metrics

The classification performance is measured here in terms of accuracy (ACC) and F_1-scores (F1). These performance measures are widely used in the machine learning literature. Let TP, TN, FP, and FN denote the true positives, true negatives, false positives, and false negatives, respectively. The accuracy is defined as the number of correctly classified objects over the total number of classified objects: ACC = (TP+TN)/(TP+FP+TN+FN). The F_1-score is the harmonic mean of precision and recall: F1 = 2TP/(2TP + FP + FN).

5.2.2 Tuning

The tuning is performed for a given classifier, a given data set, and given parameters for sparse computation (number of randomly selected columns c, grid resolution k, grid dimensionality p, and number of randomly selected rows r). As illustrated in Fig. 6, the data set is randomly partitioned into a training set (60 percent), a validation set (20 percent) and a testing set (20 percent). The union of the training and the validation set constitutes the tuning set. The union of the training and the testing set constitutes the evaluation set. The goal of tuning is to determine a preprocessing option and a combination of tuning parameter values based on the tuning set. Preprocessing modifies the input data before the classifier is applied, which sometimes improves performance. We evaluate each combination of tuning parameter values with and without first normalizing the tuning set. Normalization scales attribute values to lie between zero and one and therefore prevents attributes with large values from dominating distance or similarity computations, even though other attributes are more important for distinguishing the objects [39]. For a data set of n objects and one selected attribute, let $\mathbf{x} \in \mathbb{R}^n$ be the vector of the values of the attribute and let $\mathbf{1} \in \mathbb{R}^n$ be a vector of all ones. Then, the vector of normalized values $\mathbf{x}'$ is computed as follows:

$$\mathbf{x}' = \frac{\mathbf{x} - \min(\mathbf{x})\mathbf{1}}{\max(\mathbf{x}) - \min(\mathbf{x})}.$$

The tuning parameters and the tested ranges of values are given in Table 2 for all classifiers. KNN, for instance, is applied 2 × 25 times (once with and once without normalization) to each tuning set. Thereby, the objects in the training set serve as training objects with known class labels,

TABLE 2
Tuning Parameters

Classifier	Tuning parameter name	Values
KNN	Parameter K	$1, 2, \ldots, 25$
KSNC	Scaling parameter ϵ	$1, \ldots, 15$
	Weighting parameter λ	$0, 10^{-3}, 10^{-2}, 10^{-1}$,1,5,10,20,30
	Parameter K	$1, \ldots, 3$
SNC	Scaling parameter ϵ	$1, \ldots, 15$
	Weighting parameter λ	$0, 10^{-3}, 10^{-2}, 10^{-1}$,1,5,10,20,30
SVM	Kernel	Radial Basis Function
	Derivative parameter σ	$2^{-5}, 2^{-4}, \ldots, 2^{5}, 10^{-10}, 10^{-9}, \ldots, 10^{10}$

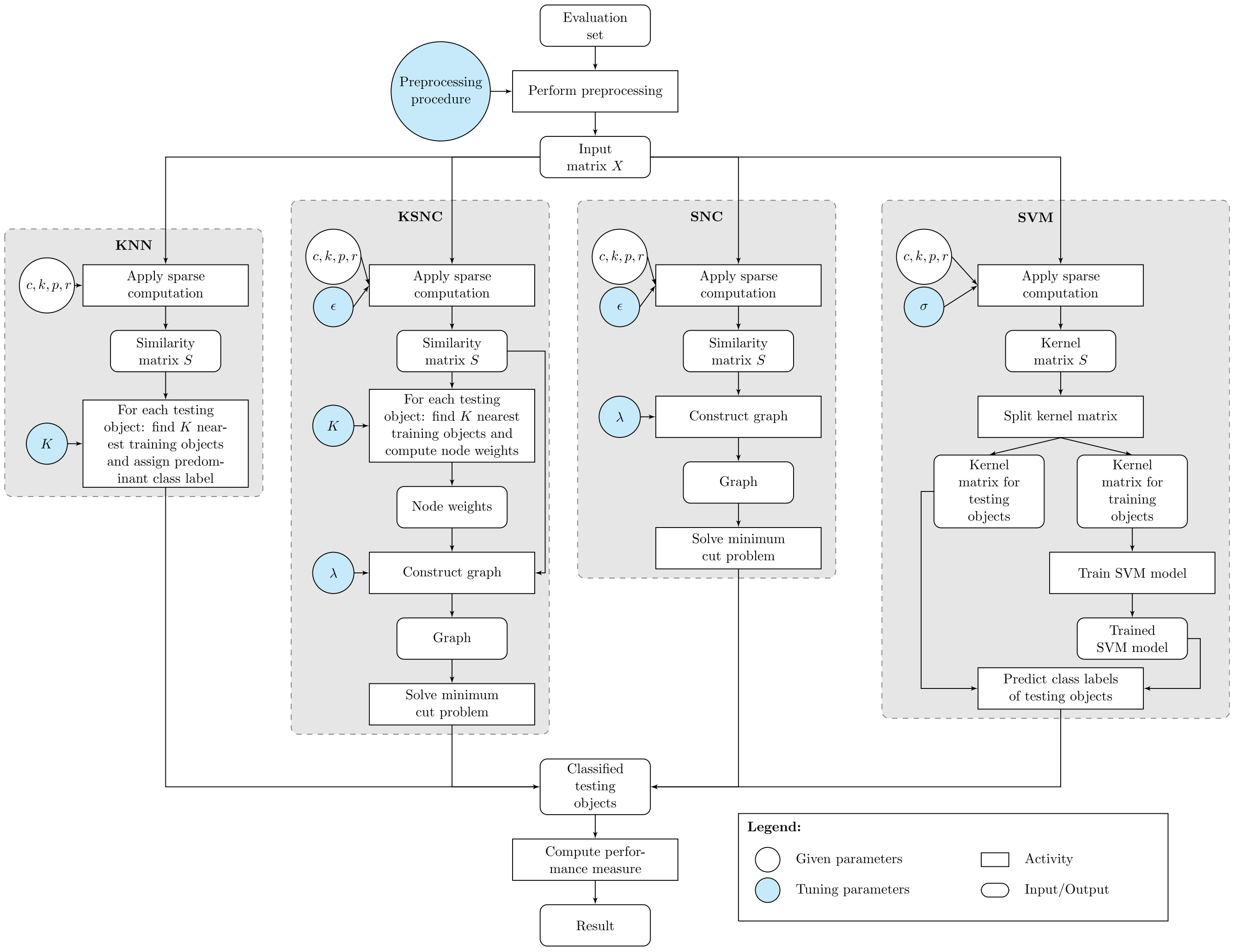

Evaluation set
Preprocessing procedure
Perform preprocessing
Input matrix X
KNN
KSNC
SNC
SVM
c, k, p, r
Apply sparse computation
Similarity matrix S
For each testing object: find K nearest training objects and assign predominant class label
K
ϵ
For each testing object: find K nearest training objects and compute node weights
Node weights
λ
Construct graph
Graph
Solve minimum cut problem
σ
Kernel matrix S
Split kernel matrix
Kernel matrix for testing objects
Kernel matrix for training objects
Train SVM model
Trained SVM model
Predict class labels of testing objects
Classified testing objects
Compute performance measure
Result
Legend:
Given parameters
Tuning parameters
Activity
Input/Output

Fig. 7. Flowchart for the testing methodology.

and the objects in the validation set are to be classified. It is possible to speed up the tuning by storing intermediate results. The extent to which intermediate results can be stored depends on the classifier.

- For all classifiers, only a single grid is computed. This grid is reused for all combinations of tuning parameter values. Note that the grid construction is not affected by the preprocessing option because we always normalize the tuning set prior to applying approximate-PCA to reduce the effect of the outliers. The distance computation is affected by the preprocessing option but not by tuning parameter values. It is therefore sufficient to compute the distances twice, once for the original tuning set and once for the normalized tuning set.
- For KNN, the number of nearest neighbors that are stored for each object in the validation set corresponds to the largest value of parameter K. The sorting is done only twice, once for the distances computed based on the normalized tuning set and once for the distances computed based on the original tuning set. For each value of K only the majority label has to be computed for each object in the validation set.
- From the three tuning parameters of KSNC, only ϵ affects the similarity computations. We therefore reuse the distance matrix for all combinations of tuning parameter values. For each value of ϵ, the distance matrix is converted into a similarity matrix. These similarity matrices are reused for all values of λ. Analogous to KNN, the nearest neighbors are determined only twice.
- For SVM, we compute the distances once and convert them into similarities for each value of σ.

5.2.3 Testing

The testing is performed for a given classifier, a given data set, the given parameters for sparse computation (number of randomly selected columns c, grid resolution k, grid dimensionality p, and number of randomly selected rows r) and a given performance measure. We select the combination of tuning parameter values and preprocessing option that achieved the best performance with respect to the given performance measure for the validation set. In the case of ties, we select the lowest index combination. The flowchart shown in Fig. 7 illustrates how testing is performed for the different classifiers. For all classifiers, sparse computation is applied to the entire evaluation set, which includes the training objects and the testing objects.

For SNC and KSNC, the resulting (sparse) similarity matrix is used to represent the evaluation set as a graph. The classification is performed by solving a single minimum cut problem on this graph. Hence, all testing objects are classified simultaneously.

