Information Retrieval Journal manuscript No.
(will be inserted by the editor)

Efficient Algorithms for Ranking with SVMs

O. Chapelle and S. S. Keerthi

July 20, 2009

Abstract RankSVM (Herbrich et al, 2000; Joachims, 2002) is a pairwise method for designing ranking models. SVMLight is the only publicly available software for RankSVM. It is slow and, due to incomplete training with it, previous evaluations show RankSVM to have inferior ranking performance. We propose new methods based on primal Newton method to speed up RankSVM training and show that they are 5 orders of magnitude faster than SVMLight. Evaluation on the Letor benchmark datasets after complete training using such methods shows that the performance of RankSVM is excellent.

Keywords ranking, support vector machines, AUC optimization

1 Introduction

Learning to rank is an important problem in web page ranking, information retrieval and other applications. Several types of machine learning algorithms have been considered for this problem: *pointwise* methods (Cossock and Zhang, 2006) (in these methods each document is encouraged to give a score that is in par with its relevance); *pairwise* methods (Herbrich et al, 2000; Joachims, 2002) (here, if document A is more relevant than document B then the score of A is encouraged to be above the score of B); and, *listwise* methods (Cao et al, 2007) (in this class of methods the training loss function is based on the list of all participating documents and their scores).

In this paper we focus on pairwise methods. Several pairwise ranking methods have been proposed in the literature: RankBoost (Freund et al, 2003), RankNet (Burges et al, 2005), LambdaRank (Burges et al, 2007), GBRank (Zheng et al, 2008) and RankSVM

Olivier Chapelle
Yahoo! Research
Santa Clara, CA
E-mail: chap@yahoo-inc.com

S. Sathiya Keerthi
Yahoo! Research
Santa Clara, CA
E-mail: selvarak@yahoo-inc.com

(Herbrich et al, 2000; Joachims, 2002). These works mainly differ in the type of loss function used and in the way the models are trained.

The RankSVM method forms a ranking model by minimizing a regularized margin-based pairwise loss. Consider a web page ranking problem whose training data comprises: a number of queries; for each query a set of documents; for each (query,document) pair a feature vector $\mathbf{x}_i \in \mathbb{R}^d,\ i = 1, \ldots, n$; and relevance judgments of these documents to the query. From these judgments, a set of preference pairs $\mathcal{P}$ can be constructed by comparing the relevance of the documents associated with a given query. If $(i, j) \in \mathcal{P}$ then document i is preferred over document j. The RankSVM model is built by minimizing the objective function (Herbrich et al, 2000; Joachims, 2002)

$$\frac{1}{2}||\mathbf{w}||^2 + C \sum_{(i,j)\in\mathcal{P}} \ell(\mathbf{w}^\top \mathbf{x}_i - \mathbf{w}^\top \mathbf{x}_j), \tag{1}$$

where ℓ is a suitable loss function such as $\ell(t) = \max(0, 1 - t)^2$.

Currently the only publicly available software for solving RankSVM is SVMLight (Joachims, 2002).[1] This software explicitly forms all difference vectors $\mathbf{x}_i - \mathbf{x}_j$ corresponding to $(i, j) \in \mathcal{P}$ to set up a standard classification problem and solves for $\mathbf{w}$ using a dual method. This method is very slow. Any SVM solver for classification can in fact be used to train RankSVM: what is needed is simply to construct a training set made of the difference vectors $\mathbf{x}_i - \mathbf{x}_j$.[2] But because the resulting training set can be large, this straightforward adaptation of an SVM classifier can also be very slow.

Most works (MSR, 2008) that determine baseline performance values for RankSVM use SVMLight and, in order to keep computational time within manageable limits they specify a limit on the number of iterations in SVMLight. This is unfortunate because this leads to incomplete training and hence poor ranking results being assigned to RankSVM. As we will see later in the paper, with full training RankSVM comes out to be a top performer; for example, on the Ohsumed dataset it is the best performing published method!

This motivates us to look for fast methods of training RankSVM. We consider the primal Newton method (Keerthi and DeCoste, 2005; Chapelle, 2007b) that is known to be fast for SVM classifier training; section 2 reviews this method. Efficient adaptation of this method for optimizing (1) requires care. In sections 3 and 4 we give two different ways of doing this. Efficiency issues mainly arise due to the large value of p, the size of $\mathcal{P}$. In the worst case $p = O(n^2)$. Our first method gives careful consideration to doing the various computations of the Newton method efficiently; in particular, any explicit formation of $\mathbf{x}_i - \mathbf{x}_j$ vectors is avoided. The complexity of this method is $O(nd + p)$. We have made code for that method publicly available. The second method exploits the fact that $\mathcal{P}$ arises due to a small number of relevance levels and yields a method that has complexity $O(nd + n \log n)$. Evaluation of efficiency and generalization performance of the new methods (section 5) show them to be impressive. There are various ways in which the methods can be further extended; this is discussed in section 6.

Finally, note that after this paper has been submitted, T. Joachims released code[3] for training RankSVM based on the structured output learning framework (Tsochan-

[1] SVMLight uses the hinge loss $\ell(t) = \max(0, 1 - t)$. Though it is different from the squared hinge loss that we use, this causes little difference in performance; see section 5.

[2] This is only feasible in the linear case.

[3] Available at: `http://www.cs.cornell.edu/People/tj/svm_light/svm_rank.html`

taridis et al, 2005). This code is much faster than SVMLight and the training times appear to be comparable to the ones obtained using our methods.

