BEN-GURION UNIVERSITY OF THE NEGEV
FACULTY OF ENGINEERING SCIENCES
DEPARTMENT OF INFORMATION SYSTEMS ENGINEERING

ACCELERATING THE RELEVANCE VECTOR MACHINE

Thesis Submitted in Partial Fulfillment of the Requirements for the M.Sc Degree

By

David Ben Shimon

APRIL 2006

BEN-GURION UNIVERSITY OF THE NEGEV
FACULTY OF ENGINEERING SCIENCES
DEPARTMENT OF INFORMATION SYSTEMS ENGINEERING

ACCELERATING THE RELEVANCE VECTOR MACHINE

Thesis Submitted in Partial Fulfillment of the Requirements for the M.Sc Degree

By

David Ben Shimon

Supervised by:

Dr. Armin Shmilovici

APRIL 2006

ACKNOWLEDGEMENTS

I would like to express my deepest sincere gratitude to my advisor, Dr. Armin Shmilovici for giving me the opportunity to work on such fascinating research, providing me great guidance, support and helpful discussions through the course of this research. His help in making me more knowledgeable has been priceless.

Thanks go to Mr. Mandy Hanner, the head of computers network in ISE at BGU, for providing the hardware devices and software for accomplishing this research.

Also I would like to express my deepest love and gratitude to my parents, brothers and sisters for their continuous encouragement and support. Last but not least, love and gratitude goes to my wife, Anat, for her endless love and support along the path of my academic pursuits.

ABSTRACT

The Relevance Vector Machine (RVM) is a method for training very sparse generalized linear models, while its accuracy is comparable to other machine learning techniques. For a dataset of size N (indicate the number of observations) the runtime complexity of the RVM is $O(N^3)$ and its space complexity is $O(N^2)$ which makes it too expensive for moderately sized problems.

The primary goal of this study was to make the RVM feasible for large data sets. In order to achieve this goal we devise and study partition algorithms, which reduce the complexity of the original algorithm to $O(N^2)$ via solving consecutive partitions of size P. Complexity analysis, statistical analysis and extensive experimentation on benchmark datasets indicate that the partition algorithms can significantly reduce the complexity of the RVM while retaining the attractive attributes of the original solution.

Another contribution of this study was to investigate for the first time the limitations of the RVM regarding the data, such as its non-linearity, its noise, and its dimensionality. Our results indicate that the RVM is capable of finding good solutions for noisy, non linearity and multi dimensional data.

The RVM is a kernel based method. The third contribution of this study was to check for the first time the sensitivity of the RVM to the kernel choice. The results of our empirical study regarding the kernels indicate that unlike assumed before, not every kernel function can assure the convergence of this algorithm. Furthermore, the Matern kernel of order 3 can generate sparser solutions than the more standard Gaussian kernel.

These results provide another important step in the popularization of the RVM algorithm in the machine learning community.

Keywords: Machine Learning, Data Mining, Relevance Vector Machine

TABLE OF CONTENTS

1. INTRODUCTION....................6

1.1 Backgrounds and Brief Problem Description....................6
1.2 The Objectives of this Study....................7
1.3 An Overview of this Dissertation....................8

2. STATE OF THE ART IN RELEVANCE VECTOR MACHINE....................10

2.1 Introduction to Machine Learning....................10
2.2 A Brief Review of Sparse Bayesian Learning....................11
2.2.1 Probabilistic Regression Framework....................14
2.2.2 Using Priors....................15
2.2.3 Bayes Rule....................15
2.2.4 Laplace Approximation....................15
2.2.5 The Bayesian Inference....................16
2.2.6 The Bayesian Learning....................16
2.3 Foundations of the RVM....................17
2.3.1 The RVM and the SVM....................17
2.3.2 Limitations of the Support Vector Machines....................18
2.4 The Relevance Vector Machine....................19
2.5 Formulation of the Relevance Vector Regression....................19
2.5.1 The Learning Algorithm in Briefly....................22
2.5.2 Priors in the RVM....................23
2.5.3 The Sparseness Perspective....................24
2.5.4 Variational Relevance Vector Machine....................25
2.5.5 Example of Relevance Vector Regression....................25
2.6 Usability and Applications of the RVM....................26
2.7 Advantages of the RVM....................27
2.8 Limitations and Disadvantages of the RVM....................28

3. PROBLEMS STATEMENT....................29

3.1 Accelerating the Relevance Vector Machine....................29
3.2 Finding the Best Partition Size....................30
3.3 Finding the Best Algorithm....................30
3.4 Choosing the Kernel Function....................30
3.5 Benchmarking the RVM....................31
3.6 Discussion....................31

4. ACCELERATING THE RELEVANCE VECTOR REGRESSION.........32

4.1 Three Data Partitioning RVM Algorithms..32
4.1.1 The Split and Merge RVM...34
4.1.2 The Incremental RVM..36
4.1.3 The Working Set RVM..37
4.1.4 Comparative Experiments with Constant P...38
4.1.4.1 Data Sets...38
4.1.4.2 Experiment Description..39
4.1.4.3 Evaluation Metrics...39
4.1.4.4 Results..40
4.1.4.5 Resulta Analysis and Discussion...40
4.2 Estimating the Partition Size P 42
4.2.1 The Partition Size Heuristics...43
4.2.2.1 Complexity Analysis..44
4.2.2 Experiments with the Partition Size Heuristics..44
4.2.2.1 Data sets and Evaluation Metrics...44
4.2.2.2 Experiment Description and Results..45
4.2.3 Reducing the Noise Variance Update Rate...46
4.3 Estimating the Acceleration Rate...47
4.3.1 Statistical Analysis of the Results...50
4.4 Comparing the Three Acceleration Heuristics..51
4.4.1 Experiments for Defining the Best Partition Algorithm..................................52
4.4.1.1 Data Sets and Evaluation Metrics...52
4.4.1.2 Results..52
4.5 Discussion...53

5. ACCELERATING THE RELEVANCE VECTOR CLASSIFICATION..55

5.1 Formulation of the Relevance Vector Classification.......................................55
5.1.1 Binary Classification...55
5.1.1.1 The Discrete Problem...56
5.1.1.2 The Continuous Problem..56
5.1.2 How the Relevance Vector Classification is working....................................57
5.1.3 Illustration of the Relevance Vector Classification..58
5.1.4 Multi Classification...59
5.1.5 Computational Complexity..60
5.2 Accelerating the RVC via Data Partitioning..61
5.2.1 The Partition Algorithm...61
5.2.1.1 Key Idea...61
5.2.1.2 The Size of the Working Set..63
5.2.1.3 The Working Set Algorithm...63
5.2.1.4 Complexity Analysis of the Working Set...64
5.3 Comparative Experiments...65
5.3.1 Measuring the Acceleration Rate...65
5.3.1.1 Statistical Analysis of the Complexity Model..68

5.3.2 Experiments with Benchmark Classification Problems....70
5.3.2.1 Data Sets and Evaluation Metrics....70
5.3.2.2 Experiment Description....70
5.3.2.3 Results....71
5.4 Summary and Discussion....72

6. SELECTING THE KERNEL FUNCTION....73

6.1 Introduction to Kernels and the Kernel Trick....73
6.1.1 Types of Kernels....74
6.1.1.1 The Gaussian kernel....75
6.1.1.2 The Matern kernel....76
6.1.1.3 The Finite Support Matern kernel....77
6.2 The RVM as a Kernel Based Method....78
6.3 Comparative Experiments....81
6.3.1 Experiment description....81
6.3.2 Data Sets and Evaluation Metrics....82
6.3.3 Results and Conclusions....82
6.4 Discussion....84

7. SUMMARY AND DISCUSSION....85

7.1 The RVM....85
7.1.1 Strengths of the RVM....85
7.1.2 Weaknesses of the RVM....86
7.2 Novelty and Contributions of this Study....87
7.2.1 The First Contribution....87
7.2.2 The Second Contribution....88
7.2.3 The Third Contribution....88
7.2.4 Practical Conclusions and Recommendations....88
7.3 Suggestions for Future Research....89

REFERENCES....91

APPENDICES....94

Appendix A. Benchmark Data Sets....94
Appendix B. Statistical Analysis of sub-section 4.3....96
Appendix C. Statistical Analysis of sub-section 5.3.3....97
Appendix D. Numerical Stability....98
Appendix E. Reintroduction of Deleted Basis Functions....100
Appendix F. Kerenels Width Parameters....102

LIST OF TABLES

Table 4.1.4.1.1 Regression benchmark datasets...39
Table 4.1.4.4.1 Comparative RMSE...40
Table 4.1.4.4.2 Comparative number of RVs...40
Table 4.1.4.4.3 Comparative time to convergence...40
Table 4.1.4.5.1 Friedman #1 partition size results..42
Table 4.2.2.1.1 Pumadyn family regression datasets...45
Table 4.2.2.2.1 Partition size heuristics results...46
Table 4.3.1 Results over different size of the same regression dataset.......................48
Table 4.3.1.1 Statistical results for the regression basic RVM...................................51
Table 4.3.1.2 Statistical results for the regression working set RVM.........................51
Table 4.4.1.2.1 Comparative RMSE..52
Table 4.4.1.2.2 Comparative number of RVs..53
Table 4.4.1.2.3 Comparative time to convergence..53
Table 5.3.1.1 Results over different size of the same classification dataset...............66
Table 4.3.1.3 Statistical results for the classification basic RVM..............................69
Table 4.3.1.4 Statistical results for the classification working set RVM....................69
Table 5.3.2.1.1 Classification benchmark datasets..70
Table 5.3.2.3.1 The results of the classification experiments....................................71
Table 6.1.1.1 Commonly used kernel functions...75
Table 6.3.3.1 Comparative RMSE..82
Table 6.3.3.2 Comparative number of RVs...83

LIST OF FIGURES

Figure 2.1.1 Example of the data set..........10
Figure 2.1.2 Possible solution to the problem..........11
Figure 2.1.3 Multiple solutions for one problem..........11
Figure 2.2.1 The data we wish to learn the function about..........12
Figure 2.2.2 Over fitted model..........13
Figure 2.2.3 Ideal model..........13
Figure 2.5.2.1 Priors-Graphical model..........23
Figure 2.5.5.1 The sinc data set example..........25
Figure 2.5.5.2 The solution for the sinc dataset..........26
Figure 4.1.1 The idea of the premature convergence flag..........34
Figure 4.1.1.1 The Split and Merge algorithm..........35
Figure 4.1.2.1 The Incremental algorithm..........37
Figure 4.1.3.1 The Working Set algorithm..........38
Figure 4.1.4.5.1 Comparative results for Boston dataset..........41
Figure 4.3.1 Comparative RMSE in regression problem..........48
Figure 4.3.2 Comparative number of RVs in regression problem..........49
Figure 4.3.3 Comparative time in regression problem..........49
Figure 4.3.1.1 Non-linear regression line for the basic RVM..........50
Figure 4.3.1.2 Non-linear regression line for the working set RVM..........50
Figure 5.1.3.1 100 examples from Ripley's synthetic data..........59
Figure 5.1.3.2 The solution of the RVM to Ripley's problem..........59
Figure 5.3.1.1 Comparative time in classification problem..........66
Figure 5.3.1.2 Comparative classification test error..........67
Figure 5.3.1.3 Comparative number of RVs in classification problem 67
Figure 5.3.1.1.1 Non-linear regression line for the basic RVC..........68
Figure 5.3.1.1.2 Non-linear regression line for the WS classification..........69
Figure 6.1.1 Kernel trick example..........74
Figure 6.1.1.1.1 The Gaussian kernel functions..........76
Figure 6.1.1.2.1 The Matern Kernel..........77
Figure 6.2.1 Student-t density function for the posterior of *w*..........80

1. INTRODUCTION

This chapter overview the problems addressed in this dissertation, and its structure.

1.1 Backgrounds and Brief Problem Description

Two of the most important problems in machine learning are regression and classification. Given a dataset of inputs targets pairs, $\{X_n, t_n\}_{n=1}^{N}$, the goal is to build a model from the dataset which can correctly predict the targets for new input observations. In the classification problem we want to predict the class membership of a new input, for example, we may wish to automatically diagnose which patients are likely to develop a certain disease using a classifier developed from a medical dataset, and in the regression we wish to predict the continuous targets values of the new inputs. We also want to avoid over fitting to the training set while building these models.

In order to study the problem of learning and to be able to generalize to new data we need an additional type of structure; a similarity structure between any two points. It means that given a new example x_j we choose t_j such that (x_j, t_j) is in some sense similar to training examples in $\{X_n, t_n\}_{n=1}^{N}$. Thus we need a similarity measure among the training examples. For example characterizing the similarity of binary classification outputs $(0,1)$ is easy since only two situations can occur: two labels can either be identical or different. However characterizing the similarity measure for the inputs such that the similarity will contribute the final learned model is a complex issue that lies at the core of machine learning area. Most of the statistical machine learning algorithms use similarity measure of the form: $k(x_i, x_j) \to R$ where k is a symmetric function that given two inputs, x_i and x_j, it returns a real number presenting their similarity. The function k is called the kernel [6] and there are kernels for presenting similarity and distance between points. Characterizing the similarity among all possible pairs of points in the input data results in a full rank matrix representation. For example, using kernel for a training set comprises five examples results in a matrix of the size of 5×5 where every column represent the similarity between a specific input to the rest of the examples in the training set. This matrix is denoted as $\mathbf{\Phi}$ and contains the transformation from the input space (X) to the feature space (H) using the kernel *k*. There is an assumption that with a suitable kernel function (k) the model can learn particular structures in the feature space. Embedding the data into H via k has additional benefits which we discuss later.

The Relevance Vector Machine (RVM) [21] is a statistical parametric technique for training sparse generalized linear models, and its accuracy is comparable to other machine learning techniques such as the Support Vector Machine (SVM) [5], [14]. The RVM combines several advantages which are difficult to achieve in a single algorithm, such as a good generalization, a sparse solution that indicates the simplicity of the final model, and a fully probabilistic model that allows us to build the confidence intervals for describing uncertainty in the predictions.

Like other statistical parametric techniques for machine learning, the RVM is computationally intensive. The complexity of the RVM increases with the size of the dataset. For a dataset of size N the runtime complexity of the RVM is $O(N^3)$ and its memory storage is $O(N^2)$. These two facts limit the usability of the RVM for datasets comprising several thousands of observations and upwards.

The RVM uses a kernel in order to represent the inputs and to find the target function of the classifier or for the regression model in this feature space. There are many types of kernels, and choosing the best kernel for a given problem dramatically affects the results. All of the implementations of the RVM we know about use Gaussian kernel. Recently, it was suggested [11] that the Matern kernel may be potentially better than the Gaussian.

Many algorithms such as support vector machines, random forest, neural networks etc., are being used in the machine learning area. It is clear that there is no single algorithm that can demonstrate its dominance over the other algorithms for all kind of problems (the NFL theorem). The RVM is a new algorithm in the machine learning area and it is not yet to known how the RVM can handle non-trivial problems.

1.2 The Objectives of this Study

1. Adapting the RVM so that it can handle large scale problems in reasonable time while simultaneously retaining its original properties: the accuracy and the sparsity of the solution. Three different algorithms are introduced in this dissertation that partitions the dataset into manageable chunks. Extensive experimentation indicates that the partition algorithms can significantly reduce the

complexity of the RVM while retaining the attractive attributes of the original solution.

2. The size of these partitions has a crucial affect on the training time. We wish to develop heuristic for estimating a partition size leading to decent training times for every dataset.
3. We wish to find which partition algorithm is the best one.
4. The RVM is kernel based method, which means it uses a bivariate kernel function for mapping the data into a high dimensional feature. We wish to check the sensitivity of the RVM to the kernel choice and specifically to investigate whether the Matern kernel can obtain better results comparable to the Gaussian kernel.
5. It is well known that the behavior of machine learning algorithms depend on the properties of the datasets. We wish to investigate if the RVM can handle problems which have high degree of non-linearity, noise and dimensionality.

1.3 An Overview of this Dissertation

The rest of this dissertation includes six main parts.

Chapter 2 - *State of the art in Relevance Vector Machine* - provides a description of the RVM. First, I introduce some concepts from machine learning and sparse Bayesian learning. Than, I set the foundations for the RVM and its principles. The Formulation of the RVM in the regression scenario followed by it's a survey of its application in the literature. This chapter concludes with the advantages and the limitations of the RVM.

Chapter 3 - *Problems Statement* - is a comprehensive description of the aims of this study and the problems of the RVM that this dissertation is addressing.

Chapter 4 - *Accelerating the Relevance Vector Regression* - provides a description of three partition algorithms which accelerating the RVM. Complexity analysis, simulations on benchmark datasets and statistical analysis demonstrate the feasibility of the partitions algorithms for regression problems. Furthermore, a heuristic that determines a decent size of the partition is provided. The last section of this chapter is dedicated to answer which one of the partition algorithms is superior.

Chapter 5 - *Accelerating the Relevance Vector Classification* - first provides the formulation and the aspects of the Relevance Vector Classification (RVC). In this chapter we provide a classification version of one of the partition algorithms that was suggested in chapter 4. Complexity analysis, statistical analysis and simulation on benchmark classification datasets are all indicate that the acceleration via the partition algorithm is achieved in the RVC as well.

Chapter 6 - *Selecting the Kernel Function* - investigates the RVM for aspects related to the kernel function and properties of the training data. The goal of this chapter is to check if there are some kernels that are more suitable for given types of datasets such as high degree of non-linearity, noise and dimensionality. We survey some common kernel function and the Matern kernel. Simulations on benchmark datasets compare the RVM for Matern kernel and the Gaussian.

Chapter 7 – *Summary and Discussion* - concludes this dissertation with a summary of the strengths and weaknesses of the RVM, the contribution of this research, and recommendations for future research.

The *Appendices* provide some in depth information dropped from the other sections for readability purposes.

2. STATE-OF-THE-ART IN RELEVANCE VECTOR MACHINE

This chapter provides a detailed literature survey on the state-of-the-art of the relevance vector machine. First, we introduce some concepts from machine learning and sparse Bayesian learning. Then, we set the foundations for the RVM, its principles and the formulation of the RVM in the regression scenario follows by a survey of its application in the literature. This chapter concludes with the advantages and the limitations of the RVM. A formulation of the Relevance Vector Classification is detailed in chapter 5.

2.1 Introduction to Machine Learning

The goal of supervised machine learning is to use training set Sx to 'learn' a function $y(x)$ that correctly explains observation/targets $\mathbf{t}$ given input data $\mathbf{x}$. That is $t = y(x), x \in S_x$. Consider we are given a set of points (x_i, t_i) and we wish to find a function $y(x)$ such that $t_i = y(x_i)$. There could be multiple functions that correctly explained the targets. Figures 2.1.1, 2.1.2, 2.1.3 illustrates six data points and different possible solutions to the problem.

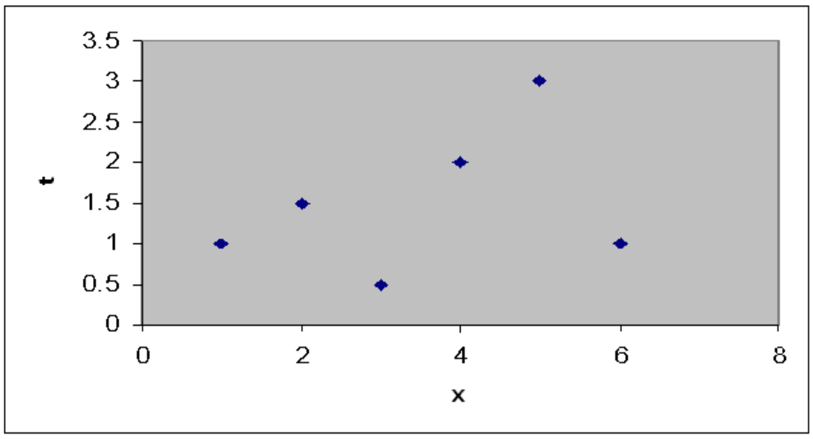

Figure 2.1.1 Example of the data set

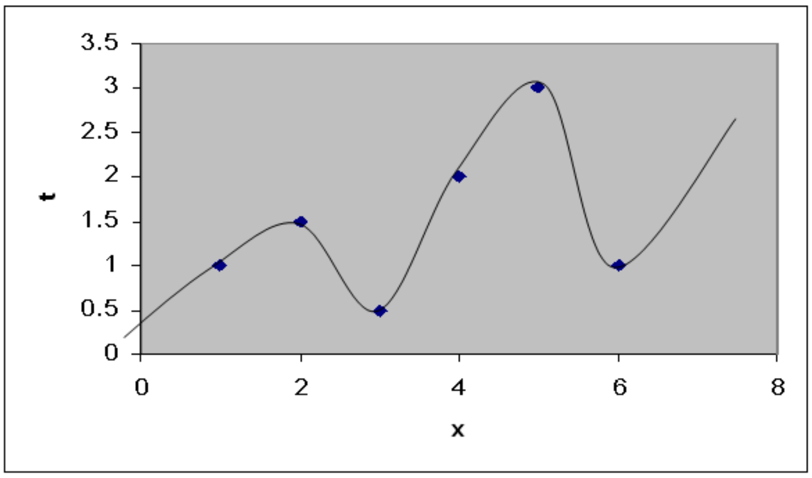

Figure 2.1.2 A possible solution to the problem

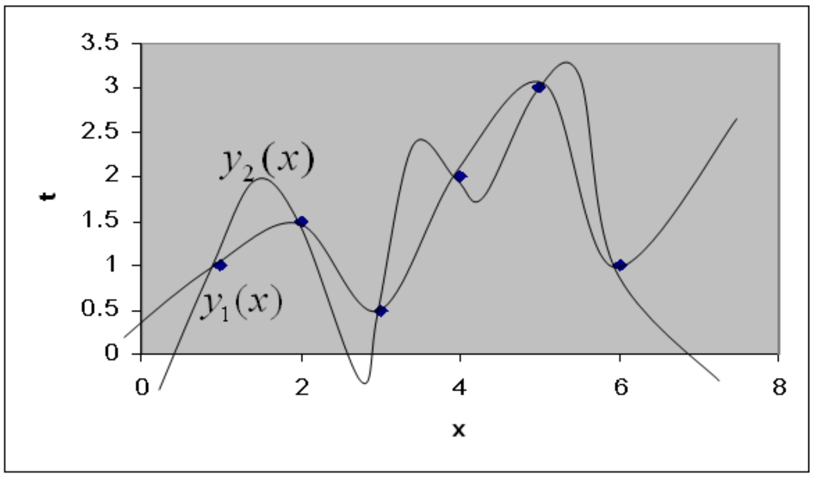

Figure 2.1.3 There are multiple possible solutions for a single problem

So practically which solution should we choose? This is what machine learning is all about; finding the best function that correctly explains the targets. Of course we would like to have this function with some additional properties such as negation of over fitting etc., however, this is the key idea.

