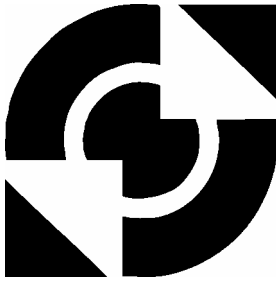
University of Twente

EEMCS / Electrical Engineering
Control Engineering



Benchmarking of the Key Sample Machine

Yu Liu

M.Sc. Thesis

Supervisors
prof.dr.ir. J. van Amerongen
dr.ir. T.J.A. de Vries

August 2005

Report nr. 028CE2005
Control Engineering
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Summary

In learning control schemes such as Learning Feed-Forward Control(LFFC), a function approximator is embedded to approximate the inverse of the plant under control. The function approximation concerns itself with finding a relation within a set of samples supplied before. The goal of the function approximator is to select the best function from a set of possible functions to approximate the true relation contained in the noisy data set.

In previously LFFC research, different function approximator has been used: the Least Squares Approximator, the neural networks such as Multilayer Perceptron (MLP)and B-splines Networks, the Support Vector Machines (SVM) and the Least Squares Support Vector Machines (LSSVM), ect. Recently a new off-line approximator called Key Sample Machine (KSM) has been developed during the Ph.D research of dr. ir. B.J. de Kruif. A series of tests had been made to evaluate the performance of KSM and showed it gives a smaller prediction error for noisy data in a shorter time compared with other methods.

However, these tests have not been performed with a sufficient number of different problems/data sets. With a smaller volume of data sets, it is impossible to characterize the behaviour of KSM in a broad statistical view. For this purpose a standard of experimentation should be valid in the sense that there are no artefacts created by random factors or by a faulty experimental setup.

The goal of this research is to test and evaluate the KSM algorithm by standard experiments and to compare it with other available methods (SVM,LSSVM,RBFN) on standard sets of regression problems through simulation.

Nine well-known approximation benchmarking problems are used in our simulation, including six real-world data sets and three artificial data sets. The comparison is performed with respect to the ability to approximate various data sets, the efficiency of learning, and the ability to train samples from the real-world data.

Simulation results show that the KSM yields good performance, especially when considering the efficiency, which is very important for its application in the online LFFC setting. In short, our results confirm the potential of KSM to be a good function approximator.

Acknowledgement

I will never forget the past eight months on my thesis. Having Started it late due to personal issues brought me lot of pressures on my work. During these amazing months, I had been heavy-hearted and frustrated, I had taken heart of grace, I had learned and made progress. By the time this thesis is finally finished, I am filled with emotions and I would like to make a grateful acknowledgement for the people who have helped me directly or indirectly to achieve my work.

I would like to give my sincere appreciate to every staff in the Control Engineering Group, where I study and make friends. I would like to thank prof. dr. ir. J. van Amerongen and dr.ir. Theo de Vries for the supervision during this graduation project. I am grateful for the understanding and consoling of all the people who know what happened to my family, especially Dr. Willemien Wallinga-de Jonge, who encourage me to get strong.

My special gratitude goes to my mother, who always give support and courage during my life in this world. The last but not the least, my affectionate deceased grandmothers, who left without seeing my face. For them I dedicated this thesis. I love you all.

Enschede, end of August, 2005

Contents

Chapter 1

Introduction

In learning control schemes such as Learning Feed-Forward Control(LFFC), a function approximator is embedded to approximate the inverse of the plant under control. The function approximation concerns itself with finding a relation within a set of samples supplied before. The goal of the function approximator is to select the best function from a set of possible functions to approximate the true relation contained in the noisy data set.

In scientific literature on function approximation, a vast amount of methods has been introduced. The most common methods used in learning control are Ordinary Least Squares and Artificial Neural Networks. Previously in LFFC research[1], B-spline Networks have been used for some advantages such like simplicity. There are also methods as Support Vector Machines which use a hypothesis space of linear functions in a high dimensional feature space to avoid the curse of dimensionality. Recently a new off-line approximator called Key Sample Machine (KSM) has been developed[2]. A series of tests on two data sets (one is show in figure 1.1) had been made to evaluate the performance of KSM and showed it gives a smaller prediction error for noisy data in a shorter time compared with other methods.

1.1 Motivation

For benchmark purpose, there is a need for standard sets of problems and rules or conventions for applying them to be used in learning algorithm evaluations[3]. It is at least necessary to use different sets of data for training and testing[4]. Normally the data set used for training and testing should be Real World Data Sets[5], which represent actual observations of phenomena in the physical world. Such data tends to contain some amount of errors and noise. However, the real data usually has characteristics that are not completely known (surprising features). So there is another kind of data

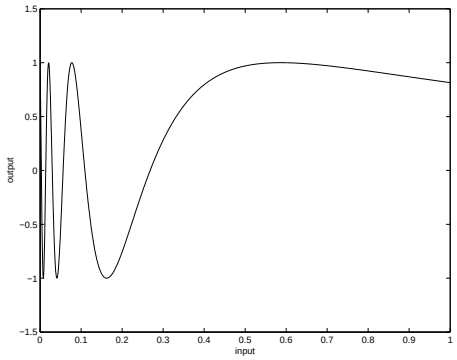


Figure 1.1: Previous data set used in test of the KSM

called Realistic Data Sets, which is synthetic data, but is constructed by developing generative statistical models with properties similar to what can be found in real problems. Algorithms of function approximator should be test on both these kinds of data sets to prove they are solving real problems.

The data in Figure 1.1 is realistic data generated by computer, but it is connected to real life applications. Another data set use in previous tests is the cogging force of a linear motor in the LFFC experiment, which is a Real World Data Set[6]. However, we still have to say that the test has not been performed with a sufficient number of different problems (data sets). From the point of view of benchmarking, empirical researches has shown that an algorithm evaluation is called acceptable if it uses a minimum of two real or realistic problems and compares the results to those of at least one alternative algorithm. With a smaller volume of data sets, it is impossible to characterize the behavior of a new algorithm in a broad statistical view. For this purpose a standard of experimentation should be valid in the sense that they are not artefact created by random factors or by a faulty experimental setup. There is a need for standard sets of problems and rules to be used in KSM evaluations.

The goal of this research is to test and evaluate the KSM algorithm on standard benchmarking data sets and to compare it with other function approximators in statistical experiments. Because the function approximator is embedded in a learning control setting, the aim is to test whether the KSM is a good method or improvement in conditions of control. For instance the computation time of a prediction and memory requirements, meaningful predictions based on the samples, coping with noisy training data or ambiguities in the data, high input dimension, the ease of use, or some combination thereof. This means that we want to quantitatively predict the behavior of the KSM compared to other function approximators for any of these criteria. Consequently, experimental evaluation is needed to validate any claims of improvement made for KSM or to characterize under which circumstances improvements can be expected. In the

research the following tasks are to be performed:

1. Literature studying about function approximators.
2. Theoretical comparison of KSM with other methods used in this thesis.
3. Comparison of KSM with other function approximators in an ensemble of aspects by standard experimental simulation.
4. Evaluation the suitability of the KSM, as well as other methods, as a function approximator embedded in learning control setting, especially in LFFC.

1.2 Report structure

The structure of the report is as follows: In chapter 2, a review of previously used function approximators, which are important for understanding the algorithm of KSM, will be done. Then the KSM algorithms and a theoretical comparison between the KSM and other methods will be given in chapter 3. The simulation comparison on new data sets in different aspects will be performed in chapter 4. Then the KSM is benchmarked against others on publicly available benchmarking data sets. In chapter 5 finally, the results will be showed and conclusion will be drawn and recommendations will be given.

Chapter 2

Available methods

In this chapter, the problem of function approximation will be addressed and the most common and basic learning method, Least Squares approximation, will be treated at the beginning to demonstrate the problem. Then an overview of the up to date learning methods, which also are compared in this thesis, will be given. Those methods are the Radial-Basis Function Network (RBFN), the Support Vector Machine (SVM), and the Least Square Support Vector Machine (LSSVM). Each learning method will be briefly introduced followed by an insight into its working principle. After all methods have been presented, their working principles will be compared and discussed in the next chapter.

2.1 Function approximation problem

The problem of approximation consists in finding a function that best fits the underlying function of a given set of training points. In other words, a function approximator finds a relation in data from some input(s) to an output. When the relation is found, it can be used to predict the output for new, unencountered inputs. In practice, data is usually noisy, so the better function approximator can achieve a smooth fit that averages out the noise in the data but dose not fit the noise.

The complete approximator setup is depicted in figure 2.1[2]. The upper-part G is the data-generator, while the bottom-part represents the function approximator. G generates a series of inputs $\mathbf{x}$. Then the supervisor S, assigns a value y (normally noisy) to each input $\mathbf{x}$. The function approximation is provided with the outputs of the supervisor and the input to the supervisor and has to select the best function from a set of possible functions to predict the behavior of the supervisor, or to fit the data supplied. Function approximators are often used in learning control, system identification and econometrics, etc. In this thesis we put our emphasis on the situation

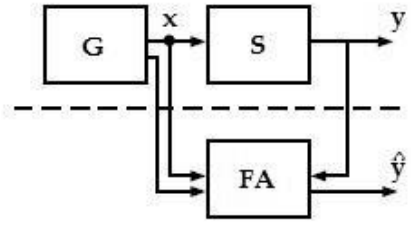


Figure 2.1: The learning problem

of learning control systems.

In learning control schemes such as Learning Feed-Forward Control, a function approximator is embedded to approximate the inverse of the plant under control. The function approximation concerns itself with finding a relation within a set of samples supplied before to approximate the true relation contained in the noisy data set. Since the function approximator is also a part of a controller, it should be suited for control or it should comply with the conditions in the situation of control. These conditions are:

- The memory space that is needed for implementation should be small. In practice, the controller is implemented in software on an embedded computer. The amount of memory is limited. Therefore, the number of parameters the function approximator requires to approximate the control signal may not be too large.
- Calculation of the output of the function approximator and adaptation of the input-output relation must be done fast.
- The learning mechanism should converge fast.
- The function approximator should possess a good generalising ability. The generalising ability is the ability to produce a sensible output for an input that has not been presented during training.
- The ability to deal with high dimensional inputs. The problem is known as *Curse of Dimensionality.*

The fact that the function approximator minimizes some risk often leads to a minimization problem. In theoretical point of view, the selection of different cost functions results to different performance of the approximation. In practice, the training of a function approximator involves two steps[3]: 1. training 2. validation. If the function approximation has selected the best approximation, the approximation should be tested

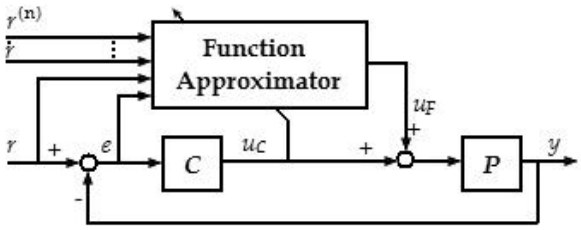


Figure 2.2: Learning Feed-Forward Control

for correctness. Cross-validation is used to test whether the approximator has selected a function that really approximates the underlying relation, and not the current noise realisation.

2.2 Least Squares Approximation

2.2.1 Ordinary Least Squares

The most common method is Least Squares(LS) approximation, which has been proposed independently by Gauss and Legendre. It is concerned with choosing the line that minimizes the sum of the squares of the distances from the training points. This technique is known to be optimal in the case of linear targets corrupted by Gaussian noise. It is also the most widely used function approximator because of its simplicity and robustness.

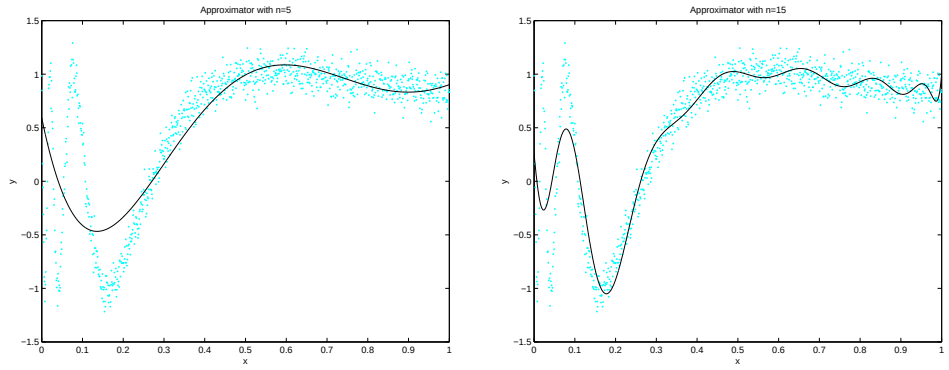
The LS (or OLS) method tries to find a linear relation between a set of fixed functions $f_i(x), i = 1 \dots n$ and the output $\hat{y}$ of the form:

$$\hat{y}(x) = b_1 f_1(x) + b_2 f_2(x) + \ldots + b_n f_n(x) \tag{2.1}$$

The functions f_i are called indicators or basis functions, which implement a mapping from the input space to a feature space. A linear approximation is made in this feature space.

The value of the parameter $\mathbf{b}$, containing the elements b_i, follows from the training samples. Define matrix $\mathbf{X}$, column-vectors $\mathbf{y}$ and $\mathbf{b}$ containing respectively the indicators for all the N samples, the output and the parameters:

$$\mathbf{X} = \begin{pmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_N) & f_2(x_N) & \cdots & f_n(x_N) \end{pmatrix}_{N \times n}, \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}, \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}, \tag{2.2}$$



(a) lower order polynomials (b) higher order polynomials

Figure 2.3: LS approximation with rigid structure

where the subscript of $\mathbf{x}$ and $\mathbf{y}$ indicates the sample number. $\mathbf{y}$ is assumed to be corrupted by Gaussian noise, denoted by ϵ:

$$\mathbf{y} = \mathbf{X}\mathbf{b} + \epsilon \tag{2.3}$$

The minimization of the approximation error with a quadratic cost function is given as:

$$\min_b \parallel \frac{1}{2}(\mathbf{X}\mathbf{b} - \mathbf{y}) \parallel_2^2 \tag{2.4}$$

The solution of the minimization problem leads to the so-called normal equation:

$$\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{y} \tag{2.5}$$

And if the inverse of $\mathbf{X}^T\mathbf{X}$ exists, the solution of the LS problem is

$$\mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{2.6}$$

If $\mathbf{X}^T\mathbf{X}$ is singular, the pseudo-inverse can be used, or else a ridge regression can be applied. When the parameter $\mathbf{b}$ is determined, the output for a new sample x_{new} can be predicted. Based on (2.1), the prediction is calculated as:

$$\hat{\mathbf{y}} = f(\mathbf{x}_{new})\mathbf{b} = \sum_{i=1}^{n} b_i f_i(\mathbf{x}_{new}) \tag{2.7}$$

In the OLS, we use n indicator functions to map the input space to the feature space. The set of n functions fixes the structure of the approximator beforehand. There

are two structural forms of the indicator functions.

a. **Rigid structure**: The function structure is unalterable during training.

b. **Flexible structure**: the structure can be altered throughout the training.

Assume we want to approximate with a polynomial function, composed of elementary basis functions with rigid structure determined by the powers 0...n of x:

$$\hat{y} = b^0 + b^1 x + b_2 x^2 + \ldots + b^n x^n \tag{2.8}$$

For example, if we use the dataset1 of Figure 1.1, which is formulated as:

$$y = \sin(\frac{1}{x+v}) + \epsilon \tag{2.9}$$

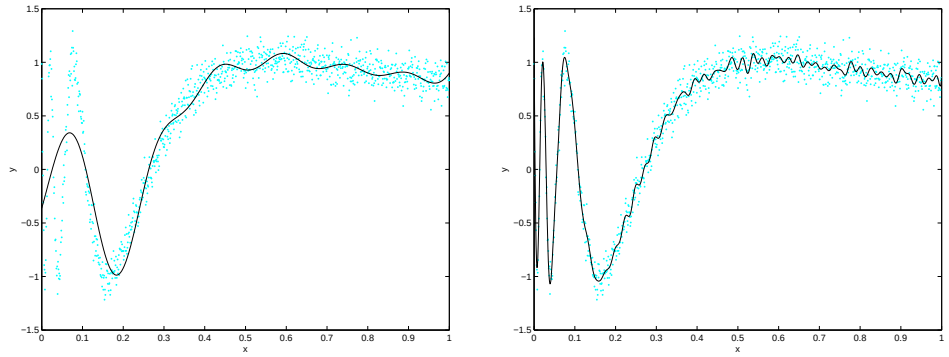
where v is a parameter affecting the fluctuation of the function with $v = 0.05$ and denote noise with a zero-mean Gaussian distribution. Figure 2.3(a) shows the result of $n = 5$; it is obvious that the approximation is not good because the order of the polynomial is too low. Then the order of the polynomial is increased to 15. The new predictions are showed in Figure 2.3(b). Even for these higher order approximations the prediction is not as good as we expect. For small values of x, the large fluctuation can not be reconstructed, although an increase of n can give a little improvement. Besides, the approximation starts to fluctuate for large value of x and fluctuates more when n is increased further. So we can say that this polynomial indicator function of rigid structure is not a good approximator for this data set, which fluctuates swiftly for small values of x and very slowly for large values of x.

If the structure can be altered throughout the training, the limited Gaussian functions are used as indicator functions to approximate the data. The Gaussian function is given as:

$$f_i(x) = \exp(-\frac{(x-c_i)^2}{2\sigma^2}) \tag{2.10}$$

The width of this function is fixed to be 0.11, while the centers, c_i, are selected to coincide with specific inputs of the training data. Because of the limited memory, the number of the Gaussian functions can not be large.

When 15 Gaussian functions are used, the result is showed in the left side of Figure 2.4(a). It is still not good at the beginning as the result of the low number of basis functions. It is not flexible enough to reveal the detail information underlying the data relation. The result of increasing the number of Gaussian functions is show in figure 2.4(b). As the number of indicator increases, the approximation then is too flexible and tries to fit all information, including noise.



(a) lower number of Gaussian functions (b) higher number of Gaussian functions

Figure 2.4: LS approximation with flexible struture

LS has been used in numerous applications. Although the fast calculations of parameter vector and the simplicity make it easy to use, the bad fitting illustrated in above examples shows its pitfalls in approximation, especially when there are swift waves in particular areas of the target function. If the structure underlying the relation is known, and only some parameters need to be estimated, then it is a good approximator. In addition, the computation of the inverse of may bring numerical stability problems. Before approximation, the indicator functions need to be decided and it is sensitive to outliers. The main difficulty is to select a good set of indicator functions beforehand.

