Fast Multiple Kernel Learning With Multiplicative Weight Updates

John Moeller moeller@...
Parasaran Raman praman@...
Avishek Saha avishek@...
Suresh Venkatasubramanian suresh@cs.utah.edu

Abstract

We present a fast algorithm for multiple kernel learning (MKL). Our matrix multiplicative weight update (MWUMKL) algorithm is based on a well-known QCQP formulation [5]. In addition, we propose a novel fast matrix exponentiation routine for QCQPs which might be of independent interest. Our method avoids the use of commercial nonlinear solvers and scales efficiently to large data sets. [1]

1 Introduction

Kernel methods have been extremely successful in a wide variety of machine learning applications [8, 3, 5, 6, 9]. The success of these methods relies on an appropriate choice of kernel (or kernels) and there has been extensive research on *learning a combination of multiple kernels* which outperform [5] algorithms that operate with a single kernel. Earliest approaches on multiple kernel learning assigned equal (or weighted) preference to kernels and simply used the weighted sum of kernel functions [6]. Lanckriet et al. [5] proposed simultaneously training an SVM and learning a convex combination of kernel functions, by framing MKL as an optimization over positive semi-definite kernel matrices. Bach et al. [3] smoothed the above formulation using a block-norm regularization method. Follow up work employed alternating optimization based techniques that alternate between updating the classifier parameters and the kernel weights (while sacrificing guarantees on accuracy). Examples of such methods include Semi-Infinite Linear Programming [9], SimpleMKL [7], LevelMKL [11] and GroupMKL [12].

In this work, we propose a fast MKL algorithm (MWUMKL) that (a) does not require commercial solvers (b) does not make explicit calls to SVM libraries (unlike alternating optimization based methods), and (c) provably converges in a fixed number of iterations to a solution with guaranteed accuracy (with respect to the underlying optimization). We focus on the original QCQP formulation of the MKL problem ([5]) and make it scalable by using the matrix multiplicative weight update (MMWU) [2] method to solve the underlying QCQP. Taking advantage of the structure of the MKL problem, we make simplifications to the original formulation, exploit a novel fast routine for *exact* matrix exponentiation, and perform a number of optimizations to the primal-dual core of the MMWU method. As we empirically demonstrate, all of these contributions taken together significantly improve the running time of our method, allowing to scale to input sizes well beyond prior methods with no significant loss in quality.

2 Background

Let $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \in \mathbb{R}^{n \times d}$ denote training samples, $\mathbf{y} = (y_1, y_2, \ldots, y_n) \in \{-1, +1\}^n$ denote the corresponding binary class labels and $\mathbf{w}$ denote the weight vector to be learned. The problem of

[1] This work was sponsored by NSF grant CCF-0953066. All the authors gratefully acknowledge the support of the grants. Any opinions, findings, and conclusion or recommendation expressed in this material are those of the author(s) and do not necessarily reflect the view of the funding agencies or the U.S. government.

multiple kernel learning can be cast as the problem of minimizing $\frac{1}{2}\|\mathbf{w}\|^2$ such that $y_j(\langle \mathbf{w}, \Phi(\mathbf{x}_j)\rangle + b) \geq 1$ for all j where Φ is the feature map associated with the (unknown) kernel κ. With the constraint that κ can be written as $\kappa = \sum_{i=1}^m \mu_i \kappa_i$ (with a trace regularizer) the dual problem takes the following form [5]:

$$\min_{\mathbf{K}} \max_{\alpha} \quad 2\alpha^\top \mathbf{1} - \alpha^\top \mathbf{G}\alpha \quad \text{s.t.} \quad \mathbf{K} = \sum_{i=1}^m \mu_i \mathbf{K}_i, \quad \operatorname{tr}(\mathbf{K}) = c, \quad \mathbf{K} \succeq 0, \quad \mu \geq 0$$

