Large-scale Kernel RankSVM

Tzu-Ming Kuo* Ching-Pei Lee† Chih-Jen Lin‡

Abstract

Learning to rank is an important task for recommendation systems, online advertisement and web search. Among those learning to rank methods, rankSVM is a widely used model. Both linear and nonlinear (kernel) rankSVM have been extensively studied, but the lengthy training time of kernel rankSVM remains a challenging issue. In this paper, after discussing difficulties of training kernel rankSVM, we propose an efficient method to handle these problems. The idea is to reduce the number of variables from quadratic to linear with respect to the number of training instances, and efficiently evaluate the pairwise losses. Our setting is applicable to a variety of loss functions. Further, general optimization methods can be easily applied to solve the reformulated problem. Implementation issues are also carefully considered. Experiments show that our method is faster than state-of-the-art methods for training kernel rankSVM.

Keywords: Kernel method, Learning to rank, Ranking support vector machines, Large-margin method

1 Introduction

Being heavily applied in recommendation systems, online advertisements and web search in recent years, learning to rank gains more and more importance. Among existing approaches for learning to rank, rankSVM [7] is a commonly used method extended from the popular support vector machine (SVM) [2, 6] for data classification.

In SVM literature, it is known that linear (i.e., data are not mapped to a different space) and kernel SVMs are suitable for different scenarios, where linear SVM is more efficient, but the more costly kernel SVM may give higher accuracy.[1] The same situation may also occur in rankSVM because it can be viewed as a special case of SVM; see more details in Section 2.1. Because of the lower training cost, linear rankSVM has been extensively studied and efficient algorithms have been proposed [11, 23, 5, 1, 14]. However, for some tasks that the feature set is not rich enough, nonlinear methods may be needed. Therefore, it is important to develop efficient training methods for large kernel rankSVM.

Assume we are given a set of training label-query-instance tuples $(y_i, q_i, \boldsymbol{x}_i), y_i \in \mathbf{R}, q_i \in S \subset \mathbf{Z}, \boldsymbol{x}_i \in \mathbf{R}^n,\ i = 1, \ldots, l$, where S is the set of queries. By defining the set of preference pairs as

$$P \equiv \{(i,j) \mid q_i = q_j, y_i > y_j\} \text{ with } p \equiv |P|, \tag{1.1}$$

rankSVM [10] solves

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{\xi}} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum\nolimits_{(i,j)\in P}\xi_{i,j} \\ \text{subject to} \quad & \boldsymbol{w}^T\left(\phi\left(\boldsymbol{x}_i\right) - \phi\left(\boldsymbol{x}_j\right)\right) \geq 1 - \xi_{i,j}, \\ & \xi_{i,j} \geq 0, \forall (i,j) \in P, \end{aligned} \tag{1.2}$$

where $C > 0$ is the regularization parameter and ϕ is a function mapping data to a higher dimensional space. The loss term $\xi_{i,j}$ in (1.2) is called L1 loss. If it is replaced by $\xi_{i,j}^2$, we have L2 loss. The idea behind rankSVM is to learn $\boldsymbol{w}$ such that

$$\boldsymbol{w}^T\phi(\boldsymbol{x}_i) > \boldsymbol{w}^T\phi(\boldsymbol{x}_j), \text{ if } (i,j) \in P.$$

A challenge in training rankSVM is to handle the possibly large number of preference pairs because p can be as large as $O(l^2)$.

In contrast to linear rankSVM that can directly minimize over a finite vector $\boldsymbol{w}$, the difficulty of solving (1.2) is on the high and possible infinite dimensionality of $\boldsymbol{w}$ after data mappings. Existing studies have proposed different methods to solve (1.2) through kernel techniques. The work [7] viewed (1.2) as a special case of SVM, so standard training methods to solve the SVM dual problem can be applied. However, the dual problem of p variables can become very large if $p = O(l^2)$. Joachims [11] reformulated rankSVM as a 1-slack structural SVM problem and considered a cutting-plane method. Although [11] only experiments with linear rankSVM, this approach is applicable to kernel rankSVM. However, cutting-plane methods may suffer from slow converge. The work [26], inspired by a generalized representer theorem [22], represents $\boldsymbol{w}$ as a linear combination of the training instances. It then

*Department of Computer Science, National Taiwan University. b99902073@csie.ntu.edu.tw

†Department of Computer Science, National Taiwan University. r00922098@csie.ntu.edu.tw

‡Department of Computer Science, National Taiwan University. cjlin@csie.ntu.edu.tw

[1] See, for example, [27] for detailed discussion.

reformulates and modifies (1.2) to a linear programming problem. However, the resulting problem is still large because of $O(p)$ variables as well as constraints.

In this paper, a method for efficiently training nonlinear rankSVM is proposed. Our approach solves (1.2) by following [26] to represent $\boldsymbol{w}$ as a linear combination of mapped training instances. In contrast to the linear programming problem in [26], we rigorously obtain an l-variable optimization problem that is equivalent to (1.2). A method for efficiently computing the loss term and its sub-gradient or gradient without going through all the p pairs is then introduced by modifying the order-statistic trees technique from linear rankSVM [1, 14]. Our approach allows the flexible use of various unconstrained optimization methods for training kernel rankSVM efficiently. We present an implementation that is experimentally faster than state-of-the-art methods.

This paper is organized as follows. In Section 2, we detailedly discuss previous studies on kernel rankSVMs. By noticing their shortcomings, an efficient method is then proposed in Section 3. An implementation with comprehensive comparisons to existing works is described in Section 4. Section 5 concludes this paper. A supplementary file including additional analysis and experiments is available at `http://www.csie.ntu.edu.tw/~cjlin/papers/ranksvm/kernel_supplement.pdf`.

2 Existing Methods

We introduce three existing methods for training nonlinear rankSVMs. The first one treats rankSVM as an SVM problem. The next way solves rankSVM under a structural SVM framework. The last approach is inspired by a generalized representer theorem to optimize an alternative linear programming problem.

