Kernel Conjugate Gradient

Nathan Ratliff
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
ndr@andrew.cmu.edu

J. Andrew Bagnell
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
dbagnell@ri.cmu.edu

Abstract

We propose a novel variant of conjugate gradient based on the Reproducing Kernel Hilbert Space (RKHS) inner product. An analysis of the algorithm suggests it enjoys better performance properties than standard iterative methods when applied to learning kernel machines. Experimental results for both classification and regression bear out the theoretical implications. We further address the dominant cost of the algorithm by reducing the complexity of RKHS function evaluations and inner products through the use of space-partitioning tree data-structures.

1 Introduction

Kernel methods, in their various incarnations (e.g. Gaussian Processes (GPs), Support Vector machines (SVMs), Kernel Logistic Regression (KLR)) have recently become a preferred approach to non-parametric machine learning and statistics. They enjoy this status because of their conceptual clarity, strong empirical performance, and theoretical foundations.

The primary drawback to kernel methods is their computational complexity. GPs require the inversion of an $n \times n$ (covariance/kernel) matrix, implying a running time of $O(n^3)$, where n is the size of the training set. SVMs require similar computation to solve the convex program, although intense research has gone into fast, specialized approximations [13].

State-of-the-art approaches to kernel learning revolve largely around two techniques: iterative optimization algorithms, and learning within a subspace of the original problem specified Reproducing Kernel Hilbert Space (RKHS). In this work, we explore a novel variant of conjugate gradient descent based on the RKHS inner product that we term Kernel Conjugate Gradient (KCG). Our results suggest that for kernel machines with differentiable loss functions, KCG both theoretically and empirically requires fewer iterations to reach a solution, with identical cost per iteration as the vanilla conjugate gradient algorithm.

We further address the dominant cost of the algorithm by reducing the complexity of RKHS function evaluations and inner products through the use of space-partitioning tree data-structures. The methods we present here are of particular importance in that they can be used to further improve many existing kernel learning algorithms and approximations.

2 Reproducing Kernel Hilbert Spaces

This section briefly reviews the basic concepts behind Reproducing Kernel Hilbert Spaces (RKHS) necessary to the discussion that follows.

An RKHS of functions $\mathcal{H}_k$ is a complete inner product space, known as a Hilbert space, that arises from the completion of a set of basis functions $\mathcal{B}_k = \{k(x,.) \mid x \in \mathcal{X}\}$, where $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a symmetric, positive-definite function known as the kernel, and $\mathcal{X}$ is a (perhaps infinite) domain sometimes known as the index set. A common kernel, and one used exclusively throughout our experiments, is the exponential Radial Basis Function (RBF) $k(x,x') = e^{\frac{-\|x-x'\|^2}{2\sigma^2}}$. The RKHS inner product between two functions $f = \sum_i \alpha_i k(x_i,.)$ and $g = \sum_j \beta_j k(x_j,.)$ is defined as

$$\langle f, g \rangle_{\mathcal{H}_k} = \sum_{i,j} \alpha_i \beta_j k(x_i, x_j).$$

Central to the idea of an RKHS is the *reproducing property* which follows directly from the above definition. It states that the basis functions $k(x,.) \in \mathcal{B}_k$ are *representers of evaluation*. Formally, for all $f \in \mathcal{H}_k$ and $x \in \mathcal{X}$, $\langle f, k(x,.) \rangle_{\mathcal{H}_k} = f(x)$. i.e. the evaluation of f at x is the scalar projection of f onto $k(x,.)$. Note that there exists a simple mapping $\phi : \mathcal{X} \rightarrow \mathcal{B}_k$ between the domain $\mathcal{X}$ and the RKHS basis $\mathcal{B}_k$ defined by the kernel as $\phi(x) = k(x,.)$. It follows that for any $x, x' \in \mathcal{X}$

$$\langle \phi(x), \phi(x') \rangle_{\mathcal{H}_k} = \langle k(x,.), k(x',.) \rangle_{\mathcal{H}_k} = k(x,x').$$

A complete overview of these concepts can be found in [1].

