European Journal of Operational Research 272 (2019) 1041–1057

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Innovative Applications of O.R.

A comparative study of the leading machine learning techniques and two new optimization algorithms

P. Baumann[a,*], D. S. Hochbaum[b], Y. T. Yang[c,1]

[a] Department of Business Administration, University of Bern, Schützenmattstrasse 14, Bern 3012, Switzerland
[b] Industrial Engineering and Operations Research Department, University of California, Berkeley, CA 94720, USA
[c] Amazon.com, 101 Main Street, Cambridge, MA 02142, USA

ARTICLE INFO

Article history:
Received 28 August 2017
Accepted 4 July 2018
Available online 24 July 2018

Keywords:
Data mining
Supervised machine learning
Binary classification
Comparative study
Supervised normalized cut

ABSTRACT

We present here a computational study comparing the performance of leading machine learning techniques to that of recently developed graph-based combinatorial optimization algorithms (SNC and KSNC). The surprising result of this study is that SNC and KSNC consistently show the best or close to best performance in terms of their F_1-scores, accuracy, and recall. Furthermore, the performance of SNC and KSNC is considerably more robust than that of the other algorithms; the others may perform well on average but tend to vary greatly across data sets. This demonstrates that combinatorial optimization techniques can be competitive as compared to state-of-the-art machine learning techniques. The code developed for SNC and KSNC is publicly available.

1. Introduction

Data mining, combinatorial optimization and practical efficiency appear to be incompatible. This comparative study contests this presumption: it demonstrates that new combinatorial graph-based optimization algorithms for classification have superior performance and robustness when compared to well-established machine learning techniques.

Binary classification is a fundamental machine learning task; it is defined as correctly assigning new objects to one of two groups based on a set of training objects. Driven by the practical importance of binary classification, numerous machine learning techniques have been developed and refined over the last three decades and their relative performance has been evaluated in several empirical studies (cf., e.g., Caruana, Karampatziakis, & Yessenalina, 2008; Caruana & Niculescu-Mizil, 2006; King, Feng, & Sutherland, 1995; Lim, Loh, & Shih, 2000). Among the best performing and most popular techniques are artificial neural networks, decision trees, ensemble methods, logistic regression, and support vector machines. Empirical performance evaluations are of great interest to practitioners and researchers, as they point out the strengths and weaknesses of the available techniques and thereby channel research efforts into promising directions.

The use of graph-based optimization models in data mining is relatively new. Graph-based optimization models represent the data set as a graph and employ different criteria to segment the graph. The normalized cut is a bipartitioning criterion, commonplace in image segmentation (cf. Shi & Malik, 2000), that captures desirable cluster properties. As such it is used to measure the quality of clustering heuristics, for example, in bioinformatics (cf. Kawaji, Takenaka, & Matsuda, 2004). Solving the normalized cut problem, however, is NP-hard and thus impractical. Hochbaum (2010) introduced the HNC (Hochbaum's Normalized Cut) problem, which is a variant of the NP-hard normalized cut problem that can be solved in polynomial time. HNC-related algorithms are transduction algorithms, as they consider all objects, not just training objects, while performing the classification task. Transduction algorithms tend to deliver robust results with fewer training objects because they also consider relationships among new objects.

The HNC criterion has been successfully applied in specific contexts. These include image segmentation (cf. Hochbaum, Lyu, & Bertelli, 2013), evaluating drug effectiveness (cf. Hochbaum, Hsu, & Yang, 2012), video tracking (cf. Fishbain, Hochbaum, & Yang, 2013), enhancing the capabilities of low-resolution nuclear detectors (cf. Yang, Fishbain, Hochbaum, Norman, & Swanberg, 2013), and recently also for identifying and tracking neurons in calcium imaging movies (cf. Spaen, Hochbaum, & Asín-Achá, 2017). In the work of Yang et al. (2013) on enhancing the capabilities of nuclear detectors, HNC was compared to several well-known data

* Corresponding author.
E-mail addresses: philipp.baumann@pqm.unibe.ch (P. Baumann), hochbaum@ieor.berkeley.edu (D.S. Hochbaum), yxy128@berkeley.edu (Y.T. Yang).

[1] This work was done prior to the Amazon involvement of the author. This paper does not reflect the views of Amazon.

https://doi.org/10.1016/j.ejor.2018.07.009
0377-2217/

mining algorithms. The major conclusions of this study were that HNC and Support Vector Machines (SVM) are by far the most successful among the machine learning methods in the comparison set, in terms of accuracy, with HNC being slightly more accurate than SVM and significantly faster. The drug evaluation study in Hochbaum et al. (2012) used a collection of machine learning techniques containing methods previously used for drug evaluation, with similar conclusions. For neuron identification in calcium imaging movies, the HNC criterion has been a top performer in the Neurofinder benchmark and has provided superior performance as compared to matrix factorization techniques (see CodeNeuro.org, as of March 9, 2017). However, there has been no systematic comparison of the performance of HNC-related algorithms to that of state-of-the art machine learning techniques.

Our goal here is to investigate whether HNC-related techniques, namely the supervised normalized cut (SNC) and the *K*-supervised normalized cut (KSNC), could be competitive for the task of binary classification compared to twelve established machine learning techniques. The comparison uses F_1-scores, precision, recall, accuracy, and running times as performance measures. The established methods are artificial neural networks (ANN), classification trees (CART), the Naive Bayes classifier (CNB), three different ensemble methods (EADA, EBAG, EGAB), lasso regression (LASSO), linear regression (LIN), logistic regression (LOG), support vector machines with different types of kernels (SVM), support vector machines with only radial basis function kernels (SVMR), and the *K*-nearest neighbor method (KNN). These techniques are considered to be state-of-the-art in binary classification due to their good performance in previous studies (cf., e.g., Caruana & Niculescu-Mizil, 2006; Cooper et al., 1997; King et al., 1995). We test here the techniques on twenty data sets that differ with respect to domain, size, and skewness.

A comparative study must be reproducible (cf. e.g., Pedersen, 2008; Sonnenburg et al., 2007). We therefore place an emphasis on selecting the implementations of the machine learning techniques. In contrast to previous comparative studies, we use here standard and accessible MATLAB implementations of the techniques. The use of accessible versions significantly enhances the reproducibility of the results. Standard versions have only a few tuning parameters with basic ranges for the values. We employ random search with a fixed time limit (cf. Bergstra & Bengio, 2012) to determine the tuning parameter values for the techniques. Bergstra and Bengio (2012) demonstrated on several data sets that random search is more efficient than grid search or manual search. Furthermore, random search requires minimal user interaction that contributes to the reproducibility of the results. To avoid overfitting, we used *k*-fold cross-validation. None of the tested techniques showed signs of systematic overfitting. To investigate the consistency of our findings, we tested several alternative experimental designs that differ with respect to the parameter tuning. We tested designs with different time limits, designs where each technique is applied with the same number of tuning parameter combinations, and designs where grid search was used instead of random search. The results obtained with the different experimental designs were all consistent. Therefore, and due to the lack of space, we report here only the results of the baseline design.

The surprising result of this study is that SNC and KSNC consistently have the best or close to best performance in terms of F_1-scores. This was unexpected because SNC and KSNC were not originally designed as machine learning techniques. Furthermore, the performance of SNC and KSNC is considerably more robust than that of the other techniques; the others perform well on average but tend to vary greatly across data sets.

Another important outcome of our study is that machine learning techniques that are similarity-based (KNN, KSNC, SNC, SVMR) tend to outperform non-similarity-based techniques with respect to all performance measures. Pairwise similarities considerably enhance the quality of prediction, pattern recognition and data mining. This has been noted in the past: by Dembczyński, Kotowski, and Słowiński (2009) for machine learning purposes, by Ryu, Chandrasekaran, and Jacob (2004) for improved medical diagnosis, and by Zhu, Ghahramani, and Lafferty (2003) in semi-supervised learning. In terms of running times, the regression-based methods LASSO, LIN, and LOG, decision trees (CART), and the Naive Bayes classifier (CNB) are among the fastest, but their F_1-scores and accuracy are generally poor. Among the best-performing algorithms KNN requires the least running time, followed by SNC and KSNC. SVMR and EBAG perform well, but they are both computationally expensive by comparison.

For massively large data sets, the scaling of similarity-based algorithms could pose a challenge due to the quadratic rate of growth in the number of similarities as a function of the number of objects in the data set. To address this, we have recently introduced the technique of *sparse computation* (cf. Baumann, Hochbaum, & Spaen, 2016; Baumann, Hochbaum, & Spaen, 2017; Hochbaum & Baumann, 2014; Hochbaum & Baumann, 2016), which provides practical efficiency while retaining accuracy, even for very large-scale data sets. With *sparse computation*, it is possible to apply the SNC, KSNC, SVMR, and KNN techniques that performed best or close to best in this study to large-scale data sets. In fact, in Hochbaum and Baumann (2016) SNC, KSNC, and KNN were successfully applied to data sets comprising of up to 8.5 million objects.

The paper is structured as follows. Section 2 reviews previous comparative studies of binary classification algorithms. Section 3 describes the new machine learning techniques that are based on the HNC optimization problem. Section 4 provides brief descriptions and implementation details of the established machine learning techniques tested in this study. Section 5 introduces the data sets used in this study and Section 6 describes the experimental setup. Section 7 reports the computational results and Section 8 concludes this paper with some final comments.

2. Previous comparative studies

Several studies focus on examining the performance of only two or three types of machine learning techniques. For example, Bauer and Kohavi (1999) present an extensive comparison of different ensemble algorithms, including bagging and AdaBoost; Perlich, Provost, and Simonoff (2003) compare decision trees and logistic regression; Bhattacharyya, Jha, Tharakunnel, and Westland (2011) compare support vector machines, random forests and logistic regression for detecting credit card fraud; and De Caigny, Coussement, and De Bock (2018) compare decision trees, logistic regression, and ensemble algorithms to a new hybrid classification algorithm that is based on logistic regression and decision trees. However, these studies provide comparisons only between a limited selection of techniques, making it difficult to draw general conclusions about the relative performance of the tested machine learning techniques.

Other studies evaluate a broad range of machine learning techniques but focus on a specific application area. LeCun et al. (1995) test fourteen algorithms on a handwriting recognition problem. They use accuracy, rejection rate, running time and memory requirement as performance metrics. LeCun et al. (1995) conclude that boosted neural networks and support vector machines perform best. Cooper et al. (1997) test ten machine learning techniques in terms of their ability to predict mortality in patients with pneumonia. The lowest error rates were obtained by neural networks, hierarchical mixtures of experts and logistic regression. Ahmed, Atiya, Gayar, and El-Shishiny (2010) evaluate eight machine learning models for time series forecasting. The models

are applied to the well-known M3 monthly time series database, and the best results are obtained by neural networks and Gaussian processes. Although these application-specific studies provide useful information on the suitability of machine learning techniques for well-defined tasks, the recommended techniques may not perform as well on general-context data sets.

Only a few studies include several machine learning techniques and a relatively large set of classification problems. A comprehensive study called STATLOG was conducted in the early nineties by King et al. (1995). They compare sixteen machine learning techniques that are variations of decision trees, discriminant and regression algorithms, the *K*-nearest neighbor algorithm, Bayesian classification algorithms and neural networks on twelve real-world problems. The comparison is organized as a competition between research groups from academia and industry, each with an interest in seeing their own algorithm perform best. The study leaves the choice of tuning parameters and their ranges to the research groups, thus making it difficult to compare the performances. Performance is measured in terms of accuracy and running time. The main conclusions of the study are that there is no dominant machine learning technique and that the performance of the algorithms depends critically on the data sets.

Lim et al. (2000) extend the results of the STATLOG Project by testing spline-based statistical algorithms and additional variations of decision trees and by an in-depth analysis of the training and testing times of the algorithms. In total, Lim et al. (2000) compare twenty-two decision trees, nine statistical algorithms and two neural network algorithms on sixteen data sets. Their results show that there is little variability among the algorithms in terms of accuracy but considerable differences with respect to training time. The study of Lim et al. (2000) identifies those algorithms that take the least amount of training and testing time.

Caruana and Niculescu-Mizil (2006) evaluate ten machine learning techniques on eleven classification problems with respect to eight performance metrics. One distinctive feature of their study is that the size of the training sets is fixed at 5000. The machine learning techniques are support vector machines, neural nets, logistic regression, Naive Bayes, memory-based learning, random forests, decision trees, bagged trees, boosted trees (including boosted stumps as a special case). Different variations of these machine learning techniques are tested and the space of tuning parameters is explored thoroughly. The performance of each technique is measured before and after calibrating its predictions with Platt Scaling and Isotonic Regression. Without calibration, bagged trees, random forests and neural nets perform best across all eight metrics and eleven classification problems. With calibration, boosted trees perform best followed by neural nets, SVMs, random forests and bagged trees. Naive Bayes, logistic regression, decision trees and KNN, in general, perform rather poorly. An interesting result is that the ranking of the machine learning techniques is generally consistent for the different performance measures. This means that performance measures are highly correlated.