2.1 SVM Approach Given a set of label-instance pairs $(y_i, \boldsymbol{x}_i), \boldsymbol{x}_i \in \mathbf{R}^n, y_i \in \{1, -1\}, i = 1, \ldots, l$, SVM solves the following optimization problem.

$$\begin{aligned} \min_{\boldsymbol{w}, \bar{\boldsymbol{\xi}}} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum\nolimits_{i=1}^{l} \bar{\xi}_i \\ \text{subject to} \quad & y_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i) \geq 1 - \bar{\xi}_i, \\ & \bar{\xi}_i \geq 0, i = 1, \ldots, l. \end{aligned} \tag{2.3}$$

Clearly, if we define

$$y_{i,j} \equiv 1, \quad \text{and} \quad \phi_{i,j} \equiv \phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_j), \forall (i,j) \in P,$$

then (1.2) is in the form of (2.3) with p instances [7]. One can then apply any SVM solver to solve (1.2).

To handle the high dimensionality of $\phi(\boldsymbol{x}_i)$ and $\boldsymbol{w}$, it is common to solve the dual problem of (2.3) with the help of kernel tricks [6]. To adopt the same technique to rankSVM, the dual problem of (1.2) is as follows.

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T \hat{Q} \boldsymbol{\alpha} - \boldsymbol{e}^T \boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \leq \alpha_{i,j} \leq C, \forall (i,j) \in P, \end{aligned} \tag{2.4}$$

where $\boldsymbol{\alpha} \in \mathbf{R}^p$ is indexed by pairs in P, $\boldsymbol{e} \in \mathbf{R}^p$ is a vector of ones, and

$$\hat{Q}_{(i,j),(u,v)} = \phi_{i,j}^T \phi_{u,v}, \forall (i,j), (u,v) \in P \tag{2.5}$$

is a p by p symmetric matrix. For example, [10] solves (2.4) using the SVM package SVM^{light} [9].

Problem (2.4) is a large quadratic programming problem because the number of variables can be up to $O(l^2)$. From the primal-dual relationship, optimal $\boldsymbol{w}$ and $\boldsymbol{\alpha}$ satisfy

$$\boldsymbol{w} \equiv \sum\nolimits_{(i,j) \in P} \alpha_{i,j} \phi_{i,j}. \tag{2.6}$$

Notice that because $\phi(\boldsymbol{x})$ may be infinite dimensional, (2.5) is calculated by the kernel function $K(\cdot, \cdot)$.

$$\begin{aligned} \hat{Q}_{(i,j),(u,v)} = & K(\boldsymbol{x}_i, \boldsymbol{x}_u) + K(\boldsymbol{x}_j, \boldsymbol{x}_v) \\ & - K(\boldsymbol{x}_i, \boldsymbol{x}_v) - K(\boldsymbol{x}_j, \boldsymbol{x}_u), \text{ where} \\ & K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j). \end{aligned}$$

Although directly computing the matrix $\hat{Q}$ requires $O(l^4)$ kernel evaluations, we can take the following special structure of $\hat{Q}$ to save the cost.

$$\hat{Q} = AQA^T, \tag{2.7}$$

where

$$Q \in \mathbf{R}^{l \times l} \quad \text{with} \quad Q_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j),$$

and $A \in \mathbf{R}^{p \times l}$ is defined as follows.

$$A \equiv \begin{array}{c} \\ \vdots \\ (i,j) \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccccc} \cdots & i & \cdots & & j & \cdots \end{array} \\ \begin{bmatrix} 0 \cdots 0 & +1 & 0 \cdots 0 & -1 & 0 \cdots 0 \end{bmatrix} \end{array}.$$

That is, if $(i,j) \in P$ then the corresponding row in A has that the i-th entry is 1, the j-th entry is -1, and other entries are all zeros. Hence, computing $\hat{Q}$ requires $O(l^2)$ kernel evaluations. However, the difficulty of having $O(l^2)$ variables remains, so solving (2.4) has not been a viable approach for large kernel rankSVM.

2.2 Structural SVM Approach To avoid the difficulty of $O(l^2)$ variables in the dual problem, Joachims showed that (1.2) is equivalent to the following 1-slack structural SVM problem [11].

$$\begin{aligned} \min_{\boldsymbol{w},\xi} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\xi \\ \text{subject to} \quad & \boldsymbol{w}^T \sum_{(i,j)\in P} c_{i,j}\phi_{i,j} \geq \sum_{(i,j)\in P} c_{i,j} - \xi, \\ & c_{i,j} \in \{0,1\}, \forall (i,j) \in P. \end{aligned} \tag{2.8}$$

He then considered a cutting-plane method to solve the dual problem of (2.8). Because of the 2^p constraints in (2.8), the corresponding dual problem contains the same amount of variables. While optimizing (2.8) seems to be more difficult at first glance, it is shown that the optimal dual solution is sparse with a small number of non-zero elements, and this number is independent of p. At each iteration, their cutting-plane method optimizes a sub-problem of the dual problem of (2.8). The sub-problem is consisted of variables in a small working set, which is empty in the beginning of the optimization procedure. After the sub-problem is solved, the variable corresponding to the most violated constraint of (2.8) is added to the working set. This approach thus avoids unnecessary computations involving variables that are zero in optima [12]. An efficient algorithm is also proposed to evaluate ξ and decide which variable is added to the working set at each iteration. Although [11] only discussed the case when kernels are not used, the method can be easily extended to kernel rankSVM. Based on the structural SVM package SVM^{struct} [24, 12], a package SVM^{rank} that can solve both linear and kernel rankSVM by a cutting plane method is released.[2]

However, empirical examinations show that in the linear case, solving rankSVM under the structural SVM framework using a cutting-plane method converges slower than other state-of-the-art methods that directly solve (1.2) [14].

