Parameter selection for support vector machines

Carl Staelin
HP Laboratories Israel[1]
HPL-2002-354 (R.1)
November 10th , 2003*

Support Vector Machines (SVM), Design of Experiments (DOE), meta-parameter selection

We present an algorithm for selecting support vector machine (SVM) meta-parameter values which is based on ideas from design of experiments (DOE) and demonstrate that it is robust and works effectively and efficiently on a variety of problems.

* Internal Accession Date Only
Approved for External Publication
1 Technion City, Haifa, 32000, Israel

Parameter selection for support vector machines

Carl Staelin, *Senior Member IEEE*

***Abstract*— We present an algorithm for selecting support vector machine (SVM) meta-parameter values which is based on ideas from design of experiments (DOE) and demonstrate that it is robust and works effectively and efficiently on a variety of problems.**

***Index Terms*— Support Vector Machines (SVM), meta-parameter selection, Design of Experiments (DOE)**

I. Introduction

SUPPORT vector machines (SVM) are a powerful machine learning method for both regression and classification problems. However, the various SVM formulations each require the user to set two or more parameters which govern the training process, and those parameter settings can have a profound affect on the resulting engine's performance. We present an algorithm based on principles from design of experiments (DOE) that can identify optimal or near optimal parameter settings an order of magnitude faster than an exhaustive grid search for SVMs using the radial basis function kernel, and we evaluate its performance on a number of standard test problems. This algorithm has been successfully applied to several real-world applications within HP.

The first issue is deciding how to evaluate the parameter's effectiveness, which is just the standard problem of evaluating machine learning method performance. The most common method is some form of N-fold cross-validation, but other approaches such as leave-one-out and Dietterich's 5x2cv test are also possible. All these methods require that the machine learning engine be trained multiple times in order to obtain a single performance number for a single parameter setting.

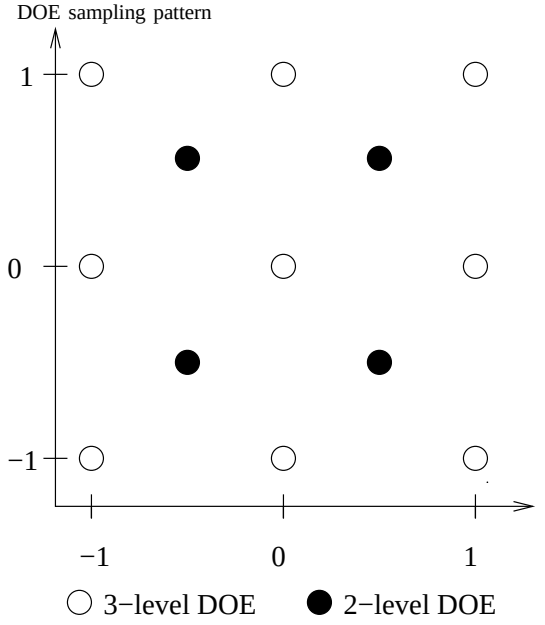
The most common and reliable approach to parameter selection is to decide on parameter ranges, and to then do an exhaustive grid search over the parameter space to find the best setting. Unfortunately, even moderately high resolution searches can result in a large number of evaluations and unacceptably long run times.

Our approach is to start with a very coarse grid covering the whole search space and iteratively refine both the grid resolution and search boundaries, keeping the number of samples at each iteration roughly constant. This is a variant on a fairly standard search method from the design of experiments field [1].

Figure 1 shows the sampling pattern for a two parameter search, which is a combination of a standard N-parameter, three-level $\{-1, 0, +1\}$, or 3^d experiment design with another standard N-parameter, two-level $\left\{-\frac{1}{2}, \frac{1}{2}\right\}$, or 2^d experiment design, resulting in thirteen points per iteration in the two-parameter case.

After each iteration the system evaluates all sampled points and chooses the point with the best performance. The search space is centered around the best point, unless this would cause the search to go outside the user-given bounds, in which case the center is adjusted so the whole space is contained within the user-given bounds. The grid resolution is then increased (currently doubled, but it might be increased by a different factor, such as $\sqrt{2}$) and the bounds shrunk, and a new set of sample points is calculated. This process is repeated as many times as desired.

Fig. 1. DOE sampling pattern



II. Prior Work

Support vector machines are a relatively recent method of machine learning based on structural risk minimization. Some standard reference works on SVM include [2], [3], [4].

From Platt [5]the definition of the quadratic programming problem that for support vector learning is:

$$max_\alpha W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} y_i y_j k(\vec{x_i}, \vec{x_j}) \alpha_i \alpha_j$$

$$s.t. \quad 0 \leq \alpha_i \leq C, \forall i$$
$$\textstyle\sum_{i=1}^{l} y_i \alpha_t = 0$$

where l is the number of $\langle x, y \rangle$ samples, $k(\vec{x_i}, \vec{x_j})$ is the kernel function of two sample input vectors $\vec{x_i}$ and $\vec{x_j}$, y_i and y_j are the corresponding sample values, C is a given parameter, and α_i are being optimized by the training process.

