The Genetic Evolution of Kernels for Support Vector Machine Classifiers

Tom Howley & Michael G. Madden

Department of Information Technology,
National University of Ireland, Galway,
thowley@vega.it.nuigalway.ie,
michael.madden@nuigalway.ie

Abstract. The Support Vector Machine (SVM) has emerged in recent years as a popular approach to the classification of data. One problem that faces the user of an SVM is how to choose a kernel and the specific parameters for that kernel. Applications of an SVM therefore require a search for the optimum settings for a particular problem. This paper proposes a classification technique, which we call the Genetic Kernel SVM (GK SVM), that uses Genetic Programming to evolve a kernel for a SVM classifier. Results of initial experiments with the proposed technique are presented. These results are compared with those of a standard SVM classifier using the Polynomial or RBF kernel with various parameter settings.

1 Introduction

The SVM is a powerful machine learning tool that is capable of representing non-linear relationships and producing models that generalise well to unseen data. SVMs initially came into prominence in the area of hand-written character recognition [1] and are now being rapidly applied to many other areas, e.g. text categorisation [2, 3] and computer vision [4]. An advantage that SVMs have over the widely-used Artificial Neural Network (ANN) is that they typically don't possess the same potential for instability as ANNs do with the effects of different random starting weights [5].

Despite this, using an SVM requires a certain amount of model selection. According to Cristianini *et al.* [6], "One of the most important design choices for SVMs is the kernel-parameter, which implicitly defines the structure of the high dimensional feature space where a maximal margin hyperplane will be found. Too rich a feature space would cause the system to overfit the data, and conversely the system might not be capable of separating the data if the kernels are too poor." However, before this stage is reached in the use of SVMs, the actual kernel must be chosen and, as the experimental results of this paper show, different kernels may exhibit vastly different performance. This paper describes a technique which attempts to alleviate this selection problem by using genetic programming (GP) to evolve a suitable kernel for a particular problem domain. We call our technique the Genetic Kernel SVM (GK SVM).

Section 2 outlines the theory behind SVM classifiers with a particular emphasis on kernel functions. Section 3 gives a very brief overview of genetic programming. Section 4 describes the proposed technique for evolution of SVM kernels. Experimental results are presented in Section 5. Some related research is described in Section 6. Finally, Section 7 presents the conclusions.

2 Support Vector Machine Classification

The problem of classification can be represented as follows. Given a set of input-output pairs $Z = \{(\boldsymbol{x_1}, y_1), (\boldsymbol{x_2}, y_2), \ldots, (\boldsymbol{x_\ell}, y_\ell)\}$, construct a classifier function f that maps the input vectors $\boldsymbol{x} \in X$ onto labels $y \in Y$. In binary classification the set of labels is simply $Y = \{-1, 1\}$. The goal is to find a classifier $f \in F$ which will correctly classify new examples $(\boldsymbol{x}, y)$, i.e. $f(\boldsymbol{x}) = y$ for examples $(\boldsymbol{x}, y)$, which were generated under the same probability distribution as the data [7]. Binary classification is frequently performed by finding a hyperplane that separates the data, e.g. Linear Discriminant Analysis (LDA) [8]. There are two main issues with using a separating hyperplane:

1. The problem of learning this hyperplane is an ill-posed one because several different solutions (hyperplanes) may exist, some of which may not generalise well to the unseen examples.
2. The data might not be linearly separable.

SVMs tackle the first problem by finding the hyperplane that realises the maximum margin of separation between the classes [9]. A representation of the hyperplane solution used to classify a new sample x_i is:

$$f(x) = \langle w \cdot x_i \rangle + b \tag{1}$$

where $\langle w \cdot x_i \rangle$ is the dot-product of the weight vector w and the input sample, and b is a bias value. The value of each element of w can be viewed as a measure of the relative importance of each of the sample attributes for the classification of a sample. It has been shown that the optimal hyperplane can be uniquely constructed by solving the following constrained quadratic optimisation problem [10]:

$$\text{Minimise } \langle w \cdot w \rangle + C \textstyle\sum_{i=1}^{\ell} \xi_i \tag{2a}$$

$$\text{subject to } \begin{cases} y_i(\langle w \cdot w \rangle + b) \geq 1 - \xi_i, i = 1, \ldots, \ell \\ \xi_i \geq 0, i = 1, \ldots, \ell \end{cases} \tag{2b}$$

