

Model Selection for Support Vector Classifiers via Direct Simplex Search

Gilles Cohen and Patrick Ruch

Medical Informatics Service,
University Hospital of Geneva,
1211 Geneva, Switzerland
{Gilles.Cohen,Patrick.Ruch}@sim.hcuge.ch

Mélanie Hilario

Artificial Intelligence Laboratory,
University of Geneva,
1211 Geneva, Switzerland
Mélanie.Hilario@unige.ch

Abstract

This paper addresses the problem of tuning hyperparameters in support vector machine modeling. A Direct Simplex Search (DSS) method, which seeks to evolve hyperparameter values using an empirical error estimate as steering criterion, is proposed and experimentally evaluated on real-world datasets. DSS is a robust hill climbing scheme, a popular derivative-free optimization method, suitable for low-dimensional optimization problems for which the computation of the derivatives is impossible or difficult. Our experiments show that DSS attains performance levels equivalent to that of GS while dividing computational cost by a minimum factor of 4.

Introduction

Support vector machines (SVM) are a powerful machine learning method for classification problems. However, to obtain good generalization performance, a necessary condition is to choose sufficiently good model hyperparameters (i.e regularization parameter (C) and kernel parameters) depending on the data. The choice of SVM model parameters can have a profound effect on the resulting model's generalization performance. Most approaches use trial and error procedures to tune SVM hyperparameters while trying to minimize the training and test errors. Such an approach may not really obtain the best performance while consuming an enormous amount of time. A more systematic and reliable approach which is very common 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. Recently others approaches to parameters tuning have been proposed (Chapelle *et al.* 2002; Keerthi 2003; Chung *et al.* 2003). These methods use a gradient descent search to optimize a validation error, a leave-one-out (LOO) error or an upper bound on the generalization error (Duan, Keerthi, & Poo 2001). However, gradient descent oriented methods may require restrictive assumptions regarding, e.g., continuity or differentiability. Typically the

criteria, such as LOO error, are not differentiable, so that approaches based on gradient descent via cross-validation are generally not applicable. For such non differentiable criteria other approaches based on Evolutionary Algorithms have been investigated (Cohen, Hilario, & Geissbuhler 2004; Friedrichs & Igel 2004; Runarsson & Sigurdsson 2004).

In the present work we propose a Direct Simplex Search methodology to tune SVM hyperparameters and illustrate its effectiveness in classification tasks. The main advantages of a DSS strategy lie in the suitability for problems for which it is impossible or difficult to obtain information about the derivatives. The paper is organized as follows. In the first Section the Direct Simplex Search algorithm is described. In the second Section DSS for SVM model selection in classification tasks are described. Experiments conducted to assess this approach as well as results are described in the third Section. Finally, the last Section draws a general conclusion.

Direct Simplex Search Algorithm

Direct search methods belong to a class of optimization methods that do not compute derivatives. The direct search method we used is the Nelder-Mead (NM) Simplex method (J.A.Nelder & R.Mead 1965). The NM method is the most popular direct search method, used in solving a lot of problems, especially in chemistry, chemical engineering, and medicine. Although there are no theoretical results on the convergence of the algorithm, it works very well on a range of practical problems. The Nelder-Mead method is conceptually simple. It performs a search in n dimensional space using heuristic ideas. Its main strength are that it requires no derivatives to be computed and that it doesn't require the objective function to be smooth. The NM method attempts to minimize a scalar-valued nonlinear function of n real variables using only function values, without any derivative information (explicit or implicit). Many of the best known direct search methods, including the NM method, maintain at each step a nondegenerate simplex, a geometric figure defined by $n+1$ vertices (real n -vectors). (For example, a simplex in two dimensions is a triangle and a tetrahedron forms a simplex in three dimensions.) Suppose that we are minimizing the function $f(\mathbf{x})$, where \mathbf{x} denotes a real n -vector.

The NM method includes four possible operations on the current simplex, each associated with a coefficient: reflection (ρ), expansion (ξ), contraction (γ), and shrinkage (σ).

