Regularized Least Squares

9.520 Class 04, 21 February 2006

Ryan Rifkin

Plan

- Introduction to Regularized Least Squares
- Computation: General RLS
- Large Data Sets: Subset of Regressors
- Computation: Linear RLS

Regression

We have a training set $S = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_\ell, \mathbf{y}_\ell)\}$. The $\mathbf{y}_i$ are *real-valued*. The goal is to learn a function f to predict the y values associated with new observed x values.

Our Friend Tikhonov Regularization

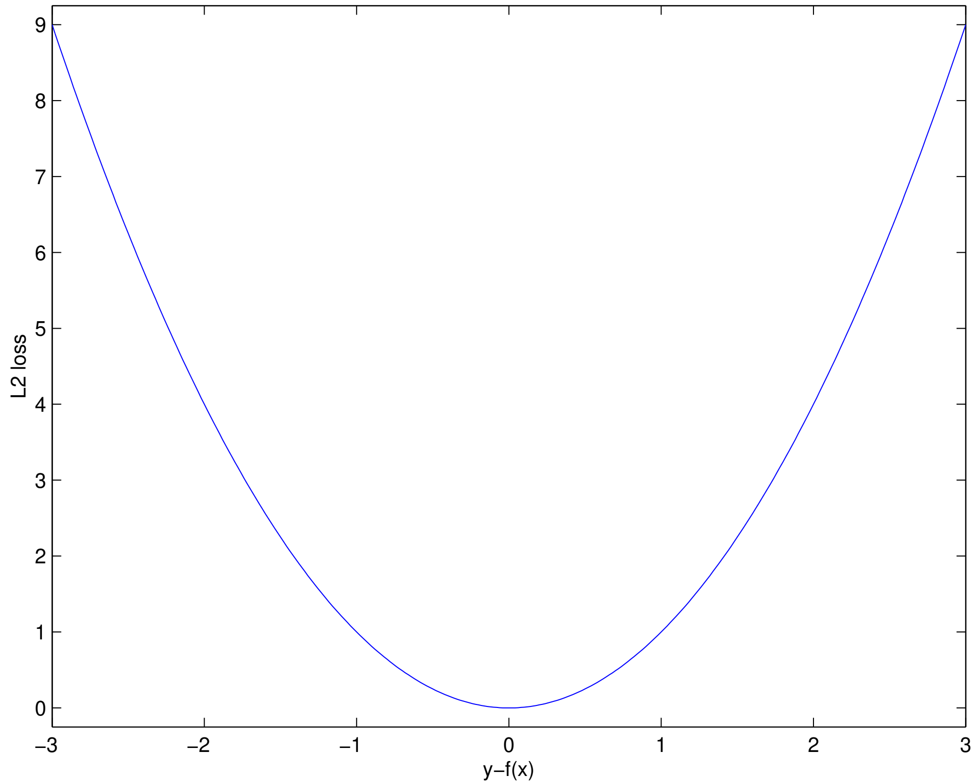
We pose our regression task as the Tikhonov minimization problem:

$$f = \operatorname{arg}\min_{f \in \mathcal{H}} \frac{1}{2} \sum_{i=1}^{\ell} V(f(\mathbf{x}_i), \mathbf{y}_i) + \frac{\lambda}{2} \|f\|_K^2$$

To fully specify the problem, we need to choose a loss function V and a kernel function K.

The Square Loss

For regression, a natural choice of loss function is the square loss $V(f(\mathbf{x}), y) = (f(\mathbf{x}) - y)^2$.



Substituting In The Square Loss

Using the square loss, our problem becomes

$$f = \arg\min_{f \in \mathcal{H}} \frac{1}{2} \sum_{i=1}^{\ell} (f(x_i) - y_i)^2 + \frac{\lambda}{2} \|f\|_K^2.$$

The Return of the Representer Theorem

Theorem. The solution to the Tikhonov regularization problem

$$\min_{f \in \mathcal{H}} \frac{1}{2} \sum_{i=1}^{\ell} V(y_i, f(\mathbf{x}_i)) + \frac{\lambda}{2} \|f\|_K^2$$

can be written in the form

$$f = \sum_{i=1}^{\ell} c_i K(\mathbf{x}_i, \cdot).$$

This theorem is exceedingly useful — it says that to solve the Tikhonov regularization problem, we need only find the best function of the form $f = \sum_{i=1}^{\ell} c_i \mathbf{K}(\mathbf{x}_i)$. Put differently, all we have to do is find the c_i.

