9.520: Statistical Learning Theory and Applications February 16th, 2010

Regularized Least Squares

Lecturer: Charlie Frogner *Scribe: Shay Maymon*

1 Introduction

Tikhonov regularization which was introduced last time is stated as the minimization of the following objective function with respect to $f \in \mathcal{H}$

$$\sum_{i=1}^{n} V(y_i, f(x_i)) + \lambda \|f\|_{\mathcal{H}}^2. \tag{1}$$

$\{(x_i, y_i)\}_{i=1}^n$ are the given data points, $V(\cdot)$ represents the loss function indicating the penalty we pay for predicting $f(x_i)$ when the true value is y_i, and $\|f\|_{\mathcal{H}}^2$ is a Hilbert space norm regularization term with a regularization parameter λ which controls the stability of the solution and trades-off regression accuracy for a function with small norm in RKHS $\mathcal{H}$.

Denote by S the training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ where each x_i is a d-dim column vector. We let X refer to the n by d matrix whose i^{th} row is x_i^T and $Y = [y_1, y_2, \ldots, y_n]^T$ refer to a column vector of labels. We assume a positive semidefinite kernel function k, which generalizes the notion of dot product in a Reproducing Kernel Hilbert Space (RKHS). Commonly used kernels include:

$$\begin{aligned}
\text{Linear} &: k(x_i, x_j) = x_i^T x_j \\
\text{Polynomial} &: k(x_i, x_j) = (x_i^T x_j + 1)^d \\
\text{Gaussian} &: k(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{\sigma^2}\right).
\end{aligned}$$

We define the kernel matrix K such that $K_{ij} = k(x_i, x_j)$. Abusing notation, we allow the kernel function to take multiple data points and produce a matrix, i.e. $k(X, X) = K$, and given an arbitrary point x, $k(X, x)$ is a column vector whose i^{th} entry is $k(x_i, x)$.

According to the representer theorem, the optimal function f_s which minimizes (1) has to be of the form

$$f_s^\lambda(x) = \sum_{i=1}^{n} c_i k(x_i, x) \tag{2}$$

for some $c_i \in \mathbb{R}$, where k represents the positive semidefinite reproducing kernel function associated with $\mathcal{H}$. This result is easily shown by decomposing an arbitrary function $f \in \mathcal{H}$ into

$$f = f_s + f_s^\perp \tag{3}$$

where $f_s = \sum_{i=1}^n \alpha_i k(x_i, \cdot)$ and $f_s^\perp$ belongs to the space orthogonal to that spanned by $k(x_i, \cdot)$. The first term of the objective function which corresponds to the loss function does not depend on $f_s^\perp$. This is shown by using the reproducing property, i.e. $f(x) = \langle f, k(x, \cdot)\rangle_{\mathcal{H}} \, \forall f \in \mathcal{H}$, and the fact that $f_s^\perp$ is in the space orthogonal to that spanned by $\{k(x_i, \cdot)\}$, i.e.

$$\begin{aligned}
f(x_i) &= \langle f, k(x_i, \cdot)\rangle_{\mathcal{H}} \\
&= \left\langle f_s + f_s^\perp, k(x_i, \cdot)\right\rangle_{\mathcal{H}} \\
&= \langle f_s, k(x_i, \cdot)\rangle_{\mathcal{H}} + \left\langle f_s^\perp, k(x_i, \cdot)\right\rangle_{\mathcal{H}} = f_s(x_i).
\end{aligned} \tag{4}$$

Furthermore, using the fact that $f_s \perp f_s^{\perp}$, the regularization term reduces to

$$\|f\|_{\mathcal{H}}^2 = \langle f, f \rangle_{\mathcal{H}} = \|f_s\|_{\mathcal{H}}^2 + \|f_s^{\perp}\|_{\mathcal{H}}^2. \tag{5}$$

Thus, to minimize (1) we set $\|f_s^{\perp}\|_{\mathcal{H}}^2$ to zero so $f_s^{\perp} = 0$ and $f \in span\{k(x_i, \cdot)\}$.

