Regularized Least Squares

Charlie Frogner [1]

MIT

2011

[1] Slides mostly stolen from Ryan Rifkin (Google).

Summary

- In RLS, the Tikhonov minimization problem boils down to solving a linear system (and this is good).
- We can compute the solution for each of a bunch of λ's, by using the eigendecomposition of the kernel matrix.
- We can compute the leave-one-out error over the whole training set about as cheaply as solving the minimization problem once.
- The linear kernel allows us to do all of this when $n \gg d$.

Basics: Data

- Training set: $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$.
- Inputs: $\mathbf{X} = \{x_1, \ldots, x_n\}$.
- Labels: $\mathbf{Y} = \{y_1, \ldots, y_n\}$.

Basics: RKHS, Kernel

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

$$\begin{aligned} \text{linear:} \quad & K(x_i, x_j) = x_i^T x_j \\ \text{polynomial:} \quad & K(x_i, x_j) = (x_i^T x_j + 1)^d \\ \text{gaussian:} \quad & K(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{\sigma^2}\right) \end{aligned}$$

- Define the kernel matrix $\mathbf{K}$ to satisfy $\mathbf{K}_{ij} = K(x_i, x_j)$.
- The kernel function with one argument fixed is $K_x = K(x, \cdot)$.
- Given an arbitrary input x_*, $\mathbf{K}_{x_*}$ is a vector whose ith entry is $K(x_i, x_*)$. (So the training set $\mathbf{X}$ is assumed.)

The RLS Setup

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

$$\operatorname*{argmin}_{f\in\mathcal{H}} \frac{1}{2}\sum_{i=1}^{n}(f(x_i) - y_i)^2 + \frac{\lambda}{2}||f||^2_{\mathcal{H}}. \tag{1}$$

- This loss function makes sense for regression. We can also use it for binary classification, where it is less immediately intuitive but works great.
- Also called “ridge regression.”

Applying the Representer

Claim: We can rewrite (1) as

$$\operatorname*{argmin}_{c \in \mathbb{R}^n} \frac{1}{2} ||\mathbf{Y} - \mathbf{K}c||_2^2 + \frac{\lambda}{2} ||f||_{\mathcal{H}}^2 .$$

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

$$f(\cdot) = \sum_{j=1}^{n} c_j K_{x_j}(\cdot)$$

for some $c \in \mathbb{R}^n$.
So $\mathbf{K}c$ gives a vector whose ith element is $f(x_i)$:

$$f(x_i) = \sum_{j=1}^{n} c_j K_{x_i}(x_j) = \sum_{j=1}^{n} c_j \mathbf{K}_{ij} = (\mathbf{K}_{i,\cdot})c$$

Applying the Representer Theorem, Part II

Claim:

$$\|f\|_{\mathcal{H}}^2 = c^T \mathbf{K} c.$$

Proof:

$$f(\cdot) = \sum_{j=1}^{n} c_j K_{x_j}(\cdot),$$

so

$$\begin{aligned}
||f||_{\mathcal{H}}^2 &= \; < f, f >_{\mathcal{H}} \\
&= \left\langle \sum_{i=1}^{n} c_i K_{x_i}, \sum_{j=1}^{n} c_j K_{x_j} \right\rangle_{\mathcal{H}} \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j \left\langle K_{x_i}, K_{x_j} \right\rangle_{\mathcal{H}} \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j K(x_i, x_j) = c^t \mathbf{K} c
\end{aligned}$$

The RLS Solution

- Putting it all together, the RLS problem is:

$$\underset{f\in\mathcal{H}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{Y} - \mathbf{K}c||_2^2 + \frac{\lambda}{2}c^T\mathbf{K}c$$

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

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

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

The RLS Solution, Comments

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

$$\begin{aligned} f(x_*) &= \sum_{j=1}^{n} c_j \mathbf{K}_{x_j}(x_*) \\ &= \mathbf{K}_{x_*} c \\ &= \mathbf{K}_{x_*} \mathbf{G}^{-1} \mathbf{Y} \end{aligned}$$

- The use of $\mathbf{G}^{-1}$ (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

$$(\mathbf{K} + \lambda I)c = \mathbf{Y}.$$

- The matrix $\mathbf{K} + \lambda I$ 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+l*I)\Y`, but to be safe, (or in octave), I suggest `R = chol(K+l*I); c = (R\(R'\Y));`.

Solving RLS, Varying λ

- Situation: We don't know what λ to use, all other hyperparameters fixed.
- Is there a more efficent method than solving $c(\lambda) = (\mathbf{K} + \lambda I)^{-1}\mathbf{Y}$ anew for each λ?
- Form the eigendecomposition $\mathbf{K} = \mathbf{Q}\Lambda\mathbf{Q}^T$, where Λ is diagonal with $\Lambda_{ii} \geq 0$ and $\mathbf{Q}\mathbf{Q}^T = I$.
- Then

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

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

Solving RLS, Varying λ

- Situation: We don't know what λ to use, all other hyperparameters fixed.
- Is there a more efficent method than solving $c(\lambda) = (\mathbf{K} + \lambda I)^{-1}\mathbf{Y}$ anew for each λ?
- Form the eigendecomposition $\mathbf{K} = \mathbf{Q}\Lambda\mathbf{Q}^T$, where Λ is diagonal with $\Lambda_{ii} \geq 0$ and $\mathbf{Q}\mathbf{Q}^T = I$.
- Then

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

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

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 $\mathbf{Q}$ and Λ, we can find $c(\lambda)$ in $O(n^2)$ time:

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

  noting that $(\Lambda + \lambda I)$ is diagonal.
- Finding $c(\lambda)$ for many λ's is (essentially) free!

