Kernel ridge Regression

Max Welling
Department of Computer Science
University of Toronto
10 King's College Road
Toronto, M5S 3G5 Canada
welling@cs.toronto.edu

Abstract

This is a note to explain kernel ridge regression.

1 Ridge Regression

Possibly the most elementary algorithm that can be kernelized is ridge regression. Here our task is to find a linear function that models the dependencies between covariates $\{x_i\}$ and response variables $\{y_i\}$, both continuous. The classical way to do that is to minimize the quadratic cost,

$$C(\mathbf{w}) = \frac{1}{2}\sum_i (y_i - \mathbf{w}^T\mathbf{x}_i)^2 \tag{1}$$

However, if we are going to work in feature space, where we replace $\mathbf{x}_i \rightarrow \Phi(\mathbf{x}_i)$, there is an clear danger that we overfit. Hence we need to regularize. This is an important topic that will return in future classes.

A simple yet effective way to regularize is to penalize the norm of $\mathbf{w}$. This is sometimes called "weight-decay". It remains to be determined how to choose λ. The most used algorithm is to use cross validation or leave-one-out estimates. The total cost function hence becomes,

$$C = \frac{1}{2}\sum_i (y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \frac{1}{2}\lambda||\mathbf{w}||^2 \tag{2}$$

which needs to be minimized. Taking derivatives and equating them to zero gives,

$$\sum_i (y_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i = \lambda\mathbf{w} \quad \Rightarrow \quad \mathbf{w} = \left(\lambda\mathbf{I} + \sum_i \mathbf{x}_i\mathbf{x}_i^T\right)^{-1}\left(\sum_j y_j\mathbf{x}_j\right) \tag{3}$$

We see that the regularization term helps to stabilize the inverse numerically by bounding the smallest eigenvalues away from zero.

2 Kernel Ridge Regression

We now replace all data-cases with their feature vector: $\mathbf{x}_i \rightarrow \Phi_i = \Phi(\mathbf{x}_i)$. In this case the number of dimensions can be much higher, or even infinitely higher, than the number of data-cases. There is a neat trick that allows us to perform the inverse above in smallest space of the two possibilities, either the dimension of the feature space or the number of data-cases. The trick is given by the following identity,

$$(P^{-1} + B^T R^{-1} B)^{-1} B^T R^{-1} = P B^T (B P B^T + R)^{-1} \tag{4}$$

Now note that if B is not square, the inverse is performed in spaces of different dimensionality. To apply this to our case we define $\Phi = \Phi_{ai}$ and $\mathbf{y} = y_i$. The solution is then given by,

$$\mathbf{w} = (\lambda \mathbf{I}_d + \Phi\Phi^T)^{-1}\Phi\mathbf{y} = \Phi(\Phi^T\Phi + \lambda\mathbf{I}_n)^{-1}\mathbf{y} \tag{5}$$

This equation can be rewritten as: $\mathbf{w} = \sum_i \alpha_i \Phi(\mathbf{x}_i)$ with $\boldsymbol{\alpha} = (\Phi^T\Phi + \lambda\mathbf{I}_n)^{-1}\mathbf{y}$. This is an equation that will be a recurrent theme and it can be interpreted as: The solution $\mathbf{w}$ must lie in the span of the data-cases, even if the dimensionality of the feature space is much larger than the number of data-cases. This seems intuitively clear, since the algorithm is linear in feature space.

We finally need to show that we never actually need access to the feature vectors, which could be infinitely long (which would be rather impractical). What we need in practice is is the predicted value for anew test point, $\mathbf{x}$. This is computed by projecting it onto the solution $\mathbf{w}$,

$$y = \mathbf{w}^T\Phi(\mathbf{x}) = \mathbf{y}(\Phi^T\Phi + \lambda\mathbf{I}_n)^{-1}\Phi^T\Phi(x) = \mathbf{y}(K + \lambda\mathbf{I}_n)^{-1}\boldsymbol{\kappa}(\mathbf{x}) \tag{6}$$

where $K(bx_i, bx_j) = \Phi(x_i)^T\Phi(x_j)$ and $\boldsymbol{\kappa}(\mathbf{x}) = K(\mathbf{x}_i, \mathbf{x})$. The important message here is of course that we only need access to the kernel K.

We can now add bias to the whole story by adding one more, constant feature to Φ: $\Phi_0 = 1$. The value of w_0 then represents the bias since,

$$\mathbf{w}^T\Phi = \sum_a \mathbf{w}_a \Phi_{ai} + \mathbf{w}_0 \tag{7}$$

Hence, the story goes through unchanged.

3 An alternative derivation

Instead of optimizing the cost function above we can introduce Lagrange multipliers into the problem. This will have the effect that the derivation goes along similar lines as the SVM case. We introduce new variables, $\xi_i = y_i - \mathbf{w}^T\Phi_i$ and rewrite the objective as the following constrained QP,

$$\text{minimize} - \mathbf{w}, \boldsymbol{\xi} \qquad \mathcal{L}_P = \sum_i \xi_i^2$$

$$\text{subject to} \qquad y_i - \mathbf{w}^T\Phi_i = \xi_i \ \ \forall i \tag{8}$$

$$||\mathbf{w}|| \leq B \tag{9}$$

This leads to the Lagrangian,

$$\mathcal{L}_P = \sum_i \xi_i^2 + \sum_i \beta_i[y_i - \mathbf{w}^T\Phi_i - \xi_i] + \lambda(||\mathbf{w}||^2 - B^2) \tag{10}$$

Two of the KKT conditions tell us that at the solution we have:

$$2\xi_i = \beta_i \ \ \forall i, \qquad 2\lambda\mathbf{w} = \sum_i \beta_i \Phi_i \tag{11}$$

Plugging it back into the Lagrangian, we obtain the dual Lagrangian,

$$\mathcal{L}_D = \sum_i (-\frac{1}{4}\beta_i^2 + \beta_i y_i) - \frac{1}{4\lambda}\sum_{ij}(\beta_i\beta_j K_{ij}) - \lambda B^2 \tag{12}$$

We now redefine $\alpha_i = \beta_i/(2\lambda)$ to arrive at the following dual optimization problem,

$$\text{maximize}-\boldsymbol{\alpha}, \lambda \qquad -\lambda^2 \sum_i \alpha_i^2 + 2\lambda \sum_i \alpha_i y_i - \lambda \sum_{ij} \alpha_i \alpha_j K_{ij} - \lambda B^2 \qquad s.t. \lambda \geq 0 \quad (13)$$

Taking derivatives w.r.t. $\boldsymbol{\alpha}$ gives precisely the solution we had already found,

$$\alpha_i^* = (K + \lambda \mathbf{I})^{-1} \mathbf{y} \quad (14)$$

Formally we also need to maximize over λ. However, different choices of λ correspond to different choices for B. Either λ or B should be chosen using cross-validation or some other measure, so we could as well vary λ in this process.

One big disadvantage of the ridge-regression is that we don't have sparseness in the $\boldsymbol{\alpha}$ vector, i.e. there is no concept of support vectors. This is useful because when we test a new example, we only have to sum over the support vectors which is much faster than summing over the entire training-set. In the SVM the sparseness was born out of the inequality constraints because the complementary slackness conditions told us that either if the constraint was inactive, then the multiplier α_i was zero. There is no such effect here.