A Scalable Modular Convex Solver for Regularized Risk Minimization

Choon Hui Teo[1], Quoc Le[1,2], Alex Smola[1], and S.V.N. Vishwanathan[1]
[1]Statistical Machine Learning, NICTA, Northbourne Avenue 218, Canberra 2601, ACT, Australia
[2]Max Planck Institute for Biological Cybernetics, Spemannstr. 38, 72076 Tübingen, Germany
{ChoonHui.Teo, Quoc.Le, Alex.Smola, SVN.Vishwanathan}@nicta.com.au

ABSTRACT

A wide variety of machine learning problems can be described as minimizing a regularized risk functional, with different algorithms using different notions of risk and different regularizers. Examples include linear Support Vector Machines (SVMs), Logistic Regression, Conditional Random Fields (CRFs), and Lasso amongst others. This paper describes the theory and implementation of a highly scalable and modular convex solver which solves all these estimation problems. It can be parallelized on a cluster of workstations, allows for data-locality, and can deal with regularizers such as ℓ_1 and ℓ_2 penalties. At present, our solver implements 30 different estimation problems, can be easily extended, scales to millions of observations, and is up to 10 times faster than specialized solvers for many applications. The open source code is freely available as part of the ELEFANT toolbox.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Optimization, Convexity

Keywords

Support Vectors, Gaussian Processes, Training Algorithms, Large-Scale, Bundle Methods, Parallel Optimization

1. INTRODUCTION

At the heart of many machine learning algorithms is the problem of minimizing a regularized risk functional. That is, one would like to solve

$$\underset{w}{\text{minimize}} \; J(w) := \lambda\Omega(w) + R_{\text{emp}}(w) \tag{1}$$

$$\text{where } R_{\text{emp}}(w) := \frac{1}{m}\sum_{i=1}^{m} l(x_i, y_i, w) \tag{2}$$

is the empirical risk. Moreover, $x_i \in X$ are referred to as training instances and $y_i \in Y$ are the corresponding labels.

KDD'07, August 12–15, 2007, San Jose, California, USA.

l is a nonnegative loss function measuring the discrepancy between y and the predictions arising from using w. We assume that it is convex in w. For instance, w might enter our model via $l(x, y, w) = (\langle w, x\rangle - y)^2$. Finally, $\Omega(w)$ is a convex function serving the role of a regularizer with regularization constant $\lambda > 0$.

If we consider the problem of predicting binary valued labels $y \in \{\pm 1\}$, we may set $\Omega(w) = \frac{1}{2}\|w\|^2$, and the loss $l(x_i, y_i, w)$ to be the hinge loss, $\max(0, 1 - y_i \langle w, x_i\rangle)$, which recovers linear Support Vector Machines (SVMs) [25, 36]. On the other hand, using the same regularizer but changing the loss function to $l(x_i, y_i, w) = \log(1 + \exp(-y_i \langle w, x_i\rangle))$, yields logistic regression. Extensions of these loss functions allow us to handle structure in the output space [1]. Changing the regularizer $\Omega(w)$ to the sparsity inducing $\|w\|_1$ leads to Lasso-type estimation algorithms [30, 39, 8].

The *kernel trick* is widely used to transform many of these algorithms into ones operating on a Reproducing Kernel Hilbert Space (RKHS). One lifts w into an RKHS and replaces all inner product computations with a positive definite kernel function $k(x, x') \leftarrow \langle x, x'\rangle$.

Examples of algorithms which employ the kernel trick (but essentially still solve (1)) include Support Vector regression [41], novelty detection [33], Huber's robust regression, quantile regression [37], ordinal regression [21], ranking [15], maximization of multivariate performance measures [24], structured estimation [38, 40], Gaussian Process regression [43], conditional random fields [28], graphical models [14], exponential families [3], and generalized linear models [17].

Traditionally, specialized solvers have been developed for solving the kernel version of (1) in the dual, e.g. [9, 23]. These algorithms construct the Lagrange dual, and solve for the Lagrange multipliers efficiently. Only recently, research focus has shifted back to solving (1) in the primal, e.g. [10, 25, 36]. This spurt in research interest is due to three main reasons: First, many interesting problems in diverse areas such as text classification, word-sense disambiguation, and drug design already employ rich high dimensional data which does not necessarily benefit from the kernel trick. All these domains are characterized by large datasets (with m of the order of a million) and very sparse features (e.g. the bag of words representation of a document). Second, many kernels (e.g. kernels on strings [42]) can effectively be linearized, and third, efficient factorization methods (e.g. [18]) can be used for a low rank representation of the kernel matrix thereby effectively rendering the problem linear.

For each of the above estimation problems *specialized* solvers exist, and the common approach is to write a new solver for

every new domain. Unfortunately, many implementations do not scale well and the scalable ones (e.g. SVMStruct [40]) are restricted to a rather specialized set of applications. Parallel solvers are even more difficult to find. Finally, the issue of data locality is rarely addressed, e.g. situations where data is owned by several entities which are unwilling to share their parts of X and Y.

In this paper, we address all the above issues by developing a fast, efficient, scalable, and parallel convex solver which can efficiently deal with data locality issues. We achieve this by decoupling the computation of the objective function and its gradient from the actual solver module. Our architecture is modular, i.e., one can plug and play with many different backend solvers many different loss functions, and different regularizers. In particular, we describe an efficient variant of the bundle method and provide rates of convergence.

The outline of our paper is as follows. In section 2 we will describe bundle methods and adapt them to our setting. We will also provide rates of convergence which are significantly better than those reported previously in literature; this stems from a sophisticated analysis which modifies and tightens previous proofs. In section 3 we will describe various loss functions implemented in our solver, while section 4 will describe the architecture of our solver. We will also demonstrate the ease with which one can plug a off-the-shelf solver like LBFGS into our framework. Section 5 is devoted to extensive experimental evaluation which shows that our implementation is up to 10 times faster than state-of-the-art specialized solvers in many applications, and we conclude with an outlook and discussion in section 6. Proofs are relegated to the appendix.

2. BUNDLE METHODS

The basic idea behind a cutting plane method is as follows: Given a convex function $g(w)$, it is always lower-bounded by its first-order Taylor approximation, i.e.,

$$g(w) \geq g(w_0) + \langle w - w_0, \partial_w g(w_0) \rangle \text{ for all } w, w_0. \tag{3}$$

This gives rise to the hope that if we have a set $W = \{w_1, \ldots, w_n\}$ of locations where we compute such a Taylor approximation, we should be able to obtain an ever-improving approximation of $g(w)$ [22]. See Figure 2 for an illustration. Formally, we have

$$g(w) \geq \max_{\bar{w} \in W} \left[g(\bar{w}) + \langle w - \bar{w}, \partial_w g(\bar{w}) \rangle \right], \tag{4}$$

which means that $g(w)$ can be lower-bounded by a piecewise linear function. Moreover, the approximation is exact at all $w_i \in W$. Note that if $g(w)$ is not differentiable everywhere, we can pick an arbitrary element of the subdifferential (which always exists) and use it instead of $\partial_w g(w)$. Finally, if $g(w)$ is twice differentiable, we can use the latter to bound the deviation between the lower bound and $g(w)$ by using the mean value theorem. We have

$$0 \leq g(w) - \max_{\bar{w} \in W} \left[g(\bar{w}) + \langle w - \bar{w}, \partial_w g(\bar{w}) \rangle \right] \leq \epsilon \tag{5}$$

where $\epsilon \leq M d^2(w, W)$. Here M is an upper bound on the largest eigenvalue of the Hessian $\left\| \partial_w^2 g(w) \right\|$, and $d^2(w, W) := \min_{\bar{w} \in W} \|w - \bar{w}\|_2^2$ denotes the squared Euclidean distance between w and the set W.

