

An Evolutionary Approach to Automatic Kernel Construction

Tom Howley and Michael G. Madden

National University of Ireland, Galway

thowley@vega.it.nuigalway.ie, michael.madden@nuigalway.ie

Abstract. Kernel-based learning presents a unified approach to machine learning problems such as classification and regression. The selection of a kernel and associated parameters is a critical step in the application of any kernel-based method to a problem. This paper presents a data-driven evolutionary approach for constructing kernels, named KTree. An application of KTree to the Support Vector Machine (SVM) classifier is described. Experiments on a synthetic dataset are used to determine the best evolutionary strategy, e.g. what fitness function to use for kernel evaluation. The performance of an SVM based on KTree is compared with that of standard kernel SVMs on a synthetic dataset and on a number of real-world datasets. KTree is shown to outperform or match the best performance of all the standard kernels tested.

1 Introduction

A major advance in recent research into pattern analysis has been the emergence of an approach known as kernel-based learning. This unified approach to problems, such as classification, regression and clustering, is based on a kernel that defines how two objects of a dataset are related. Kernel-based learning first appeared in the form of support vector machines, a powerful classification algorithm that is capable of representing non-linear relationships (via kernels) and producing models that generalise well to unseen data. A key decision in the use of any kernel-based method is the choice of kernel. In the case of SVMs, the performance exhibited by different kernels may differ considerably. Generally, kernel method practitioners will pick from a set of standard kernels, the Radial Basis Function (RBF) and Polynomial kernel being two widely used examples. An alternative to using one of these pre-defined kernels is to construct a custom kernel especially for a particular problem domain, e.g. the string kernel used for text classification [1]. This approach can yield good results, but obviously depends on the availability of expert knowledge of a particular domain.

This paper presents an approach, named KTree, that uses the evolutionary method of Genetic Programming (GP) to find a kernel for a particular data domain. KTree is a modified and extended version of the Genetic Kernel SVM (GKSVM) developed by the authors [2]: it uses a more sophisticated kernel representation that can represent standard kernels, such as RBF; a Mercer filter is used to improve performance; it uses a different fitness function (based on cross-validation), which results have shown to be superior. This study also includes a more extensive evaluation, using both a synthetic dataset and wider range of real-world datasets. KTree allows for the generation of non-standard kernels; the objective is to provide for the automatic discovery of kernels that

achieve good classification accuracy when tested on unknown data. The major goal of this research is to determine the best strategy in the use of GP to evolve kernels; key issues include choice of fitness function and the filtering of non-Mercer kernels.

Kernel methods are described in Section 2, with particular emphasis on kernel functions. Section 3 describes KTree. Experimental results and analyses are presented in Section 4. Section 5 evaluates research related to this work and Section 6 presents the main conclusions.

2 Kernel Methods and Classification

In kernel methods, the kernel function is used to recode the data into a new feature space that reveals regularities in the data that were not detectable in the original representation. This allows the use of algorithms based on linear functions in the feature space; such linear methods are both well understood and computationally efficient. With kernel functions, no explicit mapping of the data to the new feature space is carried out – this is known as the “kernel trick”. It enables the use of feature spaces whose dimensionality is more than polynomial in the original set of features, even though the computational cost remains polynomial. This unified kernel approach can be applied to a number of machine learning problems, such as supervised classification and regression, semi-supervised learning and unsupervised methods, such as clustering. The classic example of this kernel approach is found in the SVM classifier.

2.1 Kernel Functions

One key aspect of the SVM model is that the data enters both the optimisation problem and the decision function only in the form of the dot product of pairs. This enables SVMs to handle non-linear data. The dot product is replaced by a kernel function, $K(x, z) = \langle \phi(x), \phi(z) \rangle$, that computes the dot product of two samples in a feature space, where $\phi(x)$ represents the mapping to this feature space. The SVM finds the maximum margin separating hyperplane in the feature space defined by this kernel, thus yielding a non-linear decision boundary in the original input space. With the use of kernel functions, it is possible to compute the separating hyperplane in a high dimensional feature space without explicitly carrying out the mapping, ϕ , into that feature space [3]. Typical choices for kernels are the Linear, Polynomial, RBF and Sigmoid kernels. Note that using a Linear kernel is equivalent to working in the original input space. Apart from this kernel, all of the above kernels require the setting of one or more parameters, such as σ , the kernel width of the RBF kernel. One alternative to using these standard kernels is to employ a kernel that has been customised for a particular application domain, e.g. the string kernel of Lodhi *et al.* [1].

