Second order optimization of kernel parameters

Olivier Chapelle
Yahoo! Research
chap@yahoo-inc.com

Alain Rakotomamonjy
LITIS EA4108, Univ. Rouen
alain.rakoto@insa-rouen.fr

1 Introduction

In kernel methods such as SVMs, the data representation, implicitly chosen through the so-called kernel $K(\mathbf{x}, \mathbf{x}')$, strongly influences the performances. Recent applications [3] and developments based on SVMs have shown that using multiple kernels instead of a single one can enhance interpretability of the decision function and improve classifier performance. In such cases, a common approach is to consider that the kernel $K(\mathbf{x}, \mathbf{x}')$ is actually a convex linear combination of other *basis* kernels:

$$K(\mathbf{x}, \mathbf{x}') = \sum_{m=1}^{M} d_m K^m(\mathbf{x}, \mathbf{x}'), \text{ with } d_m \geq 0, \sum_m d_m = 1,$$

where M is the total number of kernels. Within this framework, the problem of data representation through the kernel is then transferred to the choice of weights d_m. Learning both the coefficients α_i and the weights d_m in a single optimization problem is known as the multiple kernel learning (MKL) problem [3]. Recently different methods for solving this problem have been proposed by [5] and [4]. The advantage of these latter formulations is that they solve the MKL problem by iteratively solving a classical SVM problem with a single kernel and updating the weights d_m either through linear programming or by a gradient descent step. Although the gradient descent algorithm, known as SimpleMKL, has been shown to be more efficient than the SILP algorithm, it still suffers from some weaknesses such as slow convergence near optimality.

In this paper, we investigate the use of second order optimization approaches for solving MKL problem. We show that the hessian of the MKL can be computed and this information can be used to compute a better descent direction and the gradient one. We then empirically show that our new approaches outperforms SimpleMKL in terms of computational efficiency.

2 Multiple Kernel Learning

We now discuss the MKL objective function to be optimized. After the introduction of the standard objective function, we present a variation where we consider the d_m as hyperparameters. This leads, in our opinion, to a more principled way of optimizing d_m.

Standard approach Consider a SVM with a kernel K.[1] The following objective function is maximized:

$$\Omega(K) \;:=\; \max_{\alpha_i} \; \sum \alpha_i y_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$
$$\text{under constraint } \; 0 \leq \alpha_i y_i \leq C \;\text{ and }\; \sum \alpha_i = 0.$$

Since finding the maximum margin solution seems to give good empirical results, it has been proposed to extend this idea for MKL [3]: find the kernel that maximizes the margin or equivalently

$$\min_{d_m} \; \Omega\left(\sum d_m K^m\right)$$

[1] Depending on the context, K might refer to the kernel function or the kernel matrix

The problem in this approach is that the SVM objective function has been derived for finding an hyperplane for a *given* kernel, not for learning the kernel matrix. It is not obvious why the K minimizing Ω should be a good kernel.

An illustration of the problem is as follows n the hard margin case: since $\Omega(dK) = \Omega(K)/d$ for any constant d, Ω can be trivially minimized by multiplying K by an arbitrary large constant. This is usually fixed by adding the constraint $\sum d_m \leq 1$. But is the L_1 norm on $\mathbf{d}$ necessarily the most appropriate norm?

Hyperparameter view A more principle approach is to consider the d_m as *hyperparameters* and to tune them on a *model selection* criterion [1]. A convenient criterion is a bound on the generalization error [1], $T(K)\Omega(K)$, where $T(K)$ is the re-centered trace, $T(K) = \sum_i K(\mathbf{x}_i, \mathbf{x}_i) - \frac{1}{n}\sum_{i,j} K(\mathbf{x}_i, \mathbf{x}_j)$. Because $\Omega(dK) = \Omega(K)/d$, this is equivalent to minimize $\Omega(K)$ with constant $T(K)$, or

$$\min_{d_m} \ \Omega\left(\sum d_m K^m\right),$$
$$\text{under constraint} \quad \sum d_m T(K^m) = 1 \ \text{ and } \ d_m \geq 0. \tag{1}$$

It is noteworthy that the linear constraint on d_m appears naturally. Also the optimization turns out be identical to the standard approach if $T(K^m) = 1, \ \forall m$. For this reason, we propose to first center in feature space the kernel matrices and then normalize them to trace 1. In this way we have $T(K^m) = 1$ and we can use the same objective function as before.

