EXACT AND EFFICIENT LEAVE-PAIR-OUT CROSS-VALIDATION FOR RANKING RLS

Tapio Pahikkala, Antti Airola, Jorma Boberg, and Tapio Salakoski

Turku Centre for Computer Science (TUCS),
Department of Information Technology,
University of Turku, Turku, Finland,
firstname.lastname@utu.fi

ABSTRACT

In this paper, we introduce an efficient cross-validation algorithm for RankRLS, a kernel-based ranking algorithm. Cross-validation (CV) is one of the most useful methods for model selection and performance assessment of machine learning algorithms, especially when the number of labeled data is small. A natural way to measure the performance of ranking algorithms by CV is to hold each data point pair out from the training set at a time and measure the performance with the held out pair. This approach is known as leave-pair-out cross-validation (LPOCV).

We present a computationally efficient algorithm for performing LPOCV for RankRLS. If RankRLS is already trained with the whole training set, the computational complexity of the algorithm is $O(m^2)$. Further, if there are d outputs to be learned simultaneously, the computational complexity of performing LPOCV is $O(m^2d)$. An approximative $O(m^2)$ time LPOCV algorithm for RankRLS has been previously proposed, but our method is the first exact solution to this problem.

We introduce a general framework for developing and analysing hold-out and cross-validation techniques for quadratically regularized kernel-based learning algorithms. The framework is constructed using a value regularization based variant of the representer theorem. We provide a simple proof for this variant using matrix calculus. Our cross-validation algorithm can be seen as an instance of this framework.

1. INTRODUCTION

Learning to rank has been a topic of interest in the machine learning community during the recent years. The problem of ordering a group of instances according to a preference relation arises naturally in domains such as information retrieval and collaborative filtering, and as a special case of ranking we encounter the task of area under curve (AUC) maximization in fields where this metric is preferred.

The ranking task is often cast as a pairwise classification problem, where instance pairs are used as training examples, and the labels indicate the direction of the preference. Algorithms based on this approach, such as the RankSVM [Herbrich et al., 1999, Joachims, 2002], have been shown to achieve high ranking performance, but often at the cost of computational efficiency. Consideration of pairs instead of individual instances results in quadratic growth in the number of training examples, and thus also increased time complexity in training. Computational shortcuts exist for special cases such as the linear version of RankSVM used with sparse data [Joachims, 2006]. However, in the general case when nonlinear kernel functions are used, methods such as RankSVM can be quite inefficient in training (see e.g. [Schölkopf and Smola, 2002, Shawe-Taylor and Cristianini, 2004] for more information about kernel functions).

Regularized least-squares (RLS) (see e.g. [Rifkin et al., 2003, Poggio and Smale, 2003] and references therein) is a kernel-based learning algorithm that is closely related to the support vector machines. Lately, it has been shown that it is possible to derive regularized least-squares (RLS) based ranking algorithms whose complexity is the same as that of standard RLS regression and which perform on state-of-the-art level. Namely, two very similar RLS based ranking algorithms have been independently proposed, the RankRLS introduced by [Pahikkala et al., 2007], and the MPRank [Cortes et al., 2007b]. In addition, these algorithms have the property that in addition to learning the pairwise ranking of the data points, they also preserve the magnitude of the preferences.

When kernels are used the complexity of training RankRLS is cubic with respect to the number of training examples. While this is an improvement over approaches such as the RankSVM, still it means that the algorithm is not in the general case suitable for large scale learning. Rather, its use is most advantageous on small datasets, the type of which are encountered frequently, for example, when solving medical and biological tasks. Getting additional validation and test data may in these kind of domains be not an option, as producing the data may be very expensive. In such cases cross-validation is frequently used for parameter choosing and performance evaluation. For example, when aiming to build a classifier maximizing the AUC metric with biological data as considered by [Parker et al., 2007], cross-validation is commonly used for performance evaluation. RankRLS can be easily modified to perform AUC maximization as considered by us in [Pahikkala et al., 2008].

Analogously to leave-one-out cross-validation, one can define leave-pair-out cross-validation, where pairs of training instances are left out of the training set. This approach is natural for the pairwise ranking tasks and it guarantees the maximal use of available data. The LPOCV estimate, taken over a training set of m examples, is an unbiased estimate of the true error over a sample of $m-2$ examples (for a proof see [Cortes et al., 2007a]).

A naive implementation for LPOCV would require training one model per holdout pair. The complexity of training RankRLS is $O(m^3)$ in the worst case, where m is the number of training examples. Because of the quadratic number of possible pairs, this approach can result in $O(m^5)$ LPOCV complexity, which is extremely inefficient even for quite small datasets. However, using techniques based on matrix calculus the closed form solution of RankRLS can be manipulated to derive efficient algorithms for cross-validation. Such methods have been previously proposed for the standard RLS regression by [Pahikkala et al., 2006] and by [An et al., 2007]. For the task of efficient LPOCV in RLS-based ranking, an approximative algorithm was recently proposed in [Cortes et al., 2007a]. In this paper we extend these results by deriving an exact and efficient LPOCV-algorithm for RLS-based ranking.