This optimisation problem minimises the norm of the vector w which increases the *flatness* (or reduces the complexity) of the resulting model and thereby improves its generalisation ability. With *hard-margin* optimisation the goal is simply to find the minimum $\langle w \cdot w \rangle$ such that the hyperplane $f(x)$ successfully separates all ℓ samples of the training data. The slack variables ξ_i are introduced to allow for finding a hyperplane that misclassifies some of the samples (*soft-margin* optimisation) as many datasets are not linearly separable. The complexity constant $C > 0$ determines the trade-off between the flatness and the amount by which misclassified samples are tolerated. A higher value of C means that more importance is attached to minimising the slack variables than to minimising $\langle w \cdot w \rangle$. Rather than solving this problem in its primal form of (2a) and (2b) it can be more easily solved in its dual formulation [9]:

$$\text{Maximise} \quad W(\alpha) = \textstyle\sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j \langle x_i \cdot x_j \rangle \tag{3a}$$

$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{\ell} \alpha_i y_i = 0 \tag{3b}$$

Instead of finding w and b the goal now is find the vector α and bias value b, where each α_i represents the relative importance of a training sample i in the classification of a new sample. To classify a new sample, the quantity $f(x)$ is calculated as:

$$f(x) = \sum_i \alpha_i y_i \langle x \cdot x_i \rangle + b \tag{4}$$

where b is chosen so that $y_i f(x) = 1$ for any i with $C > \alpha_i > 0$. Then, a new sample x_s is classed as negative if $f(x_s)$ is less than zero and positive if $f(x_s)$ is greater than or equal to zero. Samples x_i for which the corresponding α_i are non-zero are known as *support vectors* since they lie closest to the separating hyperplane. Samples that are not support vectors have no influence on the decision function. In (3b) C places an upper bound (known as the box constraint) on the value that each α_i can take. This limits the influence of outliers, which would otherwise have large α_i values [9].

Training an SVM entails solving the quadratic programming problem of (3a) and (3b) and there are many standard techniques that could be applied to SVMs including the Newton method, conjugate gradient and primal-dual interior-point methods [9]. For the experiments reported here the SVM implementation uses the Sequential Minimisation Optimisation (SMO) algorithm of Platt [11].

2.1 Kernel Functions

One key aspect of the SVM model is that the data enters the above expressions (3a and 4) only in the form of the dot product of pairs. This leads to the resolution of the second problem mentioned above, namely that of non-linearly separable data. The basic idea with SVMs is to map the training data into a higher dimensional feature space via some mapping $\phi(x)$ and construct a separating hyperplane with maximum margin there. This yields a non-linear decision boundary in the original input space. By use of a kernel function, $K(x, z) = \langle \phi(x) \cdot \phi(z) \rangle$, it is possible to compute the separating hyperplane without explicitly carrying out the map into feature space [12]. Typical choice for kernels are:

- Polynomial Kernel: $K(x, z) = (1 + \langle x \cdot z \rangle)^d$
- RBF Kernel: $K(x, z) = exp(\frac{-||x-z||^2}{2\sigma^2})$
- Sigmoid Kernel: $K(x, z) = tanh(\langle x, z \rangle - \theta)$

Each kernel corresponds to some feature space and because no explicit mapping to this feature space occurs, optimal linear separators can be found efficiently in feature spaces with millions of dimensions [13]. An alternative to using one of the predefined kernels is to derive a custom kernel that may be suited to a particular problem, e.g. the string kernel used for text classification by Lodhi *et al.* [14]. To ensure that a kernel function actually corresponds to some feature space it must be symmetric, i.e. $K(x, z) = \langle \phi(x) \cdot \phi(z) \rangle = \langle \phi(z) \cdot \phi(x) \rangle = K(z, x)$. Typically, kernels are also required to satisfy Mercer's theorem, which states that the matrix $K = (K(x_i, x_j))_{i,j=1}^n$

must be positive semi-definite, i.e. it has no non-negative eigenvalues [9]. This condition ensures that the solution of (3a) and (3b) produces a global optimum. However, good results have been achieved with non-Mercer kernels, and convergence is expected when the SMO algorithm is used, despite no guarantee of optimality when non-Mercer kernels are used [15].

