Fast and Simple Gradient-Based Optimization for Semi-Supervised Support Vector Machines☆

Fabian Gieseke*,[a], Antti Airola[b,c], Tapio Pahikkala[b,c], Oliver Kramer[a]

[a] *Computer Science Department, Carl von Ossietzky Universität Oldenburg, 26111 Oldenburg, Germany*
[b] *Department of Information Technology, 20014, University of Turku, Finland*
[c] *Turku Centre for Computer Science (TUCS), Joukahaisenkatu 3-5 B, 20520 Turku, Finland*

Abstract

One of the main learning tasks in machine learning is the one of classifying data items. The basis for such a task is usually a training set consisting of labeled patterns. In real-world settings, however, such labeled data are usually scarce, and the corresponding models might yield unsatisfying results. Unlabeled data, on the other hand, can often be obtained in huge quantities without much additional effort. A prominent research direction in the field of machine learning are semi-supervised support vector machines. This type of binary classification approach aims at taking the additional information provided by the unlabeled patterns into account to reveal more information about the structure of the data at hand. In some cases, this can yield significantly better classification results compared to a straightforward application of supervised models. One drawback, however, is the fact that generating such models requires solving difficult non-convex optimization tasks. In this work, we present a simple but effective gradient-based optimization framework to address the induced problems. The resulting method can be implemented easily using black-box optimization engines and yields excellent classification and runtime results on both sparse and non-sparse data sets.

Key words: Semi-Supervised Support Vector Machines, Non-Convex Optimization, Quasi-Newton Methods

1. INTRODUCTION

One of the most important machine learning tasks is classification. If sufficient labeled training data are given, there exists a variety of techniques like the *k-nearest neighbor*-classifier or *support vector machines* (SVMs) [2, 3] to address such a task. However, labeled data are often rare in real-world applications. One active research field in machine learning is *semi-supervised learning* [4, 5]. In contrast to supervised methods, the latter class of techniques takes both labeled and unlabeled data into account to construct appropriate models. A well-known concept in this field are *semi-supervised support vector machines* (S^3VMs) [6, 7, 8], which depict the direct extension of support vector machines to semi-supervised learning scenarios. The key idea is depicted in Figure 1: The aim of a standard support vector machine consists in finding a hyperplane which separates both classes well such that the margin is maximized. It is obvious that, in case of lack of labeled data, suboptimal models might be obtained, see Figure 1 (a). Its semi-supervised variant aims at taking the unlabeled patterns into account by searching for a partition (into two classes) such that a *subsequent* application of a modified support vector machine leads to the best result. Under certain conditions, unlabeled data can provide valuable information, see Figure 1 (b). While being very appealing from a practical point of view, semi-supervised support vector machines lead to a combinatorial optimization task that is difficult to approach.

The original problem formulation of semi-supervised support vector machines was given by Vapnik and Sterin [8] under the name of *transductive support vector machines*. From an optimization point of view, the first approaches have been proposed in the late nineties by Joachims [7] and Bennet and Demiriz [6]. In general, there are two lines of research, namely (a) *combinatorial* and (b) *continuous* optimization schemes. The brute-force approach (which tests every possible partition), for instance, is among the combinatorial schemes since it aims at directly finding a good assignment for the unknown labels.

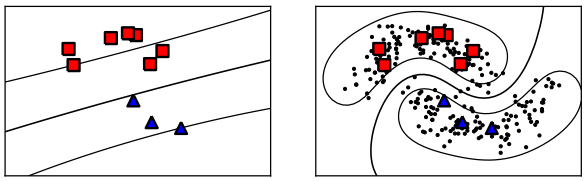
1.1. Related Work

For both the combinatorial and the continuous research direction, a variety of different techniques has been pro-

☆This work depicts an extended version of the associated conference paper that has been presented at the *1st International Conference on Pattern Recognition Applications and Methods* [1]. It contains additional theoretical derivations related to incorporating an offset term and a balancing constraint. Moreover, the experimental evaluation has been extended by adding two more semi-supervised competitors as well as a variety of high-dimensional sparse data sets.

*Corresponding author. Tel.: +494417984374 ; fax: +494417982756

Email addresses: f.gieseke@uni-oldenburg.de (Fabian Gieseke), antti.airola@utu.fi (Antti Airola), tapio.pahikkala@utu.fi (Tapio Pahikkala), oliver.kramer@uni-oldenburg.de (Oliver Kramer)



(a) SVM (b) S^3VM

Figure 1: The concepts of support vector machines and their extension to semi-supervised learning settings. Labeled patterns are depicted as red squares and blue triangles and unlabeled patterns as black points, respectively.

posed in recent years. The former one is usually addressed by label-switching strategies [7, 9, 10] or by reformulating the original task as semi-definite programming problem [11, 12]. Further, since both real and integer variables are present in the optimization task (see below), mixed-integer programming solvers can be applied to compute optimal solutions up to machine precision [6]. Another way to obtain optimal solutions are branch and bound frameworks, see Chapelle *et al.* [13] for an appropriate algorithm.

The continuous optimization perspective leads to a real-valued but non-convex task (see below). Among the first schemes that considered this perspective was the gradient descent framework of Chapelle and Zien [14], which was based on the replacement of the original loss functions by appropriate surrogates. Similar ideas led to the continuation framework [15], to deterministic annealing methods [16, 10], and to the use of the (constrained) concave-convex procedure [17, 18, 19]. An approach closely related to the one proposed in this work is the quasi-Newton framework proposed by Reddy *et al.* [20]; however, they do not consider differentiable surrogates and therefore apply more complicated sub-gradient methods.

Despite the methods mentioned above, a variety of other semi-supervised support vector machine variants have been proposed in the literature including, e.g., graph-based methods [21]. Due to lack of space, we refer to Chapelle *et al.* [4, 22] and Zhu and Goldberg [5] for comprehensive surveys. It is worth pointing out that support vector machines can also be extended to unsupervised learning settings (without any labeled patterns at all) in a very similar kind of way. This variant is known as *maximum margin clustering* and has received a considerable interest in recent years [23, 24, 25, 26, 27, 28].

1.2. Contribution

In this work, we will show that quasi-Newton schemes [29] along with direct computational shortcuts for sparse and non-sparse data depict simple but very effective approaches for the task at hand. In particular, we make use of an appropriate differentiable surrogate of the original objective and show that one can directly obtain computational shortcuts for non-sparse data (and arbitrary kernels) via the subset of regressors [30] scheme, and for sparse data (and the linear kernel) by taking advantage of the explicit structure of the objective function and its gradient. The induced optimization approaches are conceptually very simple and can be implemented easily via standard black-box optimization tools.[1]

As part of the contribution, we provide a detailed experimental evaluation and compare both the classification and runtime performance of our implementation with state-of-the-art semi-supervised support vector machine implementations on a variety of sparse and non-sparse data sets. The results clearly indicate the usability and effectiveness of our implementation.

1.3. Notations

We use $[m]$ to denote the set $\{1, \dots, m\}$. Given a vector $\mathbf{y} \in \mathbb{R}^n$, we use y_i to denote its i-th coordinate. 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 $R = \{i_1, \dots, i_r\} \subseteq [m]$ and $S = \{k_1, \dots, k_s\} \subseteq [n]$ 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 $\mathbf{M}_{R,[n]} = \mathbf{M}_R$. All vectors are assumed to be column vectors and the superscript T is used to denote the transpose of a matrix or a vector, i. e., $\mathbf{y}^{\mathrm{T}}$ is a row vector and $\mathbf{M}^{\mathrm{T}} \in \mathbb{R}^{n \times m}$ is the transpose of the matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$.