2 Newton optimization for linear SVM

We present in this section the primal training algorithm of a linear SVM classifier along the lines of (Keerthi and DeCoste, 2005; Chapelle, 2007b). Given a training set of n examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$, the aim is to find the solution of the following optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b))^2. \tag{2}$$

As argued in (Keerthi and DeCoste, 2005; Chapelle, 2007b), there is no need to solve this problem in the dual – which is a common practice for SVM solvers – and it can be solved more efficiently in the primal. Since (2) is an unconstrained and differentiable objective function, the two following standard optimization methods can be used:

Non-linear conjugate gradient This is one of the most popular gradient based methods (Shewchuk, 1994). It only requires the gradient of the objective function and a line search procedure. This one dimensional line search can be performed using standard techniques such as the secant method or a cubic polynomial interpolation. In case of SVMs this line search can be done in $O(n)$ time if the matrix vector multiplication $X\mathbf{s}$ is precomputed, $\mathbf{s}$ being the search direction and X the data matrix,

$$X := [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top.$$

Truncated Newton In Newton optimization, the weight vector is updated at each iteration as $\mathbf{w} \leftarrow \mathbf{w} - H^{-1}\mathbf{g}$, where H and $\mathbf{g}$ are respectively the Hessian and gradient of the objective function. In our case, computing the Hessian takes $O(nd^2)$ time and solving the linear system $O(d^3)$. If d is large, this can be prohibitive. In this case Keerthi and DeCoste (2005) proposed a truncated Newton method (Dembo and Steihaug, 1983) in which the Hessian is not computed explicitly and the linear system is solved by linear conjugate gradient.

Both methods are roughly equivalent (Chapelle, 2007a) and even though we could have chosen either of them as a basis for the RankSVM training, we decided to use the truncated Newton method. Details for this method are as follows.

Let sv be the set of "support vectors", $\mathsf{sv} = \{i,\ y_i(\mathbf{w}^\top \mathbf{x}_i + b) < 1\}$. The gradient of the objective (2) is,

$$\mathbf{g} := \mathbf{w} + 2C\sum_{i\in\mathsf{sv}} (\mathbf{w}^\top \mathbf{x}_i + b - y_i)\mathbf{x}_i, \tag{3}$$

and the Hessian is

$$H := I + 2C\sum_{i\in\mathsf{sv}} \mathbf{x}_i\mathbf{x}_i^\top,$$

with I being the identity matrix.

The Newton step $H^{-1}\mathbf{g}$ can be approximatively computed with *linear conjugate gradient.* Key to the algorithm is that the Hessian is never computed explicitly. Indeed the linear conjugate gradient method only requires to be able to compute the matrix vector multiplication $H\mathbf{s}$ for some vector $\mathbf{s}$. And this can be computed as:

$$H\mathbf{s} = \mathbf{s} + 2CX^\top D(X\mathbf{s}),$$

where D is a diagonal matrix with $D_{ii} = 1$ if $i \in \mathsf{sv}$, 0 otherwise.

Algorithm 1 Truncated Newton training of linear SVM. The Newton step is computed with linear conjugate gradient

```
repeat                                                        ▷ Newton Loop
    $D_{ii} = \mathbb{1}(y_i(\mathbf{w}^\top \mathbf{x}_i) < 1)$.
    $\mathbf{g} = \mathbf{w} + 2CX^\top D(X^\top \mathbf{w} - \mathbf{y})$.                       ▷ Gradient (3)
    repeat                    ▷ Solve by linear CG: $\delta\mathbf{w} = (I + 2CX^\top DX)^{-1}\mathbf{g}$.
        Update based on the computation of $\mathbf{s} + 2CX^\top D(X\mathbf{s})$ for some $\mathbf{s}$.
    until Convergence (of linear conjugate gradient)
    $\mathbf{w} \leftarrow \mathbf{w} - t\delta\mathbf{w}$ ($t$ found by line search).     ▷ Optional. In practice we take $t = 1$.
until Convergence (of Newton)
```

The bulk of the computation is due to multiplications by X which take $O(nd)$ time. Assuming that the number of Newton steps and linear conjugate gradient iterations are constant (which seems to be the case in practice), the total complexity is also $O(nd)$. Note that if the matrix X is sparse (in text classification for instance), this complexity reduces to the number of non-zero elements of X.

A sketch of the truncated Newton method for solving (2) is given in Algorithm 1, where we ignored the bias term b for simplicity. We purposely do not give any details about the steps involved in the linear conjugate gradient algorithm as it is a well documented method (Barrett and Romine, 1994). There are several possible variations around it, but all of them rely on Hessian vector multiplications. In our implementation we used the `minres` function from Matlab.

3 A first implementation for RankSVM

As mentioned in the introduction, any software for linear SVM classification can be used to train RankSVM. The training samples $\mathbf{x}_i$ simply need to be replaced by the differences $\mathbf{x}_i - \mathbf{x}_j$ for $(i, j) \in \mathcal{P}$. In matrix form, this means replacing the matrix X by AX where A is a $p \times n$ sparse matrix, $p = |\mathcal{P}|$. Each row of A encodes a preference: if $(i, j) \in \mathcal{P}$, there exists a row k of A such that $A_{ki} = 1, A_{kj} = -1$ and the rest of the row is 0.

Since the size of the new training matrix AX is $p \times d$, applying naively the truncated Newton method presented in the previous section would yield an $O(pd)$ complexity. However, it is possible to do better because the AX matrix does not need to be computed explicitly. Indeed, because of the linear conjugate gradient algorithm, the Hessian vector multiplication can be written as,

$$H\mathbf{s} = \mathbf{s} + 2CX^\top A^\top DAX\mathbf{s},$$

where now I and D are $p \times p$ matrices. By multiplying from the right, this product can be computed in $O(nd + p)$ time due to the sparseness of A.

We call this adaptation of Algorithm 1 to the RankSVM formulation *PRSVM*, which stands for Primal RankSVM. Matlab code for this algorithm is available at `http://olivier.chapelle.cc/primal/`.

4 A better algorithm

As indicated earlier, computing gradient and Hessian times vector form the crucial time-consuming operations of the Newton training algorithm. The objective function has the form $\frac{1}{2}\|\mathbf{w}\|^2 + CL$ where L is the accumulation of all loss terms. Let $\mathbf{g}_L$ and H_L denote the gradient and Hessian of L. Since $\mathbf{g} = \mathbf{w} + C\mathbf{g}_L$ and $H\mathbf{s} = \mathbf{s} + CH_L\mathbf{s}$, we focus on finding efficient ways of computing $\mathbf{g}_L$ and $H_L\mathbf{s}$. In subsection 4.1 we take up the case of a single query and two relevance levels, and, in the subsequent section explain how the ideas extend to the general case.

