Evolving Kernels for Support Vector Machine Classification

Keith Sullivan Sean Luke

Department of Computer Science, George Mason University
4400 University Drive MSN 4A5, Fairfax, VA 22030 USA
{ksulliv, sean}@cs.gmu.edu

ABSTRACT

While support vector machines (SVMs) have shown great promise in supervised classification problems, researchers have had to rely on expert domain knowledge when choosing the SVM's kernel function. This project seeks to replace this expert with a genetic programming (GP) system. Using strongly typed genetic programming and principled kernel closure properties, we introduce a new algorithm, called KGP, which finds near-optimal kernels. The algorithm shows wide applicability, but the combined computational overhead of GP and SVMs remains a major unresolved issue.

Categories and Subject Descriptors

I.5.1 [**Computing Methodologies**]: Pattern Recognition—*Statistical*

General Terms

Experimentation

Keywords

Genetic Programming, Support Vector Machines

1. INTRODUCTION

The Support Vector Machine (SVM) is a kernel-based classification learning technique which attempts to abstract out the issue of learning bias. Instead, the user provides a supervised function called a *kernel* which imparts a domain-specific transformation on the data to enable it to be classified straightforwardly. Finding such a kernel is nontrivial, and indeed many experimenters simply rely on one of a small number of predefined kernels from the literature. In this paper, we apply genetic programming (GP) to discover good kernels for the particular set of data being learned.

In supervised classification, we are given a set of training examples $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \in X \times Y$ and try to learn a function $y = f(x)$ that predicts the classification y of unseen examples x with minimal error. Support vector machines use a function Φ to first map examples into a higher dimensional space, and then construct a separating hyperplane there. The idea is to transform the data into a new space where the data is linearly separable. Then, using the hyperplane as a decision function (i.e., f), we can classify unseen data based on which side of the hyperplane they lie.

Transforming data with Φ can be expensive with high dimensional spaces. Instead, an SVM employs a kernel function k which gives the dot product of the two examples in the higher dimensional space without actually transforming them into that space. This notion, dubbed the *kernel trick*, allows us to perform the Φ transformation for purposes of classification to large dimensional spaces.

One issue with SVMs is finding an appropriate kernel for the given data. Most research relies on *a priori* knowledge to select the correct kernel, and then tweaks the kernel parameters via machine learning or trial-and-error. While there exist rules-of-thumb for choosing appropriate kernel functions and parameters, this limits the usefulness of SVMs to expert users, especially since different functions and parameters can have widely varying performance. Depending on the domain, such predefined functions are also unlikely to be optimized for the problem. A potential solution is to use genetic programming to evolve the kernel and associated parameters.

Not every function can be used as a kernel: kernels typically satisfy *Mercer's Theorem* as discussed later. Our approach is to take advantage of kernel operations which guarantee closure: for example, the sum of two kernels is a kernel. Our genetic representation is a strongly-typed genetic program [18] whose outer nodes are standard kernels and input vectors, and various kernel parameters, and whose inner nodes are operations on those kernels to produce a composite function guaranteed to be a kernel by closure properties.

This paper is organized as follows: Section 2 presents related work. Sections 3 and 4 give some background information on SVMs and genetic programming. The main algorithm is presented in Section 5, and the experimental design and results are discussed in Section 6. Section 7 discusses computational complexity. Section 8 offers some conclusions and suggestions for future work.

2. RELATED WORK

Evolutionary algorithms (EA) have been applied to SVMs in two ways: using genetic programming to evolve kernel functions, and using evolutionary algorithms to evolve ker-