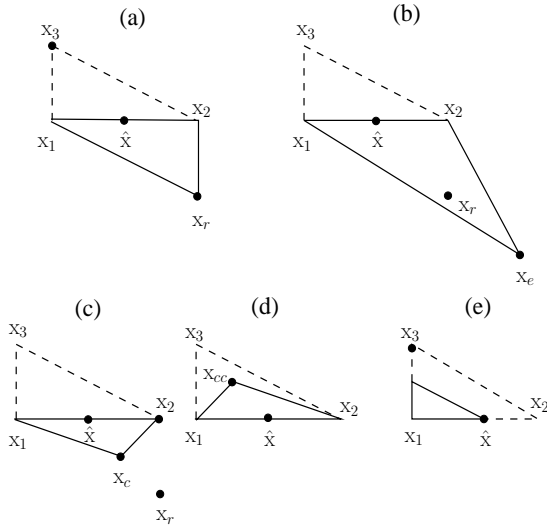


Figure 1: Nelder-Mead simplices after (a) reflection, (b) expansion, (c) outside contraction, (d) inside contraction and (e) shrinkage. The original simplex is drawn with a dashed line.

The result of each NM iteration is either: (1) a single new vertex, the accepted point which replaces the current worst vertex (the vertex with the largest function value) in the set of vertices for the next iteration; or (2) if a shrink is performed, a set of n new points that, together with the previous best vertex, form the simplex at the next iteration.

A single iteration of the NM method is defined according to the following steps.

- Order** : Order the $n + 1$ vertices to satisfy $f(x_1) \leq f(x_2) \leq \dots \leq f(x_{n+1})$ using a consistent tie-breaking rule.
- Reflect** : Compute the *reflexion point* x_r from $x_r = \hat{x} + \rho(\hat{x} - x_{n+1})$ where $\hat{x} = \sum_{i=1}^n x_i/n$ is the centroid of the n best points. Evaluate $f_r = f(x_r)$. If $f_1 \leq f_r < f_n$, accept the reflected point x_r and terminate the iteration.
- Expand** : if $f_r < f_1$ calculate the *expansion point* x_e : $x_e = \hat{x} + \rho\xi(\hat{x} - x_{n+1})$ and evaluate $f_e = f(x_e)$. If $f_e < f_r$ accept x_e and stop the iteration; otherwise accept x_r and stop the iteration.
- Contract**: If $f_r \geq f_n$, perform a *contraction* between \hat{x} and the better of x_{n+1} and x_r .
 - Outside** If $f_n \leq f_r < f_{n+1}$ perform an *outside contraction*: calculate $x_c = \hat{x} - \gamma\rho(\hat{x} - x_{n+1})$ and evaluate $f_c = f(x_c)$. If $f_c \leq f_r$, accept x_c and terminate the iteration; otherwise go to last step (perform a shrink).
 - Inside** If $f_r \geq f_{n+1}$ perform an *inside contraction*: calculate $x_{cc} = \hat{x} - \gamma(\hat{x} - x_{n+1})$ and evaluate $f_{cc} = f(x_{cc})$. If $f_{cc} < f_{n+1}$, accept x_{cc} and terminate the iteration; otherwise go to next step (perform a shrink).
- Perform a shrink step** Evaluate f at the n points $v_i = x_1 + \sigma(x_i - x_1)$, $i = 2, \dots, n + 1$. The vertices of the

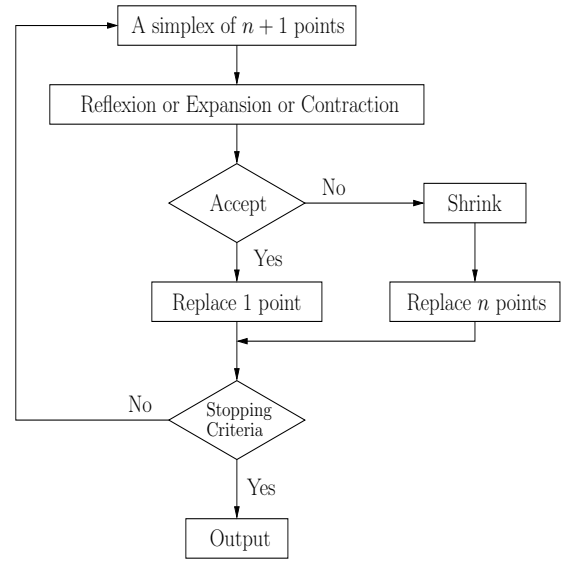


Figure 2: Structure of the Direct Simplex Search method of Nelder-Mead.