4.1 Case of single query and two relevance levels

For this case the given n examples can be partitioned into two sets that define a binary classification problem: $A = \{i : \mathbf{x}_i$ is in the higher relevance class$\}$ and $B = \{i : \mathbf{x}_i$ is in the lower relevance class$\}$. Let $\mathbf{y} = X\mathbf{w}$. Using A and B we can rewrite the loss as

$$L = \sum_{i \in A, j \in B} \ell(y_i - y_j), \tag{4}$$

where $\ell(t) = \max(0, 1-t)^2$. It is useful to define

$$\tilde{y}_i = \begin{cases} y_i - \frac{1}{2} \text{ if } i \in A, \\ y_i + \frac{1}{2} \text{ if } i \in B, \end{cases}$$

so that $\ell(y_i - y_j)$ can be rewritten more simply as $\max(0, \tilde{y}_j - \tilde{y}_i)^2$. Let

$$B_i = \{j \in B : \tilde{y}_j > \tilde{y}_i\},\ i \in A; \quad A_j = \{i \in A : \tilde{y}_j > \tilde{y}_i\},\ j \in B.$$

Writing $(\tilde{y}_j - \tilde{y}_i)^2$ as $\tilde{y}_j^2 - \tilde{y}_j\tilde{y}_i - \tilde{y}_i\tilde{y}_j + \tilde{y}_i^2$, summing the first two terms by the order $j,\ i$, reversing the order of summation for the other two terms and simplifying we get

$$L = \sum_{k=1}^{n} \alpha_k \tilde{y}_k^2 - \beta_k \tilde{y}_k,$$

where

$$\alpha_i = |B_i|,\ i \in A; \quad \alpha_j = |A_j|,\ j \in B; \quad \beta_i = \sum_{j \in B_i} \tilde{y}_j,\ i \in A; \quad \beta_j = \sum_{i \in A_j} \tilde{y}_i,\ j \in B,$$

and $|S|$ denotes the cardinality of a set S. We can also get the gradient $\mathbf{g}_L$ using:

$$\mathbf{g}_L := \frac{\partial L}{\partial \mathbf{w}} = X^\top \frac{\partial L}{\partial \mathbf{y}}; \quad \text{and} \quad \frac{\partial L}{\partial y_k} = 2(\alpha_k \tilde{y}_k - \beta_k),\ \forall k = 1, \ldots, n. \tag{5}$$

In the above expression the "2" factor applies to the β_k term so as to also take care of the dependance of β_k on $\tilde{y}_k$.

Computation of $\mathbf{y} = X\mathbf{w}$ needs $O(nd)$ time. It is then easy to get the α and β vectors in $O(n \log n)$ time by sorting $\tilde{y}$ and doing operations using it. The $X^\top \partial L/\partial \mathbf{y}$ computation requires $O(nd)$ time. Thus the time complexity of $\mathbf{g}_L$ is $O(n \log n + nd)$.

Now consider doing the Hessian vector multiplication needed to find the Newton step through linear conjugate gradient: we want to compute $H_L\mathbf{s}$ for a given vector $\mathbf{s}$. $H_L\mathbf{s}$ is given by

$$H_L\mathbf{s} = X^\top \frac{\partial}{\partial t}\left(\frac{\partial L}{\partial y}(\mathbf{w} + t\mathbf{s})\right)\bigg|_{t=0} \tag{6}$$

Let $\delta = X\mathbf{s}$. If we define γ_k using:

$$\gamma_i = \sum_{j \in B_i} \delta_j, \; i \in A; \quad \text{and} \quad \gamma_j = \sum_{i \in A_j} \delta_i, \; j \in B,$$

then, using (6) and (5) it is easy to derive $H_L\mathbf{s}$ as

$$H_L\mathbf{s} = 2X^\top \mathbf{z} \;\; \text{where } z_k = (\alpha_k \delta_k - \gamma_k), \; \forall k = 1, \ldots, n.$$

In each major iteration of Newton training (see Algorithm 1), a $\mathbf{w}$ is given, the gradient is computed at that $\mathbf{w}$, and then $H\mathbf{s}$ is computed for several $\mathbf{s}$ vectors in the inner loop (linear CG operating at the given $\mathbf{w}$). In such an inner loop the sorting operation (on $\tilde{y}$) needs to be done only once when $\mathbf{g}_L$ is computed. After that, each $H\mathbf{s}$ computation requires $O(nd)$ time.

If we compare with the first implementation of RankSVM given in section 3, we can see that, in the worst case the number of preference relations p is $O(n^2)$ (e.g., when the number of elements of A and B are nearly equal), in which case, the algorithm outlined in this section can give significant speed improvements. As we will see in section 5, even in relatively normal situations the speed-up obtained is quite decent.

The ideas of this section extend to other margin-based loss functions. Consider first the hinge loss:

$$\ell_1(t) = \max(0, 1-t),$$

For $\ell = \ell_1$ in (4) it is easy to get the following loss and sub-gradient expressions:

$$L = -\sum_{k=1}^{n} \rho_k \alpha_k \tilde{y}_k, \quad \frac{\partial L}{\partial y_k} = -\rho_k \alpha_k \;\; \forall k = 1, \ldots, n.$$

where $\rho_k = 1$ if $k \in A$ and $\rho_k = -1$ if $k \in B$. This idea for ℓ_1 is not new and is exactly the same as in (Joachims, 2006). Since L is non-differentiable for $\ell = \ell_1$, Newton method cannot be used and so a bundle method is employed in (Joachims, 2006).

Next consider Huber loss:

$$\ell_{\text{Huber}}(t) = \begin{cases} 0 & \text{if } t \geq 1, \\ (1-t)^2 & \text{if } 0 \leq t < 1, \\ 1 - 2t & \text{if } t < 0. \end{cases}$$

It is easy to see that

$$l_{\text{Huber}}(t) = \max(0, 1-t)^2 - \max(0, -t)^2.$$

The first term is the squared hinge loss in (4). The second term is nothing but a shifted squared hinge loss and can be taken care of using ideas similar to those used for squared hinge loss. Thus, for Huber loss also it is easy to give fast methods for computing gradient and doing Hessian times vector operations.