A fundamental result first proven in [8] and then generalized in [3] is the *Representer Theorem*, which makes possible the direct minimization of a particular class of functionals. It states that given a subset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n \subset \mathcal{X} \times \mathbb{R}$ (i.e. the dataset), the minimizer of a functional with the form $F[f] = c((x_1, y_1, f(x_1)), \ldots, (x_n, y_n, f(x_n))) + g(\|f\|^2_{\mathcal{H}_k})$, where $c : \mathcal{X} \times \mathbb{R}^2 \rightarrow \mathbb{R}$ is arbitrary and $g : [0, \infty] \rightarrow \mathbb{R}$ is strictly monotonically increasing, must have the form $\tilde{f} = \sum_{x_i \in \mathcal{D}} \alpha_i k(x_i,.)$. A particularly useful RKHS functional in machine learning for which this holds is

$$R[f] = \sum_{i=1}^{n} l(x_i, y_i, f(x_i)) + \frac{\lambda}{2} \langle f, f \rangle_{\mathcal{H}_k} \tag{1}$$

known as a regularized risk functional [13].

3 The Functional Gradient

Gradient based algorithms, such as steepest descent, are traditionally defined with respect to the gradient arising from the Euclidean inner product between parameter vectors. There are many ways in which a given regularized risk functional can be parameterized, though, and each gives rise to a different parameter gradient. It seems more natural to follow the gradient defined uniquely by the RKHS inner product. We review some basic concepts behind the functional gradient below.

The functional gradient may be defined implicitly as the linear term of the change in a function due to a small perturbation ϵ in its input [10]

$$F[f + \epsilon g] = F[f] + \epsilon \langle \nabla F[f], g \rangle + O(\epsilon^2).$$

Following this definition using the RKHS inner product $\langle ., . \rangle_{\mathcal{H}_k}$ allows us to define the kernel gradient of a regularized risk functional [13]. We use three basic formulae that can be easily derived from this definition:

1. *Gradient of the evaluation functional.* Define $F_x : \mathcal{H}_k \to \mathbb{R}$ as $F_x[f] = f(x) = \sum_i \alpha_i k(x_i, x)$. Then $\nabla_k F_x[f] = k(x, .)$.
2. *Gradient of the square RKHS norm.* Define $F_{\langle .,. \rangle} : \mathcal{H}_k \to \mathbb{R}$ as $F_{\langle .,. \rangle}[f] = \|f\|^2_{\mathcal{H}_k} = \langle f, f \rangle_{\mathcal{H}_k}$. Then $\nabla_k F_{\langle .,. \rangle}[f] = 2f$.
3. *Chain rule.* Let $g : \mathbb{R} \to \mathbb{R}$ be a differentiable function, and let $F : \mathcal{H}_k \to \mathbb{R}$ be an arbitrary differentiable functional. Then $\nabla_k g(F[f]) = g'(F[f]) \nabla_k F[f]$.

A straight forward application of the above formulae brings us to the following result. The kernel gradient of a regularized risk functional (Equation 1) is

$$\begin{aligned} \nabla_k R[f] &= \sum_{i=1}^{n} \frac{\partial}{\partial z} l(x_i, y_i, z)|_{f(x_i)} \nabla_k F_{x_i}[f] + \lambda f & (2) \\ &= \sum_{i=1}^{n} \frac{\partial}{\partial z} l(x_i, y_i, z)|_{f(x_i)} k(x_i, .) + \lambda \sum_{i=1}^{n} \alpha_i k(x_i, .) & (3) \\ &= \sum_{i=1}^{n} \left(\frac{\partial}{\partial z} l(x_i, y_i, z)|_{f(x_i)} + \lambda \alpha_i \right) k(x_i, .). & (4) \end{aligned}$$

Equation 4 allows us to easily find the kernel gradient of the following two well known functionals used throughout the remainder of this paper.