For SVM, sparse computation is used to compute the kernel matrix for the entire evaluation set. From this kernel matrix, a kernel matrix for the testing and a kernel matrix for the training objects are extracted. An SVM model is trained on the kernel matrix for training objects. The kernel matrix for testing objects is then fed to the trained SVM model to predict the class membership for each testing object.

For KNN, the similarity matrix generated by sparse computation is used to find for each testing object the K nearest training objects. The performance measure is then computed analogously for all classifiers based on the classified testing objects.

5.3 Numerical Results

Sparse computation can be controlled by several parameters that include the grid resolution k, the grid dimensionality p, and the number of rows (columns) selected for approximate-PCA r (c). We first report in Section 5.3.1 the performance of the classifiers for different grid resolutions. In Section 5.3.2, we study the effect of varying the grid dimensionality p for different grid resolutions. In Section 5.3.3, we study the impact of varying the number of rows r for different grid resolutions. In Section 5.3.4, we compare sparse computation to four alternative sparsification approaches. In Section 5.3.5, we apply sparse computation with δ-identical grouping to the largest data sets, KDD, RLC, and BOW2. Finally, in Section 5.3.6, we provide general recommendations for parameter selection.

5.3.1 Impact of Grid Resolution

We first focus on the sets CAR, SPL, LE1, LE2, BAN, and ADU. For these sets, the complete similarity matrices for the tuning and the evaluation sets are small enough to fit in the memory of our machine (256 GB). It is therefore possible to compare the sparse computation approach in terms of the accuracy, F_1-score and running time to that of using the complete similarity matrix. We tuned and tested the four classifiers with grid resolutions $k = \{2, 4, \ldots, 20\}$. For the value of $k = 2$ the similarity matrix is complete. The number of randomly selected rows r for the approximate-PCA was set to one percent of the total number of rows in the tuning set. If one percent of the number of rows was less than 150, we set r to 150. We did not sample columns for approximate-PCA because all these data sets have a rather small number of columns. To construct the grid, we 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.

Fig. 8 visualizes the impact of the grid resolution on the accuracy of the classifiers, the density of the similarity matrix and the tuning times of the classifiers. Each row of plots corresponds to a data set. The tuning time measures the time required to determine the best preprocessing option and the best combination of tuning parameter values.

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. In some cases, the accuracy even increases with increasing grid resolution. This is possible as increasing the grid resolution tends to remove more noise from the data set. KNN achieves the highest accuracy for all data sets except LE2 with a grid resolution greater than or equal to 10. Similarly, SNC achieves the highest accuracy for all sets except BAN and ADU with a grid resolution greater than or equal to eight. KSNC achieves the highest accuracy for all data sets with a sparse similarity matrix. The accuracy achieved by SVM is affected more by the grid resolution for sets CAR and LE1 compared to the other three classifiers. Additionally, SVM achieves the highest accuracies with the complete similarity matrix.

The density of the similarity matrices decreases with increasing grid resolution. Thereby the marginal change in density decreases for all data sets with increasing grid resolution. The tuning time of KNN, SNC and KSNC is roughly proportional to the density of the similarity matrix. Thus, the decreased density causes a proportional decrease in the tuning time. This decrease is more pronounced for the large sets LE1, LE2, BAN, and ADU, for which most of the tuning time is used for similarity computations. For data set BAN, the tuning time of KSNC and SNC could be reduced by 98.22 and 98.26 percent, respectively, by increasing the grid resolution from 2 to 20 with almost no change in accuracy. For the smaller sets CAR and SPL, the computational overhead becomes the dominant part of the tuning time for higher grid resolutions. The reduction in tuning time for SVM is less distinct. A reason for this is that the LIBSVM implementation internally converts the precomputed sparse kernel matrix into a full kernel matrix, which takes a considerable amount of time for larger sets. The tuning time of KNN for low grid resolutions is considerably smaller because the similarity computations can be reused for all values of k and the number of different tuning parameter combinations is much larger for SNC and KSNC. However, the difference in tuning time becomes smaller with increasing grid resolution.

Tables 3, 4, 5, 6, 7, 8 present the results shown in Fig. 8 in tabular form. In addition to the density, accuracy and tuning time results, the tables also list the testing times and F_1-scores. The testing time includes the time required to compute the sparse similarity matrix for the evaluation set and the time required to classify the objects in the testing set. It is reported here for applying the classifier with the preprocessing option and the tuning parameter values that achieved the best accuracy for the validation set. In order to report the F_1-scores, we applied the classifier a second time to the evaluation set, but this time with the preprocessing option and the tuning parameter values that achieved the best F_1-scores for the tuning set. For KNN, KSNC and SNC, the grid resolution again has little impact on the F_1-scores, but it reduces the testing times considerably. It appears that SVM is less robust with respect to F_1-scores compared to the other classifiers.

For the data set COV, the complete similarity matrix requires more than $1,700$ GB of storage, assuming that each entry is represented in double precision with 64 bits. The memory limit of our machine prevents a comparison of the sparse computation approach to that of using the complete similarity matrix. We therefore test sparse

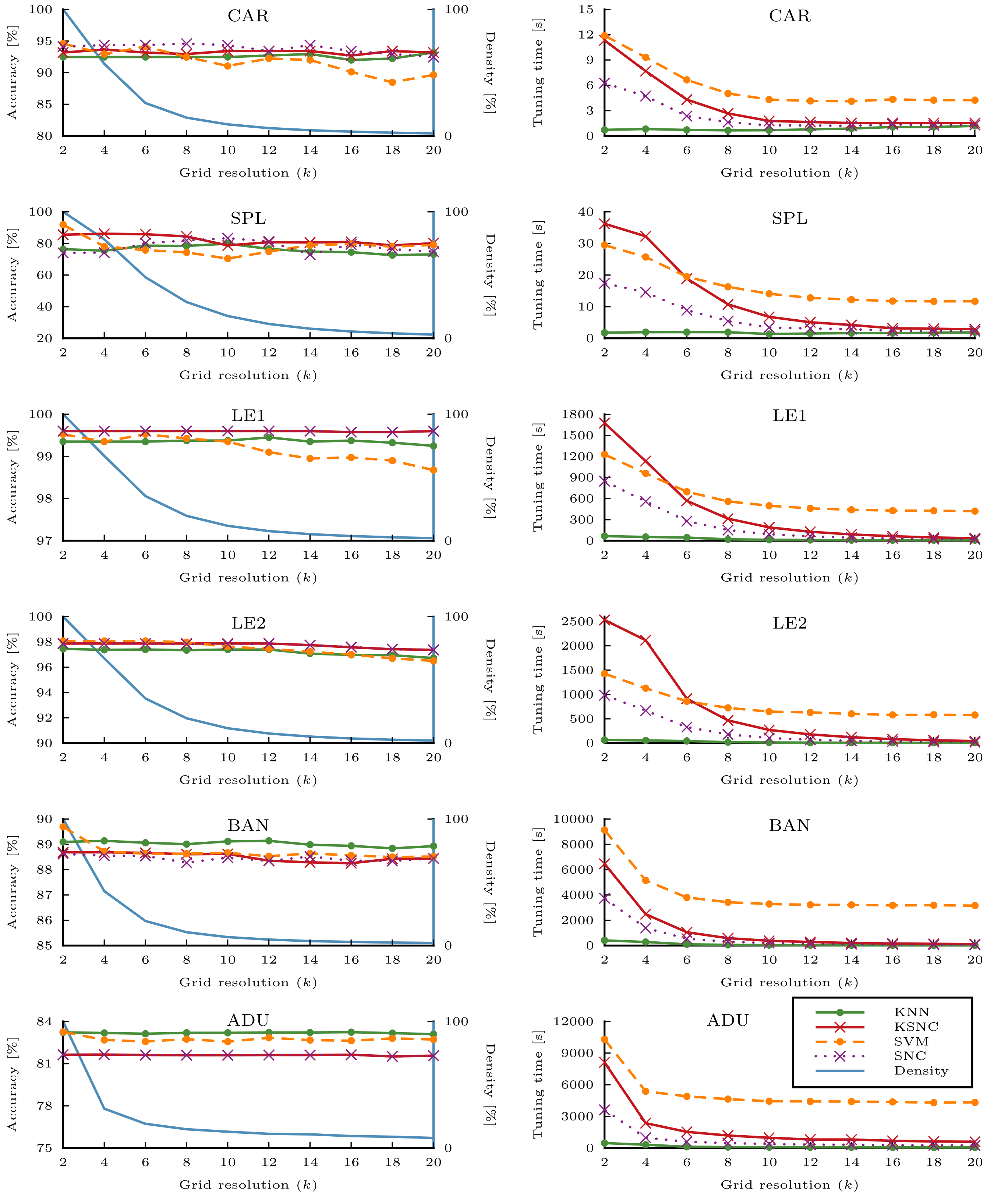


Fig. 8. Impact of grid resolution on accuracy, density and tuning times.

computation for COV with grid dimensionality $p = 2$ and grid resolutions $k = \{200, 250, \ldots, 600\}$. For grid resolutions below 200, even the sparse matrices require more than 256 GB of memory.