2.2.2 Dual Least Squares

Instead of equating the derivatives of the minimization criterion to zero as in (2.5), the cost function can also be minimized by the use of optimization theory. Thus we obtain a dual representation of the problem:

$$\begin{aligned} \mathbf{X}\mathbf{X}^T\mathbf{a} &= \mathbf{y} \\ \mathbf{b} &= \mathbf{X}^T\mathbf{a} \end{aligned} \tag{2.11}$$

The matrix $\mathbf{X}\mathbf{X}^T$ contains the inner products of all the training samples with the other training samples in the feature space.

$$\mathbf{X}\mathbf{X}^T_{i,j} = f(\mathbf{x}_i)f(\mathbf{x}_j)^T \tag{2.12}$$

The prediction of new input $\mathbf{x}_{new}$ (2.6) becomes:

$$\begin{aligned} \hat{y}(\mathbf{x}_{new}) &= f(\mathbf{x}_{new})\mathbf{b} = f(\mathbf{x}_{new})\mathbf{X}^T\mathbf{a} \\ &= \sum_{i=1}^{n} b_i f_i(\mathbf{x}_{new}) = \sum_{i=1}^{n}(\sum_{j=1}^{N} f_i(\mathbf{x}_j)a_j f_i(\mathbf{x}_{new})) = \sum_{j=1}^{N}(\sum_{i=1}^{n} f_i(\mathbf{x}_j)f_i(\mathbf{x}_{new}))a_j \end{aligned} \tag{2.13}$$

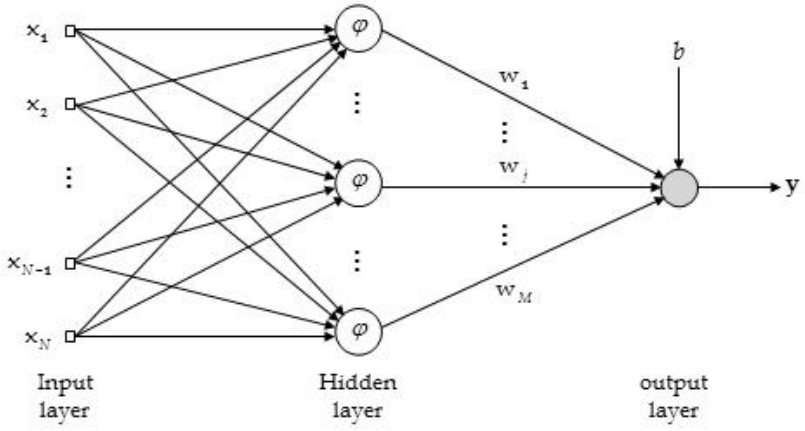


Figure 2.5: General Architecture of Radial Basis Function Networks

Now based on (2.13), when the parameter is determined, the output for a new sample x_{new} can be predicted. Note that this time the calculation does not depend on the individual elements of the feature, in other words, the number n of the indicator functions. Therefor"e, large dimensional feature spaces can be used. The parameter then has the dimension of N, means that the structure of approximation in now based on the actual data samples.

2.3 Radial Basis Function Network

Function approximation is one of the most general uses of artificial neural networks (ANN). The most popular ANN is the multilayer perceptrons (MLP). The number of parameters of an MLP determines maximum accuracy of the approximation. But the MLP has shown been proved not good for control because it does not guarantee a certain smoothness in different parts of the input space. The Radial Basis Function Networks (RBFN) have the advantage of being much simpler than the perceptrons while keeping the major property of universal approximation of functions.

Radial Basis Function Network (RBFN) is a feedforward neural network that uses the radial basis functions as its neurons in the hidden layer(s). The important point of approximation via RBFN is mapping the input space into a new space of high enough dimension. Without loss of generality, we consider a feedforward network with an input layer, a single hidden layer and an output layer consisting of a single unit. The input layer broadcasts the coordinates of the input vector to each of the nodes in the hidden layer. Each node in the hidden layer then produces an activation based on the

associated radial basis function. Finally, each node in the output layer computes a linear combination of the activations of the hidden nodes. Let N denote the dimension of the input space, then the network represents a map form the N-dimensional input space to the single-dimensional output space:

$$s : \mathbb{R}^N \rightarrow \mathbb{R}^1$$

2.3.1 From exact interpolation to function approximation

Consider the *exact interpolation problem*[7]: given a set of N different points $\{x_i \in \mathbb{R}^N \mid i = 1, 2, \ldots, N\}$ and a corresponding set of N real numbers $\{y_i \in \mathbb{R}^1 \mid i = 1, 2, \ldots, N\}$, find a function F such that:

$$F(x_i) = y_i, \quad i = 1, 2, \ldots, N \tag{2.14}$$

The solution to this problem by adopting the radial basis function approach consists of choosing a set of N basis functions, centered at the N data points, and use the function of the following form:

$$F(x) = \sum_{i=1}^{N} w_i f(\parallel x - x_i \parallel) \tag{2.15}$$

Where $\| \cdot \|$ denotes a norm that is usually taken to be Euclidean. An exact solution is possible if one can solve the unknown coefficients (weights) of the expansion:

$$\begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1N} \\ f_{21} & f_{22} & \cdots & f_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ f_{N1} & f_{N2} & \cdots & f_{NN} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} \tag{2.16}$$

where $f_{ij} = f(\parallel x_i - x_j \parallel), i, j = 1, 2, \ldots, N$. The compact matrix form of equation (2.16) is

$$\mathbf{F}\mathbf{w} = \mathbf{y} \tag{2.17}$$

It has been shown (Light, 1992) that there exists a large class of functions $\mathbf{F}(\cdot)$, including Gaussians, (inverse) multiquadrics and thin-plate splines, for which the interpolation matrix $\mathbf{F}$ is non-singular provided the data points are distinct. Hence we may solve (2.17) as:

$$\mathbf{w} = \mathbf{F}^{-1}\mathbf{y} \tag{2.18}$$

Once the weights w_i have been calculated, they can be used to predict the output by feeding the new inputs.

$$\mathbf{F}\mathbf{w} = \hat{\mathbf{y}} \tag{2.19}$$

where

$$\mathbf{F}_{new,j} = F(\parallel \mathbf{x}_{new} - \mathbf{x}_j \parallel)$$

In practice, when the training data is noisy, the interpolating function passing through every data point will lead to overfitting and thereby poor generalization. Besides, to guarantee exact interpolation, the number of the basis function required is equal to the number of samples in the data set. Thus it is apparent that for a large data set the mapping function can become very costly to evaluate. A radial basis function (RBF) model for function approximation and generalization is obtained by modifying the exact interpolation procedure. This model gives a smoother fit to the data using a reduced number of basis functions which depends on the complexity of the mapping function rather than the size of the data set. The modifications are as follows:

1. *Number of the basis function*: the number of the basis function m is typically much less than N.

2. *Centers of the basis functions*: the centers c_j of the basis functions are determined as a part of the training process or chosen by the designer, rather than being constrained to be located at each input data point.

3. *Width of the basis functions*: the width parameter σ_j are also adapted during the training process, instead of assigning the same width parameter to all the basis functions.

4. *Addition of bias parameters*: Bias in the form of a constant may be added to (2.15) to compensate for the difference between the average value over the data set of the basis function activations and the corresponding value of the targets.

By applying these modifications to the exact interpolation formula, the form of the mapping obtained is[8]

$$y_i(\mathbf{x}) = \sum_{j=1}^{m} w_j F_j(\mathbf{x}) + b \tag{2.20}$$

When Gaussian functions is used, the basis function F can be specified as G:

$$G_j(\mathbf{x}) = \exp(-\frac{\parallel x - c_j \parallel^2}{2\sigma^2}) \tag{2.21}$$

2.3.2 Radial Basis Function Network Training

We can see that constructing an RBFN involves determining the number of centers m, the center locations c_j, the bandwidth of each center σ_j, and the weights w_j . That is, training an RBFN involves determining the values of these above mentioned sets

of parameters, in order to minimize a suitable cost function. There are three different learning strategies to training the RBFN:

1. Fixed centers selected at random
2. self-organized selection of centers
3. Supervised selection of centers

Fixed centers selected at random The simplest approach is to assume fixed RBFs defining the activation functions of the hidden units. This means the locations of the centers may be chosen randomly from the training data set. Assume a normalized Gaussian RBF centered at c_j is defined as

$$G(\| x - c_j \|^2) = \exp(-\frac{m}{d_{max}^2} \| x - c_j \|^2), \quad j = 1, 2, \ldots, m \tag{2.22}$$

where d_{max} is the maximum distance between the chosen centers. The parameters that need to be trained in this way are the weights in the output layer of the network. This can be solved by using the pseudoinverse method:

$$\mathbf{w} = \mathbf{G}^{\dagger}\mathbf{y} \tag{2.23}$$

where $\mathbf{y}$ is the desired response vector in the training set and $G^{\dagger}$ is the pseudoinverse of G.

Self-organized selection of centers In this method, the network resources are allocated in a meaningful way by placing the centers of the RBFs in only those regions of the input space where significant data are present. The position is chosen according to the distribution of x_i in the space. At locations where there are few inputs x_i few nodes will be placed and conversely, a lot of nodes will be placed where there are many input data. After move the locations of the centers in a self-organized way, the linear weights of the output layer are computed using a supervised learning rule. Once the basis parameters are determined, the transformation between the inputs and corresponding outputs of the hidden units is fixed. The network can thus be viewed as an equivalent single-layer network with linear output units. The weights can be calculated by pseudoinverse method as in the previous case.

Supervised selection of centers In the method, a stochastic gradient descent method is performed on the cost function to iteratively update the parameters. Consider a cost function in form of the summed squared error (SSE) as follows:

$$\varepsilon = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.24}$$

When Gaussian basis functions are use to minimize this cost function, one readily obtains the update equation:

$$\varepsilon = \frac{1}{2}\sum_{i=1}^{N}(y_i - \sum_{j=1}^{m} w_j G) \tag{2.25}$$

Where the requirement is to find the free parameters by using the partial derivative of the cost function (2.25) with respect to w_j, c_j and σ_j.

Normally regularization is used to discourage the RBFN from overfitting the training data. A weight penalty is added to the cost function F9 as a regularization term which controls the balance between fitting the data and avoiding the penalty. In case of (local) ridge regression, the following cost function is minimised:

$$\varepsilon = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \sum_{j=1}^{m}\lambda_j w_j^2 \tag{2.26}$$

where $\{\lambda_j\}_j^m$ are the set of individual regularisation parameters associated with each basis function. The parameters λ_j not only control the bias/variance trade off, but also the smoothness in case of functions which have significantly different smoothness in different part of the input space.

2.3.3 Orthogonal Least Squares

Another strategy is to compare models made up of different subsets of basis functions drawn from a set of candidates. This is called *subset selection* in statistics[9]. To find a best subset normally requires to search from $2M - 1$ subsets (for a set of size M). So *forward selection*, which starts with an empty subset to which is added one basis function at a time - the one reduces the cost most - until some chosen criterion stops decreasing, is used. Compared with standard way of training the neural, by gradient descent, which involves the optimization of a nonlinear SSE surface in a high-dimensional space defined by the network parameters, in subset selection the optimization algorithm searches in a discrete space of subsets of a set of hidden units and tries to find the one with the lowest prediction error. The most convenient criterion of selecting is, which is developed from the leave one out (LOO) cross-validation[10], generalized cross-validation (GCV)[11]. It leads to the simplest optimization formulae. The estimated variance can be written as[12]

$$\hat{\sigma}^2_{GCV} = \frac{N\hat{y}^T P^2 \hat{y}}{(trace(P))^2} \tag{2.27}$$

where P is the projection matrix which projects vectors in N-dimensional space perpendicular to the m-dimensional subspace spanned by the neuron. By using the

projection matrix P, the error between training set output $\mathbf{y}$ and the estimated one $\hat{\mathbf{y}}$ turns out to be $\mathbf{Py}$. The cost function (2.26) then can be written as

$$\varepsilon = \mathbf{y}^T\mathbf{Py} \tag{2.28}$$

and the SSE is $\mathbf{y}^T\mathbf{P^2y}$ There are several other similar criteria which share the form (2.27)[13]. They are *unbiased estimate of variance* (UEV), *final prediction error* (FPE) and *Bayesian information criterion* (BIC). The selection criteria depend mainly on the projection matrix P. The process of adding a new basis function can be formulated as:

$$\mathbf{P}_{m+1} = \mathbf{P}_m - \frac{\mathbf{P}_m f_j f_j^T \mathbf{P}_m}{f_j^T \mathbf{P}_m f_j} \tag{2.29}$$

which expresses the relation between $\mathbf{P}_m$, the projection matrix for the m hidden units in the current subset, and $\mathbf{P}_{m+1}$, the succeeding projection matrix if the j-th member of the full set is added. The vector $\{f_j\}_{j=1}^M$ are the columns of the basis matrix F, which is the entire set of the candidate basis functions,

$$F = [f_1 \ f_2 \ \dots \ f_M] \tag{2.30}$$

of which there are $M(M > m)$ columns.

Forward selection can be speeded up even further by using a technique called **orthogonal least squares**[14]. This is a Gram-Schmidt orthogonalisation process which ensures that each new added column is orthogonal to all previous columns. This simplifies the equation for the change in SSE and results in a more efficient algorithm.

2.3.4 Discussion

The RBFN performs a nonlinear mapping from the input space to the hidden space, followed by a linear mapping from the hidden space to the output space. The RBF maps the input into the feature space. The argument of the activation function of each hidden unit computes the Euclidean norm between the input vector and the center of that unit.

The complexity of a RBFN is determined by the number of Gaussian kernels. Given that the number of RBFs and their type, training an RBFN involves determining three different parameters: the position of the centers c_i in Gaussian kernel, their variances (width) σ_i, and the weight ω_j.

Input space coverage is determined by the width and the number of RBFs. In order to represent a mapping to some desired degree of smoothness, the number of RBFs required may not be small. Furthermore, if the number of RBFs is too small or the

distance is too far, the network is not likely to be trained. Besides, if the widths are large, the RBFs may overlap each other, the need for some inputs to activate more than one RBFs may leads to more likely dependent columns represented by those RBFs. On the other hand, use of a large number of RBFs with small width will make the network behaviour closer to the exact interpolation which could result in overfitting.

The training with randomly selected center positions does not give optimal results since the results will be different for each run. While determining the centers and widths using the above mentioned self-organised training method results in that more centres are located in regions where there is more training data. But also leads to blank spots which are not covered by the RBFs. Supervised training guarantees optimal estimation of the centers and widths. However, the gradient descent method used in supervised training is a non-linear optimization technique and is computationally expensive. Since the RBF is not always convex with respective to all parameters, the gradient descent optimisation may be trapped into local minima.

The OLS learning procedure generally produces an RBFN whose hidden layer is smaller than that of an RBFN with randomly selected centres. Furthermore, the problem of numerical ill-conditioning is avoided. Due to the forward selection the computational requirements are relatively low.

RBFN suffers from the so-called curse of dimensionality, referring to the exponential increase in the number of hidden units with the dimension of the input space.

2.4 Support Vector Machines

From previous section, we have already got an impression of kernel models as used in DLS. Normally, the target concept cannot be expressed as a simple linear combination of the given attributes, but in general requires that more abstract features of the data be exploited. Kernel representations offer a solution by projecting the data into a high dimensional feature space to increase the computation power. The advantage of it is that in this representation the number of tuneable parameters does not depend on the number of attributes being used, thus overcoming the curse of dimensionality. This leads to a new domain of kernel-based learning methods, among which the most famous one is the Support Vector Machines.

The Support Vector Machines (SVM) are learning system, developed based on statistical learning theory by Vapnik and his co-workers, and form a very powerful method in classification and regression. Given a set of data points, $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, like the LS method, the SVM method tries to find a linear relation between a set of functions $f(\mathbf{x})$ and the output $\hat{y}$ of the form:

$$\hat{y}(\mathbf{x}) = \mathbf{w}^T f(\mathbf{x}) + b \tag{2.31}$$

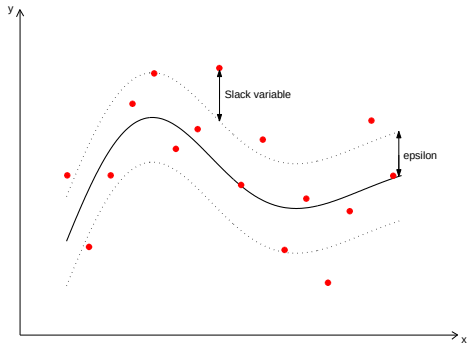


Figure 2.6: The soft margin loss setting for the SVM

where w and b are parameters to be determined by minimizing some error criterion. We define a "tube" of radius $\varepsilon(\varepsilon > 0)$ around the approximation (Figure 2.6). No error will be considered if the estimation lays inside the "tube", which leads to a ε-insensitive loss function defined as:

$$L_\varepsilon = \begin{cases} 0 & \text{for} |\hat{y} - y| < \varepsilon, \\ |\hat{y} - y| - \varepsilon & \text{otherwise.} \end{cases} \tag{2.32}$$

which says the goal of a ε-SV approximation is to find a function that has at most ε deviation from the actually obtained targets y_i for all training data, and at the same time, is as flat as possible. Flatness means that one seeks small w. One way to ensure this is to minimize the Euclidean norm $\parallel w \parallel^2$. The approximation problem can be formulated as a convex minimisation problem as follows:

$$\min_{w,b,\xi^+,\xi^-} \frac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{N} (\xi_i^- + \xi_i^+) \tag{2.33}$$

with the constraints:

$$f(\mathbf{x}_i)\mathbf{w} + b - y_i \leq \varepsilon + \xi_i^+ \tag{2.34a}$$

$$y_i - f(\mathbf{x}_i)\mathbf{w} + b \leq \varepsilon + \xi_i^- \tag{2.34b}$$

$$\xi_i^+, \xi_i^- \geq 0 \tag{2.34c}$$

The constant $C > 0$ is pre-specified value, which determins the trade off between the flatness of and the amount up to which deviations larger than ε are tolerated. ξ_i^+, ξ_i^- are slack variables representing upper and lower constraints on estimation of the target.