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

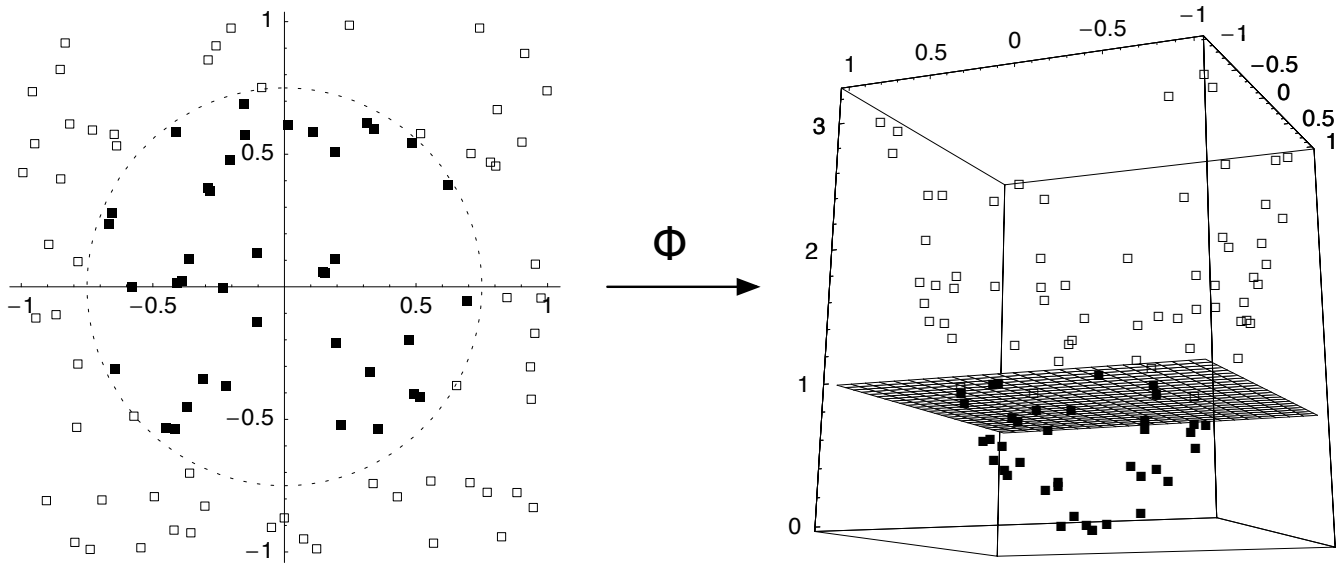

Figure 1: The transformation function $\Phi : (x, y) \rightarrow (x, y, x^2 + y^2)$ maps the non-linearly separable 2D data on the left to the linearly separable 3D data on the right

nel parameters. Recently two research groups have applied genetic programming to evolve a kernel function for classification tasks [10, 9]. Both papers use GP to evolve a function from basic functions. Nether paper guarantees that the evolved function satisfies Mercer's Theorem, and so cannot guarantee that the result is optimal. However, both report good classification accuracy on several standard datasets.

Other researchers have taken a different approach: using an EA to evolve parameter values for a known kernel function. Friedrichs and Igel use evolutionary strategies (ES) to evolve covariance matrices in a Gaussian kernel [8]. Phienthrakul and Kilsirikul [20] used ES to learn the weights in a weighted linear combination of Gaussian radial basis functions. Souza et al [6] use Particle Swarm Optimization (PSO) to learn optimal parameters in a Gaussian kernel function for multi-class classification, while Huang and Wang use a GA for the same task [13, 12]. Runarsson and Sigurdsson [21] use a parallel ES to learn optimal parameters in a Gaussian kernel. Mierswa [17] combines PSO and ES to solve the constrained optimization problem related to SVMs.

Using the idea of a weighted linear combination of kernels, machine learning has been used successfully to learn the appropriate weights [4, 5, 19, 25]. A slightly different formulation used gradient descent to learn multiple parameters in a Gaussian kernel function [3, 14].

3. SUPPORT VECTOR MACHINES

Support Vector Machines are a learning technique which trades off accuracy for generalization error. SVMs build a hyperplane which divides examples such that examples of one class are all on one side of the hyperplane, and examples of the other class are all on the other side. However, interesting data is rarely linearly separable. Thus, the SVM first transforms the data to a higher dimensional space where it *is* linearly separable (see Figure 1), and then applies the hyperplane. Because the transformation may be to many, even infinite, dimensions, the SVM does not actually perform the transformation. Instead, noting that the dot product is all that is necessary to compute the optimal hyperplane, the SVM composes the transformation Φ and the dot product in the higher dimensional space into a single kernel function k, which computes the dot product of two vectors when they are transformed into the higher space. Note that the kernel need not actually *do* any transformation to provide this dot product! This is called the *kernel trick*.