The quadratic programming problem is solved if and only if the Karush-Kuhn-Tucker (KTT) conditions are fulfilled and $Q_{ij} = y_i y_j k(\vec{x_i}, \vec{x_j})$ is positive semi-definite. Such a point may be a non-unique and non-isolated optimum. The KKT

conditions are particularly simple; the quadratic programming problem is solved when, for all i:

$$\begin{array}{rcl} \alpha_i = 0 & \Rightarrow & y_i f(\vec{x_i}) \geq 1 \\ 0 < \alpha_i < C & \Rightarrow & y_i f(\vec{x_i}) = 1 \\ \alpha_i = C & \Rightarrow & y_i f(\vec{x_i}) \leq 1 \end{array}$$

where $f(\vec{x_i})$ is the evaluation of the support vector machine at point $\vec{x_i}$, and is defined as:

$$f(\vec{x_i}) = \sum_{j=1}^{l} \alpha_j k(\vec{x_i}, \vec{x_j})$$

Those input samples with non-zero α_i values at the end of training are called support vectors, and the samples where $\alpha_i = C$ are called bound support vectors. In classification problems, these support vectos define the boundary between two classes, while the C parameter allows some "slack" in the system that permits samples to be on the "wrong" side of the decision boudnary, which helps prevent or reduce overfitting to the input set. It also affects the size of the support vector set.

LIBSVM [6] includes for kernel functions: linear, polynomial, radial basis function (RBF), and sigmoid. To train an SVM, the user must select the proper C value as well as any required kernel parameters. The various kernel functions are defined as:

$$\begin{array}{ll} linear & k(\vec{u}, \vec{v}) = \vec{u}^T \cdot \vec{v} \\ polynomial & k(\vec{u}, \vec{v}) = (\gamma \vec{u}^T \cdot \vec{v} + coef0)^{degree} \\ RBF & k(\vec{u}, \vec{v}) = e^{-\gamma |\vec{u} - \vec{v}|^2} \\ sigmoid & k(\vec{u}, \vec{v}) = \tanh(\gamma \vec{u}^T \cdot \vec{v} + coef0) \end{array}$$

which means that training a support vector machine for these kernels requires setting 1, 4, 2, and 3 parameters respectively.

Keerthi [7] proposed a method for obtaining both the leave-one-out (LOO) error and its gradient with respect to the SVM control parameters C and γ for L2-SVM with RBF kernel. Interestingly, this method is computationally faster than five-fold cross-validation. Chung et. al. [8] extended this work to also cover L1-SVMs. Both Keerthi and Chung et. al. used these differentiable error bounds to develop a gradient-descent based method for parameter selection using BFGS quasi-Newton methods. They showed that the method can converge very quickly, but it is sensitive to the initial conditions, and in those cases where the initial point was "far" from the optimum in a "flat" region, the search sometimes searched fruitlessly and aimlessly for the minimum.

SVM parameter selection can be viewed as an optimization process, and many of the standard techniques from both optimization and DOE fields may be applied directly to this problem. As Keerthi and Chung et. al. observed, obtaining a differentiable error function allows one to use gradient-based approaches, such as BFGS quasi-Newton gradient-descent optimization.

It is possible that the best solution would be to use our approach to identify a suitable initial condition, and then use the gradient-based approach for the local search to identify the optimal solution.

III. Implementation

The parameter selection algorithm was implemented in less than 100 lines of Python on top of LIBSVM's Python interface. The core of the system is a Python class *Search*, which is passed three parameters: minimums, maximums, and a problem. Minimums and maximums are lists of the minimal and maximal values for each parameter, and the problem is a pointer to a class that encapsulates the SVM problem and manages the evaluation of SVM performance for a given set of parameter setting. *Search* maintains a list of all evaluated parameter settings and their estimated error performance, and does not repeat evaluations for a given parameter setting, but rather uses the stored value.

The method *explore* creates the DOE sample pattern for the current search iteration given the current min-max range for each parameter using the template from Figure 1, and then measures the performance over those samples.

The method *search* is an iterative process that starts with the full range $< min_i, ..., max_i >$for each parameter i. (For RBF kernels, there are only two parameters, C and γ). In each iteration the system uses *explore* to sample the parameter space, and it selects the sample with the best performance (highest accuracy for classification problems, or lowest mean squared error for regression problems). At this point the range is halved, and the best point is used as the center point of the new range, unless the new range would extend outside the user-specified range. In this case the center point is the point closest to the best point where the new range is contained in the user-specified range. The system now repeats this process as many times as desired, and at the end the best sample is chosen.

By storing the search parameter bounds in a list, the search algorithm itself is independent of the number of parameters in the search space. This allows us to re-use the same algorithm and code for all four kernels. In addition, since the experiment designs are the union of a 3^d and a 2^d design, the per-iteration search pattern also scales naturally to the different number of parameters required for each of the kernels.