1. *Kernel Logistic Regression (KLR)* [16]. Let $y \in \{-1, 1\}$. Then

$$\begin{aligned} R_{klr}[f] &= \sum_{i=1}^{n} \log\left(1 + e^{y_i f(x_i)}\right) + \frac{\lambda}{2} \langle f, f \rangle_{\mathcal{H}_k} & (5) \\ \Rightarrow \nabla_k R_{klr}[f] &= \sum_{i=1}^{n} \left(\lambda \alpha_i + \frac{y_i}{1 + e^{-y_i f(x_i)}} \right) k(x_i, .). & (6) \end{aligned}$$

2. *Regularized Least Squares (RLS)* [12].

$$\begin{aligned} R_{rls}[f] &= \frac{1}{2} \sum_{i=1}^{n} (y_i - f(x_i))^2 + \frac{\lambda}{2} \langle f, f \rangle_{\mathcal{H}_k} & (7) \\ \Rightarrow \nabla_k R_{rls}[f] &= \sum_{i=1}^{n} (f(x_i) - y_i + \lambda \alpha_i) k(x_i, .). & (8) \end{aligned}$$

4 The Kernel Gradient in Parametric Form

It is important to note that Equation 4 shows a special case of the property that for functionals $F[f] = c((x_1, y_1, f(x_1)), \ldots, (x_n, y_n, f(x_n))) + g(\|f\|^2_{\mathcal{H}_k})$ for which the *Representer Theorem* holds, $\nabla_k F[f] = \sum_{i=1}^{n} \gamma_i k(x_i, .)$ for appropriate $\gamma_i \in \mathbb{R}$. In other words, the kernel gradient of F is represented in the finite-dimensional subspace $\mathcal{S}_\mathcal{D} = \text{span}\{\mathcal{B}_\mathcal{D}\}$ of $\mathcal{H}_k$. A gradient descent type method through $\mathcal{S}_\mathcal{D}$ then amounts to modifying the coefficients α_i of the current function f by γ_i. That is

$$\tilde{f} \leftarrow f - \lambda \nabla_k F[f] \Leftrightarrow \tilde{\alpha}_i \leftarrow \alpha_i - \lambda \gamma_i,$$

just as a standard gradient descent algorithm would. The difference is that the coefficients γ_i for the kernel gradient are not the same as those of the parameter gradient. One can verify that they differ by $\nabla_\alpha F[f] = K\gamma$ where K is the kernel matrix. This relation can

Algorithm 1 Kernel Conjugate Gradient

procedure KCG($F : \mathcal{H}_k \to \mathbb{R}$, $f_0 \in \mathcal{H}_k$, $\epsilon > 0$)
  $i \leftarrow 0$
  $g_0 \leftarrow \nabla_k F[f_0] = \sum_{j=1}^{n} \gamma_j^{(0)} k(x_j, .)$
  $h_0 \leftarrow -g_0$
  while $\langle g_i, g_i \rangle_{\mathcal{H}_k} > \epsilon$ **do**
    $f_{i+1} \leftarrow f_i + \lambda_i h_i$ where $\lambda_i = \arg\min_\lambda F[f_i + \lambda h_i]$
    $g_{i+1} \leftarrow \nabla_k F[f_{i+1}] = \sum_{j=1}^{n} \gamma_j^{(i+1)} k(x_j, .)$
    $h_{i+1} \leftarrow -g_{i+1} + \eta_i h_i$ where $\eta_i = \frac{\langle g_{i+1} - g_i, g_{i+1} \rangle_{\mathcal{H}_k}}{\langle g_i, g_i \rangle_{\mathcal{H}_k}} = \frac{(\gamma^{(i+1)} - \gamma^{(i)})^T K \gamma^{(i+1)}}{\gamma^{(i)T} K \gamma^{(i)}}$
    $i \leftarrow i + 1$
  end while
  return f_i
end procedure

be derived by defining the gradient as the direction of steepest ascent for a "small" change in coefficients α:

$$\nabla F[\alpha] = \max_{\gamma} F[\alpha + \gamma] \; s.t. \; \|\gamma\| < \epsilon.$$

It can be shown that taking $\|\gamma\|_\alpha^2$ as $\gamma^T \gamma$ gives the vanilla parameter gradient $\nabla_\alpha F$, while defining the norm with respect to the RKHS inner product $\|\gamma\|_{\mathcal{H}_k}^2 = \sum_{i,j} \gamma_i \gamma_j k(x_i, x_j) = \gamma^T K \gamma$ gives the functional gradient coefficients $\nabla_k F = K^{-1} \nabla_\alpha F$. (See [7] for the derivation of gradient w.r.t. arbitrary metrics.)