Caruana et al. (2008) test the same set of algorithms as Caruana and Niculescu-Mizil (2006) on several high-dimensional data sets. Caruana et al. (2008) also change the setup by using the natural training sets that were given for each classification problem. The size of these training sets is generally greater than 5000 objects. The authors focus on the three performance metrics - accuracy, area under the ROC curve and squared error - which were all among the performance measures used in the study of Caruana and Niculescu-Mizil (2006). The findings are that the performance of machine learning techniques for high-dimensional data sets is consistent with the performance reported in Caruana and Niculescu-Mizil (2006) for low-dimensional data sets. The technique of random forests performs consistently well across data sets of different dimensionality.

The results of this collection of studies are often contradictory. The contradictions could be caused by the use of different data sets, different implementations of machine learning techniques, and different tuning strategies. Our study differs from previous studies mainly in two ways. First, we use basic and widely-used MATLAB versions of machine learning techniques and employ random search to optimize tuning parameter values. This guarantees reproducibility and gives a basic understanding of the general potential of different types of machine learning techniques. Second, we include for the first time two machine learning techniques that are variants of graph-based optimization models. For the two new techniques, we used the MATLAB interface provided on the website of the authors; see Chandran and Hochbaum (2012, last updated on Aug, 2012.).

3. New graph-based machine learning techniques

The two new graph-based machine learning techniques, SNC and KSNC, are variants of the HNC problem. We describe the HNC problem first in Section 3.1. Then, we introduce SNC and KSNC in Sections 3.2 and 3.3, respectively. The proposed techniques are new in the sense that they represent the machine learning problem as a graph partitioning problem that can be solved to optimality in polynomial time. Existing graph-based machine learning methods employ heuristics (e.g., Cupertino, Zhao, & Carneiro, 2015) or use the graph representation to extract local information about the underlying data distribution (e.g., Bertini, Zhao, Motta, & de Andrade Lopes, 2011).

3.1. The HNC problem

3.1.1. HNC and a related clustering criterion

An attractive model for clustering within a data set has the goal of minimizing the ratio of two criteria. One criterion is to maximize the total similarity of objects within the cluster (the intra-similarity). The second criterion is to minimize the similarity between the cluster and its complement (the inter-similarity). The HNC problem is a ratio problem that combines these two criteria.

The HNC problem and related problems such as the normalized cut are defined on graphs. 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 the weights of the 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 $C(A, B) = \sum_{i \in A, j \in B, [i,j] \in E} w_{ij}$. 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 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 a feature vector. The edge weights w_{ij} quantify the similarity between the respective feature vectors associated with nodes i and j. Higher similarity is associated with higher weights.

With this notation, the formulation of the HNC problem is,

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

In general, there would be a seed object s that belongs to the desired cluster, and a seed object t that is not in the cluster. The problem is to find a non-empty set S^* strictly contained in V, that minimizes the ratio of the similarity between the set and its complement, inter-similarity, divided by the total similarity within the set S^*, intra-similarity.

The HNC problem was used in the context of image segmentation, where it was mistakenly confused with the NP-hard

problem of *normalized cut* (cf. Shi & Malik, 2000) and referred to by the same name in Sharon, Galun, Sharon, Basri, and Brandt (2006). The problem of normalized cut (NC) is formulated as,

$$\mathrm{NC}(S^*) = \min_{\emptyset \subset S \subset V} \frac{C(S,\bar{S})}{\sum_{i\in S} d_i} + \frac{C(S,\bar{S})}{\sum_{i\in \bar{S}} d_i}.$$

Although the HNC problem and the normalized cut problem appear similar, the normalized cut problem is intractable, whereas HNC was shown in Hochbaum (2010) to be polynomial time solvable, as a *monotone integer program* (cf. Hochbaum, 2002) with a minimum cut procedure.

The spectral method was proposed in Shi and Malik (2000) as a heuristic for the normalized cut problem. Indeed, the use of the spectral method has been dominant in image segmentation and was even used in data mining applications (cf., e.g., Jia, Ding, Xu, & Nie, 2014). An experimental study of image segmentation instances, in Hochbaum et al. (2013), compared the performance of an HNC-based classification model with the spectral method and demonstrated that even as a heuristic for the normalized cut problem, the HNC-based classification model provides superior solutions. A detailed theoretical discussion of the HNC problem and the spectral method is given in Hochbaum (2013).

The HNC problem is formulated next as an integer program, which is monotone, thus leading to polynomial time algorithms for solving it.

3.1.2. Integer programming formulation

We provide here a formulation for a slight generalization of the HNC problem, $\min_{\emptyset\subset S\subset V} \frac{C_1(S,\bar{S})}{C_2(S,\bar{S})}$, where different sets of similarity weights can be used for the numerator, w_{ij}, and the denominator, w'_{ij}. Let the binary variable x_i be 1 if $i \in S$ and 0 otherwise. Hence, the set of nodes $\{i \in V | x_i = 1\}$ form the cluster S, and the set of nodes $\{j \in V | x_j = 0\}$ form the complement $\bar{S}$. We write the edges of the graph as $[i, j] \in E$ such that $i < j$. For each edge $[i, j] \in E$, we introduce one binary variable z_{ij}, which is 1 if one of the nodes i and j is in S and the other is in $\bar{S}$ and 0 otherwise. The binary variable y_{ij}, which is also introduced for each edge $[i, j] \in E$, is 1 if both i and j are in S and 0 otherwise. With these variables, the formulation of the HNC problem is:

$$\text{(ratio-HNC)} \begin{cases} \text{Min} & \dfrac{\sum_{[i,j]\in E} w_{ij} z_{ij}}{\sum_{[i,j]\in E} w'_{ij} y_{ij}} & & (1) \\ \text{s.t.} & x_i - x_j \le z_{ij} & ([i,j] \in E) & (2) \\ & x_j - x_i \le z_{ij} & ([i,j] \in E) & (3) \\ & y_{ij} \le x_i & ([i,j] \in E) & (4) \\ & y_{ij} \le x_j & ([i,j] \in E) & (5) \\ & x_s = 1 & & (6) \\ & x_t = 0 & & (7) \\ & x_i \in \{0,1\} & (i \in V) & (8) \\ & z_{ij} \in \{0,1\} & ([i,j] \in E) & (9) \\ & y_{ij} \in \{0,1\} & ([i,j] \in E) & (10) \end{cases}$$

The objective function drives the values of z_{ij} to be as small as possible. Constraints (2) and (3) ensure that z_{ij} is set to 1 if one of the nodes i and j is in S and the other is in $\bar{S}$. Consequently, the weight between i and j is added once to the numerator of the objective function. Constraints (4) and (5) guarantee that $y_{ij} = 0$ unless both i and j are in S. In case both i and j are in S, then y_{ij} is set to 1 because the objective function drives the values of y_{ij} to be as large as possible. Consequently, the weight between i and j is added once to the denominator of the objective function. Constraint (6) ensures that there is at least one node in S and constraint (7) ensures that there is at least one node in the $\bar{S}$. Constraints (8)–(10) define the domains of the decision variables.

The optimization problem (ratio-HNC) can be formulated as a *monotone integer program* by introducing a second binary decision variable z_{ji} for each edge $[i, j] \in E$ which replaces the variable z_{ij} in constraint (3). Constraints (2) and (3) together with the objective function will guarantee that always one of the variables z_{ij} and z_{ji} will be zero. A monotone integer program has all constraints containing up to 3 variables and each constraint is of the form $a_{ij}x_i - b_{ij}x_j \le c_{ij}z_{ij}$. That is, two of the variables, x_i and x_j, appear with opposite sign coefficients, and the third variable (z_{ij} here) appears in at most one constraint. Also the variables that appear "third" must have their objective function coefficients be non-negative for minimization functions. Any formulation of an integer program that is monotone was shown in Hochbaum (2002) to be solved as a minimum cut problem on an associated graph where there is a node for each variable's integer value. It was further proved, in Hochbaum (2010) and Hochbaum (2013), that a ratio problem on monotone constraints can be solved as a *parametric cut* problem in complexity that is the same as that of a single minimum cut procedure.

To illustrate the algorithmic technique used we provide a sketch of the algorithm of Hochbaum (2010) solving (ratio-HNC). To solve the ratio problem one can find the smallest value of λ (λ^*) for which the linearized version of HNC, HNC(λ), has an optimal objective function value that is non-positive: $\min_{\emptyset\subset S\subset V} C(S,\bar{S}) - \lambda C(S,S) \le 0$. Let $S(\lambda)$ be the optimal solution for this HNC(λ). Instead of solving for each value of λ, using e.g., binary search, the parametric cut procedure finds the optimal solution for all values of λ in the complexity of a single minimum cut Hochbaum (2002, 2010, 2013) with the parametric pseudoflow algorithm, HPF (cf. Hochbaum, 2008). This parametric cut procedure produces (up to the number of nodes n in the graph) breakpoints for the value of λ where at each breakpoint the optimal solution set $S(\lambda)$ changes. One can then find the value λ^* which is the smallest for which the objective function value of HNC(λ^*) is still non-positive. The corresponding solution $S(\lambda^*)$ is the optimal solution for (ratio-HNC).

Concerning ratio problems in general, it is often the case that the optimal weighting λ^* of the ratio optimal solution is not necessarily the best in terms of the quality of the resulting cluster, and a source set associated with a non-optimal value of λ is a better cluster. After all, any arbitrary scalar multiplication of the numerator, changes the value of the optimal parameter and potentially the respective bi-partition solution. It is therefore more effective to consider a "good" weighting of the two criteria instead of solving for the ratio problem, and solve the problem HNC(λ) for a desirable value of λ. Here, this weighting value of λ is one of the tuning parameters to be determined when implementing HNC as a classification method. In fact, unless stated otherwise, we refer here to the linearized version of HNC as HNC.

In addition, it was shown in Hochbaum (2010) that (ratio-HNC) is equivalent to minimizing the first term of NC:

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

The formulation for this version is monotone as well and can be solved likewise with a parametric cut procedure in the complexity of a single minimum cut. This same algorithm, with minor adaptation, also solves more general problems than HNC: Any non-negative node weights q_i can be used to replace the weighted degrees weights of the nodes:

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

Indeed, we use here node weights different from d_i to derive the KSNC algorithm. An extensive discussion of these and other variants of HNC is provided in Hochbaum (2010, 2013).

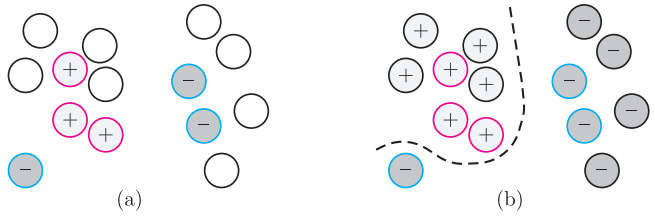

Fig. 1. (a) The input consists of labeled nodes with positive (+) or negative (−) labels and unlabeled nodes; (b) The solution consists of two sets that are separated by a cut. All nodes with positive labels form set S, and all nodes with negative labels form set $\bar{S}$, where the similarity within S and the dissimilarity between the two sets are high.

3.2. Supervised normalized cut (SNC)

The set-up of HNC does not require any labeled nodes except one node s that belongs to the cluster S and one node t that belongs to the complement. The selection of s and t is to guarantee that the solution is non-empty and strictly contained in V. When only two nodes are specified as *source* node and *sink* node, then we refer to this variant of HNC as the *unsupervised* variant of HNC. However, HNC can also be implemented as a *supervised* classification method. In the supervised case, the input graph contains labeled nodes (training data) that refer to objects for which the class label (either positive or negative) is known and unlabeled nodes that refer to objects for which the class label is unknown. By assigning all labeled nodes with a positive label to set S, as source nodes merged with s, and all labeled nodes with a negative label to set $\bar{S}$, as sink nodes merged with t, the HNC model can be used in a supervised manner (cf. Fig. 1). The goal is then to assign all unlabeled nodes either to set S or set $\bar{S}$. Due to the pre-assignment of labeled nodes, the graph's size is reduced since all labeled nodes are merged with s or t, so the "supervised" graph contains only unlabeled nodes. This size reduction implies a corresponding reduction in the running time of the algorithm. We refer to the use of HNC in a supervised manner as *supervised normalized cut* (SNC).

In this study, we choose Gaussian similarity weights that are a monotone function of Euclidean distances. The Gaussian similarity of two objects i and j with respect to the feature vectors $v^{(i)}$ and $v^{(j)}$ is:

$$w_{ij} = \exp\left(-\frac{\|v^{(i)} - v^{(j)}\|^2}{2\epsilon^2}\right),$$

where $\|v^{(i)} - v^{(j)}\|$ denotes the Euclidean distance between i and j and parameter ϵ represents a scaling factor. The Gaussian similarity function is commonly used in image segmentation and spectral clustering (cf. Von Luxburg, 2007). When implementing SNC there are two tuning parameters: the relative weighting parameter of the two objectives, λ, and the scaling factor of the exponential weights, ϵ. Table 1 lists all tuning parameters of SNC and specifies a range of values for each parameter. The minimum cut problems were solved with the MATLAB implementation of the HPF pseudoflow algorithm version 3.23 of Chandran and Hochbaum (2012, last updated on Aug, 2012.) that was presented in Hochbaum (2008).