IV. Results

We compare the performance of the proposed search method with the standard grid search method using LIBSVM and the RBF kernel, both in terms of the quality of the final result and the work required to obtain that result. Chih-Jen Lin kindly shared with us the LIBSVM datasets used in Chung et. al. [8]: banana, image, splice, tree, waveform, and ijcnn. The grid search was done with twenty samples per parameter with uniform resolution in log_2-space. For both search methods, the parameter ranges were $log_2 C \in \{-5, \ldots, 15\}$ and $log_2 \gamma \in \{-15, \ldots, 3\}$.

Table I shows the result of both our search method and an exhaustive grid-based search for each dataset using RBF kernels. Using a paired t-test on the final accuracy results, we find that the two approaches have comparable accuracy with 85% probability. However, the search method utilized five iterations and no more than 50 samples to find the C and γ values that gave the best cross-validation error, whereas

TABLE I
RESULTS

dataset	search				grid			
	$\ln_2 C$	$\ln_2 \gamma$	#fun	accuracy	$\ln_2 C$	$\ln_2 \gamma$	#fun	accuracy
banana	11.25	-1.5	49	93.0	12.89	-1.74	400	92.75
diabetes	5.62	-10.5	47	79.7	7.63	-13.10	400	79.49
ijcnn	3.44	1.59	50	98.91	3.42	2.05	400	98.89
image	8.44	-4.6	45	96.5	13.95	-7.42	400	96.69
ringnorm	7.19	-3.47	46	98.75	4.47	-3.63	400	98.75
splice	12.19	-7.97	46	87.80	2.37	-7.42	400	88.30
titanic	-1.25	-1.5	49	80.67	-0.79	-2.68	400	80.67
tree	10.62	-6.56	47	89.00	15.00	-8.37	400	89.14
twonorm	1.87	-6.00	47	98.00	1.32	-5.53	400	97.75
waveform	1.56	-5.16	47	93.50	0.26	-3.63	400	93.25

the grid search required 20 samples per dimension, for 400 samples overall, an eight-fold difference in computational cost. It is interesting to note that the actual parameter settings may differ widely from that found in the grid search, such as the *image* dataset.

Figure I contains contour and surface plots for some of the datasets. The arrows on the projected contour plot show the progression of the search method's best parameter estimate. The final, optimal parameter settings are shown with an 'x'. It is interesting to see that some of the surfaces have local minima that can trap or deceive gradient-based search algorithms. For example, the secondary ridge in banana that appears to peak around $log_2 C = 12$, $log_2 \gamma = -16$ with value around 65, while the true peak is at $11.25, -1.5$ and value 93. Also note the occassional isolated peaks in both diabetes and waveform which could trap gradient-based search methods. Finalit is interesting to note the extremely flat regions near the periphery of the graphs, which appear to give the lower bound on performance for the algorithms. These flat regions could also tend to confuse gradient-based search algorithms because the surface is so flat that the search is more vulnerable to small errors or noise in the gradient estimation.

V. CONCLUSIONS

We have presented an algorithm based on techniques from DOE that can reliably find very good parameter settings for SVMs with RBF kernels with relatively little effort across a wide range of machine learning problems. Additionally, this algorithm might be used to obtain a good starting point for Keerthi's gradient-descent method for RBF kernels, which can be sensitive to the initial conditions of its search.

Currently this work is being extended to work across each of the SVM kernels supported by LIBSVM, but some care must be taken to ensure that the search is robust in the face of infeasible solutions which are more likely with some of the other kernels. In addition, a kernel-search algorithm is being added on top of the per-kernel parameter search so the system can automatically identify the best kernel and its parameter settings.

ACKNOWLEDGEMENTS

The author should like to thank Chih-Jen Lin for LIBSVM and for his generous sharing of his expertise, code, and data sets for this problem.

REFERENCES

[1] D. C. Montgomery, *Design and analysis of experiments*, 4th ed. New York, New York: John Wiley and Sons, 1997.

[2] V. Vapnik, *The nature of statistical learning theory*, 2nd ed., ser. Statistics for engineering and information science. New York, New York: Springer-Verlag, 2000.

[3] N. Cristianini and J. Shawe-Taylor, *An introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge, UK: Cambridge University Press, 2000.

[4] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998, http://svm.first.gmd.de/papers/Burges98.ps.gz.

[5] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods - Support Vector Learning*, B. Scholkopf, C. Burges, and A. Smola, Eds. MIT Press, 1998, http://research.microsoft.com/˜jplatt/smo-book.pdf.

[6] C.-C. Chang and C.-J. Lin, "LIBSVM: a library for support vector machines," September 14 2002. [Online]. Available: http://www.csie.ntu.edu.tw/˜cjlin/papers/libsvm.pdf

[7] S. Keerthi, "Efficient tuning of SVM hyperparameters using radius/margin bound and iterative algorithms."

[8] K.-M. Chung, W.-C. Kao, T. Sun, L.-L. Wang, and C.-J. Lin, "Radius margin bounds for support vector machines with the RBF kernel," *Neural Computation*, vol. 15, no. 11, pp. 2643–2681, November 2003.

Fig. 2. Accuracy contour and surface plots