where $\mathbf{K}_i$ is the Gram matrix for $\mathbf{X}$ associated with the kernel function $\kappa_i(\cdot,\cdot)$, $\mathbf{G}_i = \operatorname{diag}(\mathbf{y})\mathbf{K}_i \operatorname{diag}(\mathbf{y})$ and $\mathbf{G} = \sum_{i=1}^m \mu_i \mathbf{G}_i$. The kernel $\mathbf{K}$ optimizing this expression can be found by solving the following convex optimization problem [5, Theorem 20]:

$$\max_{\alpha, s} \quad 2\alpha^\top \mathbf{1} - cs \quad \text{s.t.} \quad s \geq \frac{1}{r_i}\alpha^\top \mathbf{G}_i \alpha, \quad \alpha^\top \mathbf{y} = 0, \quad \alpha \geq 0 \tag{2.1}$$

where $r \in \mathbb{R}^m$ and $\operatorname{tr}(\mathbf{K}_i) = r_i$. This program is an example of a class of convex programs called *quadratically constrained quadratic programs* (QCQPs). Since the class of QCQPs is a subset of SOCPs (*second order cone program*), Lanckriet et al. [5] proposed solving the above program with efficient SOCP solvers such as Mosek [1] or SeDuMi [10].

3 Algorithm

Our algorithm is a matrix multiplicative-weight-update method [2] applied to the optimization formulated in (2.1). Such a method works as follows. Firstly, it guesses a value of the objective solution. Next, it runs a primal-dual update scheme in order to determine the variable settings that yield this value. The primal-dual scheme uses an oracle call (typically a linear program) to determine whether the current variable settings yield a feasible solution. If so, the algorithm then tries a smaller value of the objective and repeats (as in binary search). If not, the primal-dual update continues by updating the relevant variables and trying again. This last update is performed via a matrix exponentiation (for more details, see [2]).

We now describe how we implement and improve upon the three core steps of the process by exploiting structure in the MKL SDP. Algorithm 1 describes the entire procedure. For the sake of brevity, we have compressed and abstracted out key details in this extended abstract.

From an QCQP to an SDP The MMWU framework is usually applied to an SDP in a canonical form, so the first step is rewriting (2.1) in SDP form. This yields the expression

$$\begin{aligned} &\min_{\alpha, s} \quad cs - 2\alpha^\top \mathbf{1} \\ \forall i \in [1..m] \quad &\mathbf{Q}_i(\alpha) = \begin{pmatrix} \mathbf{I}_n & \mathbf{A}_i\alpha \\ (\mathbf{A}_i\alpha)^\top & cs \end{pmatrix} \succeq \mathbf{0} \\ &\alpha^\top \mathbf{y} = 0, \quad \alpha \geq \mathbf{0}, \end{aligned}$$

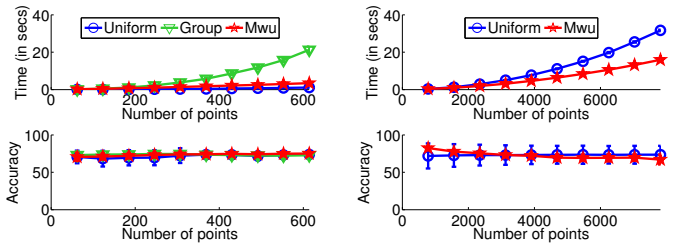
where $A_i^\top A_i = \frac{c}{r_i}\mathbf{G}_i$.

This transformation allows us to apply the MMWU framework [2].

Matrix Exponentiation. A critical step in the MWU process is computing $(1 - \varepsilon/2\rho)^{\sum_{t=1}^\tau \mathbf{M}^{(t)}} = e^{-\varepsilon' \sum_{t=1}^\tau \mathbf{M}^{(t)}}$, where $\varepsilon' = -\ln(1 - \varepsilon/2\rho)$ (see Algorithm 1). For MKL, $\mathbf{M}^{(t)}$ is a block-diagonal matrix, with m blocks of the form $\mathbf{M}_i^{(t)} = \frac{1}{2\rho}(\mathbf{Q}_i(\alpha^{(t)}) + \rho \mathbf{I}_{n+1})$. The matrix $e^{-\varepsilon' \sum_{t=1}^\tau \mathbf{M}^{(t)}}$ will also be block-diagonal, with m blocks $e^{-\varepsilon' \sum_{t=1}^\tau \mathbf{M}_i^{(t)}}$. Therefore we just need a way to exponentiate $-\varepsilon \sum_{t=1}^\tau \mathbf{M}_i^{(t)} = -\frac{\varepsilon'}{2\rho}(\mathbf{Q}_i(\sum_{t=1}^\tau \alpha^{(t)}) + \rho t \mathbf{I}_{n+1})$ for each i.