2.3 Reformulation from Representer Theorem To address the difficulty of optimizing the dual problem with $O(l^2)$ variables, [26] considered solving the primal problem. Although the number of variables may be infinite following the data mapping, they applied a generalized representer theorem [22] to have that the optimal $\boldsymbol{w}$ is a linear combination of training data with coefficients $\boldsymbol{\beta}$.[3]

$$\boldsymbol{w} \equiv \sum\nolimits_{i=1}^{l} \beta_i \phi(\boldsymbol{x}_i). \tag{2.9}$$

Therefore, $\boldsymbol{w}^T\phi_{i,j}$ can be computed by

$$\begin{aligned} \boldsymbol{w}^T\phi_{i,j} &= \sum\nolimits_{m=1}^{l} \beta_m \left(K\left(\boldsymbol{x}_i, \boldsymbol{x}_m\right) - K\left(\boldsymbol{x}_j, \boldsymbol{x}_m\right)\right) \\ &= (Q\boldsymbol{\beta})_i - (Q\boldsymbol{\beta})_j. \end{aligned}$$

To reduce the number of nonzero β_i, [26] considered a regularization term $\boldsymbol{e}^T\boldsymbol{\beta}$ with $\boldsymbol{\beta}$ being nonnegative, and obtained the following linear programming problem.

$$\begin{aligned} \min_{\boldsymbol{\beta},\boldsymbol{\xi}} \quad & \boldsymbol{e}^T\boldsymbol{\beta} + C\boldsymbol{e}^T\boldsymbol{\xi} \\ \text{subject to} \quad & (Q\boldsymbol{\beta})_i - (Q\boldsymbol{\beta})_j \geq 1 - \xi_{i,j}, \\ & \xi_{i,j} \geq 0, \forall (i,j) \in P, \\ & \beta_i \geq 0, i = 1, \ldots, l. \end{aligned} \tag{2.10}$$

A package RV-SVM is released,[4] but this approach has the following problems. First, in the representer theorem, the coefficients $\boldsymbol{\beta}$ are unconstrained. To enhance the sparsity, they added the nonnegative constraints on $\boldsymbol{\beta}$. Thus the setting does not coincide with the theorem being used. Second, they have the regularization term $\boldsymbol{e}^T\boldsymbol{\beta}$, which, after using (2.9), is equivalent to neither $\|\boldsymbol{w}\|_1$ (L1 regularization) nor $\boldsymbol{w}^T\boldsymbol{w}/2$ (L2 regularization). Therefore, after solving (2.10), we cannot use (2.9) to obtain the optimal $\boldsymbol{w}$ of the original problem. This situation undermines the interpretability of (2.10). Third, without noticing (2.7), they claimed in [26] that the number of kernel evaluations required by solving the dual problem (2.4) is $O(l^4)$, and theirs only requires $O(l^3)$. However, as we discussed earlier in (2.7) and from (2.10), if Q can be stored, both approaches requires only $O(l^2)$ kernel evaluations. Finally, the number of variables $\boldsymbol{\xi}$ in the linear programming problem can still be as large as $O(l^2)$. Solving a linear programming problem with this amount of variables is expensive, so in the experiments conducted in [26], the data size is less than $1,000$ instances.

3 Methods and Technical Solutions

An advantage of the approach in [26] is that the formulation of $\boldsymbol{w}$ in (2.9) involves only l variables. This is much smaller than $O(l^2)$ in the dual rankSVM (2.4). However, the $O(l^2)$ number of constraints in (2.10) still

[2] `http://www.cs.cornell.edu/People/tj/svm_light/svm_rank.html`. Note that as stated on the website, the method for computing the loss term in this package has a higher complexity then the algorithm proposed in [11].

[3] A similar idea, using the original representer theorem [13] for L2-regularized L2-loss kernel rankSVM, is briefly mentioned in [5]. However, their focus was linear rankSVM.

[4] `https://sites.google.com/site/postechdm/research/implementation/rv-svm`.

causes a large linear programming problem. To derive efficient optimization algorithms, in this section, we rewrite (1.2) as the following form.

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{(i,j)\in P} \max\left(0, 1-\boldsymbol{w}^T\phi_{i,j}\right), \tag{3.11}$$

where the first term is the regularization term, while the second is the loss term. We then incorporate two techniques.

1. Let $\boldsymbol{w}$ involve l variables as in (2.9).
2. Apply efficient techniques to calculate the loss term. In particular, some past developments for linear rankSVM are employed.

3.1 Regularization Term If $\boldsymbol{w}$ is represented by (2.9), then (1.2) can be written as

$$\min_{\boldsymbol{\beta}\in\mathbf{R}^l} \quad \frac{1}{2}\boldsymbol{\beta}^T Q\boldsymbol{\beta} + C\sum_{(i,j)\in P} \max(0, 1-(Q\boldsymbol{\beta})_i + (Q\boldsymbol{\beta})_j), \tag{3.12}$$

and

$$\min_{\boldsymbol{\beta}\in\mathbf{R}^l} \quad \frac{1}{2}\boldsymbol{\beta}^T Q\boldsymbol{\beta} + C\sum_{(i,j)\in P} \max(0, 1-(Q\boldsymbol{\beta})_i + (Q\boldsymbol{\beta})_j)^2, \tag{3.13}$$

respectively for L1 and L2 losses. A difference between the two problems is that (3.13) is differentiable while (3.12) is not. The small number of l variables is superior to the $O(l^2)$ variables in the dual rankSVM problem (2.4). Interestingly, this advantage does not occur for standard SVM. We explain the subtle difference below.

In SVM, the use of formulations like (3.12)-(3.13) has been considered in many places such as [19, 17, 4], where the derivation mainly follows the representer theorem. Take L1 loss as an example. The SVM optimization problem analogous to (3.12) is

$$\min_{\bar{\boldsymbol{\beta}}} \quad \frac{1}{2}\bar{\boldsymbol{\beta}}^T\bar{Q}\bar{\boldsymbol{\beta}} + C\sum_{i=1}^{l} \max\left(0, 1-\left(\bar{Q}\bar{\boldsymbol{\beta}}\right)_i\right), \tag{3.14}$$

where $\bar{Q}_{i,j} = y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and notice that in SVM problems, $y_i \in \{-1, 1\}, \forall i$. If $\bar{\boldsymbol{\alpha}}$ is the variable of dual SVM, then both $\bar{\boldsymbol{\alpha}}$ and $\bar{\boldsymbol{\beta}}$ have l components. However, $\bar{\boldsymbol{\alpha}}$ is nonnegative while $\bar{\boldsymbol{\beta}}$ is unconstrained. It is proved in [4] that if $\bar{Q}$ is positive definite, then the optimum is unique and satisfies

$$\bar{\alpha}_i = y_i\bar{\beta}_i, \forall i.$$

Therefore, (3.14) and SVM dual are strongly related. For SVM, because (3.14) does not possess significant advantages, most existing packages solve the dual problem. The situation for rankSVM is completely different. The work [17] derives (3.14) without using the representer theorem. Instead, they directly consider (3.14) and investigate the connection to the SVM dual problem via optimization theory. Following the same setting, the problem to be considered for rankSVM is

$$\min_{\hat{\boldsymbol{\beta}}\in\mathbf{R}^p} \quad \frac{1}{2}\hat{\boldsymbol{\beta}}^T\hat{Q}\hat{\boldsymbol{\beta}} + C\sum_{(i,j)\in P} \max(0, 1-(\hat{Q}\hat{\boldsymbol{\beta}})_{i,j}). \tag{3.15}$$