simplex at the next iteration consist of x_1, v_2, \dots, v_{n+1}

Common choices for setting the coefficients values are $\rho = 1, \xi = 2, \gamma = 1/2$ and $\sigma = 1/2$. Figure 1 depicts the effect of the different operations, using this setting.

Figure 2 illustrates the structure of the DSS method. One can see that if one of the first three operations is successful, then a single new point replaces the worst vertex. In this case, an iteration requires one or two function evaluations depending on the operation performed (one for reflection or two expansion or contraction). Otherwise a shrink operation is done which requires $n + 2$ function evaluations. After computing one or more trial points and evaluating the objective function at these points, each iteration generates a different simplex for the next iteration. The procedure is iterated until a stopping criterion is satisfied. Usually two stopping criteria are used: either the length of the edges of the current simplex becomes less than a prescribed positive number (the simplex becomes too small) or the function values at the vertices are too close.

Since the Direct Search method cannot be guaranteed to converge to the global optimum, standard approaches to alleviate this problem are to apply the Nelder-Mead algorithm with multiple starting points generated randomly.

SVM Model Selection via DSS

To apply DSS to SVM classification one need to define the model and model quality function to be optimized (model selection criteria).

Support vector machines (Vapnik 1998; Cortes & Vapnik 1995) (SVM) are state of the art learning machines based on the *Structural Risk Minimization principle* (SRM) from statistical learning theory. The SRM principle seeks to minimize an upper bound of the generalization error rather than minimizing the training error (Empirical Risk Minimization

(ERM)). This approach results in better generalization than conventional techniques generally based on the ERM principle.

For a separable classification task, the idea is to map each data point of the training set into a high dimensional space by some function ϕ , and search for a canonical separating hyperplane (\mathbf{w}, b) , with \mathbf{w} the weight vector and b the bias, in this space which maximises the *margin* or distance between the hyperplane and the closest data points belonging to the different classes. When nonlinear decision boundaries are not needed ϕ is an identity function, otherwise ϕ is performed by a non linear function $k(\cdot, \cdot)$, also called a *kernel*, which defines a dot product in the feature space. We can then substitute the dot product $\langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle$ in feature space with the kernel $k(\mathbf{x}, \mathbf{x}_i)$. Conditions for a function to be a kernel are expressed in a theorem by Mercer (Cristianini & J.S. 2000). The optimal separating hyperplane can be represented based on a kernel function:

$$f(\mathbf{x}) = \text{sign} \left(\sum_i^n \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b \right) \quad (1)$$

For a separable classification task, such an optimal hyperplane exists but very often, the data points will be almost linearly separable in the sense that only a few of the members of the data points cause it to be non linearly separable. Such data points can be accommodated into the theory with the introduction of slack variables that allow particular vectors to be misclassified. The hyperplane margin is then relaxed by penalising the training points misclassified by the system. Formally the optimal hyperplane is defined to be the hyperplane which maximizes the margin and minimizes some functional $\theta(\xi) = \sum_{i=1}^n \xi_i^\sigma$, where σ is some small positive constant. Usually the value $\sigma = 1$ is used since it is a quadratic programming problem (QP) and the corresponding dual does not involve ξ and therefore offers a simple optimization problem. The optimal separating hyperplane with $\sigma = 1$ is given by the solution to the following minimization problem:

$$L_P(\mathbf{w}, \xi, b) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \quad (2)$$

