SPARSE QUASI-NEWTON OPTIMIZATION FOR SEMI-SUPERVISED SUPPORT VECTOR MACHINES

Fabian Gieseke[1], Antti Airola[2], Tapio Pahikkala[2] and Oliver Kramer[1]
[1]*Computer Science Department, Carl von Ossietzky Universität Oldenburg, 26111 Oldenburg, Germany*
[2]*Turku Centre for Computer Science, Department of Information Technology, University of Turku, 20520 Turku, Finland*

Keywords: Semi-supervised support vector machines, Non-convex optimization, Quasi-Newton methods, Sparse data, Nyström approximation.

Abstract: In real-world scenarios, labeled data is often rare while unlabeled data can be obtained in huge quantities. A current research direction in machine learning is the concept of semi-supervised support vector machines. This type of binary classification approach aims at taking the additional information provided by unlabeled patterns into account to reveal more information about the structure of the data and, hence, to yield models with a better classification performance. However, generating these semi-supervised models requires solving difficult optimization tasks. In this work, we present a simple but effective approach to address the induced optimization task, which is based on a special instance of the quasi-Newton family of optimization schemes. The resulting framework can be implemented easily using black box optimization engines and yields excellent classification and runtime results on both artificial and real-world data sets that are superior (or at least competitive) to the ones obtained by competing state-of-the-art methods.

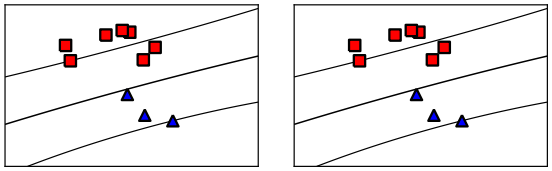
1 INTRODUCTION

One of the most important machine learning tasks is classification. If sufficient labeled training data is given, there exists a variety of techniques like the *k-nearest neighbor*-classifier or *support vector machines* (SVMs) (Hastie et al., 2009; Steinwart and Christmann, 2008) to address such a task. However, labeled data is often rare in real-world applications. One active research field in machine learning is *semi-supervised learning* (Chapelle et al., 2006b; Zhu and Goldberg, 2009). 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) (Bennett and Demiriz, 1999; Joachims, 1999; Vapnik and Sterin, 1977) 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. Its semi-supervised variant aims at taking the unlabeled patterns into account by searching for a partition (of these patterns into two classes) such that a *subsequent* application of a (modified) support vector machine leads to the best result.

1.1 Related Work

The original problem formulation of semi-supervised support vector machines was given by Vapnik and Sterin (Vapnik and Sterin, 1977) 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 (Joachims, 1999) and Bennet and Demiriz (Bennett and Demiriz, 1999). In general, there exist two lines of research, namely (a) *combinatorial* and (b) *continuous* optimization schemes. The naive 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. The continuous optimization perspective (see below) leads to a real-valued but non-convex task. For both research directions, a variety of different techniques has been proposed in recent years that are based on semi-definite programming (Bie and Cristianini, 2004; Xu and Schuurmans, 2005), the continuation method (Chapelle et al., 2006a), deterministic

Gieseke F., Airola A., Pahikkala T. and Kramer O. (2012).
SPARSE QUASI-NEWTON OPTIMIZATION FOR SEMI-SUPERVISED SUPPORT VECTOR MACHINES.
In *Proceedings of the 1st International Conference on Pattern Recognition Applications and Methods*, pages 45-54
DOI: 10.5220/0003755300450054



(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.

annealing (Sindhwani et al., 2006), the (constrained) concave-convex procedure (Collobert et al., 2006; Fung and Mangasarian, 2001; Zhao et al., 2008), and other strategies (Adankon et al., 2009; Chapelle and Zien, 2005; Mierswa, 2009; Sindhwani and Keerthi, 2006; Zhang et al., 2009). A related approach, also based on a quasi-Newton framework, is proposed by Reddy *et al.* (Reddy et al., 2010); however, they do not consider differentiable surrogates and therefore apply more complicated subgradient methods. Many other approaches exist, and we refer to Chapelle *et al.* (Chapelle et al., 2006b; Chapelle et al., 2008) and Zhu *et al.* (Zhu and Goldberg, 2009) for comprehensive surveys.

1.2 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 $\mathbf{M}_{RS}$ 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 $\mathbf{M}_{R[n]} = \mathbf{M}_R$.

2 CLASSIFICATION TASK

In supervised scenarios, we are given a training set $T_l = \{(\mathbf{x}_1, y'_1),\dots,(\mathbf{x}_l, y'_l)\}$ of labeled patterns $\mathbf{x}_i$ belonging to a set X. The general goal of classification approaches consists in building good models which can predict valuable labels for unseen patterns (Hastie et al., 2009). In semi-supervised learning frameworks, we are additionally given a set $T_u = \{\mathbf{x}_{l+1},\dots,\mathbf{x}_{l+u}\} \subset X$ of unlabeled training patterns. Here, the goal is to improve the quality of the models by taking both the labeled and the unlabeled part of the data into account.

