Proceedings of AICS-2007: 18th Irish Conference on Artificial Intelligence and Cognitive Science, Dublin, August 2007.

The Evolution of a Kernel-Based Distance Metric for k-NN Regression

Tom Howley and Michael G. Madden

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

Abstract. k-Nearest Neighbours (k-NN) is a well understood and widely-used approach to classification and regression problems. In many cases, such applications of k-NN employ the standard Euclidean distance metric for the determination of the set of nearest neighbours to a particular test data sample. This paper investigates the use of a data-driven evolutionary approach, named KTree, for the automatic construction of a *kernel*-based distance metric as an alternative to Euclidean distance. The key idea behind this approach is that a different distance metric is generated for a particular data domain. The performance of k-NN with the standard Euclidean distance measure is compared with that of k-NN based on a kernel-based distance metric evolved by KTree. This comparison is based on experiments on both synthetic and real-world datasets.

1 Introduction

k-Nearest Neighbours (k-NN) is a well understood technique that is widely used in many classification and regression problems [1]. In many applications of k-NN, the Euclidean distance is used to determine the k nearest neighbours to a particular test sample, the resulting prediction depending directly on the particular set of neighbours chosen. As noted by Yu *et al.* [2], the conventional k-NN can perform well with non-linear problems, but loses its power with some complicated problems, especially when the sample distribution is arbitrary. However, if an appropriate kernel is chosen to reshape the distribution of samples, a *kernelised* k-NN algorithm may improve its performance. This is an example a kernel-based learning method, in which a kernel function is used to transform the original data into a new feature space. The choice of kernel and associated kernel parameters is a key step in the application of any kernel method, such as kernelised k-NN, to a problem. Previous research carried out by the authors demonstrated that an evolutionary approach, named KTree, was effective in the automatic construction of kernels in Support Vector Machine (SVM) classification [3]. This paper investigates the use of the KTree approach to evolve a kernel-based distance metric for k-NN regression.

The paper begins in Section 2 with an overview of kernel methods, kernel functions and the kernelised k-NN algorithm. Section 3 then describes the KTree algorithm. 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, Kernel Functions & k-NN

In kernel methods for classification or regression, the kernel function is used to recode the data into a new feature space that may reveal regularities in the data that were not detectable in the original representation. This allows the use of algorithms based on linear functions in the new 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*. A kernel function, $K(x, z)$, calculates the dot-product of two data samples, x and z, in the feature space, ϕ, that the kernel defines: $K(x, z) = \langle \phi(x), \phi(z) \rangle$

2.1 Kernelised k-NN

A machine learning algorithm may be "kernelised" by first reformulating the algorithm so that all data enters it in the form of dot-products of sample pairs. Each dot-product calculation in the algorithm is then replaced by a kernel function, thus transforming the algorithm into the feature space defined by that kernel. The classic example of a kernel method is the SVM [4]. However, other machine learning algorithms can be reformulated as a kernel method, one example being k-NN, a method that can be used in both classification and regression settings. There have been many variants of the k-NN algorithm, but the basic idea is as follows: the distance between the test sample and each sample in the training set is calculated to determine the k samples that are closest to the test sample; in classification, the majority class of these nearest samples (or nearest single sample when k = 1) is returned as the prediction for that test sample; in regression, the (possibly weighted[1]) average value of the dependent variable for the k nearest samples is returned as the prediction.

The k nearest samples are often determined using the Euclidean distance, $d = ||x - z||$. With kernel methods, the kernel's feature space is known as a *dot product space* and therefore has a naturally defined norm: $||x|| = \langle x, x \rangle$. Any norm defines a metric d via [5]:

$$\begin{aligned} d(x, z) &= ||x - z|| \\ &= \sqrt{\langle x - z, x - z \rangle} \\ &= \sqrt{\langle x, x \rangle + \langle z, z \rangle - 2\langle x, z \rangle} \end{aligned} \tag{1}$$

A kernel distance metric is therefore defined as:

$$d_K(x, z) = \sqrt{K(x, x) + K(z, z) - 2K(x, z)} \tag{2}$$

The above equation can be used to derive distance measures from any kernel, which can substitute the Euclidean distance measure in a k-NN algorithm[2].