More generally, we can also compute efficiently the objective function and its gradient for any piecewise quadratic function. The complexity scales linearly with the number of pieces. An application of this result is the logistic loss, $\ell(t) = \log(1 + \exp(-t))$ which has been shown to be well approximated by a piecewise quadratic function (Keerthi and Shevade, 2007, figure 1). In this way, we have an efficient algorithm for computing the gradient of the RankNet objective function (Burges et al, 2005).

4.2 Extension to multiple levels/queries

The previous subsection considered the special case of one query with two levels of relevance. More generally, we may have Q queries and each query q may have a set of documents associated with it. Let us denote by $D(q)$ the set of indices of these documents: $D(q) := \{i : \ \mathbf{x}_i \text{ is associated with } q\}$. By definition $\{D(q)\}$ forms a partition of $\{1, \dots, n\}$.

Instead of binary relevance labels, documents may have relevance labels r_i from an ordinal set that we can represent without loss of generality as $\{1, \dots, R\}$, where 1 corresponds to the most irrelevant label and R is perfect relevance.

As mentioned in the introduction, the preference pairs are extracted within each query. We can thus rewrite the set of preference pairs $\mathcal{P}$ as the following disjoint union:

$$\mathcal{P} = \bigcup_{q,r} \mathcal{P}_{qr} \quad \text{with} \quad \mathcal{P}_{qr} := \{(i,j) \text{ such that } i \in D(q), j \in D(q), r_i = r, r_j < r\}.$$

We can then rewrite the objective function (1) as:

$$\frac{1}{2}||\mathbf{w}||^2 + C \sum_{q=1}^{Q} \sum_{r=1}^{R} \sum_{(i,j)\in\mathcal{P}_{qr}} \ell(\mathbf{w}^\top \mathbf{x}_i - \mathbf{w}^\top \mathbf{x}_j). \tag{7}$$

The gradient associated with the inner-most sum can be computed in $O(n_{qr} \log(n_{qr}) + n_{qr} d)$ time with $n_{qr} = |\mathcal{P}_{qr}|$ using the technique described in section 4.1 where the higher class is $A = \{i, \ i \in D(q), r_i = r\}$ and the lower one is $B = \{i : \ i \in D(q), r_i < r\}$.

Since $\sum_{q,r} n_{qr} = n$, we have $\sum_{q,r} n_{qr} \log(n_{qr}) \leq n \log(n)$ and the overall complexity of gradient computation is: $O(n \log n + nd)$. In a similar way, the Hessian times vector operation can be done in $O(nd)$ time. We denote this method by *PRSVM+* since it improves on the method presented in section 3.

5 Experiments

In this section we report results of ranking and binary AUC optimization experiments involving our methods and SVMLight. The experiments are designed to demonstrate gains in training efficiency and also point out the importance of such gains.

Table 1 Statistics associated with two of the datasets from the Letor 3.0 benchmark.

	Queries	Rel. levels	Features	Av. doc. per query	% Rel. doc.
Ohsumed	106	3	45	152	30%
TD 2004	75	2	64	989	1.5%

5.1 Ranking experiments

We experiment with the Ohsumed and TD 2004 datasets from the Letor 3.0 distribution[4] (Liu et al, 2007). Some statistics associated with these datasets are given in Table 1. We used the same 5 splits in training/validation/test as the ones provided in the Letor distribution. In the results reported below, training time refers to the total training time necessary to train on the 5 splits. The performance measure of interest is the average NDCG@10 over the 5 splits.

The reason we considered the Ohsumed and TD 2004 datasets is that they are the datasets with the largest number of preference pairs in the Letor collection. This makes the problem more interesting. On each fold, each training set generates, on average, 350,000 preference pairs for the Ohsumed dataset and 648,000 pairs for TD 2004. The two datasets are quite different in their statistics: there are three levels of relevance for Ohsumed (irrelevant, partially relevant and highly relevant) and only two levels for TD 2004. Also TD 2004 has a large number of documents per query (nearly 1000 on average), but only a few of them are relevant, whereas Ohsumed has fewer documents, but 30% of them are relevant (either partially or highly).

As far as we know, SVMLight is the only publicly available software for training RankSVM. We thus compared the following algorithms:

`PRSVM` This is the algorithm described in section 3 based on SVM primal training; it scales linearly with the number of pairs, which in the worst case is quadratic in the number of examples. Code for that method is available at `http://olivier.chapelle.cc/primal/`.

`PRSVM+` This is the improved version of the above algorithm, as described in section 4. This scales as $O(n \log n)$.

`SVMLight` The software downloaded from `http://svmlight.joachims.org/`. We ran it in ranking mode with the `-z p` option.

`SVMLight#` Same as SVMLight, but with the `-# 5000` option, which limits the number of iterations to 5000. This is how the benchmark results on the Letor website were obtained (MSR, 2008).

We begin with a study of the influence of the parameter C on the training time as well as the generalization performance. Results are shown in Figure 1. As with most SVM software optimizers, training time increases with C for PRSVM and PRSVM+. We did not run any experiments with SVMLight in this setting because of the prohibitive training time (see below). The scaling with C in our primal methods is better than with the dual based SVMLight method; also see (Keerthi and DeCoste, 2005, Table 5). As for the generalization performance, the best C is 10^{-4} on Ohsumed and 10^{-2} on TD2004.