$$\text{subject to} \quad y_i(\langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle + b) \geq 1 - \xi_i, \quad \forall i \\ \xi_i \geq 0, \quad \forall i$$

where $b/\|\mathbf{w}\|$ is the distance between origin and hyperplane, ξ_i is a positive slack variable that measures the degree of violation of the constraint. The penalty C is a regularisation parameter that controls the trade-off between maximizing the margin and minimizing the training error. This is a QP, solved by the Karush-Kuhn-Tucker theorem. Let $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_n)^T$ be the n non negative Lagrange multipliers associated with the constraints, the solution to the problem is equivalent to determining the solution of the *Wolfe dual* (Fletcher 1987) problem.

$$L_D(\boldsymbol{\alpha}) = \mathbf{e}^T \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha}^T Q \boldsymbol{\alpha}. \quad (3)$$

$$\text{subject to} \quad \boldsymbol{\alpha} \mathbf{y}^T = 0 \text{ and } 0 \leq \alpha_i \leq C \quad i = 1, \dots, n$$

where \mathbf{e} is a vector of all ones and Q is a $n \times n$ matrix with $Q_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$. The KKT conditions imply that non-zero slack variables can only occur for $\alpha_i = C$. For the corresponding points the distance from the hyperplane is less than $1/\|\mathbf{w}\|$ as can be seen from the first constraint in (2).

Table 1: Some valid Kernel functions

Kernel type	Expression	Hyperpar.
Linear	$k(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle$	C
Polynomial	$k(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + 1)^p$	C, p
Gaussian RBF	$k(\mathbf{x}, \mathbf{z}) = e^{-\gamma \ \mathbf{x} - \mathbf{z}\ ^2}$	C, γ

Model Selection Criteria To obtain a good performance, some parameters in SVMs have to be selected carefully. These parameters include

1. the regularization parameters C , which determine the tradeoff between minimizing model complexity and the training error and
2. parameter of the kernel function, encoded into a vector $\boldsymbol{\theta} = (\theta_1, \dots, \theta_n)$, that implicitly defines the non linear mapping to some high-dimensional feature space.

These ‘‘higher level’’ parameters are usually referred as metaparameters or hyperparameters. Some valid kernel functions are listed in Table 1 with corresponding hyperparameters.

The model selection problem is to select from a candidate set of models the best one so the generalization error

$$e(f_\theta) = \int_{\mathcal{X} \times \mathcal{Y}} \ell(f(\mathbf{x}, y)) dP(\mathbf{x}, y) \quad (4)$$

is minimized over all possible examples drawn from an unknown distribution $\mathcal{P}(\mathbf{x}, y)$. As the data distributions \mathcal{P} in real problems are not known in advance, generalization error is not computable and one needs some reliable estimates of the generalization performance.

There has been some work on efficient methods for estimating generalization performance such as LOO, the $\xi\alpha$ bound (Joachims 2000), the radius margin bound and span bound (Chapelle *et al.* 2002) in SVM classification. A review and a comparative analysis of all these techniques, covering both empirical and theoretical methods, can be found in (Duan, Keerthi, & Poo 2001).

To estimate the quality of the model, we use the popular cross-validation (CV) technique. In k -fold cross-validation the original training set S is randomly partitioned into k non-overlapping subsets S_j of approximately equal size. The learning machine is trained on the union of $(k - 1)$ subsets; the remaining k -th subset is used as a test set and measures the associated classification performance. This procedure is cycled over all possible k test sets, and the average test error gives an estimate of the expected generalization error. The CV error on a training set $S = \{(\mathbf{x}, y)\}_{i=1}^n$ is defined as

$$e_{kCV}(S) = \frac{1}{k} \sum_{j=1}^k \left(\sum_{i=1}^{|S_j|} \ell(f_{S \setminus S_j}(\mathbf{x}_i), y_i) \right) \quad (5)$$

where $S \setminus S_j$ denotes the dataset obtained by removing the subset S_j from the set S , $f_{S \setminus S_j}$ the corresponding classifier and $|S_j|$ the cardinality of subset S_j .

