Speedy Local Search for Semi-Supervised Regularized Least-Squares

Fabian Gieseke[1], Oliver Kramer[1], Antti Airola[2], and Tapio Pahikkala[2]

[1] Department Informatik
Carl von Ossietzky Universität Oldenburg
26111 Oldenburg, Germany
`fabiangieseke@googlemail.com`
`okramer@icsi.berkeley.edu`

[2] Turku Centre for Computer Science,
Department of Information Technology,
University of Turku, 20520 Turku, Finland
`{antti.airola,tapio.pahikkala}@utu.fi`

Abstract. In real-world machine learning scenarios, labeled data is often rare while unlabeled data can be obtained easily. Semi-supervised approaches aim at improving the prediction performance by taking both the labeled as well as the unlabeled part of the data into account. In particular, semi-supervised support vector machines favor decision hyperplanes which lie in a "low-density area" induced by the unlabeled patterns (while still considering the labeled part of the data). The associated optimization problem, however, is of combinatorial nature and, hence, difficult to solve. In this work, we present an efficient implementation of a simple local search strategy that is based on matrix updates of the intermediate candidate solutions. Our experiments on both artificial and real-world data sets indicate that the approach can successfully incorporate unlabeled data in an efficient manner.

1 Introduction

If sufficient labeled training data is available, well-known classification techniques like the *k-nearest neighbor*-classifier or *support vector machines* (SVMs) [10] often yield satisfying results. In real-world applications, however, labeled data is mostly rare. One of the current research directions in machine learning is *semi-supervised learning* [5,18]. Compared to purely supervised learning approaches, semi-supervised techniques try to take advantage not only of the labeled but also of the unlabeled data which can often be gathered more easily. One of the most prominent semi-supervised classification approaches are *semi-supervised support vector machines* (S^{3}VMs) [11], which depict the direct extension of support vector machines to semi-supervised scenarios: Given a set of labeled training patterns, the goal of a standard support vector machine consists in finding a hyperplane which separates both classes well such that the "margin" induced by the hyperplane and the patterns is maximized, see Figure 1 (a). This concept can

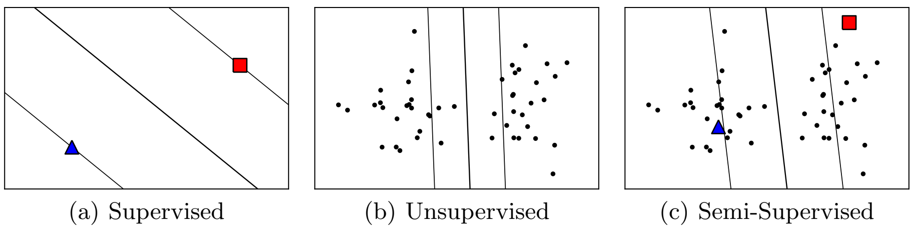


Fig. 1. In supervised scenarios, we are only given labeled patterns (squares and triangles). Thus, given only a small amount of data, a support vector machine cannot yield a good classification model, see Figure (a). Both unsupervised and semi-supervised support vector machines try to incorporate unlabeled patterns (dots) to reveal more information about the structure of the data, see Figures (b) and (c).

also be considered in learning scenarios where only unlabeled training patterns are given. Here, the goal consists in finding the optimal partition of the data into two classes (given some constraints) such that a *subsequent* application of a support vector machine leads to the best possible result, see Figure 1 (b). Semi-supervised support vector machines can be seen as "intermediate" approach between the latter two ones. Here, the aim of the learning task consists in finding a hyperplane which separates both classes well (based on the labeled part of the data) and, at the same time, passes through a "low-density area" induced by the unlabeled part of the data, see Figure 1 (c).

1.1 Related Work and Contribution

The original problem formulation of semi-supervised support vector machines was given by Vapnik and Sterin [16] under the name *transductive support vector machines*. Aiming at practical settings, Joachims [11] proposed a label-switching strategy which iteratively tries to improve an initial "guess" obtained via a (modified) support vector machine on the labeled part of the data. A variety of different techniques has been proposed in recent years which are based on semi-definite programming [2], the convex-concave procedure [8], deterministic annealing [14], the continuation method [4] and other techniques [1,7,17]. Many other approaches exist and we refer to Chapelle *et al.* [5] and Zhu *et al.* [18] for comprehensive surveys. In this paper, we propose an efficient implementation of a least-squares variant for the original problem definition. More precisely, we propose an efficient implementation of a simple local search strategy which is based on matrix update schemes of the intermediate candidate solutions. Our experiments indicate that the resulting approach can incorporate unlabeled data successfully in an extremely efficient manner.[3]

[3] The work at hand is related to our previous work for the unsupervised case [9]; we would like to point out that the new (matrix) derivations for the semi-supervised stetting comprehend the old ones as a special case.