3.3. K-Supervised normalized cut (KSNC)

KSNC is a variant of SNC in which we seek to optimize

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

where 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 to i, in terms of similarity, have labels 0, 1, and 1, then q_i is 2/3. In contrast to the weights d_i, which capture the pairwise similarities between any pairs, whether the nodes are labeled or not, the weights q_i as defined above only capture the effect of the labeled nodes on the unlabeled nodes. KSNC is therefore a sort of hybrid between SNC and KNN. Here again, we consider the linearized version, which we refer to as KSNC:

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

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 1 specifies for each of these parameters the range of values that we tested here.

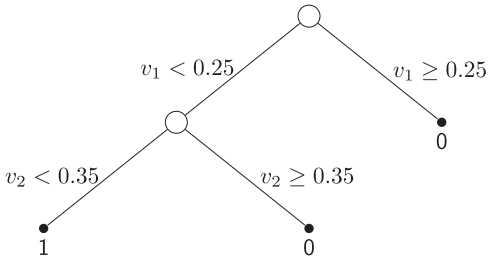

Fig. 2. Example of a classification tree.

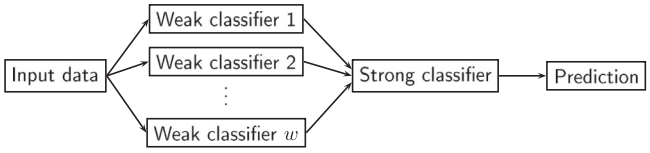

Fig. 3. Structure of an ensemble method. We used the number of weak learners w as a tuning parameter.

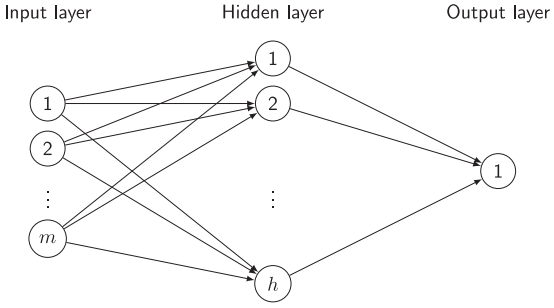

Fig. 4. Architecture of artificial neural networks tested in this study: feedforward network with one hidden layer. The number of nodes in the input layer corresponds to the number of features m in the data set.

4. Commonly used machine learning techniques

In this section, we provide a brief description of the established classification techniques tested in this study. These techniques can be divided into four groups: decision tree-based techniques, regression-based techniques, similarity-based techniques, and other techniques. Sections 4.1–4.4 describe the tested techniques in the four groups. For each considered technique there are numerous variations proposed in the literature. For the sake of uniformity and accessibility to codes, we take the basic versions of the algorithms implemented in MATLAB R2017b. A more detailed analysis of some of these techniques can be found in Carrizosa and Morales (2013).

4.1. Decision tree-based machine learning techniques

A decision tree is based on a hierarchical tree-like partition of the input data. It predicts a target variable for a new object based

Table 1
Lower bounds (LB) and upper bounds (UB) for tuning parameter values.

Technique	Tuning parameter name	LB	UB	Type
ANN	Units in hidden layer	1	200	Integer
CART	Minimum leaf size	1	50	Integer
	Minimum parent size	2	25	Integer
CNB	Threshold	0.25	0.75	Real
EADA	Number of decision trees	2	1,000	Integer
EBAG	Number of decision trees	2	1,000	Integer
EGAB	Number of decision trees	2	1,000	Integer
LASSO	Regularization parameter $\lambda_L = 2^x$ with x in [LB, UB]	−10.00	3.00	Real
LIN	Threshold	−0.50	0.50	Real
LOG	Threshold	0.25	0.75	Real
SVM	Polynomial (1) or radial basis function kernel (2)	1	2	Integer
	Degree of polynomial kernel	1	5	Integer
	Derivative param. of RBF kernel (2^x with x in [LB, UB])	−20.00	20.00	Real
	Cost (2^x with x in [LB, UB])	−20.00	20.00	Real
SVMR	Radial basis function kernel (2)	2	2	Integer
	Derivative param. of RBF kernel (2^x with x in [LB, UB])	−20.00	20.00	Real
	Cost (2^x with x in [LB, UB])	−20.00	20.00	Real
KNN	Parameter K	1	80	Integer
KSNC	Parameter K	1	3	Integer
	Weighting parameter $\lambda = 2^x$ with x in [LB, UB]	−15.00	5.00	Real
	Scaling parameter $\epsilon = 2^x$ with x in [LB, UB]	−5.00	10.00	Real
SNC	Weighting parameter $\lambda = 2^x$ with x in [LB, UB]	−15.00	0.00	Real
	Scaling parameter $\epsilon = 2^x$ with x in [LB, UB]	−5.00	10.00	Real

Table 2
Characteristics of data sets.

Abbr	Downloaded from	# Objects	# Attributes	# Positives	# Negatives	$\frac{\#\text{Positives}}{\#\text{Negatives}}$
IRS	LIBSVM	150	4	50	100	0.50
WIN	LIBSVM	178	13	59	119	0.50
PAR	UCI	195	22	147	48	3.06
SON	UCI	208	60	111	97	1.14
GLA	LIBSVM	214	9	70	144	0.49
HEA	LIBSVM	270	13	120	150	0.80
HAB	UCI	306	3	81	225	0.36
VER	UCI	310	6	210	100	2.10
ION	UCI	351	34	225	126	1.79
DIA	UCI	392	8	130	262	0.50
BCW	UCI	683	10	239	444	0.54
AUS	LIBSVM	690	14	307	383	0.80
BLD	UCI	748	4	178	570	0.31
FOU	LIBSVM	862	2	307	555	0.55
TIC	UCI	958	27	626	332	1.89
GER	UCI	1,000	24	300	700	0.43
CAR	UCI	2,126	21	1,655	471	3.51
SPL	LIBSVM	3,175	60	1,648	1,527	1.08
LE1	UCI	20,000	16	753	19,247	0.04
LE2	UCI	20,000	16	9,940	10,060	0.99

on the values of its features. Decision trees are referred to as regression trees when the target variable is continuous and as classification trees when the target variable is discrete. A classification tree consists of internal nodes and leaf nodes. Each internal node represents a test on a feature. The arcs leaving an internal node are labeled with a specific range of possible values. Each leaf node represents a class label. Given a new object, a series of tests along the internal nodes, starting from the root node, will determine a leaf node that predicts the class label. Fig. 2 shows an example of a classification tree. Numerous methods for constructing classification trees have been proposed (cf. Murthy, 1998). In this study, we tested classification and regression trees (CART) and three ensemble methods (EADA, EBAG, EGAB), each of which combines multiple classification trees into one machine learning technique.

4.1.1. Classification trees (CART)

The term classification and regression trees (CART) refers to methods introduced by Breiman, Friedman, Stone, and Olshen (1984) for constructing classification and regression trees. The method of Breiman et al. (1984) for classification trees employs the Gini impurity index for finding the features that best split the training objects. The same splitting criterion is used by default in the MATLAB function `fitctree` from the statistics and machine learning toolbox. The minimum leaf size and the minimum parent size are used as tuning parameters (cf. Table 1). The minimum leaf size specifies a lower bound on the number of objects per leaf node and the minimum parent size specifies a lower bound on the number of objects per non-leaf node. If both values are provided and the minimum parent size is smaller than twice the minimum leaf size, then the minimum parent size is set to twice the minimum leaf size.

```
% Create classification tree
cart = fitctree(trainingData,trainingClassLabels,
'minLeaf',minLeaf,'minParent',minParent);
% Use classification tree
class = predict(cart,testingData);
```

4.1.2. Ensemble of classification trees (EADA, EBAG, EGAB)

Ensemble methods combine so-called weak learners into one strong learner to obtain better predictive performance (cf. Fig. 3). We test here three different ensemble methods, namely adaptive boosting of Freund and Schapire (1997), bagging of Breiman (1996), and gentle adaptive boosting of Friedman, Hastie, Tibshirani et al. (2000). In our experimental analysis we use the abbreviations EADA for adaptive boosting, EBAG for bagging, and EGAB for gentle adaptive boosting. For all three ensemble methods we used classification trees as weak learners and treated the number of weak learners as a tuning parameter (cf. Table 1). The MATLAB function used is `fitensemble` from the statistics and machine learning toolbox. The argument `method` is *AdaBoostM1* for EADA, *Bag* for EBAG, and *GentleBoost* for EGAB.

```
% Create ensemble
ensemble = fitensemble(trainingData,training
ClassLabels,method,noOfDecisionTrees,'tree', ...
                'Type','classification');
% Use ensemble
class = predict(ensemble,testingData);
```

4.2. Regression-based machine learning techniques

A binary classification problem can be treated as a regression problem in which the dependent variable assumes values 1 or −1. The resulting regression function can be interpreted as a membership indicator. The higher the predicted value for a given object, the more likely it is that the object belongs to the positive class. By using a threshold above which the object is assigned to the positive class, a simple classification rule can be obtained. We tested three regression-based machine learning techniques.

4.2.1. Linear regression (LIN)

We computed linear regression models that contain an intercept and a linear term for each feature of the data set. An object is assigned to the positive class when the prediction is greater than a predefined threshold and to the negative class when the prediction is smaller than or equal to the threshold. The threshold is treated as a tuning parameter (cf. Table 1). The MATLAB function used is `LinearModel.fit` from the statistics and machine learning toolbox; it uses QR decomposition as the fitting method:

```
% Estimate linear regression model
lin = LinearModel.fit(trainingData,trainingClassLabels);
% Use model
prediction = predict(lin,testingData);
class = zeros(size(testingData,1),1);
class(prediction>threshold) = 1;
```

4.2.2. Logistic regression (LOG)

In logistic regression (cf. Bishop, 2006) the dependent variable is assumed to be binary. The value predicted by a logistic regression model always lies between zero and one and can therefore be interpreted as the probability that an object belongs to the positive class given its vector of feature values. To obtain probabilities, a logistic function is used that takes the prediction of a linear regression model and maps it to the interval [0,1]. We estimated logistic regression models that contain an intercept and a linear term for each feature of the data set. An object is assigned to the positive class when its prediction is greater than a predefined threshold and to the negative class when the prediction is smaller than or equal to the threshold. The threshold is treated as a tuning parameter (cf. Table 1). The MATLAB function used is `glmfit` from the statistics and machine learning toolbox. This function uses the IRLS (iteratively reweighted least squares) method to find the maximum likelihood estimates with an iteration limit of 100.

```
% Estimate logistic regression model
log = glmfit(trainingData,trainingClassLabels,'binomial');
% Use model
prediction = glmval(log,testingData,'logit');
class = zeros(size(testingData,1),1);
class(prediction>threshold) = 1;
```

4.2.3. Lasso regression (LASSO)

Regression models tend to overfit when the number of features is relatively large compared to the number of observations. An overfitted model performs well on training objects but poorly on objects from the test set. In general, overfitted models have many relatively large regression coefficients. Regularization methods aim to prevent overfitting by adding an extra term to the average loss function that penalizes large coefficients. In Lasso regression (cf. Tibshirani, 1996), there is an added penalty term

$$\sum_{i=1}^{m} \lambda_L |\beta_i|,$$

where m denotes the number of features and β_i denotes the coefficient associated with the i-th feature. The parameter λ_L is the regularization parameter that controls the trade-off between the average logistic loss and the size of the coefficient vector measured by the ℓ_1-norm. We treat the regularization parameter as a tuning parameter (cf. Table 1). To estimate the regularized logistic regression model, the MATLAB function `lassoglm` from the statistics and machine learning toolbox is used. The models contain an intercept and a linear term for each feature of the data set. An object is assigned to the positive class when its prediction is greater than 0.5 and to the negative class when the prediction is smaller than or equal to 0.5. The MATLAB function `lassoglm` also employs the IRLS method with an iteration limit of 100 for training the model.

```
% Estimate lasso regression model
[lasso,FitInfo] = lassoglm(trainingData,trainingClassLabels, ...
    'binomial','Alpha',1,'Lambda',lambda);
% Use model
prediction = glmval([FitInfo.Intercept(1);lasso],testingData,'logit');
class = zeros(size(testingData,1),1);
class(prediction>0.5) = 1;
```

4.3. Similarity-based machine learning techniques

Similarity-based machine learning techniques use, as part of the input, pairwise similarities between objects. This group includes the two new machine learning techniques introduced in Section 3, the K-nearest neighbor algorithm, and support vector machines with non-linear kernels.

4.3.1. K-nearest neighbor algorithm (KNN)

The K-nearest neighbor algorithm (cf. Fix & Hodges, 1951) uses the training objects themselves to classify new objects. It finds the K training objects most similar to the new object and then assigns to the new object the predominant class among those K neighbors.