First, we discuss building the hyperplane. Consider input data of the form (x_i, y_i), where the vectors x_i are in a dot product space $\mathcal{H}$ (such as a real-valued multi-dimensional space), and y_i are the class labels. Formally, any hyperplane in $\mathcal{H}$ is defined as

$$\{x \in \mathcal{H} | \langle \mathbf{w}, x \rangle + b = 0\} \qquad \mathbf{w} \in \mathcal{H},\, b \in \mathbb{R}$$

where $\mathbf{w}$ is a vector orthogonal to the hyperplane and $\langle \cdot, \cdot \rangle$ represents the dot product. In a SVM, the idea is to find the hyperplane that maximizes the minimum distance (called the *margin*) from any training data point (Figure 2). The following constraint problem describes the optimal hyperplane:

$$\min_{\mathbf{w} \in \mathcal{H},\, b \in \mathbb{R}} \quad \tau(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||^2 \tag{1}$$

$$\text{subject to} \quad y_i \left(\langle \mathbf{x_i}, \mathbf{w} \rangle + b \right) \geq 1 \tag{2}$$

for $i = 1, 2, \ldots m$ where m is the number of training examples. This optimal hyperplane minimizes training error while maximizing the margin, which is believed to translate to maximizing generalization ability.

We now apply the kernel trick to "kernalize" the hyperplane, (i.e., we use the kernel to transform the dot product to a higher dimensional space), which results in the following dual form of the optimization problem:

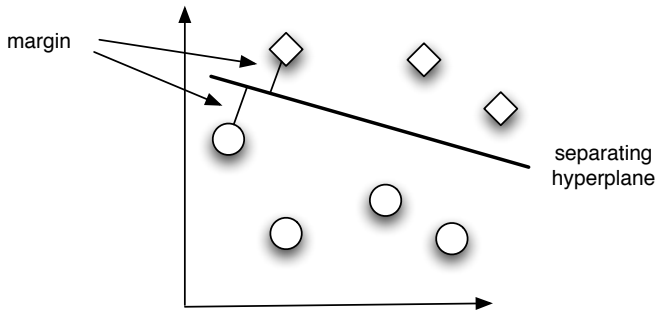

Figure 2: The margin is the distance from a serarating hyperplane and each data point. SVMs try to maximize the minimum distance to increase the ability to classify unseen data.

$$\max_{\alpha \in \mathbb{R}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \tag{3}$$

$$\text{subject to} \quad \alpha_i \geq 0, \tag{4}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0. \tag{5}$$

Any vector x_i which lies on the margin, i.e.,

$$\alpha_i \left[y_i \left(\langle \mathbf{w}, \mathbf{x_i} \rangle + b \right) \right] = 1$$

is called a *support vector*. The optimal hyperplane is thus determined by data examples which form this set of support vectors.

At this point, we do not know if a separating hyperplane is the best solution, or even if one exists. Allowing a certain fraction of outliers can alleviate this problem by making the formulation less brittle. Called *soft margin hyperplanes*, the idea is based on slack variables ζ which allow the presence of outliers without influencing the hyperplane:

$$\zeta_i \geq 0 \tag{6}$$

which results in a hyperplane

$$y_i \left(\langle \mathbf{x_i}, \mathbf{w} \rangle + b \right) \geq 1 - \zeta_i. \tag{7}$$

Thus, a classifier that generalizes well is found by controlling the sum of the slack variables. A term is added to the objective functions to prevent the trivial solution where all ζ_i take on large values. The simplest solution with slack variables, called C-SVC (support vector classifier), tries to determine the tradeoff between minimizing training error and maximizing the margin. However, there is no intuitive *a priori* manner to determine this tradeoff [22]. An alternative method, called ν-SVC, seeks to control the number of margin errors and support vectors via the following optimization problem [23]:

$$\max_{\alpha \in \mathbb{R}^m} \quad W(\boldsymbol{\alpha}) = -\frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \tag{8}$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq \frac{1}{m}, \tag{9}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0, \tag{10}$$

$$\sum_{i=1}^{m} \alpha_i \geq \nu. \tag{11}$$

ν is simultaneously an upper bound on the fraction of margin errors and a lower bound on the fraction of support vectors [23], and so it provides the tuning parameter for how soft the hyperplane is.