2.2 Brief Review of Sparse Bayesian Learning

Assume we have a data set containing 20 samples generated from the function $y = \sin(x)$ with added Gaussian noise with variance 0.3.

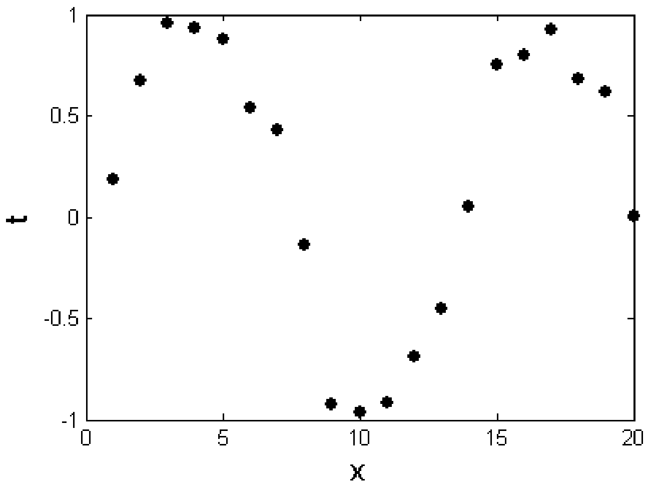

Figure 2.2.1 The data we wish to learn the function about

The input observations are denoted as $X_n, n = 1,2,...N$. There is an associated real valued target, $t_n, n = 1,2,...N$ for each Xn and we want to model this data with some parameterized function $y(x;\mathbf{w})$ where $\mathbf{w} = (\mathrm{w}_1, \mathrm{w}_2 ... \mathrm{w}_\mathrm{m})$ is the vector of the model parameters. Also we consider a model which is linear sum of non linear fixed *m* basis function such as (2.2.1).

$$y(x;\mathbf{w}) = \sum_{m=1}^{M} \mathbf{w}_m \mathbf{\Phi}_m(x) \tag{2.2.1}$$

A common choice for the basis functions is the Gaussian data centered function:

$$\mathbf{\Phi}_m(x) = e^{-(x-x_m)^2/r^2} \tag{2.2.2}$$

The goal is to find values for $\mathbf{w}$ such that $y(x;\mathbf{w})$ makes good predictions for new input observations where $\mathbf{w}$ vector models the generative sin(x) function in our example.

The classic way to do so is by using the "least squares" (LS) approach, minimizing the error measure.

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - \sum_{m=1}^{M} w_m \Phi m(x_n)\right)^2 \tag{2.2.3}$$

where $\mathrm{t} = (\mathrm{t}_1, \mathrm{t}_2 \ldots, \mathrm{t}_\mathrm{n})$ and $\mathbf{\Phi}$ is the matrix which represents the input observation using the basis function we choose such that $\mathbf{\Phi}_{nm} = \mathbf{\Phi}_m(x_n)$. Matrix notation of the LS approach is given in (2.2.4).

$$\mathbf{w}_{LS} = (\Phi^T \Phi)^{-1} \Phi^T t \tag{2.2.4}$$

Using all the 20 observations and the 20 basis functions, minimization of the squared error $E(\mathbf{w})$ will lead to severe over fitting. Figures 2.2.2 and 2.2.3 illustrate an overfitted solution and the desirable solution respectively.

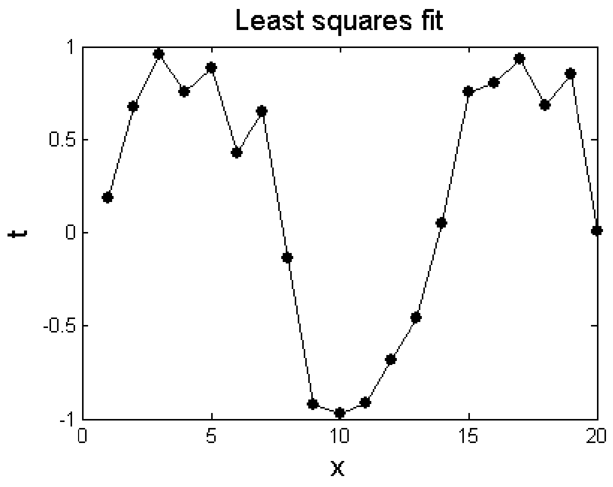

Figure 2.2.2 Overfitted solution

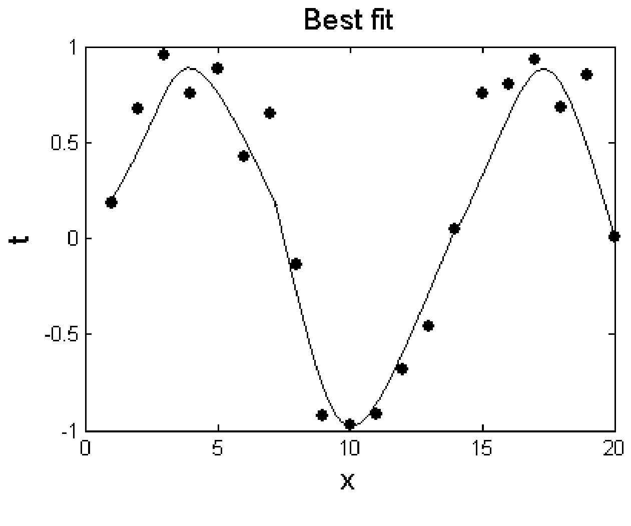

Figure 2.2.3 The best fit, we wish to have a solution with a smoothness along the spines

From here comes the necessity of the prior knowledge. The prior knowledge assists to determine which model is better. It is obvious that the smoother function is better than the LS model because of its generalization property.

In order to make the function smoother can add a weight penalty to the error term that we minimize.

$$\bar{E}_{LS+penalty}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - \sum_{m=1}^{M} w_m \Phi m(x_n)\right)^2 + \lambda E(\mathbf{w}) \tag{2.2.5}$$

Standard choice for the penalty term can be:

$$\lambda E(\mathbf{w}) = \frac{1}{2}\sum_{m=1}^{M} w^2{}_m \tag{2.2.6}$$

This procedure known as the "penalized least squares" estimation (or regularization) and its matrix notation is:

$$\mathbf{W}_{PLS} = (\Phi^T \Phi + \lambda \mathbf{I})^{-1} \Phi^T \mathbf{t} \tag{2.2.7}$$

However in order to get a smooth function we need to find the appropriate λ which leads us to the desired function. The hyper parameter λ (the regularization constant) balances the trade off between the error measure and the penalty term (which depends on the complexity of the model); between how well the function fits the data and how smooth it is.

2.2.1 A Probabilistic Regression Framework

Assume the data is from the following model.

$$t_n = y(x_n; \mathbf{w}) + \varepsilon_n \tag{2.2.1.1}$$

where $\mathrm{y}(\mathrm{x}_\mathrm{n}; \mathbf{w})$ is the earlier model discussed above and ε_n is the noise term.

For the noise model we define Gaussian distribution with zero mean and variance σ^2.

$$P(\varepsilon_n \mid \mathbf{w}, \sigma^2) = N(0, \sigma^2) \tag{2.2.1.2}$$

Since the targets are generated from (2.2.1.1) then the conditional probability to get some target (likelihood) is:

$$p(t_n \mid \mathbf{w}, \sigma^2) = N(y(x_n; \mathbf{w}), \sigma^2) \tag{2.2.1.3}$$

Assuming independency between targets, the likelihood of the complete data set is:

$$p(\mathbf{t} \mid \mathbf{w}, \sigma^2) = \prod_{n=1}^{N} p(t_n \mid \mathbf{w}, \sigma^2) = \prod_{n=1}^{N} (2\pi\sigma^2)^{-1/2} \exp\left(-\frac{(t_n - y(x_n; \mathbf{w}))^2}{2\sigma^2}\right) \tag{2.2.1.4}$$

Now we wish to maximize this likelihood, so the "maximum likelihood" estimate for $\mathbf{w}$ is that value which maximizes (2.2.1.4).

2.2.2 Using Priors

As with the penalized least squares model we use the penalty term in order to smooth the function and finding the tradeoff. We now wish to control the complexity of our new model via prior distribution which expresses our degree of belief over values that $\mathbf{w}$ might takes.

$$p(\mathbf{w} \mid \alpha) = \prod_{m=1}^{M} \left(\frac{\alpha}{2\pi} \right)^{1/2} \exp\left(-\frac{\alpha}{2} w^2{}_m \right) \tag{2.2.2.1}$$

This is a zero-mean Gaussian prior; independent for each weight with common inverse variance hyper parameter α. We are expressing a preference for smoother models by declaring smaller weights to be a-priori more probable.

2.2.3 Bayes Rule

Assume that D is a dataset and $\mathbf{w}$ are some model parameters. Bayes rule provides a way to manipulate uncertainties and beliefs.

The probability of a parameter w given the datasets D is:

$$P(w \mid D) = \frac{P(D \mid w) P(w)}{P(D)} \tag{2.2.3.1}$$

where $P(D \mid w)$ is the likelihood of w, $P(w)$ is the prior probability of w, $P(w \mid D)$ is the posterior probability of w given D and $P(D) = \int P(D \mid w) P(w) dw$.

2.2.4 The Laplace Approximation

The Laplace deterministic approximation is method of approximating the posterior distribution with a Gaussian centered at the maximum a-posteriori (MAP) estimate. Assume that D is a dataset, M is a model and $\mathbf{w}$ are the model's parameters. The posterior probability of the model using Bayes rule can be expressed as:

$$P(M \mid D) \propto \frac{P(M) P(D \mid M)}{P(D)} \tag{2.2.4.1}$$

For a dataset D containing many examples relatively to the number of parameters d in $\mathbf{w}$, the parameters' posterior is approximately Gaussian around the MAP estimate $\hat{w}_i$:

$$P(w_i \mid D, M) \approx (2\pi)^{-d/2} \mid A \mid^{0.5} \exp\left\{-\frac{1}{2}(w_i - \hat{w}_i)^T A(w_i - \hat{w}_i)\right\} \qquad (2.2.4.2)$$

Where $-A$ is $d \times d$ Hessian (the covariance of the fitted Gaussian) of the log posteriori $Aij = \dfrac{d^2}{dw_i dw_j} \ln p(w \mid D, M) \mid_{w=\hat{w}}$.

The approximation lies on the fact that using specific regularity conditions, the posterior distribution approaches Gaussian distribution as the number of examples grows. This approximation makes the posterior probability of the model parameters integrable.

Part of explanation adapted from NIPS tutorial 1999[26]. Further details about the Hessian available in appendix D.

2.2.5 The Bayesian Inference

Given the likelihood, the priors and the noise variance rather than computing a single point estimate $\mathbf{w}_{\mathrm{LS}}$ for the weights we compute the posterior distribution via Bayes rule.

$$p(\mathbf{w} \mid \mathbf{t}, \boldsymbol{\alpha}, \sigma^2) = \frac{likelihood * prior}{normalizing\ factor} = \frac{p(\mathbf{t} \mid \mathbf{w}, \sigma^2) p(\mathbf{w} \mid \boldsymbol{\alpha})}{p(\mathbf{t} \mid \boldsymbol{\alpha}, \sigma^2)} \qquad (2.2.5.1)$$

The posterior is also a Gaussian since Laplace approximation performed here.

$$p(\mathbf{w} \mid \mathbf{t}, \alpha, \sigma^2) = N(\mu, \Sigma) \qquad (2.2.5.2)$$

The mean $\boldsymbol{\mu}$ and the covariance $\boldsymbol{\Sigma}$ compute as:

$$\begin{aligned} \boldsymbol{\mu} &= (\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \sigma^2 \boldsymbol{\alpha} \mathbf{I})^{-1} \boldsymbol{\Phi}^T \mathbf{t} \\ \boldsymbol{\Sigma} &= \sigma^2 (\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \sigma^2 \boldsymbol{\alpha} \mathbf{I})^{-1} \end{aligned} \qquad (2.2.5.3)$$

2.2.6 The Bayesian learning

Lets look at how the posterior $p(\mathbf{w} \mid \mathbf{t}, \alpha, \sigma^2)$ evolves as we observe data t_n. We can compute the posterior incrementally, the data assumed independent given $\mathbf{w}$.

Notation (2.2.5.4) present the development of the posteriori where $\mathbf{t} = (t_1, t_2, t_3)$:

$$\begin{aligned} p(\mathbf{w} \mid t_1, t_2, t_3) &\propto p(t_1, t_2, t_3 \mid \mathbf{w}) p(\mathbf{w}) \\ &\propto p(t_2, t_3 \mid \mathbf{w}) p(t_1 \mid \mathbf{w}) p(\mathbf{w}) \\ &\propto \mathit{Likelihood\ \ of}\ (t_2, t_3) * \mathit{posterior\ \ having\ \ observed}\ t_1 \end{aligned} \tag{2.2.5.4}$$

So more generally, having observed $(t_1, t_2 \dots t_k)$ we can treat the posterior as the prior for the remaining data $(t_{k+1} \dots . t_N)$.

2.3 Foundations of the RVM

The RVM addresses two important problems in machine learning, regression and classification. We are familiar with known algorithms addressing those problems such as linear regression, non linear regression, neural networks etc., however, statistical parametric techniques such as the RVM and the Support Vector Machine [7] have several advantages over other machine learning methods. Usually they are more accurate, enjoy great generalizations properties, the final model of the solution is sometimes very simple, and they have the ability of solving very highly dimensional problems (hundreds or even thousands of dimensions).

Most of the familiar algorithms do not address complexity issues related to the size of the training set. For example assume we have a set of inputs of 100 records (observations) and each record has 10 dimensions along with 100 corresponding targets, if we construct a linear regression for predicting the target for a new input we will probably not receive statistically significant results. If we attempt a non linear regression, then the problem of not knowing the structure function of the model will arise. Furthermore, most of the methods except the RVM lack a direct probabilistic interpretation and can not easily provide predictive distributions.

2.3.1 The RVM and the SVM

The relevance vector machine is a probabilistic sparse kernel model and uses a Bayesian approach to learning, which means that the training of the algorithm takes place in a Bayesian framework. It was first introduced by M.E Tipping in 2000 [20] as a competitor to the state of the art Support Vector Machine. The RVM is also named by analogy to Support Vector Machine method which is also a sparse kernel model.

The Support Vector Machine (SVM) is a technique for regression and classification, combining excellent generalization properties using the kernel trick[1] representation. The algorithm is very attractive because of its sparseness properties. The formulation of the SVM will not be presented in this section since its functional form is similar to the RVM form which is detailed later in this chapter. The SVM is a leading approach for pattern recognition and machine learning, it makes predictions in term of linear combination of kernel functions with weights over the functions and those functions based on a subset of the dataset.

The basic classification SVM creates a maximum-margin[2] hyperplane[3] that lies in a transformed input space. Given training examples labeled either "yes" or "no", a maximum-margin hyperplane splits the "yes" and "no" training examples, such that the distance from the closest examples (the margin) to the hyperplane is maximized. If there exists no hyperplane that can split the "yes" and "no" examples, a soft margin method will choose a hyperplane that splits the examples as best as possible.

Applying the kernel trick to maximum-margin hyperplane was suggested by [2]. This causes the linear algorithm to operate in a different space. A version of a SVM for regression was proposed in 1997 [24] and named Support Vector Regression (SVR). Formula (2.3.1.1) presents the final model of the SVR.

$$y(x) = \sum_{n=1}^{N} w_n * k(x, x_n) + \varepsilon \tag{2.3.1.1}$$

Here $\mathbf{w}$ is the model weights vector, $\mathrm{k}(\mathrm{x}, \mathrm{x}_\mathrm{n})$ are the kernel functions centered on each of the training support points and ε is the noise term. Note that the final model is based on only a subset of the data.

2.3.2 Limitations of the Support Vector Machines

Though the SVM is a success, it still suffers from several limitations.

1. The predictions are not probabilistic. In regression the SVM output a point estimate and in classification a hard binary decision while ideally we wish to catch uncertainty in our prediction thus we have to estimate the conditional distribution

[1] The kernel trick is taking the observation and gives its representation on a desired function. A linear algorithm can easily be transformed into a non-linear algorithm using this technique

[2] Maximum-margin hyperplane is a hyperplane which separates two groups of points with equal distance from the two. The margin between the hyperplane and the groups is thus maximal.

[3] Hyperplane is a linear projective subspace for separating groups. In a two-dimensional space, a hyperplane is a line. In a one-dimensional space, a hyperplane is a point

$p(t \mid x)$, the probability to get some value t for a new input observation x. Usually the SVM obtain this probability via cross validation.

2. The output of the SVM is considered to be sparse, which means there are fewer kernel function at the end comparably to the number it started with. However, the number of the support kernel functions grows linearly with the size of the dataset.
3. Beside the kernel parameters, there are additional two parameters in the SVM that should be optimized before using the SVM, these optimizations require cross validation on a training data thus they are wasting data and computation.
4. The kernel function that the SVM uses must satisfy Mercer's condition, which states that any positive semi-definite kernel $k(x, x')$ can be expressed as a dot product[4].

2.4 The Relevance Vector Machine

The Relevance Vector Machine is a method for training a generalized linear model (GLM) such as (2.3.1.1). The algorithm starts with a set of priors over the weights we introduced at (2.3.1.1), and governed them by a set of hyper parameters[5] one for each weight. Unlike the SVM the remaining weights are prototype of the data and are not associated with examples close to the decision boundary. Those examples get the name "relevance vectors". In the RVM sparsity is obtained in a more extensive way than in the SVM. Most of the posteriors distributions over the weights are close to zero and leave only the weights for the relevance vectors. The RVM can also use the kernel trick in order to apply a pure linear model to the transformed space. Although, the RVM does not restrict the kernel function, all the implementations we know about use Gaussian Kernel function.

2.5 Formulation of the Relevance Vector Regression

The goal of the RVM is to accurately predict the target function, while retaining as few basis functions as possible in $\boldsymbol{w}$. The sparseness is achieved via the framework of sparse Bayesian learning and the introduction of an additional vector of hyper

[4] Dot product also known as scalar product, it is a binary operation which takes two vectors and returns a scalar quantity. It is a standard inner product of the Euclidian space

[5] Hyper parameters-when we create a parameter for controlling another parameter then the first one is named hyper parameters, in the RVM the α vector is a vector of hyper parameters for the w vector; the vector of the weights.

parameters α_i that controls the width of a Normal prior distribution over the precision of each element w_i.

$$p(w_i \mid \alpha_i) = \sqrt{\frac{\alpha_i}{2\pi}} \exp(-\frac{1}{2}\alpha_i w_i^2) \tag{2.5.1}$$

To complete the Bayesian specification, a Gamma distribution is used as a prior over the hyper parameters α_i - a standard choice for non-informative priors over Gaussian width parameters. The choice of its parameters is of little importance as long as it is sufficiently broad. This form of prior structure results in the marginal distribution over ***w*** being a product of Student-t distributions, and thus favors sparse solutions that lie along the hyper-spines of the distribution. A large hyper parameter α_i indicates a prior distribution sharply peaked around zero. For a sufficiently large α_i, the basis function is deemed irrelevant and w_i is set to zero, maximizing the posterior probability of the parameters' model (2.5.2). The non-zero elements of ***w*** are called Relevance Values, and their corresponding data-points are called Relevance Vectors (RVs).

$$p(\mathbf{w} \mid \boldsymbol{\alpha}) = \prod_{i=0}^{N} N(w_i \mid 0, \alpha_i^{-1}) \tag{2.5.2}$$

Approximate analytical solution for the RVM can be obtained via the Laplace method [21]. Consider a dataset of inputs-target pairs, $\{x_i, t_i\}_{i=1}^{N}$ where targets are assumed, *jointly* Normal distributed - $N(\mu, \Sigma)$, and (μ, Σ) are the unknowns to be determined by the algorithm. Each target t_i is also assumed Normally distributed with mean $y(x_i)$ and uniform variance σ^2 of the noise ε, thus $p(t \mid x) = N(t \mid y(x), \sigma^2)$. The conditional probability of the targets given the parameters and the data can now be expressed as:

$$p(\mathbf{t} \mid \mathbf{w}, \sigma^2) = (2\pi\sigma^2)^{-\frac{N}{2}} \exp\left\{ -\frac{1}{2\sigma^2} \| \mathbf{t} - \Phi\mathbf{w} \|^2 \right\} \tag{2.5.3}$$

where the data is hidden the $N \times N$ kernel function matrix $\mathbf{\Phi}$ representing all the pairs $\phi_{i,j} = k(x_i, x_j),\ i, j \in [1, \ldots, N]$. ($\mathbf{\Phi}$ could be extended to include a possible bias term). With the prior (2.5.2) and the likelihood (2.5.3) the posterior distribution over the weights using Bayes rule is:

$$p(\mathbf{w} \mid \mathbf{t}, \boldsymbol{\alpha}, \sigma^2) = \frac{p(\mathbf{t} \mid \mathbf{w}, \sigma^2) p(\mathbf{w} \mid \boldsymbol{\alpha})}{p(\mathbf{t} \mid \boldsymbol{\alpha}, \sigma^2)} \sim \frac{|\Sigma|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(w-\mu)^t \Sigma^{-1}(w-\mu)\right]}{(2\pi)^{\frac{N+1}{2}}} \qquad (2.5.4)$$

The unknown mean and covariance (μ, Σ) are computed as:

$$\boldsymbol{\Sigma} = (\boldsymbol{\Phi}^{\mathrm{T}} \mathbf{B} \boldsymbol{\Phi} + \mathbf{A})^{-1} \qquad (2.5.5)$$

$$\boldsymbol{\mu} = \boldsymbol{\Sigma} \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{B} \mathbf{t}$$

Where $A \equiv diag[\alpha_1, \ldots, \alpha_N]$ and $B \equiv \sigma^{-2} I_{NxN}$. By integrating out the weights ***w***, we can obtain the marginal likelihood for the hyper-parameters

$$p(\mathbf{t} \mid \boldsymbol{\alpha}, \sigma^2) = \frac{\left|B^{-1} + \Phi A^{-1} \Phi^T\right|^{-1/2}}{(2\pi)^{N/2}} \exp\left\{-\frac{1}{2} \mathbf{t}^T (\mathbf{B}^{-1} + \boldsymbol{\Phi} \mathbf{A}^{-1} \boldsymbol{\Phi}^T)^{-1} \mathbf{t}\right\} \qquad (2.5.6)$$

The solution is derived via the following iterative type II maximization of the marginal likelihood (2.5.6) with respect to (α, σ^2)

$$\alpha_i^{new} = \frac{1 - \alpha_i \Sigma_{ii}}{\mu_i^2} \qquad (2.5.7)$$

$$(\sigma^2)^{new} = \frac{\|\mathbf{t} - \boldsymbol{\Phi} \boldsymbol{\mu}\|^2}{N - \sum_{i=1}^{N} (1 - \alpha_i \Sigma_{ii})}$$

The basic RVM algorithm cycles between (2.5.7) and (2.5.5), reducing the dimensionality of the problem when any α_i larger than a defined threshold. The algorithm stops when the likelihood (2.5.6) ceases to increase.

