Kernel Learning by Unconstrained Optimization

Fuxin Li[(1)], **Yunshan Fu**[(2)], **Yu-Hong Dai**[(2)], **Cristian Sminchisescu**[(1)], **Jue Wang**[(3)]
(1) Institute of Numerical Simulation, University of Bonn
(2) LSEC, Institute of Computational Mathematics, Chinese Academy of Sciences
(3) CSIS, Institute of Automation, Chinese Academy of Sciences
fuxin.li@ins.uni-bonn.de

Abstract

We study the problem of learning a kernel matrix from an apriori kernel and training data. An unconstrained convex optimization formulation is proposed, with an arbitrary convex smooth loss function on kernel entries and a LogDet divergence for regularization. Since the number of variables is of order $O(n^2)$, standard Newton and quasi-Newton methods are too time-consuming. An operator form Hessian is used to develop an $O(n^3)$ trust-region inexact Newton method, where the Newton direction is computed using several conjugate gradient steps on the Hessian operator equation. On the uspst dataset, our algorithm can handle 2 million optimization variables within one hour. Experiments are shown for both linear (Mahalanobis) metric learning and for kernel learning. The convergence rate, speed and performance of several loss functions and algorithms are discussed.

1 Introduction

Kernel methods have played a central role in modern machine learning. In recent years, there has been a growing interest in learning kernels from data. One approach is to learn a linear or convex combination of base kernels (Lanckriet et al., 2004). Others specify a parametrization of the kernel, and learn it from training data (Kondor & Lafferty, 2002).

A number of methods learn the kernel matrix directly from training data with few assumptions on the kernel structure (Tsuda et al., 2005; Kulis et al., 2006; Hoi et al., 2007; Li et al., 2007) – we refer to this as the general kernel learning problem in the sequel. The typical assumption is the positive semi-definiteness of the kernel, and the existence of an apriori reference kernel. This approach is also related to metric learning, where by learning a kernel, an implicit Euclidean metric is obtained (Li et al., 2007; Zhang, 2003). Therefore, kernel learning algorithms can also be used to estimate a Mahalanobis metric in the input space, e.g., linear metric learning (Davis et al., 2007).

Kernel learning requires a similarity/dissimilarity measure between a given kernel and the reference; recently, several authors (Tsuda et al., 2005; Kulis et al., 2006; Davis et al., 2007) have used Bregman divergences for this purpose. Many Bregman divergences are generated by functions that are *essentially smooth*(Bauschke & Borwein, 1997), which means that a function is smooth in the interior of its domain, and the norm of the gradient tends to infinity on its boundary. Essential smooth functions can act as barriers that prohibit the optimizer from moving across the boundary of its domain. Kulis et al. (2006) used this property to implement a low-rank kernel learning algorithm, where the domain of the divergence is the range space of the reference kernel.

For kernel learning, the positive semi-definiteness of the kernel matrix must be guaranteed. Therefore, kernel learning is formulated naturally as a semi-definite program (SDP), although this is not easily solvable. Alternatively, by including in the objective function a Bregman divergence with the semi-definite cone as its domain, the semi-definite constraint can be made implicit. In conjunction with differentiable loss functions, we can formulate kernel learning as unconstrained optimization, which is easier to solve, and allows for a larger class of loss functions. However, even this is still too hard, since the number of variables is large and it is difficult to apply standard Newton methods. In this work, an operator form Hessian is used, which decreases the cost of computing Hessian-matrix prod-

Appearing in Proceedings of the 12^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2009, Clearwater Beach, Florida, USA. Volume 5 of JMLR: W&CP 5.

ucts to $O(n^3)$. Based on it, we use a trust-region Newton method, where inexact Newton directions are computed with several conjugate gradient steps, based on the Hessian operator equation. On the uspst dataset, *the method is capable of optimizing 2 million kernel matrix entries in less than an hour*. Notably, the regularization path of the problem coincides with the central path of the interior-point method. It is thus possible to obtain many points on the regularization path using a warm-start strategy with little computational overhead – in fact, computing a number of solutions with different regularization parameters is sometimes faster than computing a single one.

The framework can flexibly work with any second-order differentiable convex loss function on the kernel matrix. We can not only approximate the loss used in most previous work, but also introduce new loss functions. In the experiments we use several loss functions with different datasets and make comparisons.

The paper is organized as follows: Sec. 2 reviews Bregman divergences as well as our unconstrained formulation of kernel learning. Sec. 3 presents trust-region inexact Newton methods to solve the optimization problem. Algorithmic details are discussed in Sec. 4. Different loss functions and related work are summarized in Sec. 5. Experimental results are presented in Sec. 6; we conclude in Sec. 7.

2 Kernel Learning with Bregman Divergences

We use n to denote the total number of labeled and unlabeled examples, m the number of labeled examples, d the number of dimensions of the input data, c the number of training constraints, I the identity matrix and $\mathbf{1}$ the vector of all ones.

