

Regularized Least Squares

Ryan M. Rifkin

Google, Inc.

2008

- Data points $S = \{(X_1, Y_1), \dots, (X_n, Y_n)\}$.
- We let X simultaneously refer to the set $\{X_1, \dots, X_n\}$ and to the n by d matrix whose i th row is X_i^t .

- RKHS \mathcal{H} with a positive semidefinite *kernel function* k :

linear: $k(X_i, X_j) = X_i^t X_j$

polynomial: $k(X_i, X_j) = (X_i^t X_j + 1)^d$

gaussian: $k(X_i, X_j) = \exp\left(-\frac{\|X_i - X_j\|^2}{\sigma^2}\right)$

- Define the kernel matrix K to satisfy $K_{ij} = k(X_i, X_j)$.
- Abusing notation, allow k to take and produce sets:
 - $k(X, X) = K$
 - Given an arbitrary point X_* , $k(X, X_*)$ is a column vector whose i th entry is $k(X_i, X_*)$.
- The linear kernel has special properties, which we discuss in detail later.

The RLS Setup

- Goal: Find the function $f \in \mathcal{H}$ that minimizes the weighted sum of the *total* square loss and the RKHS norm

$$\min_{f \in \mathcal{H}} \frac{1}{2} \sum_{i=1}^n (f(X_i) - Y_i)^2 + \frac{\lambda}{2} \|f\|_K^2. \quad (1)$$

- Note that in this formulation, we are minimizing the total instead of the average loss. We avoid mucking around with the factor of $1/n$, which can be folded into λ .
- This loss function “makes sense” for regression. We can also use it for binary classification, where it “makes no sense” but works great.

Applying the Representer

- The representer theorem guarantees that the solution to (1) can be written as

$$f(\cdot) = \sum_{i=1}^n c_i k(X_i, \cdot),$$

for some $c \in \mathbb{R}^n$.

- We can therefore rewrite (1) as

$$\min_{c \in \mathbb{R}^n} \frac{1}{2} \|Y - Kc\|_2^2 + \frac{\lambda}{2} \|f\|_K^2.$$

Applying the Representer Theorem, II

Consider a function of the form:

$$f(\cdot) = \sum_{i=1}^n c_i k(X_i, \cdot),$$

For such a function,

$$\begin{aligned} \|f\|_K^2 &= \langle f, f \rangle_K \\ &= \left\langle \sum_{i=1}^n c_i k(X_i, \cdot), \sum_{j=1}^n c_j k(X_j, \cdot) \right\rangle_K \\ &= \sum_{i=1}^n \sum_{j=1}^n c_i c_j \langle k(X_i, \cdot), k(X_j, \cdot) \rangle_K \\ &= \sum_{i=1}^n \sum_{j=1}^n c_i c_j k(X_i, X_j) \\ &= \mathbf{c}^t \mathbf{K} \mathbf{c} \end{aligned}$$



$$\frac{1}{2} \|Y - Kc\|_2^2 + \frac{\lambda}{2} c^t Kc$$

is clearly convex in c (why?), so we can find its minimum by setting the gradient w.r.t c to 0:

$$\begin{aligned} -K(Y - Kc) + \lambda Kc &= 0 \\ (K + \lambda I)c &= Y \\ c &= (K + \lambda I)^{-1} Y \end{aligned}$$

- We find c by solving a system of linear equations.

The RLS Solution, Comments

- The solution exists and is unique (for $\lambda > 0$).
- Define $G(\lambda) = K + \lambda I$. (Often λ is clear from context and we write G .)
- The prediction at a new test point X_* is:

$$\begin{aligned}f(X_*) &= \sum c_i k(X_i, X_*) \\ &= k(X, X_*)^t c \\ &= Y^t G^{-1} k(X, X_*).\end{aligned}$$

- The use of G^{-1} (or other inverses) is formal only. We do *not* recommend taking matrix inverses.

Solving RLS, Parameters Fixed.

- Situation: All hyperparameters fixed
- We just need to solve a single linear system

$$(K + \lambda I)c = y.$$

- The matrix $K + \lambda I$ is symmetric positive definite, so the appropriate algorithm is Cholesky factorization.
- In Matlab, the “slash” operator seems to be using Cholesky, so you can just write $c = (K + \lambda * I) \backslash Y$, but to be safe, (or in octave), I suggest $R = \text{chol}(K + \lambda * I); c = (R \backslash (R' \backslash Y)) ;$.