1.2 Notations

We use $[n]$ to denote the set $\{1,\ldots,n\}$. Further, the set of all $n \times m$ matrices with real coefficients is denoted by $\mathbb{R}^{n\times m}$. Given a matrix $\mathbf{M} \in \mathbb{R}^{n\times m}$, we denote the element in the i-th row and j-th column by $[\mathbf{M}]_{i,j}$. For two sets $R = \{i_1,\ldots,i_r\} \subseteq [n]$ and $S = \{k_1,\ldots,k_s\} \subseteq [m]$ 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 $\mathbf{M}_{R,[m]} = \mathbf{M}_R$. At last, we use y_i to denote the i-th coordinate of a vector $\mathbf{y} \in \mathbb{R}^n$.

2 Semi-Supervised Regularized Least-Squares

In supervised classification scenarios, a set $T_l = \{(\mathbf{x}'_1, y'_1),\ldots,(\mathbf{x}'_l, y'_l)\}$ of training patterns $\mathbf{x}'_i$ belonging to a set X with associated class labels $y'_i \in Y = \{-1,+1\}$ is given. For semi-supervised settings, we are additionally given a set $T_u = \{\mathbf{x}_1,\ldots,\mathbf{x}_u\} \subset X$ of training patterns without any class information.

2.1 Regularized Least-Squares Classification

Our approach can be seen as an extension of the *regularized least-squares classification* [12] technique. Both support vector machines and this concept belong to regularization problems of the form

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

where $\lambda > 0$ is a fixed real number, $\mathcal{L} : Y \times \mathbb{R} \to [0,\infty)$ is a *loss function* measuring the "performance" of the prediction function f on the training set, and $||f||^2_{\mathcal{H}}$ is the squared norm in a so-called *reproducing kernel Hilbert space* $\mathcal{H} \subseteq \mathbb{R}^X = \{f : X \to \mathbb{R}\}$ induced by a *kernel function* $k : X \times X \to \mathbb{R}$ [13,15]. By using the square loss $\mathcal{L}(y,t) = (y-t)^2$, one obtains

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

Due to the *representer theorem* [13], any minimizer $f^* \in \mathcal{H}$ of (2) has the form

$$f^*(\cdot) = \sum_{j=1}^{l} c_j k(\mathbf{x}'_j, \cdot) \tag{3}$$

with appropriate coefficients $\mathbf{c} = (c_1,\ldots,c_l)^{\mathrm{T}} \in \mathbb{R}^l$. Hence, by using $||f^*||^2_{\mathcal{H}} = \mathbf{c}^{\mathrm{T}}\mathbf{K}\mathbf{c}$ [13], where $\mathbf{K} \in \mathbb{R}^{l\times l}$ is the symmetric kernel matrix with entries of the form $[\mathbf{K}]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$, we 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 $\mathbf{y}' = (y'_1,\ldots,y'_l)^{\mathrm{T}}$. This optimization problem is convex and, thus, easy to solve given standard techniques for convex optimization [3].

2.2 Semi-Supervised Extension

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 training 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

$$\begin{aligned} \underset{f \in \mathcal{H},\, \mathbf{y} \in \{-1,+1\}^u}{\text{minimize}} \quad & \frac{1}{l}\sum_{i=1}^{l} \mathcal{L}\big(y_i', f(\mathbf{x}_i')\big) + \lambda' \frac{1}{u} \sum_{i=1}^{u} \mathcal{L}\big(y_i, f(\mathbf{x}_i)\big) + \lambda ||f||_{\mathcal{H}}^2 \\ \text{s.t.} \quad & \left| \frac{1}{u} \sum_{i=1}^{u} \max(0, y_i) - b_c \right| < \varepsilon, \end{aligned} \tag{5}$$

where $\lambda', \lambda > 0$ are cost parameters and where $\varepsilon > 0$. The last equation of the above task is called the *balance constraint*; it enforces the class ratio on the unlabeled part of the data to be approximately the same as an user-defined parameter b_c. We will denote assignments $\mathbf{y}$ fulfilling the latter constraint as *valid*. Again, due to the representer theorem [13], any optimal function $f^* \in \mathcal{H}$ for a *fixed* partition vector $\mathbf{y} \in \{-1,+1\}^u$ has the form