3 Genetic Programming

A GP is an application of the genetic algorithm (GA) approach to derive mathematical equations, logical rules or program functions automatically [16]. Rather than representing the solution to a problem as a string of parameters, as in a conventional GA, a GP usually uses a tree structure, the leaves of which represent input variables or numerical constants. Their values are passed to *nodes*, at the junctions of branches in the tree, which perform some numerical or program operation before passing on the result further towards the root of the tree. The GP typically starts off with a random population of individuals, each encoding a function or expression. This population is evolved by selecting better individuals for recombination and using their offspring to create a new population (generation). Mutation is employed to encourage discovery of new individuals. This process is continued until some stopping criteria is met, e.g. homogeneity of the population.

4 Genetic Evolution of Kernels

The approach presented here combines the two techniques of SVMs and GP, using the GP to evolve a kernel for a SVM. The goal is to eliminate the need for testing various kernels and their parameter settings. With this approach it might also be possible to discover new kernels that are particularly useful for the type of data under analysis. The main steps in this procedure are:

1. Create a random population of kernel functions, represented as trees — we call these *kernel trees*
2. Evaluate the fitness of each individual by building an SVM from the kernel tree and test it on the training data
3. Select the fitter kernel trees as parents for recombination
4. Perform random mutation on the newly created offspring
5. Replace the old population with the offspring
6. Repeat Steps 2 to 5 until the population has converged
7. Build final SVM using the fittest kernel tree found

The *Grow* method [17] is used to initialise the population of trees, each tree being grown until no more leaves could be expanded (i.e. all leaves are terminals) or until a preset initial maximum depth (2 for the experiments reported here) is reached. Rank-based selection is employed with a crossover probability of 0.9. Mutation with probability 0.2 is carried out on offspring by randomly replacing a sub-tree with a newly generated (via Grow method) tree. To prevent the proliferation of massive tree structures,

pruning is carried out on trees after crossover and mutation, maintaining a maximum depth of 12. In the experiments reported here, five populations are evolved in parallel and the best individual over all populations is selected after all populations have converged. This reduces the likelihood of the procedure converging on a poor solution.

4.1 Terminal & Function Set

In the construction of kernel trees the approach adopted was to use the entire sample vector as input. An example of a kernel tree is shown in Figure 1 in Section 5. Since a kernel function only operates on two samples the resulting terminal set comprises only two vector elements: x and z. The evaluation of a kernel on a pair of samples is:

$$K(x, z) = \langle treeEval(x, z) \cdot treeEval(z, x) \rangle \tag{5}$$

The kernel is first evaluated on the two samples x and z. These samples are swapped and the kernel is evaluated again. The dot-product of these two evaluations is returned as the kernel output. This current approach produces symmetric kernels, but does not guarantee that they obey Mercer's theorem. Ensuring that such a condition is met would add considerable time to kernel fitness evaluation and, as stated earlier, using a non-Mercer kernel does not preclude finding a good solution.

The use of vector inputs requires corresponding vector operators to be used as functions in the kernel tree. The design employed uses two versions of the $+, -$ and $\times$ mathematical functions: *scalar* and *vector*. Scalar functions return a single scalar value regardless of the operand's type, e.g. $x *^{scal} z$ calculate the dot-product of the two vectors. For the two other operators ($+$ and $-$) the operation is performed on each pair of elements and the *magnitude* of the resulting vector is returned as the output. Vector functions return a vector provided at least one of the inputs is a vector. For the vector versions of addition and subtraction (e.g. $x +^{vect} z$) the operation is performed on each pair of elements as with the scalar function, but in this case the resulting vector is returned as the output. No multiplication operator that returns a vector is used. If two inputs to a vector function are scalar (as could happen in the random generation of a kernel tree) then it behaves as the scalar operator. If only one input is scalar then that input is treated as a vector of the same length as the other vector operand with each element set to the same original scalar value.

4.2 Fitness Function

Another key element to this approach (and to any evolutionary approach) is the choice of fitness function. An obvious choice for the fitness estimate is the classification error on the training set, but there is a danger that this estimate might produce SVM kernel tree models that are overfitted to the training data. One alternative is to base the fitness on a cross-validation test (e.g. leave-one-out cross-validation) in order to give a better estimation of a kernel tree's ability to produce a model that generalises well to unseen data. However, this would obviously increase computational effort greatly. Therefore, our solution (after experimenting with a number of alternatives) is to use a tiebreaker to limit overfitting. The fitness function used is:

$$fitness(tree) = Error, \text{ with tiebreaker: } fitness = \sum \alpha_i * R^2 \quad (6)$$

