Kernel ridge regression

Isabelle Guyon – Isabelle@clopinet.com, June 2005

The kernel ridge regression method (see e.g. the "The Elements of Statistical Learning" by T. Hastie R. Tibshirani J. H. Friedman, Springer, 2001) is a **regularized least square method** for classification and regression. The linear version is similar to **Fisher's discriminant** for classification. The non-linear version is **similar to an SVM**, except that a different objective is being optimized, which does not put emphasis on points close to the decision boundary. The solution depends on ALL the training examples (not on a subset of support vectors.) Hence, the method is suitable only for datasets with few training examples. For the linear version, the kernel trick is useful if the number of features is large and the number of examples small. In the opposite case, one should not use the kernel trick and work in direct space. This allows us to build ridge regression linear classifiers for a small number of features and many training examples. An advantage of kernel ridge regression is that there exist formulas to compute the leave-one-out mean-squared error and classification error rate using the results of a single training on the whole training set, i.e. without actually performing the leave-one-out. Hence, the hyper-parameters (the ridge and the kernel parameters) can be optimized efficiently. In addition, if an implementation with singular value decomposition is made, one can compute with a single training the solutions corresponding to many values of the ridge. Combined with the leave-one-out formula, this renders the ridge optimization very efficient.

Pseudo-inverse and ridge

We present an implementation of kernel ridge regression using the pseudo-inverse. We first describe the linear case and then move to the non-linear case via the kernel trick.

Let X be the data matrix of dimension (p, n), p patterns, n features. Let **y** be the target matrix of dimension (p, 1). Let X1 be the data matrix augmented by the unity vector **1** of dimension (p, 1) that contains only one values: X1=[X, **1**]. X1 is therefore of dimension (p, n+1). Linear regression seeks to solve the following matrix equation:

$$X1\ [\mathbf{w}'; b] = \mathbf{y} \qquad (1)$$

Where **w** if the weight vector of dimension (1, n) and b the bias value. Prime denotes matrix transposition.
The resulting regression machine is:

$$f(\mathbf{x}) = \mathbf{x}.\mathbf{w}'+b, \qquad (2)$$

where **x** is an input pattern of dimension (1, n). In the case of classification, vector **y** contains binary values (e.g. ±1, we discuss target values in mode details later). Function f(**x**) is then used as a linear discriminant, the sign of which is used to classify pattern **x**.
The best set of parameters in the least square sense is given by:

$$[\mathbf{w}'; b] = X1^{+}\ \mathbf{y} \qquad (3)$$

where $X1^{+}$ is the pseudo-inverse of X1. There are many ways of computing the pseudo-inverse, but they never require computationally more than what is required to invert a matrix of dimension (p, p) or (n, n), whichever is smallest. (For an introduction on

pseudo-inverse, see: Regression and the Moore-Penrose pseudo-inverse, Academic Press, 1972).

We recommend to define one of the two following matrices (whichever is smaller):

$$G = X1' X1 + \delta I \quad \text{of dim } (n+1, n+1) \quad (4)$$

or $$K = X1 X1' + \delta I \quad \text{of dim } (p, p) \quad (5)$$

where I is the identity matrix of the right dimension. The coefficient δ (ridge) can be chosen large enough that the matrix is invertible (e.g. $\delta=10^{-10}$). The method to optimize δ is discussed later. The "true" pseudo-inverse is obtained in the limit of δ going to zero, but better predictors are obtained with non-negligible values of δ, for reasons that are well understood by regularization/learning theory.

Then, the pseudo-inverse is computed as:

$$X1^+ = G^{-1} X1' \quad (6)$$

or $$X1^+ = X1' K^{-1}. \quad (7)$$

The following theorem (see Albert's book) allows in fact to use either formula (6) or (7) provided that the pseudo-inverse (not the true inverse) of G or K is taken:

$$X1^+ = G^+ X1' = X1' K^+. \quad (8)$$

This not always computationally interesting, but it will become handy in what follows for other reasons.

Formula (8) shows that this method can easily be turned into a kernel method.

Substituting (8) into (3) and applying it to (2), we get the following predictor:

$$f(\mathbf{x}) = [\mathbf{x}, 1] X1' K^+ \mathbf{y} \quad (9)$$

that we can re-write as:

$$f(\mathbf{x}) = \mathbf{k}(\mathbf{x})' K^+ \mathbf{y} \quad (10)$$

where $\mathbf{k}(\mathbf{x})$ is a (p, 1) dimensional vector whose components are dot products of $[\mathbf{x}, 1]$ and of lines of X1. To generalize to any kernel method, it suffices to replace the dot product by another kernel function $k(\mathbf{x}, \mathbf{x}')$. Then,

$$K = [k(\mathbf{x}_i, \mathbf{x}_j)] \quad (11)$$

where indices i and j run from 1 to p (all training patterns), and

$$\mathbf{k}(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}_i) \quad (12)$$

where index i runs from 1 to p.