As described above the RVM is maximizing the product of the marginal likelihood over the priors $\boldsymbol{\alpha}$ and the noise variance σ^2. Equivalent procedure to that which has been described above could be the maximization of the log of the marginal likelihood over $\log \alpha$ and $\log \frac{1}{\sigma^2}$.

$$\log p(\mathbf{t} \mid \log \boldsymbol{\alpha}, \log \frac{1}{\sigma^2}) + \sum_{i=0}^{N} \log p(\log \alpha_i) + \log p(\log \frac{1}{\sigma^2}) \qquad (2.5.8)$$

Equivalently the likelihood can be written as:

$$
\begin{aligned}
L = -\frac{1}{2}\left[\log|\sigma^2\mathbf{I}+\mathbf{\Phi}\mathbf{A}^{-1}\mathbf{\Phi}^T| + \mathbf{t}^T(\sigma^2\mathbf{I}+\mathbf{\Phi}\mathbf{A}^{-1}\mathbf{\Phi}^T)^{-1}\mathbf{t}\right] + \\
\sum_{i=0}^{N}(a\log\alpha_i - b\alpha_i) + c\log\frac{1}{\sigma^2} - \frac{d}{\sigma^2}
\end{aligned}
\qquad (2.5.9)
$$

Where $p(\log\alpha) = \alpha p(\alpha)$. Note that the sum term in (2.5.9) is vanish when $a = b = c = d = 0$. Tipping [21] mentioned that in practice maximizing the log likelihood leads to a better convergence. Also, maximizing the log-likelihood is more convenient since we assumed uniform hyper priors ($a = b = c = d = 0$) over logarithmic scale and the derivatives of the prior terms disappear in this space.

The re-estimation rule for any α_i, the first term in (2.5.7), is obtained via the derivative of (2.5.9) with respect to $\log\boldsymbol{\alpha}$, setting it to zero and solving for α_i.

The re-estimation rule for σ^2, the second term in (2.5.7), is obtained via the derivative of (2.5.9) with respect to $\log\frac{1}{\sigma^2}$, setting it to zero and solving for σ^2.

In the implementation of the RVM we used the log-likelihood procedure.

2.5.1 The Learning Algorithm in Brief

1. Create the kernel matrix – $\mathbf{\Phi}$
2. Create the weights vector – $\mathbf{w}$
3. Create the α vector – $\boldsymbol{\alpha}$

Iteratively:

1. calculate $\mathbf{\Sigma}$ matrix: $\mathbf{\Sigma} = [\mathbf{\Phi}^T\mathbf{\Phi} + diag(\boldsymbol{\alpha})]^{-1}$
2. Calculate $\boldsymbol{\mu}$ vector: $\boldsymbol{\mu} = \mathbf{\Sigma}\mathbf{\Phi}^T\mathbf{t}$
3. Re-estimate $\boldsymbol{\alpha}$ and σ^2 for every iteration (equation 2.5.7)

After we calculate $\boldsymbol{\alpha}$ and σ^2 it is easy to make predictions because the results can readily be computed as:

$$p(t_* \mid \mathbf{t}, \alpha_{MP}, \sigma^2{}_{MP}) = N(t_* \mid y_*, \sigma_*^2) \qquad (2.5.1.1)$$

where $y_* = \mu^T\Phi(x_*)$ and $\sigma_*^2 = \sigma_{MP}^2 + \Phi(x_*)^T\Sigma\Phi(x_*)$.

2.5.2 Priors in the RVM

The RVM contains several initial parameters, not considering the parameter of the kernel function that need to be tuned i.e. the width in the Gaussian kernel function. Let us consider the initial parameters and their hierarchical structure. First of all we have to initialize the $\mathbf{w}$ vector, the vector of the regression coefficients; we set this vector to vector of ones at the beginning. Then, there is the $\boldsymbol{\alpha}$ vector which is the vector of hyper parameters for the $\mathbf{w}$ vector, this $\boldsymbol{\alpha}$ vector is initialized from a Gamma distribution, thus, we need two additional values for the Gamma distribution. We denote these values as *a* and *b*. The $\mathbf{w}$ vector as well as the σ^2 (noise variance) is affecting the targets we wish to predict. The noise variance is also initialized from a Gamma distribution, so we need two additional parameters for this Gamma distribution. We denote these parameters as *c* and *d*.

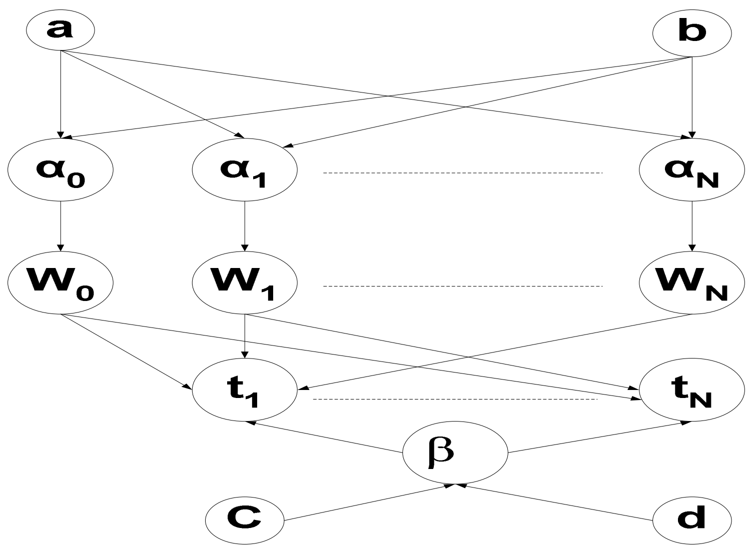

Figure 2.5.2.1 The graphical model of the priors, actually a, b, c, d affecting the rest of the parameters, w is initialized to ones at first

Many papers consider the best parameters to starts with, i.e. there are several suggestions for the best initial $\boldsymbol{\alpha}$ vector - the prior over the weights. Some suggest that $\boldsymbol{\alpha}$ should be a random vector and others suggest it should be estimated from the number of the inputs observations. The $\mathbf{w}$ vector is usually initialized to the ones vector and a, b, c, d are setting to zero in order to obtain non-informative uniform hyper priors over a logarithmic scale.

2.5.3 The Sparseness Perspective

The RVM is a specific sparse scenario of a Gaussian Process (GP). In GP regression framework [15, 17], given a training set of N input-output pairs $\{x_i, t_i\}_{i=1}^{N}$, the goal is to compute the predictive distribution of the function values $f(\mathbf{x})$ at the test location $\mathbf{x}$. The GP regression is Bayesian, and assumes a Gaussian prior such as $p(\mathbf{f} \mid \mathbf{x}_i) \sim N(0, \mathbf{\Sigma})$ where $\mathbf{\Sigma}$ is the covariance matrix created via some covariance kernel function k such as $\Sigma_{ij} = k(x_i, x_j)$. The GP treats the function values $\mathbf{f}$ as random variables, indexed by the corresponding input $\mathbf{x}_i$ and builds the model based only on the conditional of the outputs given the inputs. The model of the targets is $t_i = f(\mathbf{x}_i) + \varepsilon_i$ where $\varepsilon_i \sim N(0, \sigma^2)$ and $f(\mathbf{x}_i)$ is a set of basis functions positioned on the input observations x_i. There are no weights over those basis functions values, thus, a typical GP solution involves a high number of basis functions, somewhat close to the number of examples in the training set, thus GP in general are not considered to be sparse models.

As presented in (2.5.2) in the RVM there is a Gaussian prior for each parameter w_i in the vector of parameters $\mathbf{w}$. We know that this prior is mainly responsible for the sparse results. But how does sparsity occur? Why do most of the elements in $\mathbf{w}$ become zero? The answer to these questions is hidden in the hyper-priors. Each element of the hyper-prior controls the width of the prior over the corresponding weight. Integrating out the hyper priors result in the product of student-t distribution which is highly non-convex and takes the form $p(\alpha) = |\alpha|^{-v}$ for each prior, where v is the hyper prior parameter. The log likelihood framework provides an equivalent regularization penalty of the form $R(\alpha) = v \log|\alpha|$. If α is large, the regularization force $\frac{dR}{d\alpha} = \frac{v}{|\alpha|}$ is small thus the hyper prior has little effect on α. Conversely if α is small the regularization force becomes more significance. During optimization many priors approach infinity and their hyper prior no longer able to hold the derivative at nonzero value against this force. Thus, from (2.5.2) it is clear that the corresponding w will becomes zero.

Another perspective of the sparsity is that of the penalty function that attempts to minimize the mean of the distribution (2.5.3). Compare (2.5.3.1) to (2.2.5). $A \equiv diag[\alpha_1,\ldots,\alpha_N]$ is a regularization vector that encourages sparse solutions.

$$\frac{1}{2\sigma^2}\|t-\Phi w\|^2 + \tfrac{1}{2}w^T A w \qquad (2.5.3.1)$$

2.5.4 The Variational Relevance Vector Machine

The RVM is based on using the Type 2 "maximum likelihood" to estimate the hyper parameters that rules the model Sparsity. M. E. Tipping and C.M. Bishop developed another version of the RVM, the variational RVM [19]. In that version they developed a variational inference and provide a posterior for the parameters and the hyper parameters as well. By specifying the entire parameters and the hyper parameters of the model they implement a fully Bayesian framework. The results of using the variational inference are little bit better than the regular RVM which uses the type 2 "maximum likelihood". However, it is much more computationally expensive, thus, can only be used for limited size problems.

2.5.5 Example of Relevance Vector Regression

Let us look on an example of the relevance vector regression adapted from Tipping's web site [27].

Assume we have 100 observations taken from the function $\sin|x|/|x|$ where $x \in [-10,10]$ with added Gaussian noise of variance 0.1. Figure 2.5.5.1 illustrates the noisy sinc data and the sinc function.

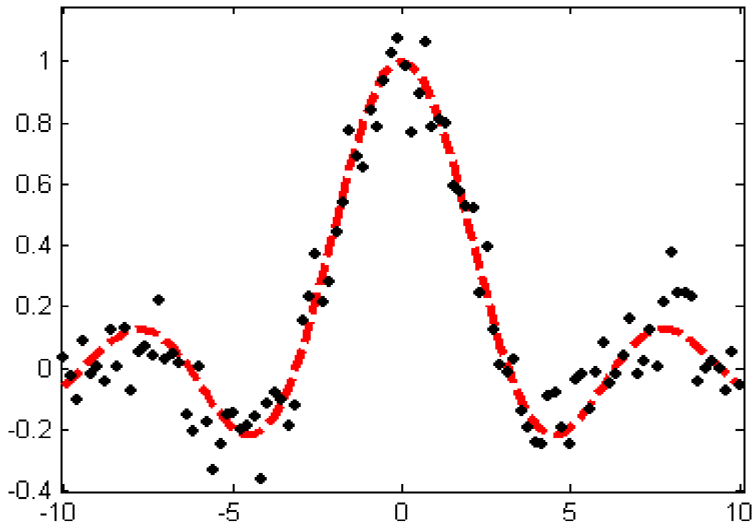

Figure 2.5.5.1 The dashed line is the real function of the data

Figure 2.5.5.2 illustrates the solution of the RVM to the sinc problem. The RVM predictor consists only of 6 relevance vectors.

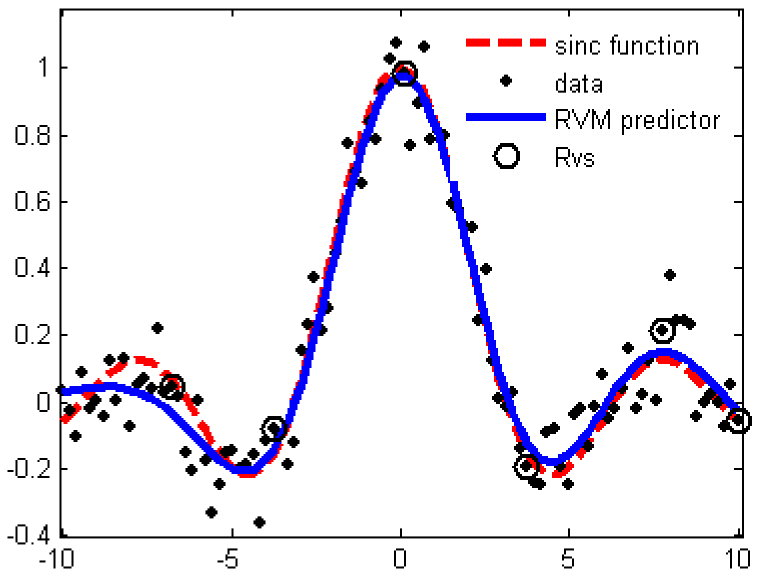

Figure 2.5.5.2 The RVM predictor, the function that the RVM learned is the solid line. Also we can see that the relevance vectors which circled in black are prototypes of the whole data.

We can see the extraordinary fitting of the predictor to the origin function also the smoothness along the spines of the RVM predictor.

2.6 Usability and Applications of the RVM

The RVM has been gaining popularity in regression and classification, due to its attractive features and the sparse results.

1. Statistical modeling and inference problems with sample sizes substantially smaller than the number of available covariates are known as "large p small n problems". In [14] they have made a comparison between the SVM and the RVM solving these types of problems. They apply a model for prediction of blood glucose concentration in diabetics. Their results show that the RVM achieve sparser models than the SVM with comparable accuracy.

2. In [1] they apply the RVM for creating a 3 dimension image from silhouettes.

3. In [13] they use the RVM for visual classification in humans. Frontal views of human faces were used for a gender classification task. They compare the result of human face classification to the results of the machine learning algorithms, SVM and RVM. Their conclusions were that the machine learning algorithms reached

better results and in the future it is possible to classify human gender through a machine learning algorithms such as RVM.

4. The RVM has been used for real time tracking [25]. The goal of the experiment was to recognize spatial movement and perturbations. They connected a camera to the computer and photographed a direct object which was human face, than passed the data of the optical flow to the computer. For recognizing the movement of the human face they used the RVM regression, which turns out to be the most efficient way of doing so.
5. The RVM has been useful for time series prediction too. In [3] they have made a few changes to the RVM by controlling the hyper parameters and the parameter of the basis function as well. They choose a hard prediction problem, the MacKey-Glass chaotic time series, which is well known for its strong non-linearity. They demonstrated the usability of the RVM for time series prediction.

We can see that the RVM algorithm becomes popular and many researchers use it for problems beyond the ones mentioned above, i.e. signal processing and more.

2.7 Advantages of the RVM

The RVM is gaining popularity not only because of its generalization properties and the accurate prediction but also because of the following reasons:

1. The RVM is a fully probabilistic model, its predictions are probabilistic (formula 2.5.1.1). It can describe uncertainty in the predictions and the conditional distribution $p(t \mid x)$. There is no need in cross validation for evaluating the accuracy of the results.
2. Thorough the framework of sparse Bayesian learning sparsity is achieved much more successfully compared to other similar algorithms such as SVM.
3. The RVM is capable of dealing with highly dimensional problems, which means it can solve very difficult problems. Note that it does not deal with problems of hidden variables.
4. While using the RVM there are no restrictions on the kernel we can use. Actually the RVM can a produce a linear model without any kernel. However, the usability of the kernel already potentially improves the accuracy.

5. For most of the methods for training a GLM, require several parameters to be tuned. The RVM requires none (except the kernel parameters).

2.8 Limitations and Disadvantages of the RVM

Despite the advantages of the RVM and its attractive features, the algorithms still suffers from several limitations.

1. The most computational intensive part in the algorithm is the matrix inversion operation in (2.5.5), which requires $O(N^3)$ operations. The computation of the error term in equation (2.5.7) requires $O(N^2)$ operations.

2. The matrices $\mathbf{\Phi}$ and $\mathbf{\Sigma}$ in (2.5.5) are full rank thus the storage complexity of the algorithm is $O(N^2)$. Practically the storage complexity is larger due to additional matrices and vectors that we need during computation. Dealing with large data sets, this fact prohibits training the RVM on a computer with a moderate amount of memory, though most of the computers can swap memory with the disk via virtual memory algorithms.

These two complexity issues make the basic RVM algorithm impractical for moderately sized problems.

3. Its solution is not unique. It is influenced by the ordering of the training examples.

4. In order to obtain the best results for a specific problem it is crucial to use a suitable kernel function in the RVM. Kernel functions usually, have some parameter(s) that need to be tuned. How to choose the most suitable kernel function for a specific problem and how to find the best parameter(s) for the given kernel remains an unresolved issue for the RVM, as it is an unresolved issue for all the other kernel based methods.

5. The algorithm stops when the likelihood is maximal. However, it is possible that the maximum is local and not global, thus, we may stop the convergence too early. This can lead to a sub optimal solution.

3. PROBLEMS STATEMENT

The *problems statement* chapter describes in details the objectives of this study and the motivation behind them

3.1 Accelerating the Relevance Vector Machine

The primary goal of this research is to develop algorithms based on the RVM that are capable of solving large scale problems in reasonable time. As we discussed in chapter 2 the most expensive operations in the RVM from a complexity point of view are:

1. The matrix inversion operation of $\mathbf{\Sigma}$ in (2.5.5). The complexity of a matrix inversion operation is known to be of the order of $O(N^3)$, where N is the number of candidate basis functions. At the beginning of the algorithm all the input observations are considered to be candidates as a relevance vectors thus this matrix is initially of full rank.

2. The error computation $\|\mathbf{t}-\mathbf{\Phi\mu}\|^2$ in (2.5.7), in order to calculate the noise variance σ^2. The run time complexity is of the order of $O(N^2)$ where N is again is the number of candidate basis functions.

While dealing with large scale problems (several thousands) with a regular desktop computer, these two computations operations can take hours or even days sometimes.

Another limitation of the RVM is its storage complexity. In the formulation of the RVM we are actually using a bivariate kernel function centered on each one of the N training data points in order to represent the data. We called this matrix $\mathbf{\Phi}$, and its storage complexity is of the order of $O(N^2)$. Dealing with a large scale problem we need a large continuous block of memory, otherwise the computer can start swapping variables with the disk and this cause the algorithm to run much slower than if we had enough memory.

There are two possible strategies for runtime complexity reduction:

1. Rather than use the exact matrix operation at a cost of $O(N^3)$, use an approximation at a cost of $O(N^2)$

2. Iterative sub sampling the data, thus reducing N.

The first approach is considered in [9] and will not be considered in this dissertation. The second approach is considered by [22]: a sequential algorithm that starts with a single basis function, and considers the inclusion of a new basis function each time the algorithm converges until exhausting the pool of basis functions. As noted by [18], depending on the distribution of the data, there is a risk of deleting a new basis function that lies far from the other training data. Also there is an overhead of manipulating so many times the RVM. Appendix E discusses some aspects of that approach. In this study we will introduce three new algorithms of sequential sub sampling of the data in order to accelerate the RVM without damaging its attractive properties.

3.2 Finding the Best Partition Size

The first goal is to accelerate the RVM via data partitioning. The second goal is to find the size of the partition since it has crucial effect on the time required for training the RVM. There could be two methods for finding the best partition size:

1. Optimize for the size of the partition. That means training the RVM several times and tuning partition size via cross validation. This approach obviously requires more computations, thus it is wasteful. In practical machine learning – which is concerned with finding the best model – it is impractical to tune the partition size.
2. Find a heuristic that obtains a reasonable partition size which obtains reasonable results for every data set.