The Lagrangian function of the standard formulation of the above optimization problem can be defined as:

$$\begin{aligned} J(\mathbf{w}, b, \xi_i^+, \xi_i^-, \alpha_i^+, \alpha_i^-, \lambda_i^+, \lambda_i^-) &= \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}(\xi_i^+ + \xi_i^-) \\ &\quad - \sum_{i=1}^{N}\alpha_i^-[f(\mathbf{x}_i)\mathbf{w} + b - y_i + \varepsilon + \xi_i^-] \\ &\quad - \sum_{i=1}^{N}\alpha_i^+[y_i - f(\mathbf{x}_i)\mathbf{w} - b + \varepsilon + \xi_i^+] \\ &\quad - \sum_{i=1}^{N}(\lambda_i^+\xi_i^+ + \lambda_i^-\xi_i^-) \end{aligned} \tag{2.35}$$

where $\alpha_i^+, \alpha_i^-, \lambda_i^+, \lambda_i^-$ are the Lagrangian multipliers. It follows that the partial derivatives of J with respect to the primal variables $\mathbf{w}, b, xi_i^+, xi_i^-$ have to vanish for optimality.

$$\partial_b J = \textstyle\sum_{i=1}^{N}(\alpha_i^+ - \alpha_i^-) = 0 \tag{2.36a}$$

$$\partial_w J = w - \textstyle\sum_{i=1}^{N}(\alpha_i^- - \alpha_i^+)x_i = 0 \tag{2.36b}$$

$$\partial_{\xi_i^{(*)}} J = C - \alpha_i^{(*)} - \eta_i^{(*)} = 0 \tag{2.36c}$$

Substituting (2.36) into (2.35) yields the corresponding dual optimization problem:

$$\min_{\alpha_i^+, \alpha_i^-} \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i^- - \alpha_i^+)(\alpha_j^- - \alpha_j^+)f(\mathbf{x}_i)^T f(\mathbf{x}_j) - \sum_{i=1}^{N}(\alpha_i^- - \alpha_i^+)y_i + \varepsilon\sum_{i=1}^{N}(\alpha_i^- + \alpha_i^+) \tag{2.37}$$

subject to

$$0 \le \alpha_i^- \le C \tag{2.38a}$$

$$0 \le \alpha_i^+ \le C \tag{2.38b}$$

$$\sum_{i=1}^{N}(\alpha_i^- - \alpha_i^+) = 0 \tag{2.38c}$$

where $f(\mathbf{x}_i)^T f(\mathbf{x_j})$ is the inner-product kernel defined as Mercer's Theorem:

$$K(\mathbf{x}_i, \mathbf{x}_j) = f(\mathbf{x}_i)^T f(\mathbf{x_j}) \tag{2.39}$$

In deriving (2.37) and (2.38) we already eliminated the dual variables $\xi_i^{(*)}$, as these variables did not appear in the dual objective function anymore, but only were present in the dual feasibility conditions. Solving equation (2.37) with constraints (2.38) determines the Lagrange multipliers α_i^-, α_i^+ and (2.36b) can be written as follows:

$$\mathbf{w} = \sum_{i=1}^{N}(\alpha_i^- - \alpha_i^+)f(\mathbf{x}_i)^T \tag{2.40}$$

Note that $\mathbf{w}$ now can be completely described as a linear combination of the training patterns $\mathbf{x}_i$. In a sense, the complexity of a function's representation by support vectors is independent of the dimensionality of the input space and depends only on the number of support vectors. Moreover, the complete algorithm can be described in terms of inner products between the data. The approximation for new data x_{new} can be written as:

$$\begin{aligned} \hat{y}(\mathbf{x}_{new}) &= \mathbf{w}^T f(\mathbf{x}_{new}) + b \\ &= f(\mathbf{x}_{new}) \sum_{i=1}^{N} (\alpha_i^- - \alpha_i^+) f(\mathbf{x}_i)^T + b \\ &= \sum_{i=1}^{N} (\alpha_i^- - \alpha_i^+) K(\mathbf{x}_{new}, \mathbf{x}_i) + b \end{aligned} \tag{2.41}$$

The computation of b can be done by exploiting the so called Karush-Kuhn-Tucker(KKT)conditions. These state that at the optimal solution the product between dual variables and constraints has to vanish.

$$\begin{aligned} \alpha_i^-(f(\mathbf{x}_i)\mathbf{w} + b - y_i + \varepsilon + \xi_i^+) &= 0 \\ \alpha_i^+(y_i - f(\mathbf{x}_i)\mathbf{w} - b + \varepsilon + \xi_i^+) &= 0 \end{aligned} \tag{2.42}$$

and

$$\begin{aligned} (C - \alpha_i^-)\xi_i^- &= 0 \\ (C - \alpha_i^+)\xi_i^+ &= 0 \end{aligned} \tag{2.43}$$

From above conditions, we can conclude that only samples (x_i, y_i) with corresponding $\alpha_i^{(*)} = C$ lie outside the 'tube'. For $\alpha_i^+, \alpha_i^- \in (0, C)$ we have $\xi_i^+, \xi_i^- = 0$ and moreover the second factor in (2.42) has to vanish. Hence b can be computed as follows:

$$\begin{aligned} b &= y_i - \langle w, x_i \rangle - \varepsilon \qquad for \quad \alpha_i^- \in (0, C) \\ b &= y_i - \langle w, x_i \rangle + \varepsilon \qquad for \quad \alpha_i^+ \in (0, C) \end{aligned} \tag{2.44}$$

2.4.1 Discussion

The prediction for a new sample is a weighted sum of the inner-products of the training samples as given in (2.41). The weights are given by the Lagrangian multipliers. When these are zero, the inner-product does not have any influence on the prediction. This means such a training sample is superfluous for prediction. From KKT condition, only for $\mid \hat{y} - y \mid \geq \varepsilon$ the lagrangian multipliers may be nonzero, or in other words, for all samples inside the tube the lagrangian multipliers α_i^-, α_i^+ has to be zero such that the KKT conditions are satisfied[16]. Therefore the support vectors are points where exactly one of the Lagrange multipliers is greater than zero.

As it is illustrated in figure 2.6, due to the ε-insensitive loss function, those samples which lie outside the tube, are support vectors and used for prediction. Thus the number of support vectors depends on the width of the tube, which is determined by ε.

A smaller ε will lead to a smaller radius of the tube, a larger of a set of support vectors will be needed, thus a tougher job of the approximation, say noisy approximation. On the other hand, if the value of ε is too large, a broader insensitive zone will result and the some underlying relations will be cut off.

The approximation problem (2.33) contains two goals. The linear term $\xi_i^- + \xi_i^+$ represents the error made in not fitting some of the data exactly. The quadratic term $\frac{1}{2} \parallel w \parallel^2$ is a regularization term, and is present to help avoid overfitting. A large C will force the optimisation process to minimise the slack variables. Since these two terms cannot typically both be simultaneously minimized, the parameter C indicates how much emphasis is to be placed on one goal versus the other.

It can be said that both parameters ε and C affect the result of approximation and also the number of support vectors.

2.5 Least Square Support Vector Machine

Different from the SVM, the Least Square Support Vector Machine (LSSVM), developed by Suykens et.[17], uses a quadratic criterion instead of an ε-insensitive cost function. Then the solution of the optimisation problem can be formulated as:

$$\min_{\mathbf{w},\mathbf{e}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \lambda\frac{1}{2}\mathbf{e}^T\mathbf{e} \tag{2.45}$$

or

$$\min_{\mathbf{w},\mathbf{e},b} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \lambda\frac{1}{2}\sum_{i=1}^{N}(y_i - (\mathbf{w}^T f(\mathbf{x}_i) + b)) \tag{2.46}$$

subject to

$$\varepsilon_i - (y_i - (\mathbf{w}^T f(\mathbf{x}_i) + b)) = 0 \tag{2.47}$$

where ε_i is the error between the estimated value $\hat{y}_i$ and the real value y_i and λ is a constant. The Lagrangian of the optimization problem can be formulated as

$$J(\mathbf{w}, e, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \lambda\frac{1}{2}\sum_{i=1}^{N} e_i^2 - \sum_{i=1}^{N}\alpha_i[f(\mathbf{x}_i)\mathbf{w} + b + \varepsilon_i - y_i] \tag{2.48}$$

where α_i is the Lagrange multiplier.

Differentiation of the Lagrangian with respect to $\mathbf{w}, e, b, \alpha$ yields the conditions for optimality. Setting the derivatives equal to zero and eliminating $\mathbf{w}$ and e leads to a solution expressed solely in terms of inner products, which results in a linear system:

$$\begin{pmatrix} 0 & bf1 \\ \mathbf{1} & f(\mathbf{x}_i)^T f(\mathbf{x}_j) + \frac{1}{\lambda} \end{pmatrix} \begin{pmatrix} b \\ \alpha_i \end{pmatrix} = \begin{pmatrix} 0 \\ y_i \end{pmatrix} \tag{2.49}$$

This optimisation result can also be written as:

$$\begin{pmatrix} 0 & \mathbf{1}^T \\ \mathbf{1} & K + \lambda^{-1} I \end{pmatrix} \begin{pmatrix} b \\ \alpha \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{y} \end{pmatrix} \tag{2.50}$$

where $\mathbf{1}$ is a column vector of ones, and $\mathbf{y}$ a vector with target values. The entries of the symmetric positive definite kernel matrix K equal $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$. The role of weight vector $\mathbf{w}$ in primal space is conveniently taken over by a directly related weight vector α in the dual space. Once the values of α and b are calculated the LSSVM approximator can be evaluated at any new point of $\mathbf{x}_{new}$ by:

$$\hat{y}(\mathbf{x}_{new}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}_{new}, \mathbf{x}_i) + b \tag{2.51}$$

2.5.1 Discussion

The LSSVM formulation solves a linear system in dual space under a least-squares cost function, in which, only one parameter λ needs to be set. λ determines how much the error is minimized, in other words, how accurate the approximation is going on. A too large value of the λ results a too smooth estimation, thus high weights will be penalized too much. On the contrast, a too small value of λ results in overfitting, thus high weights are not be penalized enough.

The SVM model solves a quadratic programming problem in dual space, obtaining a sparse solution. The sparseness is built-in due to the ε-insensitive cost function that outrules errors of points inside a 'tube' around the approximated function. In comparison to SVM, LSSVM only requires solving a linear system, but it lacks sparseness in the number of solution terms. Pruning can therefore be needed to remove those training samples that have little influence on the current approximation. The pruning mechanism for LSSVM omits those samples from the training set that do not largely influence the prediction.

2.6 Summary

In this chapter the characteristics of function approximators have been presented. Then some previous function approximators that will be compared with the KSM are introduced. The review of the algorithms from literatures are given and followed by a discussion of its performance for each method. In the next chapter, the KSM method and then a theoretical comparison will be treated.

Chapter 3

Key Sample Machine

The Key Sample Machine(KSM)is developed by Kruif from Least Squares regression[2]. The least squares approximator needs the indicator functions set beforehand. However, in many control applications the form of the underlying relation is unknown and the set of functions is difficult to decide upon. A general form can be selected, e.g. B-splines, but then the overfitting may occur. A further disadvantage is that a general set of functions requires a significant number of parameters, and therefore considerable memory.

In KSM, we don't use a set of indicator functions to come to an approximation, but to use (some of) the training samples themselves to represent the relation. This approach is also used by the dual least squares and the Support Vector Machines.

The KSM is based on the SVM methodology; both of them use a subset of the training set to represent the approximation. The number of key samples that is selected is based on the complexity of the underlying relation and the amount of noise, so as not to overfit the data.

The KSM has the advantages lie in combining the following ideas:

- Use the summed squared approximation error as cost function,
- Use the sample-based approach of SVM and LSSVM,
- Use all the samples for the training (data compression), but use some of the samples for the prediction,
- Use a subset selection scheme that fits the problem on hand to select the samples that are needed for the prediction.

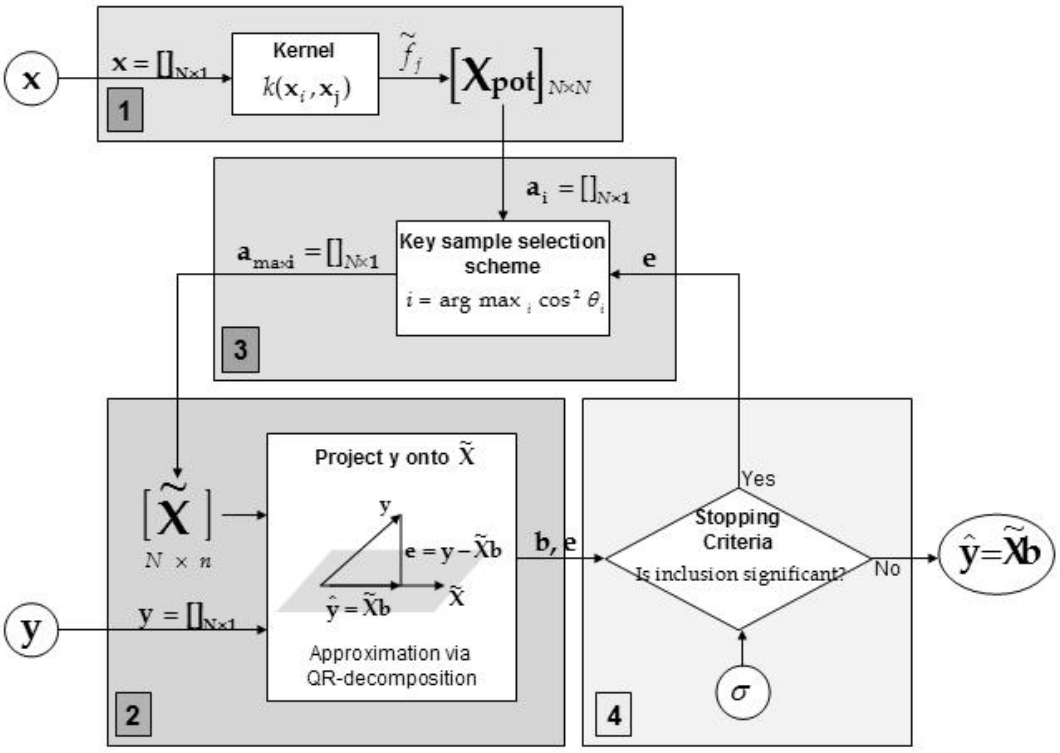


Figure 3.1: The KSM scheme

3.1 The KSM scheme

The KSM is a function approximator that adapts its structure based on the data presented. The scheme is illustrated by figure 3.1. In the scheme, there are four main blocks which execute different tasks. Before the first block, a training set should be given with N samples:$(\mathbf{x}_i, \mathbf{y}_i)_i = 1, \ldots, N$,

Figure 3.2 shows the training set of dataset1 (2.9) with noise level of 0.01. For simplicity, we use only $N = 50$ training samples. We consider a finite input space $\mathbf{x} = \{x_1, \ldots, x_N\}$, in which the x samples are equi-distributed. In our example, x_i are distributed at equal distances across the input range [0,1].

1. Apply a mapping of the input space to a kernel-induced feature space, which will be our working space for the linear approximation.

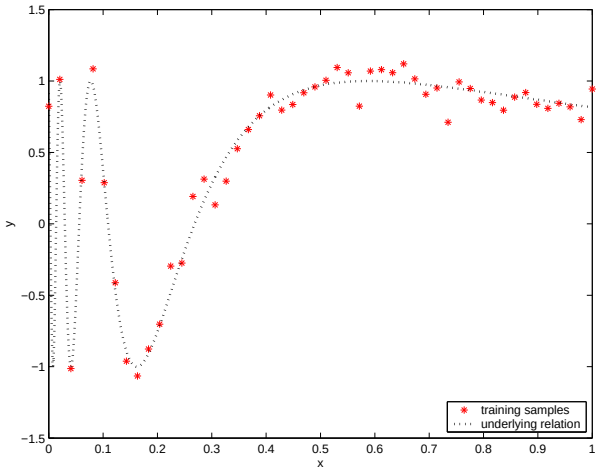


Figure 3.2: Training set with 50 samples

A kernel function needs to be selected that indicates how the approximation is interpolated between the key samples. Like the SVM, suppose the kernel function $K(x_i, x_j)$ is a symmetric positive definite function on x_i and x_j. This means the kernel function should imply a fair relation between any two samples.

The kernel function for a certain sample can be interpreted as a dual indicator function for the ordinary least squares scheme. For sample i, with input x_i, a dual indicator function can be defined as:

$$f_i(x) = K(x, x_i) = \langle f(x), f(x_i) \rangle \tag{3.1}$$

The kernel represents a legitimate inner product in feature space. More generally, the kernel function can be chosen from the following possible forms[18]:

Linear: $K(x_i, x_j) = \langle x_i, x_j \rangle$

Polynomial: $K(x_i, x_j) = (\langle x_i, x_j \rangle + \mathbf{1})^d$

Gaussian Radial Basis Function: $K(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$

Zero-order spline: $K(x_i, x_j) = \mathbf{1} + \min(x_i, x_j)$

First-order spline: $K(x_i, x_j) = \mathbf{1} + x_i x_j + \frac{1}{2}\min(x_i, x_j) - \frac{1}{6}\min(x_i, x_j)$

Additive Kernels: $K(x_i, x_j) = \sum_k K_k(x_i, x_j)$

Tensor Product: $K(x_i, x_j) = \prod_k K_k(x_i, x_j)$

More complicated kernels can be obtained by forming summing kernels or forming tensor products of kernels.

For example, we can define a kernel function use zero-order spline as

$$k(x, x_{ks}) = \mathbf{1} + \min(x, x_{ks}) \tag{3.2}$$

This results a piecewise linear approximation. All the elements in input space $\mathbf{x} = \{x_1, \ldots, x_N\}$ will be use to construct a potential key sample matrix in such ways, denoted as $\mathbf{X}_{pot}$, with the size of $N \times N$. Every column of $\mathbf{X}_{pot}$ indicates a possible key sample.

$$\mathbf{X}_{pot\ i,j} = k(x_i, x_j), \quad i, j = 1, \ldots, N \tag{3.3}$$

In our case $N = 50$, we obtain

$$\mathbf{X}_{pot} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & \\ 1 & 1.0204 & 1.0204 & 1.0204 & 1.0204 & \\ 1 & 1.0204 & 1.0408 & 1.0408 & 1.0408 & \\ 1 & 1.0204 & 1.0408 & 1.0612 & 1.0612 & \\ 1 & 1.0204 & 1.0408 & 1.0612 & 1.0816 & \\ 1 & 1.0204 & 1.0408 & 1.0612 & 1.0816 & \\ & & & & & \ddots \end{pmatrix}_{50\times 50} \tag{3.4}$$

We can see that the kernel matrix (or $\mathbf{X}_{pot}$) is symmetric and column independent. And every column of $\mathbf{X}_{pot}$ is a potential key sample to be compared in the subset selection scheme of part **3**.