A very useful algorithm and simple to derive is Regularized Least Squares (RLS) in which the square loss $V(y_i, f(x_i)) = (y_i - f(x_i))^2$ is used and the Tikhonov minimization boils down to solving a linear system. We will first show the derivation of the RLS algorithm and then discuss how to find good values for the regularization parameter λ. We will show an efficient way for solving the RLS problem for different values of λ which would be as cheap as solving the minimization problem once. We will also discuss about ways to check how good the function is for a specific value of λ.

2 Solving RLS for a Single Value of λ

RLS is a Tikhonov regularization with a square loss function, i.e.

$$\arg\min_{f \in \mathcal{H}} \frac{1}{2} \sum_{i=1}^{n} (f(x_i) - y_i)^2 + \frac{\lambda}{2} ||f||_{\mathcal{H}}^2. \tag{6}$$

For simplicity of derivation we are minimizing the total loss rather than the average loss as in Tikhonov regularization. The factor $1/n$ on the loss can be folded into the regularization factor λ. Also, the multiplication by a factor of $\frac{1}{2}$ is for simplicity and will obviously not change the minimizer. The square loss makes more sense for regression than for classification, however, as we will see later it works great also for classification.

Using the representer theorem, we can write the solution to (6) as

$$f(\cdot) = \sum_{j=1}^{n} c_j k(\cdot, x_j) \tag{7}$$

for some $c_j \in \mathbb{R}$. Specifically, $f(x_i)$ can be represented as

$$f(x_i) = \sum_{j=1}^{n} c_j k(x_i, x_j) = (K_{i,.}) \cdot c \tag{8}$$

where $(K_{i,.})$ is the i^{th} row of the kernel matrix, and $c \in \mathbb{R}^n$ is an n-dim column vector. Using (8), the minimization in (6) can be rewritten as

$$\arg\min_{c \in \mathbb{R}^n} \frac{1}{2} \|Y - K \cdot c\|_2^2 + \frac{\lambda}{2} ||f||_{\mathcal{H}}^2. \tag{9}$$

Using again the representer theorem for the regularization term, we obtain

$$\begin{aligned} \|f\|_{\mathcal{H}}^2 &= \langle f, f \rangle_{\mathcal{H}} && (10) \\ &= \left\langle \sum_{i=1}^{n} c_i k(\cdot, x_i), \sum_{j=1}^{n} c_j k(\cdot, x_j) \right\rangle_{\mathcal{H}} && (11) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j \langle k(\cdot, x_i), k(\cdot, x_j) \rangle_{\mathcal{H}} && (12) \end{aligned}$$

and by applying the reproducing property $\langle f, k(\cdot, x_j) \rangle_{\mathcal{H}} = f(x_j)$ to $k(\cdot, x_i) \in \mathcal{H}$, we obtain

$$\|f\|_{\mathcal{H}}^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) = c^T K c. \tag{13}$$

Note that (13) is specifically true for the minimizer of the Tikhonov functional and not for all $f \in \mathcal{H}$. Substituting (13) into (9), we obtain

$$\arg\min_{c\in\mathbb{R}^n} \frac{1}{2}\|Y - Kc\|_2^2 + \frac{\lambda}{2}c^T Kc \tag{14}$$

which is a convex function of c since K is a positive semidefinite matrix. The solution can be obtained by setting the derivative with respect to c to zero, i.e.

$$\begin{aligned}\frac{\partial}{\partial c}\left\{\frac{1}{2}(Y-Kc)^T(Y-Kc)+\frac{\lambda}{2}c^TKc\right\} &= -(Y-Kc)^TK+\lambda c^TK = \\ &= \left((K+\lambda I)\,c-Y\right)^TK = 0^T. \end{aligned}\tag{15}$$