We show that there is a range for the best partition size, and while we choose a value within this range the results are reasonable. Based on this range we introduce a heuristic that can find a reasonable partition size.

3.3 Finding the Best Algorithm

In this study we will introduce three partition algorithms. We prove that the partition algorithms can significantly reduce the complexity of the RVM while retaining the attractive attributes of the original solution. We also investigate which one of the three partition algorithms is the best.

3.4 Choosing the Kernel Function

The RVM is a kernel based method; which means it uses a kernel function for mapping the data into a high dimensional feature space in order to increase the computational power of the linear machine [7]. Although it is possible to use the RVM without kernel function, the necessity of the kernel function has already been shown in the RVM [21]. The most common use of the kernel in the RVM is the Gaussian kernel because of its general properties and the ability to smooth the function. It has been suggested [11] that for statistical parametric techniques the Matern kernel [17] which is another type of kernel function probably will obtain better sparsity than the Gaussian. Choosing a good kernel for a task depends on intuition and experience. In high-dimensional tasks, where prior knowledge is unlikely, the best option may be to explore simple combinations of the standard kernels [11]. However, we wish to find whether the Matern kernel is better than the Gaussian for the RVM, and also examine the sensitivity of the RVM to the kernel choice.

3.5 Benchmarking the RVM

The RVM is a new machine learning algorithm and has some attractive features. Algorithms function differently for different problems - depending on the properties of the data sets. We expect the learning method to have strengths and weaknesses that depend on the particular types of the task. A machine learning method may find a good model for a complicated non-linear problem, while may overfit noisy data or it may work well for a low-dimensionality problem while being sensitive to the dimensionality of the problem. We wish to investigate how the RVM performs for non-linear, noisy or high dimensional dataset.

3.6 Discussion

The basic RVM algorithm has some very attractive features; however, in its original form it is too complex for any realistic medium or large datasets. This study aims to develop algorithms that are capable of handling large scale problems. The RVM algorithm is new and some of its essential features – such as proper selection of basis functions – are not known. Resolving those questions will provide another essential step for popularizing the RVM algorithm, so that eventually it will turn into another "standard tool" for the practitioner of machine learning.

4. ACCELERATING THE RELEVANCE VECTOR REGRESSION

This chapter has several goals. The first one is to introduce three acceleration algorithms that are capable of handling large data sets in reasonable time via partitioning of the problem. The second goal is to present heuristics for finding a decent partition size. The last goal of this chapter is to determine the most efficient partition algorithm out of the three we suggested.

As we discussed in the previous chapters, the basic RVM has $O(N^3)$ run time complexity and $O(N^2)$ storage complexity. These two facts prevent using the algorithm on data sets with average size and upwards. There are two possible strategies for complexity reduction:

1. Performing approximate, rather than exact matrix inversion at a cost of $O(N^2)$.
2. Iterative sub-sampling of the data, thus reducing *N*.

The three partition algorithms introduced in this chapter are related to the second strategy. We provide comprehensive description of the three algorithms for the acceleration of the RVM via data partitioning followed by experiments on benchmark data sets. At first we implement the basic RVM and demonstrate some properties of the basic implementation. Experiments demonstrate that the three partition algorithms indeed accelerate the RVM. The second section provides a heuristics for the size of a partition for every dataset. The last section addresses the question of what is the best partition algorithm out of the three.

4.1 Three Data Partitioning RVM Algorithms

The key idea in the following partitioning algorithms is to benefit from the reduced complexity $O(P^3)$ by computing the basic RVM for a partition of a size *P*. Each algorithm considers differently how to merge the RVs resulting from a single partition with the rest of the data. The three acceleration partitions algorithms which are introduced in this section are based on the idea of *divide and conquer*. When the original problem does not fit the computer core memory or it is too complex from a computational point of view (large dataset), the problem is solved sequentially for a subset of the data – the partition – which fits the core memory. These algorithms present an efficient way of partitioning the problem into sub-problems; such that once a sub-problem is solved in the core memory, its solution can be composed as a part of the whole solution of the original problem. Thus, once a sub-problem is solved, its data most likely will not be cycled back in core memory

For a fair comparison of the algorithms, the training set of size *N* was partitioned into the same $M = \lceil \frac{N}{P} \rceil$ partitions of size *P*, where $\lceil \ \rceil$ is the ceiling operator, and the residual partition may be smaller than *P*. The basic RVM algorithm of formula (2.5.5) and (2.5.7) was implemented via MATLAB scripting language, with the following two modifications:

- The error $\|\mathbf{t} - \mathbf{\Phi\mu}\|$ in (2.5.7) was computed over the full training set. The reason is that otherwise the RVM may overfit the data of any single partition, eliminating possibly good basis functions. Note that the complexity of the error computation is $O(P^2)$, and for very large datasets, it may be desirable to compute the error for only a sample of the data, or reduce the update rate of σ^2.
- While training several data sets with the basic implementation of the RVM based on the formulas in section 2.5, we saw that there is a possible computational inefficiency resulting from waiting for the convergence of the basic RVM to the final solution. For example, look in Figure 4.1.1 which presents a typical convergence of the accuracy and the number of relevance vectors as a function of time for the Boston benchmark data; initially, the number of RVs drops very slowly (until 3 seconds), than, it drops very fast (before 6 seconds), and after that, the progress is very slow (after 6 seconds), and the RMSE is practically constant. Thus, [8] suggested not waiting for the convergence of the basic algorithm. Instead, every time the number of relevance vectors drops below a certain low "water mark" stop the training and save the computations time required to finish it.

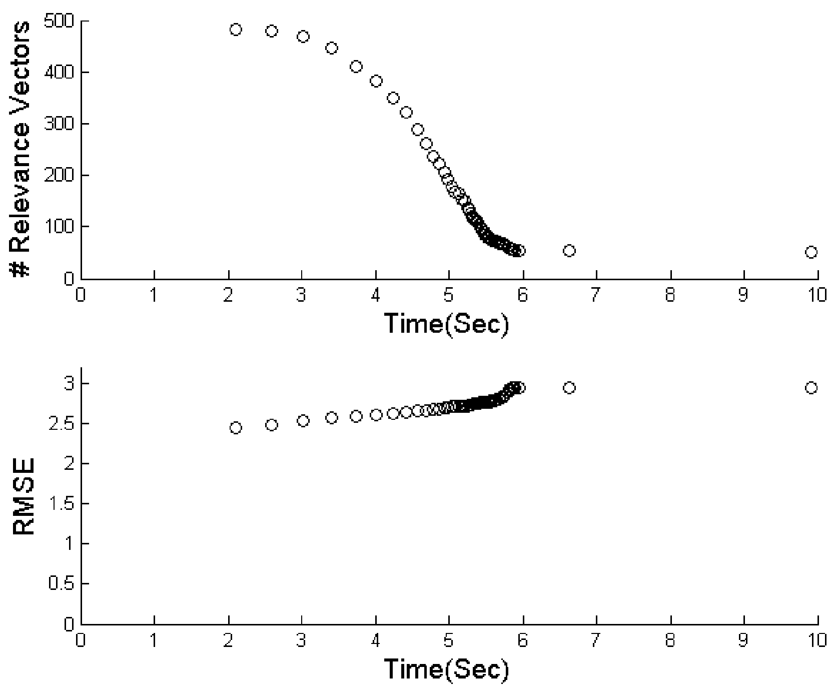

Figure 4.1.1 The number of relevance vectors as a function of time and the RMSE as a function of time for the *Boston* data. Each circle marks the time when the RVM deleted a basis function.

Similar behavior was observed in all other data sets we trained on the basic implementation. This behavior indicates that the RVM is *any time algorithm* and can be very helpful when we intend to solve a large problem via dividing it into several consecutive sub-problems. In intermediate processing of consecutive partitions, there may not be any benefit for waiting for the full and lengthy convergence (consider Figures 4.1.1), when the resulting RVs are to be integrated with the next data partition. Thus it was decided to add a premature convergence flag, so that the algorithm could stop when the number of RVs drops below $P/2$. Only on the last stage of the partition algorithm, the basic RVM is expected to run until full convergence.

In the following sub sections we describe each algorithm and its motivation.

4.1.1 The Split and Merge RVM

The key idea here was taken from the merge sort algorithm. The resulting RVs from each two partitions are merged together into a new partition, whose size may be larger than *P*, and the process is recursively repeated:

Steps:

1. Run the basic RVM on each of the original M partitions and store each set of resulting RVs.
2. Merge each two sets of resulting RVs into $\lceil M/2 \rceil$ new datasets. Run the basic RVM on each one of the new datasets and store each set of resulting RVs. For an odd M, processing the last dataset will be delayed until the next step.
3. Repeat step 2 until obtaining a single dataset.
4. For the last single dataset, let the basic RVM run until full convergence and obtain the solution.

Complexity analysis:

Considering that a sparse solution exists[6], the premature stopping flag will be triggered at $\frac{P}{2}$, and each merge operation (except possibly the last one) will generate another dataset of size P, which takes $O(P^3)$ operations to process. There are approximately $\frac{2N}{P}$ partitions to be processed. Thus the overall complexity is of the order of $2NP^2$. In a multi-processor system, a further speedup can be obtained when each partition is processed on a different processor.

Visualization of the Split and Merge Algorithm:

For a graphical illustration of the split and merge algorithm, consider Figure 4.1.1.1, which presents a case of N=4000 and P=1000.

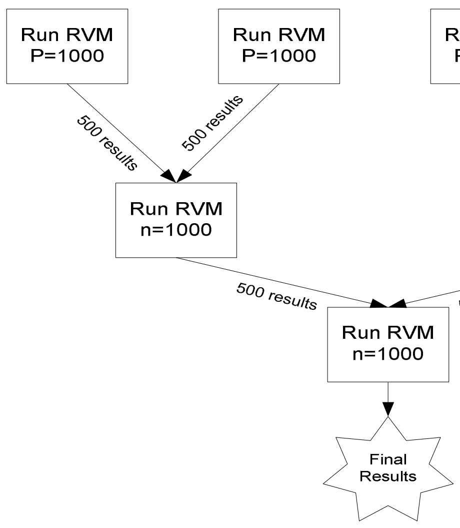

Figure 4.1.1.1 The split and merge algorithm

[6] Otherwise, the kernel function is probably not optimal, and numerical instability might be expected. Further details regarding numeric instability available in appendix D.

4.1.2 The Incremental RVM

The idea here is to merge the RVs resulting from each run of the basic RVM with the next partition. The increase in the complexity is gradual from step to step, so that if we exhaust the time or space constraints, we can stop and still have a pretty decent solution obtained from a sample of the dataset.

Steps:

1. Run the basic RVM on the first partition and store the resulting RVs.
2. Merge the resulting RVs with the next partition. Run the basic RVM on the data and store the resulting RVs.
3. Repeat step 2 until obtaining a single dataset.
4. For the last single dataset, let the basic RVM run until full convergence and obtain the solution.

Complexity analysis:

Considering that a sparse solution exists, the premature stopping of the basic RVM on each dataset halves the number of RVs in each step. Due to the merge operations, the size of the dataset gradually increases to the size of *2P*: $p, \frac{p+2p}{2}, \frac{p+2p+4p}{4}, \frac{p+2p+4p+8p}{8}, \ldots, \cong 2p$ and the complexity of processing the last dataset is $O\left((2P)^3\right)$. There are exactly $\left\lceil \frac{N}{P} \right\rceil$ partitions to be processed. Thus the overall complexity is of the order of NP^2.

Visualization of the incremental Algorithm:

For a graphical illustration of the incremental algorithm, consider Figure 4.1.2.1, which presents a case of *N*=4000 and *P*=1000.

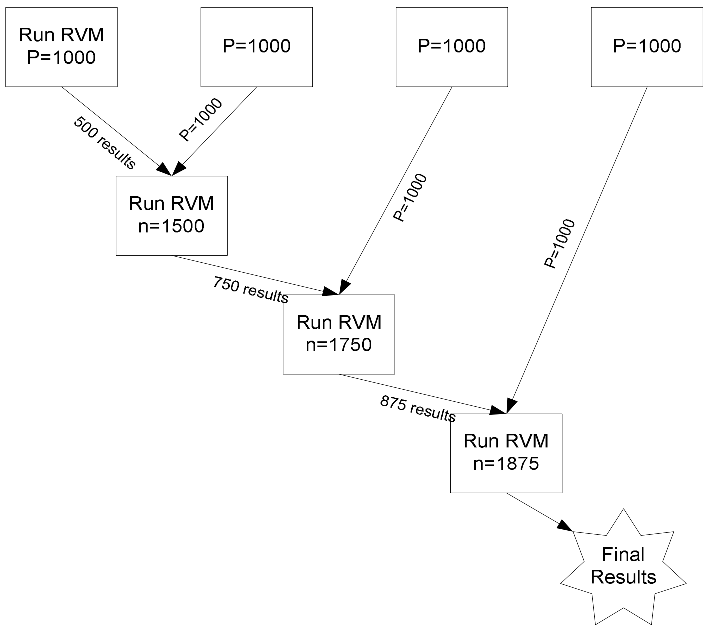

Figure 4.1.2.1 The Incremental algorithm

4.1.3 The Working Set RVM

Here we expand the idea of the incremental RVM; Instead of merging the next partition – which is an arbitrary partition – we merge with a more informative partition that is generated from our best current RVM model. The prediction error, which is evaluated during the basic RVM algorithm, is used to identify the data with the worst prediction error – which is the most informative. The partition of the most informative basis functions of size *P* is used for the next merge operation. The increase in the complexity is gradual from step to step, so if we exhaust the time or space constraints, we can stop and have a pretty decent solution obtained from possibly the most informative sample of the dataset.

Steps:

1. Run the basic RVM on the first partition and store the resulting RVs.
2. Construct a model from the last RVs, predict the error for each one of the remaining unused data, and construct a partition of size *P* from the data which generated the largest prediction error.
3. Merge the current RVs with the newly generated partition. Run the basic RVM on the data and store the resulting RVs.
4. Repeat steps 2, 3 until obtaining a single dataset.

5. For the last single dataset, let the basic RVM run until full convergence and obtain the final solution.

Complexity analysis:

The complexity analysis of this algorithm is the same as that of the incremental algorithm. Though there is a small overhead involved in constructing new partitions on the fly, identifying the best RVs earlier than the incremental algorithm does may enhance the convergence of the next step in the algorithm.

Visualization of the Working Set Algorithm:

For a graphical illustration of the working set algorithm, consider Figure 4.1.3.1, which presents a case of *N*=4000 and *P*=1000.

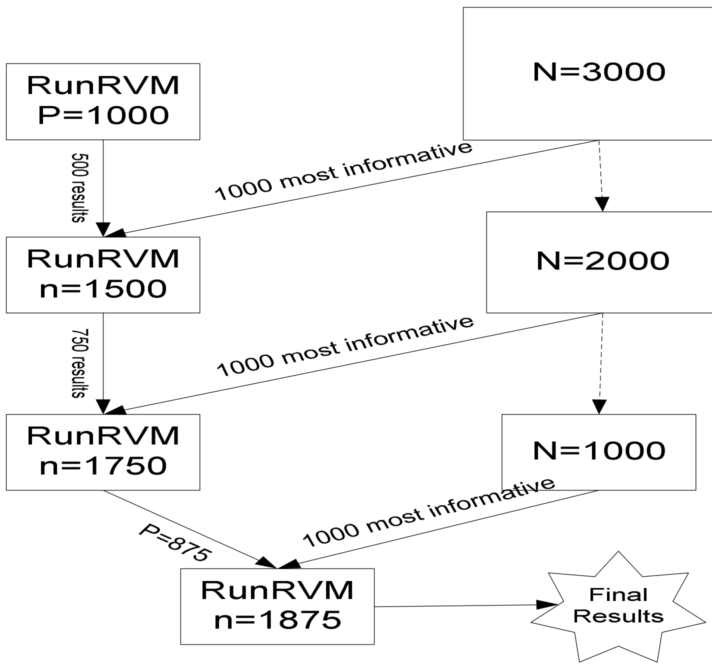

Figure 4.1.3.1 The Working Set

4.1.4 Comparative Experiments with Constant P

After implementing our three partitioning algorithms and the basic RVM we conducted the following experiments with a constant partition size P.

4.1.4.1 Data Sets

In order to validate the partition algorithms we used some popular benchmark datasets for regression methods. Those data sets have been chosen because they are familiar in machine learning, and they represent a wide sample of regression problems. Each data set has a different size (*N*) and different number of input dimensions (# Attributes).

Table 4.1.4.1.1 present details about the datasets and the split of the data sets between the training set and the test set. Further details about these datasets, as well as about the rest of the datasets used in this dissertation, are available in appendix A.

Table 4.1.4.1.1 Details of the benchmark datasets used for the comparative experiments

Name	N	d	Training set /test set
Boston	506	13	481/25
Sinc	2000	1	1000/1000
Friedman#1	3400	10	2400/1000
Abalone	4177	8	3341/836

4.1.4.2 Experiment Description

A training set was used for the learning phase, and the error was measured on the testing set. We trained the four algorithms (the three partitions algorithms and the basic RVM implementation) under the same conditions:

- For the three partition algorithms the partition size used was P=50 to simulate the most interesting case - where P is much smaller than N.
- A Gaussian kernel was utilized and its input width parameter was chosen via 5 fold cross validation over reasonable values with the basic RVM. The same width was used for all the four algorithms. All the width parameters that have been used in this study presenting in appendix F.
- All simulations have been performed on a computer equipped with a Pentium IV 2.42 GHz processor and 768 MB physical memory under the same conditions.
- Each algorithm was simulated for 100 times (10 times for the largest dataset) on different randomizations of the training set. The same randomizations were used for comparing the three heuristics and the basic RVM. Since the algorithms may be influenced by the order of the data we permuted the data for each repetition and used the same order in all four algorithms.

This kind of rigorous comprehensive experiment will definitely indicate whether the three algorithms indeed accelerate the RVM.

4.1.4.3 Evaluation Metrics

For every experiment and comparison we used the following three measures:

- Time - total run time to convergence in seconds.
- RMSE – root mean square error.

- RVs - number of relevance vectors of the final trained model.

4.1.4.4 Results

The following three tables present the comparative RMSE, the comparative number of RVs, and the comparative run time for each benchmark dataset and each algorithm. The standard deviation of each measure is also presented in the tables.

Table 4.1.4.4.1 Comparative RMSE

	BasicRVM	SplitMerge	Incremental	WorkingSet
Boston	3.8 ± 0.9	4.2 ± 1	4 ± 1	3.9 ± 0.9
Sinc	0.01 ± 0.003	0.034 ± 0.002	0.01 ± .002	0.001 ± 0.002
Friedman#1	0.56 ± 0.02	0.69 ± 0.06	1 ± 0.01	0.60 ± 0.02
Abalone	2.1 ± 0.1	2.1 ± 0.1	2.1 ± 0.1	2.1 ± 0.1

Table 4.1.4.4.2 Comparative number of RVs

	BasicRVM	SplitMerge	Incremental	WorkingSet
Boston	53.6 ± 3.2	41.3 ± 4.0	45.7 ± 3.6	43.4 ± 3.2
Sinc	7.8 ± 1.6	7.4 ± 0.7	7.6 ± 0.9	7.7 ± 0.7
Friedman#1	172.8 ± 7.7	149.5 ± 7.2	149.4 ± 3.5	150.7 ± 6
Abalone	24.6 ± 2.0	21.7 ± 2.4	23.2 ± 1.0	22.1 ± 1.6

Table 4.1.4.4.3 Comparative time to convergence

	BasicRVM	SplitMerge	Incremental	WorkingSet
Boston	4.12 ± 1.8	3.8 ± 1.2	2.9 ± 1.3	2.6 ± 1.4
Sinc	11.12 ± 1.1	1.7 ± 0.2	2.1 ± 0.3	2.4 ± 1.5
Friedman#1	193.7 ± 31.6	140.6 ± 20.1	534.2 ± 86.7	551.7 ± 1.0
Abalone	412.7 ± 18.5	34.1±2.8	19.7 ± 1.1	20.6± 1.6

4.1.4.5 Results Analysis and Discussion

Analysis of the results (especially for the larger datasets) indicates that

- The three heuristics obtained a similar accuracy to that of the basic RVM, (Table 4.1.4.4.1) thus they are correct.
- The number of RVs is also similar (Table 4.1.4.4.2) and possibly smaller than that of the basic RVM, thus retaining the sparseness of the solution.
- The time to convergence (Table 4.1.4.4.3) is much shorter in most cases; except the Friedman#1 dataset. This is due to the number of RVs in the Friedman#1 data set which is much larger than P due to its non-linearity. This important issue will discuss in section 4.2.

- Its difficult the decide which heuristic is the dominant based on the above results. Further experiments on additional datasets are required; section 4.4 is dedicated to this issue.

One motivation for the Working Set algorithm is that it is possible to stop it prematurely and still receive a decent solution. Figure 4.1.4.5.1 demonstrates a typical behavior of the heuristics for the Boston dataset during its run time. The figure presents the number of RVs as a function of the training time and the root mean square error as a function of the training time respectively for the four algorithms. As expected, RMSE of the Working Set algorithm drops and stabilizes very fast, and the number of "potential" RVs also drops very fast. Similar behavior was detected for the larger Abalone dataset. Thus, for very large datasets (that run too slow), premature stopping of the partition algorithms will indeed provide a decent solution.