Algorithm 1 MWU-MKL

Require: $\mathbf{g}^{(1)} = \mathbf{0}$;
  ρ, the width of ORACLE;
  ε, the desired approximation error
  Set $\varepsilon' = -\ln(1 - \frac{\varepsilon}{2\rho})$
  Set $\tau = \frac{8\rho^2}{\varepsilon^2}\ln(n)$
  repeat $\{\tau$ times$\}$
    Set $\alpha^{(t)} = \text{ORACLE}(\mathbf{y}, \mathbf{g}^{(t)})$
    Update $\alpha = \alpha + \alpha^{(t)}$
    if ORACLE failed **then**
      return
    Set $\mathbf{M}_i^{(t)} = \frac{1}{2\rho}\left(\mathbf{Q}_i(\alpha^{(t)}) + \rho \mathbf{I}_{n+1}\right)$
    Set $\mathbf{W}_i^{(t)} = e^{-\varepsilon' \sum_{t=1}^\tau \mathbf{M}_i^{(t)}}$
    Set $\mathbf{L}_i^{(t+1)} = \mathbf{W}_i^{(t)} / \operatorname{Tr}(\mathbf{W}_i^{(t)})$
    Compute $\mathbf{g}^{(t+1)}$ from $\mathbf{L}^{(t+1)}$, $\{\mathbf{G}_i\}$, and α
  until $t = \tau$
  return $\frac{1}{\tau}\alpha, \mathbf{L}^{(t+1)}$



(a) *Pima* with $m = 108$ kernels (b) *Adult* with $m = 12$ kernels

Figure 1: Results for *Pima* ($n = 768$, $d = 8$) and *Adult* ($n = 9768$, $d = 123$).

In general, matrix exponentiation is expensive, and Arora and Kale [2] suggest ways to approximate this computation. However, the matrix $\mathbf{M}$ depends primarily on the $\mathbf{Q}_i$, which have a very special block-diagonal structure. This allows us to compute the matrix exponent *in closed form*, replacing the (expensive) approximation step by a simple linear time procedure with a small memory footprint.

Computing The Oracle. In the MMWU formulation of an SDP, an oracle call generates dual updates from a given primal "guess". In general, the oracle solves a linear program: repeated invocations of this linear program are viewed as more efficient in principle than solving the (more complex) SDP.

In the context of MKL, the "primal" variable is the matrix $\mathbf{L}$ computed from the matrix exponentiation process. Given $\mathbf{L}$, the oracle then generates a new update for the dual variables α[2]. Once again, we take advantage of the block structure of the matrix L to design an oracle that reduces to solving a *single* linear inequality by inspection. If the inequality cannot be satisfied, the oracle returns FAIL. This is a significant improvement over the general linear program that is typically required.

Eliminating Binary Search. A final optimization is to eliminate the generic binary search used to "guess" the optimal solution to the underlying SDP. By careful use of the KKT conditions and the fact that the objective function in (2.1) is linear, we can rescale the problem to fix the optimal value (all that remains is to find the coefficients α, which are used to generate the desired weights for the kernels).

Extracting the solution from the MWU Finally, we use the KKT conditions and the α to extract the coefficients μ for the kernels. The transformation is a simple function of $\mathbf{L}$ and α, and can be computed efficiently.

Putting it all together Algorithm 1 summarizes the discussion in this section. The parameter ε is the error in approximating the objective function. We set the actual value of ε via cross-validation. The parameter ρ is the *width* of ORACLE, a parameter that indicates how much the solution can vary at each step. ρ is equal to the maximum absolute value of the eigenvalues of $\mathbf{Q}_i(\alpha^{(t)})$, for any i [2]. The total running time is $O\left(cmn\ln(n)\frac{1}{\varepsilon^2}\right)$.