To find the K-nearest neighbors, each new object is compared to each training object. We use the MATLAB function `fitcknn` with the default setting and treat K as tuning parameter (cf. Table 1). Per default Euclidean distance is used, and the nearest labeled object is considered to break ties.

```
% Build knn model
knn = fitcknn(trainingData,trainingClassLabels,
'Distance','euclidean','NumNeighbors',param_k);
% Use model
class = predict(knn,testingData);
```

4.3.2. Support vector machines (SVM, SVMR)

Support vector machine models (cf. Cortes & Vapnik, 1995) represent objects as points in space and find the maximum-margin hyperplanes that best separate positive training objects from negative training objects. Objects from the test set are mapped onto 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 onto a high-dimensional space. Various extensions of support vector machines exist that allow to detect important features and interactions among features (cf. Carrizosa, Martín-Barragán, & Morales, 2011; Gaudioso, Gorgone, Labbé, & Rodríguez-Chía, 2017), and enable the application to multiclass problems (cf. Duarte Silva, 2017). We tested here the standard version of support vector machines with two different tuning settings. In the first setting, linear, polynomial, and radial basis function kernels are used for tuning. The algorithm that uses this setting is referred to as SVM. In the second setting, only radial basis function kernels are used for tuning. The algorithm that uses this setting is referred to as SVMR. The tuning parameter ranges are given in Table 1. We used the MATLAB interface of the LIBSVM implementation (version 3.17) for support vector classification (cf. Chang & Lin, 2011). LIBSVM uses the SMO (sequential minimal optimization) algorithm for training support vector machines.

```
% Train support vector machine
svm = svmtrain(trainingClassLabels,trainingData,options);
% Use support vector machine
class = svmpredict(testingClassLabels,testingData,svm,'-q');
```

4.4. Other machine learning techniques

Other widely-used machine learning techniques that we tested in this study include the Naive Bayes classifier (cf. 4.4.1) and artificial neural networks (cf. 4.4.2).

4.4.1. Naive Bayes classifier (CNB)

A Naive Bayes classifier assigns a new object v to the positive class when the probability $P(y = 1|v)$ is greater than a predefined threshold. The probability $P(y = 1|v)$ is estimated by assuming that the features are conditionally independent, given the class label:

$$P(y = 1|v) = \frac{P(y = 1)\prod_{i=1}^{m} P(v_i|y = 1)}{P(v)}$$

Parameter m denotes the number of features of object i, $P(y = 1)$ denotes the prior probability that $y = 1$ and $P(v_i|y = 1)$ denotes the probability of obtaining a value v_i for feature i when the object belongs to the positive class. In the standard version that we tested here, the conditional probabilities $P(v_i|y = 1)$ are assumed to be Gaussian and estimated based on the training set. Note that we excluded all features with zero variance in the training and the corresponding test sets. The threshold is treated as a tuning parameter (cf. Table 1). The MATLAB function used is `fitcnb` from the statistics and machine learning toolbox:

```
% Train Naive Bayes classifier
cnb = fitcnb(trainingData,trainingClassLabels);
% Apply Naive Bayes classifier
testingData(:,exclude) = [];
[~,posterior,~] = predict(cnb,testingData);
class = zeros(size(testingData,1),1);
class(posterior(:,2)>threshold ) = 1;
```

4.4.2. Artificial neural networks (ANN)

An ANN consists of a set of interconnected nodes. Each node is able to receive input signals and transform them into an output signal using a specific transfer function. The nodes are arranged in layers, and each node is connected to every node in the adjacent layers. A typical network consists of three layers. The first layer is the input layer, where feature values of a given object are fed into the network. The second layer is called the hidden layer, and the third layer is the output layer, where the prediction of the network is made (cf. Rumelhart, Hinton, & Williams, 1986). Cybenko (1989) showed that given enough nodes in the hidden layer, an artificial neural network is able to approximate any mapping of inputs to outputs to an arbitrary level of accuracy. We tested feedforward networks with one hidden layer and sigmoid transfer functions (cf. Fig. 4). The network is trained with scaled conjugate gradient backpropagation with σ = 5e-5 (change in weight for the second derivative approximation) and λ = 5e-7 (parameter for regulating the indefiniteness of the Hessian). Training stops when a maximum number of 1,000 epochs is reached or when the performance gradient falls below 1e-6. An epoch corresponds to one forward pass and one backward pass through all the training examples. This setup is commonly used for classification and corresponds to the default setting in MATLAB R2017b. The number of nodes in the hidden layer is varied as a tuning parameter (cf. Table 1). The MATLAB function used is `patternnet` from the neural network toolbox:

```
% Train network
ann = patternnet(networkArchitecture);
ann = train(ann,trainingData,trainingClassLabels);
% Use network
class = ann(testingData);
```

5. Data sets

The machine learning techniques are evaluated on twenty data sets that were downloaded from the UCI Machine Learning Repository (cf. Asuncion & Newman, 2007) and the LIBSVM website (cf. Chang & Lin, 2011). The selected data sets represent a variety of fields including life sciences, physical sciences, engineering, social sciences, business and others. The data sets differ in

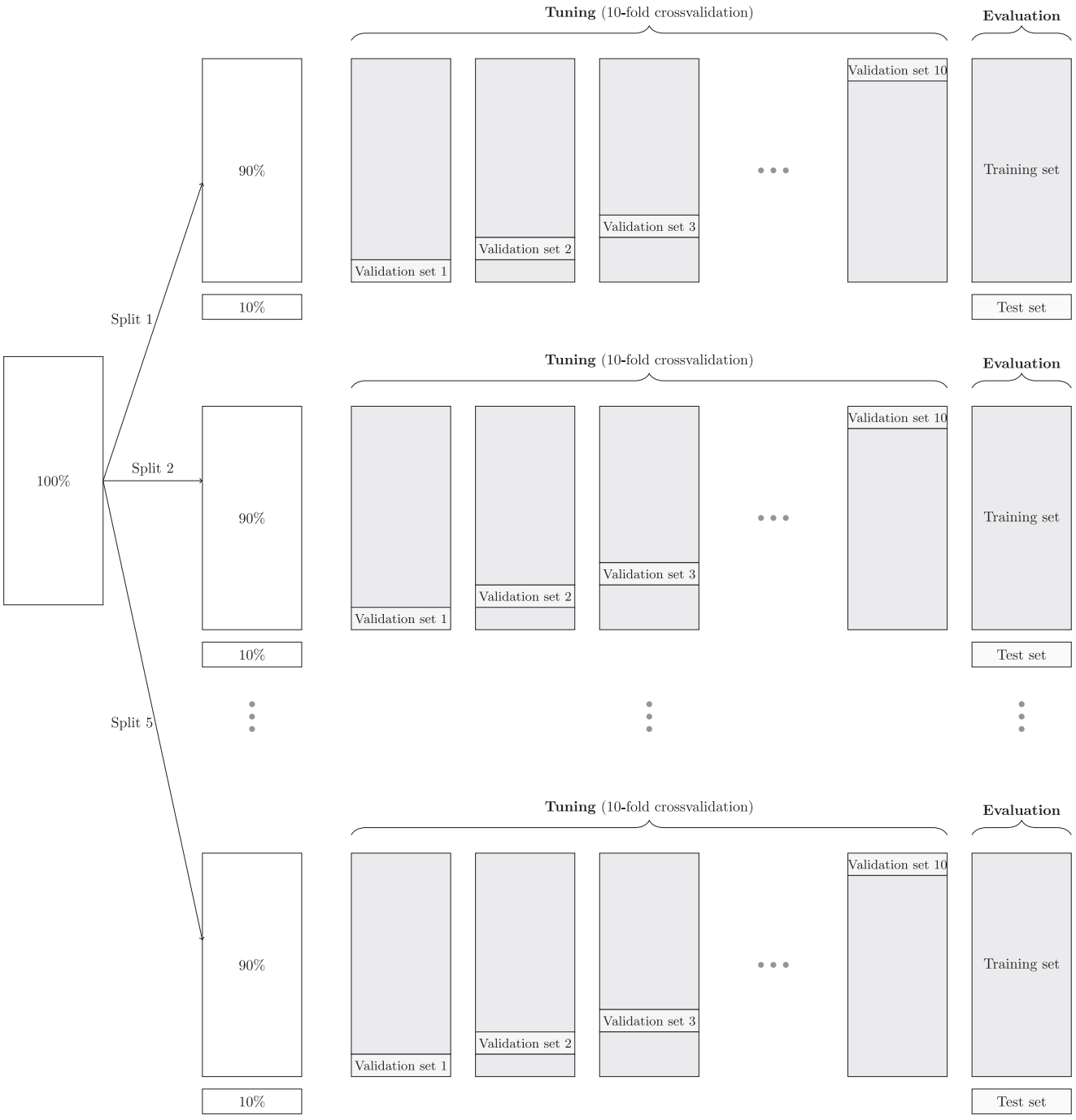

Fig. 5. Partitioning of data sets.

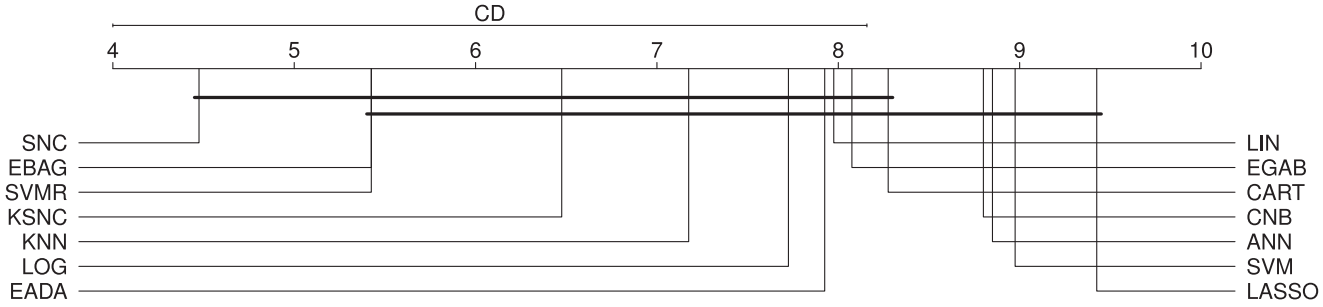

Fig. 6. Graphical comparison of all techniques in terms of average ranks (see Table 4).

Table 3
F_1-score averaged across splits. Interpretation: The highest performing techniques (SNC, KSNC, SVMR, and KNN) are all similarity-based. SNC and KSNC always deliver average F_1-scores of at least 40%, which demonstrates their robustness.

	ANN	CART	CNB	EADA	EBAG	EGAB	LASSO	LIN	LOG	SVM	SVMR	KNN	KSNC	SNC	Avg
IRS	**100.0**	**100.0**	**100.0**	0.0*	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	92.9
WIN	98.2	91.6	**100.0**	**100.0**	97.1	97.1	97.1	97.1	97.1	**100.0**	97.1	97.1	97.1	**100.0**	97.6
PAR	86.8	90.3	86.8	92.1	92.4	92.7	87.1	89.4	89.1	85.2	95.9	**96.7**	**96.1**	**98.0**	91.3
SON	83.1	76.9	79.8	**91.7**	**89.0**	**91.6**	81.6	77.8	79.9	83.8	82.1	86.2	83.6	81.0	83.4
GLA	71.3	64.6	63.7	70.6	**85.9**	71.7	69.3	74.3	73.2	**83.5**	79.1	78.9	79.9	81.5	74.8
HEA	83.0	**86.7**	82.9	79.7	85.7	77.0	**85.8**	84.7	84.4	61.0	85.3	**87.0**	84.5	84.7	82.3
HAB	30.2	31.1	36.0	25.3	32.9	26.2	14.3	**45.6**	**45.3**	37.2	29.3	35.2	40.5	40.9	33.6
VER	89.3	86.6	82.2	89.8	90.6	88.0	89.5	88.0	90.0	80.7	**91.0**	88.4	**92.7**	**91.2**	88.4
ION	91.5	92.1	92.9	94.6	96.0	94.4	89.9	90.1	92.2	**96.8**	**96.8**	92.5	93.9	**96.3**	93.6
DIA	66.5	62.3	68.7	60.0	61.7	62.9	59.7	**69.4**	**73.0**	46.1	61.0	59.4	67.6	**69.9**	63.4
BCW	93.2	94.9	94.9	93.9	**95.5**	93.3	**95.3**	94.6	**95.1**	51.0	**95.1**	**95.1**	93.4	94.2	91.4
AUS	86.5	87.9	**90.2**	89.6	89.2	85.5	**90.5**	**90.7**	89.4	51.6	89.6	88.1	89.3	90.1	86.3
BLD	33.2	43.4	**49.3**	41.3	31.2	31.5	20.1	**50.3**	**50.8**	24.1	37.0	37.3	40.0	42.6	38.0
FOU	**100.0**	98.7	73.7	90.4	99.7	90.7	65.6	67.6	67.4	43.8	**100.0**	**100.0**	**100.0**	**100.0**	85.5
TIC	97.6	96.4	84.0	97.6	98.7	98.7	97.9	97.9	97.9	**100.0**	**100.0**	**99.5**	98.4	98.8	97.4
GER	48.8	55.7	53.8	53.1	53.4	52.8	54.6	**61.1**	**59.4**	46.9	50.5	44.8	49.6	55.4	52.8
CAR	95.6	95.1	92.7	**96.4**	**96.7**	**96.5**	94.9	93.0	94.1	88.4	96.3	94.7	95.1	95.1	94.6
SPL	89.2	**94.3**	87.4	93.7	**97.1**	**93.8**	86.3	86.2	85.9	88.1	91.8	80.9	85.3	86.3	89.0
LE1	50.2	85.6	54.2	65.5	91.1	77.1	0.0	0.0	0.0	**96.8**	**97.6**	94.7	95.1	**95.7**	64.5
LE2	93.6	93.2	73.6	82.0	98.1	82.6	72.5	74.7	74.8	86.2	**98.8**	97.9	**98.1**	**98.5**	87.5
Avg	79.4	81.4	77.3	79.3*	**84.1**	80.2	72.6	76.6	77.0	72.6	83.7	82.7	**84.0**	**85.0**	
Min	30.2	31.1	36.0	25.3*	31.2	26.2	0.0	0.0	0.0	24.1	29.3	35.2	**40.0**	**40.9**	

Table 4
Rank of techniques based on the F_1-score values reported in Table 3. Interpretation: SNC is the leading technique in terms of average rank, followed by SVMR and EBAG.