2.1 Support Vector Machines

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

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

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* $k : X \times X \to \mathbb{R}$ (Steinwart and Christmann, 2008). 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 leads to the original definition of support vector machines (Schölkopf et al., 2001; Steinwart and Christmann, 2008), see Figure 2 (a).[1]

2.2 Semi-supervised SVMs

Given the additional set $T_u = \{\mathbf{x}_{l+1},\dots,\mathbf{x}_{l+u}\} \subset X$ of unlabeled training patterns, semi-supervised support vector machines (Bennett and Demiriz, 1999; Joachims, 1999; Vapnik and Sterin, 1977) 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 $\mathbf{y}^* = (y^*_1,\dots,y^*_u)^{\mathrm{T}} \in \{-1,+1\}^u$ that are optimal with respect to

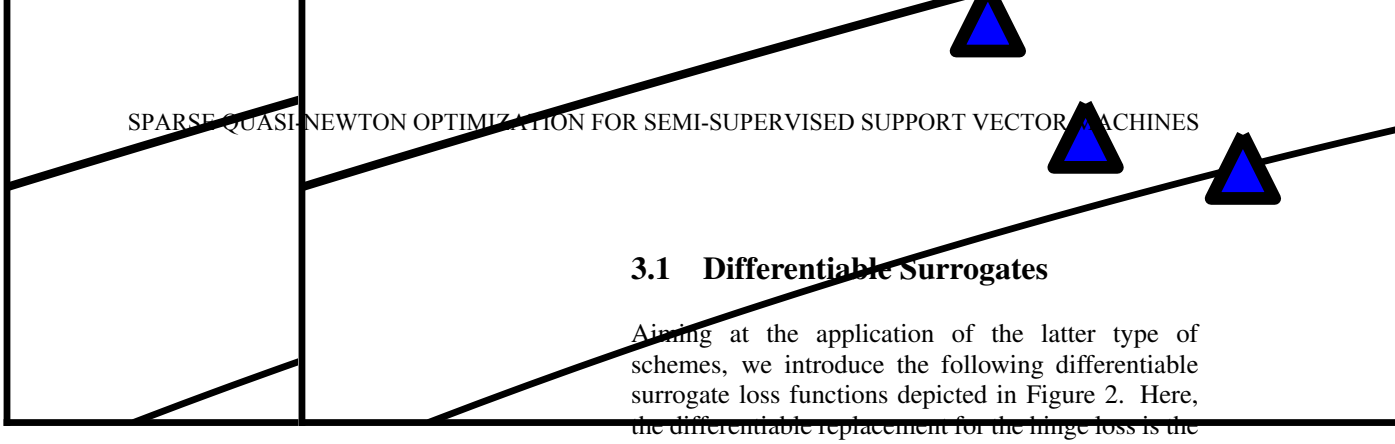
$$\begin{aligned}\underset{f\in\mathcal{H},\,\mathbf{y}\in\{-1,+1\}^u}{\text{minimize}} \; & \frac{1}{l}\sum_{i=1}^{l} \mathcal{L}\big(y'_i, f(\mathbf{x}_i)\big) \\ & + \lambda' \frac{1}{u}\sum_{i=1}^{u} \mathcal{L}\big(y_i, f(\mathbf{x}_{l+i})\big) + \lambda ||f||^2_{\mathcal{H}}\end{aligned} \qquad (2)$$

where $\lambda', \lambda > 0$ are user-defined parameters. 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.

2.3 Continuous Optimization

As mentioned above, one can derive an equivalent continuous optimization task: Using the hinge loss,

[1] The latter formulation does not include a bias term $b \in \mathbb{R}$, which addresses translated data. For complex kernel functions like the RBF kernel, adding this bias term does not yield any known advantages, both from a theoretical as well as practical point of view (Steinwart and Christmann, 2008). For the linear kernel, a regularized bias effect can be obtained by adding a dimension of ones to the input data.



(a) (b)

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).

the optimal assignments for the vector $\mathbf{y}$ for a fixed $f \in \mathcal{H}$ are given by $y_i = \operatorname{sgn}(f(\mathbf{x}_i))$ (Chapelle and Zien, 2005).[2] This yields

$$\begin{aligned}\underset{f\in\mathcal{H}}{\text{minimize}} \frac{1}{l}\sum_{i=1}^{l}\max\left(0, 1-y_i' f(\mathbf{x}_i)\right) \\ + \frac{\lambda'}{u}\sum_{i=1}^{u}\max\left(0, 1-|f(\mathbf{x}_{l+i})|\right) + \lambda||f||_{\mathcal{H}}^2.\end{aligned} \tag{3}$$

Note that the *effective loss* on the unlabeled patterns penalizes predictions around the origin; thus, the overall loss increases if the decision function f passes through these patterns, see Figure 2 (b). By applying the *representer theorem* (Schölkopf et al., 2001) for latter task, it follows that an optimal solution $f \in \mathcal{H}$ is of the form

$$f(\cdot) = \sum_{i=1}^{l} c_i' k(\mathbf{x}_i, \cdot) + \sum_{i=1}^{u} c_i k(\mathbf{x}_{l+i}, \cdot) \tag{4}$$

with coefficients $\mathbf{c} = (c_1', \ldots, c_l', c_1, \ldots, c_u)^{\mathrm{T}} \in \mathbb{R}^{l+u}$. Thus, one gets a continuous optimization task which consists in finding the optimal coefficient vector $\mathbf{c} \in \mathbb{R}^n$ with $n = l+u$. Note that the effective loss renders the task non-convex and non-differentiable (since the partial functions are non-differentiable).

