Evol. Intel. (2012) 5:189–205
DOI 10.1007/s12065-012-0068-5

SPECIAL ISSUE

Efficient recurrent local search strategies for semi- and unsupervised regularized least-squares classification

Fabian Gieseke · Oliver Kramer · Antti Airola · Tapio Pahikkala

Received: 10 March 2011 / Revised: 4 November 2011 / Accepted: 21 February 2012 / Published online: 7 March 2012

Abstract Binary classification tasks are among the most important ones in the field of machine learning. One prominent approach to address such tasks are support vector machines which aim at finding a hyperplane separating two classes well such that the induced distance between the hyperplane and the patterns is maximized. In general, sufficient labeled data is needed for such classification settings to obtain reasonable models. However, labeled data is often rare in real-world learning scenarios while unlabeled data can be obtained easily. For this reason, the concept of support vector machines has also been extended to semi- and unsupervised settings: in the unsupervised case, one aims at finding a partition of the data into two classes such that a subsequent application of a support vector machine leads to the best overall result. Similarly, given both a labeled and an unlabeled part, semi-supervised support vector machines favor decision hyperplanes that lie in a low density area induced by the unlabeled training patterns, while still considering the labeled part of the data. The associated optimization problems for both the semi- and unsupervised case, however, are of combinatorial nature and, hence, difficult to solve. In this work, we present efficient implementations of simple local search strategies for (variants of) the both cases that are based on matrix update schemes for the intermediate candidate solutions. We evaluate the performances of the resulting approaches on a variety of artificial and real-world data sets. The results indicate that our approaches can successfully incorporate unlabeled data. (The unsupervised case was originally proposed by Gieseke F, Pahikkala et al. (2009). The derivations presented in this work are new and comprehend the old ones (for the unsupervised setting) as a special case.)

Keywords Machine learning · Semi-supervised support vector machines · Maximum margin clustering · Regularized least-squares classification · Combinatorial optimization · Matrix calculus

F. Gieseke (✉) · O. Kramer
Computer Science Department, Carl von Ossietzky Universität Oldenburg, 26111 Oldenburg, Germany
e-mail: f.gieseke@uni-oldenburg.de

O. Kramer
e-mail: oliver.kramer@uni-oldenburg.de

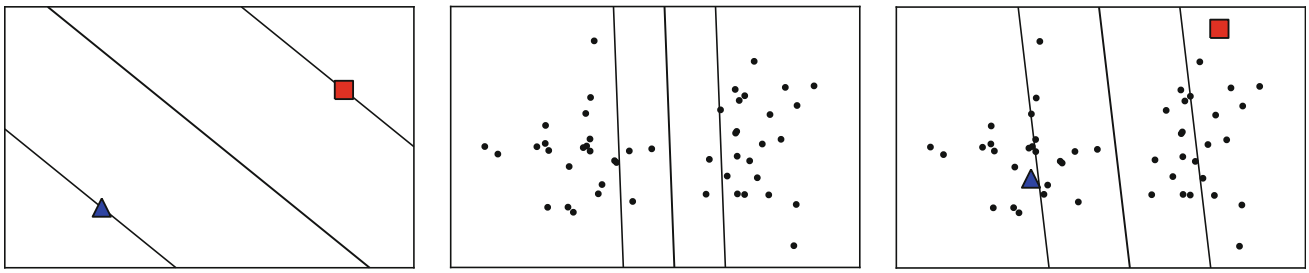
A. Airola · T. Pahikkala
Department of Information Technology,
Turku Centre for Computer Science, University of Turku,
20520 Turku, Finland
e-mail: antti.airola@utu.fi

T. Pahikkala
e-mail: tapio.pahikkala@utu.fi

1 Introduction

One important learning task in the field of machine learning is *binary classification*, where patterns from two classes are given. For instance, the task to automatically classify emails as `spam` or `not spam` belongs to this type of learning problem. In order to achieve a satisfying classification performance, sufficient labeled data is essential in most cases. Such labeled data is often scarce in real-world applications. However, unlabeled data is mostly available in great quantities. In contrast to supervised learning schemes, semi- and unsupervised techniques try to take advantage from these large amounts of (additional) unlabeled data to improve the quality of the final models.

The concepts of *semi-* and *unsupervised support vector machines* [1, 20, 35] are among the most prominent techniques in this field to address binary classification tasks and depict the direct extensions of the well-known concept of



(a) SVM **(b)** MMC **(c)** S^3VM

Fig. 1 In a fully supervised setting, we are only given labeled training patterns (*red squares* and *blue triangles*). Hence, if only a small amount of data is given, the application of a standard support vector machine cannot yield a reasonable model, (**a**). In the unsupervised case, the task is to find the optimal partition of the data into two classes such that a subsequent application of a support vector machine leads to the best result, (**b**). Finally, in the semi-supervised setting, we are given both labeled and unlabeled data. Here, the goal of the learning process consists in finding the correct assignment of the unlabeled patterns such that the induced decision hyperplane passes through the low-density area defined by all patterns

support vector machines (SVMs) [33] to the semi- and unsupervised case: given a set of labeled training patterns, the goal of a standard support vector machine consists in finding a hyperplane separating both classes well such that the distance between the hyperplane and the patterns is large, see Fig. 1a. The latter concept can also be considered in unsupervised learning scenarios. Here, the goal consists in finding the optimal partition of the data into two classes (fulfilling some constraints) such that a subsequent application of a support vector machine leads to the best possible result, see Fig. 1b. Semi-supervised support vector machines can be seen as an intermediate approach between the latter two ones. Given both parts of the data, the aim of the corresponding learning task consists in finding a hyperplane which separates both classes (induced by the labeled part of the data) and, at the same time, passes through a low-density area induced by all patterns, see Fig. 1c. Again, one is interested in the optimal assignment of the unlabeled patterns (to the two classes).[1]

In this work, we present efficient optimization approaches for variants of the original problem definitions. The key idea consists in accelerating simple optimization strategies by means of computational shortcuts for the intermediate solutions making an extensive (recurrent) search possible. In the remainder of this section, we give a formal definition of the learning tasks addressed in this work along with a brief introduction of related evolutionary optimization techniques and the corresponding literature. Afterwards, we formalize the mathematical background of the mentioned variants which we approach in Sect. 3 by means of our optimization strategies. In Sect. 4, we evaluate the benefits of the proposed schemes on several artificial and real-world data sets, followed by conclusions drawn in Sect. 5.

[1] Note that semi-supervised support vector machines do no necessarily lead to better classification models. In general, a low-density area indicating the classification boundaries is required. In the literature, this requirement is called the *cluster assumption* [8, 39].

2 Problem setting

We start by giving a short overview of related evolutionary optimization techniques and by introducing some notations.

2.1 Evolutionary optimization

We consider an *evolutionary algorithm (EA)* [2, 13, 18, 23, 27] as optimization approach which belongs to the class of stochastic optimizers for derivative-free optimization problems. A history of more than forty years of active research on evolutionary computation has proven that stochastic optimization algorithms are powerful search techniques. Inspired by evolutionary processes in nature, evolutionary algorithms optimize by evolving sets of search points until satisfying results are obtained.

The basis of evolutionary search is a *population* $\mathcal{P} := \{\mathbf{y}_1, \ldots, \mathbf{y}_\mu\}$ of candidate solutions, also called *individuals*. The optimization process takes place in three steps: (1) the *recombination operator* selects $\nu \in \mathbb{N}$ parental individuals and combines their parts to new candidate solutions, (2) the mutation operator adds random noise to these possible solutions, and (3) when ν solutions have been produced, the $\mu \in \mathbb{N}$ best-performing individuals are selected to form the new population $\mathcal{P}$ of the subsequent iteration, also called *generation*. The quality of the intermediate individuals is measured in terms of a so-called *fitness function* $F : Y \to \mathbb{R}$. Given the new population, the process starts again until a termination condition is reached. Typical termination conditions are the accomplishment of a certain solution quality or an upper bound on the number of generations. This general type of an evolutionary algorithm is denoted by $(\mu + \nu) - \text{EA}$.

We will also focus on the special case where $\mu = 1$ and $\nu = 1$, i.e., on so-called $(1 + 1) - \text{EAs}$. The latter type of evolutionary algorithms does not use a population of individuals, but produces only one mutated individual in

Algorithm 1 $(1 + 1) - \text{EA}$

1: Initialize $\mathbf{y} \in \{-1, 1\}^n$ uniformly and randomly
2: Generate mutated individual $\mathbf{y}'$ by flipping each coordinate of $\mathbf{y}$ with probability $1/n$
3: Replace $\mathbf{y}$ with $\mathbf{y}'$ if $F(\mathbf{y}') \leq F(\mathbf{y})$
4: Repeat steps 2 and 3 until termination criterion is fulfilled

each generation and selects the better one (of the two intermediate solutions) for the next generation. Usually, the $(1 + 1) - \text{EA}$ starts with a randomly chosen bit string $\mathbf{y}$ of length n and applies a bit flip mutation with probability $1/n$, see Algorithm 1. Its simplicity allows advanced theoretical investigations. In particular, it belongs to the first evolutionary Algorithm for which the optimization time has been analyzed theoretically on various function classes [11].

2.2 Notations

We use $[m]$ to denote the set $\{1, \ldots, m\}$. Further, the set of all $m \times n$ matrices with real coefficients is denoted by $\mathbb{R}^{m \times n}$. Given a matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$, we denote the element in the i-th row and j-th column by $[\mathbf{M}]_{i,j}$. For two sets $R = \{i_1, \ldots, i_r\} \subseteq [m]$ and $S = \{k_1, \ldots, k_s\} \subseteq [n]$ of indices, we use $\mathbf{M}_{R,S}$ to denote the matrix that contains only the rows and columns of $\mathbf{M}$ that are indexed by R and S, respectively. Moreover, we set $\mathbf{M}_{R,[n]} = \mathbf{M}_R$.

Note that the appendix gives a brief overview on matrix calculus that is related to the content presented in this work.

2.3 Classification framework

We will now sketch the mathematical framework of regularized risk minimization and its extension to semi- and unsupervised scenarios. More specifically, we will extend the concept of *regularized least-squares classification* [24] to these scenarios, which will pave the way for efficient implementations of the proposed local search schemes.

2.3.1 Classification scenarios

In binary classification scenarios, we are given a set $T' = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\}$ consisting of training patterns $\mathbf{x}'_i$ belonging to a set X and associated class labels $y'_i \in Y = \{-1, +1\}$. The goal of the corresponding learning task consists in finding a good model (i.e., a prediction function $f : X \to Y$) which *generalizes* well on unseen data, i.e., which is able to predict appropriate labels for new patterns not being in the training set. In unsupervised classification scenarios (i.e., clustering scenarios), we are only given a set $T = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\} \subset X$ of unlabeled training patterns. One is then interested in detecting *clusters*, i.e., in partitioning the data into groups each containing similar patterns. Finally, semi-supervised learning scenarios lie in between these two settings. Similar to the supervised case, the goal consists in finding a model which can predict meaningful labels for unseen patterns. However, in the training phase, one is given both a labeled set T' and an unlabeled set T and the aim consists in deriving a good model based on both parts of the data.

2.3.2 Regularized risk minimization

Let $\mathcal{L} : Y \times \mathbb{R} \to [0, \infty)$ denote a *loss function* measuring the performance of some prediction function f. The goal of supervised learning is to achieve minimal *expected risk*, also known as *generalization error*

$$\int_{X \times Y} \mathcal{L}(y, f(\mathbf{x})) P(\mathbf{x}, y) d\mathbf{x} dy,$$

where $P(\mathbf{x}, y)$ is the unkown (but fixed) probability distribution generating the data. Since the distribution is unknown, we are rather limited using an empirical estimate of the risk computed on a training set sampled from the distribution. In this case regularization techniques are commonly applied in order to avoid overfitting to the training set.