2. Approximation via QR decomposition

Before starting this block, we need to initialize an indicator matrix named $\mathbf{X}$, with the size $N \times n$, where n is the number of key samples (also the number of the columns of $\mathbf{X}$. Obviously, when we adopt all N potential samples as key samples, this indicator matrix $\mathbf{X}$ will be same as $\mathbf{X}_{pot}$. Because a forward selection scheme is used, at the beginning of the approximation there is not yet a prediction, and therefore the indicator matrix $\mathbf{X}$ is set to be a column vector. As the key samples are added, $\mathbf{X}$ increases its size (columns). In other words, we need to solve an approximation problem of y in the space spanned by the columns of $\mathbf{X}$.

The KSM use quadratic cost function. The problem can be formulated as solving

$$\min_b \| \mathbf{y} - \mathbf{X}\mathbf{b} \|_2^2 \qquad \mathbf{X} \in \mathbb{R}^{\mathbf{N}\times\mathbf{n}}, \quad N > n \tag{3.5}$$

where N is the number of training samples and n is the number of selected key samples, used for prediction. The problem can be looked as project the output $\mathbf{y}$ onto the column space of $\mathbf{X}$, with the projection P. In order to find a best estimation of $\mathbf{y}$, this leads to the so-called normal equation:

$$\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{y} \tag{3.6}$$

Since $\mathbf{X}$ is assumed to have full column rank, $\mathbf{X}^T\mathbf{X}$ is nonsingular and symmetric positive definite. Hence the solution b to the problem (3.3) is uniquely determined and can be expressed as

$$\mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{3.7}$$

It is well known that the accuracy of the computed normal equations solution (3.5) is not easy to guarantee. Hence the normal equation approach could introduce errors that are greater in magnitude than what we would expect with a stable algorithm. In many applications where accuracy and stability are important methods which are based on the QR decomposition are usually preferred.

Assume that $\mathbf{X} \in \mathbb{R}^{\mathbf{N}\times\mathbf{n}}$ with $N > n$ and $\mathbf{y} \in \mathbb{R}^{\mathbf{N}}$ are given and an orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{\mathbf{N}\times\mathbf{n}}$ has been computed and applied to $\mathbf{X}$ and such that

$$\mathbf{Q}^T\mathbf{X} = \begin{pmatrix}\mathbf{R}\\ 0\end{pmatrix} \begin{matrix} n\times\mathbf{1}\\ (N\times n)\times\mathbf{1}\end{matrix} \tag{3.8}$$

where $\mathbf{R}$ is upper triangular and nonsingular, and

$$\mathbf{Q}^T\mathbf{y} = \begin{pmatrix}\mathbf{r}\\ \rho\end{pmatrix} \begin{matrix} n\times\mathbf{1}\\ (N\times n)\times\mathbf{1}\end{matrix} \tag{3.9}$$

Then

$$\begin{aligned} \| \mathbf{y} - \mathbf{X}\mathbf{b} \|_2^2 &= \| \mathbf{Q}^T\mathbf{y} - \mathbf{Q}^T\mathbf{X}\mathbf{b} \|_2^2 \\ &= \begin{pmatrix}\mathbf{r}\\ \rho\end{pmatrix} - \begin{pmatrix}\mathbf{R}\\ 0\end{pmatrix}\mathbf{b} \\ &= \| \mathbf{r} - \mathbf{R}\mathbf{b} \|_2^2 + \| \rho \|_2^2 \end{aligned} \tag{3.10}$$

Note that $\| \rho \|_2^2$ is the minimal residual. Then the problem can be readily solved with $\mathbf{r} = \mathbf{R}\mathbf{b}$ once we have computed the QR factorization of $\mathbf{X}$.

From another point of view $\mathbf{R}$ and can be equivalently computed by finding the Cholesky factorization of the augmented matrix $\mathbf{X}_a^T\mathbf{X}_a = (\mathbf{X}\ \ \mathbf{y})^T(\mathbf{X}\ \ \mathbf{y})$. Since

$$(\mathbf{X}\ \ \mathbf{y})^T(\mathbf{X}\ \ \mathbf{y}) = \begin{pmatrix}\mathbf{X}^T\mathbf{X} & \mathbf{X}^T\mathbf{y}\\ \mathbf{y}^T\mathbf{X} & \mathbf{y}^T\mathbf{y}\end{pmatrix} = \begin{pmatrix}\mathbf{R}^T\mathbf{R} & \mathbf{R}^T\mathbf{r}\\ \mathbf{r}^T\mathbf{R} & \mathbf{r}^T\mathbf{r}+\rho^2\end{pmatrix} = \begin{pmatrix}\mathbf{R} & \mathbf{r}\\ 0 & \rho\end{pmatrix}^T \begin{pmatrix}\mathbf{R} & \mathbf{r}\\ 0 & \rho\end{pmatrix} \tag{3.11}$$

where $\rho^2 = \| \mathbf{y} \|_2^2 - \| \mathbf{r} \|_2^2$ and $\mathbf{R}_a = \begin{pmatrix}\mathbf{R} & \mathbf{r}\\ 0 & \rho\end{pmatrix}$ is the Cholesky factor of $\mathbf{X}_a^T\mathbf{X}_a$. By applying the Cholesky factorization we don't need to calculate the matrix Q and then the memory space can be saved. Once we get the $\mathbf{R}$ and $\mathbf{r}$, from which the parameter vector $\mathbf{b}$ can be computed. Because $\mathbf{R}$ is upper triangular matrix, the calculation for $\mathbf{b}$ is not excessive.

3. Subset selection scheme: finding potential indicator with largest correlation to current residual.

The KSM use all sample data for training, but use some data for prediction. Once add one key sample to the indicator matrix $\mathbf{X}$. Because $\mathbf{X}$ is updated after inclusion a new key sample, we use the notation $\widetilde{\mathbf{X}}$. Now we want the added the indicator, which decreases the summed squared approximation error most.

We know the residual vector e, also of dimension N, is perpendicular to the column space of $\mathbf{X}$. If we can find a vector that is linear related to the residual vector, the inclusion of that indicator vector would make the residual zero. However, this kind of vector is outside the column space of $\mathbf{X}$, this means the presence of such an indicator is not likely. The solution is to find an indicator from the potential indicator vectors that will generate a column vector in $\mathbf{X}$ that makes the smallest absolute angle with the residual vector e.

The subset selection scheme contains the steps as:

1. Calculate angle θ_i between the residual $\mathbf{e} = \widetilde{\mathbf{X}}\mathbf{b} - \mathbf{y}$, and the potential columns $\mathbf{a}_i$.
2. Find the best key samples: $i = arg\max_i \cos^2\theta_i$, Where
$$cos^2\theta_i = \frac{(\mathbf{e}^T\mathbf{a}_i)^2}{\mathbf{a}_i)^T\mathbf{a}_i(\mathbf{e}^T\mathbf{e})} \tag{3.12}$$
.
3. Include the new key sample a_{maxi} into the set thus leads to an updated indicator vector$\widetilde{\mathbf{X}}$.

Based on the Cholesky factorization, the augmented indicator matrix is set as follows
$$\mathbf{X}_a = [\mathbf{X} \;\; \mathbf{y}], \quad \mathbf{R}_a = \begin{pmatrix} \mathbf{R} & \mathbf{r} \\ 0 & \rho \end{pmatrix}_{(n+1)\times(n+1)} \tag{3.13}$$

At the beginning, the $\mathbf{X}$ is set to be a column vector with the elements of a bias is set to $+1$.
$$\mathbf{X}_a = [\mathbf{1} \;\; \mathbf{y}], \quad \mathbf{R}_a = \begin{pmatrix} 7.0711 & 3.8792 \\ 0 & 4.3266 \end{pmatrix}$$

When there is a new key sample induced to the indicator function, the updated matrices are partitioned as
$$\bar{\mathbf{X}}_a = [\mathbf{X} \;\; \mathbf{x} \;\; \mathbf{y}], \quad \bar{\mathbf{R}}_a = \begin{pmatrix} \bar{\mathbf{R}} & \mathbf{g} & \bar{\mathbf{r}} \\ 0 & \gamma & \bar{\rho} \\ 0 & 0 & \tau \end{pmatrix} \tag{3.14}$$

After the update, the relation $\bar{\mathbf{X}}_a^T \bar{\mathbf{X}}_a = \bar{\mathbf{R}}_a^T \bar{\mathbf{R}}_a$ still has to hold. Thus we have:

$$\bar{\mathbf{R}} = \mathbf{R} \quad (3.15a) \qquad \gamma^2 = \mathbf{x}^T\mathbf{x} - \mathbf{g}^T\mathbf{g} \quad (3.15d)$$
$$\mathbf{g} = \mathbf{R}^{-1}\mathbf{X}^T\mathbf{x} \quad (3.15b) \qquad \bar{\rho} = \frac{1}{\gamma}\mathbf{x}^T\mathbf{y} - \mathbf{g}^T\mathbf{r} \quad (3.15e)$$
$$\bar{\mathbf{r}} = \mathbf{r} \quad (3.15c) \qquad \tau^2 = \mathbf{y}^T\mathbf{y} - \mathbf{r}^T\mathbf{r} - \bar{\rho}^2 \quad (3.15f)$$

In our example we chose the last column of the $\mathbf{X_{pot}}$ to be a new key sample and include it in the initialized indicator matrix $\mathbf{X_a}$, the updated matrix become

$$\bar{\mathbf{X}}_a = \begin{pmatrix} 1.0000 & 1.0000 & 0.7212 \\ 1.0000 & 1.0204 & 1.0448 \\ 1.0000 & 1.0408 & -0.8724 \\ 1.0000 & 1.0612 & 0.4843 \\ 1.0000 & 1.0816 & 1.1052 \\ & \cdots & \end{pmatrix}_{100\times(2+1)}, \bar{\mathbf{R}}_a = \begin{pmatrix} 7.0711 & 10.6037 & 3.8792 \\ 0 & 2.0777 & 2.4842 \\ 0 & 0 & 3.5424 \end{pmatrix}$$

And then we have obtained a new indicator vector $\bar{\mathbf{X}}_a = [\mathbf{X} \;\; \mathbf{y}]$, which is ready for the next selection.

4. Stopping criteria

When the best potential indicator has been identified, it should be decided whether it is good enough to be included for the prediction or that the inclusion of more key samples should stop. The inclusion of the candidate as an actual indicator depends on whether adding more indicators gives a significant increase to the performance or that overfitting occurs. A statistical criterion is proposed that keeps including superfluous samples.

Define an extra sum of squares S^2, which is the difference between the summed squared approximation errors of all samples before and after the inclusion of the candidate key sample. For additive Gaussian noise on y, S^2/σ^2 is chi-square distributed X_1^2 if and only if the indicator weight bi is zero.

$$S^2/\sigma^2 \sim X_1^2 \Leftrightarrow b_i = 0$$

From the X_1^2 distributed table, we notice that if the probability for a certain error reduction is too small, then the new parameter bi is unlikely to be zero, therefore the inclusion continues. In other words, if a certain error reduction is significant, the probability for $b_i = 0$ is small, than then indicator is included. This can be formulated as following:

$$P(\frac{S^2}{\sigma^2} \geq z_\zeta \mid b_i = 0) < \zeta \quad (3.16)$$

As long as the ratio is large than z_ζ, the probability for the error reduction is smaller than some significant value ζ, which means the inclusion is significant. So the

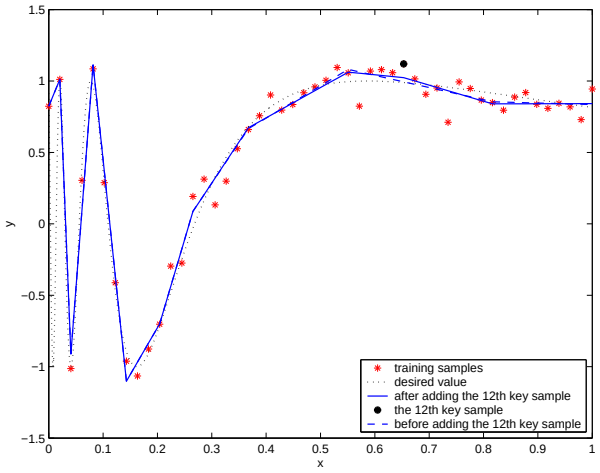


Figure 3.3: Stopping criteria

condition on which the inclusion should stop is that the S^2/σ^2 is smaller enough, thus the probability for $b_i = 0$ increases to some significant level, which means the inclusion has no influence on the approximation.

However, the noise level is often unknown for real-life problems and has to be estimated. Thus the noise variance can be looked as a design parameter which is tuned to find a reasonable significant level. After the significant level is selected, it is fixed and not considered an extra parameter.

In our example, the noise variance is 0.01. Figure 3.3 show after adding the 11^th key sample, the summed squared error change is 2.6662. The inclusion continued, which stopped after adding the 12^th key sample, the summed squared error change is 0.4866.

3.2 Theoretical Comparison of KSM with other Approximators

In chapter 2 we have already seen that for nonlinear approximation problem, the ordinary LS method could not give a satisfied solution. While the neural networks are often used in approximation problem and showed to be a powerful technique. Moreover, recently developed kernel-based method like SVM and KSM are remarkable due to their specific techniques which enable operations performed in the input space. In this section, a comparison in a theoretical view will performed. In order to make a systematic comparison, the order of comparison will be made similar with that of the method introduced in previous chapter. At last the KSM will be compared to each method previously introduced in this thesis.

3.2.1 Radial-Basis Function Networks

When use Gaussian function as indicator functions in LS approximation, it looks a bit like the RBFN. Actually it can be looked as a special case of RBFN when Gaussian functions are use in the hidden layer with fixed centers and widths. There are no adaptations for the locations and widths of Gaussian functions. Thus large number of RBFs is needed, which are often leads to overfitting. That is why in LS the fit is too rough. The RBFN works by calculating the distance between the input data and the center of the RBF. Training process involves determining three different parameters: the position of the centers in Gaussian kernel, their variances (width), and the weight. The nearer the input with the center, the closer the output of the RBF to the bias. When one input is very close to one of the neuron centers, its output dominates the output of the network while the output of other neurons is much smaller. This means RBF constructs a local approximation for each neuron.

3.2.2 Support Vector Machines

The learning in the SVM is developed based on statistical learning theory. The advantage of the SVM over RBFN lies in the training process. The SVM uses only a limited amount of the training samples for the prediction of the data. The minimisation of the cost function is a convex function where global minima are guaranteed. When use the gradient descent optimisation and all the parameters of the RBF are trained as well, a long training time and local minima is likely to occur.

The idea of the kernel function is to enable operations to be performed in the input space rather than the potentially high dimensional feature space. Hence the inner product does not need to be evaluated in the feature space. Thus it is less prone to the curse of dimensionality than the RBFN.

In SVM, both parameters ε and C, which affect the result of approximation and also the number of support vectors, are needed to be tuning to balance the trade-off between the number of support vectors and the computational efficiency. In RBFN, the centres are updated during training. The generalisation is dependent on how the neurons cover the input space. There is also a balancing between the number of RBFs and their widths such that optimal training and generalisation performance are obtained.

3.2.3 Least Squares Support Vector Machines (LSSVM, LSSVM+)

The LSSVM has the majority of the features of the SVM. Both of them build a linear model in the so-called feature space where the inputs have been transformed by means of a nonlinear mapping. This is converted to the dual space by means of the Mercer's theorem and the use of a positive definite kernel, without computing explicitly the mapping. Thus the LSSVM is also less prone to the curse of dimensionality than the

RBFN.

The difference between them is that the SVM model solves a quadratic programming problem and the use of the -insensitive cost function gives rise to undesirable behaviour. In LSSVM a quadratic cost function is used instead, and the optimization problem reduces to solving a linear set of equations. But at the same time the sparseness is lost.

The LSSVM+ is the LSSVM with a pruning mechanism. Pruning is closely related to subset selection, choosing relevant data points or variables in order to build a sparse model. The point is omitted that has the smallest support value.

Besides, both of the RBFN and LSSVM use a set of simultaneous linear equations to calculate the weight parameters, w in the RBFN and α in LSSVM. In LSSVM, only one parameter needs to be set.

3.2.4 Key Sample Machine

The KSM is developed from regression theory and statistics and represents the data by a subset of the data. This approach is similar to the SVM.

The KSM use a limited sample of the training set to represent the full training set. This is also used in the SVM. The interpolation between the key samples is done by a kernel function like in SVM. Because of the kernel-based approach, it is free to choose a kernel function, the class of functions that can be use for approximation is not limited to one class.

In contrast to SVM, the KSM uses a quadratic cost function. The LSSVM+, which also uses a quadratic cost function without a parse solution. the pruning scheme successively removing those samples from the training set that have little influence on the prediction. However, omission of samples from the training set may removes information on the underlying relation.

While the KSM also use a subset selection scheme to find the key samples, explicitly forward selection scheme is used. This scheme includes one key sample a time, until a good enough approximation is found. Thus the calculation time required is saved. The forward selection scheme can also be used in RBFN by adding one neuron at a time.

In KSM, the only parameter is the noise variance of the training data. The computation of the weight is similar to the ways in LS. But the use of QR decomposition provides numerically stable and furthermore the Cholesky factorization simplifies the calculation of the weights.

3.3 Summary

This chapter focuses on the KSM algorithm. Then it is compared with other previous function approximators introduced in Chapter 2. A theoretical review of these methods based on benchmarking point of view has been presented, such as the similarities and differences between KSM and other methods are drawn from their basic working principles. Correspondingly the empirical comparison in several aspects according to the requirements of function approximator will be done in the next chapter.

Chapter 4

Benchmarking Simulation Setup

In this chapter, the Key Sample Machine is benchmarked against other methods, namely the neural network RBFN, the kernel based SVM and LSSVM. Our primary focus is to evaluate their suitability as a function approximator embedded in learning control setting, especially in Learning Feed Forward Control(LFFC). The KSM, embedded in the feedforward controller, learns the feedforward signal as a function of the desired states to compensate for the state dependent effects.