In Appendix A, we prove that any optimal $\hat{\boldsymbol{\beta}}$ leads to an optimal

$$\boldsymbol{w} = \sum\nolimits_{(i,j)\in P} \hat{\beta}_{i,j}\phi_{i,j} \tag{3.16}$$

of (1.2). This form is the same as (2.6), but a crucial difference is that $\hat{\boldsymbol{\beta}}$ is unconstrained. Therefore, we can define

$$\boldsymbol{\beta} \equiv A^T\hat{\boldsymbol{\beta}}$$

to simplify (3.15) to an equivalent form in (3.12). This discussion explains why we are able to reduce the number of variables from $O(l^2)$ of $\hat{\boldsymbol{\beta}}$ to l of $\boldsymbol{\beta}$. From Appendix A, any optimal solution of dual rankSVM is also optimal for (3.15). Thus, we can say that (3.15) provides a richer set of possible values to construct the optimal $\boldsymbol{w}$.[5] Therefore, the simplification from $\hat{\boldsymbol{\beta}}$ to $\boldsymbol{\beta}$ becomes possible.

We mentioned that for standard SVM, if $\bar{Q}$ in (3.14) is positive definite, the dual SVM and (3.14) have the same unique solution. For rankSVM, this property requires that $\hat{Q}$ is positive definite. However, from (2.7) and the fact that each row of $(AQ)A^T$ is a linear combination of A^T's rows,

$$\text{rank}(\hat{Q}) \le \text{rank}(A^T) \le \min(p, l) \le l.$$

Therefore, $\hat{Q}$ tends to be only positive semi-definite because usually $p \neq l$. This result indicates that the connection between (3.15) and the rankSVM dual (2.4) is weaker than that between (3.14) and the SVM dual.

3.2 Loss Term After reformulating to (3.12) or (3.13), the number of variables is reduced to l. However, to calculate the loss term, we still have the summation over p preference pairs. The same difficulty occurs for linear rankSVM, so past works have proposed efficient algorithms. Among them, a recently developed method using order-statistic trees [1, 14] is the fastest, with only $O(l \log l)$ cost to compute this summation. For other values commonly used in optimization methods such as sub-gradient (if L1 loss is used), gradient, or Hessian-vector products (if L2 loss is adopted), the cost is also

[5] Note that the optimal $\boldsymbol{w}$ is unique because $\boldsymbol{w}^T\boldsymbol{w}/2$ is strictly convex.

$O(l \log l)$. We observe that the loss term of (3.12) is similar to that of linear rankSVM, which is of the form

$$\sum_{(i,j)\in P} \max(0, 1 - \boldsymbol{w}^T(\boldsymbol{x}_i - \boldsymbol{x}_j)).$$

Thus we can easily see that methods for summing up the p pairs in linear rankSVM is also applicable to nonlinear rankSVM. We give an illustration by showing the calculation of the loss term in (3.12). By defining

$$\mathrm{SV}(\boldsymbol{\beta}) \equiv \{(i,j) \mid (i,j) \in P, 1 - (Q\boldsymbol{\beta})_i + (Q\boldsymbol{\beta})_j > 0\},$$

the loss term can be written as

$$\begin{aligned} &\sum_{(i,j)\in \mathrm{SV}(\boldsymbol{\beta})} 1 - (Q\boldsymbol{\beta})_i + (Q\boldsymbol{\beta})_j \\ = &\sum_{i=1}^{l} (l_i^+(\boldsymbol{\beta}) - l_i^-(\boldsymbol{\beta}))(Q\boldsymbol{\beta})_i + l_i^-(\boldsymbol{\beta}), \end{aligned}$$

where

$$l_i^+(\boldsymbol{\beta}) \equiv |\{j \mid (j,i) \in \mathrm{SV}(\boldsymbol{\beta})\}|, \tag{3.17}$$

$$l_i^-(\boldsymbol{\beta}) \equiv |\{j \mid (i,j) \in \mathrm{SV}(\boldsymbol{\beta})\}|. \tag{3.18}$$

We can then use order-statistic trees to compute $l_i^+(\boldsymbol{\beta})$ and $l_i^-(\boldsymbol{\beta})$ in $O(l \log l)$ time; see details in [1, 14]. The calculation of sub-gradient (for L1 loss), and gradient as well as Hessian-vector products (for L2 loss) is similar. We give details in Appendix B. The major difference between the loss term of linear and nonlinear rankSVM is that the computation of $\boldsymbol{w}^T\boldsymbol{x}_i$ only costs $O(n)$ while obtaining $(Q\boldsymbol{\beta})_i$ requires l kernel evaluations, which usually amount to $O(ln)$ time. However, the $O(ln)$ cost can be reduced to $O(l)$ if Q is maintained throughout the optimization algorithm.

An important advantage of our method is that it is very general. Almost all unconstrained optimization methods can be applied.

3.3 Implementation Issues and Discussion Every time to evaluate the function value of (3.12), we need to conduct kernel evaluations. However, because Q is fixed regardless of the value of $\boldsymbol{\beta}$, it can be calculated and stored if space is permitted. Unfortunately, for large problems, storing the whole Q may not be possible because Q is a dense matrix. The same situation has occurred in SVM training, where the main solution is the decomposition method [9, 20, 3]. This method needs few kernel columns at each iteration and allocates available memory to store recently used columns (called kernel cache). The viability of decomposition methods relies on $O(l)$ cost per iteration if needed kernel columns are in the cache and $O(ln)$ if some kernel columns must be calculated. If similar decomposition methods are applied to rankSVM, the same property may not hold because calculating the sum of loss terms may become dominant by taking $O(l \log l)$ cost. Alternatively, because (3.12) and (3.13) are unconstrained, we can apply any general optimization method. Usually such methods must calculate the product between Q and $\boldsymbol{\beta}$. We can split Q to several blocks and store a fix portion in the memory. Other blocks are computed upon needed. Each Q's block can be efficiently obtained if the data set is dense and optimized numerical linear algebra subprograms (e.g., optimized BLAS [25]) are employed.