2. PRELIMINARIES

We assume the scored object ranking setting, where the learner is supplied with objects each of which are associated with a score that represents the “goodness” of the object with regards to the ranking criterion used. The ranking can be derived from these scores, with objects having higher scores being preferred over objects with lower scores. The task is to learn a transitive preference relation that can be used to order any given set of the same type of objects.

We construct a training set from a given set of m data points. A data point $z = (x, y)$ consist of an input variable $x \in \mathcal{X}$ (object) and an output variable $y \in \mathbb{R}$ (score), where $\mathcal{X}$, called the input space, can be any set. For a pair of data points $z_1 = (x_1, y_1)$ and $z_2 = (x_2, y_2)$ we say that z_1 is preferred over z_2 if $y_1 > y_2$. The magnitude of this preference is defined as be $y_1 - y_2$.

Further, let $X = (x_1, \dots, x_m) \in (\mathcal{X}^m)^{\mathrm{T}}$ be a sequence of inputs, where $(\mathcal{X}^m)^{\mathrm{T}}$ denotes the set of row vectors whose elements belong to $\mathcal{X}$. Correspondingly, we define $Y = (y_1, \dots, y_m)^{\mathrm{T}} \in \mathbb{R}^m$ be a sequence of the corresponding output variables. Altogether, we define the training set to be the pair $S = (X, Y)$.

Let us denote $\mathbb{R}^{\mathcal{X}} = \{f : \mathcal{X} \to \mathbb{R}\}$, and let $\mathcal{H} \subseteq \mathbb{R}^{\mathcal{X}}$ be the hypothesis space. In order to construct an algorithm that selects a hypothesis f from $\mathcal{H}$, we have to define an appropriate cost function that measures how well the hypotheses fit to the training data. Further, we should avoid too complex hypotheses that overfit at training phase and are not able to generalize to unseen data. Following [Schölkopf et al., 2001], we consider the framework of regularized kernel methods in which $\mathcal{H}$ is the reproducing kernel Hilbert space (RKHS) defined by a positive definite kernel function k. The kernel functions are defined as follows. Let $\mathcal{F}$ denote the feature vector space. For any mapping

$$\Phi : \mathcal{X} \to \mathcal{F},$$

the inner product

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle$$

of the mapped data points is called a kernel function. We also denote the sequence of feature mapped inputs as

$$\Phi(X) = (\Phi(x_1), \dots, \Phi(x_m)) \in (\mathcal{F}^m)^{\mathrm{T}}$$

for all $X \in (\mathcal{X}^m)^{\mathrm{T}}$. Further, we define the symmetric kernel matrix $K \in \mathbb{R}^{m \times m}$, where $\mathbb{R}^{m \times m}$ denotes the set of real matrices of type $m \times m$, as

$$K = \Phi(X)^{\mathrm{T}}\Phi(X) = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_m) \\ \vdots & \ddots & \vdots \\ k(x_m, x_1) & \cdots & k(x_m, x_m) \end{pmatrix}$$

for all $X \in (\mathcal{X}^m)^{\mathrm{T}}$. Unless stated otherwise, we assume that the kernel matrix is strictly positive definite, that is, $\vec{a}^{\mathrm{T}} K \vec{a} > 0$ for all $\vec{a} \in \mathbb{R}^m, \vec{a} \neq \vec{0}$. This can be ensured, for example, by performing a small diagonal shift.

Using RKHS as our hypothesis space, we define the learning algorithm as

$$\mathcal{A}(S) = \operatorname*{arginf}_{f \in \mathcal{H}} J(f),$$

where

$$J(f) = l(f(X), Y) + \lambda \|f\|_k^2, \tag{1}$$

$f(X) = (f(x_1), \dots, f(x_m))^{\mathrm{T}}$, l is a real valued cost function, and $\lambda \in \mathbb{R}_+$ is a regularization parameter controlling the tradeoff between the cost on the training set and the complexity of the hypothesis. By the generalized representer theorem ([Schölkopf et al., 2001]), the minimizer of (1) has the following form:

$$f(x) = \sum_{i=1}^{m} a_i k(x, x_i), \tag{2}$$

where $a_i \in \mathbb{R}$. The implication of the representer theorem is that, regardless of the cost function used, the minimizer can always be expressed as a vector of m real valued coefficients. Therefore, we are free to select such a cost function that coheres with the learning task in question and has desirable computational advantages.

Using (2) and the matrix notation, we rewrite

$$f(X) = K\vec{a} \tag{3}$$

and

$$\|f\|_k^2 = \vec{a}^{\mathrm{T}} K \vec{a},$$

where $\vec{a} = (a_1, \dots, a_m)^{\mathrm{T}}$. Therefore, the objective (1) to be minimized can be rewritten as a function of $\vec{a}$ as follows:

$$J(\vec{a}) = l(K\vec{a}, Y) + \lambda \vec{a}^{\mathrm{T}} K \vec{a}. \tag{4}$$

3. RANKRLS

Following [Pahikkala et al., 2007] we consider the following type of least-squares based ranking cost function

$$l(f(X),Y) = \frac{1}{2}\sum_{i,j=1}^{m}((y_i-y_j)-(f(x_i)-f(x_j)))^2. \quad (5)$$

The cost function is the sum of the squares of differences between the predicted and correct magnitudes of all the pairwise preferences in the training set. Note that in [Pahikkala et al., 2007], a more general formulation was considered, in which it is possible to define which of the pairs the cost is evaluated over. For the sake of simplicity, we consider here only the all-pairs case.