We now compare training times and generalization performances under two settings for the choice of C:

[4] available at `http://research.microsoft.com/en-us/um/beijing/projects/letor/`

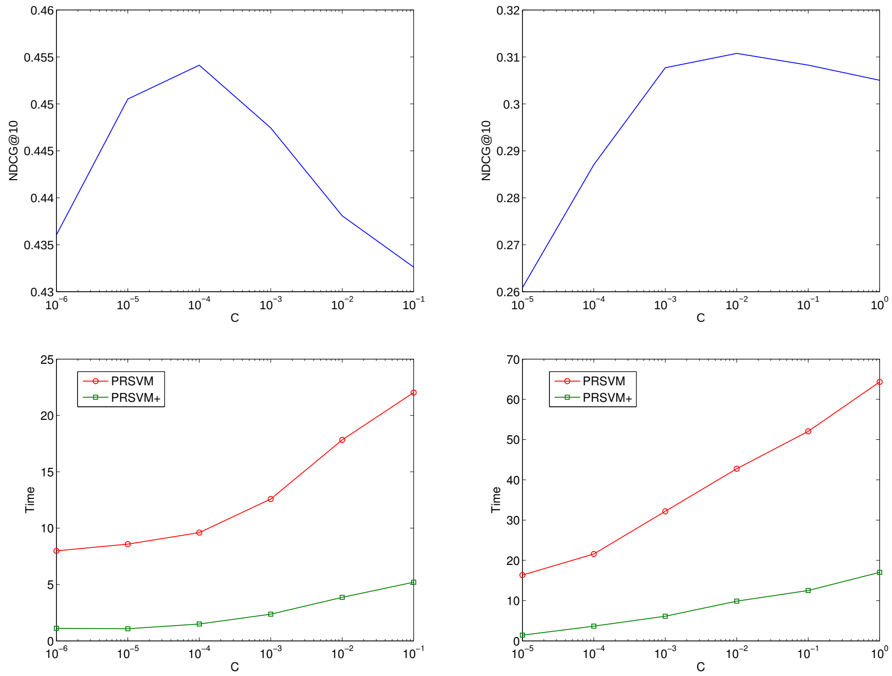

Fig. 1 Influence of C; left = Ohsumed; right = TD2004. Times are in seconds.
.

Table 2 Values of C found by model selection in (MSR, 2008).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Ohsumed	0.01	0.001	0.0001	0.6	0.1
TD2004	0.05	0.01	4	0.03	0.9

Table 3 Training time (in seconds for all folds) and NDCG@10 on the test set for 4 different RankSVM optimization methods. C is set to 10^{-4} for Ohsumed and 10^{-2} for TD2004.

	PRSVM	PRSVM+	SVMLight	SVMLight#	
NDCG@10	0.454	0.454	0.454	0.384	Ohsumed
Time	9.6	1.5	81097	5235	
NDCG@10	0.311	0.311	0.308	0.256	TD2004
Time	42.8	9.9	115675	44740	

1. Fix C for all the folds to the optimal values found in Figure 1. This is not a proper way of doing model selection, but that enables us to focus on training efficiency (which is the main point of this paper) without having to worry about model selection.
2. For each fold, take the C value as selected in (MSR, 2008). For reference, these C values are reported in Table 2.

Results for these two settings are reported in Tables 3 and 4 respectively. The following conclusions can be drawn from this experiment. First, not surprisingly, the

Table 4 Similar to Table 3, but with C selected as in (MSR, 2008); see Table 2. Note that the NDCG@10 reported in (MSR, 2008) for TD2004 is 0.308 and not 0.299: it is possible that the experiments on that dataset were not run using the -# 5000 option.

	PRSVM	PRSVM+	SVMLight	SVMLight#	
NDCG@10	0.452	0.452	0.442	0.414	Ohsumed
Time	15.3	3.4	55750	7078	
NDCG@10	0.309	0.309	0.308	0.299	TD2004
Time	63.1	16.3	60571	34687	

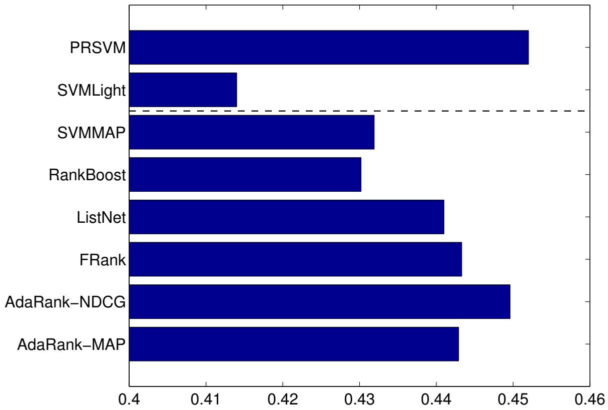

Fig. 2 NDCG@10 on the Ohsumed dataset from the Letor distribution. The first two rows are RankSVM implementations: the first one is ours and the second one is from the Letor website, SVMLight with the -# 5000 option. The other rows are competing "learning to rank" algorithms. Because of the limited number of queries (106), a difference of 0.1 is typically not statistically significant.

NDCG@10 is almost the same for SVMLight and PRSVM/PRSVM+: the only difference between these two methods is the linear vs quadratic penalization of the errors; in most real world problems, both of these penalizations give similar performance results. Second, limiting the number of iterations for SVMLight as done in (MSR, 2008) (the SVMLight# entry) can really hurt performance. This is particularly striking from Figure 2. RankSVM appears to be one of the worst algorithms reported on the Letor website, but with our implementation it becomes the best algorithm. This is unfortunate because previously people have been led to conclude that RankSVM is an inferior algorithm. Third, PRSVM/PRSVM+ are several orders of magnitude faster than SVMLight. This is also expected because SVMLight is not suited for large scale linear learning. An adaptation of SVMPerf (Joachims, 2005, 2006) to ranking would probably be much better than SVMLight. But as noted earlier, we compared to SVMLight because it is the only publicly available software to solve RankSVM. And finally, PRSVM+ is faster than PRSVM. Still, on these problems, the number of preference pairs is relatively small (less than a million) and PRSVM, for which code is publicly available, can be used without any problem.