Our method may provide an efficient alternative to train linear rankSVM when $l \ll n$. Note that (3.12)-(3.13) involve a vector variable $\boldsymbol{\beta}$ of size l. In contrast, most existing methods train linear rankSVM by optimizing $\boldsymbol{w}$, which has n variables. Therefore, if $l \ll n$, using (3.12)-(3.13) may be superior to (3.11) because of a smaller number of variables. Because kernels are not used,

$$Q = XX^T,$$

where $X \equiv [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l]^T$. We can then easily conduct some operations without storing Q. For example, $Q\boldsymbol{\beta}$ can be calculated by

$$Q\boldsymbol{\beta} = X(X^T\boldsymbol{\beta}).$$

In linear rankSVM, using only a subset of pairs in P may reduce the training time significantly by slightly trading the performance [18]. Interestingly, this approach may not be useful for kernel rankSVM. For linear rankSVM, the dominant cost is to evaluate the summation over all preference pairs. Thus reducing the number of pairs can significantly reduce the cost. In contrast, the bottleneck for kernel rankSVM is on calculating $Q\boldsymbol{\beta}$. The $O(l^2)$ or even $O(l^2n)$ cost is much larger than $O(l \log l)$ for the loss term. Therefore, kernel rankSVM shares the same bottleneck with kernel SVM and support vector regression (SVR) [2, 6]. In this regard, kernel rankSVM may not be more expensive than kernel SVR, which is also applicable to learning to rank.[6] Contrarily, training linear rankSVM may cost more than linear SVR, which does not require the summation over all pairs.

4 Empirical Evaluation

We first implement a truncated Newton method to minimize (3.13). Next, we compare this implementation with the existing methods described in Section 2.

[6] This is called the pointwise approach because it approximates the relevance level of each individual training point.

4.1 An Implementation of the Proposed Method We consider (3.13) that is derived from L2-regularized L2-loss rankSVM. An advantage of using (3.13) rather than (3.12) is that (3.13) is differentiable. We then apply a type of truncated Newton methods called trust region Newton method (TRON) [15] to minimize (3.13). Past uses of TRON for machine learning include logistic regression and linear SVM [16], and linear rankSVM [14].

At the tth iteration with iterate $\boldsymbol{\beta}^t$, TRON finds a truncated Newton step $\boldsymbol{v}^t$ by approximately solving a linear system

$$\nabla^2 f(\boldsymbol{\beta}^t)\boldsymbol{v}^t = -\nabla f(\boldsymbol{\beta}^t), \tag{4.19}$$

where we use $f(\boldsymbol{\beta})$ to denote the objective function of (3.13). Note that $f(\boldsymbol{\beta})$ is not twice differentiable, so following [19] we consider a generalized Hessian; see (B.3) in Appendix B for details.

To approximately solve (4.19), TRON applies conjugate gradient (CG) methods that conduct a sequence of Hessian-vector products. We have discussed in Section 3.2 and Appendix B on the efficient calculation of Hessian-vector products and function/gradient values.

As a truncated Newton method, TRON confines $\boldsymbol{v}^t$ to be within a region that we trust. Then $\boldsymbol{\beta}$ is updated by

$$\boldsymbol{\beta}^{t+1} = \begin{cases} \boldsymbol{\beta}^t + \boldsymbol{v}^t & \text{if the function value} \\ & \text{sufficiently decreases,} \\ \boldsymbol{\beta}^t & \text{otherwise.} \end{cases}$$

If we fail to modify $\boldsymbol{\beta}^t$, the trust region is resized so that we search for $\boldsymbol{v}^t$ in a smaller region around $\boldsymbol{\beta}^t$. In contrast, if the function value sufficiently decreases, we enlarge the trust region. We omit other details of TRON because it is now a common optimization method for linear classification. More details can be seen in, for example, [16].

4.2 Data Sets and Evaluation Criteria We consider data from LETOR 4.0 [21]. We take three sets MQ2007, MQ2008 and MQ2007-list. Each set consists 5 folds and each fold contains its own training, validation and testing data. We take the first fold for each set. Because MQ2007-list is too large for some methods in the comparison, we randomly select queries to include 5% instances and form the training setMQ2007-list 5% and use it for all methods. Note that the feature values are already in the range $[0, 1]$; therefore, no scaling is conducted. The details of data sets are listed in Table 1.

Data set	l	n	k	$\lvert S\rvert$	p
MQ2007	42,158	46	3	1,017	246,015
MQ2008	9,360	46	3	471	52,325
MQ2007-list	743,790	46	1,268	1,017	285,943,893
MQ2007-list 5%	37,251	46	1,128	52	14,090,798

Table 1: The details of the first fold of each data set. l is the number of training instances. n is the number of features. k is the number of relevance levels. $|S|$ is the number of queries. p is the number of preference pairs. MQ2007-list 5% is sub-sampled from MQ2007-list.

To evaluate the prediction performance, we first consider pairwise accuracy because it is directly related to the loss term of RankSVM.

$$\text{Pairwise Accuracy} \equiv \frac{|\{(i,j) \in P : \boldsymbol{w}^T\boldsymbol{x}_i > \boldsymbol{w}^T\boldsymbol{x}_j\}|}{p}.$$

We also consider normalized discounted cumulative gain (NDCG), which is often used in information retrieval [8]. We follow LETOR 4.0 to use mean NDCG defined below. Assume k is a pre-specified positive integer, π is an ideal ordering such that

$$y_{\pi(1)} \geq y_{\pi(2)} \geq \ldots \geq y_{\pi(l_q)}, \forall q \in S,$$

where l_q is the number of instances in query q, and $\bar{\pi}$ is the ordering being evaluated. Then

$$\text{NDCG@}m \equiv I_m^{-1} \sum_{i=1}^{m} (2^{y_{\bar{\pi}(i)}} - 1) \cdot d(i), \text{ and}$$

$$\text{Mean NDCG} \equiv \frac{\sum_{m=1}^{l_q} \text{NDCG@}m}{l_q},$$

where

$$I_m \equiv \sum_{i=1}^{m} (2^{y_{\pi(i)}} - 1) \cdot d(i) \text{ and } d(i) \equiv \frac{1}{\log_2(\max(2, i))}.$$

After obtaining mean NDCG of each query, we take the average.