Whether building complex kernels from simpler kernels, or designing custom kernels, there are conditions that the kernel must satisfy before it can be said to correspond to some feature space. Firstly, the kernel must be symmetric, i.e. $K(x, z) = \langle \phi(x), \phi(z) \rangle = \langle \phi(z), \phi(x) \rangle = K(z, x)$. 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 negative eigenvalues [4]. In SVM classification, this condition ensures that the solution of the optimisation problem 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 [5]. Furthermore, despite its wide use, the Sigmoid kernel matrix is not positive semi-definite for certain values of the parameters γ and θ [6].

3 KTree and SVM Classification

A critical stage in the use of kernel-based algorithms is kernel selection, as this can be shown to correspond to the encoding of prior knowledge about the data [7]. SVM users typically employ one of the standard kernels listed in Section 2.1. Kernels can also be constructed by using simpler kernels as building blocks, e.g. the kernel, $K(x, z) = K_1(x, z) + K_2(x, z)$ or by using the custom kernel approach. Ideally, a kernel is selected based on prior knowledge of the problem domain, but it is not always possible to make the right of choice of kernel *a priori*.

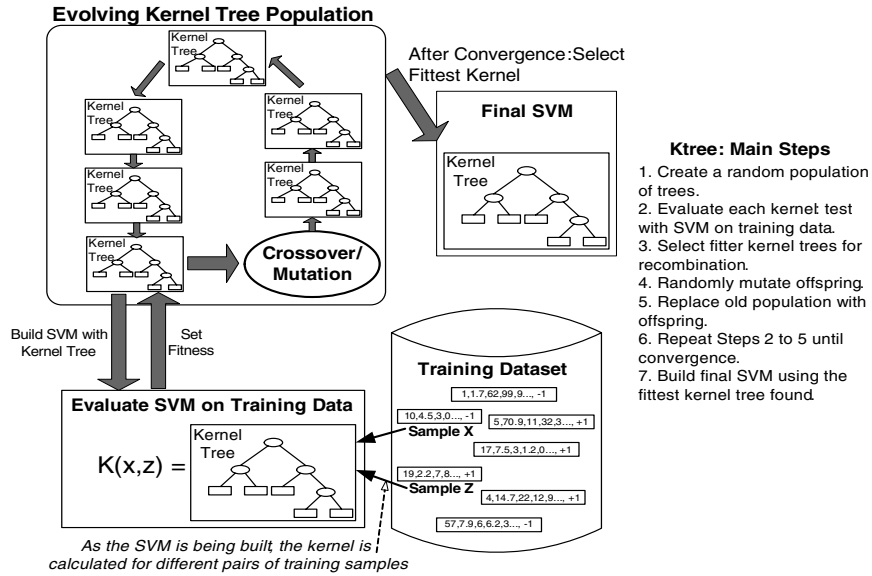


Fig. 1. Application of KTree to the SVM

The approach presented here uses an evolutionary technique to discover a suitable kernel for a particular problem. In this case, KTree is used to evolve kernels specifically for SVM classifiers, but this approach can be used with other kernelised pattern analysis algorithms. The aim of KTree is to eliminate the need for testing various kernels and parameter settings, while also allowing for the discovery of new non-standard kernels. With KTree, a tree structure, known as a *kernel tree* (see Figure 2) is used to represent a kernel function. The objective of KTree is to find a kernel tree that best represents the data. An overview of the application of KTree to the SVM is shown in Figure 1, which also includes the main steps in the building of a SVM using KTree.

3.1 Kernel Tree Representation

The kernel tree used to represent a kernel function must take two data samples as inputs and provide a scalar value as output. An example of a kernel tree is shown in Figure 2.