Solving RLS, Varying λ

- Situation: We don't know what λ to use, all other hyperparameters fixed.
- Form the eigendecomposition $K = Q\Lambda Q^t$, where Λ is diagonal with $\Lambda_{jj} \geq 0$ and $QQ^t = I$.
-

$$\begin{aligned}G &= K + \lambda I \\ &= Q\Lambda Q^t + \lambda I \\ &= Q(\Lambda + \lambda I)Q^t,\end{aligned}$$

which implies $G^{-1} = Q(\Lambda + \lambda I)^{-1}Q^t$.

Solving RLS, Varying λ , Cont'd

- $O(n^3)$ time to solve one (dense) linear system, *or* to compute the eigendecomposition (constant is maybe 4x worse). Given Q and Λ , we can find $c(\lambda)$ in $O(n^2)$ time:

$$c(\lambda) = Q(\Lambda + \lambda I)^{-1} Q^t Y,$$

noting that $(\Lambda + \lambda I)$ is diagonal.

- Finding $c(\lambda)$ for many λ 's is (essentially) free!

- We showed how to find $c(\lambda)$ quickly as we vary λ .
- But how do we decide if a given λ is “good”?
- Simplest idea: Use the training set error.
- Problem: This invariably overfits. **Don't do this!**
- Other methods are possible, but today we consider *validation*.
- Validation means checking our function's behavior on points other than the training set.

Types of Validation

- If we have a huge amount of data, we could hold back some percentage of our data (30% is typical), and use this *development* set to choose hyperparameters.
- More common is *k-fold cross-validation*, which means a couple of different things:
 - Divide your data into k equal sets S_1, \dots, S_k . For each i , train on the other $k - 1$ sets and test on the i th set.
 - A total of k times, randomly split your data into a training and test set.
- The limit of (the first kind of) k -fold validation is *leave-one-out cross-validation*.

Leave-One-Out Cross-Validation

- For each data point x_i , build a classifier using the remaining $n - 1$ data points, and measure the error at x_i .
- Empirically, this seems to be the method of choice when n is small.
- Problem: We have to build n different predictors, on data sets of size $n - 1$.
- We will now proceed to show that *for RLS, obtaining the LOO error is (essentially) free!*

Leave-One-Out CV: Notation

- Define S^i to be the data set with the i th point removed:
$$S^i = \{(X_1, Y_1), \dots, (X_{i-1}, Y_{i-1}), (X_{i+1}, Y_{i+1}), \dots, (X_n, Y_n)\}.$$
- The i th leave-one-out *value* is $f_{S^i}(X_i)$.
- The i th leave-one-out *error* is $Y_i - f_{S^i}(X_i)$.
- Define L_V and L_E to be the vectors of leave-one-out values and errors over the training set.
- $\|L_E\|_2^2$ is considered a good empirical proxy for the error on future points, and we often want to choose parameters by minimizing this quantity.

- Imagine (hallucinate) that we already know $f_{S^i}(X_i)$.
- Define the vector Y^i via

$$Y_j^i = \begin{cases} Y_j & j \neq i \\ f_{S^i}(X_i) & j = i \end{cases}$$

- Suppose we solve a Tikhonov problem with Y^i instead of Y as the labels. Then f_{S^i} is the optimizer:

$$\begin{aligned} & \frac{1}{2} \sum_{j=1}^n (Y_j^i - f(X_j))^2 + \frac{\lambda}{2} \|f\|_K^2 \\ \geq & \frac{1}{2} \sum_{j \neq i} (Y_j^i - f(X_j))^2 + \frac{\lambda}{2} \|f\|_K^2 \\ \geq & \frac{1}{2} \sum_{j \neq i} (Y_j^i - f_{S^i}(X_j))^2 + \frac{\lambda}{2} \|f_{S^i}\|_K^2 \\ = & \frac{1}{2} \sum_{j=1}^n (Y_j^i - f_{S^i}(X_j))^2 + \frac{\lambda}{2} \|f_{S^i}\|_K^2. \end{aligned}$$

- Therefore,

$$\begin{aligned}c^i &= G^{-1} Y^i \\ f_{S^i}(X_i) &= (KG^{-1} Y^i)_i\end{aligned}$$

- This is circular reasoning so far, because we need to know $f_{S^i}(X_i)$ to form Y^i in the first place.
- However, assuming we have already solved RLS for the whole training set, and we have computed $f_S(X) = KG^{-1} Y$, we can do something nice ...