$$f^*(\cdot) = \sum_{j=1}^{l} c_j k(\mathbf{x}_j', \cdot) + \sum_{j=l+1}^{l+u} c_j k(\mathbf{x}_{j-l}, \cdot) \tag{6}$$

with appropriate coefficients $\mathbf{c} = (c_1, \ldots, c_{l+u})^{\mathrm{T}} \in \mathbb{R}^{l+u}$. Hence, by plugging in the square loss, one can reformulate the above optimization problem as

$$\begin{aligned} \underset{\mathbf{c} \in \mathbb{R}^n,\, \mathbf{y} \in \{-1,+1\}^n}{\text{minimize}} \quad & J(\mathbf{c}, \mathbf{y}) = (\mathbf{\Lambda y} - \mathbf{\Lambda K c})^{\mathrm{T}} (\mathbf{\Lambda y} - \mathbf{\Lambda K c}) + \lambda \mathbf{c}^{\mathrm{T}} \mathbf{K c} \\ \text{s.t.} \quad & \left| \frac{1}{u} \sum_{i=l+1}^{n} \max(0, y_i) - b_c \right| < \varepsilon, \\ \text{and} \quad & y_i = y_i' \text{ for } i = 1, \ldots, l, \end{aligned} \tag{7}$$

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

3 Local Search Revisited

We follow the optimization scheme proposed for the unsupervised case [9] and apply a local search strategy along with efficient matrix updates for the involved intermediate candidate solutions.

Algorithm 1 Local Search

```
Require: A set of labeled training patterns $\{(\mathbf{x}'_1, y'_1), \ldots, (\mathbf{x}'_l, y'_l)\}$, a set of unlabeled
         training patterns $\{\mathbf{x}_1, \ldots, \mathbf{x}_u\}$, and model parameters $\lambda', \lambda, b_c, \varepsilon$.
Ensure: An approximation $(\mathbf{c}^*, \mathbf{y})$ for the task (7).
 1: Initialize $\mathbf{y} \subseteq \{-1, +1\}^n$ (see text)
 2: $t = 0$
 3: while $t < \tau$ do
 4:     Generate $\bar{\mathbf{y}}$ by flipping a single coordinate $j \in \{l+1, \ldots, l+u\}$ of $\mathbf{y}$
 5:     if $F(\bar{\mathbf{y}}) < F(\mathbf{y})$ and $\bar{\mathbf{y}}$ is valid then
 6:         Replace $\mathbf{y}$ by $\bar{\mathbf{y}}$
 7:     end if
 8:     $t = t + 1$
 9: end while
10: Compute $\mathbf{c}^*$ for $\text{minimize}_{\mathbf{c} \in \mathbb{R}^n} J(\mathbf{c}, \mathbf{y})$
11: return $(\mathbf{c}^*, \mathbf{y})$
```

3.1 Local Search

The local search strategy is depicted in Algorithm 1: Starting with an initial candidate solution, we iterate over a sequence of τ iterations. For each iteration, we generate a new candidate solution by flipping a single coordinate and take the best performing solution out of the two current ones. The "quality" of an intermediate solution $\mathbf{y}$ is measured in terms of

$$F(\mathbf{y}) = \underset{\mathbf{c} \in \mathbb{R}^n}{\text{minimize}}\, J(\mathbf{c}, \mathbf{y}). \tag{8}$$

Once the overall process is finished, the best final candidate solution along with its corresponding vector $\mathbf{c}^*$ is returned. For the generation of the initial candidate solution, we resort to a well-known heuristic in this field [4,8,11], i.e., we initialize the unlabeled patterns with the predictions provided via a supervised model (which is trained on the labeled part and determined by (4)). If the balance constraint is not fulfilled after this assignment, we use the largest positive (and real-valued) predictions of the model as "positive" class assignments and the remaining ones as "negative" class assignments. We will briefly investigate the benefits and drawbacks of such an initialization strategy in Section 4.

3.2 Convex Intermediate Tasks

The intermediate optimization task (8) for a fixed partition vector $\mathbf{y}$ can be solved as follows: The function $G(\mathbf{c}) = J(\mathbf{c}, \mathbf{y})$ is differentiable with gradient

$$\nabla G(\mathbf{c}) = -2(\mathbf{\Lambda K})^{\mathrm{T}}(\mathbf{\Lambda y} - \mathbf{\Lambda K c}) + 2\lambda \mathbf{K c}.$$

Further, G is convex on $\mathbb{R}^n$ since the kernel matrix $\mathbf{K}$ and thus the Hessian