2.1 Bregman Matrix Divergences

Let $\phi(X)$ be a strictly convex differentiable function of a matrix X. The Bregman matrix divergence of X is defined as (Kulis et al., 2006):

$$D_\phi(X,Y) = \phi(X) - \phi(Y) - tr((\nabla\phi(Y))^T(X-Y)), \quad (1)$$

where $tr(X)$ is the trace of X. Given $\phi(X) = \|X\|_F^2$, we have $D_\phi(X,Y) = \|X-Y\|_F^2$, the squared Frobenius norm. We are primarily interested in Bregman divergences that have $\overline{dom(D_\phi)} = \{X | X \succeq 0\}$, the cone of positive semi-definite matrices. We are also interested in those Bregman divergences that are generated from essentially smooth functions, which satisfy

$$\lim_{t\downarrow 0} \langle \nabla\phi(X + t(Y-X)), Y - X\rangle = -\infty$$
$$\forall X \in \text{ bd}(\text{dom}\phi) \text{ and } Y \in \text{int}(\text{dom}\phi).$$

A Bregman divergence generated from an essentially smooth function is a good barrier since it has the property that $\lim_{Y_n \to Y \in bd(dom(f))} D_f(X, Y_n) \to +\infty$ (Bauschke & Borwein, 1997).

In this work, we use the LogDet divergence, generated from the function $\phi(X) = -\log\det X$. Noticing that $\nabla\phi(X) = -X^{-1}$, the matrix divergence is[1]:

$$D_{LogDet}(X,Y) = tr(XY^{-1}) - \log\det(XY^{-1}) - n. \quad (2)$$

We focus on the LogDet divergence in this paper since it is easy to compute numerically, but the algorithms and the discussion applies to other divergences generated from essentially smooth functions, e.g. von Neumann divergence (Kulis et al., 2006). Instead of using (2) to compute the LogDet, we take advantage of $Y \succeq 0$ and use factorization $Y^{-1} = ZZ^T$. The trace and determinant of Z^TXZ are then computed instead of XY^{-1}. Cholesky factorization is used to compute the determinant. Using this decomposition, we can also detect when the matrix X has become indefinite.

2.2 Kernel Learning Formulation

When only the semi-definiteness of kernels is assumed, the admissible class is very large and it is easy to design kernels that fit the information given in the training data perfectly, e.g., the idealized kernel in (Cristianini et al., 2002). However, this would have little generalization ability without regularization. Our choice of regularizer is to bias the estimated kernel to an apriori one, with Bregman divergence acting as distance.

Let $\psi(x)$ be a convex second-order differentiable function; the kernel learning problem is formulated as:

$$f(K) = \min_{K\succeq 0} \frac{\gamma}{m} \sum_{i=1}^{c} \psi(tr(KA_i)) + D_\phi(K, K_0), \quad (3)$$

where K_0 is the apriori kernel matrix and A_i, $i = 1,\ldots,c$ are given constraint matrices. A list of $\psi(x)$ and A_i, including those used in previous work, are given in Table 2. Eq. (3) uses only convex mappings of linear constraints on the kernel matrix K, but arbitrary convex nonlinear constraints are also applicable.

By using a Bregman divergence with $\overline{dom(D_\phi)} = \{X | X \succeq 0\}$, the constraint $K \succeq 0$ is automatically satisfied. Another constraint common in kernel learning is centering $K\mathbf{1} = 0$. Kernels satisfying centering constraints has a 1-1 correspondence to a Euclidean distance matrix (Zhang, 2003). This can also be made implicit in (3), by specifying $K_0\mathbf{1} = 0$ and $A_i\mathbf{1} = 0$.

[1] It is also possible to define similar divergences for rank-deficit matrices (Kulis et al., 2006), just sums up only the nonzero eigenvalues for $\phi(X)$.

Using (1) with the LogDet divergence, the gradient of the optimization problem is

$$G = \frac{\gamma}{m}\sum_{i=1}^{c}\psi'(tr(KA_i))A_i + (K_0^{-1} - K^{-1}), \quad (4)$$

and gradient descent methods can be applied to solve it. However, the convergence rate can be very slow. For an intuition, consider a simplified optimization problem, where $K = \sum_{i=1}^{n}\lambda_i v_i v_i^T$, v_is form an orthogonal basis of $\mathbb{R}^n$ and $\psi(x) = 0$ for simplicity. Then the optimization will only be on the eigenvalues λ_i

$$\min_{\lambda} -\sum_{i=1}^{n}\log\lambda_i + \lambda_i tr(K_0^{-1}v_i v_i^T),$$

where we dropped constants. The Hessian is diagonal with entries $\frac{1}{\lambda_i^2}$. If λ_i is small, its condition number will be large, slowing down the convergence of gradient descent significantly.

3 An Inexact Newton Method

For unconstrained optimization, $\min_x f(x)$, the Newton method starts with a point x_0 sufficiently close to the optimal value x^*. In each step k, it sets $x_{k+1} = x_k - H_k^{-1}g_k$, where H_k is the Hessian of $f(x)$ and g_k its gradient at x_k.