This firstly differentiates between kernel trees based on their training error. For kernel trees of equal training error, a second evaluation is used as a tiebreaker. This is based on the sum of the support vector values, $\sum \alpha_i$ ($\alpha_i = 0$ for non-support vectors). The rationale behind this fitness estimate is based on the following definition of the geometric margin of a hyperplane, γ [9]:

$$\gamma = (\sum_{i \in sv} \alpha_i)^{-\frac{1}{2}} \quad (7)$$

Therefore, the smaller the sum of the α_i's, the bigger the margin and the smaller the chance of overfitting to the training data. The fitness function also incorporates a penalty corresponding to R, the radius of the smallest hypersphere that encloses the training data in feature space. R is computed as [9]:

$$R = \max_{1 \leq i \leq \ell} (K(x_i, x_i)) \quad (8)$$

where ℓ is the number of samples in the training dataset. This fitness function therefore favours a kernel tree that produces a SVM with a large margin relative to the radius of its feature space.

5 Experimental Results

Table 1 shows the performance of the GK SVM classifier compared with the two most commonly used SVM kernels, Polynomial and RBF, on a number of datasets. (These are the only datasets with which the GK SVM has been evaluated to date.) The first four datasets contain the Raman spectra for 24 sample mixtures, made up of different combinations of the following four solvents: Acetone, Cyclohexanol, Acetonitrile and Toluene; see Hennessy *et al.* [18] for a description of the dataset. The classification task considered here is to identify the presence or absence of one of these solvents in a mixture. For each solvent, the dataset was divided into a training set of 14 samples and a validation set of 10. The validation set in each case contained 5 positive and 5 negative samples. The final two datasets, Wisconsin Breast Cancer Prognosis (WBCP) and Glass2, are readily available from the UCI machine learning database repository [19]. The results for WBCP dataset show the average classification accuracy based on a 3-fold cross validation test on the whole dataset. Experiments on the Glass2 dataset use a training set of 108 instances and a validation set of 55 instances.

For all SVM classifiers the complexity parameter, C, was set to 1. An initial population of 100 randomly generated kernel trees was used for the WBCP and Glass2 datasets and a population of 30 was used for finding a model for the Raman spectra datasets. The behaviour of the GP search differed for each dataset. For the spectral datasets, the search quickly converged to the simple solution after an average of only 5 generations, whereas the WBCP and Glass2 datasets required an average of 17 and 31 generations, respectively. (As stated earlier, five populations are evolved in parallel and the best individual chosen.)

Classifier	Dataset					
Polynomial Kernel - Degree d	**Acetone**	**Cyclohexanol**	**Acetonitrile**	**Toluene**	**WBCP**	**Glass2**
1	100.00	100.00	100.00	90.00	78.00	62.00
2	90.00	90.00	100.00	90.00	77.00	70.91
3	50.00	90.00	100.00	60.00	86.00	78.18
4	50.00	50.00	50.00	50.00	87.00	74.55
5	50.00	50.00	50.00	50.00	84.00	76.36
RBF Kernel - σ						
0.0001	50.00	50.00	50.00	50.00	78.00	58.18
0.001	50.00	90.00	50.00	50.00	78.00	58.18
0.01	60.00	80.00	50.00	60.00	78.00	59.64
0.1	50.00	50.00	50.00	50.00	78.00	63.64
1	50.00	50.00	50.00	50.00	81.00	70.91
10	50.00	50.00	50.00	50.00	94.44	83.64
100	50.00	50.00	50.00	50.00	94.44	81.82
GK SVM	100.00	100.00	100.00	80.00	93.43	87.27

Table 1. Comparison of GK SVM with Polynomial and RBF Kernel SVM

The results clearly demonstrate both the large variation in accuracy between the Polynomial and RBF kernels as well as the variation between the performance of models using the same kernel but with different parameter settings: degree d for the Polynomial kernel and σ for the RBF kernel. The RBF kernel performs poorly on the spectral datasets but then outperforms the Polynomial kernel on the Wisconsin Breast Cancer Prognosis and Glass2 datasets. For the first three spectral datasets, the GK SVM achieves 100% accuracy, each time finding the same simple linear kernel as the best kernel tree:

$$K(x, z) = \langle x \cdot z \rangle \tag{9}$$

For the Toluene dataset, the GK SVM manages to find a kernel of higher fitness (according to the fitness function detailed in Section 4.2) than the linear kernel, but which happens to perform worse on the test dataset. One drawback with the use of these spectral datasets is that the small number of samples is not very suitable for a complex search procedure such as used in GK SVM. A small training dataset increases the danger of an evolutionary technique, such as GP, finding a model that fits the training set well but performs poorly on the test data.