Support vector machines can be seen as a special case of regularization problems of the form

$$\inf_{f \in \mathcal{H}} \frac{1}{l} \sum_{i=1}^{l} \mathcal{L}\left(y'_i, f(\mathbf{x}'_i)\right) + \lambda ||f||^2_{\mathcal{H}}, \tag{1}$$

where $\lambda > 0$ is a regularization parameter, and $||f||^2_{\mathcal{H}}$ the squared norm in a *reproducing kernel Hilbert space* (RKHS) $\mathcal{H} \subseteq \mathbb{R}^X = \{f : X \to \mathbb{R}\}$ induced by a kernel function $k : X \times X \to \mathbb{R}$ [12, 30].[2] With a kernel, we refer to a symmetric function that satisfies the Mercer's condition (see e.g. [30]) on X, indicating that, for any set of points $\{\mathbf{x}_1, \ldots, \mathbf{x}_l\} \subset X$, the matrix $\mathbf{K} \in \mathbb{R}^{l \times l}$, whose entries are defined as $[\mathbf{K}]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$, is positive semidefinite. Further, for a Mercer kernel k, there exists a feature space $\mathcal{F}$ and a feature mapping $\phi : X \to \mathcal{F}$ such that $k(\mathbf{x}, \mathbf{x}')$ is the inner product of $\Phi(\mathbf{x})$ and $\phi(\mathbf{x}')$ in $\mathcal{F}$.

For binary classification problems, the most natural loss function is the so-called *zero-one loss* $\mathcal{L}(y, t) = \mathbf{1}_{\{\text{sgn}(t) \neq y\}}$ with $y \in \{-1, +1\}$. This loss function has value 0

[2] For the sake of simplicity, the offset set $b \in \mathbb{R}$ is omitted in the latter formulation. From both a theoretical as well as a practical point of view, the additional term does not yield any known advantages for kernel functions like the RBF kernel [25, 30]. However, for the linear kernel, the offset term can make a difference since it addresses translated data. In case such an offset effect is needed for a particular learning task, one can add a dimension of ones to the input data to obtain a (regularized) offset term [25].

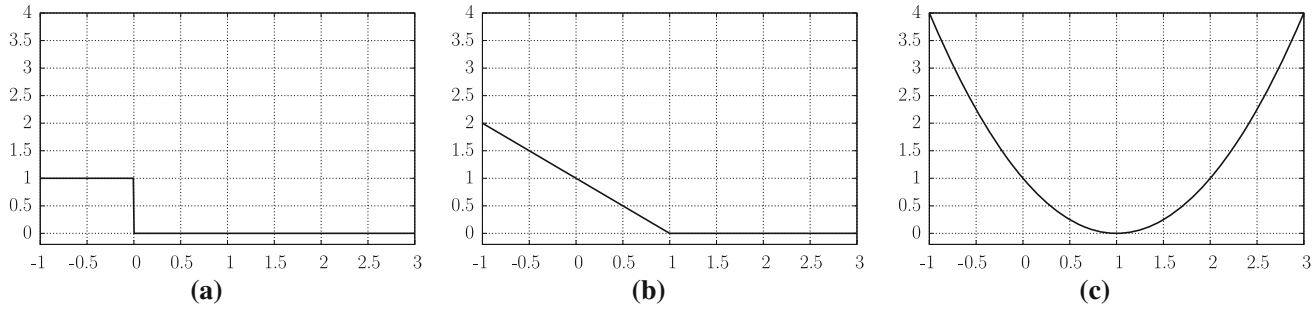


Fig. 2 A natural loss function is the zero-one loss defined as $\mathcal{L}(y,t) = \mathbf{1}_{\{\mathrm{sgn}(t)\neq y\}}$, see Figure (**a**) for the case of $y = 1$. However, using this loss function leads to intractable optimization problems. A well-known surrogate loss function for binary classification tasks is the hinge loss $\mathcal{L}(y,t) = \max(0, 1 - yt)$ that leads to standard support vector machines, see Figure (**b**). The square loss $\mathcal{L}(y,t) = (y-t)^2$, depicted in Figure (**c**), is another well-known loss function and leads to the concept of least-squares support vector machines

whenever the prediction is correct ($yt \geq 0$), and value 1 when an incorrect prediction is made ($yt < 0$), see Fig. 2a. However, it is well-known that its use in the context of the above task leads to intractable optimization problems. Hence classification methods usually resort to convex and continuous approximations of this loss function instead.

A well-known surrogate function for binary classification tasks with $Y = \{-1, +1\}$ is the *hinge loss* $\mathcal{L}(y,t) = \max(0, 1 - yt)$, which leads to the optimization criterion induced by standard support vector machines [26, 30], see Fig. 2b. In this work, we will focus on another well-known loss function in this context, the *square loss* $\mathcal{L}(y,t) = (y-t)^2$ shown in Fig. 2c. By inserting this loss function into the above minimization problem (1), one obtains

$$\inf_{f\in\mathcal{H}} \frac{1}{l}\sum_{i=1}^{l} \left(y_i' - f(\mathbf{x}_i')\right)^2 + \lambda||f||_{\mathcal{H}}^2, \tag{2}$$

which leads to the so-called *least-squares support vector machines* [31], also known as *regularized least-squares classification* [24]. Due to the *representer theorem* [26], any minimizer $f^* \in \mathcal{H}$ of the task (2) has the form

$$f^*(\cdot) = \sum_{i=1}^{l} c_i k(\mathbf{x}_i', \cdot) \tag{3}$$

with appropriate coefficients $\mathbf{c} = (c_1, \ldots, c_l)^{\mathrm{T}} \in \mathbb{R}^l$. Hence, by using $||f^*||_{\mathcal{H}}^2 = \mathbf{c}^{\mathrm{T}}\mathbf{K}\mathbf{c}$ [26], where $\mathbf{K} \in \mathbb{R}^{l\times l}$ is the symmetric kernel matrix with entries of the form $[\mathbf{K}]_{ij} = k(\mathbf{x}_i', \mathbf{x}_j')$, one can rewrite the problem (2) as

$$\underset{\mathbf{c}\in\mathbb{R}^l}{\text{minimize}} \frac{1}{l}(\mathbf{y}' - \mathbf{K}\mathbf{c})^{\mathrm{T}}(\mathbf{y}' - \mathbf{K}\mathbf{c}) + \lambda\mathbf{c}^{\mathrm{T}}\mathbf{K}\mathbf{c}, \tag{4}$$

where $\mathbf{y}' = (y_1', \ldots, y_l')^{\mathrm{T}}$. The optimization problem is convex and, thus, easy to solve given standard techniques for convex optimization [4, 24].

2.3.3 *Unsupervised learning settings*

As mentioned above, support vector machines can be extended to unsupervised learning settings as well. One of these extensions, known as *maximum margin clustering* [35], leads to a combinatorial problem. Briefly put, the idea is to search for an optimal partition of the unlabeled training patterns into two classes such that a subsequent application of a support vector machine (or one of its variants) yields the best overall result. From a regularization perspective, one searches for a function $f^* \in \mathcal{H}$ and a vector $\mathbf{y}^* \in \{-1, +1\}^u$ for the unlabeled patterns which are optimal with respect to

$$\begin{aligned} \underset{f\in\mathcal{H},\mathbf{y}\in\{-1,+1\}^u}{\text{minimize}} \; & \frac{1}{u}\sum_{i=1}^{u} \mathcal{L}(y_i, f(\mathbf{x}_i)) + \lambda||f||_{\mathcal{H}}^2 \\ \text{s.t.} \; & \left|\frac{1}{u}\sum_{i=1}^{u} \max(0, y_i) - b_c\right| < \varepsilon, \end{aligned} \tag{5}$$

where $\lambda > 0, b_c \in [0,1]$, and $\varepsilon > 0$ are user-defined parameters. By applying the representer theorem [26] to the task (5) for a *fixed* partition vector $\mathbf{y} \in \{-1, +1\}^u$, the corresponding optimal function $f^* \in \mathcal{H}$ can be shown to have the form

$$f^*(\cdot) = \sum_{i=1}^{u} c_i k(\mathbf{x}_i, \cdot) \tag{6}$$

with appropriate coefficients $\mathbf{c} = (c_1, \ldots, c_u)^{\mathrm{T}} \in \mathbb{R}^u$. Plugging in different loss functions leads to various (related) optimization tasks; using the hinge loss $\mathcal{L}(y,t) = \max(0, 1 - yt)$ yields the original optimization task [35]. By using the square loss $\mathcal{L}(y,t) = (y-t)^2$, one can reformulate the above optimization problem as

$$\begin{aligned}\underset{\mathbf{c}\in\mathbb{R}^u,\mathbf{y}\in\{-1,+1\}^u}{\text{minimize}}\ & J_1(\mathbf{c},\mathbf{y}) = (\mathbf{Dy}-\mathbf{DKc})^{\mathrm{T}}(\mathbf{Dy}-\mathbf{DKc}) \\ & \quad + \lambda \mathbf{c}^{\mathrm{T}}\mathbf{Kc} \\ \text{s.t.}\ & \left|\frac{1}{u}\sum_{i=1}^{u}\max(0,y_i) - b_c\right| < \varepsilon,\end{aligned} \tag{7}$$

where $\mathbf{K}$ is the kernel matrix (based on $\mathbf{x}_1,\ldots,\mathbf{x}_u$) and where $\mathbf{D}$ a diagonal matrix with entries of the form $[\mathbf{D}]_{i,i} = \sqrt{\frac{1}{u}}$ for $i = 1,\ldots,u$.

2.3.4 Semi-supervised learning settings

The goal of the semi-supervised learning process is to find an optimal prediction function for unseen data based on both the labeled and the unlabeled part of the data. More precisely, we search for a function $f^* \in \mathcal{H}$ and a labeling vector $\mathbf{y}^* = (y_1^*,\ldots,y_u^*)^{\mathrm{T}} \in \{-1,+1\}^u$ for the unlabeled training patterns which are optimal with respect to

$$\begin{aligned}\underset{f\in\mathcal{H},\mathbf{y}\in\{-1,+1\}^u}{\text{minimize}}\ & \frac{1}{l}\sum_{i=1}^{l}\mathcal{L}\big(y_i',f(\mathbf{x}_i')\big) \\ & \quad + \lambda'\frac{1}{u}\sum_{i=1}^{u}\mathcal{L}(y_i,f(\mathbf{x}_i)) + \lambda||f||_{\mathcal{H}}^2 \\ \text{s.t.}\ & \left|\frac{1}{u}\sum_{i=1}^{u}\max(0,y_i) - b_c\right| < \varepsilon,\end{aligned} \tag{8}$$

where $\lambda', \lambda > 0$ are cost parameters. Exactly as for the unsupervised case, the partition vector $\mathbf{y}$ has to fulfill a balance constraint (in the semi-supervised case, the parameter b_c can be estimated based on the labeled part of the data). Hence, compared to the unsupervised case, the labeled part of the data is also taken into account. Again, by applying the representer theorem [26] for the task (8) given a *fixed* partition vector $\mathbf{y} \in \{-1,+1\}^u$, the corresponding optimal function $f^* \in \mathcal{H}$ can be shown to have the form

$$f^*(\cdot) = \sum_{i=1}^{l} c_i k(\mathbf{x}_i',\cdot) + \sum_{i=l+1}^{l+u} c_i k(\mathbf{x}_{i-l},\cdot) \tag{9}$$

with appropriate coefficients $\mathbf{c} = (c_1,\ldots,c_{l+u})^{\mathrm{T}} \in \mathbb{R}^{l+u}$. As above, plugging in different loss functions yields different optimization tasks; again, using the square loss leads to

$$\begin{aligned}\underset{\mathbf{c}\in\mathbb{R}^n,\mathbf{y}\in\{-1,+1\}^n}{\text{minimize}}\ & J_2(\mathbf{c},\mathbf{y}) = (\mathbf{Dy}-\mathbf{DKc})^{\mathrm{T}}(\mathbf{Dy}-\mathbf{DKc}) \\ & \quad + \lambda \mathbf{c}^{\mathrm{T}}\mathbf{Kc} \\ \text{s.t.}\ & \left|\frac{1}{u}\sum_{i=l+1}^{n}\max(0,y_i) - b_c\right| < \varepsilon, \\ & \text{and } y_i = y_i' \text{ for } i = 1,\ldots,l,\end{aligned} \tag{10}$$

where $\mathbf{K}$ is the kernel matrix (induced by $\mathbf{x}_1',\ldots,\mathbf{x}_l'$ and $\mathbf{x}_1,\ldots,\mathbf{x}_u$), $\mathbf{D}$ a diagonal matrix with entries $[\mathbf{D}]_{i,i} = \sqrt{\frac{1}{l}}$ for $i = 1,\ldots,l$, and $[\mathbf{D}]_{i,i} = \sqrt{\frac{\lambda'}{u}}$ for $i = l+1,\ldots,n$, and where $n = l+u$.