5 The Kernel Conjugate Gradient Algorithm

Both in theory and in practice it is understood that conjugate gradient (CG) methods outperform standard steepest descent procedures [15]. These techniques have been used profusely throughout machine learning, in particular, for regularized risk minimization and kernel matrix inversion [4][13].

In this section, we present an algorithm we term Kernel Conjugate Gradient (KCG) that takes advantage of conjugate direction search while utilizing the RKHS inner product $\langle f, g \rangle_{\mathcal{H}_k} = \alpha^T K \beta$. Algorithm 1 gives the general (nonlinear) KCG algorithm in Polak-Ribière form [15].

Note that the computational complexity per iteration of KCG is essentially identical to that of the more conventional Parameter Conjugate Gradient (PCG) algorithm. Intuitively, while the kernel inner product takes time $O(n^2)$ compared to the $O(n)$ vanilla inner product used by PCG, KCG is correspondingly more efficient in the gradient computation since $\nabla_\alpha F[f] = K\gamma$, where $\nabla_k F[f] = \sum_i \gamma_i k(x_i, .)$. It is possible in the case of RLS to step through an iteration of each algorithm and show that the running time is entirely equivalent.

We emphasize that despite its somewhat involved derivation, the implementation of this algorithm is just a simple extension of PCG. The differences amount to only a change of inner product $\alpha^T \beta \to \sum_{i,j} \alpha_i \beta_j k(x_i, x_j) = \alpha^T K \beta$, and a different, though in some ways simpler, gradient computation. We also point out that the line optimization (step 6) can be solved in closed-form in the case of quadratic risk functionals (e.g. RLS). For starting point $f = \sum_i \alpha_i k(x_i, .)$ and search direction $h = \sum_i \gamma_i k(x_i, .)$ we have

$$\arg\min_\lambda F[f + \lambda h] = -\frac{\alpha^T K \gamma}{\gamma^T A \gamma}$$

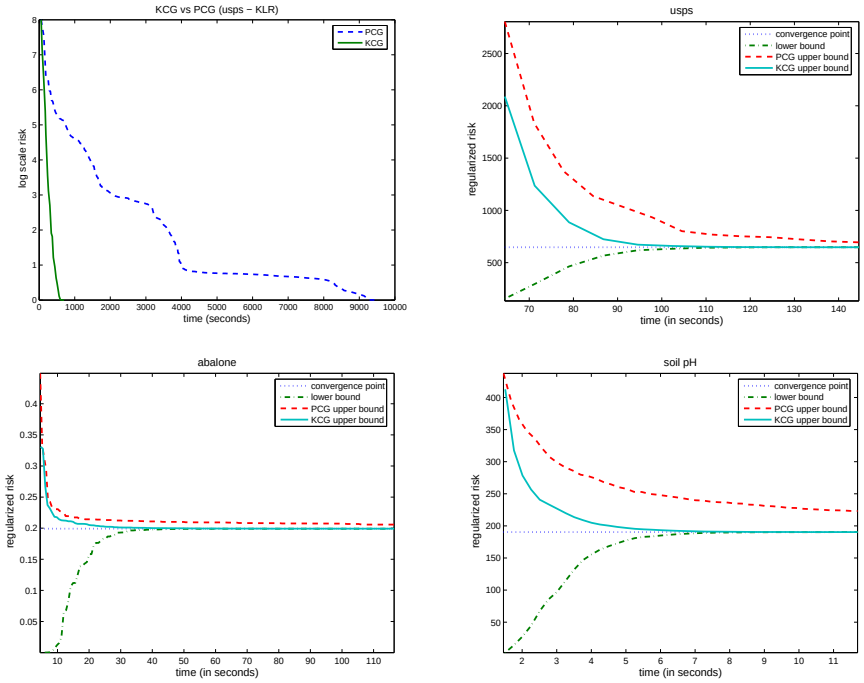

Figure 1: The first plot shows the relative performances of KCG and PCG in minimizing the KLR risk functional using the `USPS` dataset. This plot is in log scale to emphasize the improved conditioning of KCG. The remaining three plots show relative convergence rates of the RLS upper bound in linear scale. KCG provides a significantly tighter bound in all cases.