On the Wisconsin Breast Cancer Prognosis dataset, the GK SVM performs better than the best Polynomial kernel (d = 4). The best kernel tree found during the final fold of the 3-fold cross-validation test is shown in Figure 1. This tree represents the following kernel function:

$$K(x, z) = \langle (x -^{scal} (x -^{scal} z)) \cdot (z -^{scal} (z -^{scal} x)) \rangle \tag{10}$$

The performance of the GK SVM on this dataset demonstrates its potential to find new non-linear kernels for the classification of data. The GK SVM does, however, perform marginally worse than the RBF kernel on this dataset. This may be due to the

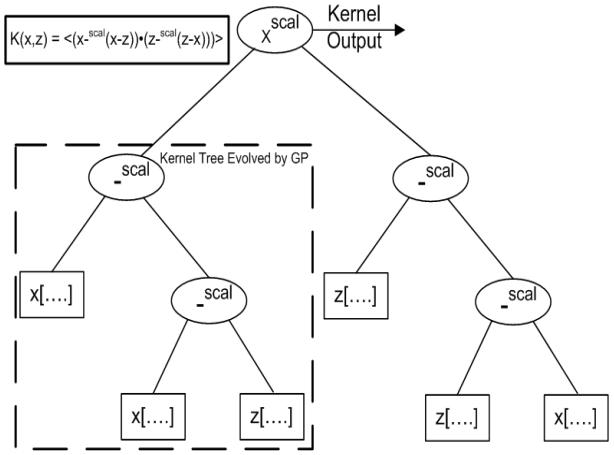

Fig. 1. Example of a Kernel found on the Wisconsin Breast Cancer Dataset

fact that the kernel trees are constructed using only 3 basic mathematical operators and therefore cannot find a solution to compete with the exponential function of the RBF kernel. Despite this apparent disadvantage, the GK SVM clearly outperforms either kernel on the Glass2 dataset.

Overall, these results show the ability of the GK SVM to automatically find kernel functions that perform competitively in comparison with the widely used Polynomial and RBF kernels, but without requiring a manual parameter search to achieve optimum performance.

6 Related Research

6.1 SVM Model Selection

Research on the tuning of kernel parameters or model selection is of particular relevance to the work presented here, which is attempting to automate kernel selection. A common approach is to use a grid-search of the parameters, e.g. complexity parameter C and width of RBF kernel, σ [20]. In this case, pairs of (C,σ) are tried and the one with best cross-validation accuracy is picked. A similar algorithm for the selection of SVM parameters is presented in Staelin [21]. That algorithm starts with a very coarse grid covering the whole search space and iteratively refines both grid resolution and search boundaries, keeping the number of samples at each iteration roughly constant. It is based on a search method from the design of experiments (DOE) field. Those techniques still require selection of a suitable kernel in addition to knowledge of a suitable starting range for the kernel parameters being optimised. The same can be said for the model selection technique proposed in Cristianini *et al.* [6], in which an on-line gradient ascent method is used to find the optimal σ for an RBF kernel.

6.2 Application of Evolutionary techniques with SVM classifiers

Some research has been carried out on the use of evolutionary approaches in tandem with SVMs. Fröhlich *et al.* [22] use GAs for feature selection and train SVMs on the reduced data. The novelty of this approach is in its use of a fitness function based on the calculation of the theoretical bounds on the generalisation error of the SVM. This approach was found to achieve better results than when a fitness function based on cross-validation error was used. A RBF kernel was used in all reported experiments.

An example of GPs and SVMs is found in Eads *et al.* [23], which reports on the use of SVMs for identification of lightning types based on time series data. However, in this case the GP was used to extract a set of features for each time series sample in the dataset. This derived dataset was then used as the training data for building the SVM which mapped each feature set or vector onto a lightning category. A GA was then used to evolve a chromosome of multiple GP trees (each tree was used to generate one element of the feature vector) and the fitness of a single chromosome was based on the cross validation error of an SVM using the set of features it encoded. With this approach the SVM kernel (along with σ) still had to be selected, in this case the RBF kernel was used.