3 QUASI-NEWTON SCHEME

We will consider a special instance of the quasi-Newton optimization framework (Nocedal and Wright, 2000). Besides the objective function itself, methods belonging to this class of schemes only require the gradient to be supplied.

[2]Note that the latter observation does only hold *without* a balance constraint of the form

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

for small $\varepsilon > 0$ and $b_c \in (0,1)$.

3.1 Differentiable Surrogates

Aiming at the application of the latter type of schemes, we introduce the following differentiable surrogate loss functions depicted in Figure 2. Here, the differentiable replacement for the hinge loss is the modified logistic loss (Zhang and Oles, 2001); the replacement for the effective loss for the unlabeled part is a well-known candidate in this field (Chapelle and Zien, 2005). Thus, the new overall surrogate objective is given by

$$\begin{aligned}F_{\lambda'}(\mathbf{c}) = \frac{1}{l}\sum_{i=1}^{l}\frac{1}{\gamma}\log\left(1+\exp(\gamma(1-y_i' f(\mathbf{x}_i)))\right) \\ + \frac{\lambda'}{u}\sum_{i=1}^{u}\exp(-3(f(\mathbf{x}_{l+i}))^2) \\ + \lambda\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j)\end{aligned} \tag{5}$$

with $f(\cdot) = \sum_{p=1}^{n} c_p k(\mathbf{x}_p, \cdot)$ and using $||f||_{\mathcal{H}}^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j)$ (Schölkopf et al., 2001). The next lemma shows that both a function and a gradient call can be performed efficiently.

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

Proof. The gradient is given by

$$\nabla F_{\lambda'}(\mathbf{c}) = \mathbf{K}\mathbf{a} + 2\lambda\mathbf{K}\mathbf{c} \tag{6}$$

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

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

Since all predictions $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)$ can be computed in $O(n^2)$ total time, one can compute the vector $\mathbf{a} \in \mathbb{R}^n$ and therefore the objective and the gradient in $O(n^2)$. The space requirements are dominated by the kernel matrix $\mathbf{K} \in \mathbb{R}^{n\times n}$. □

Note that numerical instabilities might 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 $O(n^2)$ time in a numerically stable manner.