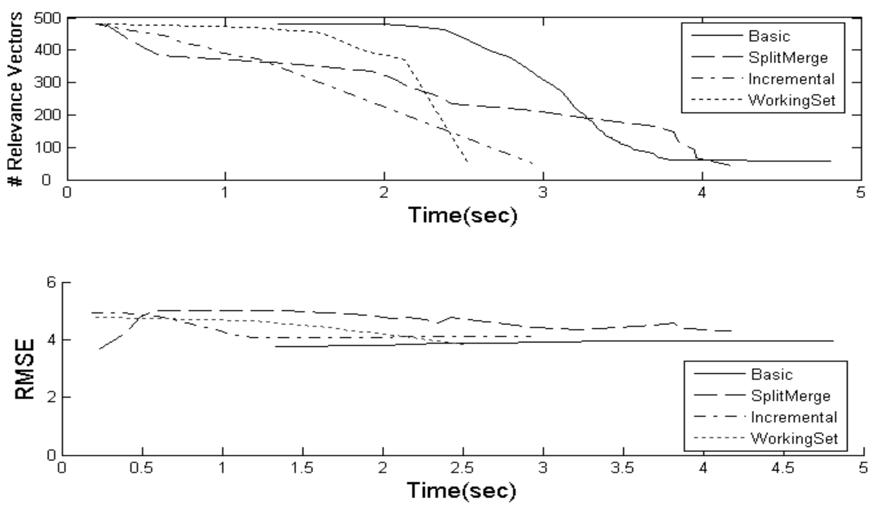

Figure 4.1.4.5.1 Comparative accuracy (on the training set) and the number of RVs over the run time of the heuristics for the Boston dataset.

Following these results we can see that the partition algorithms indeed accelerate the basic RVM without damaging the accuracy or the sparseness of the solution. However, the total time required for training the three algorithms with the Friedman#1 data set is prompt some question mark on the ability of the algorithms to accelerate the RVM for every data set. The experiment results in table 4.1.4.5.1 show us that the size of a partition has a crucial effect on the time required for training those models. In the previous experiments the partition size was fixed to 50, but if we had chosen different partition size, larger than the final number of relevance vectors, the results of the three algorithms would have been much better.

Table 4.1.4.5.1 The results of the Friedman #1 data sets if we choose P=400. We can see the big improvement in the time required for training the three algorithms comparably to the previous experiment.

	BasicRVM	SplitMerge	Incremental	WorkingSet
RMSE	0.56 ± 0.02	0.6±0.1	0.53±0.05	0.54±0.04
# RVs	172.8 ± 7.7	151±8	172±7.4	172±7.1
Time	193.7 ± 31.6	52±6.1	50±7.3	59±5.5

4.2 Estimating the Partition Size P

The purpose of this section is to provide heuristics that can select a decent partition size for every dataset. As exemplified with the Friedman#1 dataset, in Table 4.1.4.4.3 there is no saving in the run time when the number of RVs in that specific dataset was about three times larger than P. In general, when the number of RVs (which is not known in advance) is much larger than P, there will not be any premature convergence of the RVM for any partition, the number of RVs found in a partition will likely be larger than $P/2$, and the merge mechanism in the heuristics will grow the size of the datasets, until it is sufficiently large to capture the number of RVs in the final solution. The overhead of this growth process may slow the heuristics even more than the basic RVM, as exemplified with the Friedman#1 dataset, in Table 4.1.4.4.3.

The partition size *P* has a strong impact on the runtime of the three algorithms described in section 4.1. A too large partition is inefficient in terms of computational complexity and space complexity. On the other hand, a too small partition will not only generate a growth of the merged partitions, but will also require multiple error calculations in (2.5.7). The error computation requires $O(N^2)$ operations, so there is an overhead in manipulating many partitions. Therefore, there seems to be an optimal *P*. Unfortunately, we do not know in advance the optimal *P*, and in practical machine learning – which is concerned with finding the best model – it is impractical to optimize for P.

The number of RVs depends on the non-linearity of the data (e.g. the Friedman#1 is very non-linear), the dimensionality of the problem, and the kernel functions. From many experiments we performed it seems that the best results are obtained when P is about twice the number of RVs. Obviously, the number of RVs can not be determined in advance. Yet, for very large datasets, it is important to have a good guess regarding the partition size. The next sub-section will present heuristics for guessing a decent P.

4.2.1 The Partition Size Heuristics

The main idea of the heuristics presented here is to compute a *clue* regarding the complexity of the dataset. The more complex a dataset is, the more RVs are expected, and the larger P should be. The clue is computed by running the simple RVM on a subset of the data. A sparse solution for the subset indicates that it is highly likely that the solution for the full dataset is also sparse, and that P may be smaller than the size of the subset. However, if the solution for the subset is not sparse, it indicates that P could be at least as large as the size of the subset. Following are the details of the heuristics we tested in the next section.

Steps:

1. Randomly construct a subset of size $\sqrt{N \cdot d}$ from the dataset, where N denotes the size of the dataset and d is the number of dimensions.
2. Train the basic RVM on the subset and count the number of relevance vectors (#RVs). Compute the sparseness ratio: $\frac{\#RV}{\sqrt{N \cdot d}}$
3. $P = \begin{cases} 0.2\sqrt{N \cdot d} & if \quad \frac{\#RV}{\sqrt{N \cdot d}} \le 0.25 \\ 0.6\sqrt{N \cdot d} & if \quad 0.25 < \frac{\#RV}{\sqrt{N \cdot d}} \le 0.75 \\ \sqrt{N \cdot d} & if \quad \frac{\#RV}{\sqrt{N \cdot d}} > 0.75 \end{cases}$

Note that our results (as indicated in table 4.2.2.2.1 in the next subsection) confirm that when P is about twice the number of RVs in the final solution, the training time is short. The threshold values (0.25, 0.75) and the multipliers values (0.2, 0.6, and 1) were chosen based on extensive experimentation which will not detail here. The top branch represents the sparse case, and the bottom branch represents a complex dataset, where there would be many RVs in the final solution.

Note that our experience indicates that an excessive number of RVs is often associated with an improper kernel function thus, when the sparseness ratio $\frac{\#RV}{\sqrt{N \cdot d}}$ is close to 1, it advisable to further optimize the kernel function.

4.2.2.1 Complexity Analysis

The partition size heuristics using $\sqrt{N \cdot d}$ samples from the data and the run time complexity of the basic algorithm is $O(N^3)$ because the inverse matrix operation, thus the run time complexity of the heuristic is $O(N^{1.5})$. Clear representation of the complexity analysis of the heuristics is as follow: $O\left(\left(\sqrt{N \cdot d}\right)^3\right) = O\left(d^{1.5} \cdot N^{1.5}\right) = O(N^{1.5})$.

For the analysis of the overall runtime complexity of the partition algorithm combining the usage of the partition size heuristics assume we are using the split and merge algorithm. As we described in sub-section 4.1.1 the run-time complexity of the split and merge is $2NP^2$. In the worst case scenario, the size of P which generated via the heuristics is $\sqrt{N \cdot d}$. Using the split and merge algorithm, there are *exactly* $\frac{2N}{P}$ partitions to be process. Thus, the overall runtime complexity of the split and merge while using the heuristics partition size is $O(N^2)$.

Where $P = \sqrt{N \cdot d}$ a clear representation of the overall complexity analysis is as follow:

$$O\left(\frac{2N}{P} \cdot (P)^3\right) = O\left(\frac{2N}{\sqrt{N \cdot d}} \cdot (\sqrt{N \cdot d})^3\right) = O\left(2N \cdot (\sqrt{N \cdot d})^2\right) = O(2dN^2) = O(N^2)$$

4.2.2 Experiments with the Partition Size Heuristics

After implementing the heuristics, we conducted the following experiments with the working set algorithm.

4.2.2.1 Data sets and Evaluation Metrics

In order to validate the partition heuristic defined in the previous subsection, we used four data sets from the *Pumadyn* family of datasets. These datasets are variations of the same system; a realistic simulation of the dynamics of a Puma 560 robot arm. This family contains eight data sets; each and every one of them has its own characteristics regarding three properties: the level of non-linearity, the noise in the data and its dimensionality. The following table describes the datasets which were used for validating the partition size heuristic and for the experiments in section 4.4. For further details on this family of data sets look in appendix A.

Table 4.2.2.1.1 Details of the Pumadyn datasets used to validate the heuristic and the experiments in section 4.4.

Name	Size	# of Attributes	Level of noise	Level of non linearity	Training set /test set
8fh	8192	8	high	fairly linear	6144/2048
8fm	8192	8	moderate	fairly linear	6144/2048
8nh	8192	8	high	non-linear	6144/2048
8nm	8192	8	moderate	non-linear	6144/2048
32fh	8192	32	high	fairly linear	6144/2048
32fm	8192	32	moderate	fairly linear	6144/2048
32nh	8192	32	high	non-linear	6144/2048
32nm	8192	32	moderate	non-linear	6144/2048

4.2.2.2 Experiment Description and Results

In order to validate that the partition size heuristics can actually obtain a decent partition size we compare its results to the results of a several solutions obtained for different P sizes. The optimal solution of the problem via partition algorithm exists among those sizes of P. The first P was fixed to 25 in order to emphasize the case where P<<N. The following partition sizes were conducted by multiplying the previous P by two (logarithmic scale).

As a partition algorithm we choose the working set. A Gaussian kernel was utilized for all the experiments and its single width parameter was optimized via cross validation. The same measures have been used in this experiment; the time required for training the algorithm, the number of RVs and the accuracy (RMSE). All experiments were conducted on a 3.1 GHz Pentium IV processor equipped with 2 GB of physical memory. Table 4.2.2.2.1 describes the results based on averages from 20 repetitions and the standard deviation is also given in the table.

Table 4.2.2.2.1 The impact of the partition size on the results as measured on four Pumadyn datasets using the Working Set algorithm and the comparative results to the heuristic. The minimal time is underlined.

Puma-8fh	**P=25**	**P=50**	**P=100**	**P=200**	**P=400**	**P=800**	**Heur. P=45**
RMSE	3.2±.05	3.2±.05	3.1±.05	3.1±.05	3.1±.05	3.1±.05	3.1±.04
#RV	36±2.9	44±3.5	47±4.6	38±2.9	42±3.6	47±4.1	38.5±2.9
Time	46.35±5.3	25.96±2.1	24.3±2.9	50.1±3.9	103.2±13.1	217.3±15.2	26.2±2.5
Puma-8nm	**P=25**	**P=50**	**P=100**	**P=200**	**P=400**	**P=800**	**Heur. P=134**
RMSE	1.05±.03	1.06±.02	1.07±.03	1.06±.03	1.05±.03	1.05±.03	1.06±.02
#RV	79±3.2	73±3.4	75±3.1	81±3.4	80±3.4	80±3.2	82.5±2.6
Time	608.8±85	400.5±78.5	63.2±18.5	75.7±11.4	118.6±14.5	231.4±17.6	46.2±3.1
Puma-32fh	**P=25**	**P=50**	**P=100**	**P=200**	**P=400**	**P=800**	**Heur. P=89**
RMSE	0.026± $2*10^{-4}$	0.025± $2*10^{-4}$	0.026± $2*10^{-4}$	0.024± $2*10^{-4}$	0.021± $2*10^{-4}$	0.021± $2*10^{-4}$	0.021± $2*10^{-4}$
#RV	69±8.3	66±8.3	90±11.1	119±13.2	118±12.9	97±11.5	77.8±2.74
Time	46.7±6.6	25.6±5.5	22.9±5.6	29.5±5	54.4±6.6	118.8±11.4	19.11±1.1
Puma-32nm	**P=25**	**P=50**	**P=100**	**P=200**	**P=400**	**P=800**	**Heur. P=89**
RMSE	0.026± $4*10^{-4}$	0.026± $3*10^{-4}$	0.026± $3*10^{-4}$	0.026± $3*10^{-4}$	0.026± $4*10^{-4}$	0.026± $3*10^{-4}$	0.027± $3*10^{-4}$
#RV	38±3.5	42±4.4	64±4.3	68±5.6	69±8.8	54±6.9	66.1±8.6
Time	38.4±10.3	20.6±3.2	19.6±2.7	32.3±8.4	58.2±15.4	146.8±39.6	17.5±0.8

Analysis of the results indicates that:

- The partition heuristic generates very good guesses – minimal time or within the confidence interval of the minimal time in the four datasets.
- The impact of the partition P size on the time is significant. The minimal time is obtained when the partition size is somewhat close to double the number of RVs in the final solution. We can see from the results that whenever the partition size is close to be double the number of RVs, than the solution is within the optimum interval.
- According to the standard deviations of the results there isn't significance difference in neither the RMSE nor the number of RVs in most of the cases. The partition size has no affect on the accuracy and on the number of the RVs.

4.2.3 Reducing the Noise Variance Update Rate

In the regression problem we maximize the likelihood with respect to both $\mathbf{w}$ and σ^2. However, as described in chapter 2, the convergence of the algorithm is related only to the convergence of the weights and not to of noise variance. We already know that updating the noise variance has a runtime complexity of $O(N^2)$ since in every iteration of the algorithm we compute the error $\|\mathbf{t}-\mathbf{\Phi\mu}\|^2$, over all the examples in the training set.

$$(\sigma^2)^{new} = \frac{\|\mathbf{t}-\mathbf{\Phi\mu}\|^2}{N-\sum_{i=1}^{N}(1-\alpha_i\Sigma_{ii})} \tag{4.2.3.1}$$

In practice we saw that the noise variance is changing much slower than $\boldsymbol{\alpha}$ vector (as expected), thus, it is possible to reduce the update rate of the noise variance and to gain a further reduction in the run time of the algorithm. In the following experiments in this dissertation we reduced the noise variance updates to once every ten iterations. This caused a further reduction of 22% to 40% in the time required for training the models without damaging the accuracy or the sparseness of the solution.

4.3 Estimating the Acceleration Rate

The purpose of this section is to estimate the acceleration rate of the partition algorithms with the partition size heuristics. We already know that the run time complexity of the basic RVM is $O(N^3)$ and from the complexity analysis in sub-section 4.2.2.1 we conclude that the runtime complexity of the partitions algorithms is $O(N^2)$. Though the previous two sections demonstrated a significant acceleration of partitions algorithms over the basic RVM for several data sets; it seems that the acceleration could be even larger for big data sets.

In order to estimate the difference in the runtime between the basic RVM and the partition algorithm we conducted the following experiment:

1. We used the Sinc function in order to create different sizes of training set. In each training set we also generate 1000 examples of testing set.
2. The basic RVM and working set RVM was trained as follow: each algorithm was trained with the following sizes of the data sets (4 times each size): 200, 400, 800, 1200, 1600, 2000, 2400, 2800, 3200, 3000, same randomizations was used in both algorithms.
3. The Gaussian kernel was used and its width parameter was optimized to the value of 3 units.
4. Before every repetition we permute the data in order to negate a dependency of the results on the order of the training set.

5. The comparative measures were the root means square error, number of relevance vectors and the time required for training the algorithm.

Table 4.3.1 presents the results of the experiment based on the averages over four repetitions; the standard deviation is also given in the table.

Table 4.3.1 The results of the experiments over different size of the sinc data.

size	Time		RMSE		# RVs	
	Basic	Working Set	Basic	Working Set	Basic	Working Set
200	0.26±0.08	0.19±0.04	0.025±0.007	0.028±0.007	6.25±1	6.25±1.7
400	0.88±0.13	0.26±0.03	0.018±0.003	0.017±0.002	6.5±1	8±0.8
800	4.65±0.8	0.72±0.5	0.01±0.004	0.01±0.003	8.25±4	7.25±0.7
1200	13.3±1.1	2.6±1.2	0.011±0.002	0.01±0.002	8±3.2	8.25±0.5
1600	30.5±2	4.85±0.6	0.006±0.002	0.006±0.002	9.75±1	8±0.5
2000	58.8±4.8	6.13±1.9	0.008±0.009	0.008±0.0009	12.75±3	8.75±0.5
2400	100.8±7.1	6.4±0.4	0.006±0.001	0.007±0.001	13.75±4.8	8.5±0.6
2800	153.7±15.6	7.4±0.5	0.006±0.001	0.006±0.001	12.5±4.8	7.75±0.9
3200	232.6±24.7	8.3±1	0.007±0.001	0.007±0.002	9.25±2.3	8.5±0.6
3600	328.2±16	9.5±0.4	0.006±0.001	0.007±0.001	15.5±8.1	9±0.8

The following figures presented the results graphically.

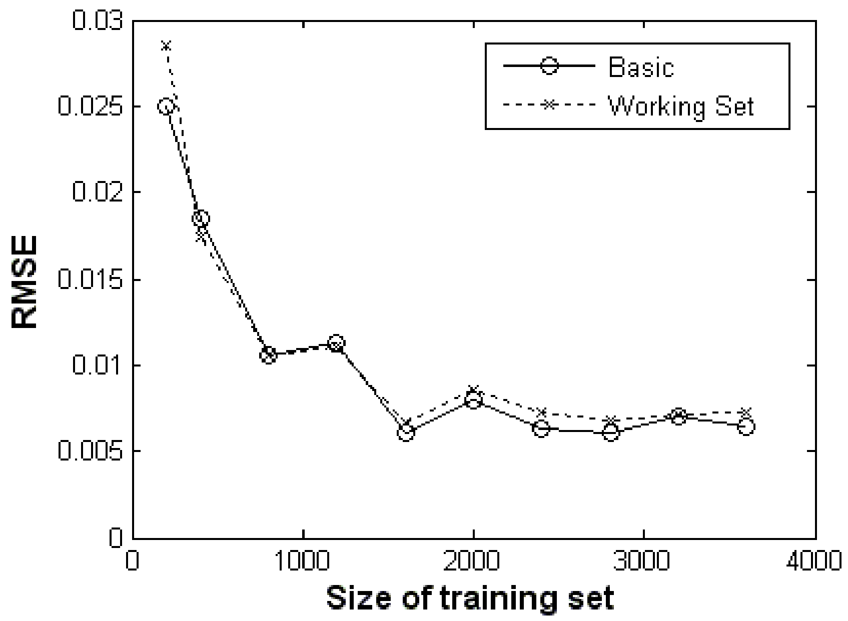

Figure 4.3.1 The RMSE as a function of the size of training set for the two algorithms. The RMSE measured on the test set.

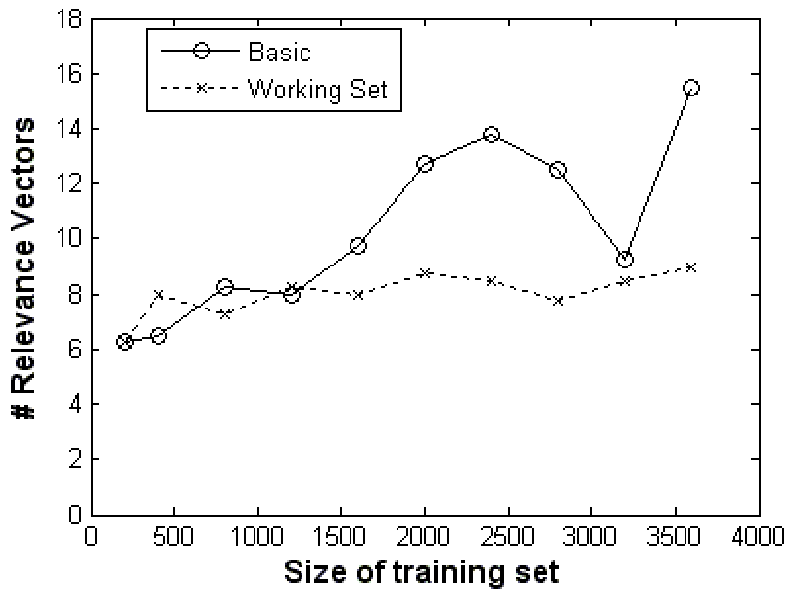

Figure 4.3.2 The indifference (according to the standard deviation in table 4.2.5) in the number of relevance vectors.

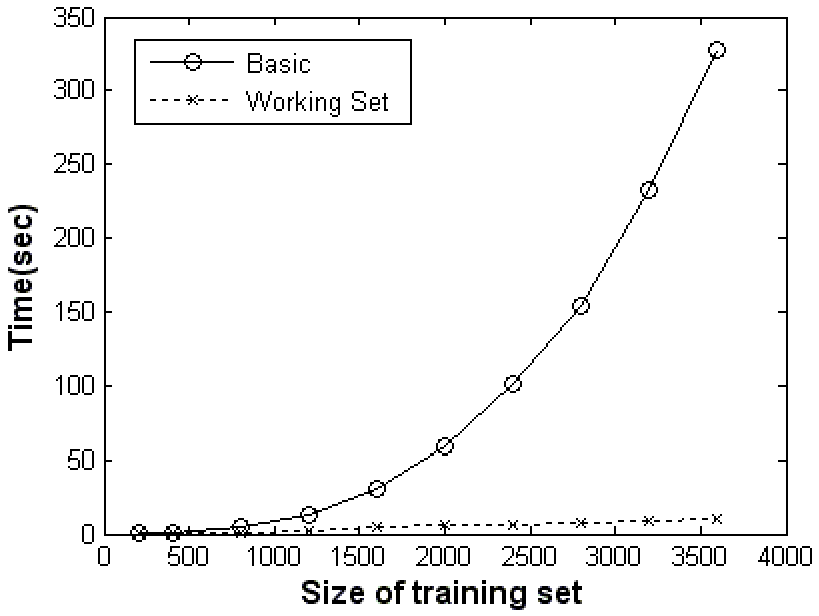

Figure 4.3.3 The difference in time required for the training the models as the size of the data set increase.

Both algorithms are very accurate according to figure 4.3.1. We can see also that when the size of the training set is increased the accuracy is also increased; one possible explanation for that could be the additional examples which make the models more accurate by introducing additional possible relevance vectors which add information to the model.

Regarding the number of relevance vectors, it seems that in large data sets the working set is deleting vectors that the basic algorithm found as relevant; however, the difference is not significant according to the standard deviation presented in table 4.3.1.