2. CLASSIFICATION TASK

In the following, we will consider a set $T_l = \{(\mathbf{x}_1, y'_1), \dots, (\mathbf{x}_l, y'_l)\}$ of labeled patterns and a set $T_u = \{\mathbf{x}_{l+1}, \dots, \mathbf{x}_{l+u}\} \subset X$ of unlabeled training patterns that belong to an arbitrary set X.

2.1. Support Vector Machines

The concept of support vector machines can be seen as instance of regularization problems of the form

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

where $\lambda > 0$ is a fixed real number, $\mathcal{L} : Y \times \mathbb{R} \to [0, \infty)$ is a *loss function* 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* $k : X \times X \to \mathbb{R}$ [3]. Here, the first term measures the loss caused by the prediction function on the labeled training set and the second one penalizes complex functions. Plugging in different loss functions leads to various models; one of the most popular choices is the *hinge loss* $\mathcal{L}(y, t) = \max(0, 1 - yt)$, which yields the original definition of support vector machines [3, 31], see Figure 2 (a).[2]

[1] The code can be obtained from the authors upon request.

[2] The latter formulation does not include a bias term $b \in \mathbb{R}$, which addresses translated data. For complex kernel functions like the RBF

2.2. Semi-Supervised SVMs

Given the additional set $T_u = \{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}\} \subset X$ of unlabeled training patterns, semi-supervised support vector machines [6, 7, 8] aim at finding 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 $\mathbf{y}^* = (y^*_{l+1}, \ldots, y^*_{l+u})^{\mathrm{T}} \in \{-1, +1\}^u$ that are optimal with respect to $\min_{f \in \mathcal{H},\, \mathbf{y} \in \{-1,+1\}^u} J(f, \mathbf{y})$ where $J(f, \mathbf{y}) =$

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

Here, $\lambda', \lambda > 0$ are user-defined parameters and $\mathcal{L}^1 : \mathbb{R} \times \mathbb{R} \to [0, \infty)$ a loss function. Thus, the main task consists in finding the optimal assignment vector $\mathbf{y}$ for the unlabeled part; the combinatorial nature of this task renders the optimization problem difficult to solve.

When using the hinge loss $\mathcal{L}^1(y, f(\mathbf{x})) = \max(0, 1 - yf(\mathbf{x}))$ for the above setting, the optimal assignments for the vector $\mathbf{y}$ and a *fixed* $f \in \mathcal{H}$ are given by $y_i = \operatorname{sgn}(f(\mathbf{x}_i))$ [14]. Thus, the induced loss $\mathcal{L}^2(f(\mathbf{x})) := \max(0, 1 - |f(\mathbf{x})|)$ on the unlabeled patterns (called the *effective loss*) penalizes predictions around the origin, i. e., the overall loss increases if the decision function f passes through these patterns, see Figure 2 (b). By applying the *representer theorem* [31] for latter task, it follows that an optimal solution $f \in \mathcal{H}$ is of the form

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

with coefficients $\mathbf{c} = (c_1, \ldots, c_n)^{\mathrm{T}} \in \mathbb{R}^n$ and $n = l + u$. Hence, one obtains a continuous optimization task that consists in finding the optimal coefficient vector $\mathbf{c} \in \mathbb{R}^n$.

3. GRADIENT-BASED OPTIMIZATION

One of the main drawbacks of the hinge loss is that the induced objective function is not differentiable, which rules out the use of some of the most mature off-the-shelf optimization tools. In this section, we will propose differentiable surrogates for the objective and will show how to efficiently apply a special gradient-based optimization framework [29] that is well-suited for the task at hand.

3.1. Differentiable Surrogates

Since the original objective function is not differentiable, we follow Chapelle and Zien [14] and propose the following (similar but slightly different) surrogate for it, see Figure 2. Here, the differentiable replacement for the hinge loss is the *modified logistic loss* [32]

$$\mathcal{L}^1(y, f(\mathbf{x})) = \frac{1}{\gamma} \log\left(1 + \exp(\gamma(1 - y'_i f(\mathbf{x}_i)))\right),$$

and the replacement for the unlabeled part is [14]

$$\mathcal{L}^2(f(\mathbf{x})) = \exp(-3(f(\mathbf{x}_{l+i}))^2).$$

Substituting these loss functions and (3) into (2), the objective to be minimized becomes

$$\begin{aligned} F_{\lambda'}(\mathbf{c}) = {} & \frac{1}{l} \sum_{i=1}^{l} \frac{1}{\gamma} \log\left(1 + \exp\left(\gamma(1 - y'_i \sum_{i=1}^{n} c_i k\left(\mathbf{x}_i, \cdot\right)\right)\right) \\ & + \frac{\lambda'}{u} \sum_{i=l+1}^{n} \exp\left(-3\left(\sum_{i=1}^{n} c_i k\left(\mathbf{x}_i, \cdot\right)\right)^2\right) \\ & + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) \end{aligned} \tag{4}$$

using $||f||^2_{\mathcal{H}} = \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j)$ [31]. The next theorem shows that both a function and a gradient call can be performed efficiently:

Theorem 1. *For a given $\mathbf{c} \in \mathbb{R}^n$, one can compute the objective $F_{\lambda'}(\mathbf{c})$ and the gradient $\nabla F_{\lambda'}(\mathbf{c})$ in $\mathcal{O}(n^2)$ time. The overall space consumption is $\mathcal{O}(n^2)$.*

Proof. The gradient is given by

$$\nabla F_{\lambda'}(\mathbf{c}) \quad = \quad \mathbf{K}\mathbf{a} + 2\lambda \mathbf{K}\mathbf{c} \tag{5}$$

with $\mathbf{a} \in \mathbb{R}^n$ and

$$a_i = \begin{cases} -\dfrac{1}{l} \cdot \dfrac{\exp(\gamma(1 - f(\mathbf{x}_i) y'_i))}{1 + \exp(\gamma(1 - f(\mathbf{x}_i) y'_i))} \cdot y'_i & \text{for } i \leq l \\ -\dfrac{6\lambda'}{u} \cdot \exp\left(-3(f(\mathbf{x}_i))^2\right) \cdot f(\mathbf{x}_i) & \text{for } i > l \end{cases}.$$

Since all predictions $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)$ can be computed in $\mathcal{O}(n^2)$ total time, one can compute the vector $\mathbf{a} \in \mathbb{R}^n$

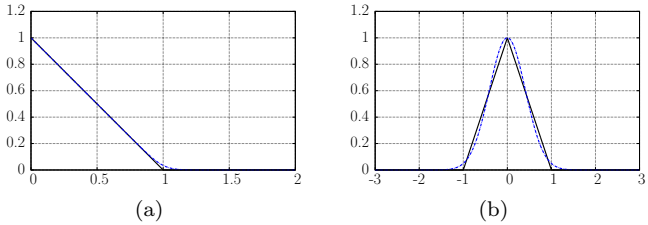


Figure 2: The hinge loss $\mathcal{L}(y, t) = \max(0, 1 - yt)$ and its differentiable surrogate $\mathcal{L}(y, t) = \frac{1}{\gamma} \log(1 + \exp(\gamma(1 - yt)))$ with $y = +1$ and $\gamma = 20$ are shown in Figure (a). The effective hinge loss function $\mathcal{L}(t) = \max(0, 1 - |t|)$ along with its differentiable surrogate $\mathcal{L}(t) = \exp(-st^2)$ with $s = 3$ are shown in Figure (b).

kernel, adding this bias term does not yield any known advantages, both from a theoretical as well as practical point of view [3]. In the remainder of this work, we will mostly omit the bias term for the sake of exposition; however, such a bias term can be explicitly incorporated into the optimization frameworks presented in this work, as we will show below.