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

are positive semidefinite. Hence, $\nabla G(\mathbf{c}) = \mathbf{0}$ is a necessary and sufficient condition for optimality [3] and an optimal solution $\mathbf{c}^*$ for (8) can be obtained via

$$\mathbf{c}^* = \mathbf{\Lambda}(\mathbf{\Lambda}\mathbf{K}\mathbf{\Lambda} + \lambda\mathbf{I})^{-1}\mathbf{\Lambda}\mathbf{y} = \mathbf{\Lambda}\mathbf{G}\mathbf{\Lambda}\mathbf{y}. \tag{9}$$

with $\mathbf{G} = (\mathbf{\Lambda}\mathbf{K}\mathbf{\Lambda} + \lambda\mathbf{I})^{-1}$. Note that $\mathbf{\Lambda}\mathbf{K}\mathbf{\Lambda}$ is positive semidefinite since $\mathbf{K}$ is positive semidefinite; hence, $\mathbf{\Lambda}\mathbf{K}\mathbf{\Lambda} + \lambda\mathbf{I}$ is positive definite and thus invertible.

3.3 Efficient Matrix Updates

The recurrent computation of the objective values in Step 5 of Algorithm 1 is still cumbersome. However, similar to our previous work [9], it is possible to update these intermediate solutions efficiently when only one coordinate (or a constant number of coordinates) of an intermediate candidate solution is flipped: Let $\bar{\mathbf{y}}$ be the current candidate solution and let $\mathbf{y}$ be its predecessor. By plugging in Equation (9) into (8), one gets

$$\begin{aligned} F(\bar{\mathbf{y}}) &= (\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda}\mathbf{K}\mathbf{c}^*)^{\mathrm{T}}(\mathbf{\Lambda}\bar{\mathbf{y}} - \mathbf{\Lambda}\mathbf{K}\mathbf{c}^*) + \lambda(\mathbf{c}^*)^{\mathrm{T}}\mathbf{K}\mathbf{c}^* \\ &= \bar{\mathbf{y}}^{\mathrm{T}}\mathbf{\Lambda}\underbrace{\left(\mathbf{I} - \overline{\mathbf{K}}\mathbf{G} - \mathbf{G}\overline{\mathbf{K}} + \mathbf{G}\overline{\mathbf{K}}\overline{\mathbf{K}}\mathbf{G} + \lambda\mathbf{G}\overline{\mathbf{K}}\mathbf{G}\right)}_{=:\mathbf{H}}\mathbf{\Lambda}\bar{\mathbf{y}}, \end{aligned} \tag{10}$$

where $\overline{\mathbf{K}} = \mathbf{\Lambda}\mathbf{K}\mathbf{\Lambda}$. Now note that, for a given coordinate j to be flipped, one can update the right hand side via

$$\mathbf{H}\mathbf{\Lambda}\bar{\mathbf{y}} = \mathbf{H}\mathbf{\Lambda}\mathbf{y} - 2y_j(\mathbf{H}\mathbf{\Lambda})_{[n],\{j\}} \tag{11}$$

in $\mathcal{O}(n)$ time, assuming that the matrix $\mathbf{H}\mathbf{\Lambda} \in \mathbb{R}^{n\times n}$ and the information $\mathbf{H}\mathbf{\Lambda}\mathbf{y} \in \mathbb{R}^n$ for the predecessor are available (in memory). Further, since $\mathbf{\Lambda}$ is a diagonal matrix and since $\mathbf{H}\mathbf{\Lambda}\bar{\mathbf{y}} \in \mathbb{R}^n$ is a vector, the remaining operations for computing (10) can be performed in $\mathcal{O}(n)$ time too. Thus, by spending $\mathcal{O}(n^3)$ time for the initialization of the corresponding matrices, the solutions of the intermediate tasks can be "updated" in $\mathcal{O}(n)$ time per iteration:

Theorem 1. *One can compute $F(\bar{\mathbf{y}})$ in Step 5 of Algorithm 1 in $\mathcal{O}(n)$ time. Further, $\mathcal{O}(n^3)$ preprocessing time and $\mathcal{O}(n^2)$ space is needed.*

We would like to point out that it is possible to integrate the so-called *Nyström approximation* in an efficient manner. More specifically, one can replace the original kernel matrix $\mathbf{K}$ by $\widetilde{\mathbf{K}} = (\mathbf{K}_R)^{\mathrm{T}}(\mathbf{K}_{R,R})^{-1}\mathbf{K}_R$, where $R = \{i_1, \ldots, i_r\} \subseteq [n]$ is a subset of indices. The acceleration can be done in an analogous way to the unsupervised case [9] by making use of the low-rank nature of $\widetilde{\mathbf{K}}$; the longish derivations are omitted due to lack of space.