3.1 Kernel Closure

Kernel functions allow us to compute dot products in higher dimensional spaces without having to explicitly map into these spaces. A kernel function must satisfy various mathematical properties. The main question is how to determine if a given function k is a kernel. Given $x_1, x_2, \ldots x_m \in X$, the $m \times m$ matrix K with elements $K_{ij} = k(x_i, x_j)$ is called the *Gram matrix* (or *kernel matrix*). If the Gram matrix is positive definite, then by Mercer's Theorem, k is a kernel [22]. While any function with a positive definite Gram matrix is a kernel, one of following functions is typically used [22]:

$$\text{Polynomial:} \quad K(x_i, x_j) = \langle x_i, x_j \rangle^d \tag{12}$$

$$\text{Gaussian:} \quad K(x_i, x_j) = e^{-\gamma * ||x_i - x_j||^2} \tag{13}$$

$$\text{Sigmoid:} \quad K(x_i, x_j) = \tanh\left(\gamma \langle x_i, x_j \rangle + r\right) \tag{14}$$

where $d, \gamma, r \in \mathbb{R}$. New kernel functions can be constructed by combining known kernel functions in an appropriate manner. The proposition below, from [24, Propositions 3.22, 3.24], presents several closure properties, i.e., mathematical manipulations which form new kernels from old ones.

PROPOSITION 1. *Let k_1 and k_2 be kernels over $X \times X$, $X \subseteq \mathbb{R}$, $a \in \mathbb{R}^+$. Then the following functions are kernels:*

(a) $k(x, z) = k_1(x, z) + k_2(x, z)$
(b) $k(x, z) = a * k_1(x, z)$
(c) $k(x, z) = k_1(x, z) * k_2(x, z)$
(d) $k(x, z) = e^{k_1(x, z)}$

4. GENETIC PROGRAMMING

We use a standard GP representation in the form of parse trees equivalent to Lisp s-expressions. Each node in the tree is a function, and its child subtrees form arguments to that function. Genetic programming relies on the principle of closure, i.e, each node may take any subtree as a child. In basic form, closure allows any non-terminal node to be a parent of any other kind of node. Strongly typed genetic programming (STGP) places additional type constraints on nodes specifying which nodes may link with which others,

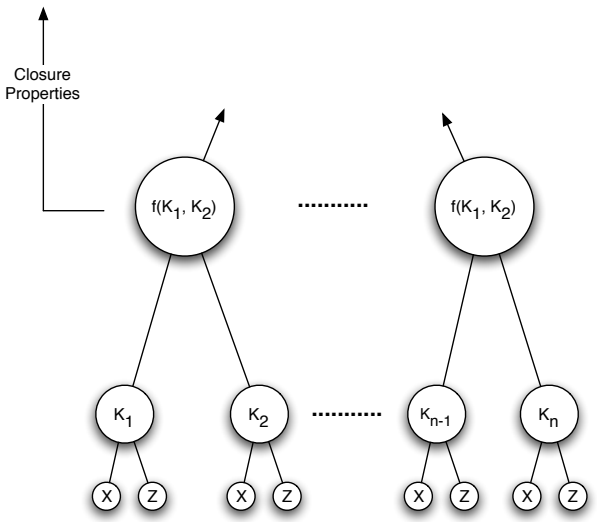

Figure 3: Inside KGP, individuals are formed starting with the three basic kernel functions, before applying the closure rules.

and is typically used to allow child nodes to pass data which their parent is guaranteed to be able to read [18]. This allows the genetic operators to generate semantically correct trees.