We observe that for any vector $r \in \mathbb{R}^m$ we can write

$$\begin{aligned} \frac{1}{2}\sum_{i,j=1}^{m}(r_i - r_j)^2 &= m\sum_{i=1}^{m} r_i^2 - \sum_{i,j=1}^{m} r_i r_j \\ &= m \cdot r^{\mathrm{T}} I r - r^{\mathrm{T}} \vec{1}\vec{1}^{\mathrm{T}} r \\ &= r^{\mathrm{T}} L r, \end{aligned}$$

where $\vec{1}$ denotes a vector whose each element is 1, and L, which we call the Laplacian matrix, is defined as

$$L_{i,j} = \begin{cases} -1 & \text{if } i \neq j \\ m-1 & \text{if } i = j \end{cases}.$$

Accordingly, we can rewrite the cost function (5) in a matrix form as follows:

$$l(f(X),Y) = (Y - K\vec{a})^{\mathrm{T}} L (Y - K\vec{a}),$$

and hence the objective function (1) becomes

$$J(\vec{a}) = (Y - K\vec{a})^{\mathrm{T}} L (Y - K\vec{a}) + \lambda \vec{a}^{\mathrm{T}} K \vec{a}.$$

Taking derivative of $J(\vec{a})$ with respect to $\vec{a}$ and setting it to zero, we can determine the value of the coefficient vector $\vec{a}$ that determines a minimizer of (1) for a training set S:

$$\vec{a} = (KLK + \lambda K)^{-1} KLY. \quad (6)$$

For detailed proofs we refer to [Pahikkala et al., 2007]. We note that if Y is a $m \times d$-matrix instead of a single column vector, that is, each column corresponds to a separate subproblem, we can calculate the corresponding d-column coefficient matrix using (6) at the cost of calculating only a single-column coefficient vector.

Calculation of the solution to (6) requires multiplications and inversions of $m \times m$-matrices. Both types of operations are usually performed with methods whose computational complexities are $O(m^3)$, and hence the complexity of RankRLS is equal to the complexity of the RLS regression.

We also consider RankRLS from the following perspective which is called value regularization by [Rifkin and Lippert, 2007]. Instead of solving the coefficients $\vec{a}$ by minimizing (4), we directly solve the predictions (3) made by the learner for its training examples, that is, we solve

$$f(X) = \operatorname*{arginf}_{\vec{p} \in \mathbb{R}^m} J(\vec{p}), \quad (7)$$

where

$$J(\vec{p}) = l(\vec{p}, Y) + \lambda \vec{p}^{\mathrm{T}} K^{-1} \vec{p}. \quad (8)$$

Here, the objective function (8) is obtained from (4) by replacing $\vec{a}$ with $K^{-1}\vec{p}$, since according to (3), the coefficients determining the prediction function can then be obtained from

$$\vec{a} = K^{-1} f(X). \quad (9)$$

In the above considerations, we have assumed the strict positive definiteness and hence the invertibility of the kernel matrix K. However, the value regularization perspective can also be used when K is singular. In that case, the term $\vec{p}^{\mathrm{T}} K^{-1} \vec{p}$ should be interpreted as

$$\lim_{\epsilon \to 0^+} \vec{p}^{\mathrm{T}} (K + \epsilon I)^{-1} \vec{p}$$

(see [Johnson and Zhang, 2008] for more thorough discussion).

When considering RankRLS from the value regularization perspective we rewrite (8) as

$$J(\vec{p}) = (Y - \vec{p})^{\mathrm{T}} L (Y - \vec{p}) + \lambda \vec{p}^{\mathrm{T}} K^{-1} \vec{p}, \quad (10)$$

which is minimized by

$$\vec{p} = (L + \lambda K^{-1})^{-1} LY. \quad (11)$$

We note again that if Y is a $m \times d$-matrix instead of a single column vector, we can calculate the minimizers with (11) for each of the d subproblems simultaneously at the cost of calculating only one.

4. LEAVE-PAIR-OUT CROSS-VALIDATION

Next, we introduce our shorthand notation. Let $U \subset \{1,\ldots,m\}$ denote an index set in which the indices refer to a certain examples in the training set. Moreover, we denote $\overline{U} = \{1,\ldots,m\} \setminus U$. Below, with $p \in \mathbb{N}$ and any matrix or vector $R \in \mathbb{R}^{m \times p}$ that has its rows indexed by the training examples, we use the subscript U so that a matrix $R_U \in \mathbb{R}^{|U| \times p}$ contains only the rows that are indexed by U. For $R \in \mathbb{R}^{m \times m}$, we also use $R_{UU} \in \mathbb{R}^{|U| \times |U|}$ to denote a matrix that contains only the rows and the columns that are indexed by U. Let $Y \in \mathbb{R}^m$ be the label vector corresponding to the training data. Further, we define this notation also for the sequence of inputs so that X_U denotes the sequence consisting of the input indexed by U. Finally, let $f_{\overline{U}}$ be the function obtained by training the RLS algorithm with the whole data set except the set of data points indexed by U.