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 (7)
        Update c_{k+1} = c_k + β_k p_k
        Update H_{k+1} via (8)
        k = k + 1
    end while
    c_0 = c_k
end for
```

3.2 Quasi-Newton Framework

One of the most popular quasi-Newton schemes is the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) (Nocedal and Wright, 2000) 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 (5) while ignoring the (non-convex) unlabeled part (i. e., $\lambda' = 0$). The influence of the unlabeled part is then increased gradually via the sequence $\alpha_1, \ldots, \alpha_\tau$.[3] 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

$$\mathbf{p}_k = -\mathbf{H}_k \nabla F_{\alpha_i \cdot \lambda'}(\mathbf{c}_k) \tag{7}$$

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

$$\mathbf{H}_{k+1} = (I - \rho_k \mathbf{s}_k \mathbf{z}_k^{\mathrm{T}}) \mathbf{H}_k (I - \rho_k \mathbf{z}_k \mathbf{s}_k^{\mathrm{T}}) + \rho_k \mathbf{s}_k \mathbf{s}_k^{\mathrm{T}} \tag{8}$$

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 $\left|\left|\nabla F_{\alpha_i \cdot \lambda'}(\mathbf{c}_k)\right|\right| > \varepsilon$ 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 usually resorts 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 (Nocedal and Wright, 2000).

[3] This sequence can be seen as *annealing sequence*, which is a common strategy (Joachims, 1999; Sindhwani et al., 2006) 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.

3.3 Computational Speed-ups

Two main computational bottlenecks arise for large-scale settings: Firstly, the recurrent computation of the objective and gradient needed by the quasi-Newton framework is cumbersome. Secondly, the approximation of the Hessian's inverse is, in general, not sparse, which leads to quadratic-time operations for the quasi-Newton framework itself (Nocedal and Wright, 2000). In the following, we will show how to alleviate these two problems.

3.3.1 Limited Memory Quasi-Newton

The non-sparse approximation of the Hessian's inverse leads to a $O(n^2)$ time and to a $O(n^2)$ space consumption. To reduce these computational costs, we consider the L-BFGS methods (Nocedal and Wright, 2000), 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 $O(mn)$ 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, 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.

3.3.2 Low-dimensional Search Space

It remains to show how to reduce the second bottleneck, i. e., the recurrent computation of both the objective and the gradient. For this sake, one can resort to *the subset of regressors method* (Rifkin, 2002) to reduce these computational costs, i. e., one approximates the original hypothesis (4) 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 (5). Similar derivations as for the non-approximation case show that the gradient $\nabla\hat{F}_{\lambda'}(\hat{\mathbf{c}})$ is then given as

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

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:

Lemma 2. *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 $O(nr)$ time spending $O(nr)$ space.*

Proof. All predictions $\hat{f}(\mathbf{x}_1),\ldots,\hat{f}(\mathbf{x}_n)$ can be computed in $O(nr)$ 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 $O(n)$ 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 $O(nr+r^2)=O(nr)$ time. The space consumption, dominated by $\mathbf{K}_R$, is $O(nr)$. □

Thus, in combination with the L-BFGS scheme depicted above, both the runtime as well as the space consumption are reduced significantly. Here, the parameter $r\in\{1,\ldots,n\}$ determines a trade-off between the achieved speed-up and the accuracy of the approximation. Another way to obtain considerable speed-ups can be achieved for the special case of a linear kernel, which we will describe next.

3.3.3 Linear Kernel and Sparse Data

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. In case of the linear kernel, one can write the kernel matrix as $\mathbf{K}=\mathbf{X}\mathbf{X}^{\mathrm{T}}\in\mathbb{R}^{n\times n}$ and can achieve substantial computational savings by avoiding its explicit construction. This is the case, for instance, if the data resides in a low-dimensional feature space (i. e., $d\ll n$) or due to the data matrix being sparse, meaning that it contains only few nonzero entries.

Lemma 3. *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 $O(nd)$ time using $O(nd)$ space for a given candidate solution $\mathbf{c}\in\mathbb{R}^n$.*

Proof. Due to the linear kernel, one can compute

$$\mathbf{K}\mathbf{c}=\mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{c}) \qquad (11)$$