Applying the Representer Theorem, I

NOTATION ALERT!!! We use the symbol K for the kernel function, and boldface $\mathbf{K}$ for the ℓ-by-ℓ matrix:

$$\mathbf{K}_{ij} \equiv K(x_i, x_j)$$

Using this definition, consider the output of our function

$$f = \sum_{i=1}^{\ell} c_i K(\mathbf{x}_i, \cdot).$$

at the training point x_j:

$$\begin{aligned} f(\mathbf{x}_j) &= \sum_{i=1}^{\ell} K(\mathbf{x}_i, \mathbf{x}_j) c_i \\ &= (\mathbf{K}\mathbf{c})_j \end{aligned}$$

Using the Norm of a "Represented" Function

A function in the RKHS with a finite representation

$$f = \sum_{i=1}^{\ell} c_i K(\mathbf{x}_i, \cdot),$$

satisfies

$$\begin{aligned}
\|f\|_k^2 &= \left\langle \sum_{i=1}^{\ell} c_i K(\mathbf{x}_i, \cdot), \sum_{j=1}^{\ell} c_j K(\mathbf{x}_j, \cdot) \right\rangle \\
&= \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} c_i c_j \left\langle K(\mathbf{x}_i, \cdot), K(\mathbf{x}_j, \cdot) \right\rangle \\
&= \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} c_i c_j K(\mathbf{x}_i, \mathbf{x}_j) \\
&= \mathbf{c}^t \mathbf{K} \mathbf{c}.
\end{aligned}$$

The RLS Problem

Substituting, our Tikhonov minimization problem becomes:

$$\min_{\mathbf{c}\in\mathbb{R}^\ell} \frac{1}{2}\|\mathbf{Kc}-\mathbf{y}\|_2^2 + \frac{\lambda}{2}\mathbf{c}^T K\mathbf{c}.$$

Solving the Least Squares Problem, I

We are trying to minimize

$$g(c) = \frac{1}{2}\|\mathbf{Kc} - \mathbf{y}\|_2^2 + \frac{\lambda}{2}\mathbf{c}^T K\mathbf{c}.$$

This is a **convex**, **differentiable** function of c, so we can minimize it simply by taking the derivative with respect to c, then setting this derivative to 0.

$$\frac{\partial g(c)}{\partial c} = \mathbf{K}(\mathbf{Kc} - \mathbf{y}) + \lambda\mathbf{Kc}.$$

Solving the Least Squares Problem, II

Setting the derivative to 0,

$$
\begin{array}{rl}
 & \dfrac{\partial g(c)}{\partial c} = \mathbf{K}(\mathbf{K}\mathbf{c} - \mathbf{y}) + \lambda \mathbf{K}\mathbf{c} = 0 \\
\rightarrow & \mathbf{K}(\mathbf{K}\mathbf{c}) + \lambda \mathbf{K}\mathbf{c} = \mathbf{K}\mathbf{y} \\
\text{``}\rightarrow\text{''} & \mathbf{K}\mathbf{c} + \lambda \mathbf{c} = \mathbf{y} \\
\rightarrow & (\mathbf{K} + \lambda I)\mathbf{c} = \mathbf{y} \\
\rightarrow & \mathbf{c} = (\mathbf{K} + \lambda I)^{-1}\mathbf{y}
\end{array}
$$

The matrix $\mathbf{K} + \lambda I$ is positive definite and will be well-conditioned if λ is not too small.

Leave-One-Out Values

Recalling that $S = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_\ell, \mathbf{y}_\ell)\}$, we define f_S to be the solution to the RLS problem with training set S. We define

$$\begin{aligned} S^i &= \{S \backslash \mathbf{x}_i\} \\ &= \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_{i-1}, \mathbf{y}_{i-1}), (\mathbf{x}_{i+1}, \mathbf{y}_{i+1}), \ldots, (\mathbf{x}_\ell, \mathbf{y}_\ell)\}, \end{aligned}$$

the training set with the ith point removed.

We call $f_{S^i}(\mathbf{x}_i)$ the ith LOO *value*, and $\mathbf{y}_i - f_{S^i}(\mathbf{x}_i)$ the ith LOO *error*. Let $LOOV$ and $LOOE$ be vectors whose ith entries are the ith LOO value and error.

Key Intuition: if $\|LOOE\|$ is small, we will generalize well.

The Leave-One-Out Formula

Remarkably, for RLS, there is a closed form formula for $LOOE$. Defining $\mathbf{G}(\lambda) = (\mathbf{K} + \lambda I)^{-1}$, we have:

$$
\begin{aligned}
LOOE &= \frac{\mathbf{G}^{-1}\mathbf{y}}{\mathrm{diag}(\mathbf{G}^{-1})} \\
&= \frac{\mathbf{c}}{\mathrm{diag}(\mathbf{G}^{-1})}.
\end{aligned}
$$

Proof: Later, Blackboard.