4.3 Settings for Experiments We compare the proposed method with the following methods.

- SVM^{light}: it is discussed in Section 2.1 by solving the dual problem of rankSVM.
- SVM^{rank}: this method, discussed in Section 2.2, solves an equivalent 1-slack structural SVM problem.
- RV-SVM: it implements the method discussed in Section 2.3 by using CPLEX[7] to solve the linear programming problem. Note that CPLEX supports parallel computing, but the extra memory requirement is beyond our machine's capacity. Hence we run CPLEX using only one thread.

[7] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

We consider Radial Bias Function (RBF) kernel

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|)$$

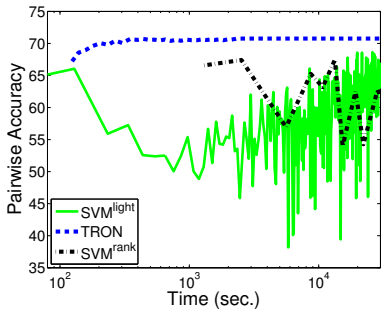
and conduct the experiment on a 64-bit machine with Intel Xeon 2.0GHz (E5504), 1 MB cache and 32GB memory.

For parameter selection, we must decide C and γ, where C is the regularization parameter in (1.2), and γ is the kernel parameter in the RBF kernel. Because the compared approaches solve problems under different loss or regularization terms, parameter selection is conducted for each method and each evaluation criterion. We search on a grid of (C, γ) values, where $C \in \{2^{-5}, 2^{-4}, \ldots, 2^{5}\}$ and $\gamma \in \{2^{-6}, 2^{-5}, \ldots, 2^{-1}\}$. We choose the one that has the best prediction performance on the validation set. The best parameter (C, γ) for each method and criterion is listed in the supplementary materials. Note that RV-SVM failed to run on MQ2007 and MQ2007-list 5% because the constraint matrix is too large to fit in the memory. Thus no best parameters are presented.

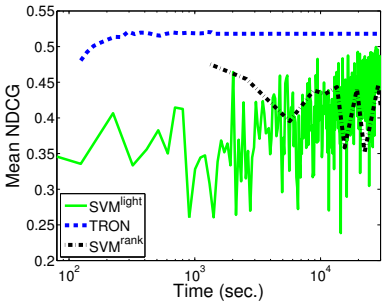
We mentioned that kernel evaluations are a bottleneck in training rankSVM. To reduce repeated kernel evaluations, our implementation stores the full kernel matrix Q , SVM^{light} caches partial $\tilde{Q}$ that can fit in the memory, and SVM^{rank} stores full kernel matrix of the dual of sub-problem of (2.8). For RV-SVM, it stores the full AQ matrix required by CPLEX for solving the linear programming problem (2.10).

4.4 Comparison Results We record prediction performances for every iteration of TRON and SVM^{rank}, and every 50 iterations of RV-SVM and SVM^{light}. Figures 1-3 present the relation between training time and test performances.

The compared methods solve optimization problems with different regularization and loss terms, so their final pairwise accuracy or NDCG may be slightly different. Instead, the goal of the comparison is to check that, for each method, how fast its optimization procedure converges. From Figures 1-3, it is clear that the proposed method very quickly achieves the performance of the final optimal solution. Because the training time is log-scaled, it is much faster than others. The package SVM^{rank} comes the second, although, in the figures, its performance is not stabilized yet in the end. For SVM^{light}, the performance is unstable because the optimization problem has not been accurately solved. For smaller data sets, we did observe the convergence to the final performance if enough running time is spent. For RV-SVM, we have mentioned that it can handle only the smaller problem MQ2008.

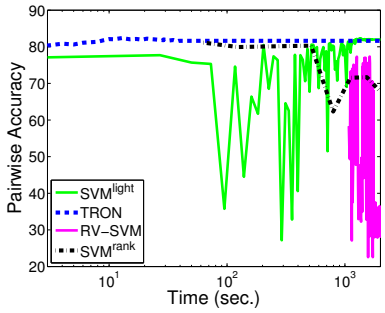


(a) Pairwise accuracy

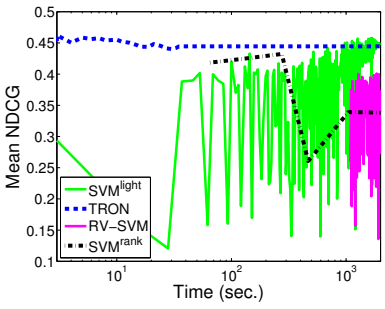


(b) Mean NDCG

Figure 1: Experimental results on MQ2007



(a) Pairwise accuracy



(b) Mean NDCG

Figure 2: Experimental results on MQ2008

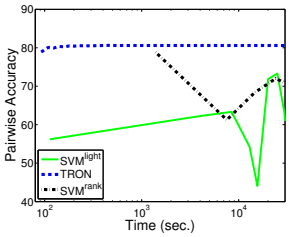


Figure 3: Experimental results on MQ2007-list 5%. Note that mean NDCG is not available for MQ2007-list 5% because of the overflow of $2^{\bar{\pi}(i)}$ caused by the large k.

5 Conclusions

In this paper, we propose a method for efficiently training rankSVMs. The number of variables is reduced from quadratic to linear to training instances. Efficient method to handle the large number of preference pairs are are incorporated. Empirical comparisons show that the training time is reduced in magnitudes when compared to state-of-the-art methods. Although only one loss function is used in our implementation for the experiments because of the lack of space, our method is applicable to a variety of loss functions and different optimization methods. The proposed approach makes kernel rankSVM a practically feasible model. The programs used for our experiment is available at `http://www.csie.ntu.edu.tw/~cjlin/papers/ranksvm/kernel.tar.gz` and we release a package based on our study. It is available at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/#large_scale_ranksvm`.

Acknowledgement

This work was supported in part by the National Science Council of Taiwan via the grant 101-2221-E-002-199-MY3. The authors thank Antti Airola for his comments.

References

[1] A. Airola, T. Pahikkala, and T. Salakoski. Training linear ranking SVMs in linearithmic time using red–black trees. *PR Letters*, 32(9):1328–1336, 2011.

[2] B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *COLT*, 1992.

[3] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM TIST*, 2:27:1–27:27, 2011.

[4] O. Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5):1155–1178, 2007.

[5] O. Chapelle and S. S. Keerthi. Efficient algorithms for ranking with SVMs. *Information Retrieval*, 13(3):201–215, 2010.