We now consider LPOCV in which every pair of training examples is held out from the training process at a time and the error corresponding to the pair is calculated. The result of LPOCV is the averaged error over the pairs. Let $U = \{h_1, h_2\}$ be the index set containing the indices h_1 and h_2 of the hold-out training examples. Then, the performance of a learner $f_{\overline{U}}$ on the the two hold-out data points can be computed with a performance measure

$$\mu(Y_{\overline{U}}, f_{\overline{U}}(X_U)). \quad (12)$$

We call this a hold-out performance of a learner. The performance measure can be, for example, the ranking loss function as in [Pahikkala et al., 2007] or the magnitude preserving loss function as used in [Cortes et al., 2007b].

In cross-validation, we have a sequence of hold-out sets $U_1, \ldots, U_{(m^2-m)/2}$. The overall cross-validation performance is obtained by averaging (12) over the hold-out sets:

$$\frac{1}{(m^2-m)/2} \sum_{j=1}^{(m^2-m)/2} \mu(Y_{\overline{U}_j}, f_{\overline{U}_j}(X_{U_j})). \qquad (13)$$

Recently, [Cortes et al., 2007a] have proposed an algorithm that approximates the result of LPOCV for the object ranking in $O(m^2)$ time, provided that an inversion of a certain $m \times m$-matrix is already computed and stored in the memory. The larger the number of training examples is, the closer the approximation to the exact result of the cross-validation is. Here, we improve their result by presenting an algorithm that calculates an exact result of LPOCV in $O(m^2)$ time, again given that the inverse of a certain $m \times m$-matrix is already computed and stored in the memory.

First, we present a theorem that provides us additional insight about how to calculate the hold-out predictions for a pair of data points if the learner is already trained with the whole data set. The theorem has already been proved by [Rifkin and Lippert, 2007] using the theory of Fenchel duality. However, we show that it can be proven in a simpler way that is based on matrix calculus only. Our proof of the theorem is presented in the appendix.

Theorem 1.

$$f_{\overline{U}}(X) = \operatorname*{arginf}_{\vec{v} \in \mathbb{R}^m} \left\{ l(\vec{v}_{\overline{U}}, Y_{\overline{U}}) + \lambda \vec{v}^T K^{-1} \vec{v} \right\}.$$

□

By comparing Theorem 1 with the value regularization perspective (7), we observe that the hold-out predictions can be obtained by removing the effect of the hold-out data points only from the loss function. They do not have to be removed from the regularizer. Note that this property holds for the objective function (8) with any cost function, and hence this provides us a powerful framework for designing cross-validation algorithms.

Next, we present a theorem that characterizes our LPOCV algorithm. The proof of the theorem is again presented in the appendix.

Theorem 2. *Let $\widetilde{D} = (m-2)I$ and let $Q = (\widetilde{D} + \lambda K^{-1})^{-1}$. Further, let $C \in \mathbb{R}^{m \times 3}$ be a matrix whose values are determined by*

$$C_{i,j} = \begin{cases} 1 & \text{if } j = 1 \\ 0 & \text{otherwise} \end{cases}.$$

We assume that we have calculated the matrices

$$Q, \widetilde{D}Y, Q\widetilde{D}Y, QC, C^TQC, C^TY, QCC^TY, \text{ and } C^TQ\widetilde{D}Y, \qquad (14)$$

and stored them into the memory before starting the hold-out calculations. Then, the hold-out predictions for two training examples can be performed in a constant time if the number of outputs is one. Moreover, if the number of outputs is d, then the predictions can be calculated in $O(d)$ time. Finally, the leave-pair-out cross-validation can be performed in $O(m^2)$ time if the number of outputs is one and in $O(m^2d)$ time if the number of outputs d. □

Concerning the matrices (14) calculated in advance, the calculation of the matrix Q is the computationally dominant one. Namely, its time complexity is $O(m^3)$ in the worst case of K being of full rank. This is the same as that of training the RankRLS algorithm in the worst case. However, if K is not of full rank, the matrix Q can be calculated as follows. Let $K = V\Lambda V^{\mathrm{T}}$ be the eigen decomposition of K, where V contains the eigenvectors of K and Λ is a diagonal matrix containing the eigenvalues of K. Then,

$$Q = V\widehat{\Lambda}V^{\mathrm{T}},$$

where $\widehat{\Lambda}$ is a diagonal matrix whose elements are determined by

$$\widehat{\Lambda}_{i,i} = \frac{\Lambda_{i,i}}{\lambda + (m-2)\Lambda_{i,i}}.$$

If many of the eigenvalues $\widehat{\Lambda}_{i,i}$ are zeros, we only need to calculate eigenvectors corresponding to the nonzero eigenvalues in order to calculate Q. This can speed up the computation, for example, if the linear kernel function is used and the dimensionality n of the feature space is smaller than the size of the training set m, and hence the number of nonzero eigenvalues is at most n.

5. DISCUSSION AND CONCLUSION

In this paper, we provide a simple proof for a value regularization based variant of representer theorem using matrix calculus, which leads to a powerful framework for developing and analysing hold-out and cross-validation techniques for quadratically regularized kernel-based learning algorithms. Using this result, we introduce the first efficient and exact leave-pair-out cross-validation algorithm for RankRLS, a kernel-based ranking algorithm. If RankRLS is already trained with the whole training set, the computational complexity of the algorithm is $O(m^2)$. Further, if there are d outputs to be learned simultaneously, the computational complexity of performing LPOCV is $O(m^2d)$.