	ANN	CART	CNB	EADA	EBAG	EGAB	LASSO	LIN	LOG	SVM	SVMR	KNN	KSNC	SNC
IRS	**7.0**	**7.0**	**7.0**	14.0	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**	**7.0**
WIN	5.0	14.0	**2.5**	**2.5**	9.5	9.5	9.5	9.5	9.5	**2.5**	9.5	9.5	9.5	**2.5**
PAR	13.0	8.0	12.0	7.0	6.0	5.0	11.0	9.0	10.0	14.0	4.0	2.0	3.0	**1.0**
SON	7.0	14.0	12.0	**1.0**	3.0	2.0	9.0	13.0	11.0	5.0	8.0	4.0	6.0	10.0
GLA	10.0	13.0	14.0	11.0	**1.0**	9.0	12.0	7.0	8.0	2.0	5.0	6.0	4.0	3.0
HEA	10.0	2.0	11.0	12.0	4.0	13.0	3.0	7.0	9.0	14.0	5.0	**1.0**	8.0	6.0
HAB	10.0	9.0	6.0	13.0	8.0	12.0	14.0	**1.0**	2.0	5.0	11.0	7.0	4.0	3.0
VER	8.0	12.0	13.0	6.0	4.0	11.0	7.0	10.0	5.0	14.0	3.0	9.0	**1.0**	2.0
ION	12.0	11.0	8.0	5.0	4.0	6.0	14.0	13.0	10.0	**1.5**	**1.5**	9.0	7.0	3.0
DIA	6.0	8.0	4.0	11.0	9.0	7.0	12.0	3.0	**1.0**	14.0	10.0	13.0	5.0	2.0
BCW	13.0	6.5	6.5	10.0	**1.0**	12.0	2.0	8.0	4.0	14.0	4.0	4.0	11.0	9.0
AUS	12.0	11.0	3.0	5.0	9.0	13.0	2.0	**1.0**	7.0	14.0	6.0	10.0	8.0	4.0
BLD	10.0	4.0	3.0	6.0	12.0	11.0	14.0	2.0	**1.0**	13.0	9.0	8.0	7.0	5.0
FOU	**3.0**	7.0	10.0	9.0	6.0	8.0	13.0	11.0	12.0	14.0	**3.0**	**3.0**	**3.0**	**3.0**
TIC	11.0	13.0	14.0	12.0	6.0	5.0	9.0	9.0	9.0	**1.5**	**1.5**	3.0	7.0	4.0
GER	12.0	3.0	6.0	8.0	7.0	9.0	5.0	**1.0**	2.0	13.0	10.0	14.0	11.0	4.0
CAR	5.0	7.0	13.0	3.0	**1.0**	2.0	9.0	12.0	11.0	14.0	4.0	10.0	8.0	6.0
SPL	6.0	2.0	8.0	4.0	**1.0**	3.0	9.0	11.0	12.0	7.0	5.0	14.0	13.0	10.0
LE1	11.0	7.0	10.0	9.0	6.0	8.0	13.0	13.0	13.0	2.0	**1.0**	5.0	4.0	3.0
LE2	6.0	7.0	13.0	10.0	4.0	9.0	14.0	12.0	11.0	8.0	**1.0**	5.0	3.0	2.0
Avg	8.85	8.28	8.80	7.92	5.42	8.07	9.43	7.97	7.72	8.97	5.42	7.17	6.47	**4.47**

the number of objects, the number of features, and the distribution of class labels. The collection comprises binary and multiclass classification problems. The labels of all multiclass classification problems were given as numbers. We converted these multiclass problems into binary classification problems by first sorting the numeric labels in ascending order and then selecting the first label as the positive class and all other labels as the negative class. With this sorting strategy, we treat all multiclass data sets in the same way and do not exploit any domain knowledge for the conversion into binary classification problems. Some data sets have missing values. In those sets, we removed the objects that contained missing values. Categorical feature values were replaced by a set of Boolean features (one Boolean feature per category). In the following, we give a short description of each data set. A summary of characteristics can be found in Table 2. The last column of the table states the ratio of the number of objects in the positive class to the number of objects in the negative class.

- The data set *Iris* (IRS) contains three classes that refer to different types of iris plants. We downloaded the data from the LIBSVM website, which gives the data so that the feature values are in the range [−1,1] and the three classes are labeled from 1 to 3. According to the above-specified conversion procedure, we treated label 1 as the positive class and the other two labels as the negative class.
- The data set *Wine* (WIN) contains three classes that refer to different types of wine. This data set also stems from the LIBSVM website and has all feature values in the range [−1,1]. The three classes are labeled from 1 to 3. According to the above-specified conversion procedure, we treated label 1 as the positive class and the other two labels as the negative class.

Table 5
Normalized F_1-score results. Interpretation: SNC achieves the highest average normalized F_1-score. EBAG performs well on average and achieves the highest minimum normalized F_1-score. Performance difference between similarity-based techniques (SNC, SVMR, KNN, and KSNC) and non-similarity-based techniques (except EBAG) increases when normalized F_1-score results are considered.

	ANN	CART	CNB	EADA	EBAG	EGAB	LASSO	LIN	LOG	SVM	SVMR	KNN	KSNC	SNC	Avg
IRS	**100.0**	**100.0**	**100.0**	0.0*	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	92.9
WIN	78.4	0.0	**100.0**	**100.0**	66.1	66.1	66.1	66.1	66.1	**100.0**	66.1	66.1	66.1	**100.0**	72.0
PAR	12.8	39.8	13.0	53.6	56.0	58.4	15.1	32.9	30.7	0.0	83.2	89.9	84.9	**100.0**	47.9
SON	41.7	0.0	19.2	**100.0**	81.7	**99.1**	31.9	5.8	20.3	46.4	35.2	62.6	45.1	27.3	44.0
GLA	34.2	3.7	0.0	30.7	**100.0**	35.7	25.0	47.4	42.8	89.0	69.4	68.6	72.8	80.3	50.0
HEA	84.6	**98.9**	84.3	72.0	95.1	61.5	**95.4**	91.3	89.9	0.0	93.5	**100.0**	90.6	91.3	82.0
HAB	51.0	53.8	69.4	35.3	59.4	38.1	0.0	**100.0**	**99.2**	73.2	48.0	66.9	83.8	85.1	61.7
VER	71.2	49.2	12.1	76.0	82.2	60.2	73.4	60.4	77.2	0.0	85.5	64.2	**100.0**	87.5	64.2
ION	22.9	31.8	44.1	68.0	88.2	65.9	0.0	2.6	33.1	**100.0**	**100.0**	37.7	57.7	93.5	53.3
DIA	75.9	60.1	84.0	51.7	57.9	62.4	50.5	86.7	**100.0**	0.0	55.4	49.6	79.9	88.3	64.4
BCW	95.0	98.9	98.9	96.5	**100.0**	95.2	**99.6**	98.0	**99.1**	0.0	**99.1**	**99.1**	95.4	97.2	90.9
AUS	89.2	92.9	**98.8**	97.2	96.3	86.8	**99.4**	**100.0**	96.6	0.0	97.1	93.4	96.4	98.4	88.7
BLD	42.7	75.9	**95.0**	68.9	36.1	37.3	0.0	**98.4**	**100.0**	12.9	55.1	56.0	64.8	73.1	58.3
FOU	**100.0**	97.7	53.2	82.9	99.4	83.5	38.9	42.4	42.0	0.0	**100.0**	**100.0**	**100.0**	**100.0**	74.3
TIC	84.9	77.8	0.0	84.9	91.7	91.9	86.9	86.9	86.9	**100.0**	**100.0**	**96.9**	89.8	92.8	83.7
GER	24.3	66.7	55.6	50.8	52.7	49.3	60.2	**100.0**	89.6	13.0	34.8	0.0	29.2	65.2	49.4
CAR	87.6	80.8	52.2	**97.0**	**100.0**	**98.2**	78.0	55.6	69.2	0.0	95.9	76.0	80.6	81.4	75.2
SPL	51.3	83.3	40.1	79.1	**100.0**	79.8	33.9	33.2	31.1	44.8	67.5	0.0	27.2	33.3	50.3
LE1	51.4	87.8	55.6	67.1	93.4	79.0	0.0	0.0	0.0	**99.2**	**100.0**	97.1	97.5	**98.1**	66.2
LE2	80.0	78.5	4.1	36.0	97.4	38.5	0.0	8.2	8.9	51.9	**100.0**	96.5	**97.4**	**98.6**	56.8
Avg	64.0	63.9	54.0	70.9*	**82.7**	69.4	47.7	60.8	64.1	41.5	79.3	71.0	77.9	**84.6**	
Min	12.8	0.0	0.0	30.7*	**36.1**	**35.7**	0.0	0.0	0.0	0.0	**34.8**	0.0	27.2	27.3	

Table 6
Relative F_1-score. Interpretation: SNC and KSNC are the leading techniques in terms of average and minimum relative F_1-score across data sets.

	ANN	CART	CNB	EADA	EBAG	EGAB	LASSO	LIN	LOG	SVM	SVMR	KNN	KSNC	SNC	Avg
IRS	**107.7**	**107.7**	**107.7**	0.0*	**107.7**	**107.7**	**107.7**	**107.7**	**107.7**	**107.7**	**107.7**	**107.7**	**107.7**	**107.7**	100.0
WIN	100.6	93.8	**102.4**	**102.4**	99.5	99.5	99.5	99.5	99.5	**102.4**	99.5	99.5	99.5	**102.4**	100.0
PAR	95.1	98.9	95.1	100.8	101.1	101.5	95.4	97.9	97.6	93.3	105.0	**105.9**	**105.2**	**107.3**	100.0
SON	99.6	92.2	95.6	**109.9**	**106.7**	**109.7**	97.9	93.2	95.8	100.4	98.4	103.3	100.2	97.0	100.0
GLA	95.3	86.3	85.2	94.3	**114.8**	95.8	92.6	99.2	97.9	**111.6**	105.7	105.5	106.7	109.0	100.0
HEA	100.8	**105.3**	100.7	96.8	104.1	93.5	**104.2**	102.9	102.5	74.2	103.6	**105.7**	102.7	102.9	100.0
HAB	90.1	92.7	107.2	75.5	97.9	78.1	42.6	**135.7**	**134.9**	110.7	87.3	104.8	120.6	121.8	100.0
VER	100.9	98.0	92.9	101.6	102.4	99.5	101.2	99.5	101.8	91.3	**102.9**	100.0	**104.9**	**103.2**	100.0
ION	97.8	98.4	99.3	101.1	102.6	100.9	96.1	96.3	98.5	**103.4**	**103.4**	98.9	100.3	**103.0**	100.0
DIA	104.8	98.2	108.3	94.6	97.2	99.1	94.1	**109.4**	**115.1**	72.7	96.2	93.7	106.5	**110.1**	100.0
BCW	102.0	103.9	103.9	102.7	**104.4**	102.1	**104.2**	103.5	**104.0**	55.8	**104.0**	**104.0**	102.2	103.1	100.0
AUS	100.2	101.9	**104.5**	103.8	103.4	99.1	**104.8**	**105.1**	103.6	59.8	103.8	102.1	103.4	104.4	100.0
BLD	87.4	114.2	**129.7**	108.6	82.0	83.0	52.8	**132.5**	**133.8**	63.3	97.4	98.2	105.3	112.0	100.0
FOU	**116.9**	115.4	86.1	105.7	116.5	106.1	76.7	79.1	78.8	51.2	**116.9**	**116.9**	**116.9**	**116.9**	100.0
TIC	100.2	99.0	86.3	100.2	101.3	101.4	100.5	100.5	100.5	**102.7**	**102.7**	**102.2**	101.0	101.5	100.0
GER	92.3	105.3	101.9	100.4	101.0	100.0	103.3	**115.6**	**112.4**	88.8	95.5	84.8	93.8	104.9	100.0
CAR	101.1	100.5	98.0	**101.9**	**102.2**	**102.0**	100.2	98.3	99.5	93.5	101.8	100.1	100.5	100.5	100.0
SPL	100.2	**106.0**	98.1	105.2	**109.0**	**105.4**	97.0	96.9	96.5	99.0	103.1	90.8	95.8	96.9	100.0
LE1	77.7	132.6	84.0	101.5	141.2	119.5	0.0	0.0	0.0	**149.9**	**151.2**	146.8	147.4	**148.3**	100.0
LE2	107.0	106.5	84.1	93.7	112.2	94.5	82.9	85.4	85.6	98.5	**113.0**	111.9	**112.2**	**112.6**	100.0
Avg	98.9	102.8	98.6	100.0*	**105.4**	99.9	87.7	97.9	98.3	91.5	105.0	104.1	**106.6**	**108.3**	
Min	77.7	86.3	84.0	75.5*	82.0	78.1	0.0	0.0	0.0	51.2	87.3	84.8	**93.8**	**96.9**	