Algorithm 1 QN-S^{3}VM

Require: A labeled training set $T_l = \{(\mathbf{x}_1, y'_1), \ldots, (\mathbf{x}_l, y'_l)\}$, an unlabeled training set $T_u = \{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_n\}$, model parameters λ', λ, an initial (positive definite) inverse Hessian approximation $\mathbf{H}_0$, and a sequence $0 < \alpha_1 < \ldots < \alpha_\tau$.

```
Initialize c_0 via supervised model.
for i = 1 to τ do
    k = 0
    while termination criteria not fulfilled do
        Compute search direction p_k via (6)
        Update c_{k+1} = c_k + β_k p_k
        Update H_{k+1} via (7)
        k = k + 1
    end while
    c_0 = c_k
end for
```

and therefore the objective and the gradient in $\mathcal{O}(n^2)$ time too. The space requirements are dominated by the kernel matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ in the above setting.[3] □

Note that numerical instabilities can occur when evaluating $\exp(\gamma(1 - f(\mathbf{x}_i)y'_i))$ for a function or a gradient call. However, one can deal with these degeneracies in a safe way since $\log(1 + \exp(t)) - t \to 0$ and $\frac{\exp(t)}{1+\exp(t)} - 1 \to 0$ converge rapidly for $t \to \infty$. Thus, each function and gradient evaluation can be performed spending $\mathcal{O}(n^2)$ time in a numerically stable manner. Throughout the work, the parameter γ is fixed to 20.

3.2. Quasi-Newton Framework

One of the most popular quasi-Newton schemes is the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) [29] method, which we will now sketch in the context of the given task. The overall algorithmic framework is given in Algorithm 1: The initial candidate solution is obtained via Equation (4) while ignoring the (non-convex) unlabeled part (i. e., $\lambda' = 0$). The influence of the unlabeled part is then increased gradually via the sequence $\alpha_1, \ldots, \alpha_\tau$.[4] For each parameter α_i, a standard BFGS optimization phase is performed, i.e., a sequence $\mathbf{c}_{k+1} = \mathbf{c}_k + \beta_k \mathbf{p}_k$ of candidate solutions is generated, where $\mathbf{p}_k$ is computed via

$$\mathbf{p}_k = -\mathbf{H}_k \nabla F_{\alpha_i \cdot \lambda'}(\mathbf{c}_k) \tag{6}$$

and where the step length β_k is computed via line search. The approximation $\mathbf{H}_k$ of the inverse Hessian is then updated via

$$\mathbf{H}_{k+1} = (I - \rho_k \mathbf{s}_k \mathbf{z}_k^{\mathrm{T}}) \mathbf{H}_k (I - \rho_k \mathbf{z}_k \mathbf{s}_k^{\mathrm{T}}) + \rho_k \mathbf{s}_k \mathbf{s}_k^{\mathrm{T}} \tag{7}$$

with $\mathbf{z}_k = \nabla F_{\alpha_i \cdot \lambda'}(\mathbf{c}_{k+1}) - \nabla F_{\alpha_i \cdot \lambda'}(\mathbf{c}_k)$, $\mathbf{s}_k = \mathbf{c}_{k+1} - \mathbf{c}_k$, and $\rho_k = (\mathbf{z}_k^{\mathrm{T}} \mathbf{s}_k)^{-1}$. New candidate solutions are generated as long as a convergence criterion is fulfilled (e. g., as long as $||\nabla F_{\alpha_i \cdot \lambda'}(\mathbf{c}_k)|| > \varepsilon$ is fulfilled for a small $\varepsilon > 0$ or as long as the number of iterations is smaller than a used-defined number). As initial approximation, one can resort to $\mathbf{H}_0 = \gamma \mathbf{I}$ for $\gamma > 0$. An important property of the update scheme is that it preserves the positive definiteness of the inverse Hessian approximations [29].

3.3. Computational Speed-Ups

Computational bottlenecks arise when applying the optimization engines described above: Firstly, the recurrent computation of the objective and gradient is cumbersome. Secondly, the approximation of the Hessian's inverse is, in general, not sparse. We will now show how to alleviate these two problems.

3.3.1. Linear Kernel and Sparse Data

For the special case of a linear kernel, one can obtain computational savings in the following way: Assume that we are given patterns in $X = \mathbb{R}^d$ and let $\mathbf{X} \in \mathbb{R}^{n \times d}$ denote the data matrix containing the training patterns as rows. Since one can write the kernel matrix as $\mathbf{K} = \mathbf{X}\mathbf{X}^{\mathrm{T}} \in \mathbb{R}^{n \times n}$, one can achieve substantial computational savings for the recurrent computation of both the objective and the gradient by avoiding its explicit construction:

Theorem 2. *For a linear kernel with patterns in $X = \mathbb{R}^d$, one can compute the objective $F_{\lambda'}(\mathbf{c})$ and the gradient $\nabla F_{\lambda'}(\mathbf{c})$ in $\mathcal{O}(nd)$ time using $\mathcal{O}(nd)$ space for a given candidate solution $\mathbf{c} \in \mathbb{R}^n$.*

Proof. Due to the linear kernel, one can compute

$$\mathbf{K}\mathbf{c} = \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{c}) \tag{8}$$

and thus all predictions $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)$ in $\mathcal{O}(nd)$ time. In the same manner, one can obtain $\mathbf{c}^{\mathrm{T}}\mathbf{K}\mathbf{c}$ and $\mathbf{K}\mathbf{a}$ in $\mathcal{O}(nd)$ time (where the vector $\mathbf{a} \in \mathbb{R}^n$ can be computed in $\mathcal{O}(n)$ time given the predictions). Thus, both the objective $F_{\lambda'}(\mathbf{c})$ and the gradient $\nabla F_{\lambda'}(\mathbf{c})$ can be obtained in $\mathcal{O}(nd)$ time. The space requirements are bounded by the space needed to store the data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, which is $\mathcal{O}(nd)$. □

Thus, if the data resides in a low-dimensional feature space (i. e., $d \ll n$), one can reduce the runtime significantly for function and gradient calls. For high-dimensional but sparse data (i. e., if the matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ contains only $s \ll nd$ nonzero entries), one can further reduce the computational cost in the following way:

Theorem 3. *For a linear kernel with patterns in $X = \mathbb{R}^d$ and data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ with $s \ll nd$ nonzero entries, one can compute the objective $F_{\lambda'}(\mathbf{c})$ and the gradient $\nabla F_{\lambda'}(\mathbf{c})$ in $\mathcal{O}(s)$ time using $\mathcal{O}(s)$ space for a given candidate solution $\mathbf{c} \in \mathbb{R}^n$.*

Proof. Without loss of generality, we assume that $s \geq n - 1$ holds. Similar to the derivations above, one can compute $\mathbf{K}\mathbf{c} = \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{c})$ and therefore the predictions

[3] The space consumption can be reduced to $\mathcal{O}(1)$. The provided bounds, however, depict the needed space consumptions needed if one resorts to matrix-based implementations.

[4] This sequence can be seen as *annealing sequence*, which is a common strategy [7, 16] to create easier problem instances at early stages of the optimization process and to deform these instances to the final task throughout the overall execution.