[1] The k-NN implementation used for the experiments reported in this paper does not employ a distance-weighting mechanism.

[2] In this work, a distinction is made between the Euclidean distance measure and kernel-based distance measures. We use the term 'Euclidean distance' to refer to the conventional k-NN distance measure defined in the original input space and we use the term 'kernel-based distance' to refer to the Euclidean distance as defined in the kernel transformed space.

2.2 Kernel Function

As with any kernel method, a key step in the application of kernel k-NN is kernel selection. With SVMs, for example, typical choices for kernels are the Linear, Polynomial, RBF and Sigmoid kernels. 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.* [6] and kernels for protein classification [7]. 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].

3 KTree

As previously highlighted, 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 [8].

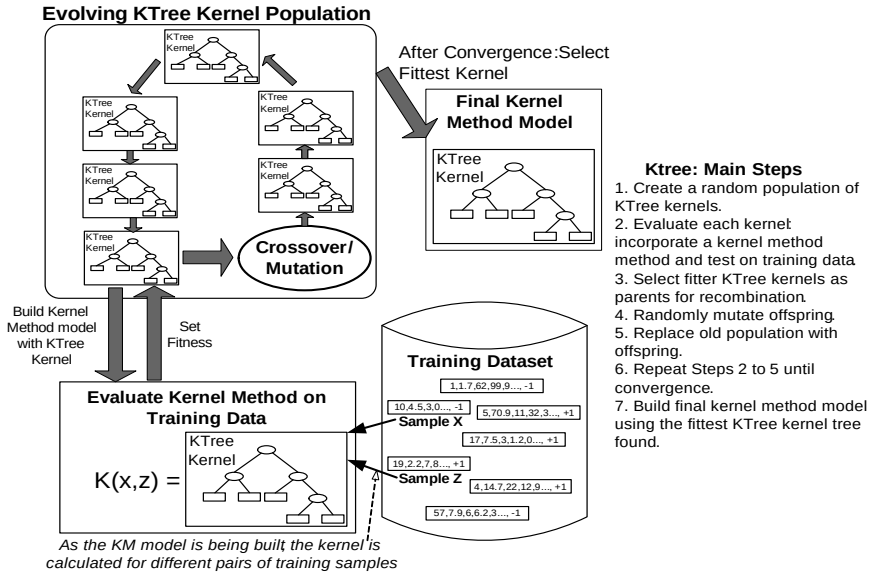

Fig. 1. KTree algorithm

Kernel method users can select from one of the standard kernels, construct new kernels using simpler kernels as building blocks, e.g. the kernel, $K(x,z) = K_1(x,z) + K_2(x,z)$, or customise a kernel for a particular problem. Ideally, a kernel is selected or customised based on prior knowledge of the problem domain, but it is not always

possible to make the right of choice of kernel *a priori*. KTree addresses this problem by using the evolutionary technique of Genetic Programming (GP) to discover a suitable kernel for a particular problem. KTree has been previously demonstrated with SVM classifiers [3], but this approach can be used with other kernelised pattern analysis algorithms. The aim of KTree is the discovery of new kernels that best represent the underlying data from a particular problem domain; in the context of kernel k-NN, KTree allows for the discovery of a new distance metric for a particular data domain. With KTree, a tree structure, known as a *KTree kernel* (see Figure 2) is used to represent a kernel function. The objective of KTree is to find a KTree kernel that best represents the data. An overview of the KTree algorithm is shown in in Figure 1.