- The data set *Parkinson* (PAR) contains voice recordings from healthy people (negatives) and people with Parkinson's disease (positives).
- The data set *Connectionist Bench, Sonar* (SON) contains patterns of sonar signals that bounce off mines (positives) or rocks (negatives).
- The data set *Glass* (GLA) contains six classes that refer to different types of glass. The data set was downloaded from the LIBSVM website and has all feature values in the range [−1,1]. The six classes are labeled {1,2,3,5,6,7}. According to the above-specified conversion procedure, we treated label 1 as the positive class and all other labels as the negative class.
- The data set *Heart disease* (HEA) is a set of patients with (positives) or without (negatives) heart disease.
- The data set *Haberman's Survival* (HAB) is a set of patients who have received breast cancer surgery. Patients who survived five years or longer after the surgery form the positive class and patients who died within five years after the surgery form the negative class.
- The data set *Vertebral Column* (VER) contains biomechanical features of patients with (negatives) and without (positives) spinal disorders.
- The data set *Ionosphere* (ION) contains good (positives) and bad (negatives) radar returns.
- The data set *Pima Indians Diabetes* (DIA) contains a set of female patients of Pima Indian heritage with (positives) and without (negatives) diabetes mellitus. As indicated on the UCI website, zero values in the data set are likely to encode missing values. We therefore discarded all objects with zero values.
- The data set *Breast Cancer Wisconsin (Original)* (BCW) contains features of malignant (positives) and benign (negatives) breast cancer tumors.

Table 7
The standard deviation of F_1-scores across splits. Interpretation: All techniques achieve similar F_1-scores for the different splits.

	ANN	CART	CNB	EADA	EBAG	EGAB	LASSO	LIN	LOG	SVM	SVMR	KNN	KSNC	SNC	Avg
IRS	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00
WIN	4.07	5.33	**0.00**	**0.00**	6.39	6.39	6.39	6.39	6.39	**0.00**	6.39	6.39	6.39	**0.00**	4.32
PAR	5.93	4.23	7.59	5.54	5.99	7.57	4.17	4.52	5.01	7.39	**1.50**	**2.23**	**2.70**	2.87	4.80
SON	6.75	15.14	**3.91**	5.73	**2.85**	**4.41**	6.73	4.69	8.77	9.45	7.45	10.15	12.87	23.22	8.72
GLA	7.66	12.17	11.93	**5.41**	11.34	**4.56**	18.94	17.00	9.64	8.34	10.04	7.78	7.75	**5.56**	9.87
HEA	7.13	8.02	6.53	**3.61**	7.44	**4.59**	6.52	**3.68**	8.50	12.08	5.15	12.39	11.23	12.64	7.82
HAB	10.47	13.34	20.65	12.21	**4.91**	13.31	14.31	19.28	16.51	23.51	14.11	21.15	**8.67**	12.00	14.60
VER	3.77	4.96	**2.21**	4.74	3.47	4.64	4.70	2.92	**1.96**	3.25	4.34	4.04	2.62	**1.49**	3.51
ION	3.99	**1.82**	3.84	4.22	**1.96**	6.12	7.68	6.49	8.65	**3.39**	**3.39**	3.95	3.95	3.45	4.49
DIA	9.68	13.32	**6.74**	8.93	12.25	14.81	8.51	7.57	**7.01**	16.32	9.51	8.66	7.80	**7.34**	9.89
BCW	5.77	2.22	2.22	3.22	**1.87**	2.82	2.85	2.31	**1.94**	9.99	**1.94**	**1.94**	2.26	2.15	3.11
AUS	5.06	3.96	2.54	3.00	3.35	4.66	**2.18**	**2.46**	2.99	9.91	**1.76**	4.39	2.95	2.84	3.72
BLD	13.49	14.44	12.42	16.63	15.64	13.89	**6.05**	8.22	10.32	23.91	10.98	10.84	**5.01**	**3.03**	11.78
FOU	**0.00**	1.32	5.35	2.27	0.71	2.07	6.11	4.73	5.00	18.22	**0.00**	**0.00**	**0.00**	**0.00**	3.27
TIC	1.56	2.82	2.88	1.99	1.92	2.17	1.65	1.65	1.65	**0.00**	**0.00**	0.46	**0.11**	2.17	1.50
GER	11.67	11.16	10.48	7.95	12.11	**6.14**	**7.26**	9.60	10.61	7.63	8.38	**6.96**	9.82	8.32	9.15
CAR	0.83	1.52	1.61	**0.66**	0.76	1.00	1.44	1.98	1.29	2.46	**0.74**	1.03	0.77	**0.54**	1.19
SPL	**0.77**	1.44	1.96	1.29	**0.98**	1.47	1.44	1.17	1.19	1.47	1.01	**0.88**	1.62	2.01	1.34
LE1	45.87	1.94	1.54	3.55	2.99	3.11	**0.00**	**0.00**	**0.00**	2.00	1.85	1.91	1.24	1.56	4.83
LE2	1.40	0.71	0.81	0.58	**0.34**	0.82	1.04	0.82	0.80	0.42	**0.20**	0.50	0.38	**0.33**	0.65
Avg	7.29	5.99	5.26	**4.58**	4.86	5.23	5.40	5.27	5.41	7.99	**4.44**	5.28	**4.41**	4.58	
Max	45.87	15.14	20.65	16.63	15.64	**14.81**	18.94	19.28	16.51	23.91	**14.11**	21.15	**12.87**	23.22	

Table 8
The best F_1-scores obtained for validation sets averaged across splits. Interpretation: There is **no indication of systematic overfitting**, as these F_1-scores are very similar to those reported for the test sets (see Table 3).

	ANN	CART	CNB	EADA	EBAG	EGAB	LASSO	LIN	LOG	SVM	SVMR	KNN	KSNC	SNC	Avg
IRS	**100.0**	**100.0**	**100.0**	0.0*	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	92.9
WIN	97.7	95.2	97.2	96.0	97.3	96.1	99.1	98.5	98.1	**100.0**	**99.7**	99.5	98.9	**99.7**	98.1
PAR	92.3	91.8	90.2	94.0	93.5	93.5	90.8	92.4	91.8	85.6	95.7	**97.1**	**97.1**	**97.2**	93.1
SON	82.4	80.1	78.7	85.5	85.5	84.3	78.9	79.0	79.2	**87.9**	**88.1**	87.0	87.7	**88.0**	83.7
GLA	59.5	72.1	61.0	69.2	**79.4**	68.2	60.7	64.2	66.1	69.2	71.4	69.8	72.3	72.5	68.3
HEA	76.5	79.8	**81.9**	76.8	79.0	74.5	**81.1**	**81.5**	81.0	59.3	80.7	79.7	77.9	78.4	77.7
HAB	34.7	41.2	40.8	33.5	31.5	33.5	23.4	**49.4**	**48.9**	36.7	31.3	37.3	**47.4**	41.8	37.9
VER	87.7	88.5	83.8	86.2	88.4	85.2	88.3	87.6	**90.1**	80.4	89.0	88.5	**89.7**	**89.9**	87.4
ION	92.9	92.7	93.6	94.5	94.7	94.9	91.9	90.8	91.4	**96.0**	**96.1**	92.8	92.9	**95.6**	93.6
DIA	61.2	**67.5**	67.0	62.1	65.4	59.7	62.9	**69.4**	**69.5**	47.2	63.4	62.4	66.9	67.1	63.7
BCW	95.0	96.2	96.4	95.1	96.4	95.1	95.7	**96.9**	96.6	51.6	96.4	96.5	**96.7**	**96.7**	93.0
AUS	83.6	**85.3**	**85.3**	84.5	84.7	82.3	**85.6**	85.1	85.2	49.5	84.5	84.5	84.8	84.9	82.1
BLD	37.6	44.0	**48.9**	39.5	37.8	36.9	19.5	**50.7**	**51.3**	27.2	34.4	40.4	45.8	45.2	39.9
FOU	**100.0**	98.2	69.4	86.4	99.1	88.6	62.7	64.7	64.5	42.6	**100.0**	**100.0**	**100.0**	**100.0**	84.0
TIC	98.7	98.0	83.3	98.9	99.2	**99.4**	98.8	98.8	98.8	**100.0**	**100.0**	99.4	98.8	99.4	98.0
GER	54.0	52.8	**59.6**	53.4	52.3	52.0	55.8	**61.4**	**61.6**	40.4	56.2	49.8	53.9	58.8	54.4
CAR	94.9	95.9	93.2	**96.2**	**97.2**	**96.7**	94.1	93.5	94.2	88.1	95.8	95.7	95.7	95.7	94.8
SPL	89.0	**95.4**	88.4	**93.9**	**97.3**	93.7	85.1	85.1	85.2	86.8	91.6	81.9	86.3	87.3	89.1
LE1	77.8	84.5	56.8	69.1	90.0	76.9	0.0	0.0	0.2	**95.7**	**96.6**	95.5	95.5	**95.9**	66.7
LE2	92.7	92.5	73.5	81.6	97.5	82.8	72.7	74.6	75.0	86.3	**98.5**	97.9	**97.9**	**98.1**	87.3
Avg	80.4	82.6	77.5	78.8*	83.3	79.7	72.3	76.2	76.4	71.5	**83.5**	82.8	**84.3**	**84.6**	
Min	34.7	41.2	40.8	33.5*	31.5	33.5	0.0	0.0	0.2	27.2	31.3	37.3	**45.8**	41.8	

- The data set *Australian Credit Approval* (AUS) contains credit card applications of people who were (positives) or were not (negatives) granted credit.
- The data set *Blood Transfusion Service Center* (BLD) contains a set of blood donors. The positive class label indicates that the donor donated blood in 2007.
- The data set *Fourclass* (FOU) is an artificially created data set in which the objects are positioned in two-dimensional space such that they are not linearly separable. The positive and negative labels are given for this data set.
- The data set *Tic-Tac-Toe Endgame* (TIC) encodes the complete set of board configurations at the end of the game when player "x" plays first. Wins for player "x" are treated as positives.
- The data set *German* (GER) contains a set of people described by a set of features as good (negatives) or bad (positives) credit risks.
- The data set *Cardiotocography* (CAR) consists of fetal cardiotocograms that belong to one of the three classes "normal" (label 1), "suspect" (label 2), and "pathologic" (label 3). According to the above-specified conversion procedure, we treated label 1 as the positive class and the other two labels as the negative class.
- The data set *Splice* 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.
- The data set *Letter Recognition* comprises numerical features of letter images. As in the study of Caruana and Niculescu-Mizil (2006), we used two binary variants of this data set. 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.

6. Experimental design

As shown in Fig. 5, we created five random partitions (split 1, 2, ..., 5) of each data set by applying stratified random sampling. Each partition divides the entire data set into a training (90%) and a test set (10%). The training set is further divided into 10 equal-sized sets, called folds, for a stratified 10-fold cross-validation that is used during tuning. The tuning and the evaluation of the machine learning techniques are performed separately for each split, and the average performance across the splits is reported (see pseudocode experimental analysis). Note that the same splits and the same folds are used for all techniques.

Sections 6.1 and 6.2 describe how the machine learning techniques are tuned and evaluated, respectively. Section 6.3 presents the four data preprocessing options that are considered during tuning, and Section 6.4 introduces the four performance measures that we use to compare the different techniques. Section 6.5 discusses alternative experimental designs that we tested and as noted, all provide consistent results. Therefore, we report in Section 7 only the results of the baseline experimental design that we describe next in Sections 6.1–6.4. The experimental analysis is implemented in MATLAB R2017b and the computations were performed on a workstation with two Intel Xeon CPUs (model E5-2687W v3) with clock speed 3.10 GHz and 256 GB of RAM.

6.1. Tuning

The goal of tuning is to find a promising preprocessing option and a set of tuning parameter values. We use random search with a predefined time limit to determine the values of the tuning parameters and the best preprocessing option. For several algorithms and data sets, Bergstra and Bengio (2012) showed that random search is superior to grid search or manual search. It is noted that we still tested alternative tuning strategies including grid search with consistent results (cf. Section 6.5).