Theorem 2. *By applying the approximation scheme, one can compute $F(\bar{\mathbf{y}})$ in Step 5 of Algorithm 1 in $\mathcal{O}(r)$ time. Further, $\mathcal{O}(nr^2)$ preprocessing time and $\mathcal{O}(nr)$ space is needed.*

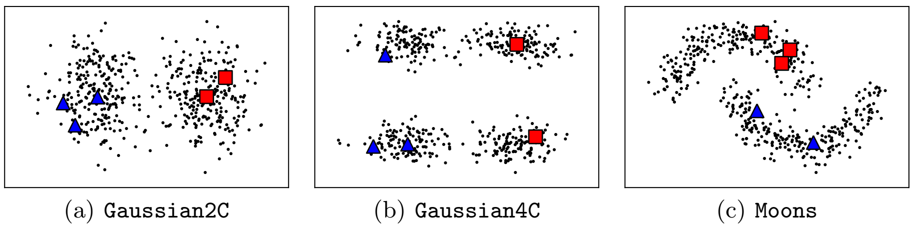


Fig. 2. The (red) squares and (blue) triangles depict the labeled part of the data; the remaining (small) points correspond to the unlabeled part.

4 Experiments

We will now describe the experimental setup and the outcome of the evaluation.

4.1 Experimental Setup

Our approach (S^{2}RLSC) is implemented in `Python` using the `Numpy` package. The runtime analyses are performed on a `2.66 GHz Intel Core™ Quad` PC running `Ubuntu 10.04`.

Data Sets The first artificial data set is composed of two Gaussian clusters; to generate it, we draw $n/2$ points from each of two 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 Figure 2 (a). 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, we draw $n/4$ points from each of four Gaussian distributions $X_i \sim \mathcal{N}(\mathbf{m}_i, \mathbf{I})$, where

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

see Figure 2 (b). 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`. Finally, we consider the well-known `Moons` data set with $n = 500$ points and $\dim = 2$, see Figure 2 (c). In addition to these artificial data sets, we make use of the `USPS` [10] data set, where `USPS(i,j)` is used to denote a single binary classification task, see Figure 3. 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 data set, where l, u, t denotes the number of labeled, unlabeled, and test patterns (e.g. `Gaussian2C[l=25,u=225,t=250]`).

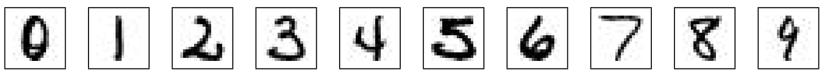


Fig. 3. The `USPS` data set [10] containing images of handwritten digits. Each digit is represented by a 16×16 gray scale image (8 bits).

Model Selection 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 is more difficult due to the lack of labeled data and is widely considered to be an open issue [5]. Due to this model selection problem, we consider two scenarios to select the (non-fixed) parameters: The first one is a *non-realistic* scenario where we make use of the test set to evaluate the model performance.[4] Note that by making use of the test set (with a large amount of labels), one can first evaluate the "flexibility" of the model, i.e., one 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. 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 5-fold cross-validation).

Parameters In both scenarios, we first tune the non-fixed parameters via grid search and subsequently retrain the final model on the training set with the best performing set of parameters. As similarity measures, we consider both a linear kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ and a RBF kernel $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2})$ with kernel width σ. To tune the cost parameters λ and λ', we consider a small grid $(\lambda, \lambda') \in \{2^{-10}, \ldots, 2^{10}\} \times \{0.01, 1, 100\}$ of parameters. Concerning the balance constraint, we set b_c to an estimate obtained from all available labels and fix $\varepsilon = 0.1$. The iterative process of the local search is stopped if no changes have occured for n consecutive iterations; further, a round-robin scheme is used to select the coordinates to be flipped per iteration.

Competing Approaches We use the regularized least-squares classifier (`RLSC`) as supervised model; to tune the parameter λ, we perform a grid search with $\lambda \in \{2^{-10}, \ldots, 2^{10}\}$. As semi-supervised competitor, we resort to the `UniverSVM` approach [8], which depicts one of the state-of-the-art semi-supervised support vector machine implementations. The parameters are also tuned via grid search with $(C, C^*) \in \{2^{-10}, \ldots, 2^{10}\} \times \{\frac{0.01}{u}, \frac{1.0}{u}, \frac{100.0}{u}\}$. The ratio between the two classes is provided to the algorithm via the `-w` option; except for the `-S` option (which we set to -0.3), the default values for the remaining parameters are used.