Since K is positive semidefinite, $K + \lambda I$ is positive definite for all $\lambda > 0$ and therefore the optimal solution is

$$c = G^{-1}(\lambda)Y \tag{16}$$

where $G(\lambda) = (K + \lambda I)$. If K is positive definite the solution to (15) is unique and given by (16). However, other solutions to (15) exist if K is not full rank which results in the same minimizer f. Note that we are not suggesting inverting G to obtain the solution, the use of G^{-1} is formal only. To obtain a solution of c for a fixed value of the parameter λ we need to solve the following linear system of equations

$$(K + \lambda I)\, c = Y. \tag{17}$$

Since the matrix $(K + \lambda I)$ is symmetric and positive definite for any positive λ, an appropriate algorithm for solving this system of equations is Cholesky factorization which is a decomposition of the matrix into the product of a lower triangular matrix and its conjugate transpose. When it is applicable, the Cholesky decomposition is roughly twice as efficient as the LU decomposition for solving systems of linear equations. Once we have c, the prediction at a new test point x_* follows from (8),

$$f(x_*) = \sum_{j=1}^{n} c_j k(x_*, x_j) = k(x_*, X)c. \tag{18}$$

3 Solving RLS for Varying λ

Usually we don't know a good value of λ in advance and solving the linear system for each value of λ in (17) is computationally expensive. The question is whether there is a more efficient method than solving $c(\lambda) = (K + \lambda I)^{-1}Y$ afresh for each λ. We will make use of the eigendecomposition of the symmetric positive semidefinite kernel, $K = Q\Lambda Q^T$, where Λ is diagonal with $\Lambda_{ii} \geq 0$ and $QQ^T = I$. We then can represent $G(\lambda)$ as

$$\begin{aligned} G(\lambda) &= K + \lambda I \\ &= Q\Lambda Q^T + \lambda QQ^T \\ &= Q(\Lambda + \lambda I)Q^T, \end{aligned}\tag{19}$$

which implies that $G^{-1}(\lambda) = Q(\Lambda + \lambda I)^{-1}Q^T$. Since the matrix $(\Lambda + \lambda I)$ is diagonal, its inverse is diagonal and $\left((\Lambda + \lambda I)^{-1}\right)_{ii} = \frac{1}{\Lambda_{ii}+\lambda}$ are the eigenvalues of $G^{-1}(\lambda)$. We see that λ plays the role of stabilizing the system. Without regularization term (i.e. $\lambda = 0$), small eigenvalues Λ_{ii} will lead

to enormous eigenvalues of G^{-1}. Increasing λ makes the solution more stable. However, increasing λ too much so that $\lambda >> \Lambda_{ii}$ will result in over-smoothing since we are ignoring the data. A good value of the regularization parameter λ can often be found between the smallest eigenvalue of K and its largest eigenvalue. When there is no regularization or equivalently when λ is too small, two problems occur: numerical stability which is due to the computer precision, and unstable behavior of the solution which results in a huge variation of the solution to two similar training sets. A good choice of λ makes the solution more generalizable.

It takes $O(n^3)$ time to solve one (dense) linear system, or to compute the eigendecomposition of the corresponding matrix (maybe 4x worse). Solving the linear system afresh for n values of λ will take $O(n^4)$ time. The proposed algorithm which uses the eigenvalue decomposition suggests to do that computation in $O(n^3)$. Decomposing K is $O(n^3)$. Once we have the decomposition (Q and Λ), we can find $c(\lambda)$ for a given λ in $O(n^2)$ time by solving

$$c(\lambda) = Q(\Lambda + \lambda I)^{-1} Q^T Y. \tag{20}$$

Note that since $(\Lambda + \lambda I)$ is diagonal, multiplying its inverse by $Q^T Y$ is a linear time operation ($O(n)$), and Q times the result is $O(n^2)$. When varying λ, only the diagonal matrix is changed and therefore finding $c(\lambda)$ for many values of λ's is essentially free. Once we have c, we can also get the predictions Kc in $O(n^2)$ time.

4 Validation

We have shown how to find $c(\lambda)$ quickly as we vary λ. In general, we need a mechanism for finding a good value for the regularization parameter λ. We would like a regularization parameter that makes the solution generalizable, i.e. a value for which the corresponding solution predicts well future examples. Since the distribution over examples is unknown, choosing the hyperparameters should be based only on the training set. For this matter, we can use a cross-validation method in which we train on certain amount of the data and validate on the holdout set, which was not used for training. If instead we use for validation the training set error it might result in overfitting. There are many other methods such as Akaike or other Bayesian methods which work good for small dimensional problems. However, for high dimensional problems cross-validation is usually used. When 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. When we have few data points, more common is k-fold cross-validation, which one meaning of it suggests to dividing the data into k equal sets. For each iteration i, train on the other $k-1$ sets and test on the i^{th} set. The limit of k-fold validation is leave-one-out (LOO) cross-validation.