and thus all predictions $f(\mathbf{x}_1),\ldots,f(\mathbf{x}_n)$ in $O(nd)$ time. In the same manner, one can obtain $\mathbf{c}^{\mathrm{T}}\mathbf{K}\mathbf{c}$ and $\mathbf{K}\mathbf{a}$ in $O(nd)$ time (where the vector $\mathbf{a}\in\mathbb{R}^n$ can be computed in $O(n)$ time given the predictions). Thus, both the objective $F_{\lambda'}(\mathbf{c})$ and the gradient $\nabla F_{\lambda'}(\mathbf{c})$ can be obtained in $O(nd)$ 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 $O(nd)$. □

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 kind of way:[4]

Lemma 4. *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 $O(s)$ time using $O(s)$ 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 $f(\mathbf{x}_1),\ldots,f(\mathbf{x}_n)$ as well as $\mathbf{a}\in\mathbb{R}^n$ in $O(s)$ 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 $O(s)$ time. Hence, both the objective $F_{\lambda'}(\mathbf{c})$ and the gradient $\nabla F_{\lambda'}(\mathbf{c})$ can be obtained in $O(s)$ time spending $O(s)$ space. □

4 EXPERIMENTS

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

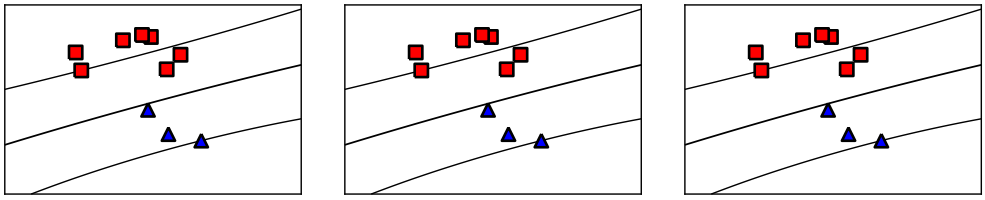
4.1 Experimental Setup

The runtime analysis are performed on a 3 GHZ Intel Core™ Duo PC running `Ubuntu 10.04`. We start by providing details related to the experimental setup.

4.1.1 Implementation Details

Our implementation is based on `Python`, the `Scipy` package (using the L-BFGS implementation `optimize.fmin_l_bfgs_b` (Byrd et al., 1995) with $m=50$), and the `Numpy` package. The function and gradient evaluations are based on efficient matrix operations provided by the `Numpy` package. As pointed out above, a direct implementation of the latter ones might suffer from numerical instabilities. To avoid these instabilities, we make use of $\log(1+\exp(t))\approx t$ and $\frac{\exp(t)}{1+\exp(t)}\approx 1$ for $t\geq 500$. We denote the resulting implementation by `QN-S`^{3}`VM`.

[4]Note that the term s is sometimes used to denote the average number of nonzero entries *per pattern* $\mathbf{x}_i\in X$ in the training set.



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

Figure 3: Distribution of all artificial data sets ($d = 2$). The red squares and blue triangles depict the labeled part of the data; the remaining black points correspond to the unlabeled part. Note that the two Gaussian data sets depict easy learning instances for $d = 2$, even given only few labeled patterns. However, the noise present in the data render the induced tasks difficult to approach in high dimensions ($d = 500$) in case only few labeled patterns are given (for supervised models).

4.1.2 Data Sets

We consider several artificial and real-world data sets, where the first half of each data set is used as training and the second half as test set. To induce semi-supervised scenarios, we split each training set 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 Gaussian2C. 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

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

see 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 Gaussian4C. Finally, we consider the well-known two-dimensional Moons data set with $n = 500$ points, see Figure 3 (c).

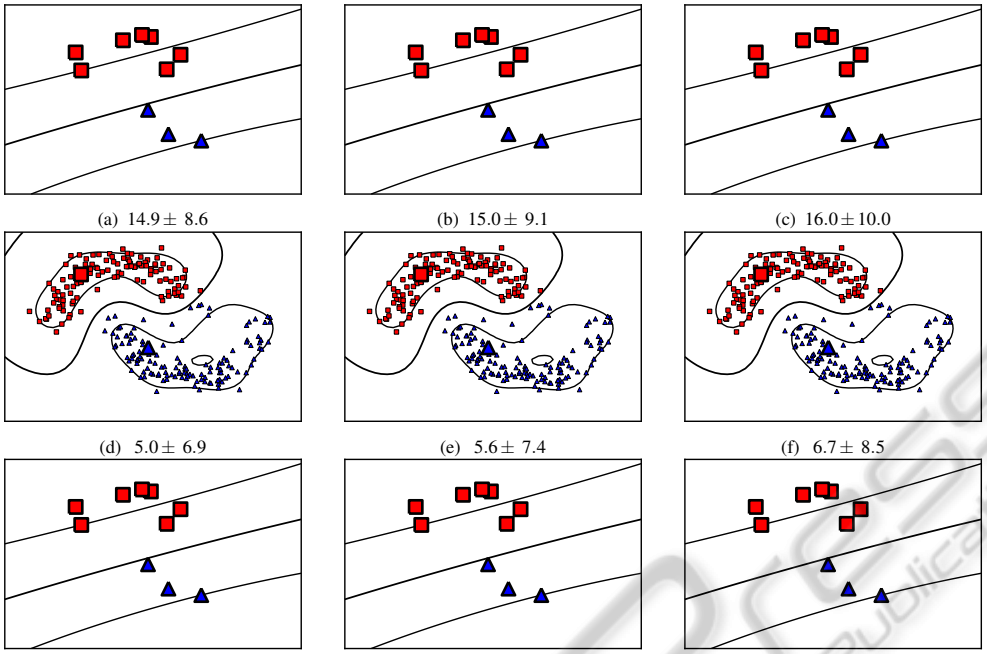
Real-world Data Sets. In addition to these artificial data sets, we consider several real-world data sets including the COIL (Nene et al., 1996) and the USPS (Hastie et al., 2009) data sets (consisting of both the training and test set of the original data set). For the COIL data set, we reduce the input 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 (using the ordering given in the data set). 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. Further, we focus on those pairs of objects/digits which are difficult to separate. Finally, we consider the Newsgroup20 and the TEXT data set. Let latter one is composed of the mac and mswindows classes of the Newsgroup20 data set (Chapelle and Zien, 2005).

4.1.3 Model Selection

In semi-supervised settings, model selection can be unreliable due to the lack of labeled data and is widely considered to be an open issue (Chapelle et al., 2006b). 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.[5] 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). 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, i.e., 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 small validation sets.

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

[5]This setup is often considered in related evaluations (Chapelle et al., 2006b).



(a) 14.9 ± 8.6 (b) 15.0 ± 9.1 (c) 16.0 ± 10.0

(d) 5.0 ± 6.9 (e) 5.6 ± 7.4 (f) 6.7 ± 8.5

(g) 0.0 ± 0.0 (h) 1.2 ± 2.5 (i) 2.6 ± 2.3