An important special case of ranking is the task of AUC maximization. The traditional approaches for AUC performance evaluation using cross-validation suffer from certain problems and pitfalls, especially in small sample settings as discussed by [Parker et al., 2007, Suominen et al., 2008]. The leave-pair-out cross-validation avoids these problems, while it can be efficiently calculated for the AUC maximizing RLS algorithm as we show in this paper and also for the basic RLS algorithm as can easily be inferred from the results presented in [Pahikkala et al., 2006, An et al., 2007].

Note that the same properties hold also for more general ranking tasks.

Acknowledgments

This work has been supported by the Academy of Finland and Tekes, the Finnish Funding Agency for Technology and Innovation. We would like to thank the anonymous reviewers for their insightful comments.

6. REFERENCES

[An et al., 2007] An, S., Liu, W., and Venkatesh, S. (2007). Fast cross-validation algorithms for least squares support vector machine and kernel ridge regression. *Pattern Recognition*, 40(8):2154–2162.

[Cortes et al., 2007a] Cortes, C., Mohri, M., and Rastogi, A. (2007a). An alternative ranking problem for search engines. In Demetrescu, C., editor, *Proceedings of the 6th Workshop on Experimental Algorithms*, volume 4525 of *Lecture Notes in Computer Science*, pages 1–21. Springer.

[Cortes et al., 2007b] Cortes, C., Mohri, M., and Rastogi, A. (2007b). Magnitude-preserving ranking algorithms. In Ghahramani, Z., editor, *Proceedings of the 24th Annual International Conference on Machine Learning*, pages 169–176. Omnipress.

[Herbrich et al., 1999] Herbrich, R., Graepel, T., and Obermayer, K. (1999). Support vector learning for ordinal regression. In *Proceedings of the Ninth International Conference on Articial Neural Networks*, pages 97–102, London. Institute of Electrical Engineers.

[Horn and Johnson, 1985] Horn, R. and Johnson, C. R. (1985). *Matrix Analysis*. Cambridge University Press, Cambridge.

[Joachims, 2002] Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*, pages 133–142, New York, NY, USA. ACM Press.

[Joachims, 2006] Joachims, T. (2006). Training linear SVMs in linear time. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conferen ce on Knowledge discovery and data mining*, pages 217–226, New York, NY, USA. ACM Press.

[Johnson and Zhang, 2008] Johnson, R. and Zhang, T. (2008). Graph-based semi-supervised learning and spectral kernel design. *IEEE Transactions on Information Theory*, 54(1):275–288.

[Pahikkala et al., 2008] Pahikkala, T., Airola, A., Suominen, H., Boberg, J., and Salakoski, T. (2008). Efficient AUC maximization with regularized least-squares. In Holst, A., Kreuger, P., and Funk, P., editors, *Proceedings of the 10th Scandinavian Conference on Artificial Intelligence (SCAI 2008)*, volume 173 of *Frontiers in Artificial Intelligence and Applications*, pages 76–83. IOS Press, Amsterdam, Netherlands.

[Pahikkala et al., 2006] Pahikkala, T., Boberg, J., and Salakoski, T. (2006). Fast n-fold cross-validation for regularized least-squares. In Honkela, T., Raiko, T., Kortela, J., and Valpola, H., editors, *Proceedings of the Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006)*, pages 83–90, Espoo, Finland. Otamedia.

[Pahikkala et al., 2007] Pahikkala, T., Tsivtsivadze, E., Airola, A., Boberg, J., and Salakoski, T. (2007). Learning to rank with pairwise regularized least-squares. In Joachims, T., Li, H., Liu, T.-Y., and Zhai, C., editors, *SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, pages 27–33.

[Parker et al., 2007] Parker, B. J., Gunter, S., and Bedo, J. (2007). Stratification bias in low signal microarray studies. *BMC Bioinformatics*, 8(326).

[Poggio and Smale, 2003] Poggio, T. and Smale, S. (2003). The mathematics of learning: Dealing with data. *Notices of the American Mathematical Society (AMS)*, 50(5):537–544.

[Rifkin and Lippert, 2007] Rifkin, R. and Lippert, R. (2007). Value regularization and fenchel duality. *Journal of Machine Learning Research*, 8:441–479.

[Rifkin et al., 2003] Rifkin, R., Yeo, G., and Poggio, T. (2003). *Regularized Least-squares Classification*, volume 190 of *NATO Science Series III: Computer and System Sciences*, chapter 7, pages 131–154. IOS Press, Amsterdam.

[Schölkopf et al., 2001] Schölkopf, B., Herbrich, R., and Smola, A. J. (2001). A generalized representer theorem. In Helmbold, D. and Williamson, R., editors, *Proceedings of the 14th Annual Conference on Computational Learning Theory and 5th European Conference on Computational Learning Theory*, pages 416–426, Berlin, Germany. Springer.

[Schölkopf and Smola, 2002] Schölkopf, B. and Smola, A. J. (2002). *Learning with kernels*. MIT Press, Cambridge, MA.

[Shawe-Taylor and Cristianini, 2004] Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge.