[6] C. Cortes and V. Vapnik. Support-vector network. *Machine Learning*, 20:273–297, 1995.

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

[8] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[9] T. Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

[10] T. Joachims. Optimizing search engines using clickthrough data. In *ACM KDD*, 2002.

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

[12] T. Joachims, T. Finley, and C.-N. J. Yu. Cutting plane training of structural SVMs. *Machine Learning*, 77, 2009.

[13] G. S. Kimeldorf and G. Wahba. A correspondence between Bayesian estimation on stochastic processes and smoothing by splines. *The Annals of Mathematical Statistics*, 41:495–502, 1970.

[14] C.-P. Lee and C.-J. Lin. Large-scale linear rankSVM. *Neural Computation*, 2014. To appear.

[15] C.-J. Lin and J. J. Moré. Newton's method for large-scale bound constrained problems. *SIAM Journal on Optimization*, 9:1100–1127, 1999.

[16] C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region Newton method for large-scale logistic regression. *JMLR*, 9:627–650, 2008.

[17] K.-M. Lin and C.-J. Lin. A study on reduced support vector machines. *IEEE TNN*, 14(6):1449–1559, 2003.

[18] K.-Y. Lin. Data selection techniques for large-scale rankSVM. Master's thesis, National Taiwan University, 2010.

[19] O. L. Mangasarian. A finite Newton method for classification. *Optimization Methods and Software*, 17(5):913–929, 2002.

[20] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, Cambridge, MA, 1998. MIT Press.

[21] T. Qin, T.-Y. Liu, J. Xu, and H. Li. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13(4):346–374, 2010.

[22] B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In *COLT*, 2001.

[23] D. Sculley. Large scale learning to rank. In *NIPS 2009 Workshop on Advances in Ranking*. 2009.

[24] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 2005.

[25] R. C. Whaley, A. Petitet, and J. J. Dongarra. Automatically tuned linear algebra software and the ATLAS project. Technical report, Department of Computer Sciences, University of Tennessee, 2000.

[26] H. Yu, J. Kim, Y. Kim, S. Hwang, and Y. H. Lee. An efficient method for learning nonlinear ranking SVM functions. *Information Sciences*, 209:37–48, 2012.

[27] G.-X. Yuan, C.-H. Ho, and C.-J. Lin. Recent advances of large-scale linear classification. *PIEEE*, 100:2584–2603, 2012.

A Optimality of $\boldsymbol{w}$ Defined in (3.16)

Assume $\boldsymbol{\alpha}^*$ and $\hat{\boldsymbol{\beta}}^*$ are optimal for (2.4) and (3.15), respectively. By the strong duality of rankSVM and the feasibility of $\boldsymbol{\alpha}^*$ to problem (3.15), we have

$$\begin{aligned}
&\text{optimal value of (1.2)}\\
&= \frac{1}{2}(\boldsymbol{\alpha}^*)^T\hat{Q}\boldsymbol{\alpha}^* + C\sum\nolimits_{(i,j)\in P}\max(0, 1-(\hat{Q}\boldsymbol{\alpha}^*)_{i,j})\\
&\geq \frac{1}{2}(\hat{\boldsymbol{\beta}}^*)^T\hat{Q}\hat{\boldsymbol{\beta}}^* + C\sum\nolimits_{(i,j)\in P}\max(0, 1-(\hat{Q}\hat{\boldsymbol{\beta}}^*)_{i,j})\\
&\geq \text{optimal value of (1.2)}.
\end{aligned}$$

The last inequality is from the fact that any $\boldsymbol{w}$ constructed by (3.16) is feasible for (1.2). Thus, the above inequalities are in fact equalities, so the proof is complete. Further, the above equation implies that any dual optimal solution $\boldsymbol{\alpha}^*$ is also optimal for (3.15).

B Calculation of First and Second Order Information of (3.12) and (3.13)

Assume Q_i is the i-th column of Q. One sub-gradient of (3.12)'s objective function is

$$\begin{aligned}
&Q\boldsymbol{\beta} - C\sum\nolimits_{(i,j)\in \mathrm{SV}(\boldsymbol{\beta})}(Q_i - Q_j)\\
&= Q\boldsymbol{\beta} + C\sum\nolimits_{i=1}^{l}(\sum\nolimits_{(j,i)\in \mathrm{SV}(\boldsymbol{\beta})} Q_i - \sum\nolimits_{(i,j)\in \mathrm{SV}(\boldsymbol{\beta})} Q_i)\\
&= Q\boldsymbol{\beta} + C\sum\nolimits_{i=1}^{l}(l_i^+(\boldsymbol{\beta}) - l_i^-(\boldsymbol{\beta}))Q_i,
\end{aligned}$$

For (3.13) that uses L2 loss, the computation is as follows. We define $p_{\boldsymbol{\beta}} \equiv |\mathrm{SV}(\boldsymbol{\beta})|$, and let $A_{\boldsymbol{\beta}} \in \mathbf{R}^{p_{\boldsymbol{\beta}}\times l}$ include A's rows corresponding to $\mathrm{SV}(\boldsymbol{\beta})$. That is, if $(i,j) \in \mathrm{SV}(\boldsymbol{\beta})$, then the (i,j)th row of A is selected. The objective function of (3.13) can then be written as

$$\begin{aligned}
f(\boldsymbol{\beta}) &\equiv \frac{1}{2}\boldsymbol{\beta}^T Q\boldsymbol{\beta} + C(A_{\boldsymbol{\beta}}Q\boldsymbol{\beta} - \boldsymbol{e}_{\boldsymbol{\beta}})^T(A_{\boldsymbol{\beta}}Q\boldsymbol{\beta} - \boldsymbol{e}_{\boldsymbol{\beta}})\\
&= \frac{1}{2}\boldsymbol{\beta}^T Q\boldsymbol{\beta} + C\boldsymbol{\beta}^T Q(A_{\boldsymbol{\beta}}^T A_{\boldsymbol{\beta}}Q\boldsymbol{\beta} - 2A_{\boldsymbol{\beta}}^T\boldsymbol{e}_{\boldsymbol{\beta}}) + p_{\boldsymbol{\beta}}, \qquad \text{(B.1)}
\end{aligned}$$

where $\boldsymbol{e}_{\boldsymbol{\beta}} \in \mathbf{R}^{p_{\boldsymbol{\beta}}}$ is a vector of ones. Its gradient is

$$\begin{aligned}
&Q\boldsymbol{\beta} - 2C\sum_{(i,j)\in P}(Q_i - Q_j)\max(0, 1-(Q\boldsymbol{\beta})_i + (Q\boldsymbol{\beta})_j)\\
&\qquad = Q\boldsymbol{\beta} + 2CQ(A_{\boldsymbol{\beta}}^T A_{\boldsymbol{\beta}}Q\boldsymbol{\beta} - A_{\boldsymbol{\beta}}^T\boldsymbol{e}_{\boldsymbol{\beta}}). \qquad \text{(B.2)}
\end{aligned}$$