Validation

- 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.

Validation

- 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 ith 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 ith point removed:

$$S^i = \{(x_1, y_1), \ldots, (x_{i-1}, y_{i-1}), \text{*poof*}, (x_{i+1}, y_{i+1}), \ldots, (x_n, y_n)\}$$

- The ith leave-one-out *value* is $f_{S^i}(x_i)$.
- The ith 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.

L_E derivation, I

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

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

L_E derivation, II

Claim: Solving RLS using $\mathbf{Y}^i$ gives us f_{S^i}, i.e.

$$f_{S^i} = \operatorname*{argmin}_{f \in \mathcal{H}} \frac{1}{2} \sum_{j=1}^{n} (y_j^i - f(x_j))^2 + \frac{\lambda}{2} \|f\|_{\mathcal{H}}^2 = \textbf{(*)}.$$

Proof:

$$\begin{array}{rc} & \textbf{(1)} = (y_i^i - f(x_i))^2 \geq 0 \ \forall f \\ \text{and} & (y_i^i - f_{S^i}(x_i))^2 = (f_{S_i}(x_i) - f_{S_i}(x_i))^2 = 0 \\ \Rightarrow & f_{S^i} \text{ minimizes } \textbf{(1)} \\ & f_{S_i} \text{ also minimizes } \sum_{j \neq i} (y_j^i - f(x_j))^2 + \frac{\lambda}{2} \|f\|_{\mathcal{H}}^2 = \textbf{(2)} \\ \Rightarrow & f_{S_i} \text{ minimizes } \textbf{(*)} = \textbf{(1)} + \textbf{(2)} \end{array}$$

L_E derivation, III

- Therefore,

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

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

L_E derivation, IV

$$\begin{aligned}
f_{S^i}(x_i) - f_S(x_i) &= \sum_j (\mathbf{K}\mathbf{G}^{-1})_{ij}(y_j^i - y_j) \\
&= (\mathbf{K}\mathbf{G}^{-1})_{ii}(f_{S^i}(x_i) - y_i) \\
f_{S^i}(x_i) &= \frac{f_S(x_i) - (\mathbf{K}\mathbf{G}^{-1})_{ii} y_i}{1 - (\mathbf{K}\mathbf{G}^{-1})_{ii}} \\
&= \frac{(\mathbf{K}\mathbf{G}^{-1}\mathbf{Y})_i - (\mathbf{K}\mathbf{G}^{-1})_{ii} y_i}{1 - (\mathbf{K}\mathbf{G}^{-1})_{ii}}.
\end{aligned}$$

L_E derivation, V

$$\begin{aligned}
L_V &= \frac{\mathbf{K}\mathbf{G}^{-1}\mathbf{Y} - \text{diag}_m(\mathbf{K}\mathbf{G}^{-1})\mathbf{Y}}{\text{diag}_v(I - \mathbf{K}\mathbf{G}^{-1})}, \\
L_E &= \mathbf{Y} - L_V \\
&= \mathbf{Y} + \frac{\text{diag}_m(\mathbf{K}\mathbf{G}^{-1})\mathbf{Y} - \mathbf{K}\mathbf{G}^{-1}\mathbf{Y}}{\text{diag}_v(I - \mathbf{K}\mathbf{G}^{-1})} \\
&= \frac{\text{diag}_m(I - \mathbf{K}\mathbf{G}^{-1})\mathbf{Y}}{\text{diag}_v(I - \mathbf{K}\mathbf{G}^{-1})} + \frac{\text{diag}_m(\mathbf{K}\mathbf{G}^{-1})\mathbf{Y} - \mathbf{K}\mathbf{G}^{-1}\mathbf{Y}}{\text{diag}_v(I - \mathbf{K}\mathbf{G}^{-1})} \\
&= \frac{\mathbf{Y} - \mathbf{K}\mathbf{G}^{-1}\mathbf{Y}}{\text{diag}_v(I - \mathbf{K}\mathbf{G}^{-1})}.
\end{aligned}$$