We compare KSM to the other algorithms, which were introduced in the second chapter, by means of standard performance measures (mean squared error, standard deviation, training time). The comparison will be performed on real and artificial benchmarking problems.

4.1 Benchmark Methodology

Most papers present performance results for a new learning algorithm only for a small number of problems - rarely more than three[4]. In most case, one or several of these problems are meaningless synthetic problems and comparisons to algorithms suggested by others are not done. This is often true because training a function approximator or a neural network usually takes quite long, hence a thorough evaluation takes a very large amount of CPU time. It is also much work to prepare data for training. Besides, even results obtained for the same problem can often not be compared directly because of different problem representations or different experiment setups. Often experiment setups are not documented properly in the papers published, making reproduction or exact comparison impossible. Even documented ones can be obscure because the same things are expressed in very different ways by different people.

As we see from above, there is a need for using standard sets of problems and rules for applying them to be used in benchmarking. It is also important to express

detailed experiment setups as well as their documentation. The following aspects guide the benchmarking rules:

- **Benchmark problems:** there is a need for using standard sets of problems which are publicly accessible and normally used by other researchers. Both real and artificial problems should be used.
- **Validity:** We need a minimum standard of experiments that guarantees that the results obtained are valid in the sense that they are not artifacts created by random factors or by a faulty experimental setup.
- **Statistical test:** All experiments should be performed in a statistical framework. Resampling techniques enable it to estimate the performance of method in a fair way.
- **Reproducibility:** All the aspects of the experimental setup are needed for other researchers to repeat the experiments.
- **Comparability:** Compare results among several methods and represent them in reasonable ways.

4.2 Benchmark Problems

In this study, standard benchmark data sets are used to test KSM as a function approximator. According to the above benchmark rules, the data sets should be selected from public benchmark data bases for general machine learning programs and be appropriate to be used in simulation of a KSM as a function approximator in a control setting.

4.2.1 Criteria of data set selection

During control, the input-output relation of the function approximator is adapted such that it learns the inverse of the plant, which is based on samples obtained during the control of the plant and is often difficult to model accurately in practice. Thus a feedforward signal is created, which nullifies the state dependent effects and makes the system output converge to the desired trajectory, based on the reference. In other words, the inputs of the feed forward controller (or function approximator) depend on the plant and thus can be the reference of the control system. For example in motion system, the inputs of the KSM consist of the reference position, and its derivatives (see Figure 2.2).

Since the input-output relation of a function approximator is actually a mapping from the system states (output and if necessary its derivatives or integrals) to the feedfoward signal, which compensates the non-linear state dependent effects, there are

several criteria for selecting the data set (input/output) of the function approximator in a simulation test:

- **Well-known benchmark problem:** The data sets should be standard machine learning benchmark problem from the public data base and used in similar researches, since the benchmark should be conducted in a convincing and reproducible way.
- **Input dimension (number of attributes):** Since in practice, we only consider multiple inputs (references) and single output (steering signal) cases, the data set should have n inputs attributes, n should range from 3 to 12. For example, a motion system commonly has a minimum input dimension of three.
- **Attribute type:** The data should be real-valued. As shown above, the inputs and output of a function approximator are reference derivatives thereof and the steering signal respectively in control, nominal or binary-valued data is avoided. A data set that contains missing attribute values is not considered either.
- **Dataset size (number of examples):** Although a large number of samples offers detailed information about the underlying relations of the inputs and output, it takes a very large amount of CPU time. As explained in Chapter 2.1, because of the small memory requirement, the data set should not be too large. On the other hand, too small number of samples is also not desirable. Thus the size between 200 to 10000 is considered.
- **Data complexity:** A high noise level contained in the data are less desirable (not necessarily). Considering the memory limitation, the problem containing complex data due to high noise or non-linearly, needs a very long time for training, thus will not be used.

4.2.2 Benchmark data sets selection

The most famous public benchmark database is the UCI Machine Learning Repository[19]. Another often referred database is contained in DELVE project(Data for Evaluating Learning in Valid Experiments)[20]. In addition, there is a repository of data sets collected mainly used for the experimental analysis of function approximation techniques, the Bilkent University Function Approximation Repository[21]. There are also useful source of data sets assembled in the R package `mlbench` (machine learning benchmark problems), which can be found in CRAN[22]. Another source of data sets is StatLib Data sets Archive[23].

The data sets used should be standard in benchmarking. As Prechelt[3] suggests, we also use artificial data. The well-known Friedman problems 1[24],2[25] and 3[26] are used. The data source is assembled using the R package `mlbench`. In this thesis all of

the data sets used originate from the above sources. Some characteristics of well known benchmark data sets for regression are listed in Table 4.1.

Problem	Size	Attributes(inputs)				Missing	Target	Source
		b	c	r	tot.	values	feature	
abalone	4177		1*	7	8	none	rings	UCI
autompg	398		3*	4	7	6	gas consumption(mpg)	UCI
autos	205	4	6*	15	25	59	price of car	UCI
cpu	209			6*	6	none	performance(PRP)	UCI
cpuSmall	8192			12	12	none	usr	DELVE
Census(8L)	22784			8	8	none	price	DELVE
Housing	506	1		12	13	none	house prices (MEDV)	UCI
Kinematics	8192			8	8	none	y	DELVE
Quake	2178			3	3	none	Richter	Bilkent
Servo	167		4		4	none	Rise time(class)	UCI
Stock Prices	950			9	9	none	stock price(comp10)	StatLib
Friedman1	1000/5000			5	5	none		CRAN
Friedman2	1000/5000			4	4	none		CRAN
Friedman2	1000/5000			4	4	none		CRAN

The upper part describes real data, the lower part synthetic data sets.
Legend: b=binary, c=categorical(nominal), r=real, *=integer

Table 4.1: Well known benchmark data sets for regression

We will choose some problems from the above data sets for our benchmark study. Since all data sets listed in Table 4.1 are contained in well known benchmark data sets, the other criteria are considered one by one and the violator is excluded. The selection is performed as follows and the results is showed in Table 4.2.

Input dimension The "autos" and "Housing" exceed 12 inputs. 25 inputs are not likely in control, so we leave the "autos" out. The "Housing" contains 12 numerical attributes and one binary categorical attribute. Since the binary one is often not used in regression problems, we keep the this data set and disregard the binary input.

Attribute type There are three data sets ("abalone","autompg","Servo") containing categorical attributes. "abalone" contains one nominal attribute (the sex of abalone: M, F, and I (infant) which can be encoded into integers of 0,1 and 2 respectively. The problem "autompg" has 3 categorical attributes, which are already in form of integers in the data set, and it also contains 6 missing values, we

leave it out. All of the four attributes of "Servo" are nominal ones, which cannot readily be learned by training and thus this problem is also left out.

Dataset size We leave out the "Census" because of the large number of samples for training.

Data complexity The largest data complexity is present in the "Kinematics", which is concerned with the forward kinematics of an 8 link robot arm. Among the existing variants of this data set we have used the variant 8nm, which is known to be highly non-linear and medium noisy. It is found in a simulation test that the training time is very long. So this data is left out. For artificial data sets Friedman 2,3, the standard deviations used were 1.125 and 0.1, respectively(these two values yield a 3 : 1 signal to noise ratio as proposed by the author). Thus the variance of the function itself (without noise) accounts for 90% of the total variance. The noise level is a little high, but considering the training time, they are still acceptable.

	Input dimension	Attribute type	Dataset size	Data complexity
abalone	×	×	×	×
autompg	×			
autos				
cpu	×	×	×	×
cpuSmall	×	×	×	×
Census(8L)	×	×		
Housing	×	×	×	×
Kinematics	×	×	×	
Quake	×	×	×	×
Servo	×			
Stock Prices	×	×	×	×
Friedman1	×	×	×	×
Friedman2	×	×	×	×
Friedman2	×	×	×	×

×=selecting a data set

Table 4.2: Data sets selection

Finally we have selected six real-world data sets and three artificial data sets for our benchmarking simulation. For particular information about these data sets, see Appendix A.

4.3 Simulation Procedure

In this section, the detailed simulation procedure is documented for reproducibility. After selecting the benchmark problems, the pretreatment of the data should be done. In order to obtain valid results, for every problem several data sets should be produced.

4.3.1 Data sets preparation

We use the holdout method, in which the data used for performing benchmarks on function approximators must be split into two parts: one part on which the training is performed, called *training set*, and another part on which the performance of the resulting prediction is measured, called the *test set*. No information about the test set examples or the test set performance must be available during the training process; otherwise the benchmark is invalid.

It is necessary to compute multiple runs of the experiment in order to avoid biased estimation. In our simulation, We perform the rotational runs on the data known as the 10-fold cross-validation. For each problem, we conduct 100 simulations. As suggested by Leisch[27], for the k-fold cross-validation, we need to train N/k times. Thus the number of training is a constant in our case, and we use 10 times repeated 10-fold cross validation. As a whole, for each problem, we need to conduct 100 experiments. In other words, 100 training sets and 100 test sets are produced.

Other than preparing the training/test sets for benchmarking, another special set of data is needed in a pre-experiment for tuning the parameters of the function approximator beforehand. To produce this special data sets, the training data is further divided into one *tuning set* and one *validation set*.

Figure 4.1 illustrates the scheme of the data sets preparation. We have two step in the scheme: Step A prepares the data sets for benchmarking; step B prepares the data set for parameter tuning. The procedure is as follows:

Step A: Generate 100 data set pairs for benchmarking

Real data:

A1. Divide the whole data into 10 partitions.

A2. Take 9/10 of partitions as training set and the rest 1/10 as test set. Totally 10 different traning/test pairs are created.

A3. Repeat A1 (in a different ways of partition) and A2, 100 traning/test sets are produced.

Artificial data:

A1. Generate a training set of 1000 samples and a test set of 5000 samples.

A2. Repeat A1 100times to produce 100 train/test sets.

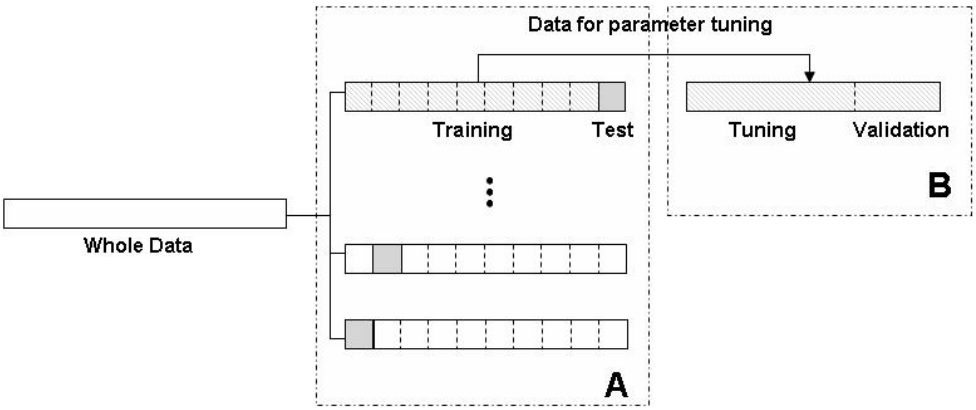


Figure 4.1: Data sets preparation(step A only for the real data)

Step B: Generate one special data set pair for parameter tuning

B1. Take one training data generated in procedure A.
B2. Divide this training data into one *tuning set* (2/3) and one *validation set* (1/3).

4.3.2 Parameters tuning

During parameter tuning, a grid search over the different parameter spaces is needed. Table 4.3 shows the parameters of each method. For the tuning results, see appendix B.

- For KSM, if the linear splines kernel is used, there is only one parameter: the noise level μ ,to be tuned; if the RBF-kernel is used, the other parameter related to the radials of RBFs, σ, is needed. A linear grid search is performed, in which μ ranges from 10^{-4} to 10^{0} and σ from 10^{-2} to 10^{2}.

- For SVM, a linear grid search over the 3-dimensional parameter space (C, ε, g) is done. C ranges from 10^{0} to 10^{6}, ε from 10^{-4} to 10^{0} and the RBF kernel parameter g ranges from 10^{-2} to 10^{2} -by training on 2/3 of the training set and choosing the one minimizing the MSE on the remaining 1/3 (validation)set.

- For LSSVM, a linear grid search over the 2-dimensional parameter space $(\lambda, sig2(\sigma^2))$. λ ranges from 0 to 100 and $sig2$ from 10^{-2} to 10^{2}. The λ is the regularization parameter, determining the trade-off between the fitting error minimization and smoothness of the estimated function. σ^2 is the RBF kernel function parameter.

- For RBFN, the centres and the radii are optimized by forward selection during the training process. But the scales applied to the radii are tuned by searching from a range of [0,100], and let the program choose the best one with the lowest model selection score(MSE).

Method	Parameters
KSM(linear)	μ
KSM(rbf)	μ, σ
SVM	C, ε, g
LSSVM	σ, λ
RBFN	scales

Table 4.3: Parameters need to be tuned

4.3.3 Data scaling

In our simulation, all data for training will be scaled within the range of [0,1]. The scaling ratios are kept for scaling the test/evaluation data in the same way. Unlike for the training data, here is no guarantee for all attributes of the test data to be strictly within the [0,1] range. Table 4.4 shows the average ranges of the data before and after scaling.

	Unscaled		Scaled training		Scaled test	
	min	max	min	max	min	max
abalone	1	29	0	1	0.0635	0.7453
cpu	6	1150	0	1	0	0.4
cpuSmall	0	99	0	1	0	0.9999
Housing	5	50	0	1	0.1641	0.8070
Quake	5.8	6.9	0	1	0	0.8087
Stock	34	62	0	1	0.2965	0.6361
Friedman1	0.664	28.59	0	1	-0.0431	1.0384
Friedman2	-264.9205	1813.9	0	1	-0.0319	1.0505
Friedman3	-0.1007	1.8375	0	1	-0.0423	1.0237

Table 4.4: Output range before/after scaling

4.3.4 Simulation setup

All experiments are conducted on a machine containing a 1.4 GHz Pentium M processor with 256Mb of memory. We ensured that all experiments were isolated, i.e,, no other users were using the machine at the same time. The software implementations of all the function approximators are as follows:

Key Sample Machine to test KSM, both linear splines kernel (version 1.0) and RBF-kernel are used in a C++ implementation developed at the University of Twente and at Imotec[28].

Support Vector Machines we use the C++ library LIBSVM (version 2.71) from Chang and Lin(2004)[29], which performed well on several occasions. Specifically, the ε-support vector regression with the radial basis function(RBF)kernel is used.

Least Squares Support Vector Machines we use the LS-SVMlab Toolbox(version 1.5),which contains Matlab/C implementations from LS-SVMlab (SISTA,KULeuven)[30].

Radial Based Functions Network we use the Matlab Functions for Radial Basis Function Networks (Version 2.2) from Mark Orr (2000)[31], which implements a variety of methods based on subset selection and ridge regression to control model complexity to generate RBF centres and radii.

4.4 Performance Measure

In this research, as performance measures we chose the following:

- **mean squared error(MSE)** The error is the average over the 10-fold cross validation and the average computation time is 10 times the average training time. Both the statistical mean and standard deviation of the MSE are reported.
- **Computation time** The CPU time spent for training is also a measure for evaluating the efficiency of the function approximators. The disadvantage of this measure is that it leaves one free factor: the efficiency of the software implementation. But the results are still valid for comparison because the CPU time for all methods are measured in the same ways, and normally the difference due to the efficiencies of different algorithms are obvious.
- **Number of support vectors** For the kernel-based methods,KSM,SVM,LSSVM, the number of support vectors(key samples in case of KSM)are also reported to evaluate the efficiency of function approximators. This number indicates the computation time need for approximation, which is very important for real-time control.

4.5 Summary

In this chapter, the simulation setup of benchmark is documented. First benchmark problems are selected which are properly in evaluation of function approximators. Then data sets are produced and processed for simulation. The whole experimental procedure has been described in next chapter, the results will be presented.

Chapter 5

Simulation Results and Analysis

In the previous chapter, the simulation setups are addressed. The benchmark of KSM will be done on well-known approximation problems. In this chapter, the simulation results will be presented and evaluated. Other than the simulation procedures depicted in the previous chapter, there are several things to be noticed:

- For each problem, 100 different data sets are produced by dividing the whole data into partitions and then combining parts of them. It is true that the training performances for these data sets can be diverse. But the statistical results is still valid because all methods share the same data sets. However, one or more extreme results can occur and affect the mean of errors, thus bringing bias. If the number of such outliers is less than 5% of number of runs of experiment(in our case 100), they can be neglected. In order to examine the result dispersion, more robust evaluation is needed.

- In scaling, the values of test data can not range strictly within the desired scaling range. As we said in previous chapter, this can be neglected since often the scaled test data only slightly exceeds the range. But it is noticed that scaling the data within the range of [0,1] leads to better performance than within the range of [-1,1].

- For the sake of a fair comparison, all methods use RBF kernels. Besides, a first order spline kernel is also tested for the KSM in order to investigate how its performance changes with respect to different kernel functions.

- Besides the mean of the test set MSE, we also use more robust median, complemented by the IQR as a dispersion measure. Note that in fact, the rankings induced by the medians sometimes differ from the ones based on means which are possibly biased by extreme values. But our main conclusion is based on the the Mean MSEs.

5.1 Results

As discussed above, for each combination of method and dataset, we compute mean and standard deviation(SD),median and interquartile range(IQR) of the test set error rates based on the 10 cross validation results. The data results are showed in Table 5.1 and 5.3 respectively. For intuitionistic comparison, we plot mean and standard deviation(SD) in one figure, median and interquartile range(IQR) likewise (see Figures from 5.1 to 5.9). In each figure, the upper left indicates the Means(SDs), the upper right the Medians(IQRs). The cputime of training and the number of support vectors (key samples for KSM) are also presented in the lower figure. In all plots, the methods are compared in parallel and from left to right are KSM with linear spline kernel, KSM with RBF kernel, SVM, LSSVM and RBFN.

In the plots of means and SDs of the errors, the horizontal line for each method marks the mean of MSEs and the vertical line depicts plus and minus the standard error; In the plots of medians and IQRs of the errors, the horizontal line marks the median and the vertical line depict the range of the interquartile range(the medians plus and minus the half of the IQRs). In the lower plot, cuptime(left vertical axis) and SV number(right vertical axis) are presented in bars with wide ones for cuptime and narrow ones for SV number.

We will analysis our results with respect to the three plots. It should be noted that our conclusion is mainly based on the Means of MSEs, but other measures are also considered as to get comprehensive evaluation.