4.2 Results

We will now provide the outcome of the experiments conducted.

[4] This setup is usually considered in related evaluations [5].

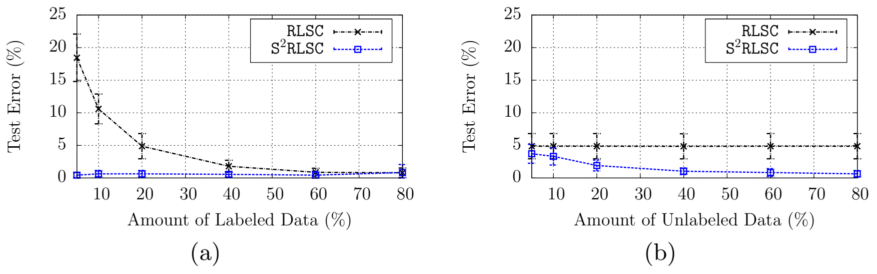


Fig. 4. Our semi-supervised approach can successfully incorporate unlabeled data to improve the generalization performance, see Figure (a). However, sufficient unlabeled data is needed as well for the learner to yield a satisfying performance, see Figure (b).

Amount of Data 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 S^{2}RLSC approach, we consider the Gaussian2C data set and vary the amount of labeled and unlabeled data. For this experiment, we consider the non-realistic scenario and use the supervised RLSC approach as baseline. 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 Figure 4 (a), the result of this experiment is shown: Even with little labeled data, the semi-supervised approach performs better compared to the supervised one. Now, let us fix the amount of labeled data to 20% and and let us vary the amount of unlabeled data from 5% to 80% with respect to (the size of) the training set, see Figure 4 (b). Clearly, the semi-supervised approach is only capable of generating an appropriate model once sufficient unlabeled data is given.

Classification Performance We consider two different ratios of labeled and unlabeled data for each particular data set. The average test errors and the 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. The results are given in Table 1. It can be clearly seen that the semi-supervised approaches yield better results compared to the supervised model. The results also indicate that by taking more labeled data into account, the gap between the performances of the supervised approach and the semi-supervised approaches becomes smaller. Hence, both semi-supervised approaches can incorporate the unlabeled data successfully. Compared to the results for the non-realistic scenario, the performances for the realistic one are clearly worse. Hence, the lack of labeled data for model selection as well as a (possibly) bad estimate for the balance parameter b_c seem to have a negative influence on the final classification performance. The overall classification performances of both semi-supervised approaches is comparable; the S^{2}RLSC approach seems work slightly better on the USPS data set whereas

Data Set	RLSC		UniverSVM		S^2RLSC	
	non-realistic	realistic	non-realistic	realistic	non-realistic	realistic
Gaussian2C[l=25,u=225,t=250]	10.6 ± 2.3	11.4 ± 2.4	1.0 ± 0.5	**1.8 ± 0.9**	**0.6 ± 0.5**	6.6 ± 2.6
Gaussian2C[l=50,u=200,t=250]	4.9 ± 1.9	5.3 ± 2.1	1.0 ± 0.4	**1.8 ± 0.8**	**0.6 ± 0.5**	3.3 ± 2.6
Gaussian4C[l=25,u=225,t=250]	16.1 ± 7.0	16.5 ± 6.8	7.6 ± 12.2	13.3 ± 15.2	**6.8 ± 12.5**	**12.0 ± 13.6**
Gaussian4C[l=50,u=200,t=250]	6.1 ± 2.1	6.5 ± 1.9	1.6 ± 0.8	**2.5 ± 1.5**	**0.8 ± 0.6**	4.1 ± 2.4
USPS(2,5)[l=16,u=806,t=823]	7.9 ± 2.9	9.3 ± 2.3	3.2 ± 0.5	9.0 ± 5.6	**2.9 ± 0.4**	**6.3 ± 2.8**
USPS(2,5)[l=32,u=790,t=823]	4.4 ± 0.5	5.8 ± 1.6	3.2 ± 0.5	**5.7 ± 1.8**	**2.9 ± 0.4**	6.4 ± 2.0
USPS(2,7)[l=17,u=843,t=861]	3.6 ± 2.4	4.0 ± 2.4	1.5 ± 0.3	6.1 ± 5.3	**1.0 ± 0.1**	**1.4 ± 0.2**
USPS(2,7)[l=34,u=826,t=861]	2.2 ± 0.7	3.1 ± 1.7	1.4 ± 0.2	3.4 ± 2.4	**1.0 ± 0.1**	**1.3 ± 0.2**
USPS(3,8)[l=15,u=751,t=766]	9.8 ± 6.6	11.0 ± 6.4	4.8 ± 1.1	8.7 ± 3.9	**4.1 ± 1.3**	**6.2 ± 2.7**
USPS(3,8)[l=30,u=736,t=766]	6.3 ± 2.0	7.7 ± 1.4	4.0 ± 0.1	7.1 ± 1.8	**3.9 ± 1.2**	**6.4 ± 2.8**
USPS(8,0)[l=22,u=1108,t=1131]	4.8 ± 1.9	6.6 ± 3.3	1.7 ± 0.7	3.2 ± 2.2	**1.2 ± 0.2**	**1.8 ± 0.6**
USPS(8,0)[l=45,u=1085,t=1131]	2.6 ± 0.8	3.3 ± 0.9	1.3 ± 0.4	3.3 ± 1.8	**1.1 ± 0.2**	**2.0 ± 0.5**