Figure 4.3.3 demonstrates that the speed up rate is very significant, especially when the size of the training set increased. The next sub-section will analyze this observation from a statistical point of view.

The experiments results demonstrate that the working set algorithm can accelerate the RVM in impressive percentages while simultaneously not damage neither the accuracy nor the sparseness of the solution.

4.3.1 Statistical Analysis of the Results

We wish to find out whether the power of the slope of the time as a function of the size of training set (figure 4.3.3) is statistically significant or not. The SPSS statistical package was used to compute a nonlinear regression of the form aN^b in order to find the most suited function for the data and to check its statistical significance. The non-linear regression generated to the graph in figure 5.3.3.1.

Figure 4.3.1.1 and table 4.3.1.1 presents the results of the non-linear regression for the basic RVM. Figure 4.3.1.2 and table 4.3.1.2 present the results of the non-linear regression for the Working Set which has been used as the partition algorithm.

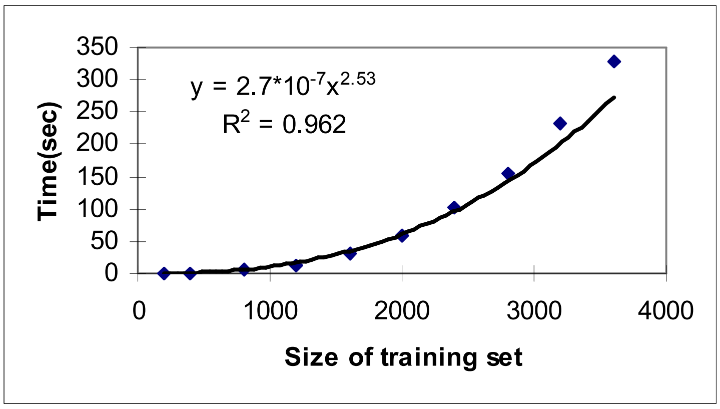

Figure 4.3.1.1 The most fitted line for the basic RVM, the R^2 emphasizes how well the data can be predicted using the function fitted line. In the equation y denote the time and x denote the size of the training set.

Table 4.3.1.1 The results of the statistical analysis for the basic RVM, we can see that the F test is significant as well as the power parameter under confidence level of 95%.

F test =99.92 Significance F=.0		
Variable	b (power)	a(coefficient)
value	2.53	$2.7*10^{-7}$
T test	33.85	1.83
Significance T	.0	.1

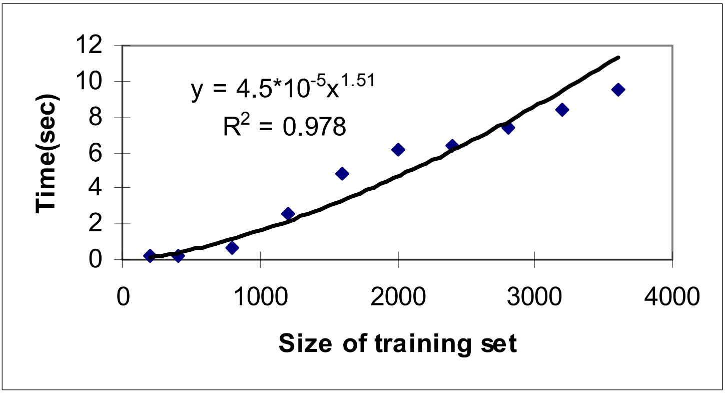

Figure 4.3.1.2 The most fitted line for the Working Set, the R^2 emphasizes how well the data can be predicted using the function fitted line. In the equation *y* denote the time and *x* denote the size of the training set.

Table 4.3.1.2 The results of the statistical analysis for the Working Set. The F test is significant as well as the power parameter under confidence level of 95%.

F test =177.05 Significance F=.0		
Variable	b (power)	a(coefficient)
value	1.51	$4.5*10^{-3}$
T test	33.85	1.2
Significance T	.0	0.263

The complexity analysis in the previous section indicate that the working set algorithm could reduce the complexity of the RVM from $O(N^3)$ to $O(N^2)$. The nonlinear regression indicates that the runtime complexity of the basic RVM is $O(N^{2.5})$ and runtime complexity of the working set is $O(N^{1.5})$, thus, reduction is well achieved. Further details on the statistical analysis available in appendix B.

4.4 Comparing the Three Acceleration Heuristics

The question investigated in this section is whether one of the three acceleration heuristics is superior to the others. From a complexity point of view, as discussed in section 4.1, we already know that the runtime complexity of the split and merge algorithm is less than the runtime complexity of the incremental approach and the working set. However, each algorithm merges differently the RVs resulting from the already processed partitions with the rest of the data and that may affect the effective complexity of the algorithms.

4.4.1 Experiments for Defining the Best Partition Algorithm

In this experiment we compared the results of the partition algorithms among themselves. The Training set was used for the learning phase, and the error was measured on the testing set. The heuristic of section 4.2 was used to determine *P*. Each algorithm was simulated for 20 different repetitions, the same randomizations and initial partitions were used for testing each algorithm. The Gaussian kernel was utilized and its single width parameter was optimized via cross validation. The same width was used for all the related experiments.

4.4.1.1 Data Sets and Evaluation Metrics

Instead of taking a set of unrelated benchmark data sets, we decided to use the *Pumadyn* family of datasets (table 4.2.2.1.1) in order to see if there is any difference among the algorithms that depends on the type of the data sets (non linear, high noise, number of dimensions) and also to check the limitations of the RVM in general. The same measures have been used in this experiment, the time required for training the algorithm, the number of RVs and the accuracy (RMSE).

4.4.1.2 Results

The following tables present the comparative RMSE, the comparative number of RVs, and the comparative run time respectively. The standard deviation of each measure is also presented in the tables.

Table 4.4.1.2.1 Comparative RMSE

Name	*SplitMerge*	*Incremental*	*WorkingSet*
pumadyn-8fh	3.16±0.04	3.16±0.04	3.16±0.04
pumadyn-8fm	1.05±0.012	1.05±0.011	1.05±0.012
pumadyn-8nh	3.27±0.06	3.28±0.06	3.28±0.06
pumadyn-8nm	1.06±0.02	1.08±0.02	1.06±0.02
pumadyn-32fh	$0.02\pm2*10^{-4}$	$0.02\pm2*10^{-4}$	$0.02\pm2*10^{-4}$
pumadyn-32fm	$0.005\pm8*10^{-5}$	$0.005\pm9*10^{-5}$	$0.005\pm1*10^{-4}$
pumadyn-32nh	$0.033\pm4*10^{-4}$	$0.033\pm4*10^{-4}$	$0.033\pm4*10^{-4}$
pumadyn-32nm	$0.027\pm3*10^{-4}$	$0.027\pm3*10^{-4}$	$0.027\pm3*10^{-4}$

Table 4.4.1.2.2 Comparative number of RVs

Name	*SplitMerge*	*Incremental*	*WorkingSet*
pumadyn-8fh	37.7 ±1.8	37.7±2.8	37.4±2.8
pumadyn-8fm	69.1±3.03	69.7±2.4	70.2±4.2
pumadyn-8nh	134.7 ±4.6	137.9±4.6	145.1±5
pumadyn-8nm	86.8±3.3	83±2.6	81.8±3.1
pumadyn-32fh	66.9±5.3	78.5±5.6	79.8±4.8
pumadyn-32fm	117.6±17	169.3±22.2	203±22.7
pumadyn-32nh	11.3±4.1	9.4±2.9	9.1±1.4
pumadyn-32nm	42.8±2.8	62.2±9	64.4±8

Table 4.4.1.2.3 Comparative time to convergence

Name	*SplitMerge*	*Incremental*	*WorkingSet*
pumadyn-8fh	33.2±2.6	20.1±4.6	24.3±4.6
pumadyn-8fm	82.1±9.5	33.8±1.7	34.1±1.1
pumadyn-8nh	278.4±31.2	212.7±44.5	239.8±15.5
pumadyn-8nm	153.3±12.5	47.1±5.2	46.2±4.3
pumadyn-32fh	18.4±0.6	17.4±0.7	18.6±1
pumadyn-32fm	21.5±0.6	29.1±0.4	29.4±1.1
pumadyn-32nh	11.2±0.4	16.4±0.4	16.9±0.6
pumadyn-32nm	13.3±0.4	15.8±0.5	16.8±0.6

Analysis of the results indicates that:

- The three heuristics obtained a similar accuracy and a similar sparseness.
- None of the heuristics demonstrated superiority over all datasets and for the three quality measures.
- The incremental algorithm and the working set algorithm demonstrate similar behavior. The split and merge algorithm demonstrates slightly different behavior than the other two.
- Although further experiments are necessary, it seems that the RVM is capable of solving problems with high degree of non-linearity, noise, and high dimensionality.

4.5 Discussion

The basic RVM algorithm has some very attractive properties; however, in its original form it is too complex for any realistic medium or large datasets.

In this chapter we suggested three algorithms for accelerating the basic RVM via splitting the dataset for consecutive applications of the basic RVM and merging the solutions. Complexity analysis, statistical analysis and simulation experiments on benchmark datasets indicate that the three algorithms indeed accelerate the RVM, while retaining the accuracy and the sparseness of the solution. We also proposed a heuristic for selecting good partition size. Complexity analysis indicates that the partition algorithms via the usage of the partition size heuristic reduces the runtime complexity to $O(N^2)$.

Although none of the heuristics demonstrated its superiority over the others, we recommend using the working set algorithm, because it is expected to provide a good solution even if it has to be prematurely aborted due to time constraints.

This chapter provides another essential step for popularizing the RVM algorithm, so that eventually it will turn into another “standard tool” for the practitioner of machine learning**.**

Another open research question is the acceleration of the classification RVM. The classification RVM [21], includes in addition to the matrix inversion of (2.5.5), a computationally intensive non-linear optimization section. It is clear that the classification RVM will also benefit from partitioning the dataset. However, a different partition heuristics may be needed. The classification RVM is studied in the next chapter.

5. ACCELERATING THE RELEVANCE VECTOR CLASSIFICATION

The RVC contains in addition to the inversion of the covariance matrix $\mathbf{\Sigma}$ (formula 2.5.5) an intensive non-linear optimization step. Thus, the efficiency of the partition algorithm described in the previous chapter has to be reexamined. The purpose of this chapter is to study the runtime complexity reduction of the Relevance Vector Classification (RVC).

The first section will detail the formulation of the RVC followed with an example of the basic implementation of the RVC. The second section will describe the acceleration of the RVC via data partitioning using the working set algorithm, followed by comprehensive experiments on benchmark datasets. The experiments as well as the statistical analysis will demonstrate that the partition algorithm accelerates the RVC as well.

5.1 Formulation of the Relevance Vector Classification

The relevance vector approach is now extended to the case of classification. The goal here is to predict the posterior probability of the class membership given the input x. The type II maximum likelihood approach is also being in used in the classification case.

5.1.1 Binary Classification

Tipping [21] used the sigmoid function: $\sigma(y) = 1/(1+e^{-y})$ to generalize the linear model. The main idea of the sigmoid function is to approximate the two-class classification problem with a regression problem. Using this link function we can adopt the Bernoulli distribution $P(t \mid x)$ and rewrite the likelihood as:

$$P(\mathbf{t} \mid \mathbf{w}) = \prod_{n=1}^{N} \sigma\{y(\mathbf{x}_n;\mathbf{w})\}^{t_n} [1-\sigma\{y(\mathbf{x}_n;\mathbf{w})\}]^{1-t_n} \qquad (5.1.1.1)$$

where $t_n \in \{0,1\}$. Note that there is no noise variance in (5.1.1.1).

The same model form which has been in used for the Relevance Vector Regression is also being used in the RVC:

$$y(x,w) = \sum_{i=1}^{N} w_i * k(x,x_i) \qquad (5.1.1.2)$$

With an additional rounding to the closest value $\{0,1\}$, model (5.1.1.2) can produce a binary classification function. Thus, in the classic RVC formulation we need to find two solutions for two different problems, a discrete problem and a continuous one.

5.1.1.1 The Discrete Problem

In the discrete problem we assume that α is known. From (5.1.1.1) and via the approximation $p(\mathbf{w} \mid \mathbf{t}, \alpha) \propto P(\mathbf{t} \mid \mathbf{w}) p(\mathbf{w} \mid \alpha)$ it is possible to minimize the following cost function for the unknown ***w*** (5.1.1.1.1). It is equivalents to maximizing the posteriors of ***w***.

$$J(w \mid t, \alpha) = \tfrac{1}{2} wAw - \sum_{i=1}^{N} \left[t_i \log\left(\sigma\left(\phi_i^T w\right)\right) + \left(1 - t_i\right) \log\left(1 - \sigma\left(\phi_i^T w\right)\right) \right] \qquad (5.1.1.1.1)$$

where $A \equiv diag[\alpha_1, \ldots, \alpha_N]$ and $\sigma(z) = \frac{1}{1 + \exp(-z)}$ is the logistic sigmoid function. At the minimum value of *J*, its gradient with respect to w is zero:

$$\nabla J = Aw + \Phi^T (\sigma - t) \qquad (5.1.1.1.2)$$

where $\sigma = \left[\sigma\left(\phi_1^T w\right), \ldots, \sigma\left(\phi_N^T w\right)\right]^T$. The second derivative of J with respect to w (the Hessian) $\nabla^2 J = (\mathbf{\Phi}^{\mathbf{T}} \mathbf{B} \mathbf{\Phi} + \mathbf{A})$ is positive definite with $\mathbf{B} \equiv diag[\beta_1, \ldots, \beta_N]$, and β_i defined as follows:

$$\beta_i = \sigma\left(\phi_i^T w\right)\left(1 - \sigma\left(\phi_i^T w\right)\right) \qquad (5.1.1.1.3)$$

Assuming that we succeeded to solve the first problem (5.1.1.1.1) – which is a nonlinear optimization problem - we can effectively locally linearize the classification problem around w - the most probable weights. Around the most probable weights, the probability of the weights can be further approximated with a Gaussian distribution (via the Laplace approximation). Thus, we may refine it to be a sparse solution. We are going to solve the following regression RVM for the continuous approximation of the targets, $\tilde{t}$:

$$\tilde{t} = \Phi^T w + B^{-1}\left(t - \sigma\left(\phi_i^T w\right)\right) \qquad (5.1.1.1.4)$$

5.1.1.2 The Continuous Problem

In the continuous problem we solve for the unknowns (μ, Σ)as with the RVR:

$$\begin{aligned}\mathbf{\Sigma} &= (\mathbf{\Phi}^{\mathrm{T}}\mathbf{B}\mathbf{\Phi} + \mathbf{A})^{-1} \\ \boldsymbol{\mu} &= \mathbf{\Sigma}\mathbf{\Phi}^{\mathrm{T}}\mathbf{B}\tilde{\mathbf{t}}\end{aligned} \qquad (5.1.1.2.1)$$

The solution is derived via the following iterative type II maximization of the marginal likelihood with respect to α :

$$\alpha_i^{new} = \frac{1-\alpha_i \Sigma_{ii}}{\mu_i^2} \qquad (5.1.1.2.2)$$

Using the covariance matrix $\mathbf{\Sigma}$ and the most probable weight $\boldsymbol{\mu}$ of the Gaussian approximation the hyper parameters α are updated using (5.1.1.2.2). The RVM algorithm cycles between (5.1.1.2.1) and (5.1.1.2.2), reducing the dimensionality of the problem when any α_i larger than the threshold. The algorithm stops when the likelihood (5.1.1.1) ceases to increase, or when any α_i remains practically unchanged between two consecutive iterations.

However, if $\mu \neq w$ approximately, the solution is not optimal. In this case, we insert the values of α_i in equation (5.1.1.1.1), resolve again for w and start all over again.

Note that without a reasonable *w*, the Gaussian approximation assumption for the regression RVM may not hold and convergence will be minor if any.

5.1.2 How the Relevance Vector Classification is working

Lets inspect how the RVM working on a classification case. Assume that we have pairs of input observation x_n along with corresponding targets $t_n \in \{0,1\}$. We can approximately represent the probability that the data set is correctly explained using classifier approximate model $\sigma\{y(\mathbf{x}_n;\mathbf{w})\}$ (σ is the sigmoid function) as:

$$P(\mathbf{t}\,|\,\mathbf{w}) = \prod_{n=1}^{N} \sigma\{y(\mathbf{x}_n;\mathbf{w})\}^{t_n}[1-\sigma\{y(\mathbf{x}_n;\mathbf{w})\}]^{1-t_n} \qquad (5.1.2.1)$$

where $\mathbf{w}$ is the weights vector of our classification GLM. The assumption that the training data is correctly labeled gives us the ability of using Bayes rule and to turn (5.1.2.1) around and infer likely values of the weights given some labeled data:

$$P(w\,|\,x,t) \propto P(w)P(t\,|\,x,w)$$

Introduce $P(w)$ as our prior belief about the values of the weights. Now we introduce another vector of parameters, the $\boldsymbol{\alpha}$ vector. Each element of this vector controls the width prior over its corresponding weight:

$$P(w) = \sum_{m=1}^{M} G(w_m \mid 0, \alpha_m^{-1}) \tag{5.1.2.2}$$

As a hyper prior over values of α we choose a very broad Gamma distribution - a standard choice for non-informative priors over Gaussian width parameters.

Considering the weight and the probability of any single basis function, there are two possibilities:

1. The corresponding weight of this basis function is set to some non zero value, the accuracy of the classifier is increased using this basis function; this increases the value of (5.1.2.1), and therefore the probability of that model given the data.
2. The basis function does not contribute any additional information to the model and there is no value for the weight which will lead to an increase in the likelihood (5.1.2.1). At this point, the prior term in the model take his part, by setting the α_m parameter to a large value, the prior distribution $P(w_m)$ becomes peaked around zero. By then setting α_m to zero the posterior probability of the model is maximized.

5.1.3 Illustration of the Relevance Vector Classification

Following is an example of the relevance vector classification adapted from Tipping's web site on the RVM [27].

There are 100 training examples taken from Ripley (1996) synthetic function. The data is derived from a two-dimensional noisy function, the number of classes is 2 and the (optimal) Bayes class overlap error is 8%. A Gaussian kernel was used in the RVM and its width parameter was set to 0.5.

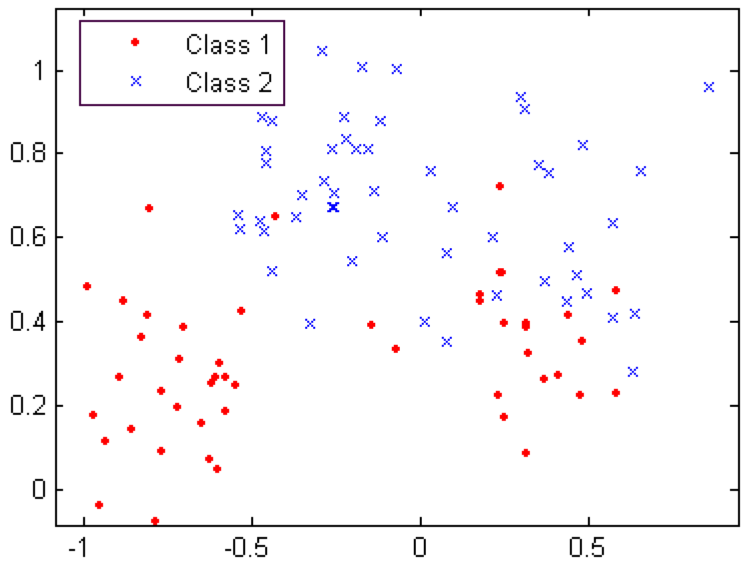

Figure 5.1.3.1 100 examples from Ripley's synthetic data

After training the RVC on this example only 4 relevance vectors remains in the result.

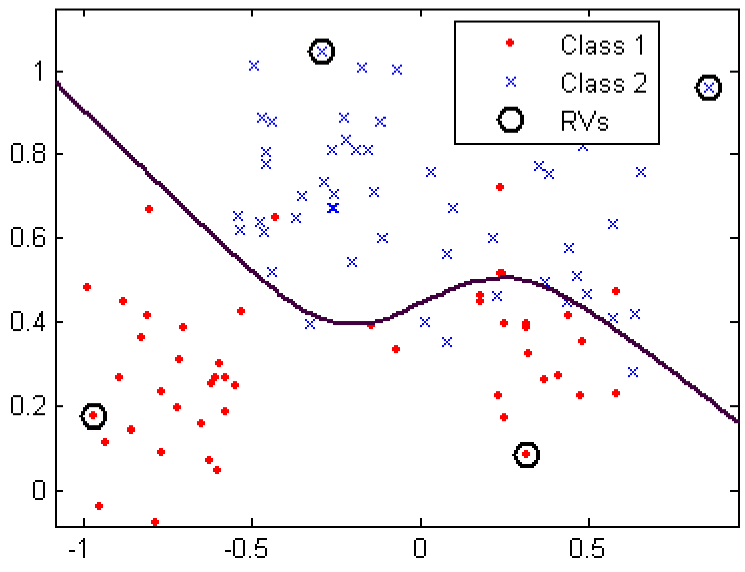

Figure 5.1.3.2 The 4 relevance vectors are circled, we can see they are a prototypes of the whole data, the decision boundary is shown in the solid line.

5.1.4 Multi Classification

Though RVC is basically designed for two-class problems, there is an extension [21] of the RVC to multi-class data sets. When the number of classes K is greater than 2 the likelihood (5.1.1.1) is generalized to the standard multinomial form:

$$p(\mathbf{t}\,|\,\mathbf{w}) = \prod_{n=1}^{N}\prod_{k=1}^{K}\sigma\{y_k(x_n; w_k)\}^{t_{nk}} \tag{5.1.4.1}$$

where σ denote the sigmoid function, K the number of classes and N the number of samples. In this case the classifier has multiple outputs $y_k(\mathbf{x};\mathbf{w}_k)$ each with its own parameter w_k and corresponding hyper parameter α_k. This multi classification technique is considered to be very highly disadvantages from a computational point of view since the size of the covariance matrix $\mathbf{\Sigma}$ scales with K.