$$\begin{aligned}f_{S^i}(X_i) - f_S(X_i) &= \sum_j (KG^{-1})_{ij} (Y_j^i - Y_j) \\&= (KG^{-1})_{ii} (f_{S^i}(X_i) - Y_i) \\f_{S^i}(X_i) &= \frac{f_S(X_i) - (KG^{-1})_{ii} Y_i}{1 - (KG^{-1})_{ii}} \\&= \frac{(KG^{-1} Y)_i - (KG^{-1})_{ii} Y_i}{1 - (KG^{-1})_{ii}}.\end{aligned}$$

$$L_V = \frac{KG^{-1}Y - \text{diag}_m(KG^{-1})Y}{\text{diag}_v(I - KG^{-1})},$$

$$L_E = Y - L_V$$

$$= Y + \frac{\text{diag}_m(KG^{-1})Y - KG^{-1}Y}{\text{diag}_v(I - KG^{-1})}$$

$$= \frac{\text{diag}_m(I - KG^{-1})Y}{\text{diag}_v(I - KG^{-1})} + \frac{\text{diag}_m(KG^{-1})Y - KG^{-1}Y}{\text{diag}_v(I - KG^{-1})}$$

$$= \frac{Y - KG^{-1}Y}{\text{diag}_v(I - KG^{-1})}.$$

We can simplify our expressions in a way that leads to better computational and numerical properties by noting

$$\begin{aligned}KG^{-1} &= Q\Lambda Q^t Q(\Lambda + \lambda I)^{-1} Q^t \\ &= Q\Lambda(\Lambda + \lambda I)^{-1} Q^t \\ &= Q(\Lambda + \lambda I - \lambda I)(\Lambda + \lambda I)^{-1} Q^t \\ &= I - \lambda G^{-1}.\end{aligned}$$

Substituting into our expression for L_E yields

$$\begin{aligned}L_E &= \frac{Y - KG^{-1}Y}{\text{diag}_V(I - KG^{-1})} \\&= \frac{Y - (I - \lambda G^{-1})Y}{\text{diag}_V(I - (I - \lambda G^{-1}))} \\&= \frac{\lambda G^{-1}Y}{\text{diag}_V(\lambda G^{-1})} \\&= \frac{G^{-1}Y}{\text{diag}_V(G^{-1})} \\&= \frac{c}{\text{diag}_V(G^{-1})}.\end{aligned}$$

The cost of computing L_E

- For RLS, we compute L_E via

$$L_E = \frac{c}{\text{diag}_v(G^{-1})}.$$

- We already showed how to compute $c(\lambda)$ in $O(n^2)$ time (given $K = Q\Lambda Q^t$).
- We can also compute a single entry of $G(\lambda)^{-1}$ in $O(n)$ time:

$$\begin{aligned} G_{ij}^{-1} &= (Q(\Lambda + \lambda I)^{-1} Q^t)_{ij} \\ &= \sum_{k=1}^n \frac{Q_{ik} Q_{jk}}{\Lambda_{kk} + \lambda}, \end{aligned}$$

and therefore we can compute $\text{diag}(G^{-1})$, and compute L_E , in $O(n^2)$ time.

Summary So Far

- If we can (directly) solve one RLS problem on our data, we can find a good value of λ using LOO optimization at essentially the same cost.
- When can we solve one RLS problem?
- We need to form K , which takes $O(n^2d)$ time and $O(n^2)$ memory. We need to perform a solve or an eigendecomposition of K , which takes $O(n^3)$ time.
- Usually, we run out of memory before we run out of time.
- The practical limit on today's workstations is (more-or-less) 10,000 points (using Matlab).
- How can we do more?

The Linear Case

- The linear kernel is $k(X_i, X_j) = X_i^t X_j$.
- The linear kernel offers many advantages for computation.
- Key idea: we get a decomposition of the kernel matrix for free: $K = XX^t$.
- In the linear case, we will see that we have two different computation options.

Linear kernel, linear function

With a linear kernel, the function we are learning is linear as well:

$$\begin{aligned}f(x) &= c^t k(X, x) \\ &= c^t Xx \\ &= w^t x,\end{aligned}$$

where we define the hyperplane w to be $X^t c$. We can classify new points in $O(d)$ time, using w , rather than having to compute a weighted sum of n kernel products (which will usually cost $O(nd)$ time).