The testing results for the large data set COV are shown in Fig. 9 and Table 9. Fig. 9 shows the accuracy, density and tuning time results for KNN, KSNC, and SNC. Because the LIBSVM implementation internally converts a precomputed sparse kernel matrix into a full kernel matrix, we could not apply SVM to the set COV. In columns 13 to 22 of Table 9, we break down the testing time into different components. The column PCA states the time required to perform

TABLE 3
Testing Results for the Data Set CAR

			Accuracy [%]				F$_1$-score [%]				Tuning time [s]				Testing time [s]			
	k	Den [%]	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM
complete matrix	2	100.00	92.47	93.18	94.12	94.59	82.02	83.24	85.03	86.23	0.7	11.3	6.3	11.9	0.34	0.74	0.46	0.54
sparse matrices	4	56.97	92.47	93.65	94.35	92.94	82.02	84.57	85.71	81.01	0.8	7.7	4.7	9.3	0.37	0.46	0.36	0.52
	6	25.96	92.47	93.18	94.35	94.12	82.02	83.62	85.71	85.03	0.7	4.3	2.4	6.6	0.41	0.38	0.37	0.48
	8	14.40	92.47	92.94	94.59	92.47	82.02	83.52	86.39	81.48	0.7	2.7	1.6	5.0	0.36	0.34	0.37	0.38
	10	9.10	92.47	93.41	94.35	91.06	82.02	84.44	85.88	80.98	0.7	1.8	1.2	4.3	0.33	0.36	0.32	0.40
	12	6.13	92.71	93.41	93.41	92.24	82.49	84.27	83.33	80.24	0.8	1.6	1.2	4.1	0.39	0.42	0.39	0.53
	14	4.45	92.94	93.41	94.35	92.00	82.76	83.91	85.88	79.01	0.9	1.5	1.2	4.1	0.48	0.47	0.45	0.51
	16	3.36	92.00	92.71	93.41	90.12	80.68	81.56	82.93	73.42	1.1	1.5	1.4	4.3	0.47	0.49	0.49	0.55
	18	2.52	92.24	93.41	92.94	88.47	80.92	82.93	81.71	69.18	1.1	1.5	1.2	4.2	0.49	0.51	0.51	0.58
	20	2.00	93.18	93.18	92.47	89.65	83.43	83.43	81.18	76.13	1.2	1.5	1.3	4.2	0.51	0.51	0.51	0.56

TABLE 4
Testing Results for the Data Set SPL

			Accuracy [%]				F$_1$-score [%]				Tuning time [s]				Testing time [s]			
	k	Den [%]	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM
complete matrix	2	100.00	76.38	85.51	74.02	91.81	73.76	87.04	73.36	92.42	1.8	36.2	17.4	29.5	0.90	1.70	1.13	1.40
sparse matrices	4	78.22	75.43	86.14	74.17	78.11	72.04	87.53	78.59	80.39	1.9	32.3	14.6	25.7	1.03	1.45	1.09	1.41
	6	48.29	78.58	85.83	80.16	75.75	76.47	87.32	79.22	81.53	1.9	18.9	8.9	19.5	0.94	0.97	0.88	1.24
	8	28.66	78.43	84.41	81.89	74.33	76.42	86.64	82.71	80.80	1.9	10.7	5.4	16.3	0.96	0.83	0.65	1.22
	10	17.61	79.84	78.74	83.31	70.39	77.80	84.92	84.23	79.08	1.4	6.8	3.4	14.1	0.69	0.71	0.69	0.90
	12	11.33	76.54	80.79	81.26	74.80	75.04	83.36	82.32	77.27	1.5	5.1	3.1	12.8	0.79	0.82	0.74	1.18
	14	7.60	74.80	80.63	72.91	78.90	72.88	81.94	79.29	81.69	1.6	4.2	2.9	12.2	0.82	0.82	0.79	1.08
	16	5.38	74.49	80.94	79.37	79.68	72.54	82.59	80.30	82.96	1.6	3.2	2.4	11.8	0.90	0.85	0.81	1.08
	18	3.94	72.60	78.74	76.38	77.95	71.26	81.07	79.94	81.87	1.8	3.0	2.2	11.7	0.93	0.82	0.80	1.03
	20	2.92	73.07	80.16	74.80	79.06	72.49	82.40	78.21	83.01	1.9	2.9	2.2	11.7	0.96	0.88	0.87	1.10

TABLE 5
Testing Results for the Data Set LE1

			Accuracy [%]				F$_1$-score [%]				Tuning time [s]				Testing time [s]			
	k	Den [%]	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM
complete matrix	2	100.00	99.35	99.60	99.60	99.52	90.85	94.41	94.41	92.94	65.6	1,675.5	847.6	1,227.0	33.4	73.4	48.9	51.9
sparse matrices	4	67.03	99.35	99.60	99.60	99.35	90.85	94.41	94.41	90.08	53.8	1,132.0	559.8	958.6	31.1	45.5	33.0	47.0
	6	35.30	99.35	99.60	99.60	99.52	90.85	94.41	94.41	92.99	45.0	567.9	278.8	697.5	22.6	24.2	17.7	32.0
	8	19.66	99.38	99.60	99.60	99.42	91.23	94.41	94.41	91.25	20.4	314.1	151.9	560.6	11.0	13.0	9.7	19.8
	10	11.77	99.38	99.60	99.60	99.35	91.17	94.41	94.41	90.30	13.3	190.2	92.9	497.6	7.1	8.5	6.1	17.9
	12	7.61	99.45	99.60	99.60	99.10	92.14	94.41	94.41	85.94	10.9	127.5	61.7	460.7	5.4	6.3	4.8	11.8
	14	5.19	99.35	99.60	99.60	98.95	90.78	94.41	94.41	82.31	8.6	89.9	43.7	440.8	4.6	4.8	4.0	10.5
	16	3.68	99.38	99.58	99.58	98.98	91.17	94.08	94.08	83.46	8.1	64.2	31.7	428.9	4.4	4.6	3.8	9.9
	18	2.69	99.33	99.58	99.58	98.90	90.53	92.78	92.78	81.97	7.9	46.3	24.3	426.4	4.2	4.2	3.5	9.6
	20	2.04	99.25	99.60	99.60	98.67	88.97	94.37	94.37	78.54	7.0	35.7	20.3	421.3	4.0	3.0	3.6	9.6

approximate-PCA, and the column Grid lists the time required to construct the grid and to identify neighboring blocks. These two tasks are identical for all classifiers. The columns under similarity list for each classifier the time required to compute euclidean distances and convert them into similarities. The columns under classification list the time required for the classification, which includes identifying the K-nearest neighbors for KNN and KSNC and constructing the similarity graph and performing the minimum cut for KSNC and SNC. The last three columns state the total testing times for each classifier.

The results show that all classifiers are able to achieve an accuracy of at least 97.24 percent with grid resolution $k = 200$. The similarity matrix obtained with this grid resolution has a density of 0.55 percent. Further increasing the grid resolution from $k = 200$ to $k = 600$ reduces the density to 0.07 percent. Even with such a small density, it is possible to achieve accuracies of at least 96.48 percent. The size of the similarity matrix obtained with a grid resolution of 600 is less than 1.2 GB, which allows for fast processing even on standard notebooks. Interestingly, KSNC and SNC achieve the same accuracy and F_1-score results for most grid resolutions. This happens when both classifiers select the value zero for tuning parameter λ. Again, the tuning time of KSNC is most affected by the grid resolution.

TABLE 6
Testing Results for the Data Set LE2

			Accuracy [%]				F_1-score [%]				Tuning time [s]				Testing time [s]			
	k	Den [%]	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM
complete matrix	2	100.00	97.45	97.88	97.88	98.08	97.48	97.90	97.85	98.10	65.5	2,531.8	984.5	1,427.7	34.8	70.9	52.6	63.8
sparse matrices	4	67.03	97.38	97.88	97.88	98.08	97.40	97.90	97.85	98.10	55.7	2,115.0	669.6	1,129.2	29.7	46.1	33.0	54.2
	6	35.30	97.40	97.88	97.88	98.08	97.43	97.90	97.85	98.10	43.5	914.4	329.3	859.7	22.4	24.7	18.5	42.2
	8	19.66	97.35	97.88	97.82	97.97	97.44	97.90	97.85	97.00	20.2	467.9	179.6	724.9	11.1	13.4	9.3	29.5
	10	11.77	97.40	97.88	97.82	97.63	97.43	97.90	97.85	97.64	13.3	271.4	105.0	648.3	7.2	8.6	6.5	26.0
	12	7.61	97.40	97.88	97.88	97.45	97.43	97.90	97.85	97.46	10.2	179.4	68.9	631.1	5.4	6.1	4.8	22.2
	14	5.19	97.08	97.75	97.75	97.22	97.10	97.78	97.78	97.23	8.8	121.6	47.3	602.1	4.6	5.0	4.1	22.4
	16	3.68	96.97	97.58	97.60	97.00	97.01	97.60	97.63	97.00	8.2	81.4	36.2	580.2	4.3	4.5	3.9	21.7
	18	2.69	96.95	97.42	97.42	96.70	96.98	97.45	97.45	96.69	7.8	58.5	26.2	583.9	4.2	4.0	3.6	21.4
	20	2.04	96.73	97.38	97.38	96.50	96.75	97.39	97.30	96.48	7.9	43.6	20.6	579.0	4.2	3.0	3.7	19.0