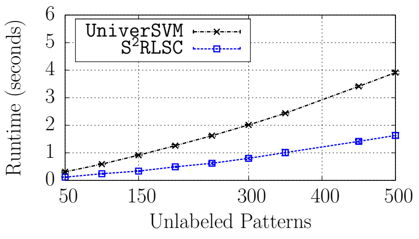
Table 1. The table shows the classification performance of all approaches on all data sets considered. Clearly, both semi-supervised approaches can successfully incorporate unlabeled data to improve the performance on the test set.

the `UniverSVM` approach works slightly better on the artificial data sets in the realistic scenario.

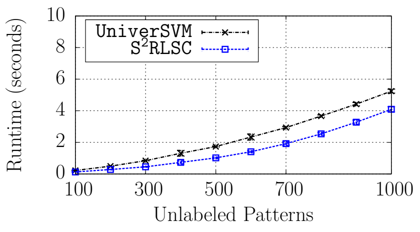
Computational Considerations Let us finally analyze the practical runtimes of the considered semi-supervised approaches. Naturally, these runtimes depend heavily on the particular implementation and the used programming language. Thus, the provided results shall only give a rough idea of the runtimes needed in practice (e. g., for generating Table 1). For this sake, we consider the `Gaussian2C` and the `USPS(8,0)` data sets. Again, we resort to a linear kernel and fix the model parameters ($\lambda = 1$ and $\lambda' = 1$ for S^2RLSC and $C = 1$ and $C^* = 1$ for `UniverSVM`). Further, the amount of labeled patterns is fixed to $l = 50$ and $l = 22$, respectively. In Figure 5, the runtime behavior for both approaches for a varying amount of unlabeled patterns is given. The plots indicate a comparable runtime performance of both approaches. Thus, the computational shortcut renders our simple local search scheme competitive to state-of-the-art implementations.[5]

More Optimization Most of the related optimization schemes use the labeled part to obtain an initial "guess" via a supervised model (e.g., `UniverSVM`) which is then improved iteratively (i. e., no restarts are performed rendering such approaches *deterministic*). While such a strategy reduces the practical runtime, a natural question is whether the results can be improved by putting more effort into optimization. To exemplarily investigate this question, we consider the `Moons` data set with RBF kernel as similarity measure (and kernel width $\sigma = 0.1s$, where the value s is an estimate of the maximum distance between any pair of samples). Further, we fix the two cost parameters to $\lambda = 2^{-10}$ and

[5] The `UniverSVM` implementation is based on `C` whereas S^2RLSC is implemented in `Python`; in general, code in `Python` is said to be much slower than pure `C` code and we expect a considerable speed-up with a pure `C` implementation of our approach.



(a) `Gaussian2C`



(b) `USPS(8,0)`

Fig. 5. Runtimes of the semi-supervised competitors on the `Gaussian2C` and the `USPS(8,0)` data sets; in both cases, the average runtimes of 10 executions are reported.

$\lambda' = 0.1$. The comparison between the single restart approach (used above) and a multiple restart scheme (with 200 restarts and random initialization) is shown in Figure 6. Clearly, the single restart variant fails on this particular problem instance while using more restarts leads to the global optimum. Thus, there are problem instances where the heuristic of improving a single guess fails and where it pays off to spend more effort into optimization. We would like to point out that for such settings, the proposed approach is well suited since performing restarts is very cheap (preprocessing has only to be done once).