Computing: Naive Approach

Suppose I want to minimize $\|LOOE\|$, testing p different values of λ.

I form $\mathbf{K}$, which takes $O(n^2 d)$ time (I assume d-dimensional data and linear-time kernels throughout).

For each λ, I form $\mathbf{G}$, I form $\mathbf{G}^{-1}$ ($O(n^3)$ time), and compute $\mathbf{c} = \mathbf{G}^{-1} y$ and diag($\mathbf{G}^{-1}$).

Over p values of λ, I will pay $O(pn^3)$ time.

We can do much better...

Computing: Eigendecomposing $\mathbf{K}$

We form the eigendecomposition $\mathbf{K} = Q\Lambda Q^t$, where Λ is diagonal with $\Lambda_{ii} \geq 0$ and $QQ^t = I$.

Key point:

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

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

Forming the eigendecomposition takes $O(n^3)$ time (in practice).

Computing $\mathbf{c}$ and $LOOE$ efficiently

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

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

Given the eigendecomposition, I can compute $\mathbf{c}, \mathrm{diag}(\mathbf{G}^{-1})$, and $LOOE$ in $O(n^2)$ time. Over p values of λ, I pay only $O(n^3 + pn^2)$. If $p < n$, searching for a good λ is effectively free!

Nonlinear RLS, Suggested Approach

- 1. Form the eigendecomposition $\mathbf{K} = Q \Lambda Q^t$.

- 2. For *each* value of λ over a logarithmically spaced grid, compute $c = Q(\Lambda + \lambda I)^{-1} Q^t \mathbf{y}$ and $\text{diag}(\mathbf{G}^{-1})$ using the formula for the last slide. Form $LOOE$, a vector whose ith entry is $\frac{\mathbf{c}_i}{\text{diag}(\mathbf{G}^{-1})_i}$.

- 3. Choose the λ that minimizes $\|LOOE\|$ in some norm (I use L_2).

- 4. Given that c, regress a new test point x with $f(x) = \sum_i \mathbf{c}_i K(\mathbf{x}_i, x)$.

Limits of RLS

RLS has proved very accurate. There are two computational problems:

- Training: We need $O(n^2)$ space (to store $\mathbf{K}$), and $O(n^3)$ time (to eigendecompose $\mathbf{K}$)

- Testing: Testing a new point x takes $O(nd)$ time to compute the n kernel products in $f(x) = \sum_i K(x, \mathbf{x}_i)$.

Next class, we will see that an SVM has a *sparse* solution, which gives us large constant factor (but important in practice!) improvements for both the training and testing problems.

Can we do better, sticking with RLS?

First Idea: Throw Away Data

Suppose that we throw away all but M of our data points, where $M << \ell$. Then we only need time $M^2 d$ to form our new, smaller kernel matrix, and we only need time $O(M^3)$ to solve the problem. Great, isn't it?

Well, if we have too much data to begin with (say 1,000,000 points in 3 dimensions) this will work just fine. In general, we will lose accuracy.

Subset of Regressors

Suppose, instead of throwing away data, we restrict our hypothesis space further. Instead of allowing functions of the form

$$f(x) = \sum_{i=1}^{\ell} \mathbf{c}_i K(\mathbf{x}_i, x),$$

we only allow

$$f(x) = \sum_{i=1}^{M} \mathbf{c}_i K(\mathbf{x}_i, x),$$

where $M << \ell$. In other words, we only allow the first M points to have non-zero c_i. However, we still measure the loss at all ℓ points.

Subset of Regressors, Cont'd

Suppose we define $\mathbf{K}_{MM}$ to be the kernel matrix on just the M points we're using to represent our function, and $\mathbf{K}_{ML}$ to be the kernel product between those M points and the entire dataset, we can derive (homework) that the minimization problem becomes:

$$(\mathbf{K}_{ML}\mathbf{K}_{LM} + \lambda\mathbf{K}_{MM})\mathbf{c} = \mathbf{K}_{ML}y,$$

which is again an M-by-M system.

Various authors have reported good results with this or similar, but the jury is still out (class project!). Sometimes called Rectangular Method.