The Newton method is best known for its quadratic convergence rate. The success of interior-point methods on conic programming largely depends on this fast rate (Boyd & Vandenberghe, 2004). However, for our problem the Hessian matrix is of order $O(n^2)$, and contains $O(n^4)$ entries. Instead of handling this directly, we resort on Hessian operators on matrices, in the Fréchet sense, which satisfy

$$\lim_{\|T\|_F\to 0}\frac{\|G(K_0+T) - G(0) - H(T)\|_F^2}{\|T\|_F^2} = 0,$$

as described in (Renegar, 2001). The Hessian (operator) of our optimization problem is[2]

$$H : \Delta K \mapsto \frac{\gamma}{m}\sum_{i=1}^{c}\psi''(tr(KA_i))tr(\Delta K A_i)A_i + K^{-1}\Delta K K^{-1}. \quad (5)$$

To obtain the exact Newton step D, one needs to compute $D = H^{-1}(G)$ or solve the Newton equation $H(D) = G$, both intensive calculations. A large-scale optimization alternative (Nocedal & Wright, 2006) is the inexact Newton method that uses conjugate gradient to find an approximate solution to the Newton equation $H(D) = G$. Solving $H(D) = G$ with conjugate gradient is equivalent to the following optimization problem: $\min_D \frac{1}{2}\mathrm{tr}(D^T H(D)) - \mathrm{tr}(D^T G)$. The inexact solver stops when the Frobenius norm of the residual $R = H(D) - G$ satisfies:

$$\|R_k\| \le \nu_k\|G_k\|. \quad (6)$$

Besides direction, we also need the step length in the inexact Newton case. One option is to line search along the estimated descent direction. However, line searches tend to miss-estimate the step length when the Hessian is badly conditioned (Nocedal & Wright, 2006). In our case, this tends to be the case for large γ, since the γ-related component of the Hessian can have low rank. To palliate this, we opt for a more sophisticated trust region method where the step length is controlled based on the size of the trust region. Specifically, a trust region is defined around the current iterate, assuming that a quadratic approximation of the objective is sufficiently accurate. We choose a step inside the trust region that minimizes the quadratic model. If the step estimate hits the boundary, the trust region is increased. If the step cannot provide satisfactory decrease of the objective, the trust region shrinks. We use the following standard second-order model for the trust region (Nocedal & Wright, 2006):

$$\begin{aligned}\min_{D_k}\quad & m_k(D_k) = \varphi(K_k) - \alpha tr(G_k D_k)\\ & \quad + \tfrac{1}{2}\alpha^2 tr(D_k H_k(D_k))\\ \text{s.t.}\quad & \|D_k\|_F \le \Delta_k, \end{aligned} \quad (7)$$

where Δ_k is the size of the trust region. We decide on sufficient decrease based on $\rho_k = \frac{\varphi(x_k)-\varphi(x_k-p_k)}{m_k(0)-m_k(p_k)}$. $\rho_k < 0$ means the objective function does not decrease. In this case, the trust region shrinks before solving (7) again. If ρ_k is near 1, the step is good, and the size of the trust region is increased; otherwise, it remains unchanged. Details are similar to the ones in (Nocedal & Wright, 2006), Chap. 4. The conjugate gradient method is used to solve (7) approximately. The convergence rate of conjugate gradient depends on the distribution of Hessian eigenvalues (Nocedal & Wright, 2006). It is therefore advisable to improve this by preconditioning the Newton equation, so to make the eigenvalue distribution better behaved in the new coordinate system. Usually, it is possible to provide a matrix $M = C^T C$ the convergence rate relates to the eigenvalue distribution of $C^{-T}HC^{-1}$ instead of H. For Hessian matrices, there are conventional preconditioning methods such as symmetric successive overrelaxation (SSOR) or incomplete Cholesky decomposition. However, these methods do not directly apply to Hessian matrix operators.

In our methods, we use the inverse of the second part of the Hessian as preconditioner. It is easy to derive as $H_s^{-1} : \Delta K \mapsto K\Delta K K$, where $H_s : \Delta K \mapsto K^{-1}\Delta K K^{-1}$ denotes the second part of the Hessian

[2] If kernel matrices are rank-deficit, K^+, the Moore-Penrose pseudo inverse of K is used instead of K^{-1}.

(5). In this case, $M^{-1} = H_s^{-1} = H_s^{-\frac{1}{2}} H_s^{-\frac{1}{2}}$. After preconditioning, $C^{-T}HC^{-1}$ becomes

$$H_s^{-\frac{1}{2}}(H(H_s^{-\frac{1}{2}}(\hat{D}))) : \\ \frac{\gamma}{m}\sum_{i=1}^{c}\psi''(tr(KA_i))tr(A_i(K^{\frac{1}{2}}\hat{D}K^{\frac{1}{2}}))K^{\frac{1}{2}}A_iK^{\frac{1}{2}} + \hat{D}.$$

The last part of the expression is an identity operator $I(D) = D$, a natural regularizer which makes the eigenvalue distribution of $C^{-T}HC^{-1}$ more preferable. Based on (Nocedal & Wright, 2006) (Sec. 7.2) with preconditioner introduced above, we provide the preconditioned conjugate gradient algorithm in Table 1, where $H_F : \Delta K \mapsto \frac{\gamma}{m}\sum_{i=1}^{c}\psi''(tr(KA_i))tr(\Delta KA_i)A_i$ is the first part of (5), $Y_j = KR_jK$ is the preconditioned residual, and $Q_j = K^{-1}P_jK^{-1}$ is an auxiliary matrix, from which we compute $H(P_j) = H_F(P_j)+Q_j$ to avoid the matrix inversion $K^{-1}DK^{-1}$ in (5).