From formula (10) we can also make apparent that $f(\mathbf{x})$ is a linear combination of kernel functions:

$$f(\mathbf{x}) = \mathbf{k}(\mathbf{x})' b = \Sigma_i \beta_i k(\mathbf{x}, \mathbf{x}_i) \quad (13)$$

where

$$b = K^+ \mathbf{y} \quad (14)$$

The connection to Fisher's linear discriminant may not be obvious, but in can be shown (Duda and Hart, 1973) that using this method is equivalent to Fisher's linear discriminant in the following case: class target values ± 1 are replaced by p/p_1 and $-p/p_2$, where p_1 and p_2 are the number of examples of the positive and negative class respectively. This does not always turn out to give better results than using targets ± 1 when the class cardinalities are different.

The kernel regression method described above has been re-discovered several times and is known under various names (e.g. "ridge regression" (Hoerl-Kennard), "regularization networks" (Poggio-Girosi), neural network "weight-decay", see historical in http://www.anc.ed.ac.uk/~mjo/intro/node19.html). Since it is over 20 year-old one should

not worry about patent infringement. Unlike SVMs it does not provide a sparse solution (the sum runs over all the examples, not only on the support vectors. But it often provides similar performance and may be faster to train (depending on implementations.)

Ridge optimization

Most of the computational time is spent to invert matrix G or K (Equations 4 and 5), whichever is smallest. To optimize the ridge, one must perform such inversion for various values of δ. Fortunately, this can be performed efficiently: One must perform an eigen value decomposition of G or K, e.g. for K:

$$K=UDV \qquad (15)$$

where U and V are orthogonal matrices and D is a diagonal matrix of eigen vectors. The inverse of K is obtained as:

$$K^{-1}=V'D^{-1}U' \qquad (16)$$

Conveniently, if we call K_0 the matrix with a ridge of zero, the inverse of $K_\delta = K_0 + \delta I$ can be computed as:

$$K_\delta^{-1}=V'(D+\delta I)^{-1}U'$$

where $K_0=UDV$. The inversion of a diagonal matrix being trivial, once the eigen decomposition of K_0 is performed, obtaining solutions for various values of the ridge is inexpensive.

Optimizing the ridge is rendered further efficient by the leave-one-out calculations explained in the next section.

Leave-one-out calculations

The performance of a predictor can be assessed in a number of ways. For regression, one typically computes the mean-square-error (mse):

$$mse = < (y - \ddot{y})^2 > \qquad (17)$$

For classification, one computes the error rate:

$$errate = < \Theta(-y\ddot{y}) > \qquad (18)$$

where $\Theta(.)$ denotes the function that has value 1 if the argument is positive and zero otherwise.

The notation $< . >$ indicates that an average is taken over a number of patterns. If the average is taken over the training examples used to adjust the parameters, the estimation of performance is biased (lower mse and error rate than on unseen examples). Therefore it is preferable to use unseen test examples to estimate performance. However, this is often impractical because examples are scarce.

An alternative method is to use cross-validation, that is to train on a subset of examples and test on the rest and rotate through the examples many times. In the limit, one trains on all the examples but one and test on the remaining example. The predictions on the examples held out are used to assess performance (leave-one-out method.)

The leave-one-out method is computationally expensive because p models need to be trained. Fortunately, in the case of the least-square linear model, there are formulas to compute the leave-one-out statistics without having to train more than one model on all the examples.

Let us call XX^+ the projection matrix onto the subspace spanned by the column vectors of X (for simplicity we leave out the **1** column used to set the bias value). Let us call r_i the i^{th} residual:

$$r_i = y_i - \ddot{y}_i \quad (19)$$

The i^{th} leave-one-out residual is given by (Efron-Tibshirani, 1993):

$$e_i = r_i / [1-(XX^{+})_{ii}] \quad (20)$$

The leave-one-out mse follows:

$$mse\_loo = (1/p) \sum_{i=1..p} e_i^2 \quad (21)$$

The mse_loo thus computed is exact.

Similarly, there exists a formula to compute the leave-one-out error rate (Opper-Winther, 1999). Such formula however is approximate, but it is a very good approximation. The method consists in replacing the predicted value $\ddot{y}$ in formula (18) by:

$$\ddot{y}_{loo\ i} = \ddot{y}_i - \lambda_i \beta_i \quad (22)$$

where the β_i coefficients are obtained from $[(XX^T+\delta I)^{-1}Y]_i$, and the λ_i coefficients are obtained from $-\delta+[1/[(XX^T+\delta I)^{-1}]_{ii}$.

The resulting leave-one-out error rate is obtained as:

$$errate\_loo = (1/p) \sum_{i=1..p} \Theta(-y_i \ddot{y}_{loo\ i}) \quad (10)$$

Noticing that XX^T is the Gram-matrix/Hessian/kernel-matrix K of kernel least square, both formulas mse_loo and errate_loo are trivially kernelized and extend to kernel least square. Formula (20) becomes $e_i = r_i / [1-(KK^{-1})_{ii}]$. For the others, substitute XX^T by K.