[Suominen et al., 2008] Suominen, H., Pahikkala, T., and Salakoski, T. (2008). Critical points in assessing learning performance via cross-validation. In *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR'08)*.

Appendix

We first present the following lemma which is often called the block inverse (see e.g. [Horn and Johnson, 1985]).

Lemma 1. *Let $R \in \mathbb{R}^{m\times m}$ be an invertible matrix, $U \subset \{1,\dots,m\}$, and $\overline{U} = \{1,\dots,m\} \setminus U$ the complement of U. Without losing generality, we can write R as a block matrix*

$$R = \left[\begin{array}{cc} R_{UU} & R_{U\overline{U}} \\ R_{\overline{U}U} & R_{\overline{U}\overline{U}} \end{array} \right]. \tag{15}$$

If $R_{\overline{U}\overline{U}}$ and R_{UU} are invertible, then the inverse matrix of R is

$$R^{-1} = \left[\begin{array}{cc} (R^{-1})_{UU} & (R^{-1})_{U\overline{U}} \\ (R^{-1})_{\overline{U}U} & (R^{-1})_{\overline{U}\overline{U}} \end{array} \right],$$

where

$$\begin{aligned}
(R^{-1})_{UU} &= S^{-1}, && (16)\\
(R^{-1})_{U\overline{U}} &= -S^{-1}R_{U\overline{U}}(R_{\overline{U}\overline{U}})^{-1}, && (17)\\
(R^{-1})_{\overline{U}U} &= -(R_{\overline{U}\overline{U}})^{-1}R_{\overline{U}U}S^{-1} &&\\
(R^{-1})_{\overline{U}\overline{U}} &= (R_{\overline{U}\overline{U}})^{-1} + (R_{\overline{U}\overline{U}})^{-1}R_{\overline{U}U}S^{-1}R_{U\overline{U}}(R_{\overline{U}\overline{U}})^{-1}, \text{ and} &&\\
S &= R_{UU} - R_{U\overline{U}}(R_{\overline{U}\overline{U}})^{-1}R_{\overline{U}U}. && (18)
\end{aligned}$$

The following result is known as the matrix inversion lemma or Sherman-Morrison-Woodbury formula (see e.g. [Horn and Johnson, 1985]).

Lemma 2. *If A, D, and $D - CA^{-1}B$ are invertible, then*

$$(A - BD^{-1}C)^{-1} = A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1}.$$

Proof of Theorem 1 Let $G = K^{-1}$. We start by splitting the infimum of the objective into two parts

$$\begin{aligned}
\inf_{\vec{v}\in\mathbb{R}^m} \left\{ l(\vec{v}_{\overline{U}}, Y_{\overline{U}}) + \lambda \vec{v}^{\mathrm{T}} G \vec{v} \right\} &= \inf_{\vec{p}\in\mathbb{R}^{|\overline{U}|}} \left\{ \inf_{\vec{h}\in\mathbb{R}^{|U|}} \left\{ l(\vec{p}, Y_{\overline{U}}) + \lambda \begin{pmatrix} \vec{h} \\ \vec{p} \end{pmatrix}^{\mathrm{T}} G \begin{pmatrix} \vec{h} \\ \vec{p} \end{pmatrix} \right\} \right\} \\
&= \inf_{\vec{p}\in\mathbb{R}^{|\overline{U}|}} \left\{ l(\vec{p}, Y_{\overline{U}}) + \lambda \inf_{\vec{h}\in\mathbb{R}^{|U|}} \left\{ \begin{pmatrix} \vec{h} \\ \vec{p} \end{pmatrix}^{\mathrm{T}} G \begin{pmatrix} \vec{h} \\ \vec{p} \end{pmatrix} \right\} \right\} \qquad (19)
\end{aligned}$$

and continue by considering the minimizer of the inner one in (19) containing only the regularizer. Let $\vec{p} \in \mathbb{R}^{|\overline{U}|}$ be an arbitrary vector and let

$$\vec{h^*} = \operatorname*{arginf}_{\vec{h}\in\mathbb{R}^{|U|}} \left\{ \begin{pmatrix} \vec{h} \\ \vec{p} \end{pmatrix}^{\mathrm{T}} G \begin{pmatrix} \vec{h} \\ \vec{p} \end{pmatrix} \right\}. \tag{20}$$

We can solve $\vec{h^*}$ by taking the derivative with respect to $\vec{h}$:

$$\begin{aligned}
\frac{\partial}{\partial \vec{h}} \begin{pmatrix} \vec{h} \\ \vec{p} \end{pmatrix}^{\mathrm{T}} G \begin{pmatrix} \vec{h} \\ \vec{p} \end{pmatrix} &= \frac{\partial}{\partial \vec{h}} \begin{pmatrix} \vec{h} \\ \vec{p} \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} G_{UU} & G_{U\overline{U}} \\ G_{\overline{U}U} & G_{\overline{U}\overline{U}} \end{pmatrix} \begin{pmatrix} \vec{h} \\ \vec{p} \end{pmatrix} \\
&= \frac{\partial}{\partial \vec{h}} \left(\vec{h}^{\mathrm{T}} G_{UU} \vec{h} + \vec{h}^{\mathrm{T}} G_{U\overline{U}} \vec{p} + \vec{p}^{\mathrm{T}} G_{\overline{U}U} \vec{h} + \vec{p}^{\mathrm{T}} G_{\overline{U}\overline{U}} \vec{p} \right) \\
&= 2G_{UU}\vec{h} + 2G_{U\overline{U}}\vec{p}.
\end{aligned}$$