4 Discussion

4.1 Convergence of the Algorithm

The following result guarantees the convergence rate:

Theorem 1 *((Nocedal & Wright, 2006)) Suppose that the Hessian H exists, is continuous near the optimal solution x^* and $H(x^*)$ is positive semi-definite. If the iterate D_k satisfies (6) and $\nu_k \leq \nu < 1$, then the algorithm converges if the initial point x_0 is sufficiently close to x^*. Moreover, the convergence rate is superlinear if $\nu_k \to 0$. If the Hessian is Lipschitz near x^* and $\nu_k = O(\|\nabla f(k)\|)$, the convergence rate is quadratic.*

The proof is basically based on using ν_k to remove the $H(D)$ terms in a Taylor expansion of G_k, to get sufficient decrease of gradient on each iteration. If we choose $\nu_k = \min(0.5, \sqrt{\|\nabla f_k\|})$, we can get superlinear rates. If $\nu_k = \min(0.5, \|\nabla f_k\|)$, then quadratic rates are expected. However, more iterations may be needed to solve the linear operator equation. In the experiments we use $\nu_k = \min(0.5, \sqrt{\|\nabla f_k\|})$.

4.2 Complexity

The worse-case time complexity is $O(n^3 + cn^2)$ for each iteration. The main bottleneck is the computation of the gradient, which requires summing up the constraints and inverting the Gram matrix. For kernel learning, constraint matrices A_i are usually sparse. In this case, the complexity for each iteration becomes $O(n^3+c)$. Since c is usually much smaller than n^3, the dependence on c is negligible. $O(n^3)$ operations are also necessary for computing the LogDet divergence and the verification of positive semi-definiteness.

4.3 Connection with the Barrier Method and Regularization Path

The barrier method is a basic interior-point method in convex programming (Boyd & Vandenberghe, 2004). To solve the optimization $\min_{X\succeq 0} f(X)$, the barrier method solves sequentially $tf(X) - \log\det X$ with increasing t. The optimal solution is found when $t \to \infty$. The set $X^*(t)$ that minimize $tf(X) - \log\det X$ forms the central path. Literally, the algorithm traverses the central path to find the optimal solution at the end of it (as $t \to \infty$).

It is often hard to optimize $tf(X) - \log\det X$ with a large t directly, since X tends to be near the boundary of the semi-definite cone where the Hessian varies rapidly (Boyd & Vandenberghe, 2004). Therefore, the barrier method usually starts with a small t and in each step scales it by a constant factor μ. The solution at t is always used as the starting point at μt. This warm-start strategy can increase the efficiency of barrier methods significantly.

A term $-\log\det X$ is also present in our cost, hence the behavior of our method relates to the one of the barrier method, with one important difference: $-\log\det X$ is not only a barrier function, but also part of the objective. Therefore, instead of finding only one optimal solution at the end of the central path, as in SDP barrier methods, our points on the central path correspond to our solutions with various γ. Otherwise said, the regularization path of kernel learning formulation in (3) is the central path of the barrier method used to solve a corresponding SDP. Therefore, a similar warm-start strategy is used in our algorithm to compute the solution path. It might be possible to extend our algorithm to provide an approximate solution to general nonlinear SDP. We note that a similar path property has been observed by Koh et al. (Koh et al., 2007) for l_1-regularized logistic regression.

5 Loss Functions and Related Work

5.1 Loss Functions and Regularization

Many existing kernel and metric learning methods can be formulated in our framework with little or no modification. Problems with new loss functions can also be solved. Euclidean metric learning algorithms can be transformed to an equivalent kernel learning by: $d^2(x,y) = k(x,x) + k(y,y) - 2k(x,y)$(Zhang, 2003). For linear metrics (Mahalanobis matrix), defining the metric as $d^2(x,y) = (x-y)^T K(x-y)$, we have $d^2(x,y) = Tr(K(x-y)(x-y)^T)$. We list some loss functions of kernel learning in Table 2. These losses are sparse for kernel learning, namely for any constraint involving x_i, x_j, x_k, only the i, j, k rows and columns

are nonzero. In Table 2, we use $A\{i,j,k\}$ to denote the i,j,k^{th} row and columns of a matrix A, and similarly $A\{i,j\}$ for the i^{th} and j^{th} row and column of A. These functions can be adapted into linear metric learning by simply change the representation of d^2.