where A is the Hessian of the quadratic functional when parameterized by α. Note that this formula differs from that derived from the parameter gradient ($-\alpha^T\gamma/\gamma^T A\gamma$) only in the numerator's inner product, as is a common theme throughout this algorithm. The theoretical and experimental results given below suggest that there is little reason why one should prefer PCG to KCG in most kernel algorithms.

6 Experimental Results - Kernel Conjugate Gradient

We bench-marked KCG against PCG for both classification and regression tasks. In all cases, KCG significantly out performed PCG.

Our first test was performed using KLR on the `USPS` dataset (with a training/test size of 7291/2007) for the common one-vs-all task of recognizing the digit 4. We used a length scale hyperparameter $\sigma = 5$ as was used in [12] for RLS classification, and a regularization constant $\lambda = 0$. Figure 1 summarizes the results in log scale.

Second, we used RLS for both regression and classification using the `Abalone` and `Soil` datasets in addition to the `USPS` dataset. The `Abalone` dataset [1] consisted of 3133 training examples and 1044 test examples in 8 attributes. Our experimental setup was equivalent to that in [14]. The `Soil` dataset contained three-dimensional examples of soil pH levels in areas of Honduras partitioned into a training set of size 1709 and a test set of size 383. The

[1] See UCI repository: `http://www.ics.uci.edu/ mlearn/MLRepository.html`

latter dataset and corresponding hyperparameters ($\lambda = 0.1514$ and $\sigma = 0.296283$) were provided by [5].

Again, the results are summarize in Figure 1. For RLS, there exists a quadratic

$$p(\alpha) = \frac{1}{2}\alpha^T(K + \lambda I)\alpha - y^T\alpha$$

that can provide a lower bound to the regularized risk [4][13]. As theory suggests (see below) the upper bound under KCG converges comparably with the lower bound, while the upper bound under PCG lags considerably behind. This implies faster convergence under a gap termination criterion [13].

7 KCG Analysis

The kernelized conjugate gradient algorithm was derived from a normative point of view where it was argued that $\langle f, g\rangle_{\mathcal{H}_k}$ defined the natural notion of inner product in the RKHS and hence for the optimization procedure. The strong empirical performance of KCG noted in the previous section, while in some sense not surprising given we are using the "right" inner product, deserves some analysis. We examine here the linear case (as in RLS) where the analysis is more transparent, although similar results hold near the optima of non-linear risk functionals.

We note a classic bound on the error reduction of CG (see [9]),

$$\|e_i\|_A \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^i \|e_0\|_A,$$

where A is the Hessian of the quadratic with corresponding condition number κ, and $\|x\|_A = x^TAx$ is known as the energy norm of x. Loosely speaking, this gives a running time complexity of $O(\sqrt{\kappa})$ for PCG.

We start by analyzing the effective condition number of KCG. As with essentially all variants of CG, the algorithm's dynamics can be described in terms of a preconditioning to the spectrum of the Hessian [15]. It can be checked that KCG is equivalent to an implicitly preconditioned conjugate gradient algorithm with preconditioner K. The following theorem relates the running time of PCG to KCG in light of the bound give above.

Theorem. *Let κ_{PCG} be the condition number of R_{rls} (Equation 2), and let κ_K be the condition number of the kernel matrix $K = [k(x_i, x_j)]_{i,j}$. Then the condition number κ_{KCG} resulting from preconditioning the RLS risk functional by K has the relation $\kappa_{PCG} = \kappa_K\kappa_{KCG}$.*

Proof. Let $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$ be the eigenvalues of K. The condition number of K is then $\kappa_K = \sigma_1/\sigma_n$. The Hessian of R_{rls} is $A = K^TK + \lambda K$ and has eigenvalues $\sigma_i^2 + \lambda\sigma_i = \sigma_i(\sigma_i + \lambda)$, given in terms of the eigenvalues of K. This implies

$$\kappa_{PCG} = \frac{\sigma_1}{\sigma_n}\left(\frac{\sigma_1+\lambda}{\sigma_n+\lambda}\right) = \kappa_K\frac{\sigma_1+\lambda}{\sigma_n+\lambda}.$$