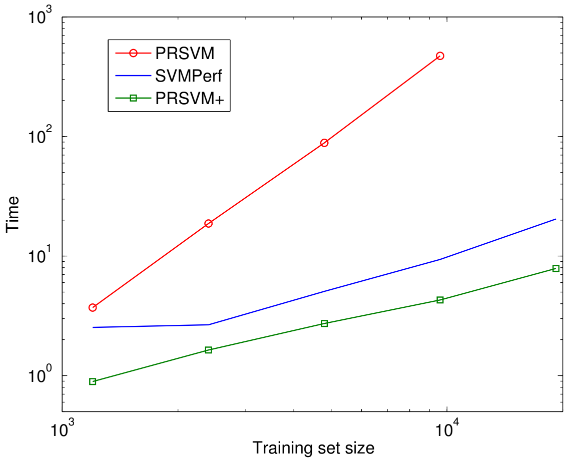

Fig. 3 Training time as a function of n for AUC optimization on the CCAT dataset.

5.2 Optimization of the AUC

We present an experiment on a binary classification problem for optimizing the area under the ROC curve (AUC). The AUC is known to be equal to the proportion of correctly classified pairs (Joachims, 2005):

$$\text{AUC} = \frac{|\{(i,j) \text{ such that } y_i = 1, y_j = -1, f(\mathbf{x}_i) > f(\mathbf{x}_j)\}|}{|\{i,\ y_i = 1\}| \times |\{i,\ y_i = -1\}|}.$$

Optimizing this quantity is thus a special case of RankSVM with one query and binary relevance. This is also the special case discussed in subsection 4.1.

The classification problem we consider here is the problem of separating documents in the `CCAT` category from the remaining documents in the RCV1 text collection (Lewis et al, 2004). We fixed C to 0.005 so as to obtain good performance on the test set using the entire training set.

Our main aim in this experiment is to illustrate that PRSVM+ has a better time complexity than PRSVM. For this purpose, we trained both methods with various training set sizes, $n \in \{1200, 2400, 4800, 9600, 19200\}$. Results are presented in Figure 3. For $n = 19200$, we could not train PRSVM because the number of pairs (around 10^8) caused memory issues. The $O(n^2)$ vs $O(n \log n)$ complexity difference appears clearly in Figure 3. In that figure, we also included the training time of SVMPerf (Joachims, 2005). It is also an $O(n \log n)$ method for optimizing the AUC, so it is not surprising to see that both methods are very near in terms of time complexity. This method was run with the `-w 3` option for the optimization method (that turned out to be the fastest one) and ϵ (the stopping criterion) set to 0.001. We chose this value of ϵ because it resulted in a relative duality gap of about 10^{-5} which is of the same order of magnitude than the relative stopping criterion that we used: Newton decrement / objective function value $< 10^{-6}$.

The current implementation of SVMPerf (Joachims, 2005) only allows AUC optimization, i.e., it only solves the one query and binary relevance case of the ranking problem. If this code were to be extended to deal with the general ranking case, it is likely that it would achieve training times that are comparable to the ones reported for PRSVM+ in section 5.1.

6 Discussion/extension

The algorithms described in this paper can be extended and improved in several ways. We discuss some of them below.

6.1 Online optimization

For very large datasets, it has been argued that online learning algorithms such as stochastic gradient descent converge faster to a good solution than batch algorithms (Bottou and Bousquet, 2008). The Pegasos algorithm (Shalev-Shwartz et al, 2007) for instance is the result of applying stochastic gradient descent to SVMs for classification.

As mentioned in the introduction, we can apply any SVM solver algorithm for solving the RankSVM. Applying stochastic gradient – which is essentially what Pegasos is doing – to RankSVM would yield Algorithm 2. Defining $\ell_{ij}(\mathbf{w}) := \frac{1}{2|\mathcal{P}|}||\mathbf{w}||^2 + C\ell(\mathbf{w}^\top \mathbf{x}_i - \mathbf{w}^\top \mathbf{x}_j)$, it is clear that the objective function (1) is $\sum_{(i,j)\in\mathcal{P}} \ell_{ij}(\mathbf{w})$ and the update in Algorithm 2 is simply the stochastic gradient $\partial\ell_{ij}/\partial\mathbf{w}$ multiplied by the standard $1/t$ learning rate.

Algorithm 2 Stochastic gradient algorithm on the pairs to solve RankSVM

```
$\mathbf{w} \leftarrow 0$
for $t = 1, \ldots, T$ do
    Pick at random a preference pair $(i, j)$ from $\mathcal{P}$.
    $\mathbf{w} \leftarrow \mathbf{w} - \frac{1}{t}\left(\frac{1}{|\mathcal{P}|}\mathbf{w} + C\ell'(\mathbf{w}^\top(\mathbf{x}_i - \mathbf{x}_j))(\mathbf{x}_i - \mathbf{x}_j)\right)$.
end for
```

A potential drawback of such an approach is that it still scales linearly with $|\mathcal{P}|$ (or quadratically with n) and does not leverage the $O(n \log n)$ trick developed in section 4. In order to get the best of both worlds, we can do the stochastic gradient descent at the query level instead of the pair level. This means that we randomly select a query, compute the gradient of the part of the objective function associated with that query – that can be done in $O(n \log n)$ time – and take a step in that direction.

Coming back to the notations used in equation (7), let us define the loss associated with query q as

$$\ell_q(\mathbf{w}) := \frac{1}{2Q}||\mathbf{w}||^2 + C \sum_{r=1}^{R} \sum_{(i,j)\in\mathcal{P}_{qr}} \ell(\mathbf{w}^\top \mathbf{x}_i - \mathbf{w}^\top \mathbf{x}_j). \tag{8}$$

The overall objective function can thus be written as $\sum_q \ell_q(\mathbf{w})$ and the associated stochastic gradient algorithm is summarized in Algorithm 3.

Algorithm 3 Stochastic gradient algorithm on the query to solve RankSVM

```
$\mathbf{w} \leftarrow 0$
for $t = 1, \ldots, T$ do
    Pick at random a query $q$.
    Compute $\mathbf{g}_t := \partial \ell_q / \partial \mathbf{w}$ using the technique of section 4, where $\ell_q$ is defined by (8).
    $\mathbf{w} \leftarrow \mathbf{w} - \frac{1}{t}\mathbf{g}_t$.
end for
```

6.2 Different margin/weights

One could be interested in giving a different cost δ_{ij} to each misclassified pair $(i, j) \in \mathcal{P}$: indeed an inversion at the top of the ranking is more costly than one at the bottom; also an inversion between two documents having several levels of difference in their labels should be more heavily penalized than one involving two adjacent labels.