One issue in genetic programming is the unconstrained growth of individuals without justifying performance improvements. Typically this growth, called *bloat*, may be limited by special genetic operators that restrict the maximum depth of an individual. An alternative technique, parsimony pressure, penalizes the size of individual by making it less fit. One simple technique is *lexicographic tournament selection* which works the same way as normal tournament selection, but if multiple individuals have the same fitness, then the one with the shortest length is chosen [16].

5. KERNEL GP

Our algorithm is called Kernel GP (KGP). KGP evolves a kernel using strongly typed genetic programming and kernel closure properties. Starting with standard kernel functions, individuals are formed through the closure rules in Proposition 1 and STGP (see Figure 3). Terminal nodes are the two feature vectors $\mathbf{x}$ and $\mathbf{z}$, and the first non-terminal node is a basic kernel function: polynomial, Gaussian, or sigmoid, with randomly-chosen parameters (d, r, and γ) embedded within it. Nodes above this first level are composition rules chosen from Proposition 1. Note that the kernels, $K_1, K_2 \ldots K_n$ shown in Figure 3 are not necessarily the same, nor do they have the same parameter values. Standard GP operators perform mutation and crossover to create the population for the next generation. Lexicographic tournament selection is used to prevent growth of the kernel functions without any increase in fitness. STGP ensures that selection, crossover, and mutation will not generate an invalid kernel function.

Evaluation uses *k-fold cross validation*: first, the training set is split into k equal pieces (called *folds*). Then, k training runs are performed, where each time, we leave one piece out and use it as an independent validation set. An individual's fitness is then the average of k validations. In k-fold cross validation, every data point appears once in the testing set, and $k-1$ times in the training set, thus reducing the dependence on how the data is divided. As k increases, the average performance estimate will be very accurate. However, computational time increases since the training algorithm is performed $k-1$ times. In pseudocode, KGP is as follows:

```
Kernel GP
Initial Settings
  For each individual i
    i = generate a random kernel based on closure rules,
        basic kernels and STGP
Algorithm
Repeat
  For each individual i
    Assess fitness by performing k-fold cross validation
      using individual i and the training dataset
  Select individuals for next generation using
        lexicographic tournament selection
  Apply crossover and mutation to create new population.
```

6. EXPERIMENTS AND RESULTS

KGP was run with several datasets to test accuracy and scalability. Table 1 shows the datasets used, all of which are from the UCI Machine Learning Repository [7]. The split into training and testing sets for the Vowel dataset is from the UCI Repository. The size of the training and testing sets for the Bupa Liver, Pima Indians, Glass2, Heart Disease, Australian Credit, German Credit, and Ionosphere datasets is the same as specified in [20]. The splits for the remaining datasets are 60% to the training set and 40% to the testing set. All datasets used 10-fold cross-validation during training.

KGP was implemented using the ECJ library [15] with individual evaluation performed in LIBSVM [2]. Experiments consisted of 100 independent random splits of the data into testing and training sets as described above. For each split, KGP was trained using a population of 400 individuals for 50 generations. The best individual found was then used to determine testing accuracy using the testing set. Final results are the average of the 100 accuracy measurements. For all experiments, $\nu = 0.05$ and the tournament size was seven.

We compared this technique to runs we performed using an alternative technique, *grid search*. Grid search finds optimal kernel parameters when using a Gaussian kernel [11]. The idea is to try multiple (ν, γ) pairs ($\nu = 2^{-5}, 2^{-3}, \ldots 2^{15}$, $\gamma = 2^{-15}, 2^{-13}, \ldots 2^{3}$), and then choose the pair with the highest cross-validation accuracy (recall that γ is a parameter in the Gaussian kernel). Performance is then measured as the classification accuracy on the testing examples.

Table 2 shows the average classification accurcy and 95% confidence interval for both KGP and SVM-Grid on each dataset. For the Ionosphere, Iris, and Wine datasets, KGP performs better than grid search, and on Wisconsin Breast Cancer, Heart Disease, and Vowel, KGP performs worse than grid search. All comparisons are statistically significant at the 95% confidence level.

Comparison with published results combining SVM and EAs is difficult due to different experimental methodologies.