Instead of minimizing $g(w)$ directly, cutting plane methods minimize it approximately by iteratively solving a linear program arising from its lower bound. Bundle methods are cutting plane methods stabilized with the Moreau-Yosida regularizer. Formally, they add a $\|w - w_{t-1}\|^2$ regularizer term to the objective to prevent the solution at time step t from moving too far away from the previous solution w_{t-1}. Note that our algorithm is closely related but not identical to bundle methods. In our case, the regularizer is already built into the objective function. We describe details now.

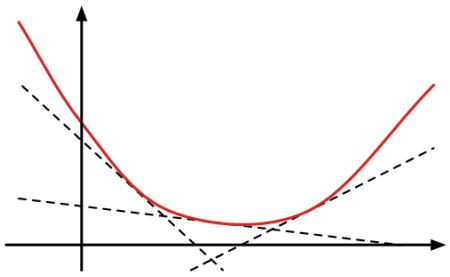

Figure 1: A convex function (solid) is bounded from below by Taylor approximations of first order (dashed). Adding more terms improves the bound.

2.1 Solving Regularized Risk Minimization

Since the loss function is l is assumed to be non-negative and convex, it follows that $R_{\text{emp}}(w)$ is also convex and lower bounded by 0. Moreover, denote

$$a_{i+1} := \partial_w R_{\text{emp}}(w_i) \text{ and } b_{i+1} := R_{\text{emp}}(w_i) - \langle a_i, w_i \rangle .$$

From (3) it follows that

$$R_{\text{emp}}(w) \geq \langle a_i, w \rangle + b_i \text{ for all } i. \tag{6}$$

$$\text{Let } R_i(w) := \max(0, \max_{j \leq i} \langle a_j, w \rangle + b_j) \tag{7}$$

$$\text{and } J_i(w) := \lambda \Omega(w) + R_i(w). \tag{8}$$

We are now able to define our algorithm to minimize $J(w)$.

Algorithm 1 Bundle Method

Initialize $i = 1$, $w_0 = 0$, and $W = \{w_0\}$.
repeat
  Compute gradient a_i and offset b_i.
  Find minimizer $w_i := \text{argmin}_w J_i(w)$
  Update $W \leftarrow W \cup \{w_i\}$ and $i \leftarrow i + 1$.
until converged

The following lemma shows that Algorithm 1 makes continuous progress towards the optimal solution.

Lemma 1 *Denote by w^* the minimizer of $J(w)$ and let J^* be its minimum value. Then, the following holds:*

$$J_i^+ := \min_{j \leq i} J_{j+1}(w_j) \geq J^* \text{ and } J_i^- := J_i(w_i) \leq J^*. \tag{9}$$

Moreover, the series J_i^- is monotonically increasing and the series J_i^+ is monotonically decreasing.

The value $J_i^+ - J_i^-$ is a lower bound on the duality gap $J_{i+1}(w_i) - J_i(w_i)$, and our algorithm stops once this quantity is reduced below a pre-specified tolerance.

2.2 Constrained Convex Optimization

In order to make progress in our analysis and in the implementation, we rewrite the problem of minimizing $J_i(w)$ as a constrained optimization problem. One can check that this amounts to solving

$$\underset{w,\xi}{\text{minimize}} \ \lambda\Omega(w) + \xi \tag{10a}$$

$$\text{subject to } \langle a_j, w\rangle + b_j \leq \xi \text{ for all } j \leq i \text{ and } \xi \geq 0. \tag{10b}$$

We proceed to analyzing ℓ_1 and ℓ_2 regularization.

Linear programming: If $\Omega(w) = \|w\|_1$ we can cast the above problem as a linear program via[1]

$$\underset{\xi,v,u}{\text{maximize}} \ -\xi - \lambda \mathbf{1}^\top (u+v) \tag{11}$$
$$\text{subject to } [-\mathbf{1}, A, -A][\xi, u, v]^\top \leq -b.$$

Here A denotes the matrix $[a_1\ a_2\ \dots\ a_i]$, b the vector $[b_1\ b_2\ \dots\ b_i]^\top$, and $\mathbf{1}$ the vector of all ones. Off-the-shelf linear programming solvers solve (11) efficiently.

Quadratic programming: If $\Omega(w) = \frac{1}{2}\|w\|_2^2$, the dual problem of (10) provides an efficient formulation:

$$\underset{\alpha}{\text{maximize}} \ D_i(\alpha) := -\frac{1}{2\lambda}\alpha^\top A^\top A\alpha + \alpha^\top b \tag{12a}$$
$$\text{subject to } \alpha \geq \mathbf{0} \text{ and } \mathbf{1}^\top\alpha \leq 1. \tag{12b}$$

Moreover, $w = -\frac{1}{\lambda}A\alpha$. Note that the size of the problem only grows in the size of i and that the gradients a_i only appear in the form of inner products. The dominant time complexity is to form $A^\top A$.

Since the instances of (10), e.g. the linear and quadratic programs, will not change substantially after every iteration (we only add one more constraint at a time), a hotstart solver can be suitable to update the solution. Hence the time spent in the solver is typically small in comparison to the cost of computing gradients.

Theorem 2 *Let $G \geq \|\partial_w R_{\text{emp}}(w)\|$ be a bound on the norm of the subdifferential, and let $\Omega(w) = \frac{1}{2}\|w\|^2$. Then the bundle method produces a duality gap of at most ϵ after t steps, where*

$$t \leq \log_2 \lambda R_{\text{emp}}(0) - 2\log_2 G + \tfrac{8G^2}{\lambda\epsilon} - 4. \tag{13}$$

Note that this bound is significantly better than that of [40, 35], since it only depends *logarithmically* on the value of the loss and offers an $O(1/\epsilon)$ rate of convergence rather than the $O(1/\epsilon^2)$ rate in previous papers. This is largely due to an improved analysis and would easily translate to algorithms of the SVMStruct type. It explains why the number of steps required is often considerably less than those predicted in theory — the previous bounds were rather loose.

Corollary 3 *Whenever the norm of the subdifferentials of $R_{\text{emp}}(w)$ is bounded for all $\|w\| \leq \sqrt{2R_{\text{emp}}(0)/\lambda}$, the bundle method will converge to any given precision.*

Corollary 4 *The bundle method converges for any continuously differentiable loss function l.*

[1]We use $x \geq \mathbf{0}$ to imply $x_i \geq 0$ for all i.

3. LOSS FUNCTIONS

A multitude of loss functions are commonly used to derive seemingly different algorithms. This often blurs the similarities as well as subtle differences between them, often for historic reasons: Each new loss is typically accompanied by at least one publication dedicated to it. We now discuss some commonly used loss functions. Tables 1 and 2 contain a choice subset of simple losses. More elaborate ones are discussed below.