Based on this intuition, it was for instance suggested by Burges et al (2007) to set δ_{ij} as the difference in DCG[5] obtained by permuting documents i and j,

$$\delta_{ij} = (2^{r_i} - 2^{r_j}) \left| \frac{1}{\log_2(p_i)} - \frac{1}{\log_2(p_j)} \right|,$$

where p_i and p_j are the positions of documents i and j in the current ranking. The problem is that these weights depend on the current ranking. Instead, if p_i and p_j are the positions in the optimal ranking, δ_{ij} is fixed. It has been shown (Grinspan, 2007) that with that choice of δ_{ij}, the following sum is an upper bound on the difference between the optimal DCG and the DCG achieved with the current ranking:

$$\sum_{(i,j)\in\mathcal{P}} \delta_{ij} \mathbb{1}(\mathbf{w}^\top \mathbf{x}_i < \mathbf{w}^\top \mathbf{x}_j), \tag{9}$$

where $\mathbb{1}$ is the indicator function.

There are now two ways of introducing δ_{ij} in the pairwise loss function $\ell = \max(0, 1 - \mathbf{w}^\top(\mathbf{x}_i - \mathbf{x}_j))^2$:

Margin rescaling Set the margin to $\sqrt{\delta_{ij}}$, i.e., take $\ell = \max(0, \sqrt{\delta_{ij}} - \mathbf{w}^\top(\mathbf{x}_i - \mathbf{x}_j))^2$.
Slack rescaling Multiply the loss by δ_{ij}, i.e., take $\ell = \delta_{ij} \max(0, 1 - \mathbf{w}^\top(\mathbf{x}_i - \mathbf{x}_j))^2$.

It is easy to verify that both methods yield an upper bound on $\delta_{ij}\mathbb{1}(\mathbf{w}^\top \mathbf{x}_i < \mathbf{w}^\top \mathbf{x}_j)$. With these modifications, the empirical loss part of (1) is an upper bound on (9). The terminology margin/slack rescaling comes from (Tsochantaridis et al, 2005). Weighting of the RankSVM objective function has also been studied in (Cao et al, 2006).

For these reasons, it is worth studying optimization methods capable of handling "rescaled" versions of (1). The first method we described in section 3 can be adapted without any difficulty to this new setting: one just has to make minor changes to the procedures used for computing the objective function, its gradient and the Hessian vector multiplication.

But it is not possible to do so for general δ_{ij} for the method described in section 4. Two special cases where the gradient can still be computed in $O(n \log n)$ time are: 1- margin rescaling and δ_{ij} is of the form $(\delta_i - \delta_j)^2$; 2- δ_{ij} only depends on r_i and r_j; note that in this case the ideas of subsection 4.2 need to be slightly modified to separately deal with the loss term corresponding to each pair of relevance levels.

[5] In fact Burges et al (2007) are interested in NDCG and there is an additional normalization factor to add in the formula for δ_{ij}.

6.3 Non-linear kernel

We have proposed in this paper a primal based optimization of linear RankSVM. More generally, one might want to obtain a non-linear function through the use of a *kernel* function k. The so-called kernel trick (Aizerman et al, 1964; Schölkopf and Smola, 2002) consists in mapping the training points $\mathbf{x}_i$ to an implicit high dimensional feature space $\mathcal{H}$, $\mathbf{x} \mapsto \phi(\mathbf{x}) \in \mathcal{H}$, where ϕ is such that $\forall \mathbf{x}, \mathbf{x}', k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$.

There is a certain misconception that kernel methods can only work in the dual. But as explained in (Chapelle, 2007b), one can also train a non-linear SVM in primal. The same holds true for RankSVM. The key observation comes from the representer theorem (Kimeldorf and Wahba, 1970): since the objective function (1) only depends on the regularizer and the function evaluation at the training samples $\mathbf{x}_i$, this theorem states that the function minimizing (1) in $\mathcal{H}$ can be written as:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \beta_i k(\mathbf{x}, \mathbf{x}_i). \tag{10}$$

This result can also easily be derived by differentiating (1) and noticing that the optimal $\mathbf{w}$ is a linear combination of the training samples.

From the expansion (10) it appears that to train a non-linear version of RankSVM one can use standard optimization techniques over the β_i. Since there are n variables a Newton method would for instance take $O(n^3)$ operations to converge. This cubic scaling is typical for kernel methods unless approximate solutions are sought.

In contrast, if one uses a dual method such as SVMLight, the number of variables to optimize is $|\mathcal{P}|$ because there is a Lagrange multiplier for each constraint associated with a preference pair. Assuming that the training time is also cubic in the number of variables and that $|\mathcal{P}| = O(n^2)$, the complexity of a dual method is $O(n^6)$. This is the reason why SVMLight is so slow in the experimental evaluation of section 5.

Finally, if one is interested in reusing the code for the linear case, one approach is to perform the optimization directly in feature space. The reason why this is feasible is that even in the case of a high dimensional feature space $\mathcal{H}$, a training set $(\mathbf{x}_1, \dots, \mathbf{x}_n)$ of size n, when mapped to this feature space, spans a vectorial subspace $E \subset \mathcal{H}$ whose dimension is at most n. By choosing a basis for E and expressing the coordinates of all the points in that basis, we can then directly work in E.

Let $(\mathbf{v}_1, \dots, \mathbf{v}_n) \in E^n$ be an orthonormal basis of E,[6] where $\mathbf{v}_p$ is expressed as

$$\mathbf{v}_p = \sum_{i=1}^{n} A_{ip} \phi(\mathbf{x}_i)$$

The orthonormality of the basis thus translates into the requirement $A^\top K A = I$, where K is the kernel matrix with entries $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. The previous equality is equivalent to $K = (AA^\top)^{-1}$. Because of rotations, there are an infinite number of matrices A satisfying this condition. One of them can be found by inverting the Cholesky decomposition of K. Another possibility is to perform the eigendecomposition of K as $K = U\Lambda U^\top$ and set $A = U\Lambda^{-1/2}$. This letter case corresponds to the *kernel PCA map* introduced in (Schölkopf and Smola, 2002, section 14.2).