3 Optimization

As first pointed out in [4], there is no need of expensive techniques such as SDP [3] or SILP [5] for solving the MKL problem. Indeed, a simple gradient descent algorithm on the convex function

$$J(\mathbf{d}) := \Omega\left(\sum d_m K^m\right)$$

converges faster to the solution and is also simpler to implement. We extend here this idea by incorporating second order information on J. For a given $\mathbf{d}$, let $\alpha^\star$ be the SVM solution and sv denote the set of *free* support vectors, i.e. the points for which $0 < \alpha_i < C$. The gradient is given by [2, 4],

$$g_m := \frac{\partial J}{\partial d_m} = -\frac{1}{2}\sum_{i,j} \alpha_i^\star \alpha_j^\star K^m(\mathbf{x}_i, \mathbf{x}_j). \tag{2}$$

To obtain second order information, we need differentiate (2) and in particular compute $\frac{\partial \alpha^\star}{\partial d_m}$. First note that all the free support vectors lie on the margin, that is:

$$K_{\mathsf{sv},:}\alpha^\star + b = Y_{\mathsf{sv}},$$

where $K := \sum d_m K_m$, $K_{\mathsf{sv},:}$ is the submatrix of K where only the rows corresponding to sv have been kept and Y_{sv} is the target vector for the support vectors. Differentiating with respect to d_m, we have,

$$K^m_{\mathsf{sv},:}\alpha^\star + K_{\mathsf{sv},:}\frac{\partial \alpha^\star}{\partial d_m} + \frac{\partial b}{\partial d_m} = 0.$$

If $i \notin \mathsf{sv}$, $\frac{\partial \alpha_i^\star}{\partial d_m} = 0$ and for the support vectors, we have the following linear system:

$$\underbrace{\begin{pmatrix} K_{\mathsf{sv},\mathsf{sv}} & \mathbf{1} \\ \mathbf{1}^\top & 0 \end{pmatrix}}_{:=A} \begin{pmatrix} \frac{\partial \alpha^\star}{\partial d_m} \frac{\partial b}{\partial d_m} \end{pmatrix}^\top = \begin{pmatrix} K^m_{\mathsf{sv},:}\alpha^\star \\ 0 \end{pmatrix}.$$

The last row comes from the constraint $\sum \alpha_i = 0$. Defining $\bar{A}$ as the inverse of A with the last row and last column removed, we have

$$\frac{\partial \alpha^\star}{\partial d_m} = -\bar{A} K^m_{\mathsf{sv},\mathsf{sv}} \alpha_{\mathsf{sv}}.$$

Let $Q := [\mathbf{q}_1 \dots \mathbf{q}_M]$ with $\mathbf{q}_m := K^m_{\mathsf{sv},\mathsf{sv}} \alpha^\star_{\mathsf{sv}}$. Then the Hessian is:

$$H = Q^\top \bar{A} Q \succeq 0. \tag{3}$$

Note that depending on the SVM solver used, $\bar{A}$ might already be available at the end of the SVM training. Indeed $\alpha^\star$ can be found as the solution of a linear system involving A.

The step direction s is a constrained Newton step found by minimizing the quadratic problem:

$$\min \ \frac{1}{2}\mathbf{s}^\top H \mathbf{s} + \mathbf{s}^\top \mathbf{g}, \tag{4}$$

$$\text{under constraints} \quad \sum s_m T(K^m) = 0 \ \text{ and } \ \mathbf{s} + \mathbf{d} \geq 0.$$

This quadratic form corresponds to the second order expansion of J, while the constraints ensure that any solution on the segment $[\mathbf{d}, \mathbf{d}+\mathbf{s}]$ satisfies the original constraints (1). Finally backtracking is performed in case $J(\mathbf{d}+\mathbf{s}) \geq J(\mathbf{d})$.

The complexity for training the SVM is $O(n n_{\mathsf{sv}} + n^3_{\mathsf{sv}})$, while the complexity for computing H is $O(n^3_{\mathsf{sv}} + M n^2_{\mathsf{sv}})$ and solving the QP problem (4) is $O(M^3)$. So if the number of kernels M is not larger than the number of support vectors n_{sv}, computing the search direction $\mathbf{s}$ is not expensive compared to the SVM trainings. The optimum is usually found in about 10 steps.

4 Numerical results

We have conducted some numerical experiments on UCI datasets in order to evaluate the efficiency of second-order approach compared to SimpleMKL which uses gradient descent. The experimental setting is the same as the one used in [4]. We have considered the same UCI datasets and the same way of building the set of base kernels. The sole difference is that in this work, kernels are centered and normalized to unit trace. For comparing algorithms efficiency, one should consider the same stopping criterion; here, we stopped the algorithms when the relative duality gap is lower than 0.01. The results presented below are averaged over 20 trials of training example random splits and for $C = 100$.

Figure 1 plots the computational time, number of SVM evaluations and number of gradient computations for SimpleMKL, HessianMKL and a mixed version of these two algorithms. We can see that HessianMKL is far more efficient than SimpleMKL and such an efficiency is explained by fewer SVM evaluations without significant overload due to the Hessian computation. Note that using the mixed strategy does not improve efficiency.