2.4 Related work

Unlike the supervised case, the optimization problems induced by the unsupervised and the semi-supervised criteria are not solvable by convex optimization, but rather require a combinatorial search. A trivial approach to solving the induced optimization tasks consists in testing every feasible assignment of the vector $\mathbf{y}$ and to report the best result found during the overall process. Of course, this brute-force approach is only possible for a very small amount of unlabeled data due to its exponential runtime. However, since the learning tasks are very appealing from a practical point of view, several heuristics addressing the optimization tasks have been proposed in the literature (which can deal with larger data sets at the cost of possibly suboptimal solutions). In the remainder of this section, we sketch the key ideas of some of these approaches.

2.4.1 Maximum margin clustering

The first attempts at coping with the combinatorial nature of the optimization task induced by the unsupervised learning setting consisted in relaxing the definition to form semidefinite programming problems [32, 35]. However, these approaches resort to solving semidefinite programming problems which is computationally expensive. The approach proposed by [36] aims at practical runtimes such that one can deal with large data sets. One of their key insights consisted in the replacement of the hinge loss by several other loss functions including the square loss. The cutting plane algorithm is one of the most recent techniques [37]. It is based on constructing a sequence of successively tighter relaxations of the problem statement and each of the intermediate tasks is solved using the so-called constrained concave-convex procedure. In addition to these methods, extensions to multiclass scenarios have been proposed in the literature [34, 38].

2.4.2 Semi-supervised support vector machines

One of the first heuristics was proposed by [20]; the key idea of his approach was to first classify all unlabeled patterns via a standard support vector machine (trained on the labeled part of the data) and to subsequently improve the objective value by a simple label switching strategy. In recent years, a variety of techniques has been proposed to

tackle the problem. Among these techniques are concave-convex procedures [10, 14], gradient descent approaches [6], semidefinite programming [3, 34], deterministic annealing [29], and various other methods [1, 7]. In addition, the problem has recently gained attention in the field of evolutionary computation [21, 28]. Due to lack of space we refer the reader to corresponding surveys [9] and books [8, 39] for an overview.

We would like to point out that most of the related semi-supervised schemes improve a single initial guess obtained via training a supervised model on the labeled part of the data [1, 6, 7, 10, 20]. This renders such approaches deterministic and, hence, no restarts are required. On the one hand, this sounds tempting since the computational cost is normally reduced dramatically by this modification. On the other hand, considering only a single candidate solution might not be sufficient for obtaining satisfying results, especially if the initial guess is bad due to a very small set of labeled patterns. In this work, we present a stochastic optimization framework that can test a huge amount of possible candidate solutions. Naturally, the approach can also be adapted to yield a comparable single-restart scheme. Still, in the latter case, the execution of this single run is accelerated significantly by means of the presented computational shortcuts (depicted below).

3 Efficient optimization approach

Computing solutions for the optimization tasks (7) and (10) is difficult since the optimal partition vectors are not known beforehand. In this section, we describe our local search strategy to address both tasks. Since the unsupervised case can be seen as a special case of the semi-supervised one, we will focus on the description of the semi-supervised case and will only sketch the derivations for the unsupervised setting.

3.1 Dealing with the semi-supervised case

We resort to the following simple evolutionary optimization strategy depicted in Algorithm 2: Starting with an initial population $\mathcal{P}_0 = \{\mathbf{y}_1, \ldots, \mathbf{y}_\mu\} \subseteq \{-1, +1\}^n$ consisting of randomly generated candidate solutions, we iterate over a sequence of τ generations.[3] In each generation, we generate ν new (mutated) individuals, compute their fitness values, and take the best performing candidates out of the set $\{\mathbf{y}_1, \ldots, \mathbf{y}_{\mu+\nu}\}$ of intermediate solutions; mutations of a (parental) individuals consist in flipping single coordinates. The resulting population obtained in each generation is then used as initial population for the next generation. To compute the fitness values for each candidate solution, we use the following fitness function:

$$F(\mathbf{y}) = \min_{\mathbf{c} \in \mathbb{R}^n} J_2(\mathbf{c}, \mathbf{y}), \tag{11}$$

where J_2 is the objective function given in (10). Once the overall process is finished, the best individual along with its corresponding vector $\mathbf{c}^*$ is returned. Throughout the execution of the algorithm, it is ensured that only valid candidate solutions are generated, i.e., partition vectors fulfilling the corresponding constraints.

Algorithm 2 $(\mu + \nu)$ − EA

Require A set of training patterns $T' = \{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\}$ with associated class labels, a set of training patterns $T = \{\mathbf{x}_1, \ldots, \mathbf{x}_u\}$ without class labels, model parameters $\lambda', \lambda, r, \varepsilon$ and optimization parameters μ, ν, τ.

Ensure An approximation $(\mathbf{c}^*, \mathbf{y})$ for the problem (10).

Initialize $\mathcal{P}_0 = \{\mathbf{y}_1, \ldots, \mathbf{y}_\mu\} \subseteq \{-1, +1\}^n$
Compute the fitness $F(\mathbf{y}_j)$ for each $\mathbf{y}_j \in \mathcal{P}_0$
$t = 0$
While $t < \tau$ **do**
  for $i = 1$ **to** ν **do**
    Randomly select parent $\mathbf{y} \in \mathcal{P}_t$
    Generate valid mutated individual $\mathbf{y}_{\mu+i}$
    Compute fitness $F(\mathbf{y}_{\mu+i})$
  end for
  Compute sorted sequence $\mathbf{y}_{i_1}, \ldots, \mathbf{y}_{i_{\mu+\nu}}$
  $\mathcal{P}_{t+1} = \{\mathbf{y}_{i_1}, \ldots, \mathbf{y}_{i_\mu}\}$
  $t = t + 1$
end while
Compute $\mathbf{c}^*$ for $\text{minimize}_{\mathbf{c} \in \mathbb{R}^n} J_2(\mathbf{c}, \mathbf{y}_{i_1})$
return $(\mathbf{c}^*, y_{i_1})$

3.1.1 Fitness computations ...

The task (11) for fixed vector $\mathbf{y}$ can be solved as follows: The function $H(\mathbf{c}) = J_2(\mathbf{c}, \mathbf{y})$ is differentiable with

$$\nabla H(\mathbf{c}) = -2(\mathbf{DK})^{\mathrm{T}}(\mathbf{Dy} - \mathbf{DKc}) + 2\lambda \mathbf{Kc}.$$

Further, H is convex since $\mathbf{K}$ and thus the Hessian

$$\nabla^2 H(\mathbf{c}) = 2(\mathbf{DK})^{\mathrm{T}}\mathbf{DK} + 2\lambda \mathbf{K}$$

is positive semidefinite. Hence, $\nabla H(\mathbf{c}) = \mathbf{0}$ is a necessary and sufficient condition for optimality [4]. An optimal solution $\mathbf{c}^*$ for the task (11) can therefore be obtained via

$$\mathbf{c}^* = \mathbf{D}(\mathbf{DKD} + \lambda \mathbf{I})^{-1}\mathbf{Dy} = \mathbf{DGDy}. \tag{12}$$

[3] The random generation of an initial candidate solution takes the class ratio given by the balance constraint into account, i.e., for an initial candidate solution $\mathbf{y}$ we have $y_i = 1$ with probability b_c and $y_i = -1$ with probability $1 - b_c$ for $i = l + 1, \ldots, n$.

with $\mathbf{G} = (\mathbf{D}\,\mathbf{K}\mathbf{D} + \lambda\,\mathbf{I})^{-1}$.[4] Hence, once the matrix $\mathbf{G}$ is computed and stored in memory, one can compute the optimal $\mathbf{c}^*$ for each (modified) partition vector $\mathbf{y}$ and the resulting optimal value $F(\mathbf{y}) = J_2(\mathbf{c}^*,\mathbf{y})$ in $\mathcal{O}(n^2)$ time by performing simple matrix calculations.

3.1.2 ... and how to speed them up

The recurrent computation of the intermediate fitness values $F(\mathbf{y}_{\mu+i})$ in Step 8 of Algorithm 2 is still cumbersome, especially if a lot of reruns are performed to tackle the problem of bad local optima (see Sect. 4). However, it is possible to update these intermediate solutions more efficiently when only a small number of coordinates is flipped per mutation.

3.1.2.1 Coordinate flips revisited For simplicity, we assume that only one coordinate is flipped per mutation. In this case, the computation of $F(\mathbf{y}_{\mu+i})$ in Step 8 can be accelerated by making use of the corresponding result for the predecessor. Various update schemes are possible. In the following, we present a closed-form update scheme which permits a direct application of an additional approximation scheme: Let $\bar{\mathbf{y}} = \mathbf{y}_{\mu+i}$ and let $\mathbf{y}$ denote its predecessor. By plugging Eq. (12) into (10), one gets

$$\begin{aligned}
F(\bar{\mathbf{y}}) &= (\mathbf{D}\bar{\mathbf{y}} - \mathbf{D}\mathbf{K}\mathbf{c}^*)^{\mathrm{T}}(\mathbf{D}\bar{\mathbf{y}} - \mathbf{D}\mathbf{K}\mathbf{c}^*) + \lambda(\mathbf{c}^*)^{\mathrm{T}}\mathbf{K}\mathbf{c}^* \\
&= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{D}\big(\mathbf{I} - \overline{\mathbf{K}}\mathbf{G} - \mathbf{G}\overline{\mathbf{K}} + \mathbf{G}\overline{\mathbf{K}}\overline{\mathbf{K}}\mathbf{G} + \lambda\mathbf{G}\overline{\mathbf{K}}\mathbf{G}\big)\mathbf{D}\bar{\mathbf{y}} \\
&= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{D}\mathbf{V}\Big(\mathbf{I} - 2\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}} + \boldsymbol{\Lambda}^2\widetilde{\boldsymbol{\Lambda}}^2 + \lambda\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}^2\Big)\mathbf{V}^{\mathrm{T}}\mathbf{D}\bar{\mathbf{y}} \\
&= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{D}\mathbf{V}\Big(\mathbf{I} + (-2\mathbf{I} + \widetilde{\boldsymbol{\Lambda}}\boldsymbol{\Lambda} + \lambda\widetilde{\boldsymbol{\Lambda}})\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}\Big)\mathbf{V}^{\mathrm{T}}\mathbf{D}\bar{\mathbf{y}} \\
&= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{D}\mathbf{V}\Big(\mathbf{I} + (-2\mathbf{I} + \widetilde{\boldsymbol{\Lambda}}(\boldsymbol{\Lambda} + \lambda\mathbf{I}))\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}\Big)\mathbf{V}^{\mathrm{T}}\mathbf{D}\bar{\mathbf{y}} \\
&= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{D}\mathbf{V}\Big(\mathbf{I} + (-2\mathbf{I} + \mathbf{I})\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}\Big)\mathbf{V}^{\mathrm{T}}\mathbf{D}\bar{\mathbf{y}} \\
&= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{D}\mathbf{V}\Big(\mathbf{I} - \boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}\Big)\mathbf{V}^{\mathrm{T}}\mathbf{D}\bar{\mathbf{y}} \\
&= 1 + \lambda' - \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{D}\mathbf{V}\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}\mathbf{V}^{\mathrm{T}}\mathbf{D}\bar{\mathbf{y}}, \qquad (13)
\end{aligned}$$

where $\overline{\mathbf{K}} = \mathbf{D}\mathbf{K}\mathbf{D}, \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}}$ is the eigendecomposition (see the Appendices 1 and 2) of $\overline{\mathbf{K}}$, and where $\widetilde{\boldsymbol{\Lambda}} = (\boldsymbol{\Lambda} + \lambda\mathbf{I})^{-1}$. Hence, given the vector $\bar{\mathbf{y}}^{\mathrm{T}}\mathbf{D}\mathbf{V} \in \mathbb{R}^n$, one can compute $F(\bar{\mathbf{y}})$ in $\mathcal{O}(n)$ time since $\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}$ is a diagonal matrix. Further, the latter auxiliary information can be updated efficiently for the successor $\bar{\mathbf{y}}$ via

$$\bar{\mathbf{y}}^{\mathrm{T}}\mathbf{D}\mathbf{V} = \mathbf{y}^{\mathrm{T}}\mathbf{D}\mathbf{V} - 2y_j(\mathbf{D}\mathbf{V})_{\{j\}}$$

in $\mathcal{O}(n)$ runtime (where j denotes the flipped coordinate). To sum up, one can update the fitness values spending $\mathcal{O}(n)$ time if a single (or a constant amount) of coordinates is flipped per mutation. The eigendecomposition of the matrix can be obtained in $\mathcal{O}(n^3)$ and the auxiliary information for all μ initial candidate solutions in $\mathcal{O}(\mu n^2)$ runtime (in practice and up to machine precision). Hence, since a *large* amount of single flips are performed during the (recurrent) execution, the computational shortcut yields a significant runtime reduction making an extensive search possible.[5] Similar to the previous results obtained for the unsupervised case [15], we have shown the following theorem:

Theorem 1 *By spending $\mathcal{O}(n^3 + \mu n^2)$ time in the preprocessing phase, each fitness evaluation $F(\mathbf{y}_{\mu+i})$ in Step 8 of Algorithm 2 can be performed in $\mathcal{O}(n)$ time. The storage requirement is $\mathcal{O}(n^2)$.*

The above shortcut yields a significant speed-up for the overall execution of the stochastic search. However, when dealing with a large amount of unlabeled data, two bottlenecks arise, namely the cubic preprocessing time and the quadratic space consumption for storing the matrices. We will now sketch a way to shorten both drawbacks.