[6] We suppose that $dim(E) = n$, i.e. that the points are linearly independent in feature space or that K has full rank. The same analysis can be followed when $dim(E) < n$.

Now we can use the following mapping $\psi : \mathcal{X} \mapsto \mathbb{R}^n$:

$$\psi_p(\mathbf{x}) := \phi(\mathbf{x})^\top \mathbf{v}_p = \sum_{i=1}^{n} A_{ip} k(\mathbf{x}, \mathbf{x}_i), \quad 1 \le p \le n.$$

It can be easily checked that $\psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$. Thus applying the linear approach presented in sections 3 and 4 to this representation is equivalent to solving the non-linear problem.

7 Conclusion

In this paper we have given fast algorithms for training RankSVM, demonstrated their usefulness in efficiently solving web page ranking and binary AUC optimization problems, and proposed several extensions. We hope that these are useful in the solution of large scale problems arising in practice. In particular we have made the code for PRSVM available: with this code researchers interested in learning to rank can now train RankSVM in several seconds (on the Ohsumed dataset for instance) instead of almost a day as required by SVMLight.

References

Aizerman M, Braverman E, Rozonoer L (1964) Theoretical foundations of the potential function method in pattern recognition learning. Automation and Remote Control 25:821–837

Barrett R, Romine C (1994) Templates for the solution of linear systems: building blocks for iterative methods. Society for Industrial Mathematics

Bottou L, Bousquet O (2008) The tradeoffs of large scale learning. In: Platt J, Koller D, Singer Y, Roweis S (eds) Advances in Neural Information Processing Systems, 20, pp 161–168

Burges C, Shaked T, Renshaw E, Lazier A, Deeds M, Hamilton N, Hullender G (2005) Learning to rank using gradient descent. In: Proceedings of the International Conference on Machine Learning

Burges CJ, Le QV, Ragno R (2007) Learning to rank with nonsmooth cost functions. In: Schölkopf, Platt J, Hofmann T (eds) Advances in Neural Information Processing Systems 19

Cao Y, Xu J, Liu TY, Li H, Huang Y, Hon HW (2006) Adapting ranking SVM to document retrieval. In: SIGIR

Cao Z, Qin T, Liu TY, Tsai MF, Li H (2007) Learning to rank: from pairwise approach to listwise approach. In: International Conference on Machine Learning

Chapelle O (2007a) Optimization techniques for support vector machines. Talk at the workshop on *Numerical Tools and Fast Algorithms for Massive Data Mining, Search Engines and Applications*, UCLA, `https://www.ipam.ucla.edu/publications/sews2/sews2_7130.pdf`

Chapelle O (2007b) Training a support vector machine in the primal. Neural Computation 19(5):1155–1178

Cossock D, Zhang T (2006) Subset ranking using regression. In: 19th Annual Conference on Learning Theory, Springer, Lecture Notes in Computer Science, vol 4005, pp 605–619

Dembo R, Steihaug T (1983) Truncated-newton algorithms for large-scale unconstrained optimization. Mathematical Programming 26(2):190–212

Freund Y, Iyer R, Schapire RE, Singer Y (2003) An efficient boosting algorithm for combining preferences. Journal of Machine Learning Research 4:933–969

Grinspan P (2007) A connection between DCG and a pairwise-defined loss function, Internal Yahoo! memo

Herbrich R, Graepel T, Obermayer K (2000) Large margin rank boundaries for ordinal regression. In: Smola, Bartlett, Schoelkopf, Schuurmans (eds) Advances in Large Margin Classifiers, MIT Press, Cambridge, MA

Joachims T (2002) Optimizing search engines using clickthrough data. In: Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD), ACM

Joachims T (2005) A support vector method for multivariate performance measures. In: International Conference on Machine Learning (ICML), pp 377–384

Joachims T (2006) Training linear SVMs in linear time. In: ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD), pp 217–226

Keerthi SS, DeCoste DM (2005) A modified finite Newton method for fast solution of large scale linear SVMs. Journal of Machine Learning Research 6:341–361

Keerthi SS, Shevade S (2007) A fast tracking algorithm for generalized LARS/LASSO. IEEE transactions on Neural Networks 18(6):1826–1830

Kimeldorf GS, Wahba G (1970) A correspondence between bayesian estimation on stochastic processes and smoothing by splines. Annals of Mathematical Statistics 41:495–502

Lewis D, Yang Y, Rose T, Li F (2004) Rcv1: A new benchmark collection for text categorization research. Journal of Machine Learning Research 5:361–397

Liu TY, Xu J, Qin T, Xiong W, Li H (2007) Letor: Benchmark dataset for research on learning to rank for information retrieval. In: LR4IR 2007, in conjunction with SIGIR 2007

MSR (2008) Ranking SVM on LETOR. Microsoft Research Asia, `http://research.microsoft.com/en-us/um/beijing/projects/letor/Baselines/RankSVM.html`

Schölkopf B, Smola A (2002) Learning with Kernels. MIT Press, Cambridge, MA

Shalev-Shwartz S, Singer Y, Srebro N (2007) Pegasos: Primal estimated sub-gradient solver for SVM. In: Proceedings of the International Conference on Machine Learning

Shewchuk JR (1994) An introduction to the conjugate gradient method without the agonizing pain. Tech. Rep. CMU-CS-94-125, School of Computer Science, Carnegie Mellon University

Tsochantaridis I, Joachims T, Hofmann T, Altun Y (2005) Large margin methods for structured and interdependent output variables. Journal of Machine Learning Research 6:1453–1484

Zheng Z, Zha H, Zhang T, Chapelle O, Chen K, Sun G (2008) A general boosting method and its application to learning ranking functions for web search. In: Advances in Neural Information Processing Systems 20, MIT Press, pp 1697–1704