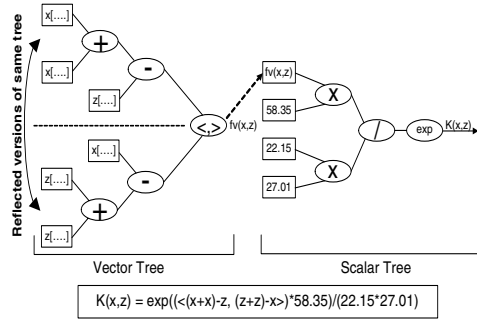


Fig. 2. Kernel Tree generated for Ionosphere Data

This particular kernel tree was generated from experiments on the Ionosphere dataset. The diagram shows that the kernel tree is split into two parts, the vector and the scalar tree. The inputs to the vector tree are the two samples, x and z , for which the kernel is being evaluated. These inputs are passed through vector operators, such as *add* or *subtract*, which in turn pass vectors onto the next node. To ensure that the output of this tree is symmetric, the entire vector tree is evaluated twice, swapping the inputs x and z for the second evaluation. The final output of the vector tree, $f_v(x, z)$, is the dot product of these two evaluations. This output becomes an input, along with randomly generated constant terminals, for the scalar tree. This design was chosen to allow for the use of complex mathematical operators, such as *exp* and *tanh*, in the scalar tree. Applying these operators directly to the vector inputs could result in overly complex and unusable kernels. A second motivation for this design is that it is also capable of representing the standard kernels, e.g. the RBF kernel and Polynomial kernel. Although symmetry is satisfied, this kernel tree design is not guaranteed to produce Mercer kernels. However, non-Mercer kernels can be filtered out (see Section 3.2).

For the initial population, each kernel tree (both vector and scalar parts) is generated by randomly creating a root node and by growing a tree from this node until either no more leaves can be expanded (i.e. all leaves are terminals) or until a preset initial maximum depth has been reached (2 for the experiments reported here). The evolutionary process shown in Figure 1 involves the application of mutation and crossover operators on selected kernel trees. For mutation, a point in either the vector or scalar tree is randomly chosen and the sub-tree at that point is replaced with a newly generated tree (vector or scalar, depending on where mutation occurred). Mutation of individual nodes (e.g. constant terminals) is not employed. Crossover between two kernel trees begins with the selection of a random point from either the vector or scalar part of the first kernel tree. The location of the crossover point on the second kernel tree is constrained so that crossover does not occur between the scalar part of one kernel tree and the vector

part of another. Rank-based selection was employed for the selection of the candidates for crossover. To prevent the proliferation of massive tree structures, pruning is carried out on kernel trees after mutation, maintaining a maximum depth of 12 (for either the vector or scalar part). A population of 500 kernel trees was used for all experiments, each being evolved over 32 generations, on average.

3.2 Fitness Function

Another key element of KTree is the choice of fitness function. Three different fitness functions were tested in experiments on a synthetic dataset (see Section 4.1). Two of the fitness functions are based on training set classification error in combination with a different tiebreaker fitness (to limit overfitting on the training set). The first tiebreaker fitness is based on kernel tree size, favouring smaller trees, in the spirit of *Ockham's Razor*. The second tiebreaker fitness is based on the sum of the support vector values, $\sum \alpha_i$ (where $\alpha_i = 0$ for non-support vectors). It favours kernels with a smaller sum and also incorporates a penalty corresponding to the radius of the smallest hypersphere, centred at the origin, that encloses the training data in feature space. The third fitness function employed is based on a 3-fold cross-validation test on the training data and also uses tree size as a tiebreaker fitness. In this case, the same kernel is used to build an SVM three times over the course of one fitness evaluation. The experimental analysis of Section 4.1 details the results of applying the above fitness functions on a synthetic dataset.

In addition to the above fitness evaluations, the use of a filter for non-Mercer kernels (referred to as the *Mercer filter*) was investigated. To estimate the Mercer condition of a kernel, the eigenvalues of the kernel matrix over the training data are calculated; if any negative eigenvalues are discovered, the kernel is marked as non-Mercer and is assigned the worst possible fitness, e.g. a cross-validation error of 100%. To reduce the computational cost when dealing with larger datasets, the kernel matrix is based on only a subset of the training data. This approach was found to be effective in the experiments (detailed in Section 4). The kernel matrix was limited to a maximum size of 250x250.

4 Experimental Results

4.1 Synthetic Dataset

To determine the best strategy for evolving kernels for use in SVM classifiers, a number of experiments were carried out on a synthetic dataset, the checkerboard dataset, shown in Figure 3(a). A checkerboard dataset (similar to that used by Mangasarian *et al.* [8]) of 10,000 samples was generated with an equal distribution of both classes. This synthetic dataset allows for the creation of a large test set that is suitable for comparing different kernel classifiers and is also useful for visually comparing kernel performance. In addition to finding a strategy that generates kernels with good classification accuracy, this research is concerned with issues such as the selection of fitness function, the effect of using non-Mercer kernels, and the contribution of genetic operators.