LOO cross-validation suggests for each data point x_i to build a classifier based on the remaining $n-1$ data points, and compute the prediction error at x_i. The problem with that method is that it requires to build n different predictors on data sets of size $n-1$. We will now show that for the case of RLS, obtaining the LOO error is essentially free.

Define S^i to be the data set with the i^{th} point removed, i.e.

$$S^i = \{(x_1, y_1), \ldots, (x_{i-1}, y_{i-1}), (x_{i+1}, y_{i+1}), \ldots, (x_n, y_n)\}. \tag{21}$$

The i^{th} leave-one-out value $f_{S^i}(x_i)$ is the value of the RLS function trained on S^i and applied at x_i. The corresponding i^{th} leave-one-out error is $(y_i - f_{S^i}(x_i))$. We define L_V and L_E to be the vectors of leave-one-out values and errors over the training set. $\frac{1}{n}\|L_E\|_2^2$ is considered a good empirical proxy for the error on future points, and we often want to choose parameters by minimizing this quantity.

We will now show that for RLS, working with L_V and L_E are computational effective. Define the vector Y^i as

$$y^i_j = \begin{cases} y_j & j \neq i \\ f_{S^i}(x_i) & j = i \end{cases} \tag{22}$$

where all its entries are the same as of Y except its i^{th} entry which is replaced by the unknown value $f_{S^i}(x_i)$. If we solve RLS with labels Y^i instead of Y, we will obtain $f_{S^i}(\cdot)$ as the optimizer. Intuitively, when discarding the i^{th} example, we will get $f_{S^i}(\cdot)$. Adding the example $(x_i, f_{S^i}(x_i))$ will add nothing to the cost. More formally, for a general $f \in \mathcal{H}$

$$\begin{aligned} & \frac{1}{2}\sum_{j=1}^{n}(y_j^i - f(x_j))^2 + \frac{\lambda}{2}||f||_{\mathcal{H}}^2 \\ \geq & \frac{1}{2}\sum_{j\neq i}(y_j^i - f(x_j))^2 + \frac{\lambda}{2}||f||_{\mathcal{H}}^2 \\ \geq & \frac{1}{2}\sum_{j\neq i}(y_j^i - f_{S^i}(x_j))^2 + \frac{\lambda}{2}||f_{S^i}||_{\mathcal{H}}^2 \\ = & \frac{1}{2}\sum_{j=1}^{n}(y_j^i - f_{S^i}(x_j))^2 + \frac{\lambda}{2}||f_{S^i}||_{\mathcal{H}}^2. \end{aligned} \tag{23}$$

where in the first transition we exclude the i^{th} data point which reduces the cost since the loss function is non-negative. The second transition follows since f_{S^i} solves the LOO problem and since it is optimal it has a lower cost than any other $f \in \mathcal{H}$. The last transition follows since the i^{th} term that was added is zero. This proves that f_{S^i} is the optimal solution for the RLS problem using Y^i as the labels. Representing the optimizer of the RLS in terms of K, we obtain

$$\begin{aligned} c^i &= G^{-1}Y^i \\ f_{S^i}(x_i) &= (KG^{-1}Y^i)_i. \end{aligned} \tag{24}$$

Note that this is circular reasoning since in order to form Y^i we need to know $f_{S^i}(x_i)$. However, since only the labeled vector Y has been changed compared to the RLS problem for the whole set of data points, we will show how to reuse calculation and exploit the solution $f_S(X) = KG^{-1}Y$ for the derivation of L_V and L_E. Computing the difference between the predicted value of f_{S^i} at x_i and the predictor $f_S(x_i)$ obtained by training on the whole set, we obtain

$$\begin{aligned} f_{S^i}(x_i) - f_S(x_i) &= \sum_j (KG^{-1})_{ij}(y_j^i - y_j) \\ &= (KG^{-1})_{ii}(f_{S^i}(x_i) - y_i) \end{aligned} \tag{25}$$

where in the second transition we used the fact that the labeled vectors are the same except of their i^{th} element. Rearranging, we obtain a closed form for the LOO projection

$$\begin{aligned} f_{S^i}(x_i) &= \frac{f_S(x_i) - (KG^{-1})_{ii}y_i}{1 - (KG^{-1})_{ii}} \\ &= \frac{(KG^{-1}y)_i - (KG^{-1})_{ii}y_i}{1 - (KG^{-1})_{ii}}. \end{aligned} \tag{26}$$