TABLE 7
Testing Results for the Data Set BAN

			Accuracy [%]				F_1-score [%]				Tuning time [s]				Testing time [s]			
	k	Den [%]	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM
complete matrix	2	100.00	89.10	88.69	88.62	89.69	36.62	43.04	36.13	35.28	410	6,465	3,739	9,135	228	407	328	350
sparse matrices	4	43.02	89.14	88.69	88.55	88.72	35.07	43.07	35.25	30.31	282	2,482	1,384	5,150	148	154	125	230
	6	19.42	89.06	88.66	88.54	88.63	37.66	43.13	35.86	21.04	100	1,053	543	3,802	64	67	50	113
	8	10.54	89.01	88.61	88.28	88.63	36.97	43.02	35.78	27.08	55	583	309	3,426	31	36	26	81
	10	6.67	89.12	88.62	88.48	88.66	35.26	42.66	35.00	23.26	37	377	193	3,284	20	22	17	66
	12	4.73	89.14	88.35	88.34	88.53	36.32	42.29	36.13	22.91	28	284	149	3,220	15	17	13	57
	14	3.54	88.98	88.29	88.52	88.64	36.28	42.70	34.67	21.77	22	201	104	3,214	12	13	10	57
	16	2.87	88.94	88.25	88.35	88.56	35.36	41.57	35.73	24.08	19	162	90	3,175	10	12	9	55
	18	2.38	88.84	88.43	88.34	88.50	35.25	41.78	36.00	24.36	18	139	78	3,190	9	10	8	55
	20	2.05	88.93	88.44	88.44	88.52	35.63	42.07	34.69	25.68	16	115	65	3,156	8	9	7	52

TABLE 8
Testing Results for the Data Set ADU

			Accuracy [%]				F_1-score [%]				Tuning time [s]				Testing time [s]			
	k	Den [%]	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM	KNN	KSNC	SNC	SVM
complete matrix	2	100.00	83.23	81.63	81.63	83.26	63.31	62.20	60.31	62.06	466	8,122	3,616	10,296	245	448	324	400
sparse matrices	4	31.06	83.18	81.65	81.65	82.68	63.23	62.25	59.57	59.37	307	2,355	974	5,377	166	121	91	230
	6	19.13	83.13	81.61	81.61	82.57	62.60	62.17	59.81	59.70	113	1,529	593	4,897	59	72	53	113
	8	14.81	83.19	81.60	81.60	82.74	62.75	62.14	60.89	58.77	88	1,183	463	4,636	46	57	43	105
	10	12.85	83.19	81.60	81.60	82.56	62.76	62.13	61.01	58.85	75	968	371	4,437	38	49	40	88
	12	11.19	83.22	81.61	81.61	82.84	62.83	62.19	60.32	59.17	62	803	317	4,412	34	41	30	83
	14	10.78	83.22	81.61	81.61	82.67	62.54	62.15	60.28	59.62	59	803	309	4,403	34	40	30	81
	16	9.41	83.24	81.63	81.63	82.63	62.79	62.12	60.27	59.43	52	680	261	4,369	30	36	27	77
	18	8.87	83.17	81.51	81.51	82.80	62.96	62.05	60.02	59.83	48	617	255	4,297	28	33	25	75
	20	7.93	83.09	81.57	81.57	82.72	62.78	62.18	60.17	59.97	49	591	241	4,329	25	30	23	103

From the breakdown of testing times, we can conclude that most of the running time is required for input preparation. While the time required for similarity computations decreases with increasing grid resolutions, the time required to construct the grid and to identify neighboring blocks increases with increasing grid resolutions and becomes the dominant part of the testing times for grid resolutions greater than 550. The approximate-PCA and the classification task together require insignificant running times.

5.3.2 *Impact of Grid Dimensionality*

We report the effect of choosing the grid dimension p on the accuracy, F_1-score, density and tuning times for the data set CAR and classifier SNC. Similar results were obtained for the other data sets and classifiers. The impact of parameter p is analyzed for different grid resolutions k, i.e., we tune and test SNC with different values for both parameters p and k. The grid resolution k was selected from the set $\{2, \ldots, 20\}$, and the grid dimension p was selected from the set $\{1, \ldots, 4\}$. Parameter r, which defines the number of randomly selected rows for approximate-PCA, was kept constant at 150. Column sampling for approximate-PCA was not performed. The resulting first four principal components account for 31.2, 18.3, 13.5, and 9.5 percent of the variance in the corresponding submatrix. When PCA is applied to the entire evaluation set (without selecting a subset of r

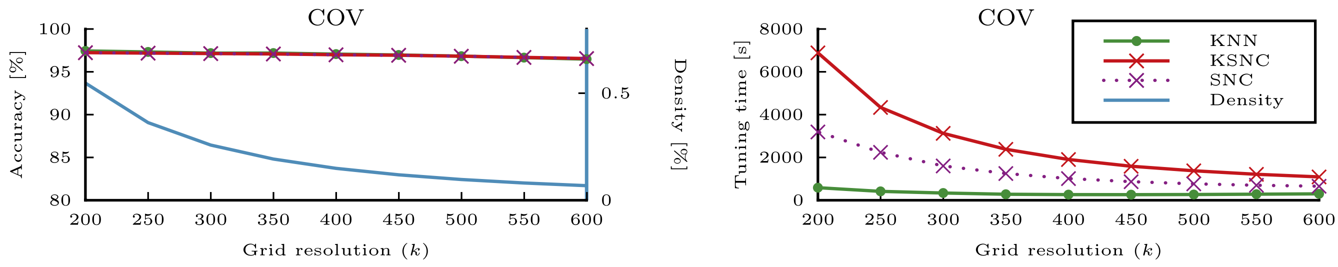


Fig. 9. Impact of grid resolution on accuracy, density and tuning times for the COV data set.

TABLE 9
Testing Results for the Data Set COV

											Testing time [s]										
											Input preparation										
		Accuracy [%]			F$_1$-score [%]			Tuning time [s]					Similarity			Classification			Total		
k	Den [%]	KNN	KSNC	SNC	KNN	KSNC	SNC	KNN	KSNC	SNC	PCA	Grid	KNN	KSNC	SNC	KNN	KSNC	SNC	KNN	KSNC	SNC
200	0.55	97.44	97.24	97.24	96.48	96.21	96.20	587	6,888	3,192	0.4	11	277	266	224	2.0	7.3	5.0	291	285	240
250	0.36	97.32	97.19	97.19	96.31	96.15	96.13	415	4,337	2,240	0.5	17	186	178	153	1.6	2.9	1.3	205	199	171
300	0.26	97.19	97.13	97.13	96.12	96.05	96.05	338	3,127	1,604	0.6	23	144	127	121	1.4	2.4	1.1	170	153	146
350	0.19	97.19	97.09	97.09	96.14	95.99	95.99	280	2,381	1,250	0.6	32	116	99	86	1.6	1.9	0.9	150	134	120
400	0.15	97.07	96.99	96.99	95.97	95.86	95.86	264	1,905	1,014	0.5	42	90	84	74	1.3	2.7	2.0	134	129	118
450	0.12	96.97	96.94	96.94	95.83	95.78	95.78	261	1,589	869	0.5	52	77	71	63	1.1	2.4	1.6	131	126	117
500	0.10	96.83	96.82	96.82	95.64	95.63	95.63	268	1,377	767	0.5	64	68	61	55	1.2	2.0	1.3	134	128	121
550	0.08	96.66	96.69	96.69	95.40	95.44	95.44	283	1,213	700	0.6	77	62	56	51	1.0	1.8	1.1	142	136	131
600	0.07	96.48	96.55	96.55	95.15	95.25	95.25	299	1,091	655	0.5	90	58	52	48	1.0	1.6	1.0	151	144	140

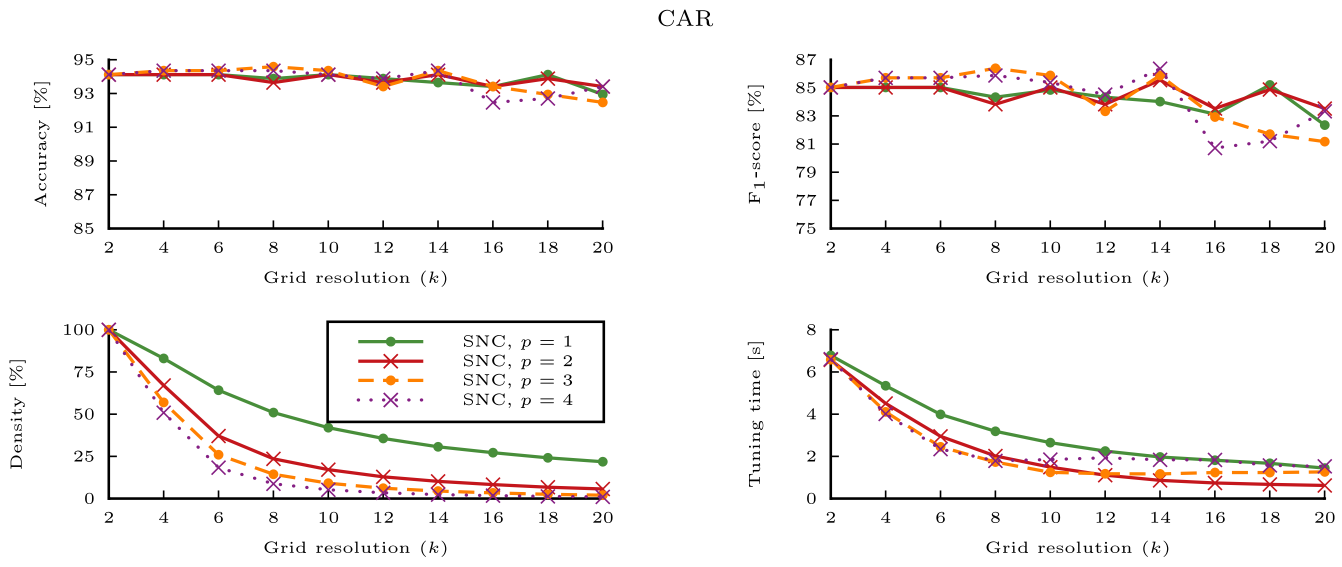


Fig. 10. Impact of grid dimensionality on accuracy, F$_1$-score, density and tuning times for the CAR data set.

rows), the first four principal components account for very similar fractions of the total variance in the evaluation set, namely 33.28, 18.50, 11.30, and 8.78 percent.