3.1 Scalar Loss Functions

In the simplest case we may write $l(x, y, w) = \bar{l}(\langle x, w\rangle, y)$, as described in Table 1. In this case a simple application of the chain rule yields that $\partial_w l(x, y, w) = \bar{l}'(\langle x, w\rangle, y) \cdot x$. For instance, for squared loss we have

$$\bar{l}(\langle x, w\rangle, y) = \tfrac{1}{2}(\langle x, w\rangle - y)^2 \text{ and } \bar{l}'(\langle x, w\rangle, y) = \langle x, w\rangle - y.$$

This means that if we want to compute l and $\partial_w l$ on a large number of observations x_i, represented as matrix X, we can make use of fast linear algebra routines to precompute

$$f = Xw \text{ and } g^\top X \text{ where } g_i = \bar{l}'(f_i, y_i).$$

This is possible for any of the 13 loss functions (and many more) listed in Table 1. The advantage of this unified representation is that implementation of each individual loss can be done in very little time. The computational infrastructure for computing Xw and $g^\top X$ is shared.

Matters are slightly more complicated when maximizing the area under the ROC curve, various F_β scores, Precision@k, and ordinal regression losses, as proposed in [25]. All those functions rely on $\langle w, x_i\rangle$ to perform classification or ranking between the observations x_i. Hence, we may use fast linear algebra to pre-compute $f = Xw$. Subsequently we sort f by the size of its values, which yields the permutation π, i.e., the vector $f_{\pi(i)}$ is sorted. We now describe the operations needed for multivariate scores:

ROC Score: Let us assume that f is sorted, n_+ denote the number of positive examples, n_- the number of negative examples, and $n = n_+ \times n_-$ the total number of pairs whose labels do not match. It is well known that the ROC score is the fraction of examples ranked in the correct order, i.e., number of pairs (i, j) such that $f_i \leq f_j$ for $y_i < y_j$ divided by n. Assume that $y_{ij} \in \{\pm 1\}$. [24] shows that this translates into the loss $l(X, y, w)$, which is given by

$$\max_{\mathbf{y}' \in \{\pm 1\}^n} \Big[\sum_{y_i > y_j} [y_{ij} - 1] \left[\langle x_i - x_j, w\rangle - \tfrac{1}{2}\right] \Big].$$

Moreover, [24] shows that this can be maximized and the terms $\sum_i [y_{ij} - 1]$ and $\sum_j [y_{ij} - 1]$ for the maximizer y_{ij} can be obtained in linear time, once f is sorted. This allows us to compute the gradient $\partial_w l(X, y, w)$

$$\sum_{y_i = 1} x_i \sum_j [y_{ij} - 1] - \sum_{y_j = -1} x_j \sum_i [y_{ij} - 1].$$

Ordinal Regression performs essentially the same operation. The only difference is that y_i need not take on binary values any more. Instead, we may have an arbitrary number of different values y_i (e.g. 1 corresponding to 'strong reject' up to 10 corresponding to 'strong accept', when it comes to ranking papers for a conference). [25] generalizes the results of [24] to show that also in this case the value and gradients of l can be computed in linear time, once f is sorted.

Table 1: Scalar loss functions and their derivatives, depending on $f := \langle w, x \rangle$, and y.

	Loss $l(f, y)$	Derivative $l'(f, y)$
Hinge [20]	$\max(0, -yf)$	0 if $yf \geq 0$ and $-y$ otherwise
Squared Hinge [26]	$\frac{1}{2}\max(0, -yf)^2$	0 if $yf \geq 0$ and f otherwise
Soft Margin [4]	$\max(0, 1 - yf)$	0 if $yf \geq 1$ and $-y$ otherwise
Squared Soft Margin [10]	$\frac{1}{2}\max(0, 1 - yf)^2$	0 if $yf \geq 1$ and $f - y$ otherwise
Exponential [14]	$\exp(-yf)$	$-y\exp(-yf)$
Logistic [13]	$\log(1 + \exp(-yf))$	$-y/(1 + \exp(yf))$
Novelty [32]	$\max(0, 1 - f)$	0 if $f \geq 0$ and -1 otherwise
Least mean squares [43]	$\frac{1}{2}(f - y)^2$	$f - y$
Least absolute deviation	$\lvert f - y \rvert$	$\operatorname{sgn}(f - y)$
Quantile regression [27]	$\max(\tau(f - y), (1 - \tau)(y - f))$	τ if $f > y$ and $\tau - 1$ otherwise
ϵ-insensitive [41]	$\max(0, \lvert f - y \rvert - \epsilon)$	0 if $\lvert f - y \rvert \leq \epsilon$ and $\operatorname{sgn}(f - y)$ otherwise
Huber's robust loss [31]	$\frac{1}{2}(f - y)^2$ if $\lvert f - y \rvert < 1$, else $\lvert f - y \rvert - \frac{1}{2}$	$f - y$ if $\lvert f - y \rvert \leq 1$, else $\operatorname{sgn}(f - y)$
Poisson regression [16]	$\exp(f) - yf$	$\exp(f) - y$

Table 2: Vectorial loss functions and their derivatives, depending on the vector $f := Wx$ and on y.

	Loss	Derivative
Soft Margin [38]	$\max_{y'}(f_{y'} - f_y + \Delta(y, y'))$	$e_{y^*} - e_y$, where y^* is the argmax of the loss
Scaled Soft Margin [40]	$\max_{y'} \Delta^\beta(y, y')(f_{y'} - f_y + \Delta(y, y'))$	$\Delta^\beta(y, y')(e_{y^*} - e_y)$, where y^* is the argmax of the loss
Softmax [14]	$\log \sum_{y'} \exp(f_{y'}) - f_y$	$\left[\sum_{y'} e_{y'} \exp(f'_y)\right] / \sum_{y'} \exp(f'_y) - e_y$
Multivariate Regression	$\frac{1}{2}(f - y)^\top M(f - y)$ where $M \succeq 0$	$M(f - y)$

Document Ranking [29] show that a large number of ranking scores (normalized discounted cumulative gain, mean reciprocal rank, expected rank utility, etc.) can be optimized directly by minimizing the following loss:

$$l(X, y, w) = \max_{\pi} \sum_i c_i \left\langle x_i - x_{\pi(i)}, w \right\rangle + \langle a - a(\pi), b(y) \rangle .$$

Here c_i is a monotonically decreasing sequence, the documents are assumed to be arranged in order of decreasing relevance, π is a permutation, the vectors a and $b(y)$ depend on the choice of a particular ranking measure, and $a(\pi)$ denotes the permutation of a according to π. In this case, a linear assignment algorithm will allow us to find the permutation maximizing the loss and compute the gradients subsequently.

A similar reasoning applies to F_β scores [24]. Note that in all cases we may use fast linear algebra routines to accelerate the computationally intensive aspects Xw and $g^\top X$.

3.2 Vector Loss Functions

Next we discuss "vector" loss functions, i.e., functions where w is a matrix (denoted by W) and the loss depends on Wx. Table 2 contains a number of cases (Note that in the table we use e_i to denote the i-th canonical basis vector). Most notable are the cases of structured estimation, where

$$l(x, y, W) = \max_{y'} \langle w_y - w_{y'}, x \rangle + \Delta(y, y')$$

is a large-margin loss [38] and $\Delta(y, y')$ is the misclassification error by confusing y with y'. Rescaled versions were proposed by [40] (we have $\beta = 1$ in Table 2). Log-likelihood scores of exponential families share similar expansions.

In these cases we may again take recourse to efficient linear algebra routines and compute $f = XW$, which is now a matrix by means of a matrix-matrix multiplication. Likewise, gradients are efficiently computed by $g^\top X$, where g is now a matrix of the dimensionality of the number of classes. Let us discuss the following two cases:

Ontologies for Structured Estimation: For hierarchical labels, e.g. whenever we deal with an ontology [7], we can use a decomposition of the coefficient vector along the hierarchy of categories.

