The Korean Communications
in Statistics Vol. 6, No. 3, 1999
pp. 843-847

Kernel Adatron Algorithm for Support Vector Regression[1]

Kyungha Seok[2], Changha Hwang[3], Daehyun Cho[2]

Abstract

Support vector machine(SVM) is a new and very promising classification and regression technique developed by Vapnik and his group at AT&T Bell Laboratories. However, it has failed to establish itself as common machine learning tool. This is partly due to the fact that SVM is not easy to implement, and its standard implementation requires the optimization package for quadratic programming. In this paper we present simple iterative Kernel Adatron algorithm for nonparametric regression which is easy to implement and guaranteed to converge to the optimal solution, and compare it with neural networks and projection pursuit regression.

1. Kernel Adatron Algorithm

The foundations of SVM have been developed by Vapnik(1995) and are gaining popularity due to many attractive features, and promising empirical performance. SVM has been successfully applied to a number of real world problems such as handwritten character and digit recognition, face detection, text categorization and object detection in machine vision. These are all classification problems. In SVM there are two types, i.e., support vector classification(SVC) and support vector regression(SVR). SVM was developed to solve the classification problem, but recently it has been extended to the domain of regression problems. However, SVC can be viewed as a special case of SVR. For an overview of SVR, see Gunn(1998), Smola & Scholkopf(1998), and Vapnik(1995, 1998).

SVC has proven to be highly effective for learning many real world datasets but has failed to establish themselves as common machine learning tools. This is partly due to the fact that this is not easy to implement, and their standard implementation requires the use of optimization packages for quadratic programming(QP). SVR has been similarly hampered by the same problem. In particular, SVM using QP is not useful for regression tasks for problems with sample size much larger than 100. Campbell & Cristianini(1998) and Cristianini, Campbell & Shawe-Taylor(1998) proposed Kernel Adatron(KA) algorithm for SVC which is easy to implement and guaranteed to converge to the optimal solution. In this paper we present KA algorithm for SVR.

---

1) This work was supported by Grant from Inje University, 1998
2) Dept. of Data Science, Inje University, Kyungnam, Korea.
3) Dept. of Statistical Information, Catholic University of Taegu-Hyosung, Kyungbuk, Korea.

We will now illustrate the KA algorithm for SVR using gradient ascent. There is no loss of generality that we just consider nonlinear SVR. Suppose we are given training data $\{(\boldsymbol{x}_i, y_i),\ i=1,\cdots,n\} \subset \mathfrak{X} \times R$, where $\mathfrak{X}$ denotes the space of the input vectors, R^d. Our goal is to find a function $f(\boldsymbol{x})$ that has at most ε deviation from the actually obtained targets y_i's for all the training data, and at the same time, is as flat as possible. The nonlinear SVR solution, using an ε-insensitive loss function, is given by

$$\text{maximize} \quad -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(a_i-a_i^*)(a_j-a_j^*)K(\boldsymbol{x}_i,\boldsymbol{x}_j)-\varepsilon\sum_{i=1}^{n}(a_i+a_i^*)+\sum_{i=1}^{n}y_i(a_i-a_i^*) \quad (1)$$

$$\text{subject to} \quad \sum_{i=1}^{n}(a_i-a_i^*)=0 \quad \text{and} \quad a_i,\ a_i^* \in [0,\ C].$$

For details, see Gunn(1998) and Smola & Scholkopf(1998). Here, K is kernel function. The kernels often used are given below.

$$K(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}^t\boldsymbol{y}+1)^p, \quad K(\boldsymbol{x},\boldsymbol{y}) = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2\sigma^2}}.$$

Here, p and σ^2 are kernel parameters. The idea of the kernel function is to enable operations to be performed in the input space rather than the potentially high dimensional feature space. This provides a way of addressing the curse of dimensionality. Solving the above equation with these constraints determines the Lagrange multipliers, a_i, a_i^*, and the optimal regression function is given by

$$f(\boldsymbol{x}) = \sum_{i=1}^{n}(a_i-a_i^*)K(\boldsymbol{x}_i,\boldsymbol{x})+b.$$

So far we neglected the issue of computing b. For details, see Smola and Scholkopf(1998).

In general, SVR training requires solving a QP problem which is a notoriously difficult business. Furthermore, standard QP routines have substantial memory resource requirements and large datasets require additional techniques such as chunking (breaking the QP problem into a series of simpler QP tasks). KA algorithm can be derived from principles for SVR optimal solution. In fact, this merge perceptron-like rules with kernel methods.