L_E derivation, VI

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

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

L_E derivation, VII

Substituting into our expression for L_E yields

$$\begin{aligned} L_E &= \frac{\mathbf{Y} - \mathbf{K}\mathbf{G}^{-1}\mathbf{Y}}{\operatorname{diag}_v(I - \mathbf{K}\mathbf{G}^{-1})} \\ &= \frac{\mathbf{Y} - (I - \lambda\mathbf{G}^{-1})\mathbf{Y}}{\operatorname{diag}_v(I - (I - \lambda\mathbf{G}^{-1}))} \\ &= \frac{\lambda\mathbf{G}^{-1}\mathbf{Y}}{\operatorname{diag}_v(\lambda\mathbf{G}^{-1})} \\ &= \frac{\mathbf{G}^{-1}\mathbf{Y}}{\operatorname{diag}_v(\mathbf{G}^{-1})} \\ &= \frac{c}{\operatorname{diag}_v(\mathbf{G}^{-1})}. \end{aligned}$$

The cost of computing L_E

- For RLS, we compute L_E via

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

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

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

  and therefore we can compute diag($\mathbf{G}^{-1}$), 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? (I.e. what are the bottlenecks?)
- We need to form $\mathbf{K}$, which takes $O(n^2 d)$ time and $O(n^2)$ memory. We need to perform a solve or an eigendecomposition of $\mathbf{K}$, 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?

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? (I.e. what are the bottlenecks?)
- We need to form $\mathbf{K}$, which takes $O(n^2 d)$ time and $O(n^2)$ memory. We need to perform a solve or an eigendecomposition of $\mathbf{K}$, 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: $\mathbf{K} = \mathbf{X}\mathbf{X}^T$.
- 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:

$$
\begin{aligned}
f(x_*) &= \mathbf{K}_{x_*} c \\
&= x_*^T \mathbf{X}^T c \\
&= x_*^T w,
\end{aligned}
$$

where we define the hyperplane w to be $\mathbf{X}^T c$. 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 $\mathbf{X}$ can be written as $\mathbf{X} = USV^T$, 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 $\mathbf{K}$.

Linear kernel, SVD approach, II

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

Linear kernel, SVD approach, III

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

Linear kernel, SVD approach, IV

$$
\begin{aligned}
c &= (\mathbf{K} + \lambda I)^{-1}\mathbf{Y} \\
&= U\left[(S^2 + \lambda I)^{-1} - \lambda^{-1} I\right] U^T\mathbf{Y} + \lambda^{-1}\mathbf{Y} \\
G^{-1}_{ij} &= \sum_{k=1}^{d} U_{ik}U_{jk}[(S_{kk} + \lambda)^{-1} - \lambda^{-1}] + [i = j]\lambda^{-1} \\
G^{-1}_{ii} &= \sum_{k=1}^{d} U_{ik}^2[(S_{kk} + \lambda)^{-1} - \lambda^{-1}] + \lambda^{-1} \\
L_E &= \frac{c}{\operatorname{diag}_v(G^{-1})} \\
&= \frac{U\left[(S^2 + \lambda I)^{-1} - \lambda^{-1} I\right] U^T\mathbf{Y} + \lambda^{-1}\mathbf{Y}}{\operatorname{diag}_v(U\left[(S^2 + \lambda I)^{-1} - \lambda^{-1} I\right] U^T + \lambda^{-1} I)}
\end{aligned}
$$

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 $n >> d$, this can represent a huge savings.

Linear kernel, direct approach, I

For the linear kernel,

$$\begin{aligned} L &= \underset{c \in \mathbb{R}^n}{\operatorname{argmin}} \frac{1}{2}||\mathbf{Y} - \mathbf{K}c||_2^2 + \frac{\lambda}{2} c^T \mathbf{K} c \\ &= \underset{c \in \mathbb{R}^n}{\operatorname{argmin}} \frac{1}{2}||\mathbf{Y} - \mathbf{X}\mathbf{X}^T c||_2^2 + \frac{\lambda}{2} c^T \mathbf{X}\mathbf{X}^T c \\ &= \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{2}||\mathbf{Y} - \mathbf{X}w||_2^2 + \frac{\lambda}{2} ||w||_2^2. \end{aligned}$$

Taking the derivative with respect to w,

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

and setting to zero implies

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

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* $\mathbf{X}^T\mathbf{X} \in \mathbb{R}^{d\times d}$.
- The algorithm is identical to solving a general RLS problem with kernel matrix $\mathbf{X}^T\mathbf{X}$ and labels $\mathbf{X}^T y$.
- Form the eigendecomposition of $\mathbf{X}^T\mathbf{X}$, 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.

Summary

- In RLS, the Tikhonov minimization problem boils down to solving a linear system:

$$\operatorname*{argmin}_{f\in\mathcal{H}} \frac{1}{2}\sum_{i=1}^{n}(y_i - f(x_i))^2 + \frac{\lambda}{2}||f||_{\mathcal{H}}^2 = \mathbf{K}_{(\cdot)}c$$

  where $(\mathbf{K} + \lambda I)c = \mathbf{Y}$.
- We can (more) cheaply compute $c(\lambda)$ for a bunch of λ's, by using the eigendecomposition of the kernel matrix: $\mathbf{K} = \mathbf{Q}\Lambda\mathbf{Q}^T$.
- We can compute the leave-one-out error over the whole training set about as cheaply as solving for c once.
- The linear kernel allows us to do all of this when $n \gg d$.

Parting Shot

“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