Table 1. Results on Checkerboard Dataset

(a) Standard Kernels			(b) KTree		
Standard Kernel	Fitness	Error	KTree	Fitness	Error
Linear ($C=1$)	43.3%	48.4%	Default	9.6%	10.74%
Poly ($C=32, d=13$)	19.6%	27.48%	Training + No. Nodes	–	14.19%
RBF ($C=16, \sigma=8$)	14.8%	11.67%	Training + $\sum \alpha_i$	–	41.26%
Sigmoid ($C=0.1, \gamma=10, \theta=1E-6$)	40.8%	48.92%	No Mercer Filter	8.00%	7.43%
			No Crossover	11.6%	7.71%
			No Mutation	11.2%	12.57%

Table 1 shows the results on the checkerboard dataset for the standard kernels and KTree. In both cases, the SVM was trained on a subset of 250 samples from the checkerboard dataset and then tested on the full dataset. For each standard kernel, a simple technique is employed for choosing parameters: an SVM with the standard kernel is tested on the training dataset over a range of settings for both kernel and SVM (C parameter). The degree parameter, d , was tested with the values: 1, 2, ..., 19, 20. C and σ were tested with the values $2^{-20}, 2^{-19}, \dots, 2^{19}, 2^{20}$, except for the Sigmoid kernel, in which case C and the two other parameters (γ and θ) were tested with the following values: $10^{-6}, 10^{-5}, \dots, 10^5, 10^6$. For each kernel type, the kernel setting of the best fitness (based on 3-fold cross-validation error) is chosen and used to build an SVM on the entire training dataset, the resulting model used for the test dataset. Table 1 shows the fitness of the final selected kernel (for both standard and KTree) along with its test error. Table 1(a) shows the RBF kernel outperforming all other standard kernels. The KTree results are based on different variations of KTree, depending on choice of fitness estimate, use of Mercer filter and crossover/mutation rates. The default KTree of Table 1(b) uses a fitness function based on 3-fold cross-validation error, employs a Mercer filter and uses both mutation and crossover. The next two KTree variations use the other two fitness estimates (based on Training error with either number of nodes or $\sum \alpha_i$ as tiebreaker) outlined in Section 3.2. The results show that the default setting achieves the best results out of the three, with KTree using training error with α -radius estimate performing very badly. Further analysis of fitness vs. test error showed the fitness based on 3-fold cross-validation to be more stable; this fitness estimate is used as the default in the remaining experiments (both synthetic and UCI datasets).

This study is also concerned with the behaviour of the genetic operators used in KTree. The traditional view of Genetic Algorithms (GAs) is that crossover is primarily responsible for improvements in fitness, and that mutation serves a secondary role of reintroducing individuals that have been lost from the population. However, an important difference with GPs (compared with GAs) is that crossover in GP swaps trees of varying sizes, shape, and position, whereas the typical GA swaps alleles at exactly the same locus [9]. Furthermore, changing a function or a terminal in a GP can have a dramatic effect on the operations of other functions and terminals not only within its own subtree, but throughout an individual. The default setting for KTree shown in Table 1 adopts the classical approach, i.e. a high crossover rate (0.8) relative to the mutation

rate (0.2). The results for KTree with two other different settings are shown in the last two rows of this table: one without crossover and the other without mutation. KTree without crossover achieved a very good test error, but the actual fitness of its best individual is worse than that produced by the default KTree. In terms of final kernel fitness, there is very little difference between KTree based on crossover alone and that based on mutation alone. This is in agreement with Luke & Spector’s conclusion that there is often no significant difference between the performance obtained by an all-crossover strategy or an all-mutation strategy. As selecting a very high mutation rate can have adverse effects on convergence and also result in a significant increase in the number of kernel evaluations required in one run, KTree used for tests on the UCI datasets uses a higher crossover rate (0.8) than mutation rate (0.2).¹

In addition to these results, the output for four different kernels, shown in Figure 3, was used to compare the performance of KTree with that of standard kernels. Two variations of the KTree are shown: the default KTree with Mercer filter and the same KTree, except without a filter for Mercer kernels. It can be seen from these figures that both kernels achieve an output that is much closer to the original checkerboard pattern than the standard kernels’ output. A comparison of the fitness versus test error of the kernels produced during the non-Mercer KTree run shows a reasonable trend, but does indicate a greater danger for finding highly fit kernels with poor test performance.