3.1 KTree Kernel Representation

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

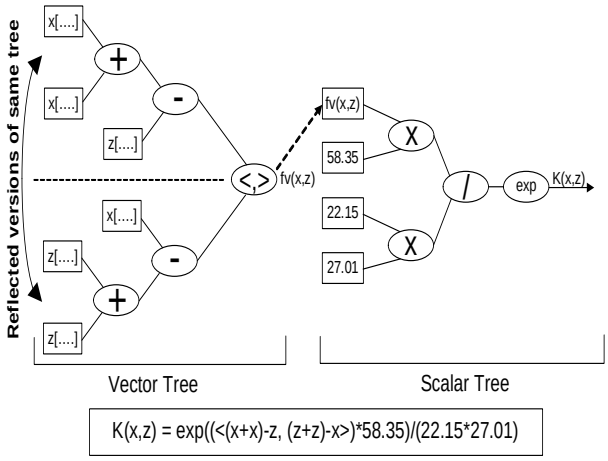

Fig. 2. Example KTree Kernel

The diagram shows that the KTree kernel 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 used in SVMs, e.g. the RBF kernel and Polynomial

kernel. Although symmetry is satisfied, this kernel design is not guaranteed to produce Mercer kernels. However, non-Mercer kernels are filtered out (see Section 3.2).

A specification of the KTree kernel is given in Table 1, showing the input terminals and operators used for the vector and scalar tree parts of a KTree kernel. The vector tree is a GP tree, where the input terminal set comprises the two vector inputs to the kernel function, x and z. The operators of the vector tree take two vectors as an input and return a single vector as output. A vector operator is calculated as follows:

$$[x_1, x_2, \ldots, x_m]\ op\ [z_1, z_2, \ldots, z_m] = [x_1\ op\ z_1, x_2\ op\ z_2, \ldots, x_m\ op\ z_m] \quad (3)$$

where op is one of the operators listed for the vector tree in Table 1 and m is the length of the vector inputs. For example, an addition in the vector tree is calculated as follows:

$$[x_1, x_2, \ldots, x_m] + [z_1, z_2, \ldots, z_m] = [x_1 + z_1, x_2 + z_2, \ldots, x_m + z_m] \quad (4)$$

The scalar tree of a KTree kernel is a GP tree, where the input terminal set comprises the output of the vector tree, denoted as $f_v(x, z)$, and a set of randomly generated constants. Note that $f_v(x, z)$ may occur multiple times in a scalar tree. The set of scalar operators (unary and binary) used in the scalar tree is listed in Table 1. Note that while the use of constants in the scalar tree of the kernel influences the decision boundary of an SVM classifier, it has no bearing on the ordering of neighbours for k-NN without distance-weighting.

Table 1. KTree kernel specification

Vector Tree	
Input Terminals:	$x[..]$, $z[..]$
Operators:	add, $subtract$, $multiply$
Scalar Tree	
Input Terminals:	$const$, $f_v(x, z)$
Operators:	add, $subtract$, $multiply$, $divide$, exp, $power$, $tanh$

As shown in Figure 1, the first step of the KTree algorithm is to create a random population of kernels. For this initial population, each KTree kernel (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 KTree kernels. 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 KTree kernels begins with the selection of a random point from either the vector or scalar part of the first KTree kernel. The location of the crossover point on the second

KTree kernel is constrained so that crossover does not occur between the scalar part of one KTree kernel 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 KTree kernels after mutation, maintaining a maximum depth of 12 (for either the vector or scalar part). A population of 500 KTree kernels was used for all experiments.

3.2 Fitness Function

As with any evolutionary algorithm, a key element of KTree is the choice of fitness function. In previous work on the use of KTree for SVM classification [3], the authors investigated a number of fitness functions and found that the best results were achieved with a fitness function based on an internal cross-validation (3-fold) coupled with a tiebreaker fitness that favours smaller KTree kernels. This investigation also found that the stability of KTree was improved by the use of a *Mercer filter*. Furthermore, a non-Mercer kernel does not define a distance metric as described in Section 2.1. The Mercer filter estimates the Mercer condition of a kernel by calculating the eigenvalues of the kernel matrix over the training data; 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 subset is randomly selected and the same subset is used in each kernel evaluation. For the experiments reported here, the kernel matrix was limited to a maximum size of 250x250.