Since K is symmetric, positive-definite, the preconditioned Hessian becomes $K^{-1}A = K^{-1}(K^TK + \lambda K) = K + \lambda I$, with corresponding eigenvalues $\sigma_i + \lambda$. Thus, $\kappa_{PCG} = \kappa_K\kappa_{KCG}$. □

The condition number κ_K of K is typically very large. In particular, as the regularization constant decreases, the asymptotic bound on the convergence of PCG approaches the square of the bound on KCG. Alternatively, as the regularization constant increases, κ_{KCG} approaches 1 implying an $O(1)$ convergence for KCG, while the convergence of PCG remains bounded below by $O(\kappa_K)$.

It is informative to note that the decrease in computational complexity from PCG to KCG ($O(\kappa^{1/2})$ to $O(\kappa^{1/4})$) is the same as that seen from steepest descent to PCG ($O(\kappa)$ to $O(\kappa^{1/2})$) [9].

8 Tree-Augmented Algorithms

For many stationary kernels, it is often the case that the majority of the basis functions in $\mathcal{B}_\mathcal{D}$ are nearly orthogonal. This often stems from a relationship between the degree of orthogonality of two basis functions $k(x,.), k(x',.) \in \mathcal{B}_\mathcal{D}$ and the Euclidean distance between x and x'. This is often the case when using the exponential RBF kernel given above, in which the orthogonality between two basis functions increases exponentially with the square Euclidean distance between the two points $\|x - x'\|^2$.

Previous work has shown how the evaluation of RKHS functions $f(x) = \sum_i \alpha_i k(x_i, x)$ can be made fast when this holds using N-body type algorithms [6][2]. Intuitively, the idea is to store the training data in a space-partitioning tree, such as a KD-tree [11] as was use in our experiments below, and recursively descend the tree during evaluation, pruning negligible contributions. The maximum and minimum impact of each set of pruned points can be easily calculated resulting in upper and lower bounds on the evaluation error. We demonstrate how such data-structures and algorithms can be used to reduce the per iteration $O(n^2)$ computational cost of both KCG and PCG during learning as well as evaluation.

The inner loop computational bottleneck of KCG is in evaluating functions and calculating the kernel inner product. If we rewrite the RKHS inner product as $\langle f, g \rangle_{\mathcal{H}_k} = \sum_{i,j} \alpha_i \beta_j k(x_i, x_j) = \sum_i \alpha_i g(x_i)$, then reducing the computational complexity of RKHS function evaluations will simultaneously encompass both of these bottlenecks. Similarly, the iteration complexity of PCG is dominated by the computation of the parameter gradient. We can rewrite the parameter gradient as $\nabla_\alpha F[f] = [\nabla_k F[f](x_1), \nabla_k F[f](x_2), \ldots, \nabla_k F[f](x_n)]^T$ (see 4), reducing the complexity to that of finding $\nabla_k F[f] \in \mathcal{H}_k$ and evaluating it n times. As was the case with the algorithms as previously described, the tradeoff still balances out so that the per iteration complexity is essentially equivalent between KCG and PCG using tree-augmented function evaluation.

Noting $[f(x_1), \ldots, f(x_n)]^T = K\alpha$ suggests that the closed form quadratic line minimization for both the upper and lower bounds of RLS under either KCG or PCG can easily be augmented as well by expanding the Hessians $A_u = K^T K + \lambda K$ in the case of the upper bound and $A_l = K + \lambda I$ in the case of the lower bound. Such a tree-augmented closed form line minimization was used in the RLS experiments described below.