Table 1: Datasets used in the experiments.

Dataset	Number of Features	Number of Classes	Number of Training Examples	Number of Testing Examples
Wisconsin Breast cancer	10	2	410	273
Heart Disease	13	2	180	90
Ionosphere	34	2	234	117
Iris Plant	4	3	90	60
Wine Recognition	13	3	107	71
Vowel Recognition	10	11	528	426

Table 2: Mean and 95% confidence intervals for KGP and Grid-SVM

Dataset name	KGP	SVM-Grid
Wisconsin Breast cancer	$0.9546 \pm 7.539 \times 10^{-4}$	$0.9637 \pm 2.000 \times 10^{-4}$
Heart Disease	$0.7759 \pm 7.819 \times 10^{-4}$	$0.8273 \pm 7.80 \times 10^{-4}$
Ionosphere	$0.9404 \pm 4.028 \times 10^{-4}$	$0.9327 \pm 4.325 \times 10^{-4}$
Iris Plant	$0.9740 \pm 5.963 \times 10^{-4}$	$0.9609 \pm 5.70 \times 10^{-4}$
Wine Recognition	$0.9747 \pm 3.599 \times 10^{-4}$	$0.9719 \pm 3.891 \times 10^{-4}$
Vowel Recognition	0.6645	0.6406

Several researchers do not test the evolved kernel function on unseen data, while others do not provide enough information for independent verification. Comparisons with published SVM results are also difficult for the same reasons. Still, some comparisons can be made. Souza et al [6] report a classification accuracy of 0.9882 using k-fold validation on the training set for the Vowel dataset. KGP returns an average accuracy of 0.9981 on the training set.

KGP uses non-obvious kernels to achieve high performance. For example, on the Ionosphere dataset, the kernels

$$K(x,y) = e^{2.8472345 * e^{-0.13789417 * ||x-y||^2}}$$

$$K(x,y) = e^{e^{e^{-0.08027894 * ||x-y||^2}}}$$

produces an accuracy of 0.9829 and 0.9145 respectively, and on the Iris dataset the kernels

$$K(x,y) = \tanh\left(38.691612 \langle x_i, x_j \rangle + 37.45266\right) * \langle x_i, x_j \rangle^{11.152326}$$

$$K(x,y) = e^{\langle x_i, x_j \rangle^{0.43298903}}$$

both produce an accuracy of 1.0. Note that all the kernels above satisfy Mercer's Theorem.

7. COMPUTATIONAL COMPLEXITY

The experiments were conducted on small datasets in the previous section due to exceptionally high runtimes for larger datasets. Despite code optimizations and a significant amount of computational power, the question remains: why do large datasets take so much longer than smaller datasets?

The discussion in [1] can partially explain these long runtimes. One issue affecting computational complexity is the number of support vectors. Recently, [26] showed that the number of support vectors k increases linearly with the number of training examples n. More specifically,

$$k/n \rightarrow 2\mathcal{B}_k \tag{15}$$

where $\mathcal{B}_k$ is the smallest classification error achievable with kernel k.

Most SVM training algorithms must store the active portion of the kernel matrix in memory. If the memory requirements exceed available memory, then the training time increases since some coefficients are computed multiple times. Thus, equation 15 suggests memory requirements scale at least like $\mathcal{B}^2 n^2$, which can be prohibitive for large and/or noisy problems. Note that computing multiple coefficients inside KGP can be expensive, especially as the size of the individual increases. This additional runtime is compounded by the size of the GP population. These reasons partially explain the exceedingly slow runtimes for KGP on large datasets such as the USPS Handwriting classification dataset.

8. CONCLUSIONS

KGP is a combination of SVM and GP designed to reduce the burden of *a priori* knowledge on the researcher. KGP has shown good generalization ability. However, KGP's computational overhead is significant, and potentially prevent its application to large datasets.

Avenues for future work include changing the fitness function to one which tries to maximize the margin, and studying the relationship between GP parameters and SVM parameters. In addition, increasing the number of closure properties and using GP to evolve basic kernel functions will increase the applicability of KGP.