$f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)$ as well as $\mathbf{a} \in \mathbb{R}^n$ in $\mathcal{O}(s)$ time using standard sparse matrix multiplication techniques. In the same way, one can compute $\mathbf{c}^{\mathrm{T}}\mathbf{K}\mathbf{c}$ and $\mathbf{K}\mathbf{a}$ in $\mathcal{O}(s)$ time. Hence, both the objective $F_{\lambda'}(\mathbf{c})$ and the gradient $\nabla F_{\lambda'}(\mathbf{c})$ can be obtained in $\mathcal{O}(s)$ time spending $\mathcal{O}(s)$ space. □

3.3.2. Low-Dimensional Search Space

For the case of a non-linear kernel, one can resort to *the subset of regressors method* [33] to reduce these computational costs, i.e., one can approximate the original hypothesis (3) via

$$\hat{f}(\cdot) = \sum_{k=1}^{r} \hat{c}_{j_k} k(\mathbf{x}_{j_k}, \cdot), \tag{9}$$

where $R = \{j_1, \ldots, j_r\} \subseteq \{1, \ldots, n\}$ is a subset of indices. Using this approximation scheme leads to a slightly modified objective $\hat{F}_{\lambda'}(\hat{\mathbf{c}})$ for $\hat{\mathbf{c}} \in \mathbb{R}^r$, where the predictions $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)$ are replaced by their corresponding approximations $\hat{f}(\mathbf{x}_1), \ldots, \hat{f}(\mathbf{x}_n)$ in the objective (4). Similar derivations as for the non-approximation case show that the gradient $\nabla \hat{F}_{\lambda'}(\hat{\mathbf{c}})$ is then given as

$$\nabla \hat{F}_{\lambda'}(\hat{\mathbf{c}}) \quad = \quad \mathbf{K}_R\, \mathbf{a} + 2\lambda \mathbf{K}_{R,R}\, \hat{\mathbf{c}}, \tag{10}$$

where f has to be replaced by $\hat{f}$ in the former definition of the vector $\mathbf{a} \in \mathbb{R}^n$. It is easy to see that one can compute both the new objective as well as its gradient in an efficient kind of way:

Theorem 4. *For $\hat{\mathbf{c}} \in \mathbb{R}^r$, the approximated objective $\hat{F}_{\lambda'}(\hat{\mathbf{c}})$ and the gradient $\nabla \hat{F}_{\lambda'}(\hat{\mathbf{c}})$ can be computed in $\mathcal{O}(nr)$ time spending $\mathcal{O}(nr)$ space.*

Proof. All predictions $\hat{f}(\mathbf{x}_1), \ldots, \hat{f}(\mathbf{x}_n)$ can be computed in $\mathcal{O}(nr)$ time for a $\hat{\mathbf{c}} \in \mathbb{R}^r$. Given these predictions, one can compute the modified vector $\mathbf{a} \in \mathbb{R}^n$ in $\mathcal{O}(n)$ time. The remaining operations for obtaining the new objective $\hat{F}_{\lambda'}(\hat{\mathbf{c}})$ and its gradient $\nabla \hat{F}_{\lambda'}(\hat{\mathbf{c}})$ can be performed in $\mathcal{O}(nr + r^2) = \mathcal{O}(nr)$ time. The space consumption, dominated by $\mathbf{K}_R$, is $\mathcal{O}(nr)$. □

Note that the subset of regressors method can also be implemented via the *kernel PCA map* [34] by considering only the first r eigenvalues and by applying the computational shortcut for the linear kernel given the associated (precomputed) kernel matrix.

3.3.3. Limited Memory Quasi-Newton

The non-sparse approximation of the Hessian's inverse for the quasi-Newton scheme leads to a $\mathcal{O}(n^2)$ time and to a $\mathcal{O}(n^2)$ space consumption. To reduce these computational costs, one can resort to the so-called *L-BFGS* method [29], which depicts a memory and time saving variant of the original BFGS scheme. In a nutshell, the idea consists in generating the approximations $\mathbf{H}_0, \mathbf{H}_1, \ldots$ only based on the last $m \ll n$ iterations and to perform low-rank updates on the fly without storing the involved matrices explicitly. This leads to an update time of $\mathcal{O}(mn)$ for all operations related to the intermediate optimization phases (not counting the time for function and gradient calls). As pointed out by Nocedal and Wright [29], small values for m are usually sufficient in practice (ranging from, e.g., $m = 3$ to $m = 50$). Thus, assuming m to be a relatively small constant, the operations needed by the optimization engine essentially scale linearly with the number n of optimization variables (per iteration).

3.4. Offset Term and Balancing Constraint

For the above derivations, the offset term b was omitted. It is worth pointing out that one can easily integrate such a term into the optimization scheme by considering an additional dimension via $\mathbf{c}^{\mathrm{T}} \to (\mathbf{c}^{\mathrm{T}}, b)$. In this case, the hypothesis (3) becomes

$$f(\cdot) = \sum_{i=1}^{n} c_i k(\mathbf{x}_i, \cdot) + b \tag{11}$$

and one has to adapt both the objective and the gradient appropriately. It can also be useful to incorporate additional knowledge via a *balancing constraint* of the form

$$\left| \frac{1}{u} \sum_{i=1}^{u} \max(0, y_i) - p \right| < \varepsilon \tag{12}$$

with user-defined $\varepsilon > 0$ and $p \in [0, 1]$, where the latter parameter is an estimate for the ratio of positive assignments for the unlabeled patterns (appropriate estimates can be obtained via the labeled part of the data).

In the remainder of this work, we consider a modified version of the above constraint having the form [14]

$$\frac{1}{u} \sum_{i=1}^{u} \langle \mathbf{w}, \mathbf{x}_i \rangle + b = \sum_{i=1}^{l} y'_i. \tag{13}$$

For the linear kernel, one can simplify this constraint by enforcing $\sum_{i=1}^{u} \mathbf{x}_i = \mathbf{0}$, which leads to $b = \sum_{i=1}^{l} y'_i$ [15].[5] The easiest way to incorporate such a constraint for non-linear kernels is to center the data in the feature space via $\Phi_i \to \Phi_i - \hat{\mathbf{m}}$ using the (approximated) mean $\mathbf{m} = \frac{1}{M} \sum_k \Phi_k$ of the mapped patterns.

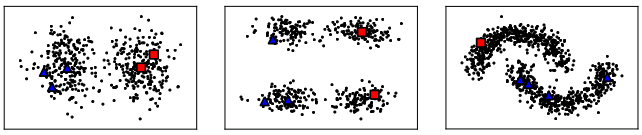
4. EXPERIMENTS

In the remainder of this work, we will present a detailed comparison of several competing semi-supervised support vector machine implementations on a variety of data sets.

4.1. Experimental Setup

All runtime analyses have been performed on a standard desktop computer with a `Intel(R) Core(TM) i5 CPU` at 2.80GHz running `Ubuntu 12.04`.

[5] Note that for sparse data, centering the data usually yields a dense data matrix. However, one can perform the desired centering steps "on the fly" when computing both the function and gradient calls without affecting the computational efforts.



(a) G2C (b) G4C (c) Moons

Figure 3: Distribution of the artificial data sets ($d = 2$). The red squares and blue triangles depict the labeled patterns; the remaining black points correspond to the unlabeled ones.

Data Set	n	d	Data Set	n	d
G2C	500	500	G4C	500	500
Moons	200	2	COIL(i,j)	144	400
USPS(8,0)	2,261	256	USPS(2,5)	1,645	256
USPS(2,7)	1,721	256	USPS(3,8)	1,532	256
MNIST(i,j)	10,000	784	pcmac	1,946	7,511
real-sim	72,309	20,958	gcat	23,119	47,236
ccat	23,119	47,236	aut-avn	71,175	20,707

Table 1: Data sets considered in the experimental evaluation, each consisting of n patterns having d features.

4.1.1. Implementation Details

Our implementation is based on `Python`, the `Scipy` package, and the `Numpy` package. The function and gradient evaluations are implemented via efficient matrix operations provided by the `Numpy` package. To avoid numerical instabilities (see above), we make use of $\log(1+\exp(t)) \approx t$ and $\frac{\exp(t)}{1+\exp(t)} \approx 1$ for $t \geq 500$ for the computation of the objective and its gradient. The quasi-Newton framework is implemented via the `optimize` module of the `Scipy` package (using `fmin_l_bfgs_b` with $m = 50$). We denote the resulting implementation by `QN-S`^{3}`VM`.