9 Experimental Results - Tree-Augmented Algorithms

For these experiments, in addition to using the `Soil` dataset described in section 6, we performed large scale RLS regressions on variable sized subsets of a `PugetSound` elevation map[2] using hyperparameters $\lambda = 0.1/n$ and $\sigma^2 = kN/(n\pi)$, where n is the size of the training set, and N is the size of the entire height map. In this case, we chose $N = 500 \times 500 = 250,000$, and $k = 15$. The largest resulting datasets were on the order of 100,000 points. It should be noted that the näive implementation in this case did not cache kernel evaluations in a kernel matrix as such matrices for datasets above $O(15,000)$ points proved intractable for the machine on which the experiments were performed.

Figure 2 shows that tree-augmentation significantly outperforms the näive algorithm. Extrapolating from the plot on the right, trees make possible accurate kernel learning on

[2] `http://www.cc.gatech.edu/projects/large_models/`

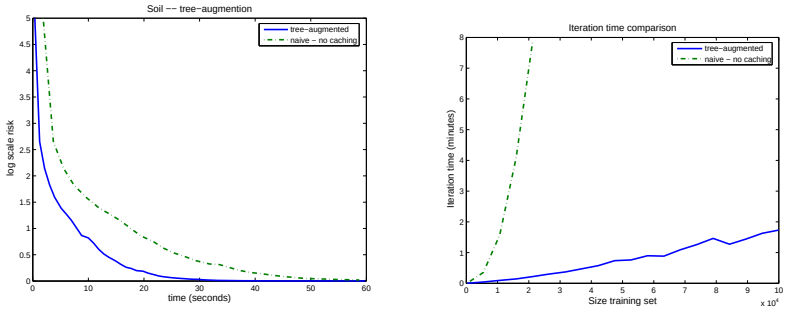

Figure 2: These plots compare the performance of tree-augmented KCG to a non-caching näive implementation on the `Soil` dataset (left) and progressively larger training set sizes from the `PugetSound` dataset (right).

datasets orders of magnitude larger than anything previously attempted.

References

[1] N. Aronszajn. Theory of reproducing kernels. In *Transactions of the American Mathematical Society*, 1950.

[2] J. Andrew Bagnell. *Learning Decision: Robustness, Uncertainty, and Approximation*. PhD thesis, Carnegie Mellon University, August 2004.

[3] Ralf Herbrich Bernhard Schölkopf and Alex J. Smola. A generalized representer theorem. In *Lecture Notes in Computer Science*, volume 2111, pages 416–426. Springer-Verlag, 2001.

[4] Mark N. Gibbs. *Bayesian Gaussian Processes for Regression and Classification*. PhD thesis, University of Cambridge, 1997.

[5] Juan Pablo Gonzalez. Carnegie Mellon University, PhD. candidate.

[6] Alexander Gray and Andrew Moore. N-body problems in statistical learning. In Todd K. Leen and Thomas G. Dietterich, editors, *Advances in Neural Information Processing Systems*. MIT Press, 2001.

[7] Sadri Hassani. *Mathematical Physics*. Springer, 1998.

[8] G. S. Kimeldorf and G. Wahba. Some results on Tchebycheffian spline functions. In *J. Math. Anal. Applic.*, volume 33, pages 82–95, 1971.

[9] David G. Luenberger. *Linear and Nonlinear Programming, Second Edition*. Springer, 2003.

[10] L. Mason, J.Baxter, P. Bartlett, and M. Frean. *Functional gradient techniques for combining hypotheses*. MIT Press, 1999.

[11] A. Moore. *Efficient Memory-Based Learning for Robot Control*. PhD thesis, University of Cambridge, November 1990.

[12] Yeo Rifkin and Poggio. Regularized least squares classification. In et. al. Suykens, editor, *Advances in Learning Theory: Methods, Models and Applications*, volume 190. IOS Press, 2003.

[13] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[14] Alex J. Smola and Bernhard Schölkopf. Sparse greedy matrix approximation for machine learning. In *International Conference on Machine Learning*, 2000.

[15] Thomas A. Manteuffel Steven F. Ashby and Paul E. Saylor. A taxonomy for conjugate gradient methods. In *SIAM Journal on Numerical Analysis*, volume 27, pages 1542–1568, 1990.

[16] J. Zhu and T. Hastie. Kernel logistic regression and the import vector machine. In *Advances in Neural Information Processing Systems*, 2001.