7 Conclusions

This paper has proposed a novel approach to tackle the problem of kernel selection for SVM classifiers. The proposed GK SVM uses a GP to evolve a suitable kernel for a particular problem. The initial experimental results show that the GK SVM is capable of matching or beating the best performance of the standard SVM kernels on the majority of the datasets tested. These experiments also demonstrate the potential for this technique to discover new kernels for a particular problem domain. Future work will involve testing the GK SVM on more datasets and comparing its performance with other SVM kernels, e.g. sigmoid. The effect of restricting the GP search to Mercer kernels will be investigated. In order to help the GK SVM find better solutions, further experimentation is also required with increasing the range of functions available for construction of kernel trees, e.g. to include the exponential or $tanh$ function.

Acknowledgements

This research has been funded by Enterprise Ireland's Basic Research Grant Programme. The authors are grateful to Dr. Alan Ryder and Jennifer Conroy for providing the spectral datasets.

References

1. Boser, B., Guyon, I., Vapnik, V.: A training algorithm for optimal margin classifiers. In Haussler, D., ed.: Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, ACM Press (1992) 144–152
2. Hearst, M.: Using SVMs for text categorisation. IEEE Intelligent Systems **13** (1998) 18–28

3. Joachims, T.: Text categorisation with support vector machines. In: Proceedings of European Conference on Machine Learning (ECML). (1998)
4. Osuna, E., Freund, R., Girosi, F.: Training support vector machines: An application to face detection. In: Proceedings of Computer Vision and Pattern Recognition. (1997) 130–136
5. Bleckmann, A., Meiler, J.: Epothilones: Quantitative Structure Activity Relations Studied by Support Vector Machines and Artificial Neural Networks. QSAR & Combinatorial Science **22** (2003) 722–728
6. Cristianini, N., Campbell, C., Shawe-Taylor, J.: Dynamically Adapting Kernels in Support Vector Machines. Technical Report NC2-TR-1998-017, NeuroCOLT2 (1998)
7. Scholkopf, B.: Support Vector Machines - a practical consequence of learning theory. IEEE Intelligent Systems **13** (1998) 18–28
8. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer (2001)
9. N.Cristianini, Shawe-Taylor, J.: An Introduction to Support Vector Machines. Cambridge University Press (2000)
10. Boser, B., Guyon, I., Vapnik, V.: A training algorithm for optimal margin classifiers. In Haussler, D., ed.: Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, ACM Press (1992) 144–152
11. Platt, J.: Using Analytic QP and Sparseness to Speed Training of Support Vector Machines. In: Proceedings of Neural Information Processing Systems (NIPS). (1999) 557–563
12. Scholkopf, B.: Statistical Learning and Kernel Methods. Technical Report MSR-TR-2000-23, Microsoft Research, Microsoft Corporation (2000)
13. Russell, S., Norvig, P.: Artificial Intelligence A Modern Approach. Prentice-Hall (2003)
14. Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., Watkins, C.: Text classification using string kernels. Journal of Machine Learning Research **2** (2002) 419–444
15. Bahlmann, C., Haasdonk, B., Burkhardt, H.: On-line Handwriting Recognition with Support Vector Machines - A Kernel Approach. In: Proceedings of the 8th International Workshop on Frontiers in Handwriting Recognition. (2002) 49–54
16. Koza, J.: Genetic Programming. MIT Press (1992)
17. Banzhaf, W., Nordin, P., Keller, R., Francone, F.: Genetic Programming - An Introduction. Morgan Kaufmann (1998)
18. Hennessy, K., Madden, M., Conroy, J., Ryder, A.: An Improved Genetic Programming Technique for the Classification of Raman Spectra. In: Proceedings of the 24th SGAI Intl. Conference on Innovative Techniques and Applications of Artificial Intelligence (to appear). (2004)
19. Blake, C., Merz, C.: UCI Repository of machine learning databases, http://www.ics.uci.edu/∼mlearn/MLRepository.html. University of California, Irvine, Dept. of Information and Computer Sciences (1998)
20. Hsu, C., Chang, C., Lin, C.: A Practical Guide to Support Vector Classification. (2003)
21. Staelin, C.: Parameter selection for support vector machines. Technical Report HPL-2002-354, HP Laboratories, Israel (2002)
22. Frolich, H., Chapelle, O., Scholkopf, B.: Feature Selection for Support Vector Machines by Means of Genetic Algorithms. In: Proceedings of the International IEEE Conference on Tools with AI. (2003) 142–148
23. Eads, D., Hill, D., Davis, S., Perkins, S., Ma, J., Porter, R., Theiler, J.: Genetic Algorithms and Support Vector Machines for Times Series Classification. In: Proceedings of the Fifth Conference on the Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation. Symposium on Optical Science and Technology of the 2002 SPIE Annual Meeting. (2002) 74–85