3.1.2.2 Kernel matrix approximation A common tool for decreasing these computational costs is the so-called *Nyström approximation*, which is based on replacing the kernel matrix $\mathbf{K} \in \mathbb{R}^{n\times n}$ by the following matrix:

$$\widetilde{\mathbf{K}} = (\mathbf{K}_R)^{\mathrm{T}}(\mathbf{K}_{R,R})^{-1}\mathbf{K}_R \in \mathbb{R}^{r\times r} \qquad (14)$$

Here, the set $R = \{i_1, \ldots, i_r\} \subset \{1, \ldots, n\}$ of indices determines the sub-columns to be selected of the original kernel matrix, see, e.g., [24]. This scheme reduces both the preprocessing time as well as the space consumption. Surprisingly, it can also be incorporated into the above framework in an efficient manner leading to an update time of $\mathcal{O}(r)$ for each fitness evaluation:

Theorem 2 *By spending $\mathcal{O}(nr^2 + \mu nr)$ time in the preprocessing phase, each fitness evaluation $F(\mathbf{y}_{\mu+i})$ in*

[4] If $\mathbf{K}$ is invertible, then

$$\begin{aligned}
&-2(\mathbf{D}\mathbf{K})^{\mathrm{T}}(\mathbf{D}\mathbf{y} - \mathbf{D}\mathbf{K}\mathbf{c}) + 2\lambda\mathbf{K}\mathbf{c} = \mathbf{0} \\
&\quad\Leftrightarrow (\mathbf{D}\mathbf{K})^{\mathrm{T}}(\mathbf{D}\mathbf{K}\mathbf{D} + \lambda\mathbf{I})\mathbf{D}^{-1}\mathbf{c} = (\mathbf{D}\mathbf{K})^{\mathrm{T}}\mathbf{D}\mathbf{y} \\
&\qquad\qquad\Leftrightarrow \mathbf{c} = \mathbf{D}\mathbf{G}\mathbf{D}\mathbf{y}
\end{aligned}$$

If $\mathbf{K}$ is not invertible, then the latter equation can be used as well since we only need a single solution (if $\mathbf{c} = \mathbf{D}\,\mathbf{G}\,\mathbf{D}\,\mathbf{y}$, then $(\mathbf{D}\,\mathbf{K})^{\mathrm{T}}\,\mathbf{G}^{-1}\,\mathbf{D}^{-1}\mathbf{c} = (\mathbf{D}\,\mathbf{K})^{\mathrm{T}}\,\mathbf{D}\,\mathbf{y}$ holds as well).

[5] As mentioned, various other update schemes are possible. Another update scheme, for instance, consists in updating the terms in (10) individually, i.e., to handle the first term by updating the vector $\mathbf{D}\mathbf{K}\mathbf{c}^* = \mathbf{D}\mathbf{K}\mathbf{D}\mathbf{G}\mathbf{D}\mathbf{y}$ and therefore $(\mathbf{D}\mathbf{y} - \mathbf{D}\mathbf{K}\,\mathbf{c}^*)^{\mathrm{T}}\,(\mathbf{D}\mathbf{y} - \mathbf{D}\mathbf{K}\,\mathbf{c}^*)$ in linear time. Similarly, the second term can be handled in linear time (by first updating $\mathbf{K}\mathbf{c}^* = \mathbf{K}\mathbf{D}\mathbf{G}\mathbf{D}\mathbf{y}$ separately). Note that one can also consider one of the predecessors of Eq. (13).

Table 1 Data sets

Data set	n	d	Comment
`Gaussian2C`	500	500	Artificial data set
`Gaussian4C`	500	500	Artificial data set
`Moons`	200	2	Artificial data set
`COIL (3, 6)`	144	400	Reduced dimension, rescaled pixel values
`COIL (5, 9)`	144	400	Reduced dimension, rescaled pixel values
`COIL (6, 19)`	144	400	Reduced dimension, rescaled pixel values
`COIL (18, 19)`	144	400	Reduced dimension, rescaled pixel values
`USPS (2, 5)`	1,645	256	Rescaled pixel values
`USPS (2, 7)`	1,721	256	Rescaled pixel values
`USPS (3, 8)`	1,532	256	Rescaled pixel values
`USPS (8, 0)`	2,261	256	Rescaled pixel values
`MNIST (1, 7)`	1,000	784	Rescaled pixel values
`MNIST (2, 5)`	1,000	784	Rescaled pixel values
`MNIST (2, 7)`	1,000	784	Rescaled pixel values
`MNIST (3, 8)`	1,000	784	rescaled pixel values

Step 8 *of Algorithm* 2 *can be performed in* $\mathcal{O}(r)$ *time. The storage requirement is* $\mathcal{O}(nr)$.

For the sake of exposition, we defer the proof to the appendix. Note that the parameter r determines a trade-off between computational savings and accuracy of the approximation. As pointed out by [25], a small amount of randomly selected indices works well in practice.[6]

3.2 Dealing with the unsupervised case

The unsupervised and the semi-supervised essentially vary only in the definition of the diagonal matrix $\mathbf{D}$. In principle, we can make use of the same optimization approach for the unsupervised case as for the semi-supervised one. The main modification to be made consists in replacing the fitness function by

$$F(\mathbf{y}) = \min_{\mathbf{c}\in\mathbb{R}^u} J_1(\mathbf{c}, \mathbf{y}). \tag{15}$$

The remaining (matrix) derivations are basically the same. For the computational shortcuts, however, the additional λ' term vanishes, i.e., we end up with

$$F(\bar{\mathbf{y}}) = 1 - \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{D}\mathbf{V}\boldsymbol{\Lambda}\widetilde{\boldsymbol{\Lambda}}\mathbf{V}^{\mathrm{T}}\mathbf{D}\bar{\mathbf{y}}. \tag{16}$$

Thus, the derivations depicted above encompass the previous results [15] as a special case.

4 Experimental analysis

We conduct several experiments to assess the performance of the proposed framework. In the remainder of this section, we describe the experimental setup and the outcome of the experimental evaluation. Since both the semi-supervised and the unsupervised settings are very similar to each other, we will again focus on the semi-supervised case. The reason for this setup is the fact that we can resort to the true labels (in the test set) to evaluate the final performance of the competing approaches (which is not possible/more complicated in real-world unsupervised settings).

4.1 Experimental setup

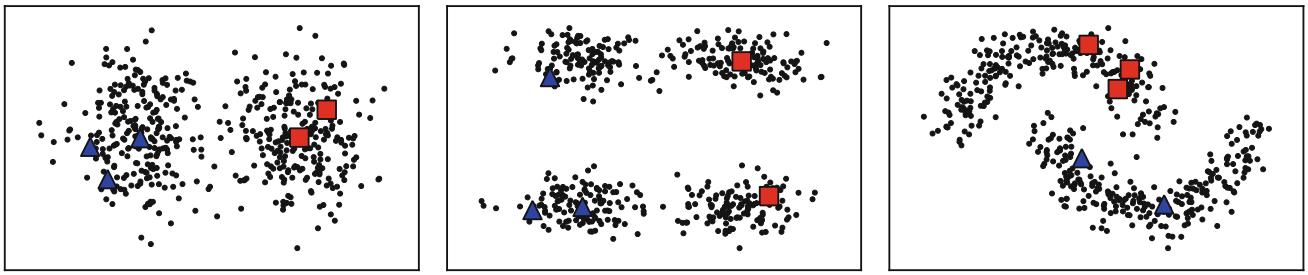
We start by describing the experimental setup including details related to the considered data sets and to the implementation of the proposed framework.

4.1.1 Data sets

For our evaluation, we resort to several artificial and real-world data sets, see Table 1 for an overview.

4.1.1.1 Artificial data sets The first artificial data set is composed of two Gaussian clusters; to generate it, $n/2$ points are drawn from each of two multivariate Gaussian distributions $X_i \sim \mathcal{N}(\mathbf{m}_i, \mathbf{I})$, where $\mathbf{m}_1 = (-2.5, 0.0, \ldots, 0.0) \in \mathbb{R}^d$ and $\mathbf{m}_2 = (+2.5, 0.0, \ldots, 0.0) \in \mathbb{R}^d$. The class label of a point corresponds to the distribution it was drawn from, see Fig. 3a. If not noted otherwise, we use $n = 500$ and $d = 500$ and denote the induced data set by `Gaussian2C`. The second artificial data set aims at generating a (possibly) misleading structure: Here, $n/4$ points are drawn from each of four multivariate Gaussian distributions $X_i \sim \mathcal{N}(\mathbf{m}_i, \mathbf{I})$, where

[6] Naturally, in case a too small subset is selected, the performance of the final model can be bad.



(a) Gaussian2C **(b)** Gaussian4C **(c)** Moons

Fig. 3 Distribution of all artificial data sets ($d = 2$). The *red squares* and *blue triangles* depict the labeled part of the data; the remaining blackpoints correspond to the unlabeled part. Note that the two Gaussian data sets depict easy learning instances for $d = 2$, even given only fewlabeled patterns. However, the noise present in the data render the induced tasks difficult to approach in high dimensions ($d = 500$) in case onlyfew labeled patterns are given (for supervised models). Further, the Gaussian4C data set exhibits a misleading structure for semi-supervisedlearning schemes since the pure clustering solution (*horizontal* separation) differs from the pure classification solution (*vertical* separation). Hence,local search schemes can easily get stuck in the wrong local optimum corresponding to the pure clustering solution

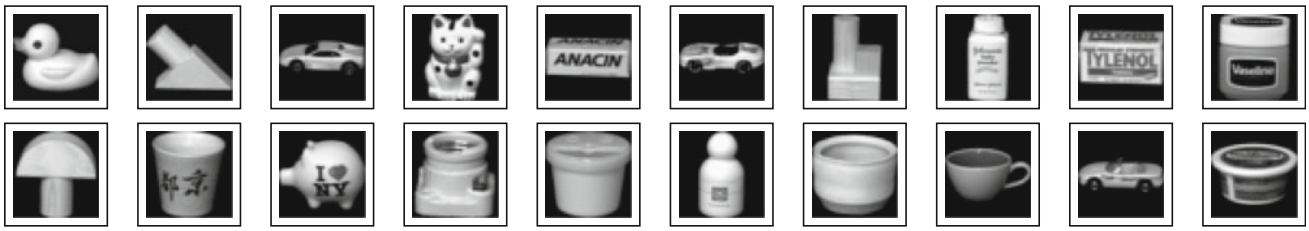


Fig. 4 COIL Data Set

Fig. 5 USPS Data Set

$$\mathbf{m}_1 = (-2.5, -5.0, 0.0, \ldots, 0.0) \in \mathbb{R}^d,$$
$$\mathbf{m}_2 = (-2.5, +5.0, 0.0, \ldots, 0.0) \in \mathbb{R}^d,$$
$$\mathbf{m}_3 = (+2.5, -5.0, 0.0, \ldots, 0.0) \in \mathbb{R}^d,$$
$$\mathbf{m}_4 = (+2.5, +5.0, 0.0, \ldots, 0.0) \in \mathbb{R}^d,$$

see Fig. 3b. The points drawn from the first two distributions belong to the first class and the remaining one to the second class. Again, we fix $n = 500$ and $d = 500$ and denote the corresponding data set by Gaussian4C. Note that these artificial data sets induce easy learning tasks in, e.g., two dimensions, even given only few labeled patterns, see again Fig. 3a, b. However, the high-dimensional variants depict quite challenging learning tasks for supervised models given only few labeled patterns due to the noise present in the data.[7] Finally, we consider the well-known two-dimensional Moons data set with $n = 200$ points; its distribution is shown in Fig. 3c.