4.1.2. Data Sets

We consider several artificial and real-world data sets, see Table 1 for an overview. For each scenario described below, we use the first half of a data set as training and the second half as test set. To induce semi-supervised scenarios, we will split each training set instance into a labeled and an unlabeled part and use different ratios for the particular setting (where l, u, t denotes the number of labeled, unlabeled, and test patterns, respectively).

Artificial 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 multivariate Gaussian distributions $X_i \sim \mathcal{N}(\mathbf{m}_i, I)$, 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 3 (a). If not noted otherwise, we use $n = 500$ and $d = 500$ and denote the induced data set by `G2C`. The second artificial data set aims at generating a possibly misleading structure: Here, we draw $n/4$ points from each of four multivariate Gaussian distributions $X_i \sim \mathcal{N}(\mathbf{m}_i, I)$, where $\mathbf{m}_1 = (-2.5, -5.0, 0.0, \ldots, 0.0)$, $\mathbf{m}_2 = (-2.5, +5.0, 0.0, \ldots, 0.0)$, $\mathbf{m}_3 = (+2.5, -5.0, 0.0, \ldots, 0.0)$, and $\mathbf{m}_4 = (+2.5, +5.0, 0.0, \ldots, 0.0)$, see Figure 3 (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 `G4C`. Finally, we consider the well-known two-dimensional `Moons` data set with $n = 1,000$ points, see Figure 3 (c).

Real-World Data Sets. In addition to these artificial data sets, we consider several real-world data sets including the `COIL` [35], the `USPS` [2], and the `MNIST`[6] data sets. 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. A similar notation is used for the binary classification tasks induced the `USPS` and `MNIST` data set. For all these data sets, we rescaled the pixels such that the resulting values are in $[0, 1]$. Finally, following Sindhwani and Keerthi [10], we consider several large-scale sparse data sets (`real-sim`, `gcat`, `ccat`, `aut-avn`, and `pcmac`) in our experimental evaluation. Due to lack of space, we refer to Sindhwani and Keerthi [10] for a detailed description of these data set instances.

4.1.3. Competing Approaches

We consider the `LIBSVM` [36] implementation as baseline. As semi-supervised competitors, we make use of the constrained concave-convex procedure (`UniverSVM`) of Collobert *et al.* [17], the fast multi-switch transductive support vector machine (`TSVMlin`) [10], and the deterministic annealing approach (`DA`) [10]. Except for the model parameters (see below), we resort to the default values provided by the corresponding implementations.

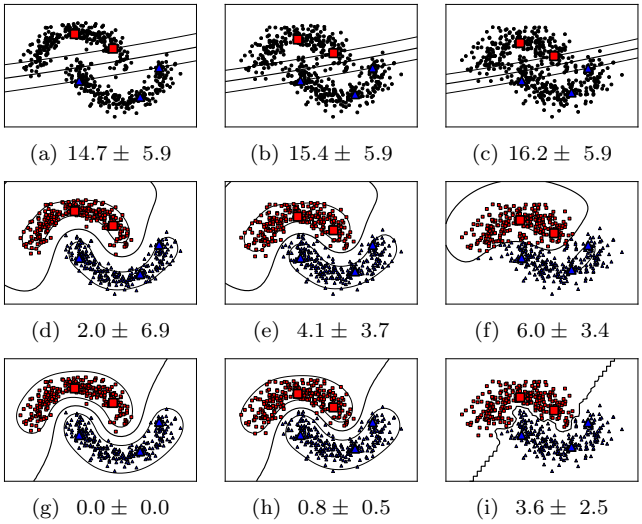
4.1.4. Model Selection

For the experimental evaluation, we will tune the non-fixed parameters via 5-fold cross-validation on the labeled part of the training set (if not stated otherwise). The final classification performances are measured on the test sets.

As similarity measures we consider a *linear kernel* $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ and a *radial basis function* (RBF) *kernel* $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-(2\sigma^2)^{-1} ||\mathbf{x}_i - \mathbf{x}_j||^2)$ with kernel width σ.[7] The cost parameters λ and λ' for `TSVMlin`, `DA`, and `QN-S`^{3}`VM` are tuned on a small grid $(\lambda, \lambda') \in \{2^{-10}, \ldots, 2^{10}\} \times \{0.01, 1, 100\}$ of possible parameters. For the `UniverSVM` and the `LIBSVM` scheme, we consider $(C, C^*) \in \{2^{-10}, \ldots, 2^{10}\} \times \{\frac{0.01}{u}, \frac{1.0}{u}, \frac{100.0}{u}\}$ and $C \in \{2^{-10}, \ldots, 2^{10}\}$ as parameter grids, respectively. Finally, we resort to a short sequence of annealing steps for

[6] `http://yann.lecun.com/exdb/mnist/`

[7] To select the kernel width σ for the RBF kernel, we consider the set $\{0.01s, 0.1s, 1s, 10s, 100s\}$ of possible assignments with $s = \sqrt{\sum_{k=1}^{d} (\max([\mathbf{x}_1]_k, \ldots, [\mathbf{x}_n]_k) - \min([\mathbf{x}_1]_k, \ldots, [\mathbf{x}_n]_k))^2}$.



(a) 14.7 ± 5.9 (b) 15.4 ± 5.9 (c) 16.2 ± 5.9

(d) 2.0 ± 6.9 (e) 4.1 ± 3.7 (f) 6.0 ± 3.4

(g) 0.0 ± 0.0 (h) 0.8 ± 0.5 (i) 3.6 ± 2.5

Figure 4: The large red squares and blue triangles depict the labeled data; the small black dots the unlabeled data. The smaller red squares and blue triangles depict the computed partitions of the unlabeled patterns. The average classification errors are reported (first row=`LIBSVM`, second row=`UniverSVM` method, and third row=`QN-S`^{3}`VM`).

`QN-S`^{3}`VM` ($\alpha_1 = 0.000001$, $\alpha_2 = 0.0001$, $\alpha_3 = 0.01$, $\alpha_4 = 0.1$, $\alpha_5 = 0.5$, $\alpha_6 = 1.0$).

All considered semi-supervised methods make use a balance constraint with an appropriate estimate for the desired ratio of positive and negative assignments for the unlabeled patterns. We provide appropriate estimates to all methods via the labeled part of the training set.

4.2. Experimental Results

We now depict the results of our experiments.

4.2.1. Model Flexibility

For the sake of exposition, we start by considering the well-known `Moons` data set that is said to be a difficult training instance for semi-supervised support vector machines due to its non-linear structure. In Figure 4, the results for `LIBSVM`, `UniverSVM`, and `QN-S`^{3}`VM` are shown for slightly varying distributions (using the RBF kernel). For all figures, the average test error (with one standard deviation) over 10 random partitions into labeled, unlabeled, and test patterns, is given (grid-search is performed on the test set). It can be clearly seen that the supervised approach is not able to generate reasonable models. The two considered semi-supervised approaches, however, can successfully incorporate the additional information provided by the unlabeled data in a stable manner.

4.2.2. Amount of Data

As shown above, sufficient labeled data is essential for supervised learning approaches to yield reasonable models. For semi-supervised approaches, the amount of unlabeled data used for training is an important issue as well. To illustrate this matter, we consider the `G4C` data set and vary both the amount of labeled and unlabeled data. For this experiment, we consider the `QN-S`^{3}`VM` implementation and resort to the `LIBSVM` scheme as baseline (using a linear kernel). 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 5 (a), the result of this experiment is shown: Given more than 20% labeled data, the semi-supervised approach performs clearly better. 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 Figure 5 (b). Clearly, the semi- supervised approach needs sufficient unlabeled data to yield appropriate models in a reliable manner.