Figure 4: The large red squares and blue triangles depict the labeled data; the small black dots the unlabeled data. Further, the smaller red squares and blue triangles depict the partition of the unlabeled patterns computed by the semi-supervised approach. Clearly, the LIBSVM implementation (top row) is not able to generate appropriate models due to the lack of labeled data. Both the UniverSVM (middle row) and the QN-S^{3}VM approach (bottom row) can successfully incorporate the unlabeled data, whereas the performance gain is higher for the latter scheme. The average errors (with one standard deviation) on the test sets over 10 random partitions are reported.

performing set of parameters. As similarity measures we consider a *linear* $k(\mathbf{x}_i,\mathbf{x}_j) = \langle \mathbf{x}_i,\mathbf{x}_j \rangle$ 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. To select the kernel width σ for the RBF kernel, we consider the set $\{0.01s, 0.1s, 1s, 10s, 100s\}$ of possible assignments, where the value s is a rough estimate of the maximum distance between any pair of samples. [6]. The cost parameters λ and λ' are tuned on a small grid $(\lambda,\lambda') \in \{2^{-10},\ldots,2^{10}\} \times \{0.01, 1, 100\}$ of possible parameters. Further, a short sequence of annealing steps is used ($\alpha_1 = 0.01, \alpha_2 = 0.1, \alpha_3 = 1.0$).

4.1.4 Competing Approaches

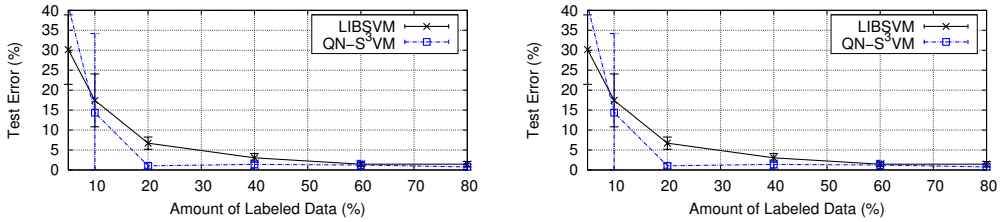
We use the LIBSVM (Chang and Lin, 2001) as supervised competitor with $C \in \{2^{-10},\ldots,2^{10}\}$. As semi-supervised competitor, we consider the UniverSVM approach (Collobert et al., 2006). Again, we perform grid search for tuning the involved parameters $((C,C^*) \in \{2^{-10},\ldots,2^{10}\} \times \{\frac{0.01}{u}, \frac{1.0}{u}, \frac{100.0}{u}\})$. The ratio between the two classes is provided to the algorithm via the -w option. Except for the option -S option (which we set to -0.3), the default values for the remaining parameters are used. We selected the UniverSVM approach since it seems to be the strongest semi-supervised competitor (with publicly available code) both with respect to the running time and classification performance. Further, the corresponding algorithmic framework is quite similar to the one proposed in this work, i.e, surrogate loss functions (ramp loss) along with a continuous local search scheme (concave-convex procedure) are employed.[7]

4.2 Experimental Results

We will now depict the outcome of several experiments demonstrating the potential of our approach.

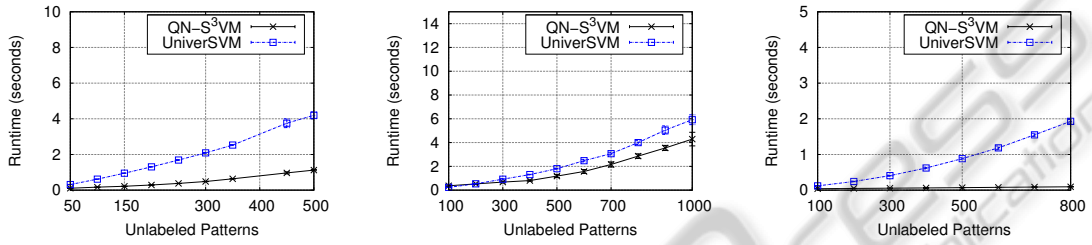
[6] $s = \sqrt{\sum_{k=1}^{d} (\max([\mathbf{x}_1]_k,\ldots,[\mathbf{x}_n]_k) - \min([\mathbf{x}_1]_k,\ldots,[\mathbf{x}_n]_k))^2}$

[7] A detailed comparison of the UniverSVM approach with other semi-supervised optimization schemes (like the TSVM approach (Joachims, 1999)) has been conducted by Collobert *et al.* (Collobert et al., 2006). Their results indicate the superior performance of UniverSVM compared to related methods. For the sake of exposition, we will therefore focus on a comparison of our approach with UniverSVM.



(a) (b)

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



(a) Gaussian2C (b) USPS(8,0) (c) TEXT

Figure 6: Runtimes for both QN-S^{3}VM and UniverSVM evaluated on three data sets.

4.2.1 Model Flexibility

The well-known Moons data set 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 the LIBSVM (top row), the UniverSVM (middle row), and the QN-S^{3}VM implementation (bottom row) are shown given slightly varying distributions (using the RBF kernel). To select the model parameters, we make use of the test set (non-realistic scenario). For all figures, the average test error (with one standard deviation) over 10 random partitions into labeled, unlabeled, and test patterns, is given. It can be clearly seen that the supervised approach is not able to generate reasonable models. Further, the semi-supervised approaches can successfully incorporate the additional information provided by the unlabeled data, whereas the QN-S^{3}VM scheme seems to perform better on this particular data set instances.

4.2.2 Amount of Data

As motivated 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 analyze how much labeled and unlabeled data is needed for our approach, we consider the Gaussian4C data set and vary the amount of labeled and unlabeled data. For this experiment, we make use of the non-realistic scenario and resort to the LIBSVM implementation 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 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.

4.2.3 Classification Performance