5.1.1 Reproducibility

Table 5.1 summarizes the means and SDs of the MSEs; Table 5.3 summarizes the medians and IQRs of the MSEs. Top scores are underlined. One of the cells is empty due to the implementation imperfections. RBFN gets stuck for "cpuSmall" during training because of memory limitation. We will first analysis the results base on the the means(SDs) and then look into the more robust result of medians(IQRs) for accessorial conclusion. Note that the IQR is not affected by the outliers which has less number of 50% of all the samples.

Consider Table 5.1, from a rough perspective, KSM yielded good performance, but were not top ranked on all data sets. Other three methods also often yielded better results than KSM. For some other problems, the MSE results of RBFN and LSSVM often distributed diffusely which can be observed from the length of SDs "line". For "cpu", "cpuSmall", "housing" and "stock", it is obvious that there exist more or less bad results which increase the mean level of MSEs. It should be noted that for "Friedman2", the RBFN has one very large extreme value of MSE, which has been looked as exceptional result and been removed.

When seeing into more details by considering all the Figures(5.1 to 5.9) of MSEs, we found:

- On problem such like "abalone", the performances of all the methods are almost comparable. This is more likely due to the simply data status which makes this result less interesting.
- On "cpu", the SVM yields the best result with mean of $6.9969e^{-3}$. The result by KSM does not deviate much from that. While the standard deviations of both linear-kernel and RBF-kernel KSMs are relatively larger than SVM, means that the results of KSM disperse more widely than KSM (not to say the LSSVM and RBFN). But before the conclusion is made, the most less IQR of KSM(linear) reveals a higher stability than others. The less the IQR, means the less number the samples that deviate from the median. More specific, in this problem, large SD of the KSM(linear) means there are the results disperse, while the small IQR means the number of those results with large deviations or the outliers are small. Thus the outliers in KSM(linear) has less significance with respect to the whole statistical result.
- On "cpuSmall", SVM again outperforms others. But except for LSSVM, the error spread are similar. The plot of medians and IQRs also confirms that the SVM performance are best on this problem.
- On "housing", KSM yields best result. But compared to LSSVM, the means are are almost the same. The deviations are simular, except for RBFN. In the IQR plot, the KSM has a less dispersion result.
- On "Quake", both the KSM (linear and RBF kernel) outperform others. The error spread status are alike.
- On "stock", the results of KSM(RBF-kernel), SVM and LSSVM are again comparable. LSSVM has a relatively larger SD but a small IQR. On the whole consideration, the KSM(RBF) performs better.
- For artificial problems: on "Friedman1" and "Friedman2", RBFN is as standout as KSM(rbf), which yields small mse and small IQR; on "Friedman3", RBFN outperforms but KSM(rbf) shows relatively lower stability than others.

5.1.2 Computational efficiency

The cpu time for training is considered. Table5.2 summarizes the time spent for every methods and data sets combination. Except for the artificial problems, the KSM are the fastest on all real problems. From Figures 5.1 to 5.9, we found the cpu time spent of other methods heavily increase as the size or dimension of data sets increases. However

the cpu time of KSM is almost increasing linearly. For the artificial problems, the SVM performs a little bit faster than KSM.

But when consider the number of support vectors presented in Table 5.4, the KSM has the smallest number of support vectors, which is necessary for online approximations since less support vectors guarantees efficiency.

5.2 Discussion

In our simulation, the RBFN is trained by forward selection procedure which involves determining the positions and widths of the RBFs in a supervised way. In this case, the RBFN can perform a precise prediction, see from the result of Friedman problems. On the other hand, since all parameters of the RBF are trained, a long training time is likely to occur. For large size data sets, the RBFN usually needs considerable cpu time and memory. This can be seen from the results of third plot in Figure5. Moreover, for relatively high dimension data set such like "cpuSmall","housing", "Stock", the RBFN can not yields good performance. In a word, the RBFN performance fluctuates with different complexity of data sets.

The kernel-based SVM uses only a limited amount of the training samples for the prediction. After the 3D grid search of parameters C,ε and the RBF width g, the SVM can yield good performance. At the mean time, more number of support vectors are needed. For larger size data set, the SVM spends more time on training than KSM (see Table 5.2). The same to the number of support vectors.

In the LSSVM, which solves a linear system, the resulting model is not sparse, namely every training data is connected to each other. In this case, the computational time can not be small, especially for large size data. Except for the RBF kernel width, the larger the regularization parameter λ, the smaller the MSE, but the tough the approximation. In our simulation, a grid search of λ is made, and show the LSSVM can yield very good approximation results with relative large λ. But due to memory limitation the λ can not be too large otherwise the training will cost a long time.

The KSM is definitely the fastest algorithm compared to others. Especially for large size data set, the KSM exhibits excellent speed. Comparing the number of support vectors for KSM and SVM, it is clear that the KSM has the least on almost all problems. For the prediction, although the KSM is not top ranked on all data sets. But even for problems of which the KSM not ranked in the first place, it is almost the one most closed to the best. The KSM has advantages over the SVM and the LSSVM for the number of parameters. The KSM use the least number of support vectors with comparable error. This shows that the using of subset selection with a statistical stopping criteria make the KSM outperform the other support vector-based methods SVM and LSSVM.

For different problems, there is no clear evidence that the KSM with RBF kernel is better than KSM with linear spline kernel on prediction accuracy or efficiency, viceversa.

5.3 Summary

In this chapter, the benchmark results are presented and analyzed. For different problems, the MSEs, the number of support vectors, as well as the training time, are compared as a performance measure. The main result for a same problem is demonstrated in one chart for intuitionistic comparison. The conclusions will be drawn in next chapter.