By setting it to zero and solving with respect to $\vec{h}$, we get

$$\begin{aligned}
\vec{h^*} &= -(G_{UU})^{-1}G_{U\overline{U}}\vec{p} \qquad (21)\\
&= -(G_{UU})^{-1}(-G_{UU}K_{U\overline{U}}(K_{\overline{U}\overline{U}})^{-1})\vec{p} \\
&= K_{U\overline{U}}(K_{\overline{U}\overline{U}})^{-1}\vec{p},
\end{aligned}$$

where $G_{U\overline{U}} = -G_{UU}K_{U\overline{U}}(K_{\overline{U}\overline{U}})^{-1}$ follows from the formula of block inverse. By substituting (21) into (20), we get

$$\begin{array}{ll} & \left(\begin{array}{c} -(G_{UU})^{-1}G_{U\overline{U}}\vec{p} \\ \vec{p} \end{array}\right)^{\mathrm{T}} G \left(\begin{array}{c} -(G_{UU})^{-1}G_{U\overline{U}}\vec{p} \\ \vec{p} \end{array}\right) \\ = & \vec{p}^{\mathrm{T}}G_{\overline{U}U}(G_{UU})^{-1}G_{UU}(G_{UU})^{-1}G_{U\overline{U}}\vec{p} \\ & -\vec{p}^{\mathrm{T}}G_{\overline{U}U}(G_{UU})^{-1}G_{U\overline{U}}\vec{p} \\ & -\vec{p}^{\mathrm{T}}G_{\overline{U}U}(G_{UU})^{-1}G_{U\overline{U}}\vec{p} \\ & +\vec{p}^{\mathrm{T}}G_{\overline{U}\overline{U}}\vec{p} \\ = & \vec{p}^{\mathrm{T}}\Big(G_{\overline{U}\overline{U}} - G_{\overline{U}U}(G_{UU})^{-1}G_{U\overline{U}}\Big)\vec{p} \\ = & \vec{p}^{\mathrm{T}}(K_{\overline{U}\overline{U}})^{-1}\vec{p}, \end{array}$$

where the last equality is due to (16) and (18). Therefore,

$$\begin{array}{rcl} \vec{p^{*}} & = & \underset{\vec{p}\in\mathbb{R}^{|\overline{U}|}}{\operatorname{arginf}} \left\{ l(\vec{p}, Y_{\overline{U}}) + \lambda \inf_{\vec{h}\in\mathbb{R}^{|U|}} \left\{ \left(\begin{array}{c} \vec{h} \\ \vec{p} \end{array}\right)^{\mathrm{T}} G \left(\begin{array}{c} \vec{h} \\ \vec{p} \end{array}\right) \right\} \right\} \\ & = & \underset{\vec{p}\in\mathbb{R}^{|\overline{U}|}}{\operatorname{arginf}} \left\{ l(\vec{p}, Y_{\overline{U}}) + \lambda\vec{p}^{\mathrm{T}}(K_{\overline{U}\overline{U}})^{-1}\vec{p} \right\} \end{array} \quad (22)$$

and

$$\vec{h^{*}} = K_{U\overline{U}}(K_{\overline{U}\overline{U}})^{-1}\vec{p^{*}}, \quad (23)$$

Analogously to (7), (22) is a vector consisting of predictions of the learning method for its own labeled training inputs, that is,

$$\vec{p^{*}} = f_{\overline{U}}(X_{\overline{U}}). \quad (24)$$

Moreover, similarly to (9), $(K_{\overline{U}\overline{U}})^{-1}\vec{p^{*}}$ contains the coefficients determining the function obtained by training with the examples indexed by $\overline{U}$. Therefore, we observe from (2) and (23) that

$$\vec{h^{*}} = K_{U\overline{U}}(K_{\overline{U}\overline{U}})^{-1}\vec{p^{*}} = f_{\overline{U}}(X_U), \quad (25)$$

that is, the vector $\vec{h^{*}}$ consists of the predictions for the data points indexed by U made by a learner trained with the data points indexed by $\overline{U}$. Finally, by combining (22), (23), (24), and (25), we get

$$\underset{\vec{v}\in\mathbb{R}^{m}}{\operatorname{arginf}} \left\{ l(\vec{v}_{\overline{U}}, Y_{\overline{U}}) + \lambda\vec{v}^{\mathrm{T}}G\vec{v} \right\} = \left(\begin{array}{c} \vec{h^{*}} \\ \vec{p^{*}} \end{array}\right) = f_{\overline{U}}(X).$$