Fig. 10 shows four plots in which the accuracy (top left), F_1-scores (top right), densities (bottom left) and tuning times (bottom right) are visualized for different values of p and k. Each curve represents a different value of p. The same results are presented in tabular form in Table 10. Increasing the grid dimensionality for a fixed value of k can only decrease the density. As shown in the bottom left plot of Fig. 10, the marginal reduction in density obtained by adding an additional dimension to the grid decreases. Regarding the impact of the grid dimensionality on the accuracy and F_1-score results, no consistent pattern can be observed, and thus, choosing a 1-dimensional grid appears to deliver roughly the same level of accuracy and F_1-scores.

5.3.3 Impact of Row Selection

The impact of the number of rows r selected for approximate-PCA is reported for data set SPL and classifier KSNC. Similar results were obtained for the other data sets and classifiers, and the impact of the r value was most noticeable for the set SPL. We again studied the impact of parameter r for different grid resolutions k, i.e., we tuned and tested

TABLE 10
Sensitivity Analysis for the Data Set CAR: Impact of Grid Dimensionality

		Accuracy [%]				F_1-score [%]				Tuning time [s]				Density [%]			
	k	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
complete matrix	2	94.12	94.12	94.12	94.12	85.03	85.03	85.03	85.03	6.79	6.57	6.58	6.61	100.00	100.00	100.00	100.00
sparse matrices	4	94.12	94.12	94.35	94.35	85.03	85.03	85.71	85.71	5.35	4.51	4.11	4.01	83.02	67.05	56.97	50.90
	6	94.12	94.12	94.35	94.35	85.03	85.03	85.71	85.71	3.99	2.95	2.45	2.34	64.18	37.09	25.96	18.34
	8	93.88	93.65	94.59	94.35	84.34	83.83	86.39	85.88	3.19	2.02	1.73	1.79	50.95	23.56	14.40	8.76
	10	94.12	94.12	94.35	94.12	84.85	85.03	85.88	85.38	2.65	1.49	1.24	1.85	41.99	17.15	9.10	5.20
	12	93.88	93.65	93.41	93.88	84.34	83.83	83.33	84.52	2.25	1.11	1.18	1.93	35.62	12.88	6.13	3.44
	14	93.65	94.12	94.35	94.35	84.02	85.55	85.88	86.36	1.97	0.86	1.17	1.84	30.67	10.19	4.45	2.35
	16	93.41	93.41	93.41	92.47	83.13	83.53	82.93	80.72	1.82	0.74	1.23	1.83	27.19	8.28	3.36	1.75
	18	94.12	93.88	92.94	92.71	85.21	84.88	81.71	81.21	1.67	0.67	1.23	1.57	24.14	6.73	2.52	1.29
	20	92.94	93.41	92.47	93.41	82.35	83.53	81.18	83.33	1.44	0.62	1.26	1.53	21.82	5.69	2.00	1.01

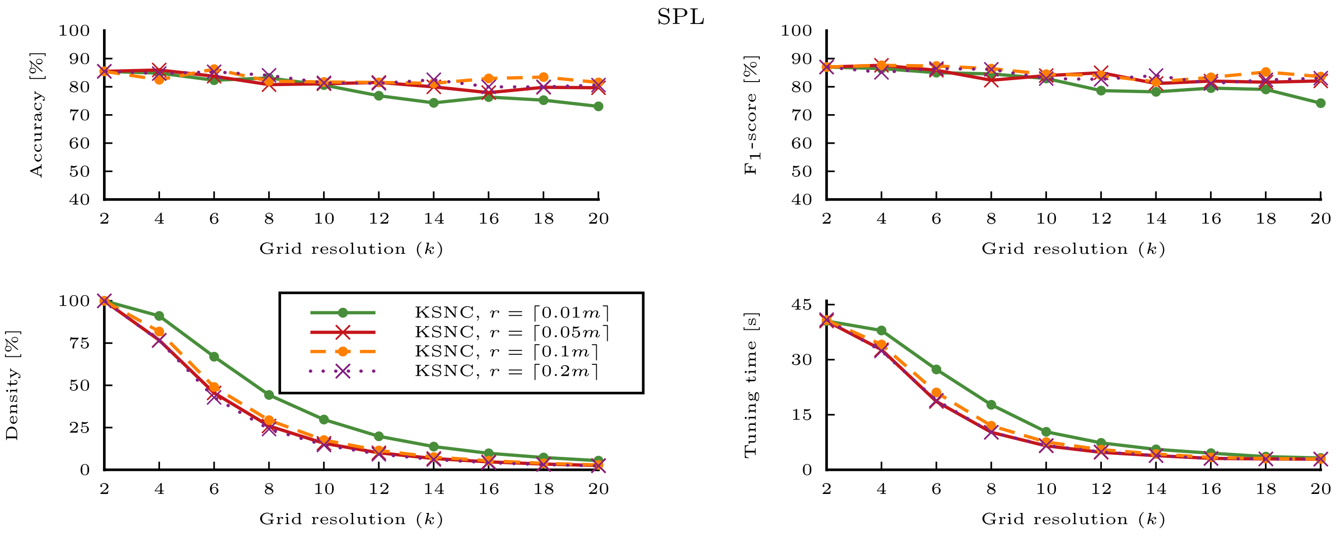


Fig. 11. Impact of the number of selected rows for approximate-PCA on accuracy, F_1-score, density and tuning times for the SPL data set.

TABLE 11
Sensitivity Analysis for the Data set SPL: Impact of the Fraction of Selected Rows for Approximate-PCA

		Accuracy [%]				F_1-score [%]				Tuning time [s]				Density [%]			
	k	0.01	0.05	0.1	0.2	0.01	0.05	0.1	0.2	0.01	0.05	0.1	0.2	0.01	0.05	0.1	0.2
complete matrix	2	85.51	85.51	85.51	85.51	87.04	87.04	87.04	87.04	40.51	40.50	40.77	40.88	100.00	100.00	100.00	100.00
sparse matrices	4	84.88	85.98	82.52	84.72	86.48	87.66	87.66	85.15	37.93	32.71	34.13	32.36	91.08	76.72	81.85	76.49
	6	82.36	83.78	86.30	85.35	85.05	85.91	87.41	86.50	27.32	18.59	21.06	18.93	66.94	45.49	49.01	42.84
	8	83.15	80.79	82.05	84.09	84.60	82.37	86.48	86.22	17.71	10.21	11.99	10.22	44.30	25.94	29.36	24.11
	10	80.63	81.10	81.73	81.42	82.99	84.00	84.49	82.95	10.34	6.51	7.56	6.62	29.74	15.67	17.61	14.68
	12	76.85	81.57	81.57	81.26	78.66	85.01	84.07	82.68	7.33	4.76	5.50	4.88	19.83	10.07	11.46	9.21
	14	74.33	80.00	81.26	82.52	78.24	81.19	81.78	84.03	5.57	3.86	4.31	3.86	13.75	6.69	7.54	6.09
	16	76.38	77.95	82.99	80.00	79.55	82.05	83.38	81.19	4.53	3.11	3.38	3.07	9.78	4.70	5.37	4.23
	18	75.28	79.84	83.46	80.00	79.09	81.66	85.27	82.49	3.59	2.96	3.06	2.94	7.27	3.41	3.84	3.07
	20	73.07	79.68	81.57	80.63	74.21	82.11	83.64	83.08	3.22	2.88	2.95	2.94	5.43	2.54	2.87	2.32

KSNC with different values for both parameters r and k. The grid resolution k was selected from the set $\{2, \ldots, 20\}$, and the parameter r was selected from the set $\{\lceil 0.01\,m\rceil, \lceil 0.05\,m\rceil, \lceil 0.1\,m\rceil, \lceil 0.2\,m\rceil\}$, where m denotes the number of rows in the tuning set or the evaluation set, which both have 1,701 rows in total. In contrast to the experiments in the previous sections, we did not impose a lower bound on r for this analysis. The grid dimensionality p was kept constant at 3. Column sampling for approximate-PCA was not performed.