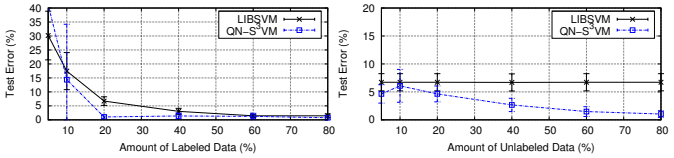


Figure 5: The `QN-S`^{3}`VM` approach can successfully incorporate unlabeled data, see Figure (a). However, sufficient unlabeled data is needed as well to reveal sufficient information about the structure of the data, see Figure (b).

4.2.3. Classification Performance

Motivated by the two initial experiments depicted above, we consider up to five different amounts of labeled, unlabeled, and test patterns per data set instance to analyze the classification performance of all competing approaches. For all data sets and for all competing approaches, a linear kernel is used. In Table 2, the test errors (and one standard deviations) averaged over 10 random partitions are given. As mentioned above, the parameters are tuned via 5 fold cross-validation on the labeled part; thus, a realistic setting for tuning the parameters is considered.

It can be seen that the classification performances for the semi-supervised methods are, in general, superior or at least competitive to those of the supervised `LIBSVM`-baseline. The `QN-S`^{3}`VM` approach yields a surprisingly good performance on the non-sparse data sets, which is superior to all other semi-supervised competitors. For the sparse data sets, `TSVMlin`, `DA`, and `QN-S`^{3}`VM` perform similarly, and none of these three methods can outcompete the other two ones. Note that the semi-supervised methods yield a significantly better performance on the sparse data sets compared to the `LIBSVM`-baseline.

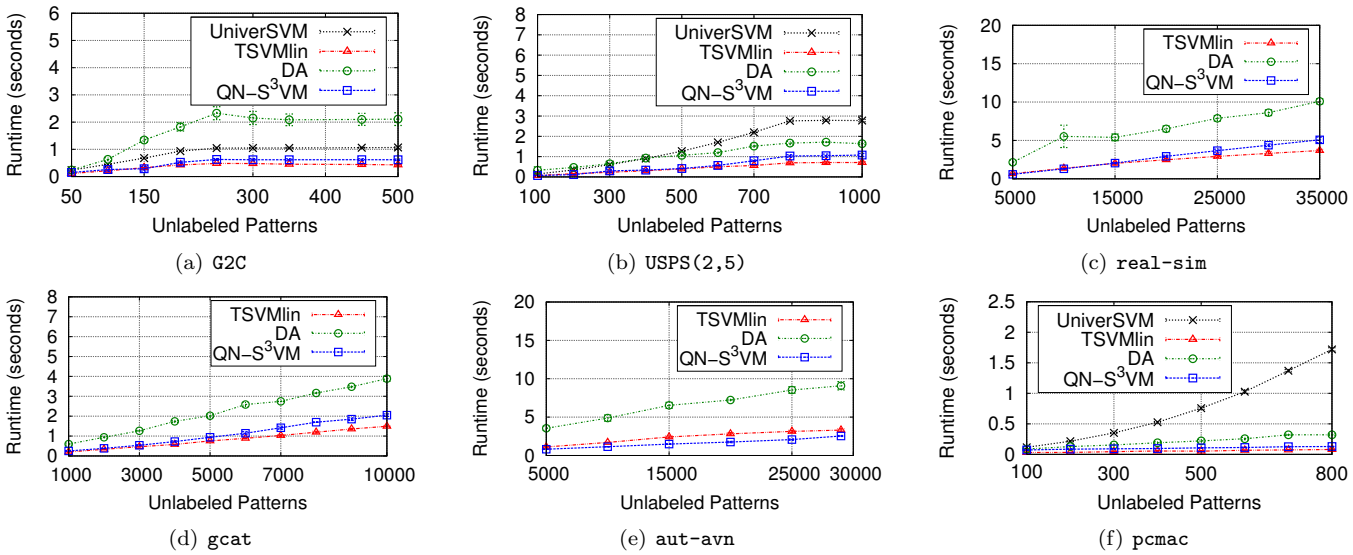
4.2.4. Computational Considerations

We will finally analyze the practical runtimes. For this sake, we fix the model parameters ($\lambda = 1$, $\lambda' = 1$, $C = 1$, $C^* = 1$) and make again use of the linear kernel.