4 Experimental Results

The goal of the experiments presented here is to determine if KTree can evolve kernel-based distance metrics that improve on the standard Euclidean distance when embedded in a k-NN regression algorithm. The next two sections describe experiments based on synthetic and real-world data and discuss the results.

4.1 Synthetic Dataset

A synthetic dataset, named *FeatureSpace*, was devised to comprise two predictor attributes and one dependent attribute, the value of which is to be predicted using k-NN regression. *FeatureSpace* is based on a specified feature mapping from a two-dimensional to a three-dimensional space. To create this dataset, 1000 two-dimensional points were randomly generated. The following feature space mapping was then applied to each point, x_i:

$$\begin{aligned} \phi_1(x_i) &= (x_{i1} - x_{i2})^2 \\ \phi_2(x_i) &= (x_{i1} + x_{i2} + 1)^3 \\ \phi_3(x_i) &= x_{i1} x_{i2} \end{aligned} \tag{5}$$

where x_{i1} is the value of the first attribute of sample x_i, x_{i2} is the value of the second attribute of sample x_i, and $\phi_p(x_i)$ is the value of the mapping of x_i along the

p-th axis in the new feature space. To generate the value of the dependent variable for each sample, y_i, for each sample, the following simple function is used:

$$y_i = f(x_i) = \phi_1(x_i) + \phi_2(x_i) + \phi_3(x_i) \quad (6)$$

Table 2 compares the performance of k-NN on the FeatureSpace dataset with different distance metrics. This experiment uses 200 samples of the FeatureSpace dataset for training and the remainder of the dataset as the test set. Table 2 shows the root relative squared error of prediction [9] achieved by k-NN with each distance metric on the test set. This table also shows the fitness of the final kernel selected by KTree and compares this with the 'fitness' of the two other distance metrics; the fitness of the Euclidean and FeatureSpace kernel distance is calculated using the same evaluation function as used by KTree, i.e. the average root relative squared error over a 3-fold cross-validation run on the training subset[3]. The results indicate that 3-NN with a distance metric based on the evolved KTree kernel improves on the performance of 3-NN with the standard Euclidean distance, both in terms of test error and fitness on the training data. Note that the higher errors over the training set may be due to the smaller training set used within the 3-fold fitness evaluation than that used for the test set.

Table 2. Results of 3-NN with different distance metrics on the FeatureSpace dataset. Both fitness and error values are the root relative squared error of prediction achieved by k-NN using each distance metric. Fitness is the 3-Fold CV root relative squared error over the training subset.

Distance Metric	**Training Set Fitness (3-Fold CV Error)**	**Test Set Error**
Euclidean	17.61%	14.51%
KTree	15.56%	11.15%
FeatureSpace Kernel	11.81%	6.54%

The final row of Table 2 shows the fitness and test error for 3-NN based on the FeatureSpace kernel. This kernel explicitly maps its two inputs according to the mapping defined in Equation 5 to generate two vectors of length 3, and then returns the dot-product of these two vectors. Using 3-NN with the FeatureSpace kernel is equivalent to operating 3-NN in the original three-dimensional feature space, in which the target function was defined. This result represents the best result that could be achieved with 3-NN. Note that the FeatureSpace kernel does not achieve 0% error as the training set does not provide 100% coverage of the target function; this was confirmed by calculation of the minimum theoretical error for 1-NN with the same training and test sets, which was found to be 5.16%. The results show that the performance of KTree on the test set is roughly half-way between that of the Euclidean distance and the FeatureSpace kernel distance. Despite the good performance of KTree relative to the benchmark of the Euclidean distance, the FeatureSpace kernel distance result shows that better kernels could possibly be found, e.g. by increasing population size or by increasing the maximum number of generations allowed.

[3] Since fitness is computed using an error measure, lower fitness values are better.