4.1.1.2 Real-world data sets In addition to these artificial data sets, we make use of several real-world data sets including the COIL [22] data set and the USPS data set (where the latter one consists of both the training and test set of the original USPS data set [17]). For the COIL data set, we reduce the input dimensions of each image from 128×128 to 20×20 and use COIL (i, j) to denote the binary classification task induced by the objects i and j out of the available 20 objects, see Fig. 4 (numbering is from left to right and from top to bottom). A similar notation is used for the binary classification tasks induced by the 10 classes present in the USPS data set. For both the COIL and the USPS data set, we rescaled all pixels such that the resulting values lie between 0.0 and 1.0. Finally, we consider the digits data set from the MNIST database,[8] where we again focus on pairs of classes, which seem to be difficult to differentiate. Also, we restrict the size of the data set to $n = 1{,}000$ (taking the first patterns in the original training set containing 60,000 elements) and perform the same rescaling for the pixel values as for the USPS and the COIL data sets (Fig. 5).

[7] Similar data sets are often used in related experimental evaluations, see, e.g., [8]

[8] http://yann.lecun.com/exdb/mnist.

4.1.2 Implementation details

We implemented our approach with `Python 2.6.5` including the `Numpy` package. The runtime analyses are performed on a `2.66 GHz Intel Core`™ `Quad` PC running `Ubuntu 10.04`. We call the resulting implementation *semi-supervised regularized least-squares classification* (`S`^{2}`RLSC`).

4.1.3 Competing approaches

In addition to the supervised regularized least-squares classifier (`RLSC`), we use standard support vector machines to compare the efficiency of our approach (where we resort to the `LIBSVM` implementation provided by [5]). To tune the involved parameters λ (for `RLSC`) and C (for `LIBSVM`), we again perform grid search with $\lambda, C \in \{2^{-10}, \ldots, 2^{10}\}$. As semi-supervised competitor, we consider the `UniverSVM` approach proposed by [10] and perform a grid search for tuning C and C^* with $(C, C^*) \in \{2^{-10}, \ldots, 2^{10}\} \times \left\{\frac{0.01}{u}, \frac{1.0}{u}, \frac{100.0}{u}\right\}$. The ratio between the two classes is provided to the algorithm via the `-w` option. Except for the option `-S` option (which we set to -0.3), the default values for the remaining parameters are used.

4.2 Model selection

If not noted otherwise, the first half of each data set is used as training and the second half as test set. To induce a semi-supervised scenario, we split up the training set into a labeled and an unlabeled part and use different ratios for the particular setting; the specific amount of data is given in brackets for each (instance of a) data set, where l, u, t denotes the number of labeled, unlabeled, and test patterns, respectively (e.g., `Gaussian2C[l=25,u=225,t=250]`). For all experiments the average classification error (along with the one standard deviation) on the test set over 10 random partitions into labeled, unlabeled, and test patterns, is reported.

4.2.1 Parameters

To select the final models, several parameters need to be tuned. In a fully supervised setting, this is usually done via an extensive grid search over all involved parameters. However, in a semi-supervised setting, model selection (i.e., the selection of the actual approach along with its parameters) is more difficult due to the lack of labeled data and is widely considered to be an open issue [8]. Given only a small amount of labeled patterns, common approaches (like grid search along with cross-validation) might yield meaningless models. In the related literature this problem is often tackled by setting the involved parameters to reasonable values based on expert knowledge or by using the test set to tune the parameters (which is not possible in real-world scenarios).[9]

Due to this model selection problem, we consider two scenarios to select (non-fixed) parameters. The first one is a *non-realistic* scenario where we make use of the test set to evaluate the model performance. The second one is a *realistic* scenario where only the labels of the labeled part of the training set are used for model evaluation (via fivefold cross-validation). The reason for the non-realistic scenario is the following: By making use of the test set (with a large amount of labels), we can first evaluate the *flexibility* of the model. More precisely, we can first investigate if the model is in principle capable of adapting to the inherent structure of the data while ignoring the (possible) problems caused by a small validation set. Note that even given good parameters one still has to solve the combinatorial problem(s). In both scenarios, we first tune the non-fixed parameters via grid search and subsequently retrain the final model on the whole training set with the best performing set of parameters.

4.2.1.1 Model parameters We pursue the following way of selecting (and evaluating) the models: As similarity measures we consider both a *linear*

$$k(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x} \rangle \tag{17}$$

and a *radial basis function (RBF)* kernel

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}||^2}{2\sigma^2}\right) \tag{18}$$

with *kernel width* σ [17]. To select σ for the RBF kernel, we consider a small set $\{0.1\ s, 0.5\ s, 1, 5\ s, 10\ s\}$ of possible assignments, where the value s is given by

$$\sqrt{\sum_{k=1}^{d} \left(\max([\mathbf{x}_1]_k, \ldots, [\mathbf{x}_n]_k) - \min([\mathbf{x}_1]_k, \ldots, [\mathbf{x}_n]_k)\right)^2}$$

and is thus a rough estimate of the maximum distance between any pair of samples. To tune the cost parameters λ and λ', we consider a small grid $(\lambda, \lambda') \in \{2^{-10}, \ldots, 2^{10}\} \times \{0.1, 1\}$ of parameters. Concerning the balance constraint, we set b_c to an estimate obtained from all available labels (i.e., using all labels available in the data sets in the non-

[9] For instance, [8] propose to make use of the test set to select the model parameters: "This allowed for finding hyperparameter values by minimizing the test error, which is not possible in real applications; however, the results of this procedure can be useful to judge the potential of a method. To obtain results that are indicative of real world performance, the model selection has to be performed using only the small set of labeled points."

realistic scenario and using only those labels given in the training set for the realistic scenario). Further, we fix $\varepsilon = 0.1$ and $\varepsilon = 0.2$ in the non-realistic and realistic scenario, respectively.

4.2.1.2 Optimization parameters Concerning the local search, we again consider two setups: In the first one we fix $\mu = 5$ and $\nu = 25$, and stop the process if no changes have occured for n iterations. Also, we select the coordinate to be flipped for a single mutation uniformly and randomly. In the second one, we consider the special case of a $(1 + 1) - \text{EA}$, i.e., we set $\mu = 1$ and $\nu = 1$, stop the iterative process if no changes have occured for n iterations, and use a round-robin scheme to select the coordinate. Also, since our approach is susceptible to the problem of local optima, we take the best out of 10 runs for model selection (i.e., during grid search) and put more optimization effort into generating the final models by taking the best out of 50 runs.

4.2.2 Model flexibility

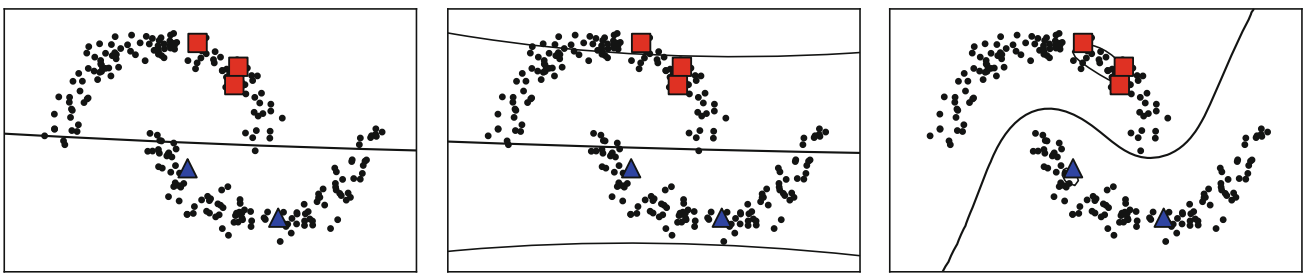
As sketched in Sect. 1, semi-supervised approaches try to take advantage from unlabeled data to improve the generalization performance. This is possible since unlabeled data can reveal more information about the underlying structure of the data. To depict this issue, we consider the artificial `Moons[l=5,u=95,t=100]` data set. To select appropriate model parameters, we make use of the test set (i.e., we consider the non-realistic scenario and perform a grid search over the non-fixed parameters) and consider the optimization setup with $\mu = 5$ and $\nu = 25$. As similarity measure, we resort to the RBF kernel. In Fig. 6, the outcome of the two supervised approaches and our semi-supervised approach is shown. It can be clearly seen that the supervised approaches are not able to generate a reasonable model. The semi-supervised approach, however, can successfully incorporate the additional information given by the unlabeled data and generates an appropriate model.

4.2.3 Amount of data

As motivated above, sufficient labeled data is essential for supervised learning approaches to yield good models. For semi-supervised approaches, the amount of unlabeled data used for training is an important issue as well. To analyze how much labeled and unlabeled data is needed for our semi-supervised approach to induce good models, we consider the `Gaussian2C` and the `Gaussian4C` data sets and vary the amount of labeled and unlabeled data. For this experiment, we consider the non-realistic scenario, use the optimization setup with $\mu = 5$ and $\nu = 25$, and resort to the linear kernel as similarity measure. As baseline, we consider the supervised `RLSC` approach. First, we vary the amount of labeled data from 5 to 80% with respect to the size of the training set; the remaining part the training set is used as unlabeled data. In Fig. 7a, b, the result of this experiment is shown: Given more than 20% labeled data, the semi-supervised approach performs better compared to the pure supervised approach whereas the performance gain is higher on the `Gaussian2C` as on the `Gaussian4C` data set. Now, we fix the amount of labeled data to 20% and vary the amount of unlabeled data from 5% to 80% with respect to the size of the training set, see Fig. 7c, d. Clearly, the semi-supervised approach is only capable of generating an appropriate model once sufficient unlabeled data is given.

4.3 Classification performance

We consider both the realistic and the non-realistic scenario to evaluate the classification performance of all competing approaches. Further, we consider up to three different ratios of labeled and unlabeled data for each particular data set. Again, the average test errors and the



(a) `RLSC` (6.3 ± 1.6) **(b)** `LIBSVM` (7.6 ± 3.1) **(c)** S^2`RLSC` (0.0 ± 0.0)

Fig. 6 The *red squares* and the *blue triangles* depict the labeled data; the *black dots* the unlabeled data. Both supervised approaches (`RLSC` and `LIBSVM`) are not able to infer a good/correct model due to the lack of labeled data whereas our semi-supervised approach (S^2`RLSC`) can successfully incorporate the unlabeled data. The average test error (with one standard deviation) over 10 random partitions into labeled, unlabeled, and test patterns, are given in brackets

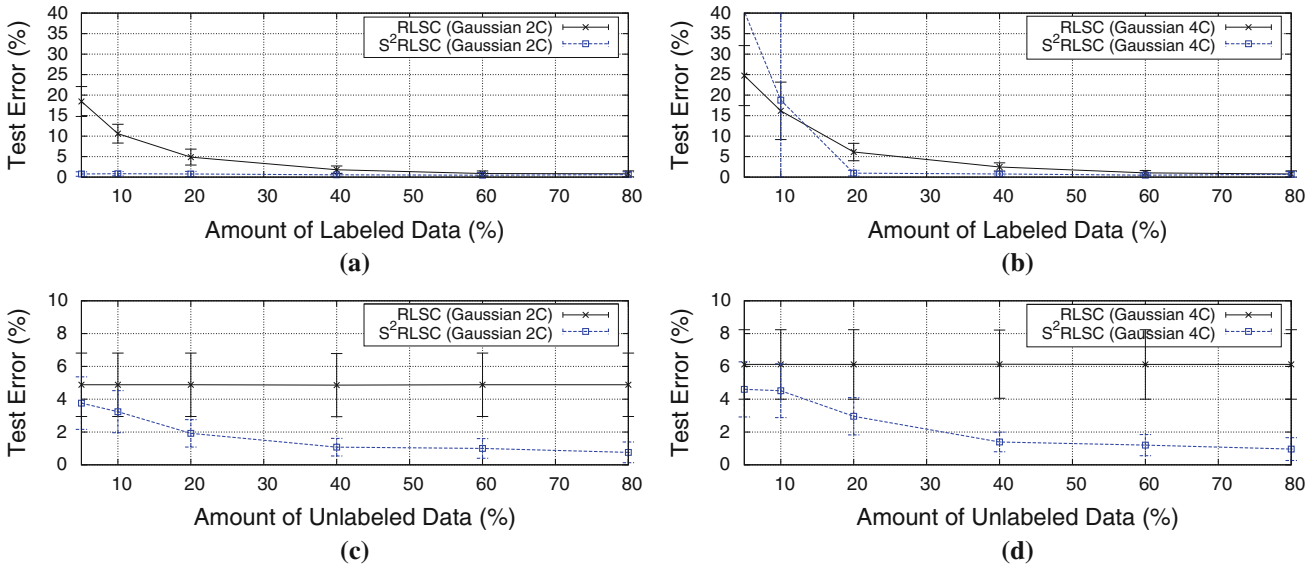


Fig. 7 If not sufficient labeled data is available, the supervised approach (RLSC) fails to yield good models on both data sets. Our semi-supervised approach (S^{2}RLSC) can successfully incorporate unlabeled data to improve the generalization performance, see Figures (**a**) and (**b**). However, sufficient unlabeled data is needed for the learner to yield a small test error, see Figures (**c**) and (**d**)

one standard deviation obtained on 10 random partitions of each data set into labeled, unlabeled, and test patterns are reported. For all data sets and for all competing approaches, a linear kernel is used. For our approach, we consider the optimization setup with $\mu = 5$ and $\nu = 25$.