Data Set	l	u	t	LIBSVM	UniverSVM	TSVMlin	DA	QN-S^{3}VM
G2C	25	225	250	13.2 ± 2.8	1.8 ± 0.9	5.6 ± 2.4	2.2 ± 0.7	1.9 ± 0.9
G2C	50	200	250	6.3 ± 2.4	1.8 ± 0.8	2.8 ± 1.5	2.5 ± 1.7	2.1 ± 0.9
G4C	25	225	250	20.6 ± 11.5	13.3 ± 15.2	14.7 ± 12.0	14.2 ± 12.4	11.4 ± 11.7
G4C	50	200	250	6.9 ± 1.6	3.0 ± 1.8	3.4 ± 0.7	2.5 ± 1.4	2.2 ± 1.0
C(3,6)	14	101	29	16.2 ± 7.2	16.9 ± 13.9	12.4 ± 7.4	20.7 ± 17.8	8.3 ± 7.9
C(3,6)	28	87	29	3.8 ± 4.2	5.2 ± 5.8	4.1 ± 5.5	4.1 ± 6.5	4.5 ± 4.1
C(5,9)	14	101	29	13.4 ± 7.8	19.3 ± 10.9	17.6 ± 10.6	14.1 ± 7.6	12.1 ± 8.9
C(5,9)	28	87	29	4.5 ± 5.6	7.2 ± 9.1	6.6 ± 9.9	6.2 ± 7.5	6.9 ± 10.1
C(6,19)	14	101	29	15.5 ± 13.1	21.0 ± 12.0	12.8 ± 11.9	15.5 ± 13.2	10.7 ± 10.8
C(6,19)	28	87	29	3.4 ± 3.4	4.5 ± 5.1	4.8 ± 6.6	3.8 ± 5.7	3.1 ± 4.2
C(18,19)	14	101	29	6.9 ± 8.0	7.6 ± 8.8	12.4 ± 9.0	10.7 ± 9.6	3.4 ± 8.3
C(18,19)	28	87	29	1.4 ± 4.1	5.2 ± 9.7	3.1 ± 7.3	2.4 ± 6.2	1.0 ± 3.1
M(1,7)	20	480	500	4.2 ± 1.6	4.3 ± 2.8	8.2 ± 5.2	9.9 ± 5.7	2.8 ± 1.2
M(1,7)	50	450	500	2.6 ± 1.0	3.7 ± 2.5	2.8 ± 1.5	2.7 ± 1.5	2.7 ± 1.1
M(2,5)	20	480	500	10.2 ± 3.1	6.3 ± 3.9	8.6 ± 4.5	10.3 ± 4.6	6.3 ± 2.2
M(2,5)	50	450	500	5.8 ± 1.7	4.2 ± 1.6	5.5 ± 2.1	5.7 ± 2.2	4.3 ± 1.2
M(2,7)	20	480	500	7.9 ± 4.4	8.0 ± 4.7	10.7 ± 5.1	11.5 ± 5.9	5.3 ± 2.3
M(2,7)	50	450	500	5.0 ± 1.5	5.1 ± 1.6	4.7 ± 1.9	4.4 ± 1.8	4.0 ± 0.9
M(3,8)	20	480	500	18.8 ± 11.5	16.2 ± 4.0	15.8 ± 4.5	16.1 ± 5.3	12.9 ± 5.2
M(3,8)	50	450	500	9.0 ± 2.4	9.5 ± 4.1	9.4 ± 3.4	9.4 ± 3.0	8.3 ± 2.9
U(2,5)	16	806	823	10.5 ± 4.7	9.0 ± 5.6	12.2 ± 7.6	14.3 ± 9.7	4.6 ± 1.7
U(2,5)	32	790	823	5.4 ± 0.8	5.6 ± 1.8	5.8 ± 3.5	5.8 ± 3.7	4.5 ± 1.4
U(2,7)	17	843	861	4.9 ± 2.9	6.1 ± 5.3	9.2 ± 7.4	10.5 ± 12.7	2.3 ± 1.0
U(2,7)	34	826	861	2.8 ± 1.1	3.4 ± 2.4	5.6 ± 4.6	5.8 ± 4.9	2.0 ± 1.1
U(3,8)	15	751	766	12.9 ± 8.3	8.7 ± 3.9	9.8 ± 5.7	9.3 ± 4.8	5.4 ± 1.9
U(3,8)	30	736	766	7.3 ± 2.1	6.4 ± 1.6	7.8 ± 3.5	8.3 ± 3.3	5.7 ± 1.8
U(8,0)	22	1108	1131	5.0 ± 2.0	3.2 ± 2.2	6.4 ± 5.5	8.2 ± 6.0	2.2 ± 1.2
U(8,0)	45	1085	1131	3.0 ± 0.9	3.3 ± 1.8	3.5 ± 2.1	3.6 ± 2.2	8.1 ± 11.4
real-sim	90	36064	36155	28.7 ± 1.6	–	11.7 ± 2.5	11.7 ± 2.7	14.1 ± 1.5
real-sim	180	35974	36155	23.9 ± 5.9	–	9.7 ± 1.4	12.0 ± 3.1	13.0 ± 1.4
real-sim	361	35793	36155	17.3 ± 5.5	–	8.1 ± 0.7	10.2 ± 1.7	11.6 ± 1.7
real-sim	1446	34708	36155	8.3 ± 1.7	–	6.0 ± 0.2	7.1 ± 0.5	9.0 ± 0.8
real-sim	2892	33262	36155	6.8 ± 1.1	–	5.5 ± 0.2	6.6 ± 0.3	8.8 ± 0.5
gcat	57	11517	11575	24.5 ± 2.9	–	7.7 ± 0.9	7.6 ± 1.7	8.4 ± 2.3
gcat	231	11343	11575	10.6 ± 1.5	–	6.7 ± 0.7	6.7 ± 0.8	6.3 ± 0.5
gcat	462	11112	11575	7.6 ± 0.8	–	6.0 ± 0.6	6.2 ± 0.9	5.7 ± 0.3
gcat	925	10649	11575	6.2 ± 0.5	–	5.6 ± 0.3	5.6 ± 0.4	5.5 ± 0.3
gcat	1851	9723	11575	5.4 ± 0.2	–	5.2 ± 0.2	5.3 ± 0.3	5.3 ± 0.2
ccat	57	11517	11575	25.1 ± 7.7	–	17.1 ± 2.0	17.6 ± 3.3	20.1 ± 6.0
ccat	115	11459	11575	18.0 ± 2.4	–	14.2 ± 1.3	13.6 ± 1.8	14.4 ± 1.7
ccat	231	11343	11575	14.0 ± 1.4	–	11.8 ± 1.2	11.8 ± 1.4	11.5 ± 1.2
ccat	462	11112	11575	11.3 ± 0.4	–	10.2 ± 0.8	10.1 ± 0.7	10.5 ± 0.5
ccat	925	10649	11575	9.5 ± 0.3	–	9.0 ± 0.5	9.2 ± 0.6	10.0 ± 0.5
aut-avn	177	35410	35588	20.7 ± 6.5	–	5.7 ± 0.7	4.4 ± 0.8	5.7 ± 0.6
aut-avn	355	35232	35588	10.8 ± 1.3	–	5.8 ± 0.9	3.8 ± 0.7	5.2 ± 0.4
aut-avn	1423	34164	35588	6.0 ± 0.3	–	4.5 ± 0.7	3.4 ± 0.2	5.1 ± 0.2
aut-avn	2847	32740	35588	4.7 ± 0.2	–	4.0 ± 0.4	3.4 ± 0.2	5.0 ± 0.2
aut-avn	5694	29893	35588	3.8 ± 0.1	–	3.6 ± 0.2	3.4 ± 0.1	5.1 ± 0.2
pcmac	48	924	974	24.8 ± 9.6	10.4 ± 2.6	9.6 ± 2.3	8.5 ± 2.1	7.3 ± 1.4
pcmac	97	876	973	11.6 ± 4.1	8.5 ± 4.2	8.1 ± 3.5	7.6 ± 3.5	7.1 ± 2.7
pcmac	145	828	973	8.5 ± 1.6	6.9 ± 2.2	7.3 ± 1.4	7.0 ± 1.4	6.6 ± 1.1
pcmac	243	730	973	6.7 ± 1.0	5.6 ± 1.2	6.0 ± 1.0	6.0 ± 1.1	5.5 ± 0.8
pcmac	291	681	974	6.2 ± 1.0	5.4 ± 1.2	6.1 ± 1.1	5.7 ± 0.9	5.1 ± 1.0

Table 2: Classification performances of all competing approaches. For all methods and for all data sets, the average error on the test set along with the one standard deviation is provided. The best results with respect to the average test errors are highlighted.

Practical Runtimes. The practical runtimes of all semi-supervised methods for six data set instances are depicted in Figure 6. The plots indicate a similar runtime behavior on the non-sparse data sets. Since the UniverSVM implementation does not directly take advantage of sparse data set properties, it is significantly slower than the other three methods (see, e. g., pcmac). For the remaining three ones, TSVMlin and QN-S^{3}VM slightly outperform DA.



(a) `G2C` (b) `USPS(2,5)` (c) `real-sim`

(d) `gcat` (e) `aut-avn` (f) `pcmac`

Figure 6: The practical runtimes for all semi-supervised implementations are given for six data set instances. These results indicate that our gradient-based implementation can effectively handle all considered data set instances.

These runtime results shall, however, only indicate the practical runtimes needed to generate the classification results shown in Table 2. It is worth pointing out that they naturally depend on various issues like the particular parameter assignments, the stopping criteria, and the programming languages used for implementing the approaches. The proposed `QN-S3VM` implementation is based on (slow) `Python`; however, since the main computational bottlenecks can be implemented using only matrix-based operations, it is surprisingly efficient.

Large-Scale Dense Data. The `MNIST` data set instances have been restricted in their size up to now to render a detailed comparison of all methods possible. To sketch the applicability of `QN-S3VM` for large-scale settings given dense data sets, we consider the complete `MNIST(1,7)` and the `MNIST(3,8)` instances and vary the size of the training set from $2,000$ to $10,000$ patterns. Further, we make use of the kernel matrix approximation scheme for `QN-S3VM` with $r = 2,000$ (and randomly selected basis vectors). The practical runtimes of all semi-supervised methods are given in Figure 7. It can be seen that all methods can handle such settings efficiently.

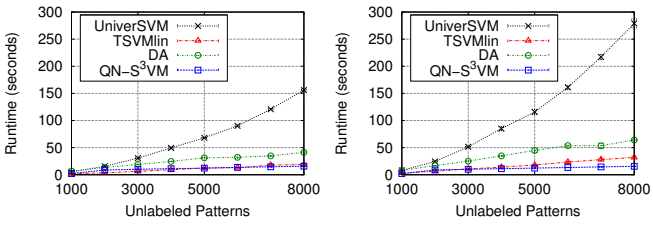


Figure 7: Runtime results for the large-scale dense `MNIST(1,7)` (left) and `MNIST(3,8)` (right) data set instances.

ACKNOWLEDGEMENTS

This work has been supported in part by funds of the *Deutsche Forschungsgemeinschaft* (DFG) (Fabian Gieseke, grant KR 3695) and by the Academy of Finland (Tapio Pahikkala, grant 134020). The authors would like to thank the anonymous reviewers for valuable comments and suggestions on an early version of this work.