9. REFERENCES

[1] G. H. Baker, L. Bottou, and J. Weston. Breaking SVM complexity with cross-training. In L. Saul, Y. Weiss, and L. Bottou, editors, *Proceedings of Advances in Neural Information Processing Systems*, pages 81 – 88, 2005.

[2] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for

support vector machines. *Machine Learning*, 46:131 – 159, 2002.

[4] K. Crammer, J. Keshet, and Y. Singer. Kernel design using boosting. In S. Becker, S. Thrun, and K. Obermayer, editors, *Proceedings of Advanced in Neural Information Processing Systems*, pages 537 – 544, Vancouver, Canada, 2002. MIT Press.

[5] N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. Kandola. On kernel-target alignment. In T. Diettrich, S. Becker, and Z. Ghahramani, editors, *Proceedings of Advanced in Neural Information Processing Systems*, pages 367 – 373, Vancouver, Canada, 2001. MIT Press.

[6] B. F. de Souza, A. C. de Carvalho, R. Calvo, and R. P. Ishii. Multiclass SVM model selection using particle swarm optimization. In *Proceedings of the Sixth International Conference on Hybrid Intelligent Systems*, 2006.

[7] C. B. D.J. Newman, S. Hettich and C. Merz. UCI repository of machine learning databases, 1998.

[8] F. Friedrichs and C. Igel. Evolutionary tuning of multiple SVM parameters. *Neurocomputing*, 64:107 – 117, 2005.

[9] C. Gagne, M. Schoenauer, M. Sebag, and M. Tomassini. Genetic programming for kernel based learning with co-evolving subsets selection. In *Proceedings of Parallel Problem Solving in Nature*, 2006.

[10] T. Howley and M. G. Madden. The genetic kernel support vector machine: Description and evaluation. *Artificial Intelligence Review*, 24(3 – 4):379 – 395, 2005.

[11] C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, 2003.

[12] C.-L. Huang, M.-C. Chen, and C.-J. Wang. Credit scoring with a data mining approach based on support vector machines. *Expert Systems with Applications*, 2006.

[13] C.-L. Huang and C.-J. Wang. A GA-based feature selection and parameter optimization for support vector machines. *Expert Systems with Applications*, 31:231 – 240, 2006.

[14] S. S. Keerthi. Efficient turing of SVM hyperparameters using radius/margin bound and iterative algorithms. *IEEE Transactions on Neural Networks*, 13(5):1225 – 1229, 2002.

[15] S. Luke. ECJ 13: A java-based computation and genetic programming research system. `http://cs.gmu.edu/~eclab/projects/ecj/`, 2005.

[16] S. Luke and L. Panait. Lexicographic parsimony pressure. In W. B. Langdon, editor, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 829 – 836, 2002.

[17] I. Mierswa. Evolutionary learning with kernels: A generic solution for large margin problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1553–1560, 2006.

[18] D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199 – 230, 11995.

[19] C. S. Ong, A. J. Smola, and R. C. Williamson. Hyperkernels. In S. Becker, S. Thrun, and K. Obermayer, editors, *Proceedings of Advanced in Neural Information Processing Systems*, pages 478 – 485, Vancouver, Canada, 2002. MIT Press.

[20] T. Phienthrakul and B. Kijsirikul. Evolutionary strategies for multi-scale radial basis function kernels in support vector machines. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 905 – 911, Washington, DC, June 2005. ACM Press.

[21] T. P. Runarsson and S. Sigurdsson. Asynchronous parallel evolutionary model selection for support vector machines. *Neural Information Processing – Letters and Reviews*, 3(3):59 – 67, June 2004.

[22] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[23] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207 – 1245, 2000.

[24] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University, 2004.

[25] S. Sonnenburg, R. Gunnar, and S. Christin. A general and efficient multiple kernel learning algorithm. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Proceedings of Advances in Neural Information Processing Systems*, pages 1273–1280, Vancouver, Canada, 2005. MIT Press.

[26] I. Steinwart. Sparseness of support vector machines – some asymptotically sharp bounds. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems*, volume 16, 2004.