Linear kernel, SVD approach, I

- Assume n , the number of points, is bigger than d , the number of dimensions. (If not, the best bet is to ignore the special properties of the linear kernel.)
- The economy-size SVD of X can be written as $X = USV^t$, with $U \in \mathbb{R}^{n \times d}$, $S \in \mathbb{R}^{d \times d}$, $V \in \mathbb{R}^{d \times d}$, $U^t U = V^t V = VV^t = I_d$, and S diagonal and positive semidefinite. (Note that $UU^t \neq I_n$).
- We will express the LOO formula directly in terms of the SVD, rather than K .

$$\begin{aligned}K &= XX^t = (USV^t)(VSU^t) = US^2U^t \\K + \lambda I &= US^2U^t + \lambda I_n \\&= \begin{bmatrix} U & U_{\perp} \end{bmatrix} \begin{bmatrix} S^2 + \lambda I_d & \\ & \lambda I_{n-d} \end{bmatrix} \begin{bmatrix} U^t \\ U_{\perp}^t \end{bmatrix} \\&= U(S^2 + \lambda I_d)U^t + \lambda U_{\perp}U_{\perp}^t \\&= U(S^2 + \lambda I_d)U^t + \lambda(I_n - UU^t) \\&= US^2U^t + \lambda I_n\end{aligned}$$

Linear kernel, SVD approach, III

$$\begin{aligned} & (K + \lambda I)^{-1} \\ = & (US^2U^t + \lambda I_n)^{-1} \\ = & \left(\begin{bmatrix} U & U_{\perp} \end{bmatrix} \begin{bmatrix} S^2 + \lambda I_d & \\ & \lambda I_{n-d} \end{bmatrix} \begin{bmatrix} U^t & \\ U_{\perp}^t & \end{bmatrix} \right)^{-1} \\ = & \begin{bmatrix} U & U_{\perp} \end{bmatrix} \begin{bmatrix} S^2 + \lambda I_d & \\ & \lambda I_{n-d} \end{bmatrix}^{-1} \begin{bmatrix} U^t & \\ U_{\perp}^t & \end{bmatrix} \\ = & U(S^2 + \lambda I)^{-1}U^t + \lambda^{-1}U_{\perp}U_{\perp}^t \\ = & U(S^2 + \lambda I)^{-1}U^t + \lambda^{-1}(I - UU^t) \\ = & U \left[(S^2 + \lambda I)^{-1} - \lambda^{-1}I \right] U^t + \lambda^{-1}I \end{aligned}$$

Linear kernel, SVD approach, IV

$$\begin{aligned} \mathbf{c} &= (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{Y} \\ &= \mathbf{U} \left[(\mathbf{S}^2 + \lambda \mathbf{I})^{-1} - \lambda^{-1} \mathbf{I} \right] \mathbf{U}^t \mathbf{Y} + \lambda^{-1} \mathbf{Y} \end{aligned}$$

$$\mathbf{G}_{ij}^{-1} = \sum_{k=1}^d U_{ik} U_{jk} [(S_{kk} + \lambda)^{-1} - \lambda^{-1}] + [i = j] \lambda^{-1}$$

$$\mathbf{G}_{ii}^{-1} = \sum_{k=1}^d U_{ik}^2 [(S_{kk} + \lambda)^{-1} - \lambda^{-1}] + \lambda^{-1}$$

$$\begin{aligned} L_E &= \frac{\mathbf{c}}{\text{diag}_v(\mathbf{G}^{-1})} \\ &= \frac{\mathbf{U} \left[(\mathbf{S}^2 + \lambda \mathbf{I})^{-1} - \lambda^{-1} \mathbf{I} \right] \mathbf{U}^t \mathbf{Y} + \lambda^{-1} \mathbf{Y}}{\text{diag}_v(\mathbf{U} \left[(\mathbf{S}^2 + \lambda \mathbf{I})^{-1} - \lambda^{-1} \mathbf{I} \right] \mathbf{U}^t + \lambda^{-1} \mathbf{I})} \end{aligned}$$

Linear kernel, SVD approach, computational costs

- We need $O(nd)$ memory to store the data in the first place. The (economy-sized) SVD also requires $O(nd)$ memory, and $O(nd^2)$ time.
- Once we have the SVD, we can compute the LOO error (for a given λ) in $O(nd)$ time.
- Compared to the nonlinear case, we have replaced an $O(n)$ with an $O(d)$, in both time and memory. If $n \gg d$, this can represent a huge savings.