The most obvious way of maximizing a concave Lagrangian under linear constraints is gradient ascent. The Lagrangian to be maximized is:

$$L(\boldsymbol{a}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(a_i-a_i^*)(a_j-a_j^*)K(\boldsymbol{x}_i,\boldsymbol{x}_j)-\varepsilon\sum_{i=1}^{n}(a_i+a_i^*)+\sum_{i=1}^{n}y_i(a_i-a_i^*).$$

We will maximize this Lagrangian using stochastic gradient ascent based on the derivative of the Lagrangian, thus:

$$\delta a_k = \eta_1\frac{\partial L}{\partial a_k} = \eta_1\left(-\sum_{i=1}^{n}(a_i-a_i^*)K(\boldsymbol{x}_k,\boldsymbol{x}_i)-\varepsilon+y_k\right),$$

$$\delta a_k^* = \eta_2\frac{\partial L}{\partial a_k^*} = \eta_2\left(-\sum_{i=1}^{n}(a_i-a_i^*)K(\boldsymbol{x}_k,\boldsymbol{x}_i)-\varepsilon-y_k\right).$$

By the way, we will enforce the constraints $\alpha_i \geq 0$, $\alpha_i^* \geq 0$ by setting $\alpha_i \rightarrow 0$, $\alpha_i^* \rightarrow 0$ for those α_i's and α_i^*'s which would become negative. Let us consider the change in this Lagrangian due to an updating $\alpha_k \rightarrow \alpha_k + \delta\alpha_k$ and $\alpha_k^* \rightarrow \alpha_k^* + \delta\alpha_k^*$ for a particular kth pattern:

$$\begin{aligned}\delta L &= L(\alpha_k^{(*)} + \delta\alpha_k^{(*)}) - L(\alpha_k^{(*)}) \\ &= \delta\alpha_k\Big(-\sum_{j=1}^{n}(\alpha_j - \alpha_j^*)K(\boldsymbol{x}_k, \boldsymbol{x}_j) - \varepsilon + y_k\Big) + \delta\alpha_k^*\Big(\sum_{j=1}^{n}(\alpha_j - \alpha_j^*)K(\boldsymbol{x}_k, \boldsymbol{x}_j) - \varepsilon - y_k\Big) \\ &\quad - \frac{1}{2}[\delta(\alpha_k - \alpha_k^*)]^2 K(\boldsymbol{x}_k, \boldsymbol{x}_k),\end{aligned}$$

where $\alpha_k^{(*)}$ means α_k and α_k^*. For simplicity, we put $\eta_1 = \eta_2$. Since SVR solution does not depend very much on the type of kernel, we choose Gaussian kernel $K(\boldsymbol{x}, \boldsymbol{y}) = \exp\{-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2\sigma^2}\}$ in this paper. Then the change δL can be simplified into

$$\begin{aligned}\delta L &= \eta[2c_k^2 + 2y_k^2 - 4c_k y_k + 2\varepsilon^2] - \frac{1}{2}\eta^2[4c_k^2 + 4y_k^2 - 8c_k y_k] \\ &= 2(c_k - y_k)^2(\eta - \eta^2) + 2\eta\varepsilon^2,\end{aligned}$$

where

$$c_k = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)K(\boldsymbol{x}_k, \boldsymbol{x}_i)$$

and this c_k becomes the estimate of y_k. Now we can optimize δL with respect to η giving:

$$\eta = \frac{1}{2} + \frac{\varepsilon^2}{(c_k - y_k)^2}.$$

In general, the optimal value for the learning rate η is pattern dependent. We notice that η becomes equal to 0.5 when $\varepsilon = 0$. Furthermore, η tends to become large when $\varepsilon > 0$ so that c_k can estimate well y_k.

2. Simulation

We compare KA with neural networks using standard backpropagation(BP) and robust backpropagation(RBP) by Chen & Jain(1994), and projection pursuit regression(PPR) for five bivariate nonlinear functions $g^{(j)}: [0, 1] \rightarrow R$ in Hwang et al.(1994). We use simulated examples so that we may compare the fits on a sizable test set. Our metric for comparison is the fraction of variance unexplained(FVU), which is defined as

$$FVU = \frac{\sum_{l=1}^{N}(\hat{g}(\boldsymbol{x}_l) - g(\boldsymbol{x}_l))^2}{\sum_{l=1}^{N}(g(\boldsymbol{x}_l) - \bar{g}(\boldsymbol{x}_l))^2},$$

where $g(\boldsymbol{x}_i)$ is the true value of the function and $\hat{g}(\boldsymbol{x}_i)$ is the fitted value. We evaluate the FVU for a fit by replacing the expectation with averages over a set of 10,000 test set values. We use 225 pairs of predictors drawn uniformly from the unit square. In the noiseless examples, the response is simply the value of the function $g(x_{1i}, x_{2i})$, while in the noisy examples the response is $g(x_{1i}, x_{2i}) + 0.25\varepsilon_i$, where the errors ε_i are i.i.d $N(0,1)$. The functions are given in Hwang et al.(1994).