4.3.1 Non-realistic scenario

In Table 2, the results for the non-realistic scenario are reported. It can be clearly seen that the semi-supervised approaches yield better results compared to the supervised ones. The results also indicate that by taking more labeled data into account, the gap between the performances of the supervised (RLSC and LIBSVM) and the semi-supervised approaches (UniverSVM and S^{2}RLSC) becomes smaller. Still, in the latter cases, the semi-supervised approaches yield better results. Hence, both semi-supervised approaches can incorporate the unlabeled data and seem being able to approximate the induced combinatorial problems successfully.

4.3.2 Realistic scenario

The results for the realistic scenario are given in Table 3. Compared to the results for the non-realistic scenario, the performances of the semi-supervised approaches are clearly worse. Hence, the lack of labeled data for model evaluation as well as a (possibly) bad estimate for the balance parameter b_c seem to have a significant influence on the final classification performance. As mentioned above, this is a common problem in semi-supervised learning. Naturally, this point depends heavily on the amount of labeled data. The results indicate that once a reasonable amount of data is used for training in this scenario, the negative influence of the latter disturbing factors is reduced. We would like to point out that a bad estimate for the balance parameter b_c can jeopardize the overall outcome; this issue, however, is not severe since good estimates for this (one-dimensional) parameter should be obtainable given a reasonable amount of labeled patterns.

4.4 Computational considerations

The computational shortcut reduces the runtime for a single fitness evaluation from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$. In addition, the approximation scheme shortens both drawbacks, the cubic preprocessing time and the quadratic storage requirements. In this subsection, we will investigate the efficiency of the shortcut and the influence of the approximation scheme on the classification performance. Further, we will apply the approach to a large-scale scenario induced by the MNIST data set. For the remainder of this section, we set $\mu = 1$ and $\nu = 1$, thus considering the special case of the $(1 + 1) - \text{EA}$.[10]

4.4.1 Efficiency of shortcut

To investigate the practical benefits of the proposed computational shortcut, we compare the naive way for

[10] As a side note, we would like to point out that this special type shows a very similar behavior with respect to the classification performance compared to the more general setup with $\mu = 5$ and $\nu = 25$ (if sufficient restarts are performed).

Table 2 The table shows the classification performance of all competing approaches for the *non-realistic scenario*, i.e., the test set was used to tune the non-fixed parameters

Data Set	RLSC	LIBSVM	UniverSVM	S^2RLSC
Gaussian2C [l = 25, u = 225,t = 250]	10.6 ± 02.3	13.0 ± 02.9	01.0 ± 00.5	00.8 ± 00.5
Gaussian2C [l = 50, u = 200,t = 250]	04.9 ± 01.9	05.8 ± 02.2	01.0 ± 00.4	00.8 ± 00.6
Gaussian4C [l = 25, u = 225,t = 250]	16.1 ± 07.0	17.4 ± 06.6	07.6 ± 12.2	16.4 ± 22.2
Gaussian4C [l = 50, u = 200,t = 250]	06.1 ± 02.1	06.7 ± 01.5	01.6 ± 00.8	01.0 ± 00.7
COIL (3, 6) [l = 14,u = 101, t = 29]	14.5 ± 07.7	13.4 ± 07.0	02.8 ± 03.7	05.1 ± 09.5
COIL (3, 6) [l = 28, u = 87,t=29]	04.1 ± 04.8	03.1 ± 03.3	00.3 ± 01.0	00.0 ± 00.0
COIL (5, 9) [l = 14, u = 101,t=29]	13.1 ± 08.0	13.4 ± 07.8	06.9 ± 07.2	01.7 ± 03.5
COIL (5, 9) [l = 28, u = 87, t = 29]	03.1 ± 04.2	03.4 ± 04.1	01.4 ± 03.2	00.0 ± 00.0
COIL (6, 19) [l = 14, u = 101, t = 29]	10.0 ± 06.8	12.1 ± 09.8	04.5 ± 08.2	00.3 ± 01.0
COIL (6, 19) [l = 28, u = 87, t = 29]	02.4 ± 03.1	03.1 ± 03.3	00.7 ± 02.1	01.0 ± 03.1
COIL (18, 19) [l = 14,u = 101, t = 29]	10.0 ± 08.6	06.9 ± 08.0	01.0 ± 03.1	02.1 ± 06.2
COIL (18, 19) [l = 28, u = 87, t = 29]	03.4 ± 05.8	01.4 ± 04.1	00.0 ± 00.0	00.0 ± 00.0
USPS (2, 5) [l = 16, u = 806, t=823]	07.9 ± 02.9	09.4 ± 05.1	03.2 ± 00.5	04.6 ± 02.8
USPS (2, 5) [l = 32, u = 790, t = 823]	04.4 ± 00.5	04.7 ± 00.7	03.2 ± 00.5	03.7 ± 00.7
USPS (2, 7) [l = 17, u = 843, t = 861]	03.6 ± 02.5	04.6 ± 03.0	01.5 ± 00.3	01.0 ± 00.2
USPS (2, 7) [l = 34, u = 826, t = 861]	02.2 ± 00.7	02.5 ± 01.0	01.4 ± 00.2	01.0 ± 00.2
USPS (3, 8) [l = 15, u = 751, t = 766]	09.8 ± 06.6	12.0 ± 08.2	04.8 ± 01.1	04.5 ± 01.8
USPS (3, 8) [l = 30, u = 736, t = 766]	06.3 ± 02.0	06.6 ± 02.1	04.0 ± 00.1	04.3 ± 01.5
USPS (8, 0) [l = 22, u = 1108, t = 1131]	04.8 ± 01.9	04.8 ± 01.7	01.7 ± 00.7	01.2 ± 00.2
USPS (8, 0) [l = 45, u = 1085, t = 1131]	02.6 ± 00.8	02.7 ± 00.8	01.3 ± 00.4	01.2 ± 00.2
MNIST (1, 7) [l = 10, u = 490, t = 500]	04.5 ± 03.6	07.3 ± 04.6	02.9 ± 00.9	02.0 ± 00.6
MNIST (1, 7) [l = 20, u = 480, t = 500]	02.7 ± 01.1	03.5 ± 01.3	02.6 ± 01.0	01.9 ± 00.5
MNIST (1, 7) [l = 50, u = 400, t = 500]	02.1 ± 01.1	02.3 ± 01.0	02.2 ± 00.9	01.7 ± 00.8
MNIST (2, 5) [l = 10, u = 490, t = 500]	14.9 ± 08.9	15.9 ± 08.4	03.7 ± 01.1	04.7 ± 03.2
MNIST (2, 5) [l = 20, u = 480, t = 500]	07.7 ± 03.3	09.3 ± 03.2	03.4 ± 00.7	03.4 ± 02.1
MNIST (2, 5) [l = 50, u = 400, t = 500]	04.0 ± 00.9	04.7 ± 01.2	03.4 ± 00.7	02.6 ± 00.6
MNIST (2, 7) [l = 10, u = 490, t = 500]	08.7 ± 05.5	12.0 ± 08.0	03.0 ± 00.7	03.1 ± 00.9
MNIST (2, 7) [l = 20, u = 480, t = 500]	05.6 ± 02.7	06.7 ± 03.7	02.9 ± 00.6	03.0 ± 01.4
MNIST (2, 7) [l = 50, u = 400, t = 500]	03.5 ± 00.9	04.2 ± 01.2	02.5 ± 00.5	02.6 ± 00.9
MNIST (3, 8) [l = 10, u = 490, t = 500]	19.4 ± 05.6	20.7 ± 05.9	12.3 ± 03.8	08.3 ± 01.6
MNIST (3, 8) [l = 20, u = 480, t = 500]	13.7 ± 04.1	15.2 ± 04.1	08.6 ± 03.3	07.7 ± 01.8
MNIST (3, 8) [l = 50, u = 400, t = 500]	08.2 ± 02.2	08.6 ± 02.5	06.3 ± 02.0	07.4 ± 03.0

The results clearly show that the semi-supervised approaches are able to incorporate the unlabeled data successfully. While these results are not indicative for real-world scenarios, they demonstrate the potential of both semi-supervised approaches. They also show that both approaches can find good local optima for their corresponding optimization tasks

obtaining the fitness values (taking $\mathcal{O}(n^2)$ time per fitness evaluation) with its accelerated variant (taking $\mathcal{O}(n)$ time per fitness evaluation). For this purpose, we consider the following experiment on the Gaussian2C data set (of varying size). For reasons of simplification, we fix the cost parameters $\lambda = 1$ and $\lambda' = 1$ and use a linear kernel. To aim at practical settings, we take the best out of 50 runs and measure the overall runtime (including the preprocessing time) of both induced implementations. In Fig. 8 the runtime behavior of both implementations is given. Clearly, the computational shortcut reduces the practical runtime dramatically.

4.4.2 Sparse approximation

To evaluate the influence of the approximation scheme, we consider the four USPS data set instances (i.e., USPS (2, 5), USPS (2, 7), USPS (3, 8), and USPS (8, 0)) and vary the assignment for the approximation parameter r from 0.01 to 0.1 n; all patterns are used as possible basis vectors. Again, we fix the cost parameters $\lambda = 1$ and $\lambda' = 1$, and use a linear kernel as similarity measure. The results is shown in Fig. 9. The plots indicate that the classification performance is not affected by the approximation scheme as long as a reasonable amount of data is

Table 3 The table shows the classification performance for the *realistic scenario*, i.e., only the labels in the training set are used for model selection and for estimating the balance parameter b_c