Therefore, for all data points

$$L_V = \frac{KG^{-1}Y - diag_m(KG^{-1})Y}{diag_v(I - KG^{-1})}, \tag{27}$$

where the division in (27) is elementwise, $diag_m(M)$ denotes the diagonal matrix whose diagonal elements are M_{ii}, and $diag_v(M)$ denotes a column vector whose elements are M_{ii}. The corresponding

error is

$$\begin{aligned} L_E &= Y - L_V \\ &= Y + \frac{diag_m(KG^{-1})Y - KG^{-1}Y}{diag_v(I - KG^{-1})} \\ &= \frac{Y - KG^{-1}Y}{diag_v(I - KG^{-1})} = \frac{\left(I - KG^{-1}\right)Y}{diag_v(I - KG^{-1})}. \end{aligned} \tag{28}$$

We can simplify the expressions for L_E in a way that leads to better computational and numerical properties by using the eigenvalue decomposition of K as follows

$$\begin{aligned} KG^{-1} &= Q\Lambda Q^T Q(\Lambda + \lambda I)^{-1} Q^T \\ &= Q\Lambda(\Lambda + \lambda I)^{-1} Q^T \\ &= Q(\Lambda + \lambda I - \lambda I)(\Lambda + \lambda I)^{-1} Q^T \\ &= I - \lambda G^{-1}. \end{aligned} \tag{29}$$

which yields

$$I - KG^{-1} = \lambda G^{-1}. \tag{30}$$

Substituting (30) into the expression of L_E in (28) yields

$$\begin{aligned} L_E &= \frac{\lambda G^{-1} Y}{diag_v(\lambda G^{-1})} \\ &= \frac{G^{-1} Y}{diag_v(G^{-1})} \\ &= \frac{c}{diag_v(G^{-1})}. \end{aligned} \tag{31}$$

where c is the solution to the RLS problem when using all data points. We already showed how to compute $c(\lambda)$ in $O(n^2)$ time given that we already computed the eigenvalue decomposition $K = Q\Lambda Q^T$. We can also compute a single entry of G^{-1} in $O(n)$ time as follows

$$\begin{aligned} G^{-1}_{ij} &= (Q(\Lambda + \lambda I)^{-1} Q^T)_{ij} \\ &= \sum_{k=1}^{n} \frac{Q_{ik} Q_{jk}}{\Lambda_{kk} + \lambda}, \end{aligned} \tag{32}$$

and therefore we can compute $diag(G^{-1})$, and compute L_E, in $O(n^2)$ time, just the same as solving RLS once!

5 Summary

In RLS, the Tikhonov minimization problem boils down to solving a linear system $(K + \lambda I)c = Y$ since

$$\arg\min_{f \in \mathcal{H}} \frac{1}{2} \sum_{i=1}^{n} (f(x_i) - y_i)^2 + \frac{\lambda}{2} \|f\|^2_{\mathcal{H}} = k(\cdot, X)c. \tag{33}$$

It was shown how efficient the RLS algorithm can be solved by using the eigendecomposition of the kernel matrix: $K = Q\Lambda Q^T$. Specifically, we can cheaply compute $c(\lambda)$ for a bunch of λ's

and compute the leave-one-out error over the whole training set about as cheaply as solving for c once, i.e. in the same time it takes to solve a single RLS problem on our data. The primary disadvantage of nonlinear RLS is the need to work with the entire kernel matrix K. Forming the kernel K takes $O(n^2 d)$ time and $O(n^2)$ memory. An eigendecomposition of the kernel matrix K 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). The linear kernel offers many advantages for computation.