4 Results

There is a natural trade-off in MKL methods between classification accuracy and measured running time. The quest for scalability asks for the largest sizes (number of points, number of kernels) we can perform MKL on without sacrificing accuracy in any significant way. We present two prototypical results here (our full version has results for more data sets and more methods). In the small data regime ($n \leq 2000$), we show that the accuracy we obtain is comparable to the best of the related methods (usually UNIFORM and GROUPMKL). When we move to larger n ($n \geq 10,000$)

[2]The MKL formulation introduces additional linear constraints on α that are not strictly part of the original SDP. It is easy to show however that these can be folded into the oracle computation.

most methods (other than UNIFORM) suffer severe degradation in running time, to the point where comparisons are futile. Comparing against UNIFORM, which fixes uniform weights on all kernels and runs an SVM, our method has comparable accuracy while actually performing faster. These results are summarized in Figure 1. Our results are robust to different choices of data sets, number of kernels used, as well as error parameters.

A major bottleneck limiting further scaling of our work is the memory utilization of the method. As currently stated, it requires all kernel matrices to be stored in memory. For m kernels on n points, this yields a memory requirement of $\Omega(mn^2)$. By computing columns of the kernel matrices on demand, we can reduce the memory footprint to $O(mn)$, improving scalability without affecting solution quality. We are currently testing this implementation: preliminary results suggest that we can indeed scale well beyond tens of thousands of points, as well as many kernels. We note that recent work on scalable MKL [4] indicates the ability to deal with millions of kernels, but in effect also has a memory footprint of $\Omega(mn)$ which limits their approach to *either* many kernels *or* many points, but not both. Since they do not provide accuracy numbers, a direct head-to-head comparison is difficult to make.

5 Discussion

Our approach suggests that the matrix multiplicative weight update, while expensive in general, can be adapted to provide combinatorial algorithms for certain nonlinear optimizations that scale well. We expect that this approach will prove to be useful for other QCQP-based optimizations such as distance metric learning as well. We note that the *matrix* multiplicative weight update method generalizes the traditional approach in that each "expert" that is updated is really a linear combination of experts. In effect, the method seems to identify correlations between the original experts and transforms the space so that the new "experts" are independent of each other and can be updated independently. It would be interesting to see if this insight can be made explicit, for example in multi-task learning.

References

[1] E. D. Andersen and K. D. Andersen. *The MOSEK interior point optimization for linear programming: an implementation of the homogeneous algorithm*, pages 197–232. Kluwer Academic Publishers, 1999.

[2] S. Arora and S. Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC*, San Diego, California, USA, 2007.

[3] F. R. Bach, G. R. G. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *ICML*, Banff, Canada, 2004.

[4] A. Jain, S. V. N. Vishwanathan, and M. Varma. Spf-gmkl: generalized multiple kernel learning with a million kernels. In Q. Yang, D. Agarwal, and J. Pei, editors, *KDD*, pages 750–758. ACM, 2012. ISBN 978-1-4503-1462-6.

[5] G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan. Learning the kernel matrix with semidefinite programming. *JMLR*, 5:27–72, December 2004.

[6] P. Pavlidis, J. Cai, J. Weston, and W. N. Grundy. Wn: Gene functional classification from heterogeneous data. In *In Proceedings of the Fifth Annual International Conference on Computational Biology*, 2001.

[7] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. More efficiency in multiple kernel learning. In *ICML*, Corvalis, USA, 2007.

[8] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001. ISBN 0262194759.

[9] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *JMLR*, 7: 1531–1565, December 2006.

[10] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999.

[11] Z. Xu, R. Jin, I. King, and M. R. Lyu. An extended level method for efficient multiple kernel learning. In *NIPS*, Vancouver, Canada, 2008.

[12] Z. Xu, R. Jin, H. Yang, I. King, and M. R. Lyu. Simple and efficient multiple kernel learning by group lasso. In *ICML*, Haifa, Israel, 2010.