We consider both the realistic and the non-realistic scenario to evaluate the classification performance of our approach. For each data set, we analyze the behavior of all competing approach given up to three amounts of labeled, unlabeled, and test patterns. For all data sets and for all competing approaches, a linear kernel is used. In Table 1, the test errors (and one standard deviations) averaged over 10 random partitions are given for both scenarios. It can be clearly seen that, for the non-realistic scenario, the semi-supervised approaches mostly yield better results compared to LIBSVM, even if only few labeled patterns are given. Thus, they can successfully incorporate the additional information provided by the unlabeled data. Due to lack of labeled data for model selection, the results for the realistic scenario are worse

Table 1: Classification performances of all competing approaches for both the non-realistic and the realistic scenario. The best results with respect to the average test errors are highlighted.

Data Set	l	u	t	LIBSVM		UniverSVM		QN-S^{3}VM	
				non-realistic	realistic	non-realistic	realistic	non-realistic	realistic
Gaussian2C	25	225	250	13.0± 2.9	13.2± 2.8	1.0± 0.5	1.8± 0.9	0.5± 0.5	1.8± 0.6
Gaussian2C	50	200	250	5.8± 2.2	6.3± 2.4	1.0± 0.4	1.8± 0.8	0.5± 0.6	2.1± 1.0
Gaussian4C	25	225	250	17.4± 6.6	20.6±11.5	7.6±12.2	13.3±15.2	6.8±11.9	10.7±13.9
Gaussian4C	50	200	250	6.7± 1.5	6.9± 1.6	1.6± 0.8	2.5± 1.5	0.9± 0.6	2.5± 1.0
COIL(3,6)	14	101	29	13.4± 7.0	16.2± 7.2	2.8± 3.7	16.9±13.9	7.6± 9.1	13.1±12.5
COIL(3,6)	28	87	29	3.1± 3.3	3.8± 4.2	0.3± 1.0	5.2± 5.8	2.1± 3.5	3.1± 5.0
COIL(5,9)	14	101	29	13.4± 7.0	13.4± 7.8	6.9± 7.2	19.3±10.9	10.0± 8.5	17.9±12.0
COIL(5,9)	28	87	29	3.4± 4.1	4.5± 5.6	1.4± 3.2	7.2± 9.1	3.1± 5.0	3.8± 4.7
COIL(6,19)	14	101	29	12.1± 9.8	15.5±13.1	4.5± 8.2	21.0±12.0	10.3±10.9	13.4±11.8
COIL(6,19)	28	87	29	3.1± 3.3	3.4± 3.4	0.7± 2.1	4.5± 5.1	1.7± 3.2	3.4± 4.1
COIL(18,19)	14	101	29	6.9± 8.0	6.9± 8.0	1.0± 3.1	7.6± 8.8	10.0± 9.4	14.1± 9.9
COIL(18,19)	28	87	29	1.4± 4.1	1.4± 4.2	0.0± 0.0	5.2± 9.7	3.1± 5.4	5.9± 9.1
USPS(2,5)	16	806	823	9.4± 5.1	10.5± 4.7	3.2± 0.5	9.0± 5.6	3.1± 0.3	4.7± 1.2
USPS(2,5)	32	790	823	4.7± 0.7	5.4± 0.8	3.2± 0.5	5.7± 1.8	2.6± 0.6	4.0± 1.1
USPS(2,7)	17	843	861	4.6± 3.0	4.9± 2.9	1.5± 0.3	6.1± 5.3	1.2± 0.2	1.5± 0.2
USPS(2,7)	34	826	861	2.5± 1.0	2.8± 1.1	1.4± 0.2	3.4± 2.4	1.2± 0.1	1.5± 0.5
USPS(3,8)	15	751	766	12.0± 8.2	12.9± 8.3	4.8± 1.1	8.7± 3.9	6.5± 7.7	8.7±11.1
USPS(3,8)	30	736	766	6.6± 2.1	7.3± 2.1	4.0± 0.1	7.1± 1.8	3.7± 1.2	5.5± 2.8
USPS(8,0)	22	1,108	1,131	4.8± 1.7	5.0± 2.0	1.7± 0.7	3.2± 2.2	1.4± 0.6	2.4± 1.5
USPS(8,0)	45	1,085	1,131	2.7± 0.8	3.0± 0.9	1.3± 0.4	3.3± 1.8	1.4± 0.7	1.7± 0.6
MNIST(1,7)	20	480	500	3.5± 1.3	4.2± 1.6	2.6± 1.0	4.3± 2.8	1.8± 0.7	2.5± 0.9
MNIST(1,7)	50	450	500	2.3± 1.0	2.6± 1.0	2.2± 0.9	3.7± 2.5	1.8± 0.9	2.3± 1.1
MNIST(2,5)	20	480	500	9.3± 3.2	10.2± 3.1	3.4± 0.7	6.3± 3.9	4.0± 3.1	6.4± 3.7
MNIST(2,5)	50	450	500	4.7± 1.2	5.8± 1.7	3.4± 0.7	4.2± 1.6	2.4± 0.6	4.2± 1.3
MNIST(2,7)	20	480	500	6.7± 3.7	7.9± 4.4	2.9± 0.6	8.0± 4.7	2.9± 1.1	3.9± 1.3
MNIST(2,7)	50	450	500	4.2± 1.2	5.0± 1.5	2.5± 0.5	5.1± 1.6	2.2± 0.4	3.7± 2.1
MNIST(3,8)	20	480	500	15.2± 4.1	18.8±11.5	8.6± 3.3	16.1± 3.9	8.6± 5.1	12.7± 5.9
MNIST(3,8)	50	450	500	8.6± 2.5	9.0± 2.4	6.3± 2.0	9.5± 4.1	6.2± 2.2	7.5± 2.9
TEXT	48	924	974	23.5± 6.7	24.8± 9.6	6.5± 1.0	10.4± 2.6	8.2± 4.7	21.2±13.8
TEXT	97	876	973	11.4± 4.1	11.6± 4.1	5.8± 0.6	8.5± 4.2	5.3± 0.9	8.3± 2.5
TEXT	194	779	973	7.4± 1.3	7.6± 1.3	5.2± 0.8	6.5± 1.5	4.9± 0.8	5.4± 1.0
TEXT	389	584	973	4.8± 0.7	4.8± 0.7	4.2± 0.6	4.8± 0.6	4.0± 0.6	4.5± 0.6

compared to the non-realistic one. Still, except for the COIL data sets, the results for QN-S^{3}VM are at least as good as the ones of LIBSVM.

4.2.4 Computational Considerations

We will finally focus on the runtime behavior. To simplify the setup, we fix the model parameters (i.e., $\lambda = 1$ and $\lambda' = 1$) and use a linear kernel.

Medium-scale Scenarios. To give an idea of the runtime needed to obtain the results provided in Table 1, we consider the Gaussian2C, the USPS(8,0) and the sparse TEXT data set (with $l = 25$, 22, and 48 labeled patterns, respectively). Further, we vary the amount of unlabeled patterns as shown in Figure 6; the average runtime over 10 runs is provided. The plots indicate a comparable runtime behavior on the non-sparse data sets. On the sparse text data set, however, the QN-S^{3}VM is considerably faster; note that even with 1,000 unlabeled examples, the practical runtime is less than 0.1 seconds per single execution.

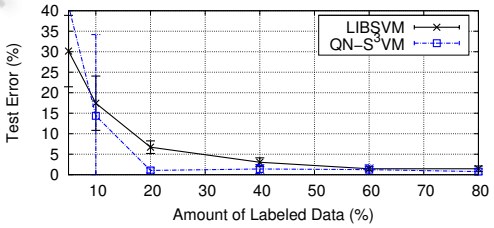


Figure 7: Large-Scale experiment.

Large-scale Scenarios. To sketch the applicability in large-scale settings, we consider the MNIST(1,7) 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 with $r = 2{,}000$ (and randomly selected basis vectors). The runtime and the needed function calls are given in Figure 7. As it can be seen, the runtime is still moderate. Further, it seems that a constant amount of function calls (being independent of the size of the training set) is needed.