The tuning is performed for a given technique, a given split, and a given performance measure referred to here as the tuning criterion (see pseudocode tuning). For each tuning parameter, a uniformly distributed random number is selected from a prespecified interval. Table 1 lists the lower and upper bounds of these intervals for all tuning parameters and all techniques. The set of randomly selected tuning parameter values is then evaluated with each preprocessing option (see Section 6.3) based on the training set of the given split using 10-fold cross-validation. This process is repeated with different sets of randomly selected tuning parameter values until the time limit is reached. Then, the best performing combination (of preprocessing option and set of tuning parameter values) with respect to the tuning criterion is identified. This combination is used for the evaluation that we will explain in the next section. The time limit was determined for each data set with the formula

$$\text{time limit} = \left\lfloor \left(\frac{n}{100}\right)^{1.25} + 0.5 \right\rfloor,$$

where n denotes the number of objects in the data set. The exponent of the formula is chosen such that the time limit grows over-proportionally with the size of the data set.

Using random search has several advantages. First, it is possible to control the tuning time and roughly allocate the same amount of tuning time to all techniques. Second, it is not necessary to choose a discrete set of values for each tuning parameter as it is in grid search. Instead, it is sufficient to define only a tuning-parameter-specific lower and upper bound, which reduces user influence significantly.

6.2. Evaluation

The evaluation is performed for a given technique, a given split, a given performance measure, and a given combination of tuning parameter values and preprocessing option. The entire training set of the given split is used to classify the objects in the test set. In Section 7 we report the respective performance measures averaged across splits.

6.3. Preprocessing

Preprocessing procedures can sometimes improve the performance of machine learning techniques by modifying the input data before the machine learning technique is applied. Some preprocessing procedures scale the feature values to a certain range to prevent features with large values from dominating distance or similarity computations even though other features are more important for distinguishing the objects. Attribute scaling can also reduce the running time of machine learning techniques that use gradient descent due to faster convergence. Other preprocessing procedures reduce the dimensionality of the input data in order to reduce noise (non-relevant features). Four preprocessing options are considered here:

- *No preprocessing* is performed.
- *Normalization* scales the values of the features to the interval [0,1]. If v_i represents the vector of values of a given feature i, then the vector of normalized values v_i' is computed as follows:

$$v_i' = \frac{v_i - \min(v_i)}{\max(v_i) - \min(v_i)}.$$

- *Dimensionality reduction* is used to reduce the number of features in the data sets. Here we reduce the number of features by performing Principal Component Analysis (PCA) and selecting only the first leading principal components that explain at least 80% of the variance in the original data set.
- *First normalization and then dimensionality reduction* is applied.

6.4. Performance measures

The study in Caruana and Niculescu-Mizil (2006) demonstrated that different performance measures are highly correlated. Based on this result, we focused here on the four most widely used performance measures: F_1-score, precision, recall, and accuracy. Let TP, TN, FP, and FN denote the number of true positives, true negatives, false positives, and false negatives, respectively. F_1-score, precision, recall, and accuracy are then defined as follows:

$$F_1\text{-score} = \frac{2TP}{2TP + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

Note that the F_1-score is the harmonic mean of precision and recall. As some of the techniques do not have a probabilistic output, we do not report here the performance measure AUC (area under the curve).

6.5. Alternative experimental designs

It is important to note that although we present one specific experimental design, we tested numerous alternative designs. It turned out that irrespective of the choice of experimental design, the observed general performance of the techniques coincides with the performance we report for the baseline design in the next section. In particular, SNC or KSNC consistently showed best or close to best performance.

Specifically, we tested designs D1 to D4, which are all based on the partitioning of data sets shown in Fig. 5 but differ with respect to parameter tuning.

- Design D1 corresponds to the above-described design but imposes smaller time limits for the different data sets.
- Design D2 is also based on random search but instead of a time limit, a fixed number of c randomly chosen combinations of tuning parameter values is applied. The value c was selected from the set {15,30}.
- Design D3 uses grid search instead of random search. The set of values for each tuning parameter was defined such that each technique was tested with roughly the same number of tuning parameter combinations.
- Design D4 uses grid search instead of random search. The set of values for each tuning parameter was defined individually for each tuning parameter. In this design, techniques with a greater number of tuning parameters (e.g., KSNC) were applied with a greater number of tuning parameter combinations than techniques with a smaller number of tuning parameters (e.g., LOG).

7. Computational results

In this section, we report and comment on the results obtained when tuning and evaluating the techniques with respect to the F_1-scores. Table 3 presents the F_1-scores of each machine learning technique for the different data sets averaged across the splits. The rows of the table refer to data sets, and the columns refer to machine learning techniques. The data sets are listed in ascending order with respect to the number of objects they contain, i.e., the smallest set is listed in the first row. For each data set, the top three values that are within 5% of the best result are stated in bold. An F_1-score of zero results when a machine learning technique assigned all objects of the test set to the negative class. This happened once to the regression-based algorithms (LIN, LOG, LASSO) and to EADA. The zero F_1-score of EADA for data set IRS occurs because the first weak learner already classified all objects in the validation sets correctly. If this happens, then the MATLAB implementation of AdaBoost does not add any weak learners and classifies all objects as negatives. The last two rows of the table state the average and the minimum value for each algorithm. As indicated by the asterisks, the average and the minimum for EADA are computed without the zero F_1-score. The unnormalized average across data sets should be considered with caution, as different data sets have different natural scales for the F_1-score. We therefore rank the techniques for each data set separately such that the best-performing technique is given rank 1, the second best, rank 2, etc., as shown in Table 4. In the case of ties (such as in IRS), average ranks are assigned. The ranks are computed based on the F_1-score values reported in Table 3. The average rank is more meaningful than the average F_1-score across data sets. In addition, we normalized the values of Table 3 by subtracting from each value the minimum value obtained for the respective data set and dividing the result by the difference between the maximum value and the minimum value obtained for the respective data set. In this way, the highest F_1-score receives the value 100, and the lowest F_1-score receives the value zero. The normalized F_1-score results are shown in Table 5. Table 6 reports the F_1-score relative to the average F_1-score that was obtained for each data set. This allows to compare the performance of each technique with the data set-specific average performance of all techniques.

The similarity-based algorithms SNC, SVMR, KSNC, and KNN are all among the top five techniques. As can be seen in Tables 4 and 5, the highest average rank as well as the highest average normalized F_1-score was achieved by SNC. With the exception of EBAG, the average performance of all non-similarity-based machine learning techniques is considerably lower than the performance of SNC. Perhaps the most surprising result is that SNC is also superior to all other machine learning techniques in terms of robustness. The SNC algorithm not only achieves the highest average normalized F_1-score but also the highest average relative F_1-score and the highest minimum relative F_1-score. The comparison of the techniques in terms of average ranks is visualized in Fig. 6 using CD (critical difference) diagrams introduced by Demšar (2006). We applied the Friedman test to statistically compare the average ranks of the techniques. The null-hypothesis of the Friedman test states that all of the techniques are equivalent and so their ranks should be equal. We can reject the null-hypothesis for $\alpha = 0.1$. According to the Nemenyi test, the performance of two techniques is significantly different if the corresponding average ranks differ by at least a critical difference. This critical difference is also shown in Fig. 6. At $\alpha = 0.1$, the performance of SNC is significantly better than the performance of ANN, CNB, LASSO, and SVM. The performance of the other techniques are not significantly different from each other according to the Nemenyi test. This is to be expected, as the Nemenyi test is known to be very conservative (cf. Garcia & Herrera, 2008; Ulaş, Yıldız, & Alpaydın, 2012). More powerful post hoc tests such as the Bergmann and Hommel procedure (see Bergmann & Hommel, 1988) are not applied here, as they lead to intense computation for fourteen techniques and twenty data sets.

Algorithm 1 Pseudocode experimental analysis.

```
for each technique do
    for each data set do
        for each split do
            Perform tuning
            Perform evaluation
        end for
        Calculate the average performance across splits
    end for
end for
```

Algorithm 2 Pseudocode tuning.

```
function TUNING(Technique, split, tuning criterion, time limit)
    do
        Randomly draw a value for each tuning parameter from prespecified range
        for each preprocessing option do
            for each fold do
                Hold-out objects from the respective validation set
                Use remaining training objects to classify hold-out objects
            end for
            Calculate the average performance across folds
        end for
    while time limit not reached
    Determine the best combination of tuning parameter set and preprocessing option
end function
```

Table 9
Total running time in seconds for performing an evaluation of all splits. Interpretation: The Naive Bayes classifier (CNB), the *K*-nearest neighbor algorithm (KNN), classification trees (CART), linear regression (LIN), and logistic regression (LOG) are the fastest techniques. The running time of similarity-based techniques increases with the increasing size of the data sets. The running time of SVM with polynomial and rbf kernel (SVM) is erratic and strongly depends on the tuning parameter values in addition to the data set size. The running times of SNC, KSNC, and KNN are stable and predictable.

	ANN	CART	CNB	EADA	EBAG	EGAB	LASSO	LIN	LOG	SVM	SVMR	KNN	KSNC	SNC
IRS	0.964	0.037	0.040	0.099	9.672	11.657	0.070	0.047	0.083	**0.014**	**0.013**	0.040	0.014	**0.013**
WIN	1.027	0.038	0.037	11.009	10.517	11.868	0.055	0.046	0.061	**0.013**	0.014	0.038	**0.014**	**0.012**
PAR	0.974	0.039	0.037	10.988	11.603	12.209	0.727	0.065	0.019	**0.015**	0.018	0.036	**0.015**	**0.014**
SON	1.088	0.041	0.043	11.316	12.906	12.449	0.522	0.056	0.067	0.028	**0.027**	0.038	**0.017**	**0.016**
GLA	0.940	0.038	0.038	10.718	11.810	11.989	0.150	0.046	**0.019**	0.091	0.544	0.038	**0.015**	**0.014**
HEA	0.926	0.038	0.048	10.900	12.776	12.018	0.041	0.047	**0.020**	46.955	0.024	0.037	**0.018**	**0.016**
HAB	0.935	0.038	0.037	10.676	13.048	11.770	0.036	0.045	**0.019**	25.068	0.687	0.038	**0.018**	**0.016**
VER	1.035	0.038	0.036	11.427	12.451	12.157	0.049	0.046	0.023	**0.016**	0.019	0.039	**0.019**	**0.017**
ION	1.033	0.041	0.049	11.888	12.877	12.552	0.120	0.065	0.168	**0.028**	0.028	0.039	**0.021**	**0.021**
DIA	0.967	0.040	0.038	11.519	13.605	12.477	0.039	0.050	**0.018**	73.582	**0.026**	0.040	0.030	**0.029**
BCW	0.976	0.039	0.036	11.602	12.698	12.405	0.043	0.045	**0.022**	**0.012**	**0.019**	0.040	0.069	0.064
AUS	1.106	0.043	**0.040**	12.154	16.288	12.720	**0.040**	0.049	**0.028**	91.560	0.088	0.041	0.071	0.065
BLD	1.172	0.040	**0.036**	11.587	15.959	12.494	**0.039**	0.047	**0.034**	179.684	10.024	0.041	0.073	0.069
FOU	1.529	0.041	**0.036**	11.700	12.735	12.656	0.039	0.046	**0.019**	145.422	**0.029**	0.041	0.100	0.096
TIC	1.421	**0.049**	**0.052**	12.218	17.273	13.139	0.319	0.078	0.351	0.240	0.248	**0.041**	0.132	0.120
GER	1.304	0.056	**0.051**	12.396	19.632	13.097	0.089	0.070	**0.046**	14.386	5.701	**0.043**	0.158	0.143
CAR	2.047	**0.075**	**0.053**	13.663	20.412	14.074	0.389	0.081	0.317	60.031	0.384	**0.049**	0.637	0.595
SPL	2.549	**0.100**	0.174	15.514	32.727	15.539	0.409	**0.166**	0.239	1.787	6.137	**0.099**	1.402	1.318
LE1	53.913	**0.197**	**0.154**	57.473	76.001	21.330	1.491	**0.241**	0.383	38.082	3.898	0.343	60.359	56.400
LE2	60.357	0.441	**0.274**	60.164	119.240	37.368	1.399	**0.244**	**0.312**	41.841	174.805	0.345	62.060	57.843
Sum	136.263	**1.468**	**1.310**	319.011	464.229	285.968	6.066	1.580	2.247	718.855	202.735	**1.466**	125.242	116.882

Table 10
Total running time in seconds for performing tuning of all splits. Interpretation: The observed running time for tuning is close to the imposed time limit for most techniques. Exceptions are EADA, EBAG, EGAB, and SVM, whose running times vary strongly between tuning parameter values.