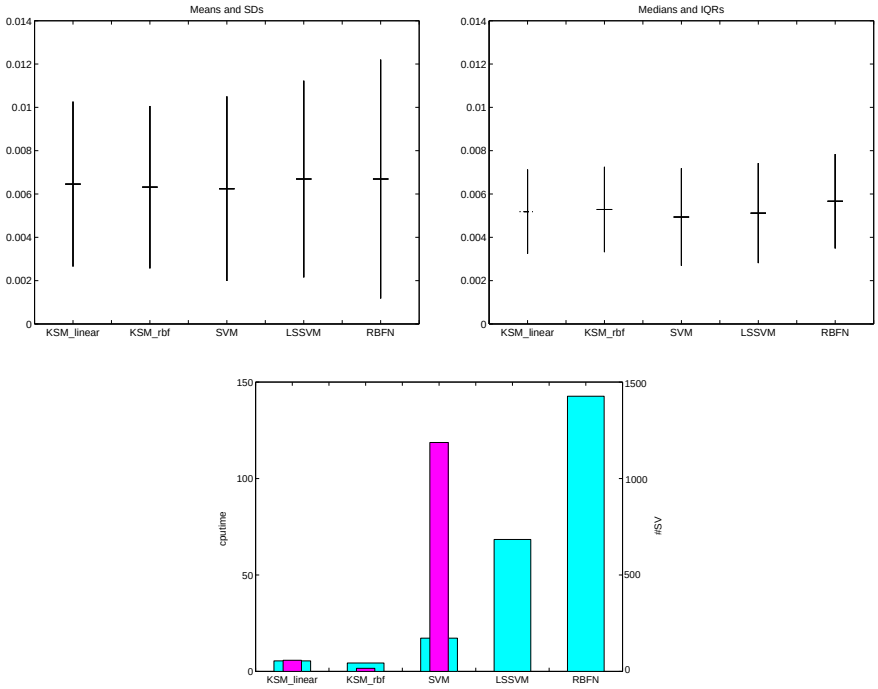


Figure 5.1: "abalone" results

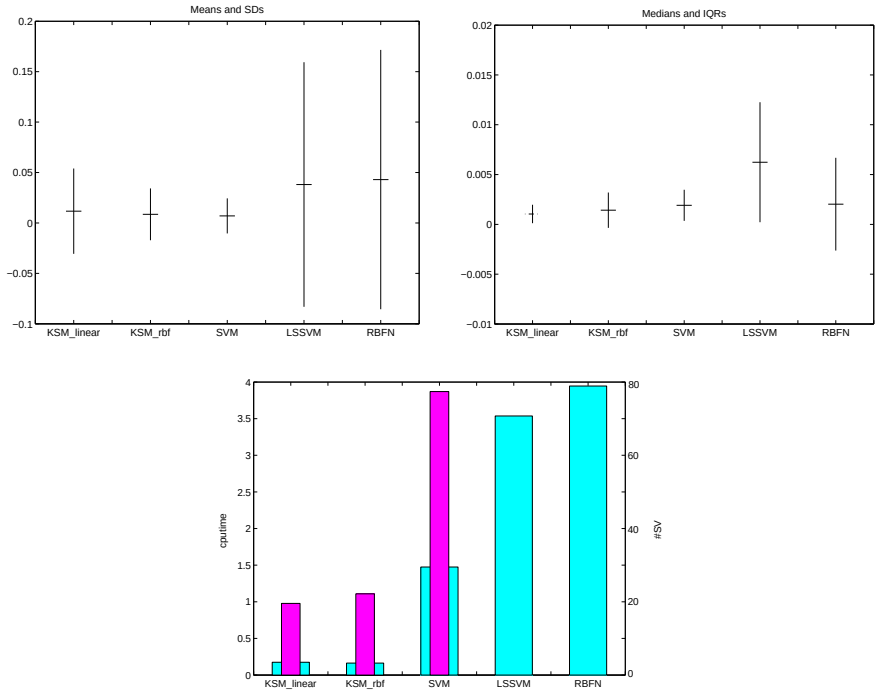


Figure 5.2: "cpu" results

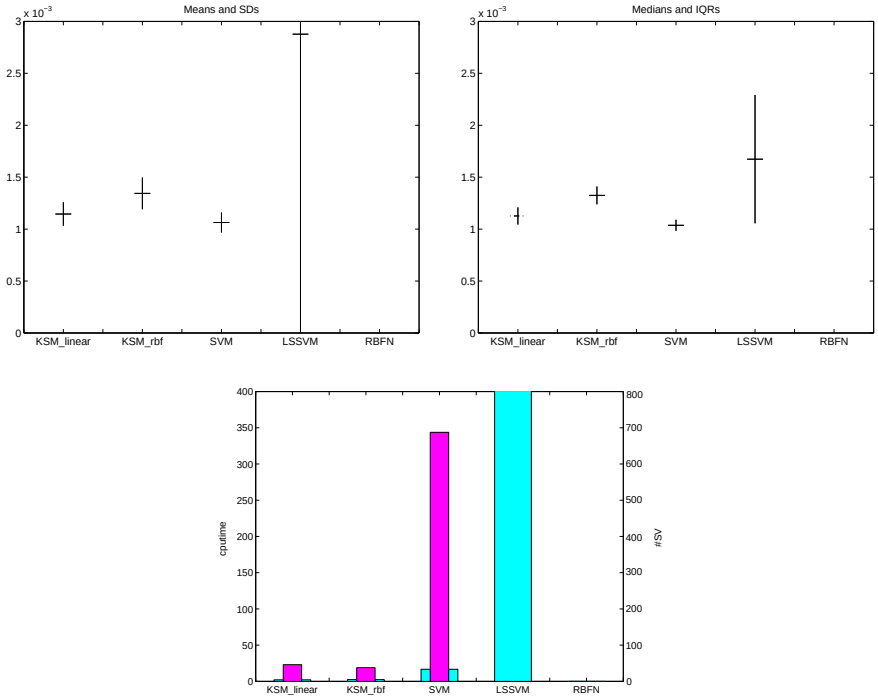


Figure 5.3: "cpuSmall" results

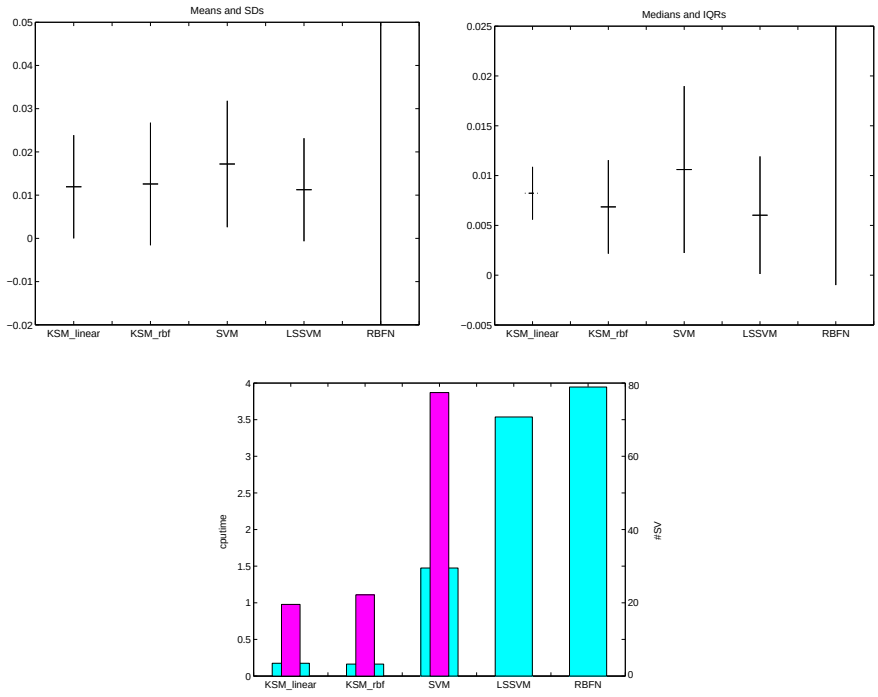


Figure 5.4: "housing" results

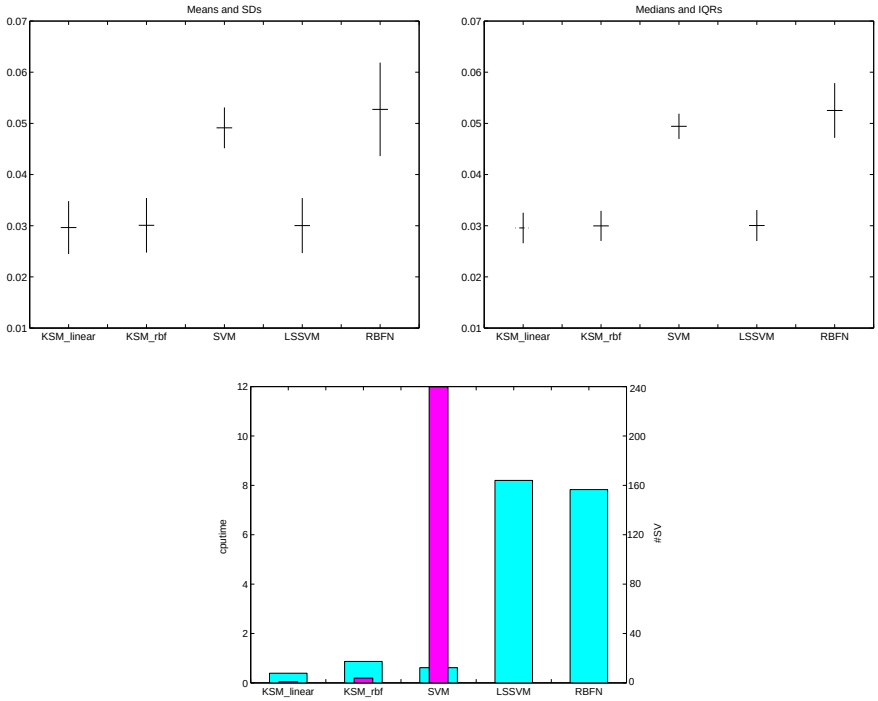


Figure 5.5: "quake" results

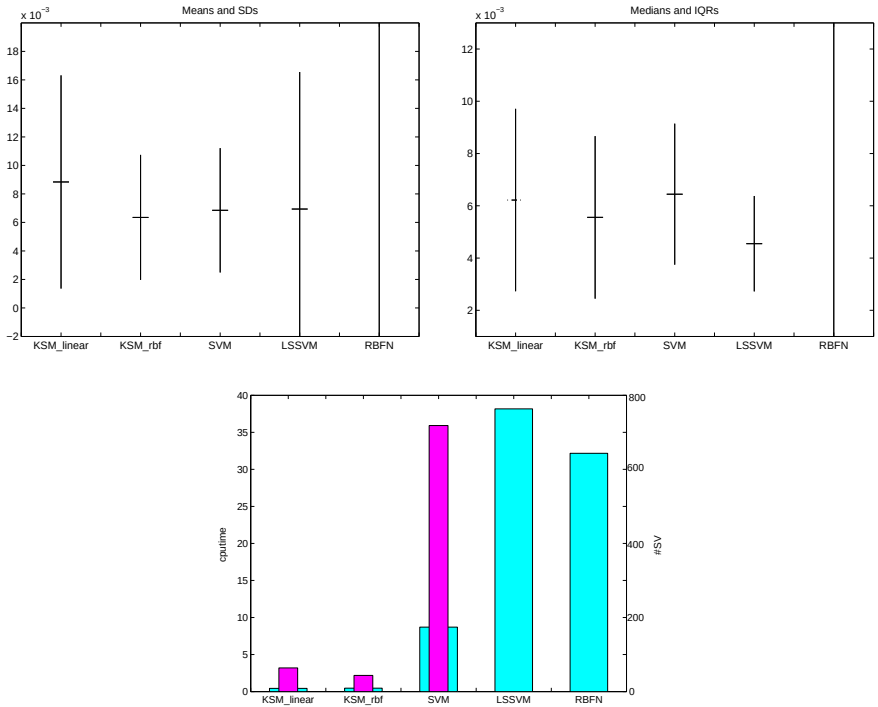


Figure 5.6: "stock" results

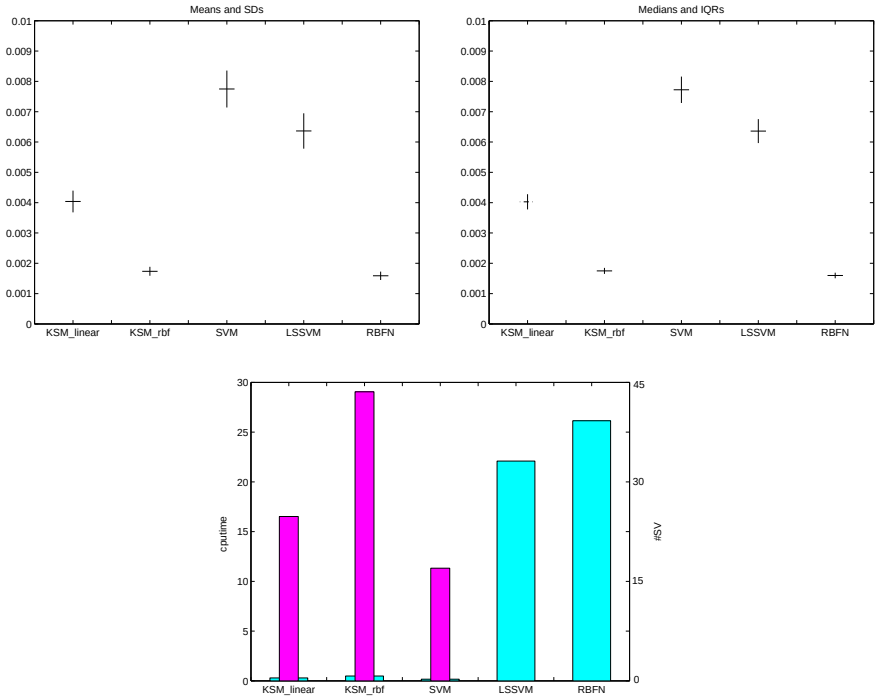


Figure 5.7: "friedman1" results

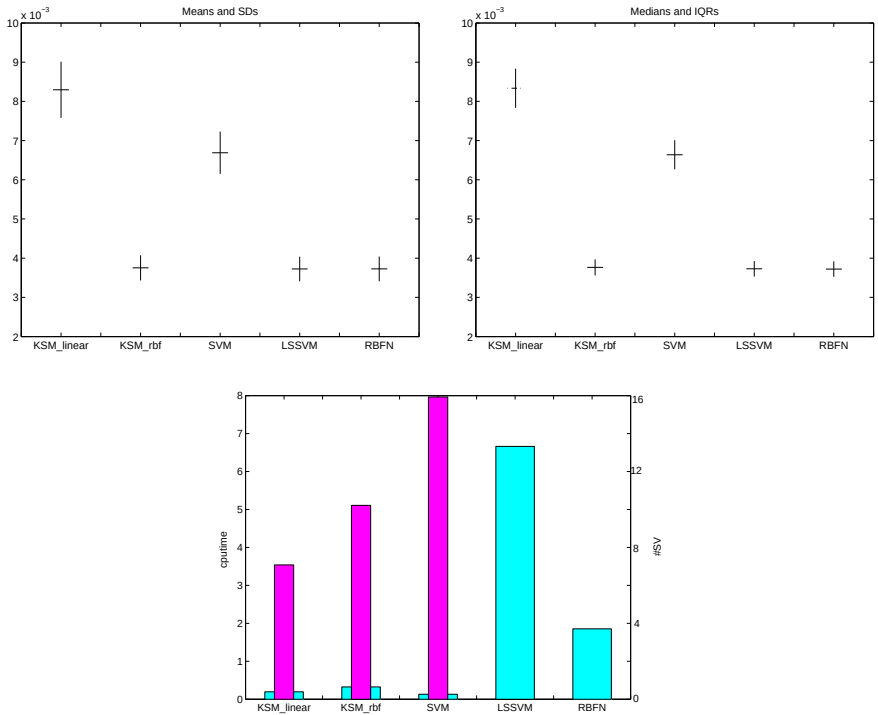


Figure 5.8: "friedman2" results

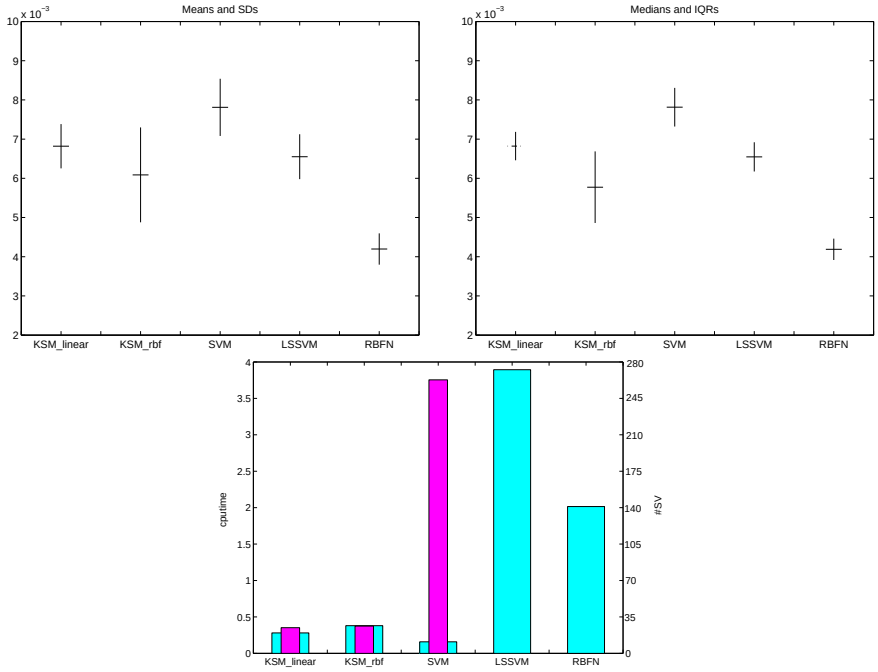


Figure 5.9: "friedman3" results

	KSM(linear)	KSM(rbf)	SVM	LSSVM	RBFN
abalone	6.4573 e-3	6.3161 e-3	6.2430 e-3	6.6724 e-3	6.6905 e-3
(4177 × 8)	(3.8039 e-3)	(3.7462 e-3)	(4.2618 e-3)	(4.5386 e-3)	(5.5121 e-3)
cpu	1.1697 e-2	8.5399 e-3	6.9969 e-3	3.8094 e-2	4.3000 e-2
(209 × 6)	(4.2299 e-2)	(2.5692 e-2)	(1.7368 e-2)	(1.2126 e-1)	(1.2857 e-1)
cpuSmall	1.1454 e-3	1.3439 e-3	1.0638 e-3	2.8766 e-3	
(8192 × 12)	(1.1303 e-4)	(1.5267 e-4)	(9.6239 e-5)	(2.9221 e-3)	
Housing	1.1936 e-2	1.2585 e-2	1.7209 e-2	1.1255 e-2	1.1350 e-1
(506 × 12)	(1.1936 e-2)	(1.4174 e-2)	(1.4624 e-2)	(1.1915 e-2)	(1.6149 e-1)
Quake	2.9633 e-2	3.0074 e-2	4.9126 e-2	3.0020 e-2	5.2736 e-2
(2178 × 3)	(5.1719 e-3)	(5.3391 e-3)	(3.9920 e-3)	(5.3756 e-3)	(9.1321 e-3)
Stock	8.8383 e-3	6.3531 e-3	6.8523 e-3	6.9382 e-3	9.5636 e-2
(950 × 9)	(7.4793 e-3)	(4.3866 e-3)	(4.3608 e-3)	(9.6125 e-3)	(1.8894 e-1)
Friedman1	4.0388 e-3	1.7359 e-3	7.7480 e-3	6.3640 e-3	1.5875 e-3
(1000/5000 × 5)	(3.5876 e-4)	(1.4805 e-4)	(6.0877 e-4)	(5.8267 e-4)	(1.3795 e-4)
Friedman2	8.2949 e-3	3.7530 e-3	6.6898 e-3	3.7245 e-3	3.7262 e-3*
(1000/5000 × 4)	(7.1677 e-4)	(3.2071 e-4)	(5.4007 e-4)	(3.1364 e-4)	(3.1509 e-4)
Friedman3	6.8190 e-3	6.0878 e-3	7.8109 e-3	6.5514 e-3	4.1958 e-3
(1000/5000 × 4)	(5.6454 e-4)	(1.2104 e-3)	(7.2950 e-4)	(5.7258 e-4)	(3.9919 e-4)

The upper part describes real data, the lower part synthetic data sets.
*the original result has one extreme value, which is removed

Table 5.1: Mean Squared Test Set Errors (Means and SDs)

	KSM(linear)	KSM(rbf)	SVM	LSSVM	RBFN
abalone (4177 × 8)	5.4516	4.3346	17.2209	68.3958	142.67
cpu (209 × 6)	0.095	0.08610	0.060	0.1573	0.1701
cpuSmall (8192 × 12)	2.1265	2.4405	16.7046	401.352	
Housing (506 × 12)	0.1743	0.1633	1.4747	3.5371	3.9460
Quake (2178 × 3)	0.3989	0.8678	0.6159	8.1995	7.8273
Stock (950 × 9)	0.4295	0.4587	8.7039	38.1642	32.1668
Friedman1 (1000/5000 × 5)	0.2931	0.4888	0.1697	22.0919	26.145
Friedman2 (1000/5000 × 4)	0.1950	0.3254	0.1316	6.6623	1.8553
Friedman3 (1000/5000 × 4)	0.3221	0.8424	0.2052	3.8933	2.0154

Table 5.2: Training time spent(sec)

	KSM(linear)	KSM(rbf)	SVM	LSSVM	RBFN
abalone	5.1865 e-3	5.2845 e-3	4.9373 e-3	5.1169 e-3	5.6643 e-3
(4177 × 8)	(3.8880 e-3)	(3.9320e -3)	(4.4985 e-3)	(4.6132 e-3)	(4.3512 e-3)
cpu	1.0430 e-3	1.4240 e-3	1.9113 e-3	6.2379 e-3	2.0280 e-3
(209 × 6)	1.8520 e-3	3.5510 e-3	3.1269 e-3	1.2048 e-2	9.3116 e-3
cpuSmall	1.1265 e-3	1.3245 e-3	1.0366 e-3	1.6735 e-3	
(8192 × 12)	(1.6500 e-4)	(1.7200 e-4)	(1.0629 e-4)	(1.2355 e-3)	
Housing	8.2255 e-3	6.8530 e-3	1.0603 e-2	6.0196 e-3	4.7149 e-2
(506 × 12)	(5.2885 e-3)	(9.3865 e-3)	(1.6748 e-2)	(1.1802 e-2)	(9.6285 e-2)
Quake	2.9562 e-2	2.9966 e-2	4.9426 e-2	3.0030 e-2	5.2521 e-2
(2178 × 3)	(5.9770 e-3)	(5.8700 e-3)	(4.9469 e-3)	(6.0591 e-3)	(1.0703 e-2)
Stock	6.2200 e-3	5.5565 e-3	6.4448 e-3	4.5497 e-3	2.9748 e-2
(950 × 9)	(6.9800 e-3)	(6.2225 e-3)	(5.3969 e-3)	(3.6490 e-3)	(5.9360 e-2)
Friedman1	4.0275 e-3	1.7495 e-3	7.7229 e-3	6.3605 e-3	1.5991 e-3
(1000/5000 × 5)	(5.0300 e-4)	(2.0050 e-4)	(8.6838 e-4)	(7.8821 e-4)	(1.8425 e-4)
Friedman2	8.3360 e-3	3.7640 e-3	6.6370 e-3	3.7281 e-3	3.7218 e-3
(1000/5000 × 4)	(9.9950 e-4)	(4.1200 e-4)	(7.4217 e-4)	(3.9613 e-4)	(3.9322 e-4)
Friedman3	6.8215 e-3	5.7735 e-3	7.8145 e-3	6.5473 e-3	4.1881 e-3
(1000/5000 × 4)	(7.2500 e-4)	(1.8265 e-3)	(9.8544 e-4)	(7.4539 e-4)	(5.4495 e-4)

The upper part describes real data, the lower part synthetic data sets.

Table 5.3: Mean Squared Test Set Errors (Medians and IQRs)

	KSM(linear)	KSM(rbf)	SVM
abalone (4177 × 8)	57.51	15.81	1187
cpu (209 × 6)	9.35	6.83	21.7
cpuSmall (8192 × 12)	46.07	37.75	687.55
Housing (506 × 12)	19.56	22.17	77.39
Quake (2178 × 3)	1	3.9	239.72
Stock (950 × 9)	63.83	43.69	718.32
Friedman1 (1000/5000 × 5)	24.78	43.58	16.99
Friedman2 (1000/5000 × 4)	7.08	10.22	15.93
Friedman3 (1000/5000 × 4)	24.65	26.18	262.81

Table 5.4: Number of support vectors

Chapter 6

Conclusions and Recommendations

This chapter presents the conclusions that have been drawn from this thesis and gives recommendations for future work.

6.1 Conclusions

In the introduction four tasks in this project are described. They can be summarized as two aspects: theoretical comparison and experimental comparison.

In chapter 2 and chapter 3, after literature studying about the function approximators, a theoretical review and comparison of them based on our benchmarking purpose has been done. Among the four methods, except for the neural network RBFN, all other three are support vector-based or kernel-based learning methods. There difference lies in the ways of dealing with the feature space. The neural network uses the feature space directly, while the second use it implicitly by the inner-product of the mapping samples in the input space rather than the potentially high dimensional feature space. Thus the RBFN is more likely prone to the curse of dimensionality.

Among the support vector-based methods, the SVM solves a quadratic programming problem in dual space and use the ε-insensitive cost function, obtaining a sparse solution. But three parameters are needed to balance of the trade-off between efficiency and the veracity of the prediction. While the KSM and the LSSVM both use a quadratic cost function. For the KSM, by using limited sample of the training set, sparsity and efficiency are achieved. For the LSSVM, if without pruning, the use of a full training set leads to a large memory demand.

The KSM use a forward selection scheme to find the key samples, the calculation time is saved. Only one parameter need to be tuned during training is the noise vari-

ance. But the use of QR decomposition provides numerically stable and simplicity of calculations.

In chapter 4 and chapter 5, a standard experiment is set. The learning methods are tested on benchmarking learning problems. From the simulation results analyzed in chapter 5, we found:

By using supervised selection training method, the RBFN can perform a precise prediction. However for large size data sets (such like over 8000 samples) or high dimensional problem, the RBFN usually needs considerable cpu time and memory. Thus the RBFN is not suitable as a function approximator in control process.

The kernel-based SVM uses only a limited amount of the training samples for the prediction and can often yield good performance. At the mean time, large number of support vectors are needed. Which is not good in online process. For larger size data set, the SVM spends pretty more time on training than KSM, which is extraordinarily fast.

In the LSSVM, which solves a linear system, the resulting model is not sparse, namely every training data is connected to each other. The simulation results shows that the LSSVM performs very good prediction. In fact the results of our experiment for LSSVM can be better if the parameter λ getting larger. But due to memory limitation the λ can not be too large otherwise the training will cost a very long time. Again the less of efficiency disqualifies the LSSVM to be a good function approximator.

The MSE, is not top ranked on all problems. But even for problems of which the KSM not ranked in the first place, it is almost the one most closed to the best. However the KSM is definitely the fastest algorithm compared to others. Especially for large size data set, the KSM exhibits excellent speed.

The simulation results is consistent with our previously theoretical analysis. The KSM yields good performance, especially when considering the efficiency, which is very important for its application in the online LFFC setting. From the view of reproducibility, for different problems, it is hard to say which one is the best. But by analyzing the statistical dispersion characteristics of the errors, the results of KSM compared to other methods show average precision. In short, our results confirm the potential of KSM to be a good function approximator in application of learning control.

6.2 Recommendations

From this thesis there are some recommendations for future work. These are:

1. The KSM and the SVM are C++ implemented, while the LSSVM and the RBFN are implemented in Matlab. For a more impartial comparison, they should be implemented in the same softwares. Besides, because of hardware limitation, the

RBFN get stuck during training for large size data set. This could be improved.

2. The LSSVM with a pruning scheme which remove those samples that have little influence on the prediction, named LSSVM+, could be a good competitor in future work.

3. In our simulation, two different kernel are used to investigate the influence of kernel type on the KSM behavior. But our result do not show direct relation nor distinct influence. A particular benchmarking on this aspect of KSM can be done in future work.

4. The performance of the new KSM algorithm (BKSM), which handles a block of samples in a way similar to the off-line algorithm, is not satisfied in our simulation. It indicates the different data set may have impact on the algorithm, or there is intrinsic flaw in the algorithm itself. The reason affecting the performance of the BKSM on the benchmarking problems used in this project should be exploited .

5. The KSM benchmarked in this thesis is an off-line method. It can be extended to on-line benchmarking of the RKSM.

Appendix A

Information of benchmarking problems

A.1 Abalone

Description

This data set can be used to obtain a model to predict the age of abalone from physical measurements. The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope – a boring and time-consuming task. Other measurements, which are easier to obtain, are used to predict the number of rings which determines the age. Further information, such as weather patterns and location (hence food availability) may be required to solve the problem.

Data comes from an original (non-machine-learning) study: Warwick J Nash, Tracy L Sellers, Simon R Talbot, Andrew J Cawthorn and Wes B Ford (1994) "The Population Biology of Abalone (Haliotis species) in Tasmania. I. Blacklip Abalone (H. rubra) from the North Coast and Islands of Bass Strait", Sea Fisheries Division, Technical Report No. 48 (ISSN 1034-3288).

Sources

1. Original owners of database: Marine Resources Division Marine Research Laboratories - Taroona Department of Primary Industry and Fisheries, Tasmania GPO Box 619F, Hobart, Tasmania 7001, Australia (contact: Warwick Nash +61 02 277277, wnash@dpi.tas.gov.au)

2. Donor of database: Sam Waugh (Sam.Waugh@cs.utas.edu.au) Department of Computer Science, University of Tasmania GPO Box 252C, Hobart, Tasmania

7001, Australia

3. Date received: December 1995

Number of Instances: 4177

Number of Attributes: 8 (not including the goal)

Attribute information:

Given is the attribute name, attribute type, the measurement unit and a brief description. The number of rings is the value to predict: either as a continuous value or as a classification problem.

Name	Data Type	Meas.	Description
Sex	nominal		M, F, and I (infant)
Length	continuous	mm	Longest shell measurement
Diameter	continuous	mm	perpendicular to length
Height	continuous	mm	with meat in shell
Whole weight	continuous	grams	whole abalone
Shucked weight	continuous	grams	weight of meat
Viscera weight	continuous	grams	gut weight (after bleeding)
Shell weight	continuous	grams	after being dried
Rings	integer		+1.5 gives the age in years

Statistics for numeric domains:

	Length	Diam	Height	Whole	Shucked	Viscera	Shell	Rings
Min	0.075	0.055	0.000	0.002	0.001	0.001	0.002	1
Max	0.815	0.650	1.130	2.826	1.488	0.760	1.005	29
Mean	0.524	0.408	0.140	0.829	0.359	0.181	0.239	9.934
SD	0.120	0.099	0.042	0.490	0.222	0.110	0.139	3.224
Correl	0.557	0.575	0.557	0.540	0.421	0.504	0.628	1.0

Missing Attribute Values: None

A.2 CPU Performance Data

Description

The estimated relative performance values were estimated by the authors using a linear regression method. See their article (pp 308-313) for more details on how the relative performance values were set.