Another approach to the *k*-class problem is to consider the problem as a collection of binary classification problems. Following the strategy used in the SVM literature - There are two possible implementations of this approach:

- The *one-against-the-rest* technique requires *k* binary classifiers to be constructed (when the label 1 is assigned to each class in its turn and the label 0 is assigned to the other *k-1* classes). In the prediction stage, a voting scheme is applied to classify a new point when a conflict occurs among the classifiers.
- The *one-against-one* technique trains a binary RVC for any two classes of data and obtains a decision function. Thus, for a k-class problem, there are $k(k-1)/2$ decision functions where a voting scheme is designated to choose the class with the maximum number of votes.

Both techniques considered are computational intensive since many classifiers have to be trained. Another disadvantage of those techniques is the optimization of the kernel parameters for each classifier. Since each classifier is trained as a binary problem it is obvious that there may not be a single best parameter which optimizes the kernel for all the classifiers. Thus, specific kernel parameter optimization for each classifier might be required.

5.1.5 Computational Complexity

It is difficult to accurately quantify the computational complexity of the RVM, since the size of the matrices in (5.1.1.2.1) may reduce from cycle to cycle. The matrix inversion operation in (5.1.1.2.1), which requires $O(N^3)$ operations, and the nonlinear optimization of (5.1.1.1.1) are the computationally intensive parts of the algorithm. The matrices $\mathbf{\Phi}$ and $\mathbf{\Sigma}$ are full rank[7], thus require initially $O(N^2)$ space complexity. Furthermore, large matrices can appear numerically non-positively definite even when they analytically are, and make the matrix become ill-conditioned after several cycles

[7] Some kernel functions can produce a sparse representation

even (look in appendix D). Furthermore, the nonlinear optimization problem may have many local minima. These problems limit the practicality of the basic RVC algorithm for moderately sized problems.

One possible strategy for complexity reduction is Iterative sub-sampling of the data, thus reducing N. The next section will describe our suggested sub sampling algorithm.

Another possible computational inefficiency in RVC results from excessive waiting for convergence. Typically, the initial convergence could be very fast, and than slows down considerably. Thus, the premature convergence flag which described in section 4.1 might be helpful also in the classification problem.

5.2 Accelerating the RVC via Data Partitioning

The first step before implementing a partition algorithm on the classification problem was to implement the basic RVC. As described in section 5.1 we are also need to maximize the likelihood (5.1.1.1) with respect to **w**. However the likelihood function (after using the sigmoid link function) is not continuous, thus we cannot integrate out the weights as in the regression case. An approximate procedure based on Laplace's approximation solves the problem: the $\mathbf{w}_{\mathrm{mp}}$ (most probable weights) locates the mode of the posterior distribution.

5.2.1 The Partition Algorithm

Since all the three partition algorithms discussed in chapter 4 demonstrated somehow the similar results, we decided to use the working set algorithm. Its seems to be little bit superior because of the fact it used the most informative example in every part of the training phase and the possibility of deletion 'good' basis functions (RVs) is very small. Further details about possible reintroduction of deleted RVs available in appendix E.

5.2.1.1 Key Idea

The key idea of the working set algorithm is to construct the best model gradually based on the most informative examples found so far. Consider that somehow we can guess an initial solution. It is not too expensive to use this initial solution for predicting the class membership for each one of the training examples. The examples with the wrong prediction class are potentially the most informative for improving the solution, thus they are introduced as part of the working set. A working set of size $P << N$ is used to compute an initial solution and this iterative process is repeated until a predetermined convergence criterion is being met, the data is exhausted, or the computing time is exhausted. The basic RVC algorithm is used for computing the initial solution and the RVs in each working set.

There is no need to wait for the possibly lengthy convergence of the RVC algorithm for the working set, since a new distribution assumption is produced even from the first iteration of the RVM. To avoid excessive increase in the size of the working set, it was decided to prematurely stop the convergence of the RVC algorithm when the size of the (candidate) RVs dropped to half the current size of the working set. To further simplify the management of the size of the working set, it was decided that its contents will be updated only in whole units of its initial size *P* (except possibly the last step where the reminder of $\frac{N}{P}$ may be smaller than *P*). A further computational gain is obtained from passing various parameters estimate (such as the noise level) from working set to working set – effectively enhancing its convergence rate.

For a graphical illustration of the working set algorithm, consider Figure 5.2.1.1.1, which presents a case of *N*=4000 and *P*=1000.

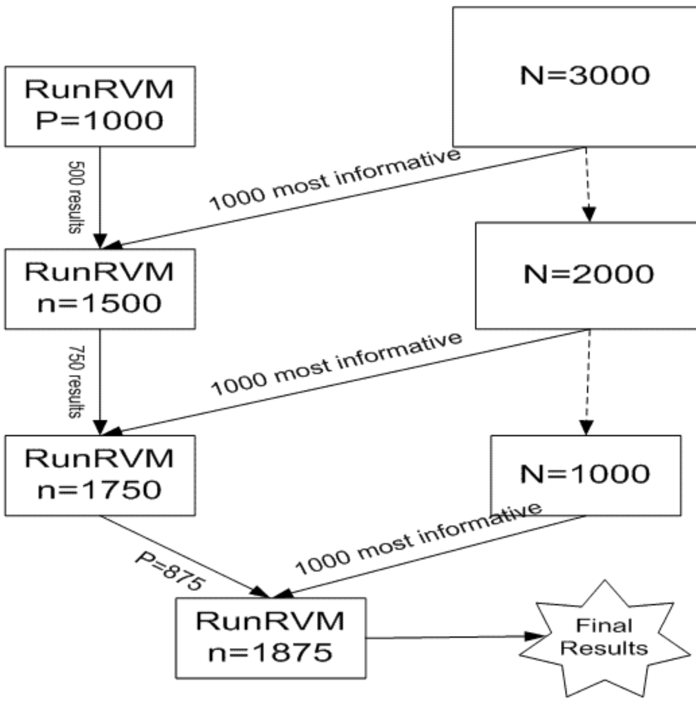

Figure 5.2.1.1.1 Working Set algorithm.

5.2.1.2 The Size of the Working Set

The key parameter in the efficiency of the working set algorithm is the size *P*. The runtime complexity of processing each working set is $O(P^3)$. *P* should be sufficiently large to contain all the RVs of the final solution. Otherwise, running the RVM on the working set may fail to converge to a decent solution and the size of the next working set will increase in the next step to 2P. In practical machine learning – which is concerned with finding the best model – it is not possible to optimize for P. The sparseness of the RVM solution may depend on poorly understood attributes of the data, such as its non-linearity, noise, dimensionality, and the chosen kernel functions. Nevertheless, it is possible to compute a clue regarding the complexity of the data. The idea is to train the basic RVC on a randomly sampled subset of size $P^0 = \sqrt{N \cdot d}$, where *N* denotes the size of the dataset and *d* is the dimensionality of the data. Count the number of relevance vectors (#RVs) in the subset solution trained from the basic RVC and compute the sparseness ratio $\gamma = \frac{\#RVs}{\sqrt{N \cdot d}}$. Based on tuning experiments a good sub-optimal P is obtained as follows:

$$P = \begin{cases} 0.2P^0 & \text{if} \quad \gamma \le 0.25 \\ 0.6P^0 & \text{if} \quad 0.25 < \gamma \le 0.75 \\ P^0 & \text{if} \quad \gamma > 0.75 \end{cases} \tag{5.2.1.2.1}$$

The two top branches indicate a potentially sparse solution that merits a decrease in the size of the initial working set P. The bottom branch indicates a potentially complex dataset which may result in eventual increase in the size of the working set. Our experience indicates that an excessive number of RVs is often associated with an improper kernel function. Thus, when the sparseness ratio $\gamma \cong 1$ it is advisable to further optimize the kernel function parameters.

5.2.1.3 The Working Set Algorithm

The outline of the working set algorithm is as follows:

1. Run the basic RVC on a randomly selected working set of size P^0 and estimate *P* from (5.2.1.2.1).
2. Run the basic RVC on the first working set of size *P* and store the resulting RVs.

3. Construct a model from the RVs, predict the class membership for each one of the remaining (unused) data, and construct a new partition of size *P* from the data associated with misclassification prediction.
4. Update the working set by merging the current RVs with the newly generated partition.
5. Run the basic RVC on the working set and store the resulting RVs.
6. Repeat steps 3, 4, and 5 until exhausting the data.
7. For the last working set, let the basic RVC run until convergence and obtain the final solution.

Further attention is needed in step 3 of the algorithm; we take care that the distribution of classes in the working set will approximate the distribution of classes in the full training set. Thus, at first we merge into the working set cases of wrong classification selected from each class according to the approximate proportion of classes in the training set. When one of the classes is exhausted of misclassified examples then correctly classified examples are selected for the working set.

5.2.1.4 Complexity Analysis of the Working Set

Considering that sparse solution exists, the premature stopping of the basic RVC on each working set halves the number of RVs in each step. Due to the merging process of the current RV candidates with new chunks of data of size *P*, there is a gradual increase in the size of the working set to a limit of *2P.* $P, P+\frac{P}{2}, P+\frac{3P}{4}, P+\frac{7P}{8}, \cdots, \approx 2P$. Thus in the worst scenario (last iteration) the size of the data being processed in the working set algorithm can be 2P. Consider Figure 5.3.1.1. There are approximately $\frac{N}{P}$ working sets to process. Consider (5.2.1.2.1) each one of them is of size order $P \approx \sqrt{N \cdot d}$ since $P^0 = \sqrt{N \cdot d}$, and its runtime complexity is of the order of 3 due to the complexity of the basic RVC. Thus, the overall runtime complexity of the working set algorithm is $O(N^2)$.

Where $P \approx \sqrt{N \cdot d}$ a sketch of the complexity analysis is as follow:

$$O\left(\frac{N}{P} \cdot (2P)^3\right) = O\left(\frac{N}{\sqrt{N \cdot d}} \cdot 8(\sqrt{N \cdot d})^3\right) = O\left(N \cdot 8 \cdot (\sqrt{N \cdot d})^2\right) = O(8dN^2) = O(N^2)$$

5.3 Comparative Experiments

The purpose of the following experiments is to validate the acceleration of the working set algorithm and to check the working set on benchmark classification problems.

5.3.1 Measuring the Acceleration Rate

We wish to estimate the acceleration rate in the classification problem. We already know from the previous sub-section that the run time complexity of the basic RVC is $O(N^3)$ and the run time complexity of the working set is $O(N^2)$ but in order to validate the reduction we conducted the following experiment:

1. Ripley's (1996) synthetic data was used in this experiment; the data set contains 1250 examples with two dimensions and two classes. Both classes were generated from mixtures of two Gaussians. We generate additional 1250 examples by taking the values of the original 1250 examples and contaminate them with additive Gaussian noise with standard deviation of 0.05 to each one of the input variables (two dimensions). The targets vector was cloned respectively so we actually have data set with 2500 example.

2. Before every repetition we permute the data in order to negate a dependency of the results on the order of the training set.

3. The basic RVC and working set RVC were trained as follow: each algorithm was trained with the following sizes of the training sets (4 times each size): 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800 and 2000. Same randomizations were used in both algorithms. In every repetition we used 500 examples as test set.

4. A Gaussian kernel was used and its single width parameter was optimized to the value 0.5.

5. The comparative measures were the classification test error (%), number of relevance vectors and the time required for training the algorithm.

Table 5.3.1.1 present the results of the experiment based on the averages over four repetitions; the standard deviation is also presented in the table.

Table 5.3.1.1 The results of the experiments over different size of the same Ripley's data set.

Size	Classification test Error (%)		# RVs		Time	
	Basic	Working Set	Basic	Working Set	Basic	Working Set
200	8.5±1.35	8.3±1.36	4.5±0.58	4.5±0.58	1.43±0.18	1.6±0.25
400	9.25±1.4	9.05±1.25	5.25±0.5	5±0.8	9.4±0.83	5.5±0.73
600	9.25±0.37	9.1±0.5	6±1.16	6.5±0.58	25.8±0.9	11.4±3.4
800	9.2±0.74	9.25±0.85	5.25±0.5	5.5±0.58	60.5±0.9	18.8±3.2
1000	9.4±1.34	9.15±1.2	5.75±0.96	5.75±0.5	108.6±9.8	24.38±2
1200	9.45±1.7	9.3±1.7	6±0.5	5.25±0.5	190.5±6.87	33.46±6.8
1400	9.1±1.8	8.9±1.55	6.25±1.7	7±0.8	278.5±8.2	44.6±2.3
1600	9.25±0.47	9±0.7	6±0.8	6.25±1.26	415.7±65.3	59.2±5.2
1800	8.55±1.8	8.7±1.8	6.5±1	6.75±0.96	564.1±30.8	69.3±3.4
2000	9.6±0.9	9.75±0.53	5.75±0.96	6.75±0.5	808±58.4	88.5±2.8

Figures 5.3.1.1, 5.3.1.2 and 5.3.1.3 illustrate the results.

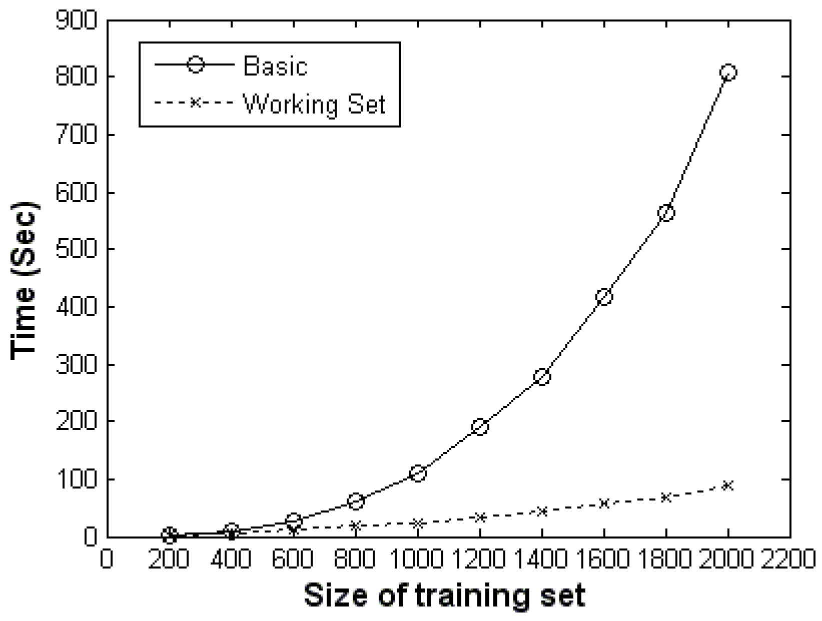

Figure 5.3.1.1 Comparing the times required for training the models as the size of the data set increase.

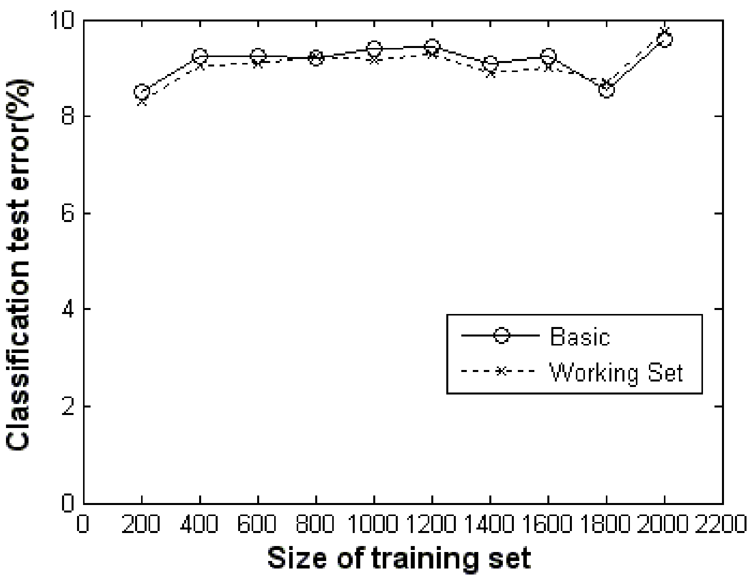

Figure 5.3.1.2 Comparing classification test error over the test set between the two algorithms and over different sizes of training sets

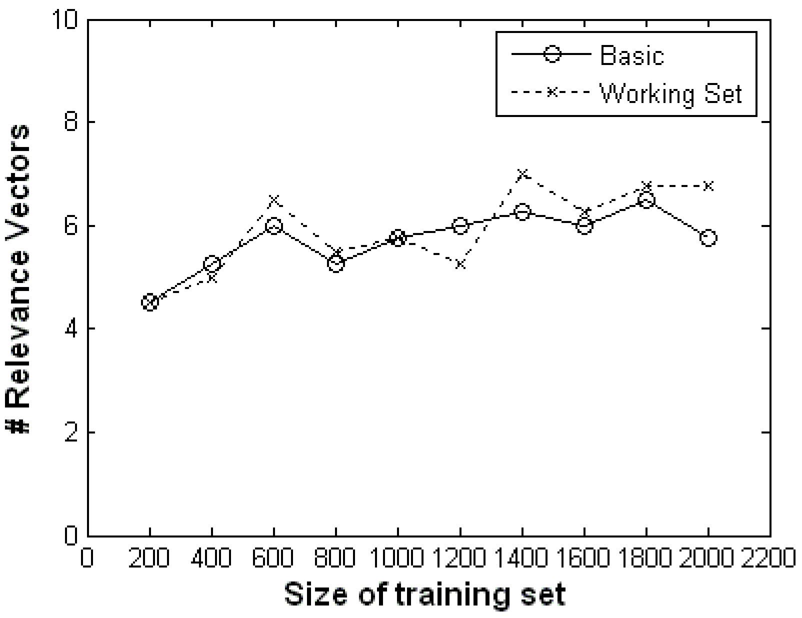

Figure 5.3.1.3 Comparing the number of relevance vectors.

Considering that for Ripley's synthetic data the optimum class overlap (Bayes error) is 8% both algorithms are remarkably accurate according to figure 5.3.1.2 which indicates classification test error around 9% in all simulations. The additional 1250 examples which we generate in order to enlarge the size of the data set may influence the number of the RVs in the final solution which was supposed to be 4 RVs [21]. A possible explanation for that could be the additional 1250 examples we generated, which might caused a shift and distortion in the shape of the decision boundary due to the additive noise. Thus the growth in the number of relevance vectors from 4 RVs to 6 RVs (on average) seems reasonable. Nevertheless, it is clear that sparsity is achieved in a remarkable way.

Figure 5.3.1.1 demonstrates the significant speed up rate. The next sub-section will analyze the speed up from a statistical point of view. The experiment results indicate that the working set is indeed accelerating the RVC while simultaneously not damaging neither accurate nor the sparseness of the solution.

5.3.1.1 Statistical Analysis of the Complexity Model

We wish to find out whether the power of the slope of the time as a function of the size of training set (figure 5.3.1.1) is statistically significant or not. The SPSS statistical software was used to compute a nonlinear regression of the form aN^b in order to find the best function for the data and to check its statistical significance. Figure 5.3.1.1.1 and table 5.3.1.1.1. Present the results of the non-linear regression for the basic RVC. Figure 5.3.1.1.2 and table 5.3.1.1.2. Present the results of the non-linear regression for the Working Set.

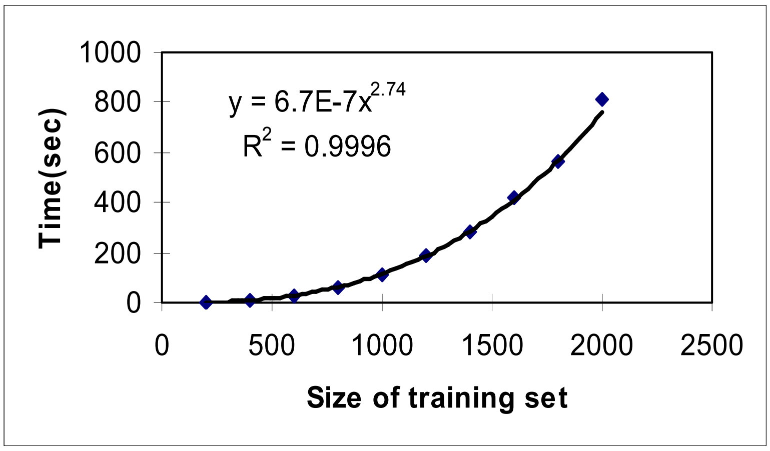

Figure 5.3.1.1.1 The most fitted line for the basic RVC, the R^2 emphasizes how well the data can be predicted using the function fitted line. In the equation y denote the time and x denote the size of the training set.

Table 4.3.1.3 The results of the statistical analysis for the basic RVC, we can see that the F test is significant as well as the model parameters under a 95% confidence level.