As can be seen from Fig. 11 and Table 11, the impact of r on the accuracy, F_1-scores, densities, and tuning times is rather limited. It appears that setting $r = \lceil 0.01\,m\rceil$ results in a slightly worse accuracy and F_1-score for higher grid resolutions. For this reason, we selected at least 150 rows for sets CAR and SPL in the

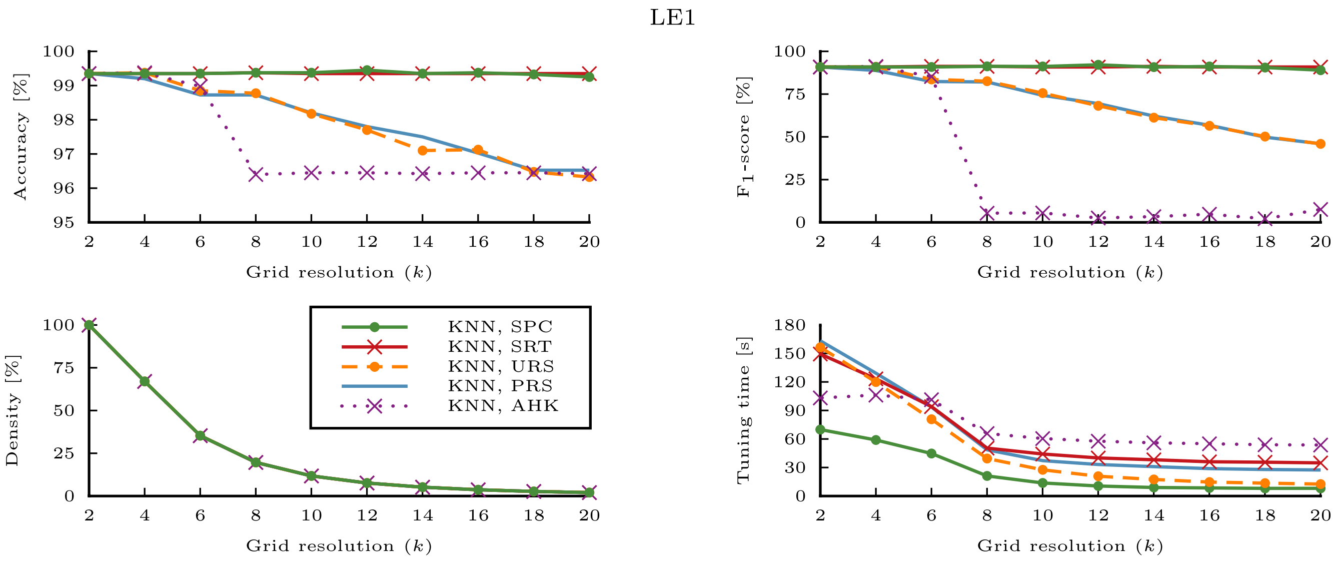


Fig. 12. Comparison with alternative sparsification approaches.

experiments discussed in the previous sections. Increasing r also resulted in lower densities.

5.3.4 Comparison with Alternative Sparsification Approaches

We compare sparse computation (SPC) to four alternative sparsification approaches on the data set LE1 with classifier KNN. The benchmark sparsification approaches used here are referred to as SRT, URS, PRS, and AHK. The approach SRT sorts all similarity values of the complete matrix and sets the smallest values to zero such that a prespecified target density is obtained. The approaches URS and PRS randomly sample entries from the complete similarity matrix to obtain a sparse similarity matrix with a prescribed target density. URS and PRS differ in the selection probabilities of the entries. URS employs uniform probabilities, while PRS uses probabilities that are proportional to the magnitude of the entries. The random sampling implementations of URS and PRS are different. URS generates a vector containing t unique integers selected randomly from 1 to $\binom{m}{2}$, where m denotes the number of rows/columns of the similarity matrix. PRS first computes all similarities which serve as weights for the algorithm of [40]. This algorithm then determines a weighted random sample without replacement of size t. AHK is the random-sampling-based sparsification algorithm of [8], which has been designed for efficient computation of approximate eigenvectors. The approach AHK runs in a single pass over the similarity matrix and sets entries to zero according to probabilities that are proportional to the magnitude of the entry. In the algorithm of [8], some entries are quantized, i.e., their original values are replaced with either a positive or a negative constant, depending on the sign of the original entry.

We applied KNN in combination with each of these sparsification approaches. For a meaningful comparison, we first applied sparse computation with $r = 150$, $p = 3$, and $k = 2, 4, \ldots, 20$. Column sampling for approximate-PCA was not performed. The densities of the resulting similarity matrices were then used as the target densities for the benchmark sparsification approaches. The sparsification approaches SRT, URS, and PRS always reach the target density exactly, as we can define the number of entries to be retained. It is possible that the density of the sparse matrix computed with AHK differs from the target density because we generate a random number for each entry of the matrix and select only those entries for which the random number is equal to or greater than the respective selection probability. However, it turns out that this difference is rather small.

Fig. 12 and Table 12 display the accuracy, F_1-score values, tuning times and densities for KNN applied with each

TABLE 12
Comparison of Alternative Sparsification Approaches

	Accuracy [%]					F_1-score [%]					Tuning time [s]					Density [%]				
k	SPC	SRT	URS	PRS	AHK	SPC	SRT	URS	PRS	AHK	SPC	SRT	URS	PRS	AHK	SPC	SRT	URS	PRS	AHK
2	99.35	99.35	99.35	99.35	99.35	90.85	90.85	90.85	90.85	90.85	69.96	149.48	156.50	163.35	103.27	100.00	100.00	100.00	100.00	99.86
4	99.35	99.35	99.38	99.20	99.38	90.85	90.85	90.97	88.81	91.17	58.99	123.36	119.98	129.27	106.26	67.03	67.03	67.03	67.03	67.02
6	99.35	99.35	98.85	98.72	98.98	90.85	91.23	83.57	82.35	85.30	44.75	94.12	80.81	93.66	101.43	35.30	35.30	35.30	35.30	35.30
8	99.38	99.38	98.78	98.72	96.40	91.23	91.23	82.56	82.11	5.39	21.16	50.44	39.50	48.63	65.82	19.66	19.66	19.66	19.66	19.66
10	99.38	99.35	98.17	98.20	96.45	91.17	90.85	75.59	74.27	5.52	13.76	44.23	27.59	37.26	60.52	11.77	11.77	11.77	11.77	11.77
12	99.45	99.35	97.70	97.80	96.45	92.14	90.85	68.15	69.44	2.59	10.56	40.09	20.84	33.19	57.74	7.61	7.61	7.61	7.61	7.62
14	99.35	99.35	97.10	97.50	96.43	90.78	91.23	61.21	62.15	3.37	9.00	38.17	17.45	30.95	56.01	5.19	5.19	5.19	5.19	5.20
16	99.38	99.35	97.13	97.02	96.45	91.17	90.85	56.45	56.79	4.79	8.54	36.07	14.71	28.86	55.05	3.68	3.68	3.68	3.68	3.68
18	99.33	99.35	96.47	96.53	96.45	90.53	90.85	50.18	49.82	2.14	8.07	35.61	13.57	27.95	53.98	2.69	2.69	2.69	2.69	2.70
20	99.25	99.35	96.33	96.53	96.43	88.97	90.85	45.90	45.98	7.57	8.11	34.90	12.58	27.43	53.70	2.04	2.04	2.04	2.04	2.04

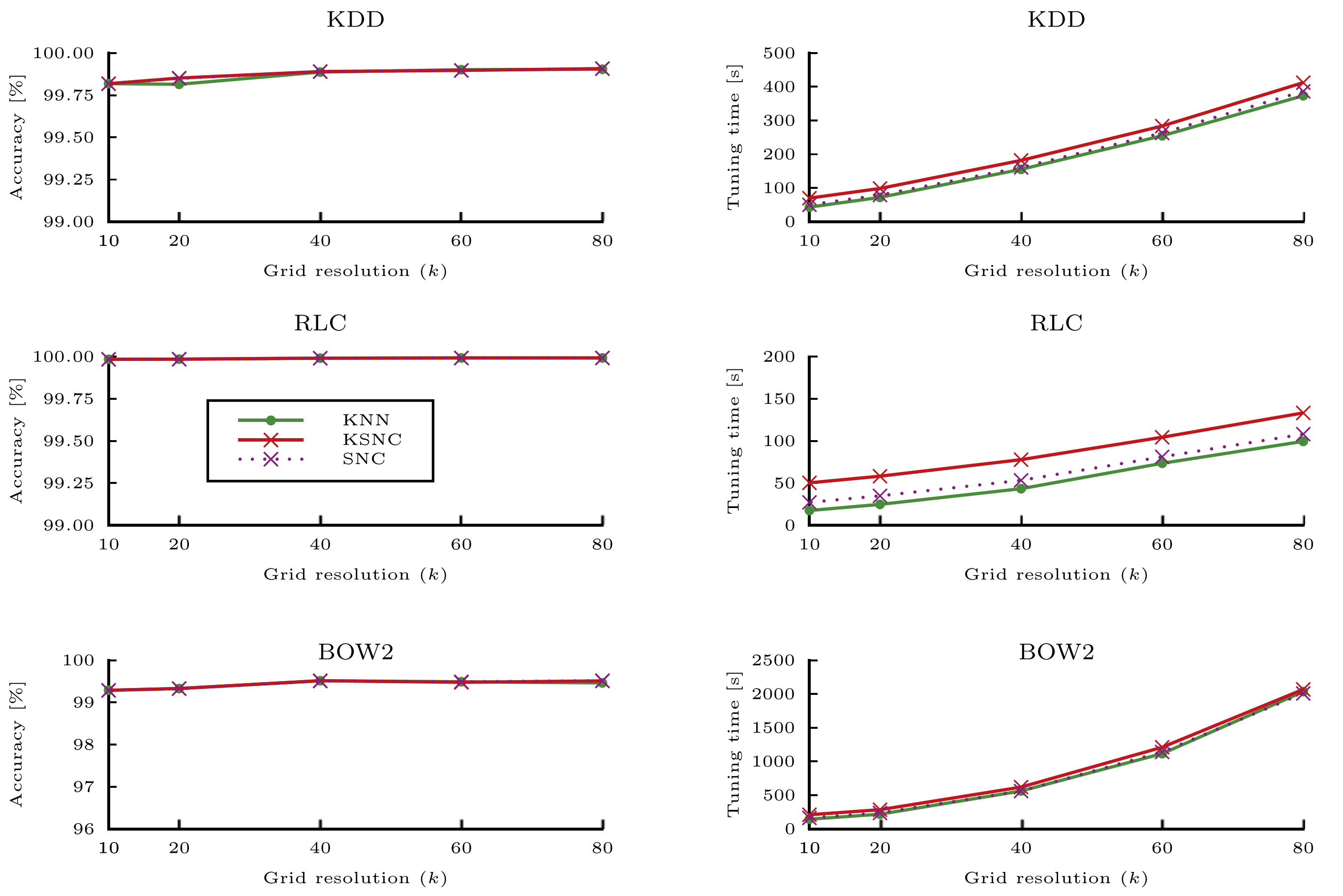


Fig. 13. Testing results for the data sets KDD, RLC and BOW2 using δ-identical grouping.