Because $f(\boldsymbol{\beta})$ is not twice differentiable, $\nabla^2 f(\boldsymbol{\beta})$ does not exist. We follow [19] and [16] to define a generalized Hessian matrix $\nabla^2 f(\boldsymbol{\beta}) \equiv Q + 2CQA_{\boldsymbol{\beta}}^T A_{\boldsymbol{\beta}}Q$. Because this matrix may be too large to be stored, some optimization methods employ the Hessian-free techniques so that only the Hessian-vector products are needed. For any vector $\boldsymbol{v} \in \mathbf{R}^l$,

$$\nabla^2 f(\boldsymbol{\beta})\boldsymbol{v} = Q\boldsymbol{v} + 2CQA_{\boldsymbol{\beta}}^T A_{\boldsymbol{\beta}}Q\boldsymbol{v}. \qquad \text{(B.3)}$$

We notice that (B.1)-(B.3) share some common terms, which can be computed by the same method. For $A_{\boldsymbol{\beta}}^T\boldsymbol{e}_{\boldsymbol{\beta}}$ and $p_{\boldsymbol{\beta}}$ needed in (B.1) and (B.2),

$$A_{\boldsymbol{\beta}}^T\boldsymbol{e}_{\boldsymbol{\beta}} = \begin{bmatrix} l_1^-(\boldsymbol{\beta}) - l_1^+(\boldsymbol{\beta}) \\ \vdots \\ l_l^-(\boldsymbol{\beta}) - l_l^+(\boldsymbol{\beta}) \end{bmatrix}, \text{ and } p_{\boldsymbol{\beta}} = \sum\nolimits_{i=1}^{l} l_i^-(\boldsymbol{\beta}),$$

where $l_i^+(\boldsymbol{\beta})$ and $l_i^-(\boldsymbol{\beta})$ are defined in (3.17) and (3.18). Next we calculate $A_{\boldsymbol{\beta}}^T A_{\boldsymbol{\beta}}Q\boldsymbol{\beta}$ and $A_{\boldsymbol{\beta}}^T A_{\boldsymbol{\beta}}Q\boldsymbol{v}$. From

$$(A_{\boldsymbol{\beta}}^T A_{\boldsymbol{\beta}})_{i,j} = \sum_s (A_{\boldsymbol{\beta}})_{i,s}^T (A_{\boldsymbol{\beta}})_{s,j} = \sum_s (A_{\boldsymbol{\beta}})_{s,i}(A_{\boldsymbol{\beta}})_{s,j},$$

and each row of $A_{\boldsymbol{\beta}}$ only contains two non-zero elements, we have

$$(A_{\boldsymbol{\beta}}^T A_{\boldsymbol{\beta}})_{i,j} = \begin{cases} l_i^+(\boldsymbol{\beta}) + l_i^-(\boldsymbol{\beta}) & \text{if } i = j,\\ -1 & \text{if } i \neq j, \text{ and}\\ & \quad (i,j) \text{ or } (j,i) \in \mathrm{SV}(\boldsymbol{\beta}),\\ 0 & \text{otherwise.} \end{cases}$$

Consequently,

$$\begin{aligned}
&(A_{\boldsymbol{\beta}}^T A_{\boldsymbol{\beta}}Q\boldsymbol{v})_i = \sum\nolimits_{j=1}^{l}(A_{\boldsymbol{\beta}}^T A_{\boldsymbol{\beta}})_{i,j}(Q\boldsymbol{v})_j\\
&= \big(l_i^+(\boldsymbol{\beta}) + l_i^-(\boldsymbol{\beta})\big)(Q\boldsymbol{v})_i - \sum\nolimits_{j:(j,i) \text{ or } (i,j)\in \mathrm{SV}(\boldsymbol{\beta})}(Q\boldsymbol{v})_j.
\end{aligned}$$

Finally we have

$$\begin{aligned}
&QA_{\boldsymbol{\beta}}^T A_{\boldsymbol{\beta}}Q\boldsymbol{v}\\
&= Q\begin{bmatrix} \big(l_1^+(\boldsymbol{\beta}) + l_1^-(\boldsymbol{\beta})\big)(Q\boldsymbol{v})_1 - \big(\gamma_1^+(\boldsymbol{\beta},\boldsymbol{v}) + \gamma_1^-(\boldsymbol{\beta},\boldsymbol{v})\big) \\ \vdots \\ \big(l_l^+(\boldsymbol{\beta}) + l_l^-(\boldsymbol{\beta})\big)(Q\boldsymbol{v})_l - \big(\gamma_l^+(\boldsymbol{\beta},\boldsymbol{v}) + \gamma_l^-(\boldsymbol{\beta},\boldsymbol{v})\big) \end{bmatrix},
\end{aligned}$$

where

$$\begin{aligned}
\gamma_i^+(\boldsymbol{\beta},\boldsymbol{v}) &\equiv \sum\nolimits_{j:(j,i)\in \mathrm{SV}(\boldsymbol{\beta})}(Q\boldsymbol{v})_j,\\
\gamma_i^-(\boldsymbol{\beta},\boldsymbol{v}) &\equiv \sum\nolimits_{j:(i,j)\in \mathrm{SV}(\boldsymbol{\beta})}(Q\boldsymbol{v})_j.
\end{aligned}$$

It has been shown in [14] that $\gamma_i^+(\boldsymbol{\beta},\boldsymbol{v})$ and $\gamma_i^-(\boldsymbol{\beta},\boldsymbol{v})$ can be calculated by the same technique of order-statistic trees for $l_i^+(\boldsymbol{\beta})$ and $l_i^-(\boldsymbol{\beta})$.