Let d denote the depth of the hierarchy tree, and assume that each leaf of this tree corresponds to a label. We represent each leaf as a vector in $\mathbb{N}^d$, which encodes the unique path from the root to the leaf. For instance, if the tree is binary, then we have $y \in \{0, 1\}^d$.

We may describe the score for class y by computing $\langle w_y, x \rangle = \sum_{j=1}^d \langle w_{y_1,\ldots,y_j}, x \rangle$, i.e., by summing over the vectors $w_{y_1,\ldots,y_j}$ along the path from w_{y_1} to $w_{y_1,\ldots,y_d}$. Assuming lexicographic order among vectors w, we can precompute the inner products by a matrix-matrix multiplication between the matrix of all observations X and W. A simple dynamic programming routine (depth-first recursion over the ontology tree) then suffices to obtain the argmax and the gradients of the loss function for structured estimation. See [7] for implementation details.

Logistic Hierarchical Model: The same reasoning applies to estimation when using an exponential families model. The only difference is that we need to compute a *soft-max* over paths rather than exclusively choosing the best path over the ontology. Again, a depth-first recursion suffices.

3.3 Structured Estimation

In this case, we need to solve one of the two problems:

$$l(x, y, w) = \max_{y'} \Delta^\beta(y, y') \left\langle \phi(x, y') - \phi(x, y), w \right\rangle + \Delta(y, y')$$

or $l(x, y, w) = \log \sum_{y'} \exp \left\langle \phi(x, y'), w \right\rangle - \langle \phi(x, y), w \rangle .$

In both cases, $\phi(x, y)$ is a user-defined feature map which describes the relationship between data and labels, e.g. by

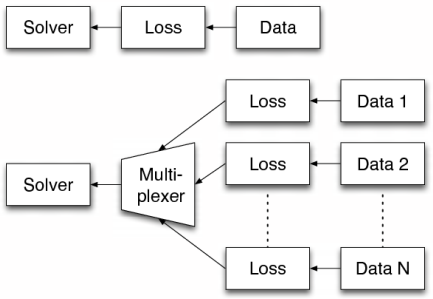

Figure 2: The architecture of our solver. Top: serial version. Bottom: parallel version. Loss values and their gradients, computed on subsets of the data (Data 1 to Data N), are combined for the overall gradient computation.

means of a set of cliques as in the case of conditional random fields [28] or a Max-Margin-Markov network [38]. Note that in [38] $\beta = 0$, whereas in [40] $\beta = 1$.

Gradients and function values are computed by dynamic programming. In the first case this is done by solving for the argmax y' of the loss, which yields $\phi(x, y') - \phi(x, y)$ as the gradient. In the second case, this is achieved by computing a *soft-max*, or equivalently the expected value of $\phi(x, y')$ with respect to the exponential families distribution induced by the sufficient statistics $\phi(x, y)$.

Note that the user only needs to implement these operations for his model to take advantage of the overall optimization and parallelization infrastructure, including the access to a range of different regularizers $\Omega(w)$.

4. ARCHITECTURE

Recall that Algorithm 1 has two distinct computationally intensive stages: generating a_i and b_i, which requires computing the value and derivative of the empirical risk R_{emp}, and subsequently solving a quadratic program. The latter, however, only contributes to a small amount to the overall cost of solving the optimization problem, in particular for large amounts of data, as we shall see in Section 5. Note also that in addition to our bundle method approach there exists a large number of alternative optimization methods which are able to solve (1) efficiently, provided that they have access to the value $R_{\text{emp}}(w)$ and the gradient $\partial_w R_{\text{emp}}(w)$.

Keeping in line with this observation, our architecture abstracts out the computation of $R_{\text{emp}}(w)$ and $\partial_w R_{\text{emp}}(w)$ from the bundle method solver (see Figure 2). The solver part deals with the regularizer $\Omega(w)$ and is able to query the loss function for values of $R_{\text{emp}}(w)$ and $\partial_w R_{\text{emp}}(w)$ as needed. This is very similar to the design of the Toolkit for Advanced Optimization (TAO) [5].

Depending on the type of loss function, computing R_{emp} can be very costly. This is particularly true in cases where $l(x, y, w)$ is the log-likelihood of an intractable conditional random fields or of a corresponding Max-Margin-Markov network. This effect can be mitigated by observing that both $R_{\text{emp}}(w)$ and its gradient can be computed in parallel by chunking the data and computing the contribution due to each chunk separately. In effect, we insert a multiplexer between the solver and the loss functions (see Figure 2). This multiplexer has the sole purpose of broadcasting the values of w to the losses and summing over the values and gradients of the individual losses before passing those values on to the solver. In practice, this is achieved by invoking `MPI::AllReduce()`, which allows hierarchical aggregation.

The loss functions themselves only interact with the solver via an interface which queries their values and gradients. This means that it is trivial to add losses or solvers without changing the overall architecture.

4.1 Benefits

By taking advantage of the unified design of Algorithm 1 our algorithm has several advantages over custom-built solvers:

Simplicity: Our setup is surprisingly straightforward. This translates into a relatively compact code, despite the large number of problems solved.

Modularity in the solver: It is possible to add more solvers without any need to change the problem itself, provided that all they need is value and first order information about the empirical risk term. In fact, we can take advantage of off-the-shelf packages instead of having to build our own in many cases (see next section).

Modularity in the loss function: In the same fashion it is possible to add more loss functions without any need to change the rest of the architecture. This has allowed us to plug in over 30 loss functions.

Parallelization: Since dealing with $R_{\text{emp}}(w)$ is the dominant part it is easy to parallelize this by distributing calculations on the data over a number of computers.

Vectorization: Many of the computations can be done more efficiently by using floating point accelerators such as General Purpose GPUs.

One of the major advantages of our approach is that data can be stored locally on the units computing values and gradients of the loss functions. It is not necessary that the main solver has access to the data. Nor is it necessary that individual nodes *share* the data, since all communication revolves around sharing values and gradients of $R_{\text{emp}}[w]$.

- This has the added benefit of preserving a large degree of privacy [12] between the individual database owners and the system using the solver. At every step the data owner will only return a gradient which is the convex combination of a set of observations. Assuming that we use a loss function whose derivative can only take on a small number of different values, we will not be able to reconstruct the vectors x_i from it. In addition to that, each data owner may check whether $R_{\text{emp}}[w] + \lambda\Omega[w]$ is decreasing on his portion of the data. If this is severely violated it gives him a good indication that the interface is being abused for snooping on the content of the database.
- Note that distributed data also has the advantage that the problem of file I/O is diminished since disk access is local. Given that modern disks are a factor of 50 slower than memory bandwidth and by a factor of 2 slower than computer networks, this alleviates a significant bottleneck.

4.2 Off-the-shelf Methods

Since our architecture is modular (see figure 2), we show as a proof of concept that it can deal with different types of solvers, such as an implementation of LBFGS [6] from TAO [5]. There are two additional requirements: First, we also

need to provide a subdifferential and value of the regularizer $\Omega(w)$. This is easily achieved via

$$\frac{1}{2}\partial_w \|w\|_2^2 = w \text{ and } \partial_w \|w\|_1 \ni \operatorname{sgn} w. \quad (14)$$

Second, we need a method to assess solution quality.