Examples of convergence behavior of HessianMKL and SimpleMKL are given in Figure 2. We note that for both datasets, SimpleMKL tends to decreases the duality gap faster than HessianMKL but has more difficulties to insure convergence when near optimality. Such a behavior can also be illustrated by the evolution of the weights in Figure 3. For SimpleMKL, the weights rapidly reach a nearly optimal solution but then need many iterations for which weights vary a little, to achieve convergence.

5 Extensions

A straightforward extension is the tuning of C for an SVM with L_2 penalization of the slacks. Indeed in that case, there is no upper bound on the $\alpha_i y_i$, but a ridge of magnitude $1/C$ is added to the kernel matrix. We can thus simply add the identity matrix to the set of the M base kernel matrices and the MKL algorithm will adjust the ridge automatically.

Another extension is to consider arbitrary kernel hyperparameters such as the widths of a anisotropic RBF kernel. The Hessian of J with respect to the hyperparameters is the same as in (3), but one needs to subtract from it the second order term $(\alpha^\star)^\top \frac{\partial^2 K}{\partial d_m \partial d_{m'}} \alpha^\star$. This term vanishes in the the case of a linear combination of kernel, but in the general case, it will result in a non-convex optimization.

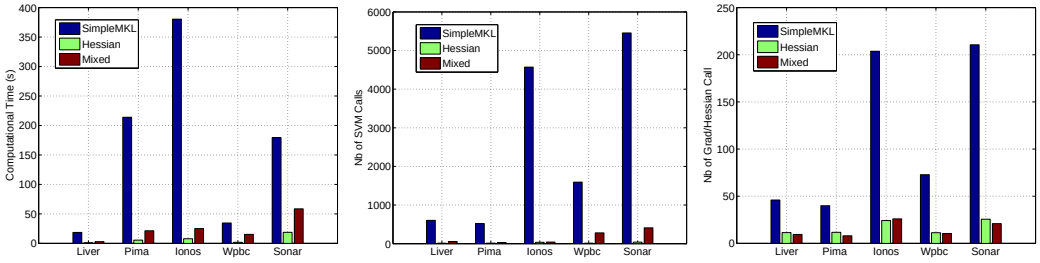

Figure 1: Comparing HessianMKL with SimpleMKL and a mixed strategy which uses a SimpleMKL iteration as a first iteration. left) Computational time. middle) Number of SVM evaluations. right) Number of gradient and/or Hessian computations.

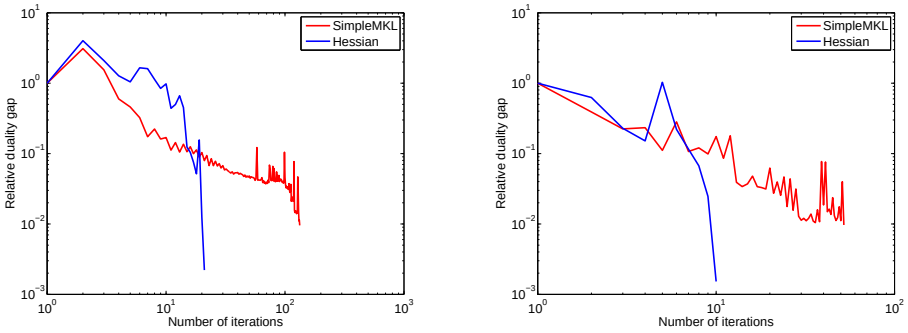

Figure 2: Convergence of SimpleMKL and HessianMKL on the *Ionosphere* and *Liver* datasets.

References

[1] O. Bousquet and D. Herrmann. On the complexity of learning the kernel matrix. In *Advances in Neural Information Processing Systems 15*. MIT Press, 2003.

[2] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1-3):131–159, 2002.

[3] G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M.I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[4] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. Simplemkl. *Journal of Machine Learning Research, to appear*, 9:1–34, 2008.

[5] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7(1):1531–1565, 2006.

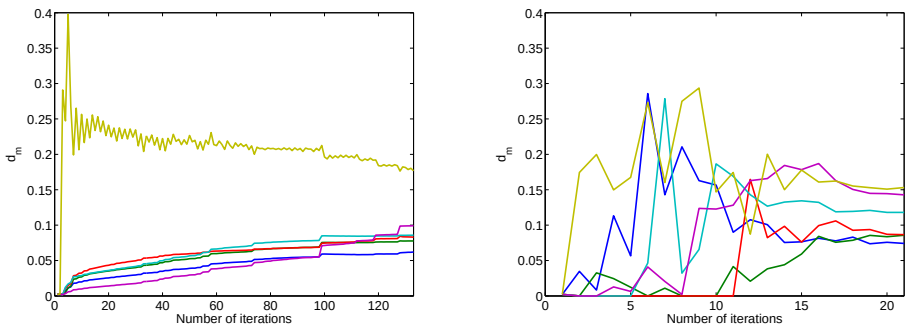

Figure 3: Weights evolution on the *Ionosphere* dataset. left) SimpleMKL right) HessianMKL.