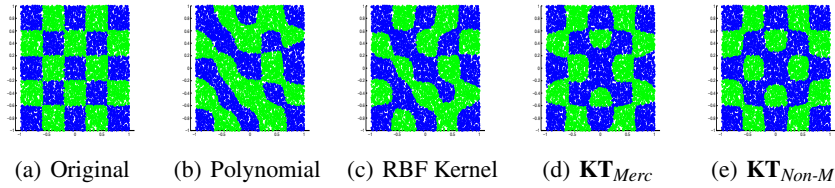


Fig. 3. Output of standard and KTree kernels

4.2 UCI Datasets

The overall conclusion from the experiments on the synthetic dataset is that KTree is capable of producing kernels that outperform the best standard kernels. A further test on nine binary classification datasets from the UCI machine learning repository [10] was carried out. The results of Table 2 show the average error from a single 10-fold cross validation test on each dataset. For each dataset, the lowest average error is highlighted in bold. A pairwise comparison between kernels over all datasets (see Table 3) was carried out using a two-tailed Wilcoxon Signed Rank test [11] at a confidence level of 5%. Table 3 shows that KTree (with Mercer filter) significantly outperforms all other kernels, with the exception of the RBF kernel (no significant difference found). These results show that KTree is capable of outperforming or matching the best results of the most widely used standard kernels. Further tests on the UCI data showed that KTree without Mercer filter yielded poor results (not shown) in comparison with the KTree that incorporates the Mercer filter.

¹ We note that this is still a reasonably high mutation rate.

Table 2. Classifier 10-fold Error Rates(%): see Table 3 for pairwise comparisons of kernels over all datasets

Dataset	Linear	Polynomial	RBF	Sigmoid	KTree
Ionosphere	13.66±3.62	8.30±4.12	5.72±2.68	9.49±5.02	5.70±2.32
Heart	17.78±7.96	17.78±7.96	18.15±8.27	18.52±5.52	17.78±8.69
Hepatitis	17.91±9.84	25.33±22.00	18.66±12.18	21.09±14.05	14.08±8.82
Sonar	21.72±9.56	16.84±10.61	14.43±8.98	18.25±9.18	11.58±7.25
Glass2	29.55±12.89	27.36±13.66	15.72±13.06	27.12±13.08	16.31±11.42
Pima	25.91±11.15	23.44±3.44	23.05±3.96	22.66±4.44	22.53±4.48
WBCP	26.85±9.20	32.09±17.99	22.62±5.44	30.93±9.33	24.2±2.72
Liver	31.54±7.71	30.63±9.31	29.18±8.30	27.15±8.41	27.73±8.93
Tic-Tac-Toe	1.67±1.12	0.10±0.33	0.21±0.65	0.00±0.00	0.42±0.72

Table 3. Performance on Independent Test Sets: pairwise comparison of kernels using Wilcoxon Test (W=Win, L=Loss, D=Draw—no sig. difference). Overall, KTree exhibits the best results.

Kernel	Lin	Poly	RBF	Sig	KTree
Lin	-	D	W	D	W
Poly	D	-	W	D	W
RBF	L	L	-	D	D
Sig	D	D	D	-	W
KTree	L	L	D	L	-
W/L/D	0/2/2	0/2/2	2/0/2	0/1/3	3/0/1

Table 4. Average kernel fitness (based on 3-fold error) on the Training Sets: this shows that KTree kernels achieve the best fitness

Dataset	Linear	Polynomial	RBF	Sigmoid	KTree
Ionosphere	12.38±0.64	8.00±0.58	4.72±0.47	11.05±3.6	4.12±0.77
Heart	16.13±0.77	16.09±0.79	15.56±0.93	15.68±0.94	13.91±0.84
Hepatitis	15.63±1.49	14.27±1.79	14.70±1.88	17.06±1.64	12.76±2.03
Sonar	22.54±2.32	14.69±2.06	12.82±1.77	19.66±2.70	8.01±1.70
Glass2	28.09±2.58	20.31±2.04	15.13±2.47	27.06±2.33	13.98±2.82
Pima	22.51±0.45	22.02±0.43	22.05±0.50	22.18±0.51	21.76±0.52
WBCP	23.63±0.16	23.23±0.53	21.44±1.24	23.12±0.66	22.17±0.99
Liver	30.08±1.61	24.80±1.24	24.83±1.17	25.60±1.16	23.41±1.17
Tic-Tac-Toe	1.67±0.12	0.94±0.21	0.51±0.25	0.59±0.28	0.38±0.19

All methods compared in Table 2 use the same basic fitness evaluation for selecting the best model for a given training set, namely 3-fold cross-validation error. Therefore, ten different kernels are selected over the course of a 10-fold cross-validation run. Table 4 shows the average fitness (or 3-fold error rate) of the ten models selected for each

kernel type. The best fitness (or lowest error) is highlighted in bold. It was found that KTree significantly outperformed (using the same Wilcoxon test as before) all of the standard kernels in terms of the average fitness of the final kernels selected. This result suggests that with a better fitness function (i.e. one that follows the actual test error more closely), KTree may be able to improve its performance on test data. On the other hand, the datasets used in these tests may be the cause of some of the problems; the presence of noise in these datasets may be adversely affecting the usefulness of this particular fitness estimate. Although the 3-fold error fitness results in good performance, further investigation is required to find a more suitable (and possibly more efficient) fitness measure. For example, it may be possible to use the training error (which is quicker to compute) as a fitness estimate when dealing with larger datasets, where there is less danger of overfitting.