Even for KA C, ε and σ should be pre-specified. Here, we choose same $C=10$, $\eta=0.5$, $\varepsilon=0$ for five functions. However, σ's are different. The value of σ for $g^{(1)}$ and $g^{(2)}$ is 0.4. And σ for $g^{(3)}$, $g^{(4)}$ and $g^{(5)}$ is 0.2. We determined these all parameter values using model selection methods based on VC-dimension. For model selection see Shao(1999). The number of iterations of KA is 1,000. It was enough to get good estimates. Table 1 presents the simulation results on this example. In our simulation study, PPR and KA have quite comparable training speed, and RBP and KA achieve comparable accuracy for test data. Overall KA shows good performance. In particular, KA does fitting very well for nosy data. It will be important to make a careful comparative evaluation of RBP, PPR and KA for a set of higher dimensional functions.

References

[1] Campbell, C. and Cristianini, N. (1998). Simple Learning Algorithms for Training Support Vector Machines, Dept. of Engineering Mathematics Technical Report, U. of Bristol.

[2] Chen, D.S. and Jain, R.C. (1994). A robust Back Propagation Learning Algorithm for Function Approximation, IEEE Transactions on Neural Networks, 5, 467-479.

[3] Cristianini, N., Campbell, C. and Shawe-Taylor, J. (1998). Dynamically Adapting Kernels in Support Vector Machines, NeuroCOLT2 Technical Report, NeuroCOLT.

[4] Gunn, S. (1998). Support Vector Machines for Classification and Regression, ISIS Technical Report, U. of Southampton.

[5] Hwang, J-N, Lay, S-R, Maechler, M., Martin, D. and Schimert, J. (1994), *Regression Modeling in Back-Propagation and Projection Pursuit Learning*, IEEE Transactions on Neural Networks, 5, 342-353.

[6] Shao, X. (1999). Model Selection Using Statistical Learning Theory, Ph. D. Thesis, U. of Minnesota.

[7] Smola, A.J. and Scholkopf, B. (1998). A Tutorial on Support Vector Regression, NeuroCOLT2 Technical Report, NeuroCOLT.

[8] Vapnik, V. (1995). The Nature of Statistical Learning Theory, Springer.

[9] Vapnik, V. (1998). Statistical Learning Theory, Springer.

Table 1. Accuracy determined by the error measure FVU

		Noiseless Data			Noisy Data		
Function	Method	No. of Node	FVU train	FVU test	No. of Node	FVU train	FVU test
g(1)	BP	5	0.000534	0.001471	5	0.064836	0.070192
		10	0.000227	0.001338	10	0.064651	0.072336
	RBP	5	0.000534	0.001266	5	0.064836	0.007300
		10	0.000227	0.001285	10	0.064651	0.007984
	PPR	3	0.000075	0.001077	3	0.053925	0.080629
		5	0.000048	0.001095	5	0.050652	0.080352
	KA		0.000297	0.001632		0.005243	0.008412
g(2)	BP	5	0.005934	0.007648	5	0.068849	0.083526
		10	0.003537	0.005400	10	0.058644	0.073024
	RBP	5	0.005949	0.006711	5	0.068866	0.023277
		10	0.003537	0.005127	10	0.059375	0.014247
	PPR	3	0.025906	0.034620	3	0.057967	0.082920
		5	0.001163	0.006069	5	0.058581	0.084407
	KA		0.000356	0.004492		0.006249	0.013994
g(3)	BP	5	0.524021	0.424022	5	0.399383	0.566565
		10	0.142020	0.151487	10	0.146998	0.231935
	RBP	5	0.567275	0.495448	5	0.401656	0.472818
		10	0.142935	0.132279	10	0.147073	0.169004
	PPR	3	0.360373	0.577145	3	0.185445	0.321773
		5	0.112422	0.242643	5	0.138678	0.341155
	KA		0.012061	0.130577		0.028167	0.104874
g(4)	BP	5	0.045945	0.019688	5	0.073419	0.086255
		10	0.004273	0.006913	10	0.059601	0.075581
	RBP	5	0.015946	0.019215	5	0.073427	0.025834
		10	0.004283	0.005262	10	0.061691	0.015751
	PPR	3	0.000389	0.000692	3	0.041929	0.091219
		5	0.000427	0.001915	5	0.038907	0.089882
	KA		0.000589	0.008519		0.017026	0.055434
g(5)	BP	5	0.214445	0.236293	5	0.249337	0.286426
		10	0.025470	0.070035	10	0.096857	0.138735
	RBP	5	0.214724	0.234898	5	0.262230	0.260099
		10	0.025487	0.065402	10	0.099910	0.086531
	PPR	3	0.140191	0.227764	3	0.294997	0.543458
		5	0.016889	0.038313	5	0.060511	0.192520
	KA		0.001209	0.030118		0.019193	0.024875