4.2 Real-world Datasets

To extend the results of KTree with k-NN on the synthetic dataset, a similar comparison of KTree with Euclidean distance was carried out, using a number of real-world regression datasets [10]. Table 3 shows the average error from a single 10-fold cross-validation run over 10 regression datasets; due to the relatively large size of the Abalone dataset (4177 instances) and the resultant significant increase in computation that would be required for KTree, a single subset of 250 instances was used for the training phase with the remainder being used as the test set. This table shows the results for 3-NN based on Euclidean distance and the results for 3-NN based on a KTree-evolved kernel distance. The lowest numerical test error on each dataset is highlighted in bold.

Table 3. k-NN regression 10-fold CV error rates (RMSEP): KTree-evolved distance vs. Euclidean distance. *The errors reported for this dataset are based on a single train/test split. Table also includes the number of samples and attributes of each dataset.

	No. Samples	**No. Attributes**	**Average Test Error**	
			Euclidean	**KTree**
Auto-MPG	398	8	2.86±0.47	**2.58±0.51**
CPU	209	7	0.04±0.04	**0.03±0.03**
Boston-Nox	506	14	0.04±0.01	**0.036±0.01**
Boston-Price	506	14	4.18±1.14	**3.45±1.11**
Octane	82	5	**0.61±0.16**	0.62±0.24
Deathrate	60	16	46.79±14.89	**46.63±10.49**
Bodyfat	252	15	2.87±0.36	**2.75±0.45**
Houseprice	117	7	212.97±62.89	**210.31±70.73**
Tecator	240	101	2.47±0.48	**2.35±0.38**
NO2	500	7	0.56±0.08	**0.55±0.08**
Abalone*	4177	11	2.69	**2.60**

For these experiments, the error is the root mean squared error of prediction (RMSEP). 3-NN using KTree achieves the lowest numerical test error on all datasets, except for the Octane dataset. A pairwise comparison over all datasets based on a Wilcoxon Signed Rank test [11] at a confidence level of 5% shows that 3-NN based on KTree outperforms the standard 3-NN. This demonstrates the ability of KTree to derive new distance metrics for a given data domain. In a similar analysis to that carried out for the synthetic dataset, the average fitness (based on 3-fold cross-validation RMSEP) of the Euclidean distance and KTree-evolved kernel distance was compared; the results are shown in Table 4. As was previously found with KTree for SVM classification, KTree is the clear winner in terms of the fitness of the distance metric it derives; this is also confirmed by the Wilcoxon Signed Rank test at a confidence level of 5%.

Overall, the results on both synthetic and real datasets demonstrate the effectiveness of the data-driven evolutionary approach of KTree when applied to a k-NN regression task. KTree is capable of evolving a kernel-based distance metric that is suited to a particular dataset. Distance metrics evolved by KTree could be used in library search applications, where the goal is to search for a list of the closest matches to a test sample.

Table 4. Average fitness (based on 3-fold CV RMSEP): KTree-evolved distance vs. Euclidean distance. *The fitness values reported for this dataset are based on a single train/test split.

	Average Fitness on Training Data			Average Fitness on Training Data	
	Euclidean	**KTree**		**Euclidean**	**KTree**
Auto-MPG	3.07±0.06	**2.57±0.07**	Bodyfat	3.44±0.12	**2.99±0.19**
CPU	0.06±0.01	**0.04±0.00**	Houseprice	221.38±13.42	**176.35±11.23**
Boston-Nox	0.05±0.00	**0.04±0.00**	Tecator	2.57±0.1	**2.44±0.1**
Boston-Price	5.12±0.21	**4.14±0.44**	NO2	0.57±0.01	**0.55±0.01**
Octane	0.71±0.04	**0.67±0.043**	Abalone*	2.48	**2.27**
Deathrate	48.82±3.27	**41.07±2.02**			

5 Related Research

Some previous work on the use of evolutionary algorithms with kernel-based learning has focussed on the optimisation of a single kernel, e.g. the RBF kernel for SVM classification [12]. In Lessmann *et al.* [13], a GA is used 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. This approach is guaranteed to produce Mercer kernels, provided the Sigmoid kernel component setting does not break Mercer's condition. In comparison with KTree, however, the approach of Lessmann *et al.* is significantly restricted in the range of kernels that it can generate.