Table 1: Preconditioned conjugate gradients for the trust region subproblem

input $\nu_k, Z_0 = 0, R_0 = -G_k, Y_0 = -K_kG_kK_k, P_0 = K_kG_kK_k, Q_0 = G_k$.
output D_k
  if $\|R_0\|_F < \epsilon_k$ **then**
    return $D_k = Z_0 = 0$
  end if
  for $j = 1, 2, \ldots$ **do**
    $\alpha_j = tr(Y_jR_j)/(tr(P_jQ_j) + tr(P_j, H_F(P_j)))$
    $Z_{j+1} = Z_j + \alpha_jP_j$ {Update the direction}
    if $\|Z_{j+1}\|_F \geq \Delta_k$ **then**
      Binary search for $\tau \geq 0$ so that $D_k = Z_j + \tau P_j$ satisfies $\|D_k\|_F = \Delta_k$
      return D_k **{Hit boundary}**
    end if
    $R_{j+1} = R_j + \alpha_j(H_F(P_j) + Q_j)$ {Update the residual}
    if $\|R_{j+1}\|_F < \nu_k\|G_k\|_F$ **then**
      return $D_k = Z_{j+1}$ **{Stopping condition}**
    end if
    $Y_{j+1} = KR_{j+1}K = Y_j + \alpha_j(K_kH_F(P_j)K_k + P_j)$
    $\beta_{j+1} = tr(Y_{j+1}R_{j+1})/tr(Y_jR_j)$
    $P_{j+1} = -Y_{j+1} + \beta_{j+1}P_j$
    $Q_{j+1} = -R_{j+1} + \beta_{j+1}Q_j$
  end for

For piecewise differentiable loss functions, our algorithm can be extended to do subgradient descent. However, a simpler alternative is to use a smooth surrogate, for example, approximate $\max(1-x, 0)$ with

$$f(x) = \begin{cases} \frac{2+2t}{3+t} - \frac{4}{3+t}x, & x \leq t \\ \frac{2}{(1-t)(3+t)}(1-x)^2, & t < x \leq 1 \\ 0, & x > 1 \end{cases} \quad (8)$$

For $t < 1$, this function is second-order smooth and a good approximation for $\max(1-x, 0)$ when $t \to 1$.

The regularizers used in previous work are similar to the ones we use (e.g. the one in (Kulis et al., 2006) is identical). The term of (Weinberger et al., 2006) can be written as $Tr(K\sum_{\eta_{ij}}(x_i - x_j)(x_i - x_j))^T$, where $\eta_{ij} = 1$ if x_i and x_j are of the same class. Substituting $\sum_{\eta_{ij}}(x_i - x_j)(x_i - x_j)^T$ as K_0^{-1} in (3) the difference remains in the LogDet term $-\log\det K$, which can be interpreted as the barrier for the positive definite constraint. This also holds for (Hoi et al., 2007) and (Li et al., 2007).

5.2 Related Work

While the general formulation of previous metric and kernel learning methods is similar to ours, substantial differences lie in the algorithms used. Weinberger et al. (2006) solve a linear metric learning problem with alternating projections between subgradient descent and projection to the semi-definite cone. For every projection, an eigenvalue problem needs to be solved. Kulis *et al* (2006) use Bregman projection for kernel learning. In each step, the Gram matrix is projected in $O(n^2)$ onto the half-plane specified by a linear constraint. The semidefinite constraint is implicit so no computation of eigenvalues is needed. The algorithm is also used in (Davis et al., 2007). Hoi *et al* (2007) use an SMO-based algorithm to solve kernel learning in the dual. The number of variables in the optimization is c. At each iteration, the algorithm needs to solve an optimization problem on a quadratic function with n variables in $O(n^2)$.

Most of the above methods are iterative and process a single constraint per iteration with cost $O(n^2)$. For convergence, every constraint must be visited several times. For t sweeps, the overall time complexity is $O(tcn^2)$. It is quite possible that c is of the same magnitude, or even much larger than n, when one constraint is generated for each neighboring pair or all neighboring triplets of labeled items, as in (Weinberger et al., 2006). In such cases, a $O(cn^2)$ algorithm needs a much larger factor than the $O(n^3)$ algorithm we proposed.

Weinberger and Saul (2008) propose a method that exploits the piecewise-linear property of the hinge loss to compute the gradient in a time independent on c, in linear metric learning. This makes their algorithm faster than ours (notice that their speed-up can also be used for our method using the hinge-loss and subgradient methods). However, for more general loss functions, as pursued here, the speed-up does not apply.

6 Experiments

We refer to our algorithm as TRIN (Trust-Region Inexact Newton). All experiments are run using a single core on a dual-core Pentium Xeon 5130 PC.

6.1 Speed in Linear Metric Learning

Here we compare the speed of TRIN and the ITML algorithm (Davis et al., 2007) in linear metric learning. We use the same loss function and constraint matrices as in (Davis et al., 2007). Note that TRIN is not particularly profiled for linear metric learning, since every iteration takes $O(cd^2)$ if c is larger than d. However, this evens the ground with ITML, since a sweep in ITML also takes $O(cd^2)$.