Data Set	RLSC	LIBSVM	UniverSVM	S^2RLSC
Gaussian2C [l = 25, u = 225, t = 250]	11.4 ± 02.4	13.2 ± 02.8	01.8 ± 00.9	03.3 ± 01.8
Gaussian2C [l = 50, u = 200,t=250]	05.3 ± 02.1	06.3 ± 02.4	01.8 ± 00.8	02.4 ± 01.5
Gaussian4C [l = 25, u = 225, t = 250]	16.5 ± 06.8	20.6 ± 11.5	13.3 ± 15.2	22.4 ± 19.1
Gaussian4C [l = 50, u = 200,t=250]	06.5 ± 01.9	06.9 ± 01.6	02.5 ± 01.5	03.2 ± 02.2
COIL (3, 6) [l = 14, u = 101,t=29]	15.5 ± 08.3	16.2 ± 07.2	16.9 ± 13.9	10.7 ± 11.9
COIL (3, 6) [l = 28, u = 87, t = 29]	06.2 ± 05.7	03.8 ± 04.2	05.2 ± 05.8	01.7 ± 03.5
COIL (5, 9) [l = 14, u = 101,t=29]	16.2 ± 09.5	13.4 ± 07.8	19.3 ± 10.9	15.5 ± 12.0
COIL (5, 9) [l = 28, u = 87,t=29]	04.5 ± 07.4	04.5 ± 05.6	07.2 ± 09.1	02.8 ± 04.6
COIL (6, 19) [l = 14, u = 101, t = 29]	13.4 ± 11.1	15.5 ± 13.1	21.0 ± 12.0	04.8 ± 13.4
COIL (6, 19) [l = 28, u = 87, t = 29]	03.8 ± 03.3	03.4 ± 03.4	04.5 ± 05.1	01.7 ± 03.2
COIL (18, 19) [l = 14,u = 101, t = 29]	10.7 ± 08.1	06.9 ± 08.0	07.6 ± 08.8	13.1 ± 11.8
COIL (18, 19) [l = 28, u = 87, t = 29]	03.8 ± 06.1	01.4 ± 04.2	05.2 ± 09.7	01.4 ± 04.1
USPS (2, 5) [l = 16, u = 806, t = 823]	09.3 ± 02.3	10.5 ± 04.7	09.0 ± 05.6	13.7 ± 12.2
USPS (2, 5) [l = 32, u = 790,t=823]	05.8 ± 01.6	05.4 ± 00.8	05.7 ± 01.8	05.0 ± 01.3
USPS (2, 7) [l = 17, u = 843, t = 861]	04.0 ± 02.4	04.9 ± 02.9	06.1 ± 05.3	04.0 ± 06.7
USPS (2, 7) [l = 34, u = 826, t = 861]	03.1 ± 01.7	02.8 ± 01.1	03.4 ± 02.4	01.3 ± 00.3
USPS (3, 8) [l = 15, u = 751, t = 766]	11.0 ± 06.4	12.9 ± 08.3	08.7 ± 03.9	13.4 ± 15.0
USPS (3, 8) [l = 30, u = 736, t = 766]	07.7 ± 01.4	07.3 ± 02.1	07.1 ± 01.8	07.2 ± 04.4
USPS (8, 0) [l = 22, u = 1108, t = 1131]	06.6 ± 03.3	05.0 ± 02.0	03.2 ± 02.2	02.7 ± 01.8
USPS (8, 0) [l = 45, u = 1085, t = 1131]	03.3 ± 00.9	03.0 ± 00.9	03.3 ± 01.8	02.0 ± 00.8
MNIST (1, 7) [l = 10, u = 490, t = 500]	05.4 ± 03.6	07.8 ± 05.2	09.5 ± 07.5	09.2 ± 12.0
MNIST (1, 7) [l = 20, u = 480, t = 500]	03.9 ± 01.3	04.2 ± 01.6	04.3 ± 02.8	02.7 ± 01.1
MNIST (1, 7) [l = 50, u = 400, t = 500]	02.8 ± 01.7	02.6 ± 01.0	03.7 ± 02.5	02.2 ± 01.0
MNIST (2, 5) [l = 10, u = 490, t = 500]	17.4 ± 11.5	17.9 ± 12.3	15.7 ± 12.5	26.0 ± 17.9
MNIST (2, 5) [l = 20, u = 480, t = 500]	09.2 ± 02.6	10.2 ± 03.1	06.3 ± 03.9	10.1 ± 09.3
MNIST (2, 5) [l = 50, u = 400, t = 500]	05.4 ± 02.1	05.8 ± 01.7	04.2 ± 01.6	04.5 ± 01.8
MNIST (2, 7) [l = 10, u = 490, t = 500]	10.2 ± 05.9	12.1 ± 07.9	13.3 ± 14.5	17.8 ± 16.6
MNIST (2, 7) [l = 20, u = 480, t = 500]	06.6 ± 02.7	07.9 ± 04.4	08.0 ± 04.7	06.8 ± 03.4
MNIST (2, 7) [l = 50, u = 400, t = 500]	04.0 ± 00.8	05.0 ± 01.5	05.1 ± 01.6	04.7 ± 02.6
MNIST (3, 8) [l = 10, u = 490, t = 500]	21.3 ± 05.7	26.1 ± 11.1	19.9 ± 06.5	28.6 ± 15.1
MNIST (3, 8) [l = 20, u = 480, t = 500]	18.4 ± 11.7	18.8 ± 11.5	16.1 ± 03.9	11.7 ± 08.2
MNIST (3, 8) [l = 50, u = 400, t = 500]	09.0 ± 02.2	09.0 ± 02.4	09.5 ± 04.1	09.0 ± 02.9

Compared to the results for the non-realistic scenario, the results are clearly worse. Hence, both the model selection problem as well as a (possibly) bad estimate for b_c seem to affect the classification performance. However, once sufficient labeled data is given, our semi-supervised approach yields better classification models in most cases

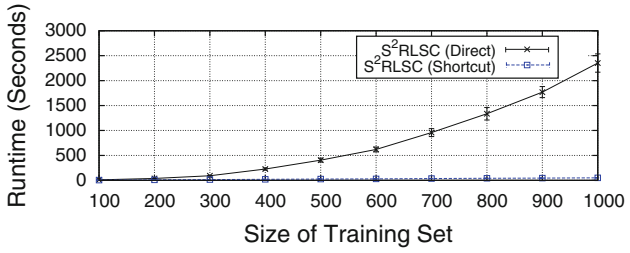


Fig. 8 Efficiency of the computational shortcut compared to the direct way for obtaining the fitness values

used for the approximation scheme (in these cases roughly 5% of all patterns).

4.4.3 *Large-scale scenario*

To test the scalability of our approach on real-world data, we consider the MNIST (1, 7) data set and vary the size of the training set (from 500 to 5,000 patterns in each the training and test set). In the training set, the amount of labeled patterns is set to 1.0%; the remaining data is used as unlabeled data. Further, we use a linear kernel as

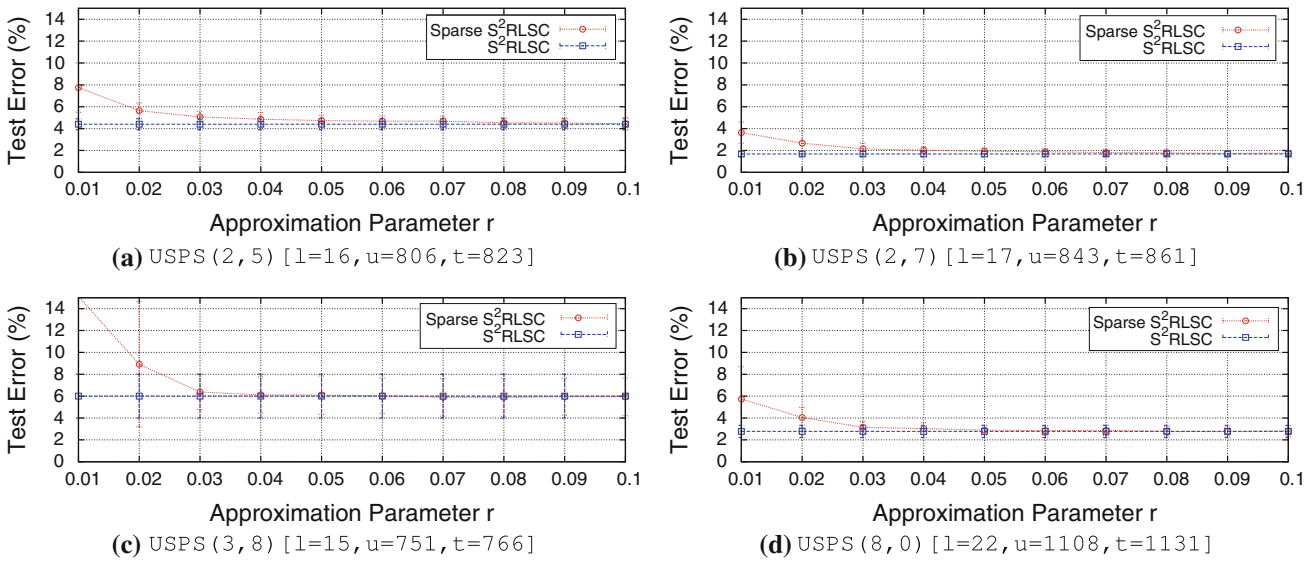


Fig. 9 The influence of the approximation parameter r is shown in these *plots*. They indicate that if a reasonable amount of data is used as basis vectors, the classification performance is not affected by the approximation scheme

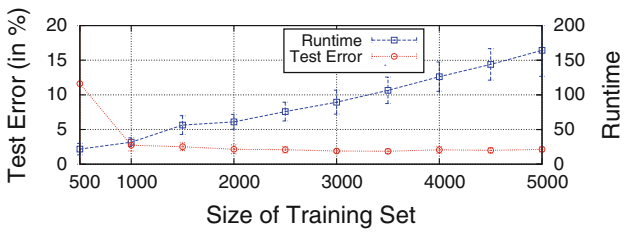


Fig. 10 Large-scale experiment

similarity measure, fix the parameters $\lambda = 1$ and $\lambda' = 1$, and apply the approximation scheme with $r = 100$. The estimate for the balance parameter b_c is obtained via the labeled patterns given in the training set and the parameter ε is set to 0.2. The results are shown in Fig. 10. They indicate that our approach can deal efficiently with large amounts of data. Note that the results depict the runtime needed for the $(1 + 1) - \text{EA}$ to build the final model (where the best result with respect to the fitness value is taken out of 50 runs).

5 Conclusions

We proposed a general optimization framework for semi- and unsupervised regularized least-squares classification. The key idea consists in using simple local search strategies which can be accelerated based on efficient matrix update schemes for the intermediate candidate solutions. Our experimental evaluation demonstrates the performance of our approach, both with respect to the runtime as well as with respect to the classification performance. The results indicate that the optimization task(s) can successfully be addressed via the simple local search strategy once sufficient labeled data is given for model evaluation and for obtaining good estimates for the balance constraint.

Acknowledgments This work has been supported in part by funds of the *Deutsche Forschungsgemeinschaft* (Fabian Gieseke, grant KR 3695) and by the Academy of Finland (Tapio Pahikkala, grant 134020).

Appendix 1: Sparse approximation

We will now depict the approximation scheme for the kernel matrix $\mathbf{K}$, which is based on the so-called *Nyström approximation*

$$\widetilde{\mathbf{K}} = (\mathbf{K}_R)^{\mathrm{T}}(\mathbf{K}_{R,R})^{-1}\mathbf{K}_R, \tag{19}$$

see, e.g., [24]. Plugging in this approximation into (10), we get

$$(\mathbf{D}\bar{\mathbf{y}} - \mathbf{D}\widetilde{\mathbf{K}}\mathbf{c}^*)^{\mathrm{T}}(\mathbf{D}\bar{\mathbf{y}} - \mathbf{D}\widetilde{\mathbf{K}}\mathbf{c}^*) + \lambda(\mathbf{c}^*)^{\mathrm{T}}\widetilde{\mathbf{K}}\mathbf{c}^* \tag{20}$$

as new objective value. The matrix $\overline{\mathbf{K}} = \mathbf{D}\widetilde{\mathbf{K}}\mathbf{D}$ has (at most) r non-zero eigenvalues. To compute them efficiently, we make use of the following derivations: Let $\mathbf{B}\,\mathbf{B}^{\mathrm{T}}$ be the Cholesky decomposition of the matrix $(\mathbf{K}_{R,R})^{-1}$ and $\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}}$ be the thin singular value decomposition of $\mathbf{B}^{\mathrm{T}}\,\mathbf{K}_R\mathbf{D}$. The r nonzero eigenvalues of

$$\overline{\mathbf{K}} = \mathbf{D}\widetilde{\mathbf{K}}\mathbf{D} = \mathbf{D}(\mathbf{K}_R)^{\mathrm{T}}\mathbf{B}\mathbf{B}^{\mathrm{T}}\mathbf{K}_R\mathbf{D} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^{\mathrm{T}}\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}}$$

can then be obtained from $\boldsymbol{\Sigma}^2 \in \mathbb{R}^{r\times r}$ and the matrix $\mathbf{V} \in \mathbb{R}^{n\times r}$ consists of the corresponding eigenvectors (we have $\mathbf{U}^{\mathrm{T}}\,\mathbf{U} = \mathbf{I}$, see below). By assuming that these non-zero eigenvalues are the first r elements in the matrix

$\mathbf{\Lambda} \in \mathbb{R}^{n \times n}$ of eigenvalues (of $\widetilde{\mathbf{K}}$), we have $[\mathbf{\Lambda}\tilde{\mathbf{\Lambda}}]_{i,i} = 0$ for $i = r+1, \ldots, n$; hence, the remaining eigenvectors (with eigenvalue 0) do not have to be computed for the evaluation of (13). To sum up, $\mathbf{y}^{\mathrm{T}}\ \mathbf{D}\ \mathbf{V}$ can be updated in $\mathcal{O}(r)$ time per single coordinate flip. Further, all preprocessing matrices can be obtained in $\mathcal{O}(nr^2)$ runtime (in practice and up to machine precision) using $\mathcal{O}(nr)$ space.

Appendix 2: Matrix calculus

For completeness, we summarize some basic definitions and theorems of the field of matrix calculus that may be helpful when reading the paper. The following definitions and facts are taken from [19] and [16].