Lemma 5 *Let $x \in \mathbb{R}^n$, assume that f is convex on $\mathbb{R}^n$ and let $\lambda > 0$. Moreover let $g(x) \in \partial_x f(x) + \lambda x$. Then we have*

$$\min_x f(x) + \tfrac{\lambda}{2}\|x\|_2^2 \geq f(x_0) + \tfrac{\lambda}{2}\|x_0\|_2^2 - \tfrac{1}{2\lambda}\|g(x_0)\|_2^2. \quad (15)$$

Using the above lemma we can lower bound the minimum of the regularized risk functional. Using the definition

$$J_-(w) := R_{\text{emp}}(w) + \tfrac{\lambda}{2}\|w\|_2^2 - \tfrac{1}{2\lambda}\|\partial_w R_{\text{emp}}(w) + \lambda w\|_2^2$$

we are able to bound the number of significant figures via

$$\text{SigFig}(w) \geq \log_{10} \tfrac{2(J(w)-J_-(w))}{|J(w)|+|J_-(w)|}. \quad (16)$$

This provides us with a good convergence monitor for all cases using quadratic regularization. Note that bound is not applicable when we use the ℓ_1 regularizer $\Omega(w) = \|w\|_1$. However, this is a situation where convergence with LBFGS is poor, since $\|w\|_1$ is not continuously differentiable in w.

While the bundle methods use the past gradients to lower bound the convex objective function, BFGS is a quasi-Newton method that uses the past gradients to estimate the inverse of the Hessian. Furthermore, to bound the memory and computational costs associated with storing and updating the complete inverse Hessian, the LBFGS algorithm uses only the past n gradients (n is user defined).

LBFGS is known to perform well on continuously differentiable problems, such as logistic regression, least-mean-squares problems, or conditional random fields [34]. But, if the functions are not continuously differentiable (e.g., the hinge loss and its variants) then LBFGS may fail. Empirically, we observe that LBFGS does converge well even for the hinge losses if we have a large amount of data at our disposition. We conjecture that this is due to the fact that unlike in the $\|w\|_1$ regularization, the non-differentiability is "smoothed-out" by averaging over many losses.

5. EXPERIMENTS

We now demonstrate that our algorithm is a) scalable in the number of observations, b) scalable when λ decreases c) it parallelizes well, d) it is versatile, and e) it is faster than any competing specialized solver on the subproblems it solves.

The experiments were carried out on a cluster of 1928 1.6Ghz Itanium2 cpus and 3.2 Gbytes/s bidirectional bandwidth per link[2]. The time reported for the experiments are the CPU time without IO. One exception is for parallel experiments where we report the CPU time and network IO time.

5.1 Datasets

We focus on the following 6 datasets. Our choice is largely influenced by the choices in [24, 25] concerning classification tasks for the purpose of reproducing results reported in the work. For regression tasks, we pick some of the largest datasets in Luís Torgo's website[3]. Since some of the regression datasets are highly nonlinear, we perform a low rank incomplete Cholesky factorization [19] and keep the dimension less than 1000.

For ranking tasks, we use an MSN web search data with 1000 queries.

Table 3: Datasets their properties.

dataset	abbr.	#examples	dimension	density %
CLASSIFICATION				
Adult9	adult9	32561	123	11.28
Real & Simulated	real-sim	57763	20958	0.25
Autos & Aviation	aut-avn	56862	20707	0.25
Web8	web8	45546	300	4.24
KDDCup-99	kdd99	4898431	127	12.86
Covertype	covertype	522911	54	22.22
Reuters CCAT	ccat	23149	47236	0.16
Reuters C11	c11	23149	47236	0.16
20 Newsgroups	news20	15960	1355181	0.03
REGRESSION				
MV	mv	40000	126	99.82
Friedman	fried	40000	1000	98.77
2D planes	cart	40000	1000	98.04
Pole Telecomm	pol	10000	1000	96.49
RANKING				
MSN ranking	msn	45882 1000 queries	368	99.73
PARALLEL				
Reuters CCAT	ccat	804414	47236	0.16
Reuters C11	c11	804414	47236	0.16
Arxiv astro-ph	astro-ph	62369	99757	0.08
Covertype	covertype	522911	54	22.22

5.2 Software

For classification experiments, we compare our software with `svmperf` and `libsvm`. For regression experiments, we compare our solver with `svmlight` and `libsvm`. To the best of our knowledge, these are the state-of-the-art solvers at the moment. Our solver is called `bmrm`.

5.3 Classification

In this experiment, we carry out experiments with soft-margin (hinge) loss with various values of {1.0, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00001, 3e-05, 1e-05, 3e-06}. The results are shown in Figure 3. As can be seen from the figure, our solver is usually faster than `svmperf` and significantly outperforms `libsvm`. The difference between our method and `svmperf` is very small in log-log scale but in reality our method can be a lot faster than `svmperf` ranging from 2 to 10 times.

5.4 Regression

The next step, we investigate the behavior of our solver for regression problems and particularly the ϵ-insensitive loss function. The values of λ that we choose belong to the set {1.0, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00001, 3e-05, 1e-05, 3e-06}. The results are shown in Figure 4. As can be seen from the figure, our solver is significantly faster than both `svmlight` and `libsvm`.

5.5 Scaling behavior with sample size

For the scaling experiments in the sample size, we fixed $\lambda = 10^{-5}$ and computed the time for a binary soft-margin classifier. To see how the algorithm behaves for different values of λ, we used the full datasets and show how λ affects the total runtime.

As can be seen in Figure 5, the time required to solve a problem is essentially linear in the sample size. This behavior is similar to `svmperf` as observed by [25]. The remark-

[2] `http://nf.apac.edu.au/facilities/ac/hardware.php`

[3] `http://www.liacc.up.pt/~ltorgo/Regression/DataSets.html`

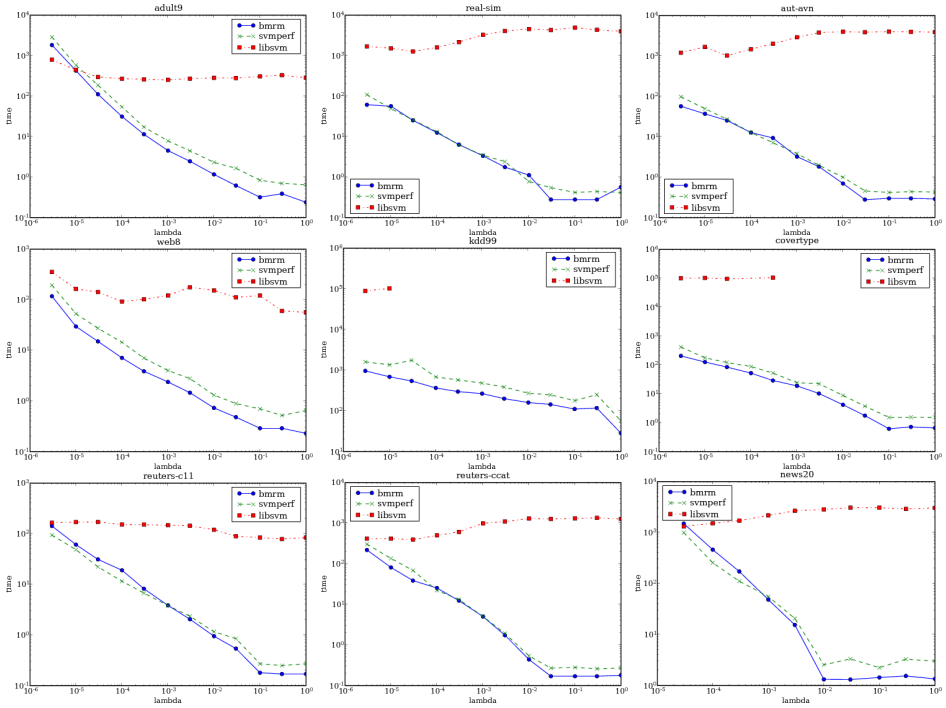

Figure 3: Classification runtime of three algorithms (log-log scale)