Experiments are run on the `Spambase` dataset from the UCI machine learning repository, which has 4601 examples in 57 dimensions. We randomly choose 400,

Table 2: List of the loss functions used in previous work and the newly proposed loss function. $y_i = y_j$ means that items i and j are in the same class, while $y_i \neq y_j$ means that i and j are in different classes.

Loss Function	Pattern of Constraint Matrix	Literature
Log-loss: $\psi(x) = \log(1 + \exp(-2x))$	$A\{i,j,k\} = \begin{bmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix}$	New
LMNN-loss: $\psi(x) = \max(1 - x, 0)$	$A\{i,j,k\} = \begin{bmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix}$	(Weinberger et al., 2006)
Linear loss: $\psi(x) = -x$	$A\{i,j,k\} = \begin{bmatrix} 0 & -1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix}$	(Li et al., 2007)
ITML loss: $\psi(x) = \begin{cases} 1(x > u)(\frac{x}{u} - \log \frac{x}{u} - 1), & y_i = y_j \\ 1(x < l)(\frac{x}{l} - \log \frac{x}{l} - 1), & y_i \neq y_j \end{cases}$	$A\{i,j\} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$	(Davis et al., 2007)
Hoi loss: $\psi(x) = \max(1 - x, 0)$	$A\{i,j\} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	(Hoi et al., 2007)

800 or 6400 item pairs and add a constraint for each. The convergence tolerance is set to 10^{-3}. The initial kernel K_0 is set to the identity matrix. Both algorithms are tested on 9 parameters $10^{-4} - 10^4$. γ is not normalized by the number of labeled samples m here to provide equal comparison with ITML. The result is averaged over 20 random trials. The ITML code was made available by the authors (Davis et al., 2007).

The results are shown in fig. 1. ITML scales unfavorably with the number of constraints. With increasing number of constraints, the performance of ITML deteriorates heavily, whereas TRIN is virtually unaffected. For 6400 constraints, ITML only converges on the first 4 parameters in the 8000 sweeps limit. In fig. 1 we notice that the error rate drops significantly as more constraints are added, which suggests that methods that can deal with many constraints time-effectively are likely to be more successful learners.

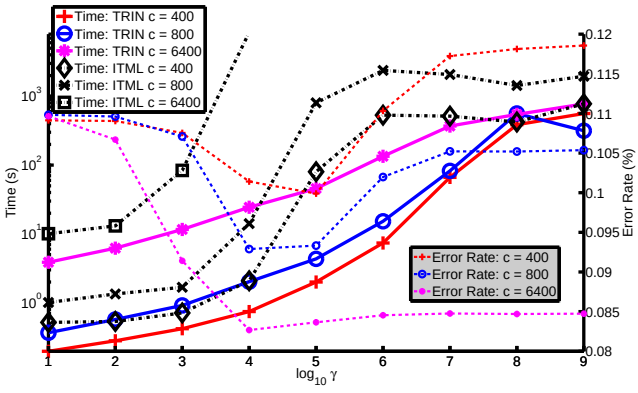

Figure 1: Comparison of the speed of ITML and the trust-region Newton method. The performance of ITML drops when the number of constraints increases. In contrast the speed of TRIN is roughly unaffected.

6.2 Regularization Path and Speed in Kernel Learning

Here we test the number of steps required for convergence and running times for kernel learning. Also, we compare the speed of computing many solutions on the regularization path, with the one computing a single one. Comparisons are not done with the low-rank ITML kernel learning method (Kulis et al., 2006), since in this setting (we use $50n$ constraints), all methods that depend on the number of constraints ($O(cn^2)$ complexity) are too slow to terminate in a reasonable time. Instead, we compare with a Polak-Ribiere$^+$ conjugate gradient method (Nocedal & Wright, 2006) on our unconstrained formulation.

We use the log-loss in this experiment and test on several UCI datasets. A random 50% of the items are labeled. For each labeled item, 100 constraints are generated from each pair between 10 in-class neighbors and 10 out-class neighbors of the item – total $50n$ constraints. The initial kernel used for penalty is the pseudo-inverse of the Laplacian of the adjacency graph, built using the Gaussian kernel $p_{ij} = \exp(-\frac{\|x_i - x_j\|^2}{\sigma^2})$. Convergence tolerance is set to 10^{-3} and results are averaged over 50 random trials.

To test the regularization path of the algorithm, we search γ in the range $2^{-12} - 2^{16}$. Like barrier methods, for each γ we initialize using the solution from solving for the previous regularization parameter. By multiplying γ each time with a multiplier α we instantiate points on the regularization path. The baseline is the solution for $\gamma = 2^{16}$. For each multiplier α, we also compute the baseline solution for $\gamma = 2^{16}$ when $\alpha\gamma \geq 2^{16}$. The results are shown in Table 3 and 4. The average number of iterations needed for TRIN ranges from 10 to 263, with convergence rate seemingly depending more on the intrinsic 'hardness'

Table 3: Speed, convergence rate and path of TRIN. The number of examples and problem dimension of the problem are shown in parentheses. The number of distinct optimization variables is also shown under "No. var".