TABLE 13
Testing Results for the Data Set KDD

										Testing time [s]										
										Input preparation										
	Accuracy [%]			F_1-score [%]			Tuning time [s]					Similarity			Classification			Total		
k	KNN	KSNC	SNC	KNN	KSNC	SNC	KNN	KSNC	SNC	PCA	Grid	KNN	KSNC	SNC	KNN	KSNC	SNC	KNN	KSNC	SNC
10	99.818	99.818	99.818	99.89	99.89	99.89	43	70	50	8.5	7.3	0.2	0.2	0.2	<0.1	<0.1	<0.1	19.9	20.2	20.6
20	99.815	99.852	99.850	99.88	99.91	99.91	72	99	79	8.5	21.9	0.6	0.6	0.6	<0.1	<0.1	<0.1	35.1	35.2	35.9
40	99.887	99.890	99.890	99.93	99.93	99.93	155	182	161	8.5	59.3	1.7	1.6	1.6	0.1	0.1	<0.1	75.9	78.6	75.2
60	99.901	99.897	99.897	99.94	99.94	99.94	255	284	263	8.6	104.3	2.9	2.9	2.9	0.1	0.1	<0.1	126.9	127.6	122.5
80	99.904	99.908	99.908	99.94	99.94	99.94	374	412	387	8.5	162.9	4.3	4.4	4.3	0.1	0.1	<0.1	184.8	186.6	183.2

of the five sparsification approaches. The main conclusion is that only SPC and SRT retain a good accuracy for the sparse matrices. However in terms of the running time performance, SRT is inferior to sparse computation. We conclude that for the purpose of sparsifying similarity matrices for data sets, sparse computation is the leading technique.

5.3.5 Using δ-Identical Grouping

The complete similarity matrices for the KDD, RLC, and BOW2 data sets comprise $15,356$, $21,153$, and $46,725$ billion entries, respectively. For these data sets, we set δ to be 1, i.e., in each grid block, we replace all positive training objects by a single representative, all negative objects by a single representative and all testing objects by a single representative. The total number of representatives increases with increasing grid resolution because a higher grid resolution leads to a higher number of grid blocks. The values for the remaining parameters of sparse computation are $c = d$ for KDD and RLC and $c = \lceil 0.01\, d \rceil$ for BOW2 with d denoting the number of attributes of the corresponding data set, $k = \{10, 20, 40, 60, 80\}$, $p = 3$, and $r = \lceil 0.01\, m \rceil$, where m denotes the number of rows in the tuning set or the evaluation set. For the high-dimensional data set BOW2, we computed the similarities with respect to the low-dimensional space.

The KDD data set is unusual in that it contains many groups of objects that are identical in the original space. They are therefore identical in the projected space, and by our terminology, they are 0-identical. These objects will be grouped into δ-identical sets for any positive δ.

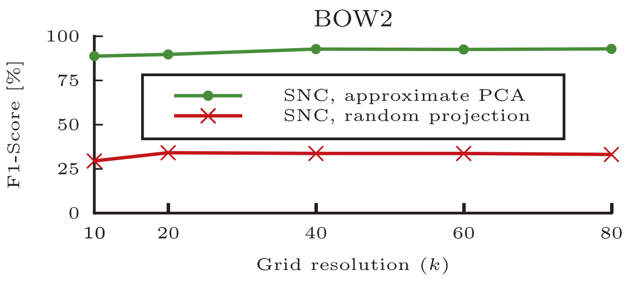
Fig. 13 and Tables 13, 14, 15 present the testing results for the KDD, RLC, and BOW2 data sets. Fig. 13 shows the accuracy and the tuning time results for KNN, KSNC, and SNC.

TABLE 14
Testing Results for the Data Set RLC

										Testing time [s]										
										Input preparation										
	Accuracy [%]			F_1-score [%]			Tuning time [s]					Similarity			Classification			Total		
k	KNN	KSNC	SNC	KNN	KSNC	SNC	KNN	KSNC	SNC	PCA	Grid	KNN	KSNC	SNC	KNN	KSNC	SNC	KNN	KSNC	SNC
10	99.984	99.984	99.984	97.76	97.76	97.76	18	50	27	1.3	5.1	< 0.1	< 0.1	< 0.1	0.1	< 0.1	< 0.1	8.5	7.9	8.4
20	99.985	99.985	99.985	97.93	97.87	97.83	25	58	35	1.7	10.2	0.1	0.1	0.1	< 0.1	< 0.1	< 0.1	13.6	14.1	14.2
40	99.991	99.991	99.991	98.74	98.72	98.71	43	78	53	1.3	23.7	0.4	0.4	0.4	0.1	0.1	< 0.1	27.8	28.6	28.1
60	99.992	99.992	99.992	98.93	98.91	98.95	74	104	81	1.2	44.0	0.9	0.7	0.7	0.1	0.1	< 0.1	48.0	48.6	48.9
80	99.992	99.992	99.992	98.90	98.93	98.93	100	133	108	1.2	66.0	1.3	1.1	1.1	0.1	0.1	< 0.1	70.9	71.5	71.5

TABLE 15
Testing Results for the Data Set BOW2

										Testing time [s]										
										Input preparation										
	Accuracy [%]			F_1-score [%]			Tuning time [s]					Similarity			Classification			Total		
k	KNN	KSNC	SNC	KNN	KSNC	SNC	KNN	KSNC	SNC	PCA	Grid	KNN	KSNC	SNC	KNN	KSNC	SNC	KNN	KSNC	SNC
10	99.288	99.288	99.284	88.80	88.81	88.76	144	210	163	53.0	14.3	0.2	0.2	0.2	0.1	0.1	0.1	69.9	69.6	69.0
20	99.329	99.329	99.329	89.70	89.74	89.71	218	284	237	52.9	46.8	0.8	0.7	0.7	0.1	0.1	0.1	102.1	102.6	102.3
40	99.513	99.515	99.507	92.80	92.84	92.74	562	620	561	53.2	191.6	3.9	3.8	3.7	0.1	0.1	0.1	273.6	251.0	251.5
60	99.490	99.477	99.490	92.44	92.17	92.53	1,116	1,210	1,141	52.6	456.3	9.6	9.3	9.3	0.2	0.2	0.1	523.4	523.0	522.9
80	99.462	99.511	99.516	92.12	92.75	92.85	2,035	2,068	2,008	53.7	851.0	17.6	17.6	17.3	0.3	0.3	0.2	918.5	935.7	929.6



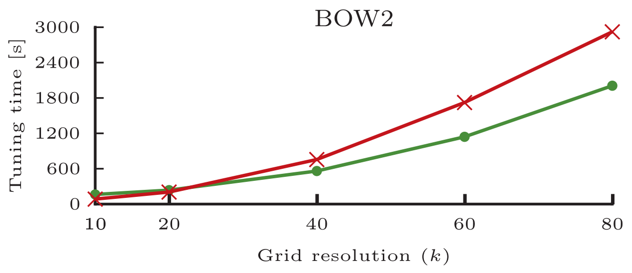


Fig. 14. Approximate-PCA versus random projections.

Tables 13-15 provide the corresponding numerical values and also state the F_1-scores and display a break-down of the testing times. As can be seen from the accuracy and F_1-score results, even for small grid resolutions (small number of representatives) the classification performance of all classifiers is surprisingly high. Increasing the grid resolution (increasing the number of representatives) improves the performance but, at the same time, increases the tuning and testing times of the algorithms. As the breakdown of the testing times shows, the increase in running times is due to the grid construction and the similarity computations. The last three columns of Table 13 contain the total testing times, which include, in addition to the sum of columns 11-18, the time required to identify the representatives.

When using δ-identical grouping, the quality of the projection becomes important because objects of the same type that are close together in the low-dimensional space may be grouped together. Fig. 14 and Table 16 compare for the BOW2 data set the testing results obtained with SNC and approximate-PCA to the testing results obtained with SNC and random projections.