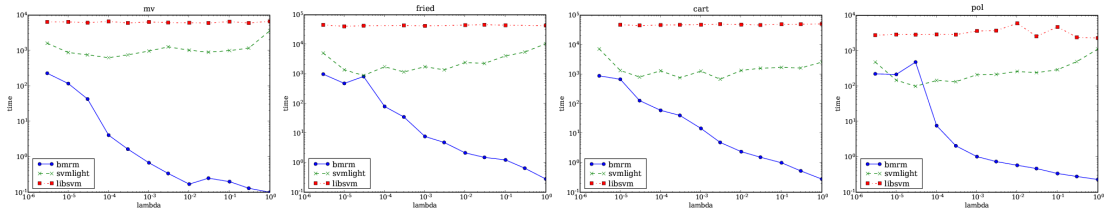

Figure 4: Regression runtime of three algorithms (log-log scale)

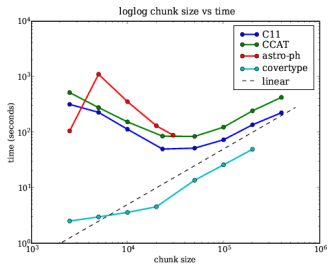

Figure 5: Chunk size vs time (log-log scale)

able fact is that these good properties are consistent between both choices of regularization. Both are problems which typically require very different solvers to perform well, yet in our approach they are both effectively tractable by using the same method. Also note that the time required to *load* the dataset often exceeded the time to perform estimation on it.

It is interesting to see that λ influences the runtime performance of our algorithm more than the sample size does.

5.6 Scaling behavior with number of computers

A second test investigates the scalability of our method. For this purpose we ran our solver on a cluster of workstations. We perform the experiments with three different losses: soft-margin loss, NDCG loss [29], and ordinal regression with several values of λ and datasets.

The number of computers we use ranges from {1, 2, 4, 8, 16, 32, 40, 48, 56, 64}. We report the CPU time + network communication time in Figure 6.

Figure 6 shows the algorithm for hinge loss scales linearly in the number of nodes, i.e. the time is given by $T(n) = a + b/n$. Here a is the time required to solve the inner subproblem using linear or quadratic programming solvers. This part is not parallelized yet in the current implementation. The advantage of our parallelization would be even more apparent if we were using datasets several orders of magnitude larger than the publicly available collections, since in this case a would remain constant, however b scales with the amount of data required for processing.

For ranking experiments, the algorithm does not scales as well as soft-margin. The main reason is that the dataset is grouped into queries, and some queries have more documents than the other. The nodes that have queries that have more documents tend to take much longer than other nodes and this forces other nodes to wait for results. However, the

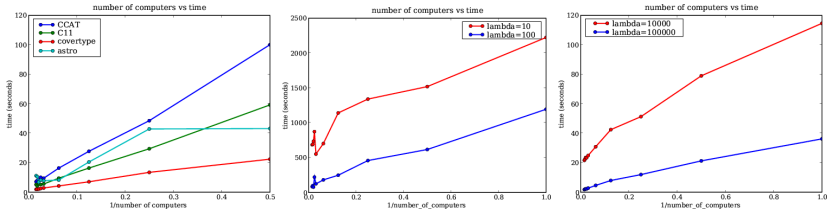

Figure 6: Parallel classification, NDCG ranking, and ordinal regression with bmrm (log-log scale)

results are still very promising.

5.7 Convergence

Last we address the issue of the speed of convergence and its dependence on the value of λ. In Figure 7, we plot the tolerance as a function of number of interations for various values of λ for many datasets. Not surprisingly we see that the number of iterations depends on λ: for decreasing λ the number of iterations increases. This is consistent with Theorem 2. For most datasets, the rate starts very fast then slows down and becomes linear.

5.8 Versatility

It is worth noting that throughout previous experiments, we have demonstrated that our solver can deal with many loss functions from classification, regression to ranking. There are many more loss functions that our solver can deal with such as quantile estimation, novelty detection, least squared regression etc. For these problems our method works really well and exhibits similar nice convergence properties as described in previous sections. Unfortunately, it is very hard to find competing software to compare our solver with, and due to space constraint, we choose to omit the results. Results for these problems will be published in a separate report.

6. OUTLOOK AND DISCUSSION

Related Work Our work is most closely related to the prize-winning paper of Joachims [25]. In fact, for a *particular* solver, namely a bundle method, a set of loss functions l, namely binary ℓ_1 soft-margin loss, ROC-scores, F_β scores and ordinal regression, and a *particular* regularizer Ω, namely quadratic regularization, both methods are equivalent. The advantage in our solver is the use of efficient linear algebra tools via PETSc [2], the modular structure, the considerably higher generality in both loss functions and regularizers, and the fact that data may remain local.

Moreover, our work is related to [11], where MapReduce is used to accelerate machine learning on parallel computers. We use similar parallelization techniques to distribute the computation of values and gradients of the empirical risk $R_{\text{emp}}(w)$ to a cluster of workstations. Given the lack of availability of a robust MapReduce implementation (only Java versions such as Hadoop are freely accessible), our implementation details differ significantly.

Finally, several papers [26, 10] advocate the use of Newton-like methods to solve Support Vector Machines in the "primal". However, they need to take precautions when dealing with the fact that the soft-margin loss function used in an SVM is only piecewise differentiable. Instead, our method only requires *subdifferentials*, which always exist for convex functions, in order to make progress. The large number of and variety of implemented problems shows the flexibility.

Extensions and Future Work We believe that the framework presented in this paper will prove very fruitful in the future. Besides applications of the solver to other estimation problems, such as covariate shift correction and distributed novelty detection there are a number of extensions:

- The acceleration of the optimization by means of a specialized stream processor, i.e. high end graphics cards.
- Note that we can perform the optimization in feature space, as long as inner products between the gradients can be computed efficiently. This is the case for certain string and graph kernels.
- The quadratic programming part of the our algorithm can be reduced by the simple line-search used in the proof of Theorem 2. This makes our method amenable to implementation in computationally constrained systems, such as sensor networks.

Summary On a wide variety of datasets our algorithm outperformed state-of-the-art custom built solvers, converging to the optimal solution within 20 - 200 iterations through the dataset. Unlike other bespoke custom built solvers, our software is open source and freely available for download. We invite the research community to contribute to the codebase as a shared resource (`http://elefant.developer.nicta.com.au/`).

Acknowledgments We thank Sam Roweis and Yasemin Altun for helpful discussions. Part of this work is done when Quoc Le is at NICTA. We sincerely thank ANUSF team for their great help with the experiments. NICTA is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

7. REFERENCES

[1] G. Bakir, T. Hofmann, B. Schölkopf, A. Smola, B. Taskar, and S. V. N. Vishwanathan. *Predicting Structured Data*. MIT Press, Cambridge, Massachusetts, 2007.

[2] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11, Argonne National Laboratory, 2006.

[3] O. E. Barndorff-Nielsen. *Information and Exponential Families in Statistical Theory*. John Wiley and Sons, New York, 1978.

[4] K. P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optim. Methods Softw.*, 1:23–34, 1992.

[5] S. Benson, L. Curfman-McInnes, J. Moré, and J. Sarich. TAO user manual. Technical Report ANL/MCS-TM-242, Argonne National Laboratory, 2004.