Dataset		Changing multiplier α							Baseline
Multiplier c		2	4	8	16	32	64	128	
No. of parms		29	15	11	8	7	6	5	1
`Breast` (699×9)	Iter:	101.9	54.8	42.6	34.5	32.0	30.1	26.9	14.3
No. var: 243951	Time (s):	400.0	221.6	171.6	139.1	129.6	122.1	106.4	55.4
`Sonar` (208×61)	Iter:	262.4	212.3	267.8	308.3	604.0	465.0	1343.8	> 10000
No. var: 21528	Time (s):	70.8	64.5	106.9	157.3	520.4	261.0	2452.2	> 10000
`Iono` (351×33)	Iter:	348.7	539.0	797.9	1411.6	1746.9	1731.9	3173.1	> 10000
No. var: 61425	Time (s):	293.9	538.9	1051.5	2930.3	4067.6	4460.5	9232.4	> 10000
`Heart` (270×14)	Iter:	78.8	51.6	39.9	32.3	29.6	26.4	26.3	13.2
No. var: 36315	Time (s):	29.4	27.2	21.6	17.5	16.2	15.0	13.7	5.4
`Wine` (178×9)	Iter:	73.4	45.8	35.5	29.5	27.0	24.6	22.2	10.1
No. var: 15753	Time (s):	17.6	10.6	9.0	7.6	6.0	5.6	5.1	1.8

Table 4: Speed, convergence rate and path of CG. The number of examples and the problem dimension are shown in parentheses. The number of distinct optimization variables is also shown under 'No. var'.

Dataset		Changing multiplier α							Baseline
Multiplier c		2	4	8	16	32	64	128	
No. of parms		29	15	11	8	7	6	5	1
`Breast` (699×9)	Iter:	245.3	152.5	121.2	104.9	90.7	85.9	76.3	96.4
No. var: 243951	Time (s):	766.1	483.6	401.7	343.6	296.9	285.1	253.1	815.2
`Sonar` (208×61)	Iter:	14826.3	9191.7	7863.1	6733.9	6111.5	5307.5	5458	5220.8
No. var: 21528	Time (s):	4759.6	3125.7	2688.5	2279.4	2127.4	1969.1	1996.0	3080.8
`Iono` (351×33)	Iter:	> 10000	> 10000	> 10000	> 10000	> 10000	8322	7123.6	> 10000
No. var: 61425	Time (s):	> 10000	> 10000	> 10000	> 10000	> 10000	11293.5	9784.7	> 10000
`Heart` (270×14)	Iter:	364.1	231.9	178.9	155.1	143.4	118.9	119.6	119.7
No. var: 36315	Time (s):	97.4	63.3	50.5	45.7	42.8	35.6	36.6	73.5
`Wine` (178×9)	Iter:	220.5	134.6	105.7	90.7	80.4	71.3	71.5	72.5
No. var: 15753	Time (s):	24.6	15.0	11.8	10.2	9.1	8.4	8.4	12.9

of the dataset than the size of the kernel matrix, per se. Both algorithms seem to be able to compute a number of solutions with no significant overhead compared to the work required to compute a single one, sometimes even faster. In the 'hard' `Iono` and `Sonar` datasets, the speed of conjugate gradient is very slow, whereas TRIN solves the problem fairly quickly. But even for the 'easy' datasets where conjugate gradient has good convergence rates, TRIN is still about twice as fast.

6.3 Loss Functions

In this experiment we use the same setting as previously but focus on different loss functions. We test the first four losses in Table 2, where LMNN is approximated by (8) with $t = 0.7$. For classification, we follow (Li et al., 2007) to do Kernel PCA on the learned kernels to obtain a 10-dimensional representation of each item, then 1-NN is used to classify. The results are shown in Table 5. There are no significant differences between the log-Loss, ITML loss and LMNN loss. However these losses are better than the simple linear ones in several datasets. All methods perform better than the baseline Laplacian Eigenmaps, where label information is only used for 1-NN classification.

6.4 uspst Dataset

We did kernel learning experiments on the testset of USPS, which has 2007 examples. The kernel matrix has 2015028 distinct entries, which makes optimization very difficult. Using log-loss and a multiplier $\alpha = 32$, we are able to get the kernel learning solution for 7 different γ in less than 1 hour. The number of constraints used is 40100. The results are averaged over 10 random trials. The results are shown in fig. 2.

7 Conclusion

A framework for kernel learning based on Bregman divergences is proposed. Different from existing algorithms, the problem is formulated as unconstrained optimization solved with a trust-region inexact Newton method. Each iteration needs $O(n^3 + cn^2)$ but the method can take advantage of the sparsity of constraint matrices to obtain an effective complexity $O(n^3)$. The regularization path of our problem coincides with the corresponding central path of the barrier method, so we can obtain a number of solutions on the regularization path with no significant additional com-

Table 5: Test set error rates on UCI datasets, lowest error rate in bold.