Sources

1. Creator: Phillip Ein-Dor and Jacob Feldmesser
2. Donor of database: Faculty of Management; Tel Aviv University; Ramat-Aviv; Tel Aviv, 69978; Israel
3. Date received: October, 1987

Number of Instances: 209

Number of Attributes: 10 (6 predictive attributes, 2 non-predictive, 1 goal field, and the linear regression's guess)

Attribute information:

1. vendor name: 30
(adviser, amdahl,apollo, basf, bti, burroughs, c.r.d, cambex, cdc, dec, dg, formation, four-phase, gould, honeywell, hp, ibm, ipl, magnuson, microdata, nas, ncr, nixdorf, perkin-elmer, prime, siemens, sperry, sratus, wang)
2. Model Name: many unique symbols
3. MYCT: machine cycle time in nanoseconds (integer)
4. MMIN: minimum main memory in kilobytes (integer)
5. MMAX: maximum main memory in kilobytes (integer)
6. CACH: cache memory in kilobytes (integer)
7. CHMIN: minimum channels in units (integer)
8. CHMAX: maximum channels in units (integer)
9. PRP: published relative performance (integer)
10. ERP: estimated relative performance from the original article (integer)

Missing Attribute Values: None

A.3 cpuSmall

Description

cpuSmall is one of the prototasks of the database CompAct (Computer system activity). The Computer Activity databases are a collection of computer systems activity measures. The data was collected from a Sun Sparcstation 20/712 with 128 Mbytes of memory running in a multi-user university department. Users would typically be doing a large variety of tasks ranging from accessing the internet, editing files or running very cpu-bound programs. The data was collected continuously on two separate occasions.

On both occassions, system activity was gathered every 5 seconds. The final dataset is taken from both occasions with equal numbers of observations coming from each collection epoch.

Sources
Original owners: DELVE repository of data.

Number of Instances: 8192

Number of Attributes: 27 (including 4 goals) Notice that there are two prototasks: "cpu" (23 of all attributes) and "cpuSmall" (13 of all attributes) .

Attribute information:

	Name	Data type	Data range	Description
1	time	continuous	?	Time of day
2	lread	continuous	0..Inf	Number of system buffer reads per second
3	lwrite	continuous	0..Inf	Number of system buffer writes per second
4	scall	continuous	0..Inf	Number of system calls per second
5	sread	continuous	0..Inf	Number of system call reads per second
6	swrite	continuous	0..Inf	Number of system call writes per second
7	fork	continuous	0..Inf	Number of system call forks per second
8	exec	continuous	0..Inf	Number of system call execs per second
9	rchar	continuous	0..Inf	Number of characters transferred by sys reads
10	wchar	continuous	0..Inf	Number of characters transferred by sys writes
11	pgout	continuous	[0,Inf]	Page-out requests per second
12	ppgout	continuous	[0,Inf]	Pages paged out per second
13	pgfree	continuous	[0,Inf]	Pages freed per second
14	pgscan	continuous	[0,Inf]	Pages scanned for freeing per second
15	atch	continuous	[0,Inf]	Page faults/sec satisfied by attaches
16	pgin	continuous	[0,Inf]	Page in requests per second
17	ppgin	continuous	[0,Inf]	Pages paged in per second
18	pflt	continuous	[0,Inf]	"Copy-on-write" page faults
19	vflt	continuous	[0,Inf]	Page faults caused by pages not in memory
20	runqsz	continuous	[0,Inf]	Process run queue size
21	runocc	continuous	0..100	time that that run queue is occupied
22	freemem	continuous	0..512	Number of free memory pages
23	freeswap	continuous	0..15000	Free disk swap blocks
24	usr	continuous	0..100	CPU utilization (user)
25	sys	continuous	0..100	CPU utilization (system)
26	wio	continuous	0..100	CPU utilization (Idle and waiting for I/O)
27	idle	continuous	0..100	CPU utilization (Idle)

- **Prototask: cpuSmall**
  Cases: all
  Inputs: 2 3 4 5 6 7 8 9 10 20 22 23
  Order: Random-order Targets: 24

- **Prototask: cpu**
  ases: all
  Inputs: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 22 23
  Order: Random-order Targets: 24

Missing Attribute Values: None

A.4 Boston Housing Data

Description

This data set concerns the task of predicting housing values in areas of Boston.

Sources

1. Origin: This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.
2. Creator: Harrison, D. and Rubinfeld, D.L. "Hedonic prices and the demand for clean air", J. Environ. Economics and Management, vol.5, 81-102, 1978.
3. Date received: July 7, 1993

Number of Instances: 506

Number of Attributes: 13 continuous attributes (including "class" attribute "MEDV"), 1 binary-valued attribute.

Attribute information:

1. CRIM per capita crime rate by town
2. ZN proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS proportion of non-retail business acres per town
4. CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX nitric oxides concentration (parts per 10 million)
6. RM average number of rooms per dwelling
7. AGE proportion of owner-occupied units built prior to 1940

8. DIS weighted distances to five Boston employment centres
9. RAD index of accessibility to radial highways
10. TAX full-value property-tax rate per $10,000
11. PTRATIO pupil-teacher ratio by town
12. B $1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town
13. LSTAT % lower status of the population
14. MEDV Median value of owner-occupied homes in $1000's

Missing Attribute Values: Noe

A.5 Kinematics

Description

The problem "Kinematics" (specified "KIN8nm") used in this research is taken from the Kin family of data sets. There are eight data sets in this family . They are all variations on the same model: a realistic simulation of the forward dynamics of an 8 link all-revolute robot arm. The task in all data sets is to predict the distance of the end-effector from a target. The inputs are things like joint positions, twist angles, etc. The family has been specifically generated for the delve environment and so the individual data sets span the corners of a cube whose dimensions represent:

- Number of inputs (8 or 32).

- degree of non-linearity (fairly linear or non-linear).

- amount of noise in the output (moderate or high).

All datasets in this family have "kin" as the base of their name (KINematics). Followed by: An integer value signifying the number of input attributes in each case, for example "32". One of the characters "f" or "n" signifying "fairly linear" or "non-linear", respectively. One of the characters "m" or "h" signifying "medium unpredictability/noise" or "high unpredictability/noise", respectively. In this research, we use the "KIN8nm" data set. More background on the rationale for these choices can be found in the delve task array document (http://www.cs.toronto.edu/ delve/data/task-array.html). Detailed documentation on the kin family is also available at (http://www.cs.toronto.edu/ delve/data/kin/kin.ps.gz)

Number of Instances: 8192

Number of Attributes: 9 or 33 (including one goal)

Attribute information:

The inputs are continues values represent joint positions, twist angles, etc.

Missing Attribute Values: None

A.6 Quake

Description

The "Quake" is from a collection of 37 data sets used in the book "Smoothing Methods in Statistics," by Jeffrey S. Simonoff, X Springer-Verlag, New York, 1996. Submitted by Jeff X Simonoff (jsimonoff@stern.nyu.edu). The description of data sources, and further information about the data sets, can be found in the "Descriptions of the data sets" section of the book (http://pages.stern.nyu.edu/ jsimonof/SmoothMeth/).

Number of Instances: 2178

Number of Attributes: 4 (including 1 target)

Attribute information:

1. attribute focal-depth (integer)
2. attribute latitude (real)
3. attribute longitude (real)
4. attribute richter (real)

Missing Attribute Values: None

A.7 Stock Price

Description

The data provided are daily stock prices from January 1988 through October 1991, for ten aerospace companies.

Sources

Number of Instances: 950

Number of Attributes: 10 (including the target feature: stock price of company10)

Attribute information:

continuous values of company1,..,10.

Missing Attribute Values: None

A.8 Friedman #1

Description

The regression problem Friedman 1 as described in Friedman (1991) and Breiman (1996). Inputs are 10 independent variables uniformly distributed on the interval [0,1], only 5 out of these 10 are actually used. Outputs are created according to the formula:

$$y = 10sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + e$$

where x is input values (independent variables); y is output values (dependent variable). e is a Gaussian random noise N(0,1), and the inputs variables are uniformly distributed over the domain [0,1].

References

Breiman, Leo (1996) Bagging predictors. Machine Learning 24, pages 123-140.

Friedman, Jerome H. (1991) Multivariate adaptive regression splines. The Annals of Statistics 19 (1), pages 1-67.

A.9 Friedman #2

The regression problem Friedman 2 as described in Friedman (1991) and Breiman (1996). Inputs are 4 independent variables uniformly distrtibuted over the ranges:

$$0 <= x_1 <= 100$$

$$40\pi <= x_2 <= 560\pi$$

$$0 <= x_3 <= 1$$

$$1 <= x_4 <= 11$$

The outputs are created according to the formula

$y = (x_1^2 + (x_2 x_3 - (1/(x_2 x_4)))^2)^{0.5} + e$

where e is a Gaussian random noise N(0,sd).sd is Standard deviation of noise. The default value of 125 gives a signal to noise ratio (i.e., the ratio of the standard deviations) of 3:1. Thus, the variance of the function itself (without noise) accounts for 90% of the total variance.

A.10 Friedman #3

The regression problem Friedman 2 as described in Friedman (1991) and Breiman (1996). Inputs are 4 independent variables uniformly distrtibuted over the ranges:

$0 <= x_1 <= 100$

$40\pi <= x_2 <= 560\pi$

$0 <= x_3 <= 1$

$1 <= x_4 <= 11$

The outputs are created according to the formula

$y = \arctan((x_2 x_3 - (1/(x_2 x_4)))/x_1) + e$

where e is a Gaussian random noise N(0,sd).sd is Standard deviation of noise. The default value of 125 gives a signal to noise ratio (i.e., the ratio of the standard deviations) of 3:1. Thus, the variance of the function itself (without noise) accounts for 90% of the total variance.

Appendix B

Parameters

The parameters are tuned beforehand. Table 4.3 indicates the parameter need to be tuned for each method. Here is the final values of the parameters used in our benchmarking.

Abalone

KSM(linear)	KSM(rbf)	SVM	LSSVM	RBFN
$\mu = 0.05$	$\mu = 0.05, \sigma = 15$	ε=0.06, C=10, g=1.4	$\sigma = 10, \lambda = 0.1$	scales=0.4

cpu

KSM(linear)	KSM(rbf)	SVM	LSSVM	RBFN
$\mu = 0.05$	$\mu = 0.05, \sigma = 50$	$\varepsilon = 0.06$, C=10 ,g=0.035	$\sigma = 15, \lambda = 0.1$	scales=1

cpuSmall

KSM(linear)	KSM(rbf)	SVM	LSSVM	RBFN
$\mu = 0.05$	$\mu = 0.05, \sigma = 10$	$\varepsilon = 0.05$, C=10 ,g=1	$\sigma = 5, \lambda = 2$	

Housing

KSM(linear)	KSM(rbf)	SVM	LSSVM	RBFN
$\mu = 0.1$	$\mu = 0.1, \sigma = 2.1$	$\varepsilon = 0.1$, C=1000 ,g=0.1	$\sigma = 30, \lambda = 0.5$	scales=6

Quake

KSM(linear)	KSM(rbf)	SVM	LSSVM	RBFN
$\mu = 0.5$	$\mu = 0.5, \sigma = 0.051$	$\varepsilon = 0.3$, C=10 ,g=0.03	$\sigma = 17, \lambda = 0.1$	scales=0.8

Stock

KSM(linear)	KSM(rbf)	SVM	LSSVM	RBFN
$\mu = 0.035$	$\mu = 0.035, \sigma = 1.1$	$\varepsilon = 0.005$, C=10 ,g=1	$\sigma = 50, \lambda = 0.1$	scales=0.56

Friedman1

KSM(linear)	KSM(rbf)	SVM	LSSVM	RBFN
$\mu = 0.05$	$\mu = 0.05, \sigma = 1$	ε0.2, C=10 ,g=0.2	$\sigma = 15, \lambda$=50	scales=2

Friedman2

KSM(linear)	KSM(rbf)	SVM	LSSVM	RBFN
$\mu = 0.11$	$\mu = 0.11, \sigma = 5$	$\varepsilon = 0.2$, C=10 ,g=0.05	$\sigma = 80, \lambda = 65$	scales=25

Abalone

KSM(linear)	KSM(rbf)	SVM	LSSVM	RBFN
$\mu = 0.09$	$\mu = 0.09, \sigma = 1$	$\varepsilon = 0.08$, C=10 ,g=0.05	$\sigma = 80, \lambda = 65$	scales=1

Bibliography

[1] Velthuis W.J.R., *Learning feed-forward control - theory, design and applications*, Ph.D Thesis, University of Twente, Enschede, 2000.

[2] Kruif, B.J. de . *Function Approximation for Learning Control a key sample based approach*. Ph.D Thesis, University of Twente, Enschede, 2004.

[3] Prechelt, L. PROBEN1 - *a set of neural network benchmark problems and benchmarking rules*, Technical Report 21/94, Fakultät Für Informatik, Universität Karlsruhe, September 1994. URL ftp://ftp.ira.uka.de/pub/uni-karlsruhe/papers/techreports/1994/1994-21.ps.gz.

[4] Arthur Flexer. *Statistical Evaluation of Neural Network Experiments: Minimum Requirements and Current Practice*. Technical Report oefai-tr-95-16.

[5] Thomas G. Dietterich, *Experimental Methodology for Benchmarking*. URL http://wwwipd.ira.uka.de/ prechelt/nipsbench/expmeth.ps.gz

[6] Gerco Otten, Theo J.A. de Vries and Job van Amerongen, *Linear Motor Motion Control Using a Learning Feedforward Controller*, IEEE transactions on mechatronics, Vol.2, No.3, September 1997.

[7] Simon Haykin. *Neural Networks. A Comprehensive Foundation*. Prentice Hall, New Jersey, 1994.

[8] Robert J. Howlett, Lakhmi C. Jain. Radial basis function networks. Heidelberg, New York, Physica-Verlag. 2001.

[9] J.O.Rawlings. *Applied Regression Analysis: a research tool*. Wadsworth and Brooks/Cole, Pacific Grove, CA, 1988.

[10] D.M.Allen. *The relationship between variable selection and data augmentation and a method for prediction*. Technometrics, 16(1):125-127, 1974.

[11] G.H.Gohub, M.Heath, and G.Wahba. *Generalised cross-validation as a method for choosing a good ridge parameter.* Technometrics, 21(2):215-223, 1979.

[12] M. J. L. Orr. *Introduction to radial basis function networks.* Technical report, Center for Cognitive Science, University of Edinburgh, 1996.
URL http://www.anc.ed.ac.uk/ mjo/papers/intro.ps.gz.

[13] B.Efron and R.J.Tibshirani. *An introduction to the Boostrap.* Chapman and Hall, 1993.

[14] S.Chen, C.F.N. Cowan, and P.M.Grant. Orthogonal least squares learning for radial basis function networks. IEEE Transactions on Neural Networks, 2(2):302-309, 1991.

[15] Nello Cristianini, *John Shawe-Taylor. An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, Cambridge, UK, 2000.

[16] Alex J.Smola and B.Sch¨lkopf. *A tutorial on support vector regression.* NeuroCOLT Technical Report NC-TR-1998-030, Royal Holloway Colleae, University of London, UK, October, 1998. URL http://www.kernel-machines.org/papers/tr-30-1998.ps.gz.

[17] J.A.K.Suykens, T.Van Gestel, J.De Brabanter, B.De Moor, J.Vandewalle. *Least Squares Support Vector Machines.* World Scientifc, Singapore, July 2002. URL http://www.esat.kuleuven.ac.be/sista/natoasi/suykens.pdf.

[18] S.R. Gunn. *Support Vector Machines for Classification and Regression.* Technical Report, School of Electronics and Computer Science University of Southampton, (Southampton, U.K.), 1998. (Chapter 3) URL http://www.ecs.soton.ac.uk/ srg/publications/pdf/SVM.pdf.

[19] Hettich, S.Blake, C.L.Merz, C.J. (1998). UCI Repository of machine learning databases. Irvine, CA: University of California, Department of Information and Computer Science. URL http://www.ics.uci.edu/ mlearn/MLRepository.html.

[20] Niel, 1998. Data for Evaluating Learning in Valid Experiments. URL http://www.cs.toronto.edu/˜ delve/.

[21] Dr. H. Altay Guvenir and Ilhan Uysal, 2000. Function Approximation Repository. Bilkent University. URL http://funapp.cs.bilkent.edu.tr/DataSets/.

[22] Leisch and Dimitriadou, (2001). `mlbench` - a collection for artificial and real-world machine learning benchmarking problems. R package, Version 0.5-6. Abaible in CRAN(Comprehensive R Archive Network) at http://www.r-project.org/.

[23] StatLib(a system for distributing statistical software, datasets, and information). URL http://lib.stat.cmu.edu/datasets.

[24] Friedman,J. *Multivariate adaptive regression splines.* The Annals of Statistics, 19(1):1. 1991.

[25] Friedman,J. *Greedy function approximation: Agradient boosting machine.* The Annals of Statistics, 29(5):1189-1202, 2001.

[26] Friedman,J. *Stochastic gradient bossting.* Conputational Statistics and Data Analysis, 38(4):367-378.

[27] Leisch,F. Ensemble Methods for Neural Clustering and Classification. Ph.D thesis. Technische Universität Wiem. URL http://www.ci.tuwien.ac.at/~ leisch/papers/Leisch:1998.ps.gz.

[28] Imotec, a compony which develop and analyse mechatronische products and systems. http://www.imotec.nl/

[29] Chih-Chung Chang and Chih-jen Lin. LIBSVM: a library for support vector machines, 2004. Sorfware available at http://www.csie.ntu.edu.tw/~ cjlin/libsvm.

[30] LS-SVMlab (SISTA,KULeuven) http://www.esat.kuleuven.ac.be/sista/lssvmlab/.

[31] Mark Orr. Institute for Adaptive and Neural Computation of the Division of Informatics at Edinburgh University in Edinburgh, Scotland, UK. Sorfware available at http://www.anc.ed.ac.uk/~ mjo/rbf.html.