λ is Still Free

To solve

$$(\mathbf{K}_{ML}\mathbf{K}_{LM} + \lambda\mathbf{K}_{MM})\mathbf{c} = \mathbf{K}_{ML}y,$$

consider a Cholesky factorization $\mathbf{K}_{MM} = GG^t$:

$$\begin{aligned}
& (\mathbf{K}_{ML}\mathbf{K}_{LM} + \lambda\mathbf{K}_{MM})\mathbf{c} = \mathbf{K}_{ML}y \\
\rightarrow\ & (\mathbf{K}_{ML}\mathbf{K}_{LM} + \lambda GG^t)\mathbf{c} = \mathbf{K}_{ML}y \\
\rightarrow\ & (\mathbf{K}_{ML}\mathbf{K}_{LM} + \lambda GG^t)G^{-t}G^t\mathbf{c} = \mathbf{K}_{ML}y \\
\rightarrow\ & (\mathbf{K}_{ML}\mathbf{K}_{LM}G^{-t} + \lambda G)G^t\mathbf{c} = \mathbf{K}_{ML}y \\
\rightarrow\ & G(G^{-1}\mathbf{K}_{ML}\mathbf{K}_{LM}G^{-t} + \lambda I)G^t\mathbf{c} = \mathbf{K}_{ML}y,
\end{aligned}$$

and we use the “standard” RLS free-λ algorithm on an eigendecomposition of $G^{-1}\mathbf{K}_{ML}\mathbf{K}_{LM}G^{-t}$.

Linear Kernels

An important special case is the linear kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j.$$

The solution function f simplifies as:

$$\begin{aligned} f(x) &= \sum \mathbf{c}_i \mathbf{x}_i \cdot x \\ &= (\sum \mathbf{c}_i \mathbf{x}_i) \cdot x \\ &\equiv w^t \cdot x. \end{aligned}$$

We can evaluate f in time d rather than ℓd.

This is a general property of Tikhonov regularization with a linear kernel, not related to the use of the square loss.

Linear RLS

In the linear case, $\mathbf{K} = \mathbf{X}^t\mathbf{X}$ ($\mathbf{x}_i$ is the ith column of $\mathbf{X}$). Note that $w = \mathbf{X}\mathbf{c}$.

We work with an "economy-sized SVD" $\mathbf{X} = U\Sigma V^t$, where U is $d \times d$ orthogonal, Σ is $d \times d$ diagonal spd, and V is $n \times d$ with orthogonal columns ($V^tV = I$).

$$
\begin{aligned}
w &= \mathbf{X}(\mathbf{X}^t\mathbf{X} + \lambda I)^{-1}\mathbf{y} \\
&= U\Sigma V^t(V\Sigma^2V^t + \lambda I)^{-1}\mathbf{y} \\
&= U\Sigma(\Sigma^2 + \lambda I)^{-1}V^t\mathbf{y}.
\end{aligned}
$$

We need $O(nd^2)$ time and $O(nd)$ memory to form the SVD. Then we can get $w(\lambda)$ in $O(d^2)$ time. Very fast.

Linear RLS, Sparse Data

Suppose that d, the number of dimensions, is enormous, and that n is also large, but the data are *sparse*: each dimension has only a few non-zero entries. Example: document classification. We have dimensions for each word in a "dictionary". Tens of thousands of words, but only a few hundred appear in a given document.

The Conjugate Gradient Algorithm

The conjugate gradient algorithm is a popular algorithm for solving linear systems. For this class, we need to know that CG is an **iterative** algorithm. The major operation is multiplying taking a matrix-vector product Av. A need not be supplied explicitly.

CG is the method of choice when there is a way to multiply by A "quickly".

CG and Sparse Linear RLS

Remember, we are trying to solve

$$
\begin{aligned}
& (\mathbf{K} + \lambda I)\mathbf{c} = y \\
\rightarrow \quad & (\mathbf{X}^t\mathbf{X} + \lambda I)\mathbf{c} = y.
\end{aligned}
$$

$\mathbf{K}$ is too big to write down. $\mathbf{X}$ is "formally" too big, so we can't take its SVD, but it's sparse. We can use CG, because we can form the matrix vector-product $(\mathbf{X}^t\mathbf{X}+\lambda I)\mathbf{c}$ quickly:

$$(\mathbf{X}^t\mathbf{X} + \lambda I)\mathbf{c} = \mathbf{X}^t(\mathbf{X}\mathbf{c}) + \lambda\mathbf{c}$$

Cost per iteration: $2\bar{d}\ell$, where $\bar{d}$ is the **average** number of nonzero entries per data point.

Square-Loss Classification

There is nothing to formally stop us for using the above algorithm for **classification**. By doing so, we are essentially treating our classification problem as a regression problem with y values of 1 or -1.

How well do you think this will work?