Dataset	Log-Loss	LMNN Loss	ITML Loss	Linear Loss	Laplacian
Breast	**3.54**	**3.54**	3.62	4.78	4.78
	(±0.73)	(±0.74)	(±1.26)	(±0.84)	(±0.87)
Sonar	17.79	17.52	17.37	**16.22**	18.50
	(±4.14)	(±5.06)	(±4.10)	(±4.15)	(±3.51)
Ionosphere	9.40	8.42	9.08	**7.07**	9.68
	(±1.46)	(±2.05)	(±1.70)	(±1.81)	(±1.85)
Heart	**18.22**	**18.22**	18.46	23.64	23.53
	(±2.65)	(±2.61)	(±3.29)	(±3.61)	(±3.35)
Wine	**2.57**	3.01	2.63	3.09	3.39
	(±1.39)	(±1.41)	(±1.51)	(±1.95)	(±1.09)

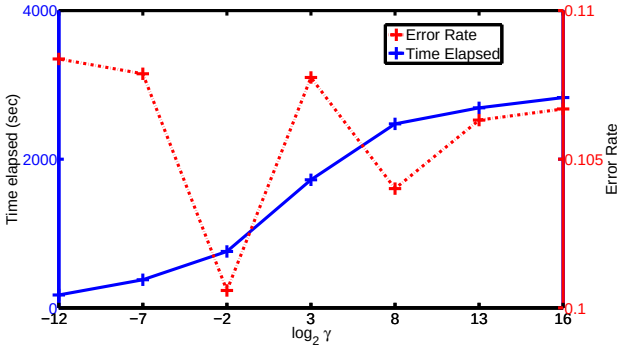

Figure 2: Time required to traverse the regularization path on the uspst dataset and the error rate. Best error rate is obtained for $\gamma = 0.25$ ($\log_2 \gamma = -2$).

putation time. The computation time and accuracy of the algorithm is systematically tested on a number of datasets and loss functions. The results show that our method is more efficient than the state-of-the-art ITML algorithm for problems with many constraints. We plan to investigate using von Neumann divergence in the framework and possible improvements by better utilizing the sparsity of constraints.

Acknowledgements

Part of the work was done when FL was a student at Institute of Automation, CAS. FL and CS was supported in part by the European Commission under award MCEXT-025481. FL and JW were supported by the 973 Program of China (No.2004CB318103), National Natural Science Foundation of China (NSFC) grant 60575001 and 60835002. YF and YHD were supported by NSFC grants 10571171 and 10831006 and the CAS grant kjcx-yw-s7-03.

References

Bauschke, H. H., & Borwein, J. M. (1997). Legendre functions and the method of random bregman projections. *Journal of Convex Analysis*, *4*, 27–67.

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.

Cristianini, N., Kandola, J., Elisseeff, A., & Shawe-Taylor, J. (2002). On kernel-target alignment. *NIPS 14*.

Davis, J. V., Kulis, B., Jain, P., Sra, S., & Dhillon, I. S. (2007). Information-theoretic metric learning. *ICML*.

Hoi, S. C. H., Jin, R., & Lyu, M. R. (2007). Learning nonparametric kernel matrices from pairwise constraints. *ICML*.

Koh, K., Kim, S.-J., & Boyd, S. (2007). An interior-point method for large-scale l1-regularized logistic regression. *JMLR*, *8*, 1519–1555.

Kondor, R., & Lafferty, J. (2002). Diffusion kernels on graphs and other discrete structures. *ICML*.

Kulis, B., Sustik, M., & Dhillon, I. S. (2006). Learning low-rank kernel matrices. *ICML*.

Lanckriet, G. R. G., Cristianini, N., Bartlett, P., Ghaoui, L. E., & Jordan, M. I. (2004). Learning the kernel matrix with semidefinite programming. *JMLR*, *5*, 27–72.

Li, F., Yang, J., & Wang, J. (2007). A transductive framework of metric learning by spectral dimensionality reduction. *ICML*.

Nocedal, J., & Wright, S. J. (2006). *Numerical optimization (2nd ed.)*. Springer-Verlag.

Renegar, J. (2001). *A mathematical view of interior-point methods in convex optimization*. Philadelphia, PA, USA: SIAM.

Tsuda, K., Rätsch, G., & Warmuth, M. (2005). Matrix exponentiated gradient updates for online learning and bregman projection. *JMLR*, *5*, 27–72.

Weinberger, K. Q., Blitzer, J., & Saul, L. K. (2006). Distance metric learning for large margin nearest neighbor classification. *NIPS 18*.

Weinberger, K. Q., & Saul, L. (2008). Fast solvers and efficient implementations for distance metric learning. *ICML*.

Zhang, Z. (2003). Learning metrics via discriminant kernels and multidimensional scaling: Toward expected euclidean representation. *ICML*.