5 CONCLUSIONS

We proposed a quasi-Newton optimization framework for the non-convex task induced by semi-supervised support vector machines. It seems that this type of optimization schemes is well suited for the task at hand since it (a) can be implemented easily due to its conceptual simplicity and (b) admits direct accelerations for sparse and non-sparse data. The experiments indicate that the resulting approach can successfully incorporate unlabeled data, even in realistic scenarios where the lack of labeled data complicates the model selection phase.

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.

REFERENCES

Adankon, M., Cheriet, M., and Biem, A. (2009). Semisupervised least squares support vector machine. *IEEE Transactions on Neural Networks*, 20(12):1858–1870.

Bennett, K. P. and Demiriz, A. (1999). Semi-supervised support vector machines. In *Adv. in Neural Information Proc. Systems 11*, pages 368–374. MIT Press.

Bie, T. D. and Cristianini, N. (2004). Convex methods for transduction. In *Adv. in Neural Information Proc. Systems 16*, pages 73–80. MIT Press.

Byrd, R. H., Byrd, R. H., Lu, P., Lu, P., Nocedal, J., Nocedal, J., Zhu, C., and Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208.

Chang, C.-C. and Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

Chapelle, O., Chi, M., and Zien, A. (2006a). A continuation method for semi-supervised SVMs. In *Proc. Int. Conf. on Mach. Learn.*, pages 185–192.

Chapelle, O., Schölkopf, B., and Zien, A., editors (2006b). *Semi-Supervised Learning*. MIT Press, Cambridge, MA.

Chapelle, O., Sindhwani, V., and Keerthi, S. S. (2008). Optimization techniques for semi-supervised support vector machines. *Journal of Mach. Learn. Res.*, 9:203–233.

Chapelle, O. and Zien, A. (2005). Semi-supervised classification by low density separation. In *Proc. Tenth Int. Workshop on Art. Intell. and Statistics*, pages 57–64.

Collobert, R., Sinz, F., Weston, J., and Bottou, L. (2006). Trading convexity for scalability. In *Proc. Int. Conf. on Mach. Learn.*, pages 201–208.

Fung, G. and Mangasarian, O. L. (2001). Semi-supervised support vector machines for unlabeled data classification. *Optimization Methods and Software*, 15:29–44.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*. Springer.

Joachims, T. (1999). Transductive inference for text classification using support vector machines. In *Proc. Int. Conf. on Mach. Learn.*, pages 200–209.

Mierswa, I. (2009). *Non-convex and multi-objective optimization in data mining*. PhD thesis, Technische Universität Dortmund.

Nene, S., Nayar, S., and Murase, H. (1996). Columbia object image library (coil-100). Technical report.

Nocedal, J. and Wright, S. J. (2000). *Numerical Optimization*. Springer, 1 edition.

Reddy, I. S., Shevade, S., and Murty, M. (2010). A fast quasi-Newton method for semi-supervised SVM. *Pattern Recognition*, In Press, Corrected Proof.

Rifkin, R. M. (2002). *Everything Old is New Again: A Fresh Look at Historical Approaches in Machine Learning*. PhD thesis, MIT.

Schölkopf, B., Herbrich, R., and Smola, A. J. (2001). A generalized representer theorem. In Helmbold, D. P. and Williamson, B., editors, *Proc. 14th Annual Conf. on Computational Learning Theory*, pages 416–426.

Sindhwani, V., Keerthi, S., and Chapelle, O. (2006). Deterministic annealing for semi-supervised kernel machines. In *Proc. Int. Conf. on Mach. Learn.*, pages 841–848.

Sindhwani, V. and Keerthi, S. S. (2006). Large scale semi-supervised linear SVMs. In *Proc. 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 477–484, New York, NY, USA. ACM.

Steinwart, I. and Christmann, A. (2008). *Support Vector Machines*. Springer, New York, NY, USA.

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

Xu, L. and Schuurmans, D. (2005). Unsupervised and semi-supervised multi-class support vector machines. In *Proc. National Conf. on Art. Intell.*, pages 904–910.

Zhang, K., Kwok, J. T., and Parvin, B. (2009). Prototype vector machine for large scale semi-supervised learning. In *Proc. of the Int. Conf. on Mach. Learn.*, pages 1233–1240.

Zhang, T. and Oles, F. J. (2001). Text categorization based on regularized linear classification methods. *Information Retrieval*, 4:5–31.

Zhao, B., Wang, F., and Zhang, C. (2008). Cuts3vm: A fast semi-supervised svm algorithm. In *Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 830–838.

Zhu, X. and Goldberg, A. B. (2009). *Introduction to Semi-Supervised Learning*. Morgan and Claypool.