5 Related Research

Some research has been carried out on the use of evolutionary approaches in tandem with SVMs. Frohlich *et al.* use GAs for feature selection and train SVMs on the reduced data [12]. 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. Other work has used evolutionary algorithms to optimise a single kernel, typically the RBF Kernel [13,14]. Similarly, Lessmann *et al.* [15] used a GA to optimise a set of parameters for five kernel types and the SVM C parameter, and is also used to determine how the result of each kernel is combined (addition or multiplication) to give the final kernel output. A separate hold-out validation set is used to assess the fitness of each kernel candidate.

6 Conclusions

This paper has described an evolutionary method for constructing the kernel of a kernel-based classifier, in this case the SVM. KTree is a data-driven approach that uses GP to evolve a suitable kernel for a particular problem. Experiments on a synthetic dataset were carried out to determine suitable settings for KTree. Using a fitness function based on an internal cross-validation test was found to yield the best result. In addition, both mutation and crossover operators were found to be useful for the discovery of better kernels. Tests on a number of standard datasets show that KTree is capable of matching or beating the best performance of any of the standard kernels tested. When compared using the fitness measure, the kernels produced with KTree clearly outperform the best standard kernels. The results also highlight the need for future work into finding a more effective fitness estimate, with which the performance of KTree could be improved. Future work will also involve testing on more datasets and using the KTree approach for regression problems and cluster analysis.

References

1. Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., Watkins, C.: Text classification using string kernels. *Journal of Machine Learning Research* **2** (2002)
2. Howley, T., Madden, M.G.: The Genetic Kernel Support Vector Machine: Description and Evaluation. *Artificial Intelligence Review* **24** (2005)
3. Scholkopf, B.: Statistical Learning and Kernel Methods. Technical Report MSR-TR-2000-23, Microsoft Research, Microsoft Corporation (2000)
4. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines. Cambridge University Press (2000)
5. Bahlmann, C., Haasdonk, B., Burkhardt, H.: On-line Handwriting Recognition with Support Vector Machines - A Kernel Approach. In: Proc. of the 8th Intl. Workshop on Frontiers in Handwriting Recognition. (2002)
6. Lin, H., Lin, C.: A Study on Sigmoid Kernels for SVM and the Training of non-PSD Kernels by SMO-type Methods. Technical report, Dept. of Computer Science and Information Engineering, National Taiwan University (2003)
7. Cristianini, N., Shawe-Taylor, J.: Kernel Methods for Pattern Analysis. Cambridge University Press (2004)
8. Mangasarian, O., Musicant, D.: Lagrangian Support Vector Machines. *Journal of Machine Learning Research* **1** (2001)
9. Luke, S., Spector, L.: A Comparison of Crossover and Mutation in Genetic Programming. In: Genetic Programming: Proc. of the 2nd Annual Conference, Morgan Kaufmann (1997)
10. Newman, D., Hettich, S., Blake, C., Merz, C.: UCI Repository of machine learning databases (1998)
11. Wilcoxon, F.: Individual comparisons by ranking methods. *Biometrics* **1** (1945) 80–83
12. Frolich, H., Chapelle, O., Scholkopf, B.: Feature Selection for SVMs by Means of Genetic Algorithms. In: Proc. of the Intl. IEEE Conference on Tools with AI. (2003) 142–148
13. Runarsson, T., Sigurdsson, S.: Asynchronous Parallel Evolutionary Model Selection for Support Vector Machines. *Neural Information Processing - Letters and Reviews* **3** (2004)
14. Friedrichs, F., Igel, C.: Evolutionary Tuning of Multiple SVM Parameters. In: Proc. of the 12th European Symposium on Artificial Neural Network. (2004) 519–524
15. Lessmann, S., Stahlbock, R., Crone, S.: Genetically constructed kernels for support vector machines. In: Proc. of General Operations Research (GOR). (2005)