Results

The experimental goal was to assess a DSS optimization method for tuning SVM hyperparameters. For this, a standard grid search method was used as a baseline for comparing the quality of the final result and the computational cost of obtaining that result. To train our SVM classifiers we use an RBF kernel. Thus the corresponding hyperparameter set to tune is $\theta : (\gamma, C)$. Since γ and C have to be positive, we transform them to remove these constraints. We thus minimize with respect to $C = \log_2(C)$ and $\gamma = \log_2(\gamma)$.

We use five different real-world datasets known as banana, image, splice, waveform and tree. Detailed information concerning the first four datasets can be found in (Ratsch 1999). The last dataset (Bailey *et al.* 1993) was formed from geological remote sensing data; one class consists of patterns of trees, and the second class of non-tree patterns. Table 2 lists their basic characteristics.

For both approaches we used 5-fold cross-validation to evaluate our model. Grid search was done by varying with a fixed step-size a range of values taken in $C \in \{2^i | i = -5, \dots, 15\}$ and $\gamma \in \{2^i | i = -16, \dots, 3\}$; performance was assessed using cross-validated accuracy. The step-size corresponds to twenty points per parameter with uniform resolution in the region spanned by (C, γ) . We applied DSS with three randomly generated starting points and kept the best results; such probabilistic restarts allow us to achieve a certain degree of global coverage.

Table 3 shows the comparative results between our search method (DSS) and an exhaustive grid-based search (GS) for each dataset. As one can see optimal parameter settings found by the different approaches may differ widely, as on the waveform dataset. Final results were compared using McNemar’s test (Salzberg 1997; 1999) which revealed no significant difference between the two approaches at the 95% confidence level. However, the DSS method utilized about 100 function evaluations¹ to find the best pair (C, γ) (i.e., which yielded the lowest cross-validation error), whereas GS required four times that number of evaluations. Table 4 shows the significant advantage of using the DSS method in terms of number of evaluations. Furthermore, the computational time incurred for individual function evaluation is typically lower for DSS than for GS. The computational cost of function evaluation is hyperparameter-dependent; for instance training time for large values of the C parameter is necessarily higher than for small values. GS systematically explores large values of C which are on the grid whereas DS explores such large values only if they lead to a solution. Figure 3 shows a surface plot of performance measures for the banana dataset. It can be seen clearly that there are local minima which can trap or deviate local (i.e. gradient-based) methods. For example, for $\log_2 C = 12$ and $\log_2 \gamma = -14$ accuracy is 65% whereas the true peak is at

¹One function evaluation corresponds to building 5 SVM models.

$\log_2 C = 11.25$ et $\log_2 \gamma = -1.5$ where accuracy is 94.5%. The plot also shows extremely flat regions corresponding to very weak performance. In such regions, gradient-based algorithms are particularly vulnerable to noise or to slight inaccuracies in gradient estimation.

Table 2: Dataset main characteristics

Dataset	#att.	#training	#test	#class
banana	2	400	4900	2
image	18	1300	1010	2
splice	60	1000	2175	2
waveform	21	400	4600	2
tree	18	700	11692	2

Table 3: Performance of SVMs for optimum parameter settings using an RBF Gaussian kernel (γ, C) found via DSS and GS methods. Columns γ and C represent $\log_2 \gamma$ and $\log_2 C$ respectively.

Data	DSS			GS		
	C	γ	acc.	C	γ	acc.
banana	3.5	3.5	88.7	8.7	3	88.23
image	3.19	2.62	98.22	7.63	2	98.11
splice	7.06	-2.75	90.43	3.42	-3.9	90.11
waveform	4.74	1.13	89.09	-0.79	-1.2	88.8
tree	3.85	0.71	87.86	10.79	-1	89.09

Table 4: Comparative computational load for both approaches (number of evaluations)

	banana	image	splice	waveform	tree
DSS	79	95	105	101	110
GS	400	400	400	400	400

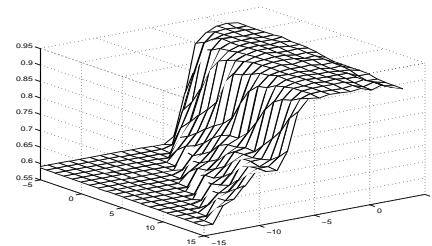


Figure 3: Performance plot for the banana dataset. Accuracy (the vertical axis) is plotted against different values of hyperparameters C ($\log_2 C \in [-5, 15]$) and γ ($\log_2 \gamma \in [-15, 5]$).