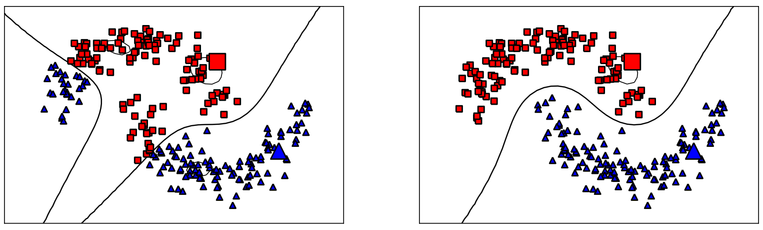
5 Conclusions and Outlook

We proposed an optimization framework for semi-supervised regularized least-squares classification. The key idea consists in making use of a simple local search strategy which is accelerated by means of efficient matrix updates for the intermediate candidate solutions. Our experimental evaluation demonstrates that such a simple (but accelerated) search scheme yields classification results comparable to state-of-the-art methods in an extremely efficient manner.

We think that the derivations presented in this work are extendible and applicable in various directions: For instance, Adankon *et al.* [1] have recently proposed an alternating optimization scheme (also based on the square loss) which can potentially be accelerated by means of the matrix-based updates given here. Further, we expect the computational shortcuts to be beneficial in the context of (exact) branch-and-bound strategies for the task at hand like the one proposed by Chapelle *et al.* [6]. We plan to investigate these issues in near future.

References

1. M. Adankon, M. Cheriet, and A. Biem. Semisupervised least squares support vector machine. *IEEE Transactions on Neural Networks*, 20(12):1858–1870, 2009.
2. T. D. Bie and N. Cristianini. Convex methods for transduction. In *Adv. in Neural Information Proc. Systems 16*, pages 73–80. MIT Press, 2004.
3. S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge Uni. Press, 2004.



(a) $\texttt{S}^2\texttt{RLSC (single)}$ (21.3 ± 14.9) (b) $\texttt{S}^2\texttt{RLSC (multiple)}$ (0.0 ± 0.0)

Fig. 6. If only one restart (with initial guess) is performed, the local search approach can converge to a (bad) local optimum, see Figure (a). Performing sufficient restarts (with random initial candidate solutions) yields good solutions, see Figure (b). The average test error (with one standard deviation) on over 10 random partitions is given.

4. O. Chapelle, M. Chi, and A. Zien. A continuation method for semi-supervised SVMs. In *Proc. Int. Conf. on Machine Learning*, pages 185–192, 2006.
5. O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning.* MIT Press, Cambridge, MA, 2006.
6. O. Chapelle, V. Sindhwani, and S. S. Keerthi. Branch and bound for semi-supervised support vector machines. In *Adv. in Neural Information Proc. Systems 19*, pages 217–224. MIT Press, 2007.
7. O. Chapelle and A. Zien. Semi-supervised classification by low density separation. In *Proc. 10th Int. Workshop on Artificial Intell. and Statistics*, pages 57–64, 2005.
8. R. Collobert, F. Sinz, J. Weston, and L. Bottou. Trading convexity for scalability. In *Proc. International Conference on Machine Learning*, pages 201–208, 2006.
9. F. Gieseke, T. Pahikkala, and O. Kramer. Fast evolutionary maximum margin clustering. In *Proc. Int. Conf. on Machine Learning*, pages 361–368, 2009.
10. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer, 2009.
11. T. Joachims. Transductive inference for text classification using support vector machines. In *Proc. Int. Conf. on Machine Learning*, pages 200–209, 1999.
12. R. Rifkin, G. Yeo, and T. Poggio. Regularized least-squares classification. In *Adv. in Learning Theory: Methods, Models and Applications.* IOS Press, 2003.
13. B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In D. P. Helmbold and B. Williamson, editors, *Proc. 14th Annual Conf. on Computational Learning Theory*, pages 416–426, 2001.
14. V. Sindhwani, S. Keerthi, and O. Chapelle. Deterministic annealing for semi-supervised kernel machines. In *Proc. Int. Conf. on Machine Learning*, pages 841–848, 2006.
15. I. Steinwart and A. Christmann. *Support Vector Machines.* Springer-Verlag, New York, NY, USA, 2008.
16. V. Vapnik and A. Sterin. On structural risk minimization or overall risk in a problem of pattern recognition. *Aut. and Remote Control*, 10(3):1495–1503, 1977.
17. K. Zhang, J. T. Kwok, and B. Parvin. Prototype vector machine for large scale semi-supervised learning. In *Proceedings of the International Conference on Machine Learning*, 2009.
18. X. Zhu and A. B. Goldberg. *Introduction to Semi-Supervised Learning.* Morgan and Claypool, 2009.