	ANN	CART	CNB	EADA	EBAG	EGAB	LASSO	LIN	LOG	SVM	SVMR	KNN	KSNC	SNC
IRS	41	10	10	12	382	451	10	11	11	10	10	11	10	10
WIN	40	10	10	433	413	461	10	11	11	10	10	10	10	10
PAR	38	11	10	422	440	467	27	11	10	165	10	10	10	10
SON	38	11	12	451	477	483	30	11	12	29	10	10	10	10
GLA	37	15	15	436	449	470	16	15	15	30	39	16	15	15
HEA	37	16	16	471	484	477	15	16	15	401	15	16	15	15
HAB	37	21	21	429	511	457	20	21	20	228	23	21	20	20
VER	38	21	20	455	481	462	20	21	20	282	20	20	20	20
ION	41	26	26	454	481	490	26	26	27	73	26	26	25	25
DIA	37	31	31	452	523	477	31	31	30	643	31	30	30	30
BCW	78	55	56	455	476	491	55	56	55	243	56	56	55	55
AUS	83	55	56	488	587	480	56	55	55	796	70	56	55	55
BLD	92	60	61	448	577	484	61	60	61	1,482	94	60	60	60
FOU	103	76	75	462	502	475	75	75	75	1,287	76	75	75	75
TIC	112	86	85	490	703	493	87	86	88	119	87	85	85	85
GER	96	91	91	480	798	504	91	91	91	114	155	91	90	90
CAR	261	231	231	521	822	540	231	231	232	977	260	231	232	232
SPL	424	377	378	641	1,520	617	380	377	378	496	524	376	379	378
LE1	4,293	3,762	3,763	4,140	6,341	3,834	3,767	3,763	3,763	48,772	4,049	3,765	4,047	3,838
LE2	5,733	3,766	3,764	3,828	5,577	3,883	3,775	3,763	3,762	5,201	5,814	3,764	3,846	3,954
Sum	11,660	8,730	8,730	15,968	22,544	16,498	8,784	8,731	8,733	61,359	11,379	8,729	9,091	8,989

Table 7 reports the standard deviation of the F_1-scores across splits. The lowest standard deviations are achieved by KSNC, SVMR, and EADA. However, all techniques exhibit rather low standard deviations.

None of the techniques appears to systematically suffer from overfitting as the F_1-scores that we reported in Table 3 are similar to the best F_1-scores obtained for the validation sets averaged across splits (see Table 8).

In additional experiments, we tuned the techniques for F_1-scores and evaluated for precision and recall. The best average precision results were obtained by SVMR (86.4%) followed by EBAG (86.3%) and KNN (86.0%). The best average recall results were obtained by SNC (89.1%) followed by KSNC (87.0%) and CNB (83.6%). Finally, we tuned for accuracy and evaluated for accuracy. The best average accuracy results were obtained by SNC (90.1%) followed by SVMR (89.9%) and EBAG (89.9%).

Turning to an analysis of the running time of the different machine learning techniques, Table 9 lists for each technique and data set the total running time required to evaluate all splits. The evaluation of a split includes the time required to train the technique on the training set and the time required to classify the objects of the test set. Overall, the Naive Bayes classifier (CNB), the *K*-nearest neighbor algorithm (KNN), classification trees (CART), linear regression (LIN), and logistic regression (LOG) are clearly the fastest techniques. SNC has very low evaluation times for data sets that are small in terms of the number of objects. However, the evaluation times increase considerably when the number of objects increases. Also the evaluation times of ANN, EADA, EBAG, EGAB, SVM, and SVMR go up sharply as the number of objects increases. For the

similarity-based machine learning techniques (KNN, KSNC, SNC, and SVM), the evaluation time increase occurs because the number of pairwise similarities grows quadratically in the size of the data set. This hinders the applicability of similarity-based machine learning techniques for very large data sets. However, recently in Hochbaum and Baumann (2014), Hochbaum and Baumann (2016), Baumann et al. (2016), Baumann et al. (2017), we devised a method called sparse computation that generates only the relevant similarities, resulting in sparse similarity matrices even for massively large data sets. The results in Hochbaum and Baumann (2014), Hochbaum and Baumann (2016), Baumann et al. (2016), Baumann et al. (2017) demonstrate that with this technique, significant improvements in running time can be achieved with minimal loss in accuracy.

The tuning time measures the time required to determine the best combination of preprocessing option and tuning parameter values. As mentioned in Section 6.1, we provided the same tuning time for all techniques. In Table 10, we report for each technique and each data set the actual total tuning time required for all splits. The reported tuning times for ANN, EADA, EBAG, EGAB, SVM, and SVMR exceed the prescribed time limit for some data sets quite considerably. This indicates that performing a single or a few runs (one or a few random combinations of tuning parameter values) with these algorithms sometimes exceeds the prescribed time limit. Fast techniques such as CART, CNB, KSNC, LASSO, LIN, LOG, and SNC could test a large number of tuning parameter combinations within the given time limit. Note that the tuning times of SNC and KSNC can be reduced substantially when grid search is used instead of random search because the similarities computed for a specific value of ϵ can be reused for all values of λ and k.

8. Conclusions

This paper presents a detailed comparison of twelve established and two new machine learning techniques applied to twenty data sets. The machine learning techniques are considered in their basic forms with well-defined sets of tuning parameters. The study demonstrates that the new combinatorial optimization algorithms consistently show the best or close to best performance, and their performance is also the most robust. An important insight derived from this study is that similarity-based algorithms perform considerably better than non-similarity-based machine learning algorithms. This implies that further investigations of effective machine learning techniques should focus on similarity-based algorithms and on combinatorial optimization algorithms.

Acknowledgment

We would like to thank the anonymous reviewers for their valuable comments and suggestions that helped to improve the quality of the paper.

References

Ahmed, N. K., Atiya, A. F., Gayar, N. E., & El-Shishiny, H. (2010). An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews, 29*, 594–621.

Asuncion, A., & Newman, D. (2007). UCI machine learning repository. http://www.ics.uci.edu/~mlearn/MLRepository.html.

Bauer, E., & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: bagging, boosting, and variants. *Machine Learning, 36*, 105–139.

Baumann, P., Hochbaum, D. S., & Spaen, Q. (2016). Sparse-reduced computation: : enabling mining of massively-large data sets. In *Proceedings of the 5th international conference on pattern recognition applications and methods* (pp. 224–231).

Baumann, P., Hochbaum, D. S., & Spaen, Q. (2017). High-performance geometric algorithms for sparse computation in big data analytics. In *Proceedings of the IEEE international conference on big data* (pp. 546–555).

Bergmann, B., & Hommel, G. (1988). Improvements of general multiple test procedures for redundant systems of hypotheses. In *Multiple hypotheses testing* (pp. 100–115). Springer.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research, 13*, 281–305.

Bertini, J. a. R., Jr, Zhao, L., Motta, R., & de Andrade Lopes, A. (2011). A nonparametric classification method based on k-associated graphs. *Information Sciences, 181*, 5435–5456.

Bhattacharyya, S., Jha, S., Tharakunnel, K., & Westland, J. C. (2011). Data mining for credit card fraud: a comparative study. *Decision Support Systems, 50*, 602–613.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*, 123–140.

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. CRC Press.

Carrizosa, E., Martín-Barragán, B., & Morales, D. R. (2011). Detecting relevant variables and interactions in supervised classification. *European Journal of Operational Research, 213*, 260–269.

Carrizosa, E., & Morales, D. R. (2013). Supervised classification and mathematical optimization. *Computers & Operations Research, 40*, 150–165.

Caruana, R., Karampatziakis, N., & Yessenalina, A. (2008). An empirical evaluation of supervised learning in high dimensions. In *Proceedings of the 25th international conference on machine learning* (pp. 96–103).

Caruana, R., & Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on machine learning* (pp. 161–168).

Chandran, B. G., & Hochbaum, D. S. (2012, last updated on Aug, 2012.). HPF: pseudoflow parametric maximum flow solver version 3.23. http://riot.ieor.berkeley.edu/Applications/Pseudoflow/maxflow.html.

Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology, 2*, 27:1–27:27.

CodeNeuro.org (2017). The neurofinder challengeas of March 9, http://neurofinder.codeneuro.org/.

Cooper, G. F., Aliferis, C. F., Ambrosino, R., Aronis, J., Buchanan, B. G., Caruana, R., et al. (1997). An evaluation of machine-learning methods for predicting pneumonia mortality. *Artificial Intelligence in Medicine, 9*, 107–138.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning, 20*, 273–297.

Cupertino, T. H., Zhao, L., & Carneiro, M. G. (2015). Network-based supervised data classification by using an heuristic of ease of access. *Neurocomputing, 149*, 86–92.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems, 2*, 303–314.

De Caigny, A., Coussement, K., & De Bock, K. W. (2018). A new hybrid classification algorithm for customer churn prediction based on logistic regression and decision trees. *European Journal of Operational Research, 269*, 760–772. (in press).

Dembczyński, K., Kotowski, W., & Słowiński, R. (2009). Learning rule ensembles for ordinal classification with monotonicity constraints. *Fundamenta Informaticae, 94*, 163–178.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research, 7*, 1–30.

Duarte Silva, A. P. (2017). Optimization approaches to supervised classification. *European Journal of Operational Research, 261*, 772–788.

Fishbain, B., Hochbaum, D., & Yang, Y. (2013). A new approach for real-time target tracking in videos. *SPIE Newsroom*, 1–3. doi:10.1117/2.1201301.004648.

Fix, E., & Hodges, J. L., Jr (1951). Discriminatory analysis, nonparametric discrimination, consistency properties. *Project 21-49-004, Report No. 4*. Randolph Field, Texas.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences, 55*, 119–139.

Friedman, J., Hastie, T., Tibshirani, R., et al. (2000). Additive logistic regression: a statistical view of boosting. *The Annals of Statistics, 28*, 337–407.

Garcia, S., & Herrera, F. (2008). An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research, 9*, 2677–2694.

Gaudioso, M., Gorgone, E., Labbé, M., & Rodríguez-Chía, A. M. (2017). Lagrangian relaxation for SVM feature selection. *Computers & Operations Research, 87*, 137–145.

Hochbaum, D. S. (2002). Solving integer programs over monotone inequalities in three variables: a framework for half integrality and good approximations. *European Journal of Operational Research, 140*, 291–321.

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

Hochbaum, D. S. (2010). Polynomial time algorithms for ratio regions and a variant of normalized cut. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 32*, 889–898.

Hochbaum, D. S. (2013). A polynomial time algorithm for rayleigh ratio on discrete variables: replacing spectral techniques for expander ratio, normalized cut and cheeger constant. *Operations Research, 61*, 184–198.

Hochbaum, D. S., & Baumann, P. (2014). Sparse computation for large-scale data mining. In *Proceedings of the IEEE international conference on big data* (pp. 354–363).

Hochbaum, D. S., & Baumann, P. (2016). Sparse computation for large-scale data mining. *IEEE Transactions on Big Data, 2*, 151–174.

Hochbaum, D. S., Hsu, C.-N., & Yang, Y. T. (2012). Ranking of multidimensional drug profiling data by fractional-adjusted bi-partitional scores. *Bioinformatics, 28*, i106–i114.

Hochbaum, D. S., Lyu, C., & Bertelli, E. (2013). Evaluating performance of image segmentation criteria and techniques. *EURO Journal on Computational Optimization, 1*, 155–180.
Jia, H., Ding, S., Xu, X., & Nie, R. (2014). The latest research progress on spectral clustering. *Neural Computing and Applications, 24*, 1477–1486.
Kawaji, H., Takenaka, Y., & Matsuda, H. (2004). Graph-based clustering for finding distant relationships in a large set of protein sequences. *Bioinformatics, 20*, 243–252.
King, R. D., Feng, C., & Sutherland, A. (1995). Statlog: comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence: An International Journal, 9*, 289–333.
LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., et al. (1995). Comparison of learning algorithms for handwritten digit recognition. In *Proceedings of the international conference on artificial neural networks* (pp. 53–60).
Lim, T.-S., Loh, W.-Y., & Shih, Y.-S. (2000). A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning, 40*, 203–228.
Murthy, S. K. (1998). Automatic construction of decision trees from data: a multi-disciplinary survey. *Data Mining and Knowledge discovery, 2*, 345–389.
Pedersen, T. (2008). Empiricism is not a matter of faith. *Computational Linguistics, 34*, 465–470.
Perlich, C., Provost, F., & Simonoff, J. S. (2003). Tree induction vs. logistic regression: a learning-curve analysis. *Journal of Machine Learning Research, 4*, 211–255.
Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*, 533–536.
Ryu, Y. U., Chandrasekaran, R., & Jacob, V. (2004). Prognosis using an isotonic prediction technique. *Management Science, 50*, 777–785.
Sharon, E., Galun, M., Sharon, D., Basri, R., & Brandt, A. (2006). Hierarchy and adaptivity in segmenting visual scenes. *Nature, 442*, 810–813.
Shi, J., & Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22*, 888–905.
Sonnenburg, S. A., Braun, M. L., Ong, C. S., Bengio, S., Bottou, L., Holmes, G., et al. (2007). The need for open source software in machine learning. *Journal of Machine Learning Research, 8*, 2443–2466.
Spaen, Q., Hochbaum, D. S., & Asín-Achá, R. (2017). HNCcorr: a novel combinatorial approach for cell identification in calcium-imaging movies. arXiv:1703.01999.
Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B, 58*, 267–288.
Ulaş, A., Yıldız, O. T., & Alpaydın, E. (2012). Cost-conscious comparison of supervised learning algorithms over multiple data sets. *Pattern Recognition, 45*, 1772–1781.
Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing, 17*, 395–416.
Yang, Y. T., Fishbain, B., Hochbaum, D. S., Norman, E. B., & Swanberg, E. (2013). The supervised normalized cut method for detecting, classifying, and identifying special nuclear materials. *INFORMS Journal on Computing, 26*, 1–14.
Zhu, X., Ghahramani, Z., & Lafferty, J. (2003). Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the twentieth international conference on machine learning* (pp. 912–919).