References

[1] F. Gieseke, A. Airola, T. Pahikkala, O. Kramer, Sparse quasi-Newton optimization for semi-supervised support vector machines, in: Proc. of the 1st Int. Conf. on Pattern Recognition Applications and Methods, 2012, pp. 45–54.

[2] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer, 2009.

[3] I. Steinwart, A. Christmann, Support Vector Machines, Springer, New York, NY, USA, 2008.

[4] O. Chapelle, B. Schölkopf, A. Zien (Eds.), Semi-Supervised Learning, MIT Press, Cambridge, MA, 2006.

[5] X. Zhu, A. B. Goldberg, Introduction to Semi-Supervised Learning, Morgan and Claypool, 2009.

[6] K. P. Bennett, A. Demiriz, Semi-supervised support vector machines, in: Adv. in Neural Information Proc. Systems 11, MIT Press, 1999, pp. 368–374.

[7] T. Joachims, Transductive inference for text classification using support vector machines, in: Proc. Int. Conf. Mach. Learn., 1999, pp. 200–209.

[8] V. Vapnik, A. Sterin, On structural risk minimization or overall risk in a problem of pattern recognition, Aut. and Remote Control 10 (3) (1977) 1495–1503.

[9] M. Adankon, M. Cheriet, A. Biem, Semisupervised least squares support vector machine, IEEE Trans. Neural Netw. 20 (12) (2009) 1858–1870.

[10] V. Sindhwani, S. S. Keerthi, Large scale semi-supervised linear SVMs, in: Proc. 29th annual international ACM SIGIR conference on Research and development in information retrieval, ACM, New York, NY, USA, 2006, pp. 477–484.

[11] T. D. Bie, N. Cristianini, Convex methods for transduction, in: Adv. in Neural Information Proc. Systems 16, MIT Press, 2004, pp. 73–80.
[12] L. Xu, D. Schuurmans, Unsupervised and semi-supervised multi-class support vector machines, in: Proc. National Conf. on Art. Intell., 2005, pp. 904–910.
[13] O. Chapelle, V. Sindhwani, S. S. Keerthi, Branch and bound for semi-supervised support vector machines, in: Adv. in Neural Information Proc. Systems 19, MIT Press, 2007, pp. 217–224.
[14] O. Chapelle, A. Zien, Semi-supervised classification by low density separation, in: Proc. Tenth Int. Workshop on Art. Intell. and Statistics, 2005, pp. 57–64.
[15] O. Chapelle, M. Chi, A. Zien, A continuation method for semi-supervised SVMs, in: Proc. Int. Conf. Mach. Learn., 2006, pp. 185–192.
[16] V. Sindhwani, S. Keerthi, O. Chapelle, Deterministic annealing for semi-supervised kernel machines, in: Proc. Int. Conf. Mach. Learn., 2006, pp. 841–848.
[17] R. Collobert, F. Sinz, J. Weston, L. Bottou, Trading convexity for scalability, in: Proc. Int. Conf. Mach. Learn., 2006, pp. 201–208.
[18] G. Fung, O. L. Mangasarian, Semi-supervised support vector machines for unlabeled data classification, Optim. Methods Softw. 15 (2001) 29–44.
[19] B. Zhao, F. Wang, C. Zhang, Cuts3vm: A fast semi-supervised svm algorithm, in: Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2008, pp. 830–838.
[20] I. S. Reddy, S. Shevade, M. Murty, A fast quasi-Newton method for semi-supervised SVM, Pattern Recognit. 44 (10–11) (2011) 2305–2313.
[21] K. Zhang, J. T. Kwok, B. Parvin, Prototype vector machine for large scale semi-supervised learning, in: Proc. Int. Conf. Mach. Learn., 2009, pp. 1233–1240.
[22] O. Chapelle, V. Sindhwani, S. S. Keerthi, Optimization techniques for semi-supervised support vector machines, J. Mach. Learn. Res. 9 (2008) 203–233.
[23] F. Gieseke, T. Pahikkala, O. Kramer, Fast evolutionary maximum margin clustering, in: Proc. Int. Conf. Mach. Learn., 2009, pp. 361–368.
[24] Y.-F. Li, I. W. Tsang, J. T. Kwok, Z.-H. Zhou, Tighter and convex maximum margin clustering, in: Proceedings of the 12th International Conference on Artificial Intelligence and Statistics, JMLR: W&CP 5, 2009, pp. 344–351.
[25] H. Valizadegan, R. Jin, Generalized maximum margin clustering and unsupervised kernel learning, in: Adv. in Neural Information Proc. Systems 19, 2007, pp. 1417–1424.
[26] F. Wang, B. Zhao, C. Zhang, Linear time maximum margin clustering, IEEE Transactions on Neural Networks 21 (2) (2010) 319–332.
[27] L. Xu, J. Neufeld, B. Larson, D. Schuurmans, Maximum margin clustering, in: Adv. Neur. Inf. Proc. Syst. 17, 2005, pp. 1537–1544.
[28] K. Zhang, I. W. Tsang, J. T. Kwok, Maximum margin clustering made practical, in: Proc. Int. Conf. Mach. Learn., 2007, pp. 1119–1126.
[29] J. Nocedal, S. J. Wright, Numerical Optimization, 1st Edition, Springer, 2000.
[30] R. Rifkin, G. Yeo, T. Poggio, Regularized least-squares classification, in: Adv. in Learning Theory: Methods, Models and Applications, IOS Press, 2003.
[31] B. Schölkopf, R. Herbrich, A. J. Smola, A generalized representer theorem, in: Proc. 14th Annual Conf. on Computational Learning Theory, 2001, pp. 416–426.
[32] T. Zhang, F. J. Oles, Text categorization based on regularized linear classification methods, Inf. Retr. Boston 4 (2001) 5–31.
[33] R. M. Rifkin, Everything old is new again: A fresh look at historical approaches in machine learning, Ph.D. thesis, MIT (2002).
[34] B. Schölkopf, A. J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, MIT Press, Cambridge, MA, USA, 2001.
[35] S. Nene, S. Nayar, H. Murase, Columbia object image library (coil-100), Tech. rep. (1996).
[36] C.-C. Chang, C.-J. Lin, LIBSVM: a library for support vector machines, software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm` (2001).

Fabian Gieseke received his Diploma degrees in mathematics and computer science from the University of Münster, Germany, and his PhD in computer science from the Carl von Ossietzky University Oldenburg, Germany. He is currently working as a postdoctoral researcher at the University of Oldenburg. His research interests include support vector machines and their extensions to semi- and unsupervised learning settings, and applications in astronomy and energy systems.

Antti Airola is a postdoctoral researcher at University of Turku, Department of Information Technology. He received the D.Sc. degree from University of Turku in 2011. His research interests include both basic research in machine learning as well as applied data analysis.

Tapio Pahikkala received his Bachelors, Masters, and Doctoral degrees from University of Turku, Finland, in 2002, 2003, and 2008, respectively, and his Adjunct Professorship of Computer Sciences in 2011. He currently holds a three-year postdoctoral research grant from the Academy of Finland. His research focuses on machine learning, pattern recognition, algorithmics, and computational intelligence. He has authored more than seventy peer reviewed scientific publications and served in program committees of numerous scientific conferences.

Oliver Kramer is Juniorprofessor for Computational Intelligence at the University of Oldenburg in Germany. His main research interests are machine learning, optimization, and the application of computational intelligence techniques to renewable energy systems. He received a PhD from the University of Paderborn, Germany, in 2008. After a postdoc stay at the TU Dortmund, Germany, from 2007 to 2009, and the International Computer Science Institute in Berkeley (USA) in 2010, he became Juniorprofessor at the Bauhaus University Weimar, Germany. Since August 2011 he is affiliated to the Department of Computing Science at the University of Oldenburg.