Definition 1 (*Positive (Semi-)Definite Matrices*) A symmetric matrix $\mathbf{M} \in \mathbb{R}^{m \times m}$ is said to be *positive definite* if

$$\mathbf{v}^{\mathrm{T}}\mathbf{M}\mathbf{v} > 0 \text{ holds for all } \mathbf{v} \in \mathbb{R}^m \text{ with } \mathbf{v} \neq 0 \tag{21}$$

and *positive semidefinite* if

$$\mathbf{v}^{\mathrm{T}}\mathbf{M}\mathbf{v} \geq 0 \text{ holds for all } \mathbf{v} \in \mathbb{R}^m \text{ with } \mathbf{v} \neq 0. \tag{22}$$

We use the notations $\mathbf{M} \succ 0$ and $\mathbf{M} \succeq 0$ if $\mathbf{M}$ is positive definite or positive semidefinite, respectively. It is straightforward to derive that if $\mathbf{M}_1, \ldots, \mathbf{M}_p \in \mathbb{R}^{m \times m}$ are positive definite matrices and $\alpha_1, \ldots, \alpha_p \in \mathbb{R}$ are positive coefficients, then

$$a_1\mathbf{M}_1 + \ldots + a_p\mathbf{M}_p \tag{23}$$

is positive definite as well, i.e., any positive linear combination of positive definite matrices is positive definite ([19], pp. 396–398). A *lower triangular matrix* is a matrix, where the entries above its diagonal are zero.

Fact 1 (**Cholesky Decomposition**) Any symmetric positive definite matrix $\mathbf{M} \in \mathbb{R}^{m \times m}$ can be factorized as

$$\mathbf{M} = \mathbf{N}\mathbf{N}^{\mathrm{T}}, \tag{24}$$

where $\mathbf{N} \in \mathbb{R}^{m \times m}$ is a lower triangular matrix whose diagonal entries are strictly positive. This factorization is known as the *Cholesky decomposition*.

The Cholesky decomposition for a $m \times m$-matrix can be obtained in $O(m^3)$ time (in practice and up to machine precision, see [16] pp. 141–145).

Definition 2 (*Orthogonal Matrix*) A matrix $\mathbf{M} \in \mathbb{R}^{m \times m}$ is called *orthogonal* if

$$\mathbf{M}^{\mathrm{T}}\mathbf{M} = \mathbf{M}\mathbf{M}^{\mathrm{T}} = \mathbf{I},$$

i.e., if the inverse $\mathbf{M}^{-1}$ of a $\mathbf{M}$ equals its transpose $\mathbf{M}^{\mathrm{T}}$.

Fact 2 (**Singular Value Decomposition**) A matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ can be written in the form

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}, \tag{25}$$

where $\mathbf{U} \in \mathbb{R}^{m \times m}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ are orthogonal, and where $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ is a diagonal matrix with non-negative entries. The decomposition is called the *singular value decomposition* (SVD) of $\mathbf{M}$.

The values on the diagonal matrix are called the *singular values* of $\mathbf{M}$; they are usually arranged in descending order, i.e., $[\mathbf{\Sigma}]_{1,1} \geq \ldots \geq [\mathbf{\Sigma}]_{p,p}$ with $p = \min(n, m)$.

Fact 3 (**Thin Singular Value Decomposition**) The *thin* or *economy-size singular value decomposition* of $\mathbf{M} \in \mathbb{R}^{m \times n}$ with $m \geq n$ is of the form

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}, \tag{26}$$

where $\mathbf{U} \in \mathbb{R}^{m \times n}, \mathbf{\Sigma} \in \mathbb{R}^{n \times n}$, and $\mathbf{V} \in \mathbb{R}^{n \times n}$. Further, we have $\mathbf{U}^{\mathrm{T}}\ \mathbf{U} = \mathbf{V}^{\mathrm{T}}\ \mathbf{V} = \mathbf{V}\ \mathbf{V}^{\mathrm{T}} = \mathbf{I}$ (but not $\mathbf{U}\ \mathbf{U}^{\mathrm{T}} = \mathbf{I}$).

Note that the thin singular value decomposition for a matrix $\mathbf{M} \in \mathbb{R}^{n \times m}$ can be computed in $O(nm^2)$ time (in practice and up to machine precision, see [16] p. 239).

Fact 4 (**Eigendecomposition**) If $\mathbf{M} \in \mathbb{R}^{m \times m}$ is symmetric, then it can be factorized as

$$\mathbf{M} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathrm{T}}, \tag{27}$$

where $\mathbf{V} \in \mathbb{R}^{m \times m}$ is an orthogonal matrix containing the eigenvectors of $\mathbf{M}$ and $\mathbf{\Lambda}$ is a diagonal matrix containing the corresponding eigenvalues ([19], p. 107).

Note that if the nonzero eigenvalues are stored in the first r diagonal entries of $\mathbf{\Lambda}$, then (analogously to the economy-sized singular value decomposition) the matrix $\mathbf{M}$ can be written as in (27) but with $\mathbf{V} \in \mathbb{R}^{m \times r}$ and $\mathbf{\Lambda} \in \mathbb{R}^{r \times r}$.

Fact 5 (**SVD and Eigendecomposition**) We have the following relationship between the SVD and the eigendecomposition. If (25) or (26) is the SVD of $\mathbf{M} \in \mathbb{R}^{m \times m}$, then

$$\mathbf{M}^{\mathrm{T}}\mathbf{M} = \mathbf{V}\mathbf{\Sigma}^{\mathrm{T}}\mathbf{U}^{\mathrm{T}}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}} = \mathbf{V}\mathbf{\Sigma}^{\mathrm{T}}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}} \tag{28}$$

is the eigendecomposition of $\mathbf{M}^{\mathrm{T}}\mathbf{M}$. Here, the eigenvalues of the matrix $\mathbf{M}^{\mathrm{T}}\mathbf{M}$ are the squares of the singular values of $\mathbf{M}$. Note that an analogous relationship also holds between the economy-sized decompositions.

Fact 6 (**Further Matrix Properties**) If $\mathbf{M} \in \mathbb{R}^{m \times m}$ is a (symmetric) positive definite matrix and $\mathbf{M} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathrm{T}}$ is its eigendecomposition, then

$$[\mathbf{\Lambda}]_{i,i} > 0 \text{ holds for all } i \in \{1, \ldots, m\}, \tag{29}$$

that is, the eigenvalues of positive definite matrices are strictly positive real numbers ([19], p. 398). From this, it follows that all positive definite matrices are invertible and their inverse matrices are also positive definite. Moreover, we have

$$\mathbf{M}_{L,L} \succ 0 \text{ for all } L \subseteq \{1, \ldots, m\}, \tag{30}$$

that is, all principal submatrices of $\mathbf{M}$ are positive definite ([19], p. 397). Further, if $\mathbf{M} \in \mathbb{R}^{m \times m}$ is symmetric positive definite, then we have

$$\mathbf{N}^{\mathrm{T}}\mathbf{M}\mathbf{N} \geq 0, \forall \mathbf{N} \in \mathbb{R}^{m \times n}, n \in \mathbb{N}. \tag{31}$$

This is a special case of the Observation 7.7.2 given by Horn and Johnson [19].

References

1. Bennett KP, Demiriz A (1998) Semi-supervised support vector machines. In: Kearns MJ, Solla SA, Cohn DA (eds) Advances in neural information processing systems 11, MIT Press, pp 368–374
2. Beyer HG, Schwefel HP (2002) Evolution strategies—a comprehensive introduction. Nat Comput 1:3–52
3. Bie TD, Cristianini N (2003) Convex methods for transduction. In: Advances in neural information processing systems 16, MIT Press, pp 73–80
4. Boyd S, Vandenberghe L (2004) Convex optimization. Cambridge University Press, New York
5. Chang CC, Lin CJ (2001) LIBSVM: a library for support vector machines. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm
6. Chapelle O, Zien A (2005) Semi-supervised classification by low density separation. In: Proceedings of the tenth international workshop on artificial intelligence and statistics, pp 57–64
7. Chapelle O, Chi M, Zien A (2006) A continuation method for semi-supervised svms. In: Proceedings of the international conference on machine learning, pp 185–192
8. Chapelle, O, Schölkopf, B, Zien, A (eds) (2006) Semi-supervised learning. MIT Press, Cambridge, MA
9. Chapelle O, Sindhwani V, Keerthi SS (2008) Optimization techniques for semi-supervised support vector machines. J Mach Learn Res 9:203–233
10. Collobert R, Sinz F, Weston J, Bottou L (2006) Trading convexity for scalability. In: Proceedings of the international conference on machine learning, pp 201–208
11. Droste S, Jansen T, Wegener I (2002) On the analysis of the (1+1) evolutionary algorithm. Theor Comput Sci 276(1–2):51–81
12. Evgeniou T, Pontil M, Poggio T (2000) Regularization networks and support vector machines. Adv Comput Math 13(1):1–50, http://dx.doi.org/10.1023/A:1018946025316
13. Fogel DB (1966) Artificial intelligence through simulated evolution. Wiley, New York
14. Fung G, Mangasarian OL (2001) Semi-supervised support vector machines for unlabeled data classification. Optim Methods Softw 15:29–44
15. Gieseke F, Pahikkala T, Kramer O (2009) Fast evolutionary maximum margin clustering. In: Proceedings of the international conference on machine learning, pp 361–368
16. Golub GH, Van Loan C (1989) Matrix computations, 2nd edn. Johns Hopkins University Press, Baltimore and London
17. Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning. Springer, New York
18. Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
19. Horn R, Johnson CR (1985) Matrix analysis. Cambridge University Press, Cambridge
20. Joachims T (1999) Transductive inference for text classification using support vector machines. In: Proceedings of the international conference on machine learning, pp 200–209
21. Mierswa I (2009) Non-convex and multi-objective optimization in data mining. PhD thesis, Technische Universität Dortmund
22. Nene S, Nayar S, Murase H (1996) Columbia object image library (coil-100). Tech. rep
23. Rechenberg I (1973) Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution. Frommann-Holzboog, Stuttgart
24. Rifkin R, Yeo G, Poggio T (2003) Regularized least-squares classification. In: Advances in learning theory: methods, models and applications, IOS Press, pp 131–154
25. Rifkin RM (2002) Everything old is new again: a fresh look at historical approaches in machine learning. PhD thesis, MIT
26. Schölkopf B, Herbrich R, Smola AJ (2001) A generalized representer theorem. In: Proceedings of the 14th annual conference on computational learning theory and 5th European conference on computational learning theory. Springer, London, pp 416–426
27. Schwefel HP (1977) Numerische optimierung von computermodellen mittel der evolutionsstrategie. Birkhuser, Basel
28. Silva C, Santos JS, Wanner EF, Carrano EG, Takahashi RHC (2009) Semi-supervised training of least squares support vector machine using a multiobjective evolutionary algorithm. In: Proceedings of the eleventh conference on congress on evolutionary computation, IEEE Press, Piscataway, NJ, USA, pp 2996–3002
29. Sindhwani V, Keerthi S, Chapelle O (2006) Deterministic annealing for semi-supervised kernel machines. In: Proceedings of the international conference on machine learning, pp 841–848
30. Steinwart I, Christmann A (2008) Support vector machines. Springer, New York
31. Suykens JAK, Vandewalle J (1999) Least squares support vector machine classifiers. Neural Process Lett 9(3):293–300
32. Valizadegan H, Jin R (2007) Generalized maximum margin clustering and unsupervised kernel learning. In: Advances in neural information processing systems, MIT Press, vol 19, pp 1417–1424
33. Vapnik V (1998) Statistical learning theory. Wiley, New York
34. Xu L, Schuurmans D (2005) Unsupervised and semi-supervised multi-class support vector machines. In: Proceedings of the national conference on artificial intelligence, pp 904–910
35. Xu L, Neufeld J, Larson B, Schuurmans D (2005) Maximum margin clustering. In: Advances in neural information processing systems vol 17, pp 1537–1544
36. Zhang K, Tsang IW, Kwok JT (2007) Maximum margin clustering made practical. In: Proceedings of the international conference on machine learning, pp 1119–1126
37. Zhao B, Wang F, Zhang C (2008a) Efficient maximum margin clustering via cutting plane algorithm. In: Proceedings of the SIAM international conference on data mining, pp 751–762
38. Zhao B, Wang F, Zhang C (2008b) Efficient multiclass maximum margin clustering. In: Proceedings of the international conference on machine learning, pp 1248–1255
39. Zhu X, Goldberg AB (2009) Introduction to semi-supervised learning. Morgan and Claypool, Seattle