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

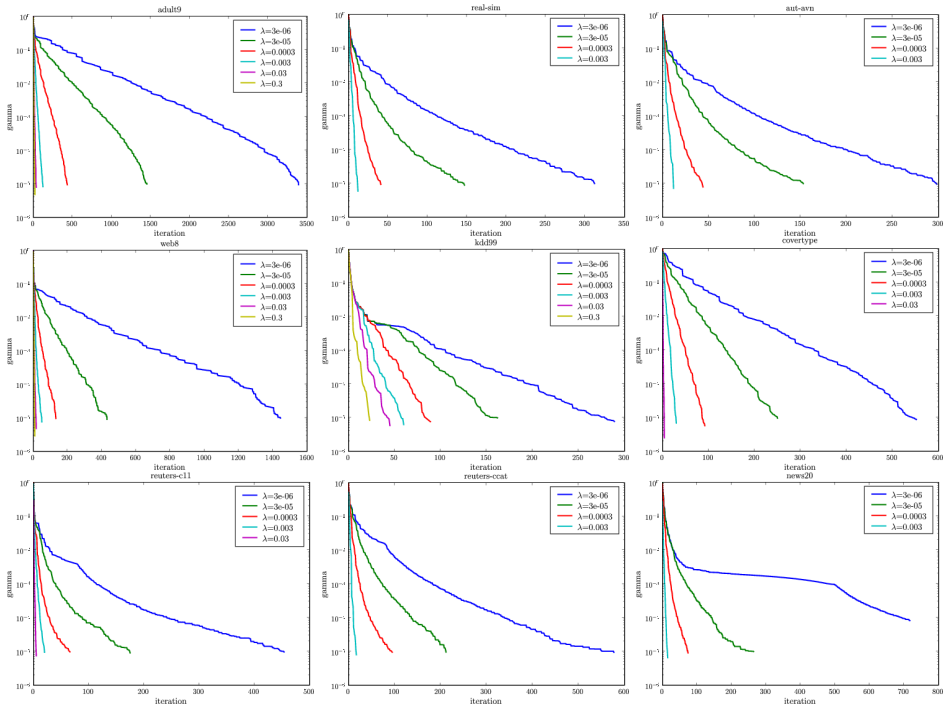

Figure 7: Convergence of bmrm (semilog-y), γ_i (see Appendix) as a function of number of iterations.

[7] L. Cai and T. Hofmann. Hierarchical document categorization with support vector machines. In *Proc. of ACM conference on info. and knowledge mgmt.*, pages 78–87, New York, NY, USA, 2004. ACM Press.

[8] E. Candes and T. Tao. Decoding by linear programming. *IEEE Trans. Info Theory*, 51(12):4203–4215, 2005.

[9] C. Chang and C. Lin. *LIBSVM: a library for support vector machines*, 2001.

[10] O. Chapelle. Training a support vector machine in the primal. Technical Report TR.147, Max Planck Institute for Biological Cybernetics, 2006.

[11] C. Chu, S. Kim, Y. A. Lin, Y. Y. Yu, G. Bradski, A. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *NIPS 19*, 2007.

[12] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, , and M. Zhu. Tools for Privacy Preserving Distributed Data Mining. *ACM SIGKDD Explorations*, 4(2), December 2002.

[13] M. Collins, R. E. Schapire, and Y. Singer. Logistic regression, AdaBoost and Bregman distances. In *COLT*, pages 158–169. Morgan Kaufmann, San Francisco, 2000.

[14] R. Cowell, A. David, S. Lauritzen, and D. Spiegelhalter. *Probabilistic Networks and Expert Sytems*. Springer, New York, 1999.

[15] K. Crammer and Y. Singer. Online ranking by projecting. *Neural Computation*, 17(1):145–175, 2005.

[16] N. A. C. Cressie. *Statistics for Spatial Data*. John Wiley and Sons, New York, 1993.

[17] L. Fahrmeir and G. Tutz. *Multivariate Statistical Modelling Based on Generalized Linear Models*. Springer, 1994.

[18] S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representation. Technical report, IBM Watson Research Center, New York, 2000.

[19] S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *JMLR*, 2001.

[20] C. Gentile and M. K. Warmuth. Linear hinge loss and average margin. In *NIPS 11*, pages 225–231, Cambridge, MA, 1999.

[21] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 115–132, Cambridge, MA, 2000. MIT Press.

[22] J. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms, I and II*. 305 and 306. Springer-Verlag, 1993.

[23] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184, Cambridge, MA, 1999. MIT Press.

[24] T. Joachims. A support vector method for multivariate performance measures. In *ICML*, pages 377–384, San Francisco, California, 2005. Morgan Kaufmann Publishers.

[25] T. Joachims. Training linear SVMs in linear time. In *KDD*, 2006.

[26] S. S. Keerthi and D. DeCoste. A modified finite Newton method for fast solution of large scale linear SVMs. *JMLR*, 6:341–361, 2005.

[27] R. Koenker. *Quantile Regression*. Cambridge University Press, 2005.

[28] J. D. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic modeling for segmenting and labeling sequence data. In *ICML*, volume 18, pages 282–289, 2001.

[29] Q. Le and A. Smola. Direct optimization of ranking measures. *JMLR*, 2007. submitted.

[30] O. L. Mangasarian. Linear and nonlinear separation of patterns by linear programming. *Oper. Res.*, 13:444–452, 1965.

[31] K.-R. Müller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In *ICANN'97*, pages 999–1004, 1997.

[32] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. TR 87, Microsoft Research, Redmond, WA, 1999.

[33] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Comput.*, 13(7):1443–1471, 2001.

[34] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proceedings of HLT-NAACL*, pages 213–220, 2003.

[35] S. Shalev-Shwartz and Y. Singer. Online learning meets optimization in the dual. In *COLT*, 2006. extended version.

[36] V. Sindhwani and S. Keerthi. Large scale semi-supervised linear svms. In *SIGIR '06*, pages 477–484, New York, NY, USA, 2006. ACM Press.

[37] I. Takeuchi, Q. Le, T. Sears, and A. Smola. Nonparametric quantile estimation. *JMLR*, 2006.

[38] B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In *NIPS*, pages 25–32, 2004.

[39] R. Tibshirani. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 58:267–288, 1996.

[40] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 6:1453–1484, 2005.

[41] V. Vapnik, S. Golowich, and A. J. Smola. Support vector method for function approximation, regression estimation, and signal processing. In *NIPS*, pages 281–287, 1997.

[42] S. V. N. Vishwanathan and A. J. Smola. Fast kernels for string and tree matching. In *NIPS*, pages 569–576, 2003.

[43] C. K. I. Williams. Prediction with Gaussian processes: From linear regression to linear prediction and beyond. In M. I. Jordan, editor, *Learning and Inference in Graphical Models*, pages 599–621. Kluwer Academic, 1998.

A. PROOFS

PROOF LEMMA 1. Recall that the Taylor expansion is exact for all w_j. Therefore $J(w_j) = J_{j+1}(w_j)$ for all $j \leq i$, and the upper bound on J^* follows. Since $J(w) \geq J_i(w)$ for all w it also follows that their minima satisfy the same inequality. Since w_i is the minimizer of $J_i(w)$ the lower bound follows.