Conclusion

We presented an algorithm based on a DSS method that can reliably find very good hyperparameter settings for SVMs

with RBF kernels in a fully automated way. We selected the DSS method because of its robustness, simplicity and ease of implementation. Our experiments have shown that DSS and GS attain equivalent generalization performance, but that DSS is at the very least four times faster than GS. To improve our method we plan to merge DSS with Genetic Algorithms by first performing a coarse search for the global minimum by means of a genetic algorithm and then refining the solution by a DSS approach. We plan to extend this work to other kernels as well as to a larger set of hyperparameters such as γ , C_+ and C_- in asymmetrical-margin RBF SVMs.

References

- Bailey, R.; Pettit, E.; Borochoff, R.; Manry, M.; and Jiang, X. 1993. Automatic recognition of usgs land use/cover categories using statistical and neural networks classifiers. In WA:SPIE., ed., *SPIE OE/Aerospace and Remote Sensing*. Bellingham.
- Chapelle, O.; Vapnik, V.; Bousquet, O.; and Mukherjee, S. 2002. Choosing multiple parameters for support vector machines. *Machine Learning* 46(1):131–159.
- Chung, K.; Kao, W.; Sun, C.; Wang, L.; and Lin, C. 2003. Radius margin bounds for support vector machines with the rbf kernel. *Neural Comput.* 15(11):2643–2681.
- Cohen, G.; Hilario, M.; and Geissbuhler, A. 2004. Model selection for support vector classifiers via genetic algorithms. An application to medical decision support. In *International Symposium on Biological and Medical Data Analysis*.
- Cortes, C., and Vapnik, V. 1995. Support vector networks. *Machine Learning* 20(3):273–297.
- Cristianini, N., and J.S., T. 2000. *An Introduction to Support Vector Machines*. Cambridge University Press.
- Duan, K.; Keerthi, S.; and Poo, A. 2001. Evaluation of simple performance measures for tuning svm hyperparameters. In *Technical Report CD-01-11, Singapore*.
- Fletcher, R. 1987. *Practical Methods of Optimization*. John Wiley and Sons.
- Friedrichs, F., and Igel, C. 2004. Evolutionary tuning of multiple svm parameters. In *12th European Symposium on Artificial Neural Networks(ESANN)*.
- J.A.Nelder, and R.Mead. 1965. A simplex method for function minimization. *ComputerJournal* 7:308–313.
- Joachims, T. 2000. Estimating the generalization performance of a SVM efficiently. In Langley, P., ed., *Proceedings of ICML-00, 17th International Conference on Machine Learning*, 431–438. Stanford, US: Morgan Kaufmann Publishers, San Francisco, US.
- Keerthi, S. 2003. Efficient tuning of svm hyperparameters using radius/margin bound and iterative algorithms. In *IEE Transactions on Neural Networks*, 1225–1229.
- Ratsch, G. 1999. Benchmark data sets. available on-line: <http://ida.first.gmd.de/raetsch/data/benchmarks.htm>.
- Runarsson, T., and Sigurdsson, S. 2004. Asynchronous parallel evolutionary model selection for support vector machines. *Neural Information Processing - Letters and Reviews* 3:1065–1076.
- Salzberg, S. 1997. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery* 1(3).
- Salzberg, S. 1999. On comparing classifiers: A critique of current research and methods. *Data Mining and Knowledge Discovery* 1(12).
- Vapnik, V. 1998. *Statistical Learning Theory*. Wiley.