In more recent research, Gagne *et al.* [14] have proposed an approach for evolving kernels for a k-NN classifier. This approach is called the Evolutionary Kernel Machine (EKM). Although EKM bears some similarity to KTree in its use of GP to evolve a kernel, it differs considerably in a number of areas, including the kernel function representation and fitness measure used to evaluate candidate kernels. Their approach does not guard against non-Mercer kernels and the fitness function is based on k-NN specifically, i.e. the same fitness could not be used with SVMs, as is the case with the fitness function of KTree. Furthermore, EKM uses a co-evolutionary framework to evolve two subsets of the training dataset, a *fitness* and a *prototype* set, which are used in the fitness measure (see Gagne *et al.* for more details on this fitness measure). The authors do note that this competitive co-evolution can be problematic in that there is a danger of the fitness subset capturing noisy examples, thus resulting in a poor final model.

6 Conclusions

This paper has demonstrated the use of KTree to evolve a kernel-based distance metric for use in a k-NN regression algorithm. Experiments on both synthetic and real-world data showed that KTree is capable of evolving a distance metric that can improve on the widely used Euclidean distance. This represents a novel approach that facilitates the automatic discovery of a custom distance metric for a particular data domain. In building on previous work with KTree, these results also demonstrate that KTree can

be applied to different kernel methods and to different machine learning problems, i.e. classification and regression. One of the great advantages in the use of kernel methods is that they are easily adapted to work with different types of data; provided a kernel function can be defined for comparing two data objects, any kernel method can be applied to this data and a kernel-based distance metric may also be derived. Future work could investigate the use of KTree in structured data domains, such as the protein structure classification problems tackled by Wang & Scott [7]. The extension of this KTree research to tackle such problems may require the definition of new operators for the KTree kernel, which would allow it to manipulate structured data objects.

Acknowledgements

This first author's research has been funded by Enterprise Ireland's Basic Research Grant Programme. The second author acknowledges the support of a Marie Curie Transfer of Knowledge Fellowship of the European Communitys Sixth Framework Programme, Contract MTKD-CT-2005-029611. The authors are also grateful to the High Performance Computing Group at NUI Galway, funded under PRTLI I and III, for providing access to HPC facilities.

References

1. Karakoc, E., Cherkasov, A., Cenk Sahinalp, S.: Distance based algorithms for small biomolecule classification and structural similarity search. **22** (2006)
2. Yu, K., Ji, L., Zhang, X.: Kernel Nearest-Neighbour Algorithm. Neural Processing Letters **15** (2002) 147–156
3. Howley, T., Madden, M.G.: An evolutionary approach to automatic kernel construction. In: Proceedings of the International Conference on Artificial Neural Networks (ICANN). (2006)
4. Cristianini, N., Shawe-Taylor, J.S.: An Introduction to Support Vector Machines. (2000)
5. Scholkopf, B., Smola, A.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press (2002)
6. 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
7. Wang, C., Scott, S.D.: New Kernels for Protein Structural Motif Discovery and Function Classification. In: Proc. of the 22nd International Conference on Machine Learning. (2005)
8. Cristianini, N., Shawe-Taylor, J.: Kernel Methods for Pattern Analysis. (2004)
9. Witten, I., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann Publishers (2000)
10. Newman, D.J., Hettich, S., Blake, C.L., Merz, C.J.: UCI Repository of machine learning databases (1998)
11. Demsar, J.: Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research **7** (2006) 1–30
12. Friedrichs, F., Igel, C.: Evolutionary Tuning of Multiple SVM Parameters. In: Proc. of the 12th European Symposium on Artificial Neural Network. (2004) 519–524
13. Lessmann, S., Stahlbock, R., Crone, S.: Genetically constructed kernels for support vector machines. In: Proc. of General Operations Research (GOR). (2005)
14. Gagne, C., Schoenauer, M., Sebag, M., Tomassini, M.: Genetic Programming for Kernel-based Learning with Co-evolving Subsets Selection. In: Parallel Problem Solving from Nature (PPSN IX). (2006)