The third claim follows from $J_i(w) \geq J_j(w)$ for all w and all $i \geq j$, hence the series J_i^- is monotonic. Since the min is being taken over a larger set at every iteration, the series J_i^+ is monotonically decreasing. □

To prove Theorem 2 we need an auxiliary lemma:

Lemma 6 *The minimum of $\frac{1}{2}dx^2 - cx$ with $c, d > 0$ and $x \in [0, 1]$ is bounded from above by $-\frac{c}{2}\min(1, c/d)$.*

PROOF. The unconstrained minimum of the problem is $x^* = c/d$ with minimum value $-c^2/2d$. If $c \leq d$, this is also the constrained minimum. For $c > d$ we take $x^* = 1$, which yields $d/2 - c$. The latter is majorized by $-c/2$. Taking the maximum over both minima proves the claim. □

PROOF THEOREM 2. Denote by α^i the maximizer of $D_i(\alpha)$, i.e. the dual problem (12). We know by strong duality that $D_i(\alpha^i) = J_i(w_i)$, which provides a *lower* bound on the minimum J^* of $J(w)$. Moreover, we know by Lemma 1 that

$$\epsilon_i := J_{i+1}(w_i) - J_i(w_i) \geq J_i^+ - J_i^- =: \gamma_i$$

is an *upper* bound on the size of the duality gap and therefore on the quality of the solution. After solving $D_{i+1}(\alpha)$ we will obtain a new lower bound, i.e. $J_{i+1}^- = D_{i+1}(\alpha^{i+1})$. Since J_i^+ is monotonically decreasing, it follows that if we can lower bound the increase in J_i^- in terms of ϵ_i we are going to obtain a convergence guarantee for $J_i^+ - J_i^-$, too.

Obviously, we cannot compute $J_{i+1}(w_{i+1})$ explicitly, since it involves solving a quadratic program. However, in order to obtain a lower bound on the improvement, we simply consider a linesearch along the line joining $[\alpha^i, 0]$ to $[0, 1]$. Denote by $\bar{\alpha}(\eta) := \big[\eta\alpha^i, (1-\eta)\big]$ the solution obtained by optimizing (12) along this line, for $\eta \in [0, 1]$. Clearly, any such $\bar{\alpha}(\eta)$ is feasible.

We need to analyze two separate regimes of the line search: a) the maximum is attained for $\eta \in (0, 1)$ and b) the maximum is attained for $\eta = 1$. In order to apply Lemma 6 we need to compute linear and quadratic terms of $D_i(\bar{\alpha}(\eta))$ and express ϵ_i in terms of α^i.

Since the Taylor expansion is exact for all w_j, we have that $J_{i+1}(w_i) = \frac{\lambda}{2}\|w_i\|^2 + \langle a_{i+1}, w_i\rangle + b_{i+1}$. Denote by $\bar{\mathrm{A}}$ denote the extended matrix $[A\ a_{i+1}]$ and by $\bar{\mathrm{b}}$ the extended vector $[b\ b_{i+1}]$. Then $a_{i+1} = \bar{\mathrm{A}}\, e_{i+1}$, $b_{i+1} = \bar{\mathrm{b}}^\top e_{i+1}$, and $w_i = -\frac{1}{\lambda}\bar{\mathrm{A}}[\alpha^i, 0]$. This allows us to write

$$J_{i+1}(w_i) = \underbrace{\tfrac{1}{2\lambda}[\alpha^i, 0]^\top \bar{\mathrm{A}}^\top \bar{\mathrm{A}}[\alpha^i, 0]}_{\lambda\Omega(w_i)} \underbrace{-\tfrac{1}{\lambda} e_{i+1}^\top \bar{\mathrm{A}}^\top \bar{\mathrm{A}}[\alpha^i, 0]}_{\langle a_{i+1}, w_i\rangle} + \underbrace{b^\top e_{i+1}}_{b_{i+1}}$$

Moreover, since $J_i(w_i) = D_i(\alpha^i)$ we may write

$$J_i(w_i) = -\tfrac{1}{2\lambda}[\alpha^i, 0]^\top \bar{\mathrm{A}}^\top \bar{\mathrm{A}}[\alpha^i, 0] + \bar{\mathrm{b}}^\top[\alpha^i, 0].$$

Taking differences yields

$$\epsilon_i = \tfrac{1}{\lambda}[\alpha^i, -1]^\top \bar{\mathrm{A}}^\top \bar{\mathrm{A}}[\alpha^i, 0] + \bar{\mathrm{b}}^\top[-\alpha^i, 1].$$

Next we compute the linear and quadratic terms of $D_i(\bar{\alpha}(\eta))$.

$$\begin{aligned} l &= \partial_\eta D_i(\bar{\alpha}(\eta)) = \epsilon_i \text{ and} \\ q &= \partial_\eta^2 D_i(\bar{\alpha}(\eta)) = \tfrac{1}{\lambda}[\alpha^i, -1]^\top \bar{\mathrm{A}}^\top \bar{\mathrm{A}}[\alpha^i, -1] = \lambda \left\|\partial_w J[w_i]\right\|^2. \end{aligned}$$

Using Lemma 6 it follows directly that the improvement in the dual and therefore the decrease in $\gamma_i - \gamma_{i+1}$ is at least $\frac{l}{2}\min(1, l/q)$. We now bound q further:

Since λw_i is in the convex hull of $a_1 \dots a_i$, it follows that $\|\lambda w_i\|^2 \leq G^2$, and hence $q = \frac{1}{\lambda}\|\lambda w_i - a_{i+1}\|^2 \leq \frac{4G^2}{\lambda}$. This means that $\gamma_i - \gamma_{i+1} \geq \frac{\gamma_i}{2}\min(1, \gamma_i\lambda/4G^2)$. Furthermore, until $\gamma_i \leq \frac{4G^2}{\lambda}$ we have $\gamma_{i+1} \leq \frac{\gamma_i}{2}$. This happens for at most $\log_2\left(\frac{\gamma_0\lambda}{4G^2}\right)$ steps. But, the initial dual optimization problem is empty, hence its value is 0, and we know that γ_0 is $R_{\mathrm{emp}}(0)$. Therefore, for $\log_2 R_{\mathrm{emp}}(0)\lambda - 2\log_2 G - 2$ steps we halve the upper bound on the improvement.

Subsequently, the reduction in γ_{i+1} is at least $\gamma^2\lambda/8G^2$, leading to the difference equation $\gamma_{i+1} \leq \gamma_i - \gamma_i^2\lambda/8G^2$. Since this is monotonically decreasing, we can upper bound this by solving the differential equation $\gamma'(t) = -\gamma^2(t)\lambda/8G^2$, with the boundary conditions $\gamma(0) = 4G^2/\lambda$. This in turn yields $\gamma(t) = \frac{8G^2}{\lambda(t+2)}$, and hence $t \leq \frac{8G^2}{\lambda\gamma} - 2$. For a given γ we will need $\frac{8G^2}{\lambda\gamma} - 2$ more steps to converge. □

PROOF COROLLARY 3. The initial value of the regularized risk is $R_{\mathrm{emp}}(0)$ and the empirical risk is a nonnegative function. Hence we know that w never must exceed a ball of radius $\sqrt{2R_{\mathrm{emp}}(0)/\lambda}$. Since we start from $w = 0$, we will never exit this ball. Hence G is bounded within this ball and Theorem 2 applies. □

PROOF LEMMA 5. Since f is convex,

$$f(x) + \tfrac{\lambda}{2}\|x\|^2 \geq f(x_0) + \langle x - x_0, \partial_x f(x_0)\rangle + \tfrac{\lambda}{2}\|x\|^2$$

for all x and x_0. The right hand side is minimized by setting $x = -\frac{1}{\lambda}\partial_x f(x_0)$. Plugging this into the lower bound yields the right hand side of (15). □