5.3.6 Recommendations for Parameter Selection

We first note that there are no conclusive results on whether or not to normalize the values of the attributes. Based on our experiments with different parameter settings, the following selection of parameter values appears to work best. The row selection parameter r is set to $r = \max(150, \lceil 0.01\, m \rceil)$, where m denotes the number of rows in the tuning set or the evaluation set. For high-dimensional data sets with more than $d = 150$ attributes, the column selection parameter c is set to $c = \max(150, \lceil 0.01\, d \rceil)$. The grid dimensionality is set to

TABLE 16
Comparison of Approximate-PCA (APCA) and Random Projections (RP) on the Data Set BOW2

	Accuracy [%]		F1-score [%]		Tuning time [s]	
k	APCA	RP	APCA	RP	APCA	RP
10	99.28	96.74	88.76	29.51	163.30	85.08
20	99.33	96.82	89.71	34.14	236.67	204.64
40	99.51	96.84	92.74	33.72	560.99	756.08
60	99.49	96.86	92.53	33.72	1,141.00	1,723.09
80	99.52	96.86	92.85	33.16	2,007.76	2,922.80

$p = 2$ for large-scale data sets with more than half a million objects and to $p = 3$ for data sets with less than half a million objects. The grid resolution k has the strongest impact on the density of the similarity matrix and should also be selected based on the size of the data set (higher k for larger data sets). For data sets with up to 100,000 objects, we recommend to set k to a value between 20 and 50. For data sets between 100,000 and 1 million objects, we recommend to set k to a value between 50 and 600 and for data sets with more than 1 million objects, we recommend to set k to a value larger than 600. The δ-identical grouping should be used when the data sets contain a large number of identical or highly-similar objects. This is detected when the density of the similarity matrix remains high even for large values of k. For large values of k it is usually sufficient to set $\delta = 1$ as the blocks are already small enough and do not need to be partitioned further into even smaller sub-blocks (see [41]). For further refinement, we recommend to perform a parameter tuning based on a validation set.

6 Conclusions

Similarity-based algorithms have not been used for large-scale data mining due to the substantial computational effort that is required to generate a complete similarity matrix for a large-scale data set. Here, we propose a novel method, referred to as sparse computation, that provides practical efficiency for similarity-based algorithms while retaining their performance. The method generates only the relevant similarities without performing all pairwise comparisons between objects in the data set. Sparse computation is implemented with an efficient algorithm, named "Approximate Principal Component Analysis" that projects the data onto a low-dimensional space. In the resulting low-dimensional space, grid neighborhoods are applied in order to identify groups of objects with potentially high similarity. Our empirical analysis demonstrates that sparse computation significantly improves running times, with minimal loss in accuracy and F_1-scores.

For future research, we intend to consider local sample density during the grid construction, i.e., using a higher grid resolution in dense regions, and to use sparse computation to find isolated components in a data set, which allows the machine learning task to be performed separately for each of the isolated components.

References

[1] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. 13, no. 1, pp. 21–27, Jan. 1967.

[2] D. S. Hochbaum, "Polynomial time algorithms for ratio regions and a variant of normalized cut," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 5, pp. 889–898, May 2010.

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

[4] 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 J. Comput.*, vol. 26, no. 1, pp. 45–58, 2013.

[5] D. S. Hochbaum, C. Lu, and E. Bertelli, "Evaluating performance of image segmentation criteria and techniques," *EURO J. Comput. Optim.*, vol. 1, pp. 155–180, 2013.

[6] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, Cambridge MA, USA: MIT Press, 2001.

[7] P. Baumann, D. S. Hochbaum, and Y. T. Yang, "A comparative study of leading machine learning techniques and two new algorithms," UC Berkeley manuscript, 2016.

[8] S. Arora, E. Hazan, and S. Kale, "A fast random sampling algorithm for sparsifying matrices," in *Proc. 9th Int. Workshop Approx. Randomization Comb. Optim. Algorithms Tech.*, 2006, pp. 272–279.

[9] D. A. Spielman and S.-H. Teng, "Spectral sparsification of graphs," *SIAM J. Comput.*, vol. 40, pp. 981–1025, 2011.

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

[11] W. Wang, J. Yang, and R. Muntz, "STING: A statistical information grid approach to spatial data mining," in *Proc. 23rd Int. Conf. Very Large Data Bases*, vol. 97, 1997, pp. 186–195.

[12] E. Schikuta, "Grid-clustering: An efficient hierarchical clustering method for very large data sets," in *Proc. 13th Int. Conf. Pattern Recognit.*, vol. 2, 1996, pp. 101–105.

[13] G. Sheikholeslami, S. Chatterjee, and A. Zhang, "Wavecluster: A multi-resolution clustering approach for very large spatial databases," in *Proc. 24th, Int. Conf. Very Large Data Bases*, vol. 98, 1998, pp. 428–439.

[14] E. Fix and J. L. Hodges Jr., "Discriminatory analysis, nonparametric discrimination, consistency properties," Randolph Field., Austin, Texas, Project 21-49-004, Report No. 4, 1951.

[15] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, pp. 273–297, 1995.

[16] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Tran. Intell. Syst. Technol.*, vol. 2, pp. 27:1–27:27, 2011.

[17] J. B. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, Aug. 2000.

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

[19] D. S. Hochbaum, "A polynomial time algorithm for Rayeleigh ratio on discrete variables: Replacing spectral techniques for expander ratio, normalized cut and Cheeger constant," *Oper. Res.*, vol. 61, pp. 184–198, 2013.

[20] U. V. Luxburg, "A tutorial on spectral clustering," *Stat. Comput.*, vol. 17, pp. 395–416, 2007.

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

[22] D. S. Hochbaum, "The pseudoflow algorithm: A new algorithm for the maximum-flow problem," *Oper. Res.*, vol. 56, pp. 992–1009, 2008.

[23] A. McCallum, K. Nigam, and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," in *Proc. 6th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2000, pp. 169–178.

[24] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. 30th Annu. ACM Symp. Theory Comput.*, 1998, pp. 604–613.

[25] M. Norouzi, D. J. Fleet, and R. Salakhutdinov, "Hamming distance metric learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1061–1069.

[26] 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. Comput.*, vol. 36, pp. 158–183, 2006.

[27] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Benigo, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Proc. 28th Int. Conf. Mach. Learn.*, 2011, pp. 833–840.

[28] J. Gao, Q. Shi, and T. S. Caetano, "Dimensionality reduction via compressive sensing," *Pattern Recognit. Lett.*, vol. 33, pp. 1163–1170, 2012.

[29] Y. Saeys, I. Iñaki, and P. Larrañaga, "A review of feature selection techniques in bioinformatics," *Bioinformatics*, vol. 23, pp. 2507–2517, 2007.

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

[31] W. B. Johnson and J. Lindenstrauss, "Extensions of Lipschitz mappings into a Hilbert space," *Contemp. Math.*, vol. 26, pp. 189–206, 1984.

[32] P. Frankl and H. Maehara, "The Johnson-Lindenstrauss lemma and the sphericity of some graphs," *J. Comb. Theory Series B*, vol. 44, pp. 355–362, 1988.

[33] D. Fradkin and D. Madigan, "Experiments with random projections for machine learning," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2003, pp. 517–522.

[34] A. Asuncion and D. J. Newman, "UCI Machine Learning Repository," 2007. [Online]. Available: http://www.ics.uci.edu/~mlearn/ MLRepository.html

[35] Y. Zhao and J. Song, "Gdilc: A grid-based density-isoline clustering algorithm," in *Proc. Int. Conf. Info-tech Info-net*, vol. 3, 2001, pp. 140–145.

[36] ACM SIGKDD, KDD cup website, *Association for Computing Machinery*, [Online]. Available: http://www.kdd.org/kdd-cup.

[37] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 161–168.

[38] S. Moro, R. Laureano, and P. Cortez, "Using data mining for bank direct marketing: An application of the CRISP-DM methodology," in *Proc. Eur. Simul. Modelling Conf.*, 2011, pp. 117–121.

[39] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann, 2005.

[40] P. S. Efraimidis and P. G. Spirakis, "Weighted random sampling with a reservoir," *Inf. Process. Lett.*, vol. 97, pp. 181–185, 2006.

[41] P. Baumann, D. S. Hochbaum, and Q. Spaen, "Sparse-reduced computation: Enabling mining of massively-large data sets," in *Proc. 5th Int. Conf. Pattern Recognit. Appl. Methods*, 2016, pp. 224–231.

Dorit S. Hochbaum is a full professor and chancellor chair at UC Berkeley's Department of Industrial Engineering and Operations Research. Her research interests are in the areas design and analysis of computer algorithms, approximation algorithms, and discrete and continuous optimization. Her recent work focuses on efficient techniques related to network flows with novel applications in ranking, pattern recognition, data mining, and image segmentation problems. She is the author of more than 150 papers that appeared in the operations research, management science and theoretical computer science literature. She was awarded a honorary doctorate of sciences by the University of Copenhagen recognizing Hochbaum's ground-breaking achievements and leadership in optimization in general and in the field of approximation algorithms for intractable problems in particular. She was awarded the title of INFORMS fellow in fall 2005 for the extent of her contributions to operations research, management science and design of algorithms. She is the winner of the 2011 INFORMS Computing Society prize for best paper dealing with the operations research/computer science interface. She was named fellow of the Society of Industrial and Applied Mathematics in 2014.

Philipp Baumann received the BSc, MSc, and PhD degrees in business administration from the University of Bern, Switzerland, in 2007, 2009, and 2013, respectively. He did his postdoc at the Department of Industrial Engineering and Operations Research, University of California, Berkeley. Currently, he is an assistant professor at the Department of Business Administration, University of Bern (Switzerland). His research interests include optimization in finance, project management and project scheduling, production planning and control, data mining, and network-based optimization. He has published in the *European Journal of Operational Research*, the *Flexible Services*, and the *Manufacturing Journal*, the *International Journal of Production Research*, and the *Mathematical Methods of Operations Research*.