Linear kernel, direct approach, I

For the linear kernel,

$$\begin{aligned}L &= \min_{c \in \mathbb{R}^n} \frac{1}{2} \|Y - Kc\|_2^2 + \frac{\lambda}{2} c^t Kc \\&= \min_{c \in \mathbb{R}^n} \frac{1}{2} \|Y - XX^t c\|_2^2 + \frac{\lambda}{2} c^t XX^t c \\&= \min_{w \in \mathbb{R}^d} \frac{1}{2} \|Y - Xw\|_2^2 + \frac{\lambda}{2} \|w\|_2^2.\end{aligned}$$

Taking the derivative with respect to w ,

$$\frac{\partial L}{\partial w} = X^t Xw - X^t Y + \lambda w,$$

and setting to zero implies

$$w = (X^t X + \lambda I)^{-1} X^t Y.$$

Linear kernel, direct approach, II

- If we are willing to give up LOO validation, we can skip the computation of c and just get w directly.
- We can work with the *Gram matrix* $X^tX \in \mathbb{R}^{d \times d}$.
- The algorithm is identical to solving a general RLS problem with kernel matrix X^tX and labels X^ty .
- Form the eigendecomposition of X^tX , in $O(d^3)$ time, form $w(\lambda)$ in $O(d^2)$ time.
- Why would we give up LOO validation? Maybe n is very large, so using a development set is good enough.

Comparing the direct and SVD approaches

- Asymptotic complexity is actually the same: it takes $O(nd^2)$ time to form the SVD of X , or to form X^tX .
- The constant in forming the SVD is about 25.
- Forming X^tX can be (relatively) easily parallelized.
- Recommendation: Use the SVD when possible, switch to the direct approach when it gets too slow.

Introducing the Subset of Regressors

- Suppose that n is too large to apply nonlinear RLS, but we need a nonlinear kernel.
- (In some circumstances, we can *explicitly* construct nonlinear feature features, such as 2nd-order polynomial, and then use the linear approach. See my ICASSP 2007 paper.)
- Another idea is the *subset of regressors* approach.

Subset of Regressors

- The representer theorem guarantees that the Tikhonov solution can be written as

$$f(\cdot) = \sum_{i=1}^n c_i k(X_i, \cdot),$$

for some $c \in \mathbb{R}^n$.

- Suppose we divide our data into two pieces, X_R and X_S , and *require a priori* that only the points in X_R have nonzero coefficients in the expansion:

$$f(\cdot) = \sum_{i=1}^{|R|} c_i k(X_i, \cdot),$$

for some $c \in \mathbb{R}^{|R|}$: this is the subset of regressors method.

Subset of Regressors, Derivation

- Defining $T = R \cup S$, we want to find

$$\min_{c \in \mathbb{R}^n} \frac{1}{2} \|Y - K_{TR}c\|_2^2 + \frac{\lambda}{2} c^t K_{RR}c$$

- Setting the derivative to zero,

$$\begin{aligned} -K_{RT}Y + K_{TR}^t K_{TR}c + \lambda K_{RR}c &= 0 \\ (K_{RT}K_{TR} + \lambda K_{RR})c &= K_{RT}Y. \end{aligned}$$

Finding $c(\lambda)$ is still cheap

Using the Cholesky factorization $K_{RR} = GG^t$,

$$\begin{aligned} & K_{RT}K_{TR} + \lambda K_{RR} \\ = & K_{RT}K_{TR} + \lambda GG^t \\ = & GG^{-1}(K_{RT}K_{TR} + \lambda GG^t)G^{-t}G^t \\ = & G(G^{-1}K_{RT}K_{TR}G^{-t} + \lambda I)G^t. \end{aligned}$$

We handle varying λ using an eigendecomposition of $G^{-1}K_{RT}K_{TR}G^{-t}$.

Can we do LOO this way? Good question ...

Random Features

- “Random Features for Large-Scale Kernel Machines,” Rahimi and Recht, NIPS 2007.
- Instead of using the kernel trick, project the data to a low-dimensional Euclidean space using a random map z s.t.:

$$k(x, y) = \langle \phi(x), \phi(y) \rangle \approx z(x)'z(y).$$

- Two different proposals for z :
 - Fourier features $\cos(\omega'x + b)$, where ω is drawn from the Fourier transform of the kernel function and b is uniform on $[0, 2\pi]$.
 - Random binning features: similarity w.r.t. randomly shifted grids of varying sizes.

“You should be asking how the answers will be used and what is *really* needed from the computation. Time and time again someone will ask for the inverse of a matrix when all that is needed is the solution of a linear system; for an interpolating polynomial when all that is needed is its values at some point; for the solution of an ODE at a sequence of points when all that is needed is the limiting, steady-state value. A common complaint is that least squares curve-fitting couldn't possibly work on *this* data set and some more complicated method is needed; in almost all such cases, least squares curve-fitting will work just fine because it is so very robust.”

Leader, Numerical Analysis and Scientific Computation