□

Proof of Theorem 2 Let $U = \{h_1, h_2\}$ be the index set containing the indices h_1 and h_2 of the hold-out training examples. We start by considering the predictions of RankRLS for the training examples:

$$f(X) \quad = \quad (L + \lambda K^{-1})^{-1}LY.$$

According to Theorem 1, the predictions of RankRLS are independent of such examples for which the cost is not calculated. Thus, we can remove the effect of the hold-out examples by excluding them from the cost as follows:

$$\begin{array}{rcl} l_{\overline{U}}(\vec{p}, Y) & = & \frac{1}{2}\sum_{i,j\in\overline{U}} ((y_i - y_j) - (p_i - p_j))^2 \\ & = & m\sum_{i\in\overline{U}}(y_i - p_i)^2 - \sum_{i,j\in\overline{U}}(y_i - p_i)(y_j - p_j) \\ & = & (Y - \vec{p})^{\mathrm{T}}\widetilde{L}(Y - \vec{p}), \end{array}$$

where the modified Laplacian matrix $\widetilde{L}$ is defined as

$$\widetilde{L}_{i,j} = \begin{cases} -1 & \text{if } i \neq j \text{ and } i, j \in \overline{U} \\ m - 3 & \text{if } i = j \text{ and } i \in \overline{U} \\ 0 & \text{otherwise} \end{cases}.$$

The kernel matrix can be used as such, as it appears only in the regularizer. The predictions for the hold-out data points can be calculated from

$$f_{\overline{U}}(X_U) \quad = \quad I_U(\widetilde{L} + \lambda K^{-1})^{-1}\widetilde{L}Y, \quad (26)$$

We continue by observing that we can also write

$$\widetilde{L} = \widetilde{D} - BB^{\mathrm{T}}, \tag{27}$$

where $B \in \mathbb{R}^{m\times 3}$ is a matrix whose values are determined by

$$B_{i,j} = \begin{cases} 1 & \text{if } i \in \overline{U} \text{ and } j = 1 \\ \sqrt{m-2} & \text{if } i = h_1 \text{ and } j = 2 \\ \sqrt{m-2} & \text{if } i = h_2 \text{ and } j = 3 \\ 0 & \text{otherwise} \end{cases},$$

By substituting (27) into (26) and by using the Sherman-Morrison-Woodbury formula, we can write

$$\begin{aligned} f_{\overline{U}}(X_U) &= I_U(\widetilde{D} - BB^{\mathrm{T}} + \lambda K^{-1})^{-1}\widetilde{L}Y \\ &= I_U(Q^{-1} - BB^{\mathrm{T}})^{-1}\widetilde{L}Y \\ &= I_U(Q - QB(-I + B^{\mathrm{T}}QB)^{-1}B^{\mathrm{T}}Q)\widetilde{L}Y \\ &= (Q\widetilde{L}Y)_U - (QB)_U(-I + B^{\mathrm{T}}QB)^{-1}B^{\mathrm{T}}Q\widetilde{L}Y. \end{aligned} \tag{28}$$

Let

$$R = \begin{pmatrix} -1 & \sqrt{m-2} & 0 \\ -1 & 0 & \sqrt{m-2} \end{pmatrix},$$

that is,

$$R = B_U - C_U.$$

To compute (28), we consider the following equations:

$$\begin{aligned} B^{\mathrm{T}}QB &= C^{\mathrm{T}}QC + R^{\mathrm{T}}(QC)_U + (R^{\mathrm{T}}(QC)_U)^{\mathrm{T}} + R^{\mathrm{T}}Q_{UU}R \\ B^{\mathrm{T}}Y &= C^{\mathrm{T}}Y + R^{\mathrm{T}}Y_U \\ B^{\mathrm{T}}Q\widetilde{L}Y &= C^{\mathrm{T}}Q\widetilde{D}Y + R^{\mathrm{T}}(Q\widetilde{D}Y)_U - B^{\mathrm{T}}QBB^{\mathrm{T}}Y \\ (QB)_U &= (QC)_U + Q_{UU}R \\ (Q\widetilde{L}Y)_U &= (Q\widetilde{D}Y)_U - (QB)_U B^{\mathrm{T}}Y. \end{aligned}$$

The first equation can be proven as follows:

$$\begin{aligned} B^{\mathrm{T}}QB &= \begin{pmatrix} B_U \\ B_{\overline{U}} \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} Q_{UU} & Q_{U\overline{U}} \\ Q_{\overline{U}U} & Q_{\overline{UU}} \end{pmatrix} \begin{pmatrix} B_U \\ B_{\overline{U}} \end{pmatrix} \\ &= \begin{pmatrix} C_U + R \\ C_{\overline{U}} \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} Q_{UU} & Q_{U\overline{U}} \\ Q_{\overline{U}U} & Q_{\overline{UU}} \end{pmatrix} \begin{pmatrix} C_U + R \\ C_{\overline{U}} \end{pmatrix} \\ &= C^{\mathrm{T}}QC + R^{\mathrm{T}}(QC)_U + (R^{\mathrm{T}}(QC)_U)^{\mathrm{T}} + R^{\mathrm{T}}Q_{UU}R. \end{aligned}$$

The other equations can be shown to hold analogously.

Given that the matrices (14) are already calculated and stored in memory, the right hand sides of the equations can be computed in a constant time. Therefore, by substituting these into (28), we conclude that the hold-out predictions for a pair of examples can be calculated in a constant time. After the matrices (14) are calculated once at the time the RankRLS learner is trained, the overall LPOCV can be performed in $O(m^2)$ time. Further, if we consider the number of subproblems to be a variable d instead of a constant, that is, Y is a $m \times d$-matrix instead of a single column vector, then the computational complexities are multiplied with d. $\square$