F test =18592.48	Significance F=.0	
Variable	b (power)	a(coefficient)
value	2.74	$6.74*10^{-7}$
T test	136.35	7.26
Significance T	.0	.0001

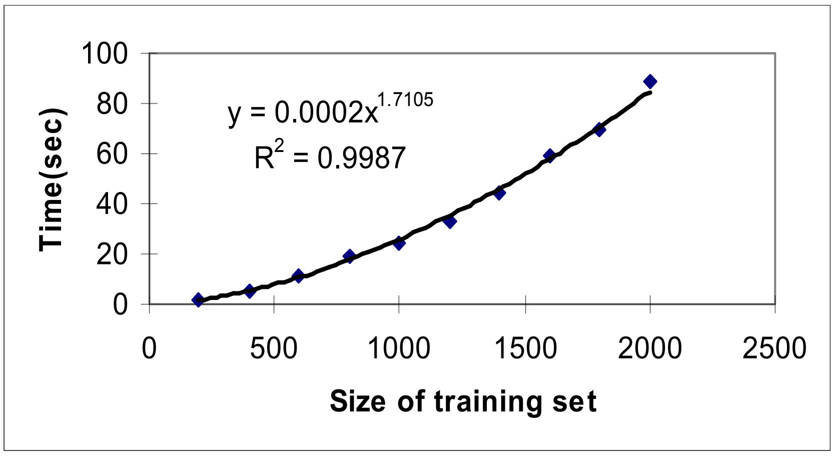

Figure 5.3.1.1.2 The most fitted line for the Working Set, the R^2 emphasizes how well the data can be predicted using the function fitted line. In the equation y denote the time and x denote the size of the training set.

Table 4.3.1.4 The results of the statistical analysis for the Working Set, we can see that the F test is significant as well as the model parameters under 95% confidence level.

F test =6119.3	Significance F=.0	
Variable	b (power)	a(coefficient)
value	1.71	0.0002
T test	78.22	6.6
Significance T	.0	.002

In a previous section it was argued that the working set is reducing the complexity of the algorithm from $O(N^3)$ to $O(N^2)$. The equations in the plots indicate that the run time complexity of the basic RVC is $O(N^{2.74})$ and the run time complexity of the working set is $O(N^{1.7})$, thus, complexity reduction is achieved. Look in appendix C for further details on this statistical analysis.

5.3.2 Experiments with Benchmark Classification Problems

We simulated the working set algorithm from section 5.2.1 together with its partition size heuristics and the premature convergence flag against the basic RVC. The goal of the experiments is to study the acceleration of the working set algorithm compared to the basic RVC on a wide sample of benchmark classification problems.

5.3.2.1 Data Sets and Evaluation Metrics

We used some popular benchmark datasets for classification problem, the list and the details of those data sets are described in table 5.3.2.1.1. We choose those data sets because they are familiar in the machine learning community and seem representative of wide sample of classification problems. All problems are two classes and the split of the datasets between training set and test set is also presented.

Table 5.3.2.1.1 Details of the benchmark datasets used for the classification experiments.

Data Set Name	Size	# Attributes	Training set /Test set
Breast cancer	277	9	250/27
Pima Indian diabetes	768	8	692/76
German	1000	20	900/100
Titanic	2201	3	1981/220
Banana	5300	2	3500/1800

For every experiment and comparison we used the following 3 measures:

- Time - total run time to convergence in seconds
- Classification test error- the percentage of the wrong predictions, computed as the number of wrong prediction divided by the number of the examples in the test set.
- #RVs - number of relevance vectors of the final trained model

5.3.2.2 Experiment Description

- Gaussian kernel was utilize and its single width parameter was optimized via cross validation using only a subset of the training set since the sizes of the training sets we used were quite large. The same width was used for both algorithms.
- Each algorithm was simulated for 10 times on different randomizations of the training set; since the algorithms may be influenced by the order of the data we permute the data in each repetition and used the same order for both algorithms.
- A training set was used for the learning phase, and the classification test error was measured on the testing set.

- All simulations and comparisons have been done on a computer equipped with a Pentium IV 3.1 GHz processor and 2 GB physical memory under the same conditions.

5.3.2.3 Results

Table 5.3.2.3.1 presents the comparative classification test error, the comparative number of RVs and the comparative runtime for each benchmark dataset and each algorithm. The standard deviation of each measure is also presented in the tables.

Table 5.3.2.3.1 The results of the classification experiments

Data set name	**Classification test error (%)**		**# RVs**		**Time(sec)**	
	Basic RVMC	**Working SetC**	**Basic RVMC**	**Working SetC**	**Basic RVMC**	**Working SetC**
Breast cancer	29.7±6.5	30±7.5	3.4±1	3.8±0.64	1.8±0.12	1.6±0.2
PimaIndian diabetes	23±4.1	22±3.9	9.8±1.9	10±1.5	28.4±2.4	9.2±0.9
German	26.5±4.6	25.9±4.2	25.1±5.4	24.7±3.5	73.9±3.3	24.7±1.5
Titanic	21.6±0.023	21.6±0.023	503.9±5.4	458.9±85.2	1344±287	597±145
Banana	9.45±0.63	9.52±0.56	18.4±1.27	18.1±1.6	4066±194	389±30

Analysis of the results indicates that:

- The working set algorithm obtained a similar accuracy to that of the basic RVC, thus it is correct.
- The number of RVs is also similar, thus retaining the sparseness of the solution.
- The time to convergence is much shorter in all cases, therefore acceleration is achieved.

5.4 Summary and Discussion

The basic RVC algorithm has some very attractive properties; however, in its original form it is too complex for any realistic medium or large datasets.

In this chapter we introduced a working set algorithm for accelerating the basic RVC via splitting the dataset and consecutive applications of the basic RVC. Complexity analysis indicates that the partition algorithm reduces the runtime complexity to $O(N^2)$ and the statistical analysis via non linear regression verifies the reduction. Simulation experiments on benchmark datasets also indicate that the working set algorithm indeed accelerates the RVM, while retaining the accuracy and the sparseness of the basic RVC.

We also proposed a heuristic for selecting a reasonable partition size. Although tuning the multipliers and the thresholds of the size partition heuristics might required, the heuristics obtained very good results by achieving reasonable partition size.

Analyzing the working set classification algorithm with the MATLAB profiler demonstrated that the most computationally intensive part is the optimization process of (5.1.1.2.1) which attempts to locate the mode of the posterior distribution for every $\mathbf{w}_{MAP}$ (in the calculation of the Hessian) . This optimization consumes about 60% of the classifier training time. There may be less expensive methods to perform those computations (alternatively, at least in the initial iterations, an approximate solution may be sufficient). Thus, the working set algorithm may be further improved.

This chapter provides another essential step for popularizing the RVM algorithm, so that eventually it will turn into another "standard tool" for the practitioner of machine learning.

6. SELECTING THE KERNEL FUNCTION

The quality of the RVM solution is dramatically affected by the kernel function and its parameters. In the previous chapters we used the "universal" kernel function – the Gaussian – and its parameter was usually tuned via cross-validation of the training set. The RVM is supposed to work with any kernel function [20], however, to the best of our knowledge, there is no published work about using another kernel other the Gaussian. Obviously there could be kernels more suitable than the Gaussian, which will make the training phase more stable from a numerical point of view and achieve sparser results [11] and maybe better accuracy. The purpose of this chapter is to investigate the sensitivity of the RVM to the kernel choice. Another issue is whether there are kernels which are more suitable for a specific task regarding the characteristics of the data set such as noise data and non linearity.

Sections 6.1 and 6.2 introduce kernels, and specifically their use in the RVM; section 6.3 presents some experimental results; section 6.4 concludes with a discussion.

6.1 Introduction to Kernels and the Kernel Trick

The use of kernels has received considerable attention in machine learning [11]. The kernel matrix is symmetric and positive definite matrix, thus, it can be defined as some kind of similarity between pairs of data points such as $k(x,x') := \langle \Psi(x), \Psi(x') \rangle$. The transform $\Psi : X \to H$ from *X*, the input space to *H*, is often used to embed the training data into a high dimensional feature space. The assumption is that it could be easier to obtain the solution for a specific problem in the feature space. Using this technique, there is no restriction on the dimensionality of the data, since usually the number of examples *N* is much larger than the number of dimensions *d*.

The freedom in the choice of the mapping Ψ enables us to design a large variety of similarity measures adapted to the given problem [11]. In practice, most applications of kernel methods just use the Gaussian kernel $k(x,x_i) = \exp^{-\|x-x_i\|^2/2\sigma^2}$ where the $\|\|$ operator indicates the distance between the any two points, and σ is the width parameter of the Gaussian. The Gaussian kernel is also called the universal kernel. It is an infinitely smooth function, thus, may not be the best choice for spike functions.

Figure 6.1.1 present the concept of the kernel trick via a classification example: a linear classifier in three dimensional feature space is equivalent to a non-linear classifier in two dimensional input space.

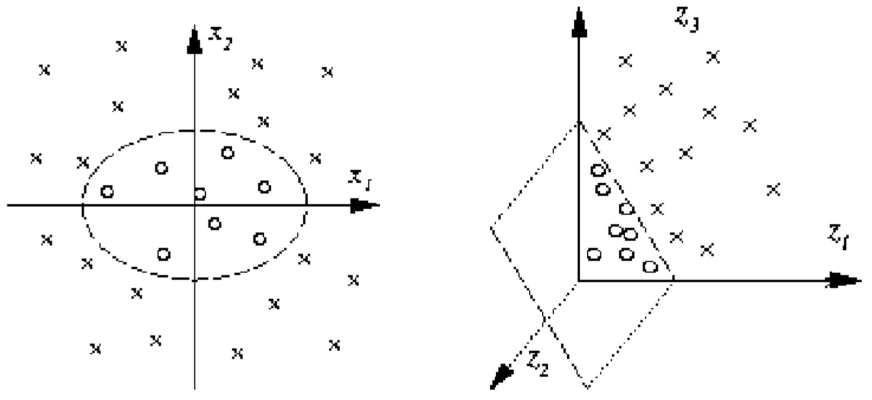

Figure 6.1.1 The idea of the kernel trick is to map the training data into a higher dimensional feature space via the kernel function, and construct a separating plane in this feature space. This yields a nonlinear decision boundary in input space. In the following two-dimensional classification example, the transformation is $\Psi : \mathbf{R}^2 \to \mathbf{R}^3$, $(x_1, x_2) \to (z_1, z_2, z_3) \equiv \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$. The decision boundary drowns as a surface and can be analytically found. Figure adapted from Muller et al. (2001).

6.1.1 Types of Kernels

Many classes of kernel function were investigated in machine learning. The main classes are anisotropic stationary kernels, isotropic stationary kernels, compactly supported kernels, locally stationary kernels, non stationary kernels and separable no stationary kernels. For further details about those kernels families look in [11]. Some commonly used kernels are presented in table 6.1.1.1.

The two kernel functions we are mostly interested in are the Gaussian and the Matern.

Table 6.1.1.1. Commonly used kernel functions

Kernel	$K(\mathbf{x}, \mathbf{x}_i)$
Radial Basis Function	$\exp\left(-\gamma\lVert\mathbf{x}-\mathbf{x}_i\rVert^2\right), \quad \gamma > 0$
Inverse multi quadratic	$\frac{1}{\sqrt{\lVert\mathbf{x}-\mathbf{x}_i\rVert + \eta}}$
Polynomial of degree *d*	$\left(\gamma\left(\mathbf{x}^T \cdot \mathbf{x}_i\right)+\eta\right)^d, \quad \gamma > 0$
Sigmoidal	$\tanh\left(\gamma\left(\mathbf{x}^T \cdot \mathbf{x}_i\right)+\eta\right), \quad \gamma > 0$
Linear	$\mathbf{x}^T \cdot \mathbf{x}_i$
Gaussian	$\exp\left(-\frac{\lVert x-x_i \rVert^2}{2\sigma^2}\right)$
Matern	$M(x,x_i) = \frac{2(\sqrt{v}/\sigma \lVert x-x_i \rVert)^v}{\Gamma(v)} K_v(2\sqrt{v}/\sigma \lVert x-x_i \rVert)$

6.1.1.1 The Gaussian kernel

The most used kernel for the RVM, as well as for other kernel based methods (such as SVM), is the Gaussian kernel function which is related to the family of isotropic stationary kernels.

$$k(x,x_i) = \exp-\left(\frac{\lVert x-x_i \rVert^2}{2\sigma^2}\right) \quad (6.1.1.1.1)$$

where the || || operator indicates the similarity between the two points, and σ is the width parameter of the Gaussian. Different widths will generate different Gaussians as illustrate in figure 6.1.1.1.1.

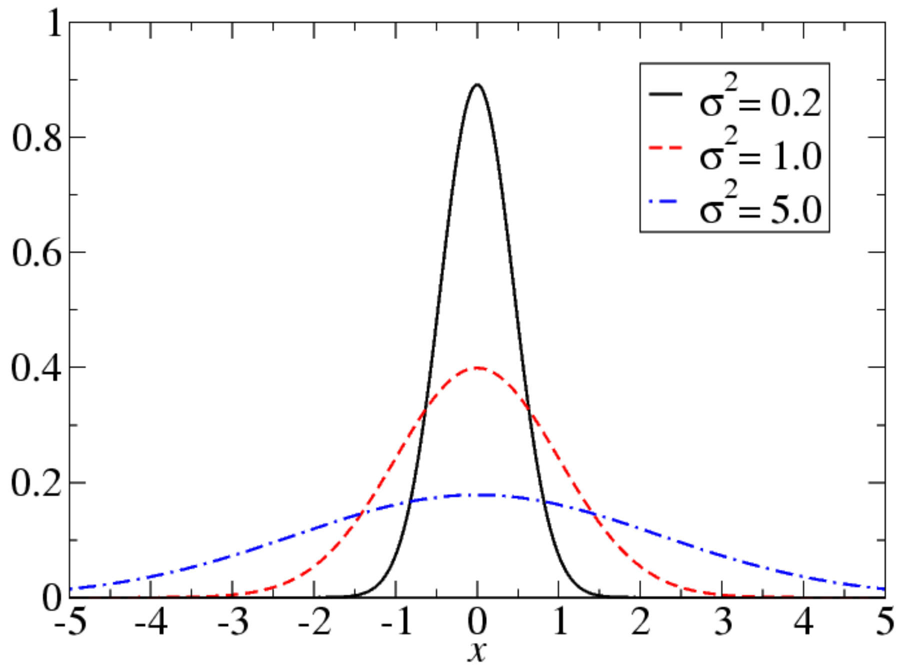

Figure 6.1.1.1.1 The Gaussian kernel functions over different width parameter.

The Gaussian function is a versatile kernel and has the ability to smooth the desired function in the feature space. Also it is empirically known to be a good choice in many kernel based methods. The Gaussian function has one parameter to tune - the width of the Gaussian (σ^2). If the width is too large, it may not be possible to closely fit the dependent variables of the model we wish to learn, and it may lead to a long training time and poor prediction model. On the other hand, if the width is too narrow, over fitting may occur around each training example leading to a non-sparse solution with very poor generalization capabilities. For further information of the affect of the width parameter look in [3] where they developed RVM with an adaptive kernel which maximize the likelihood function with respect to not only the weights and the noise variance in regression problems but also with respect to the parameters of the kernel function such as the width in the Gaussian function. This RVM adaptive kernel method obviously increased the computational complexity of the training model.

6.1.1.2 The Matérn kernel

The Matérn kernel [16], [11] is unique because it has an extra parameter ν to explicitly control the smoothness of the kernel. The Matérn function of order ν belongs to the class of functions bounded by a polynomial of order ν. One formulation of the Matérn kernel takes the following form:

$$M(x, x_i) = \frac{2(\sqrt{v}/_{\sigma} \| x - x_i \|)^v}{\Gamma(v)} K_v(2\sqrt{v}/_{\sigma} \| x - x_i \|) \quad (6.1.1.2.1)$$

where $\Gamma(v)$ is the gamma function and $K_v(x)$ is the modified Bessel function of the second kind of order v, and σ in this case is the width scaling parameter of the Matérn function.

When $v \to \infty$, the Matérn kernel degenerates to the Gaussian kernel and when v=0.5 it degenerates to the exponential kernel $\exp(-\sqrt{2}/_{\sigma} \| x - x_i \|)$. Thus, the Matérn kernel is able to define a wide range of kernel functions. Figure 6.1.1.2.1 illustrates the Matérn kernel with different degrees of smoothness and its ability to behave as the Gaussian and the exponential kernels.

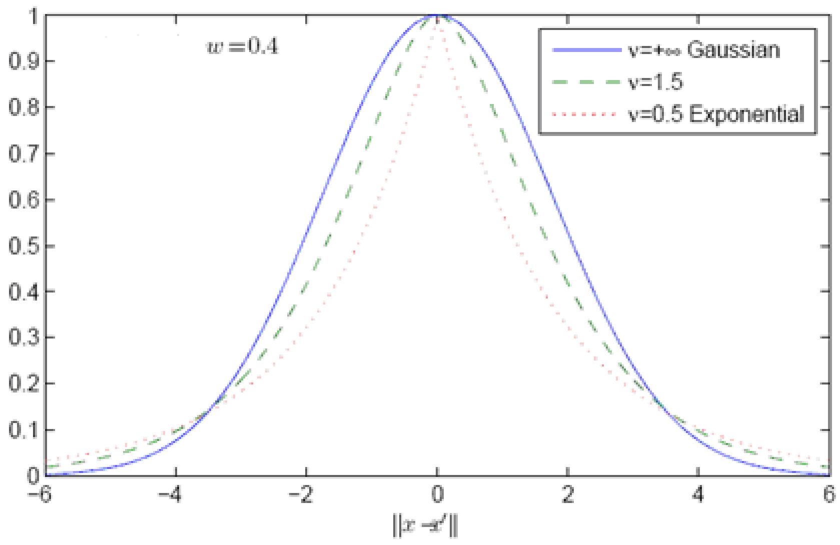

Figure 6.1.1.2.1 The Matérn Kernel; *w* denotes the standard deviation (the width) and *v* is the smoothness parameter.

6.1.1.3 The Finitely Support Matérn Kernel

Compactly supported kernels are kernels that are nonzero only over a finite supported domain [a, b]. The typical kernel measures the distance/similarity between any two points in the data. In a finitely supported kernel, whenever the similarity between two points x_i and x_j is below a certain threshold, we set the corresponding value in the kernel matrix to zero. In a typical dataset, most data is distributed among separate clusters in the multi-dimensional space. Effectively, a data is similar only to data in the same cluster, and will be considered not similar to data from different clusters. Basically, we will obtain a kernel matrix containing a majority of zero value in it (a sparse matrix). There exist efficient sparse linear algebra and sparse matrix computation techniques [12] that can reorder the nonzero values to be around the diagonal of the kernel matrix and can invert a sparse matrix in $O(N^2)$. Simply truncating the kernel below a certain threshold does not result in a positive definite matrix in general. However, any kernel can easily become a compactly supported kernel by multiplying it with the "hyper-triangular" kernel:

$$\max\left\{\left(1-\frac{\| x-x_i \|}{\sigma}\right)^{v} \ , 0\right\} \qquad (6.1.1.3.1)$$

where $\sigma > 0$ is a width parameter (same as the one used in the regular Matérn kernel) and $\mathrm{v} \geq (\mathrm{d}+1)/2$ in order to insure positive definiteness (*d* is the dimensionality of the data which generated the kernel matrix).

6.2 The RVM as a Kernel Based Method

The RVM is a kernel based method. In a typical machine learning problem we have a data set D, which contains a pairs of input observations along with the corresponding values y. $D = \{(x_1, y_1),(x_2, y_2),(x_3, y_3)....(x_N, y_N)\}$. Using the RVM there is an assumption that there is some kernel function $k(x_i, x_j)$ such that for every input value x_i the corresponding $\mathrm{y_i}$ value can be expressed as a weighted sum of the form:

$$(6.2.1)\ y(x_i) = \sum_{n=1}^{N} w_n * k(x_i, x_n) + \varepsilon$$

This form can be expressed in a notational form as $\mathbf{y} = \mathbf{\Phi w} + \boldsymbol{\varepsilon}$, where $y = (y_1, y_2, ... y_n)^T$, $w = (w_1, w_2, ... w_n)^T$, $\varepsilon = (\varepsilon_1, \varepsilon_2, ... \varepsilon_n)^T$ and

$$\mathbf{\Phi} = \begin{bmatrix} \mathrm{K}(\mathrm{x}_1, \mathrm{x}_1) & \mathrm{K}(\mathrm{x}_1, \mathrm{x}_2) & \cdots & \mathrm{K}(\mathrm{x}_1, \mathrm{x}_n) \\ \mathrm{K}(\mathrm{x}_2, \mathrm{x}_1) & \mathrm{K}(\mathrm{x}_2, \mathrm{x}_2) & \cdots & \mathrm{K}(\mathrm{x}_2, \mathrm{x}_n) \\ \vdots & \vdots & \vdots & \vdots \\ \mathrm{K}(\mathrm{x}_n, \mathrm{x}_1) & \mathrm{K}(\mathrm{x}_n, \mathrm{x}_2) & \cdots & \mathrm{K}(\mathrm{x}_n, \mathrm{x}_n) \end{bmatrix}$$

$\mathbf{\Phi}$ could be expanded to include a possible bias term. The noise terms are assumed to be zero mean Gaussian random variables with the same variance σ^2 that is: $\varepsilon_i \sim N(0, \sigma^2)$. As a result, the likelihood (conditional probability function) for each sample is also Gaussian:

$$p(y \mid \mathbf{w}, \sigma^2) = N(\mathbf{y} \mid \mathbf{\Phi w}, \sigma^2 \mathbf{I}_n) \qquad (6.2.2)$$

Using all the *N* basis function in the model may lead to severe over fitting. However, in kernel based regression models such as (6.2.1) the assumption is that the final model will use only a subset of the basis function thus over fitting is negligible.

The sparse representation of the model is obtained via a carefull choice of some priors for the values that $\mathbf{w}$ might take. These priors are responsible for the sparse representation. Tipping [21] used a zero mean Gaussian prior for each coefficient element in $\mathbf{w}$, so called $\boldsymbol{\alpha}$, and the probability to get some w_i given the corresponding α_i is also Gaussian.

$$p(w_i \mid \alpha_i) = N(w_i \mid 0, \alpha_i^{-1}) \tag{6.2.3}$$

As hyper parameters for α and σ^2, a broad Gamma distribution is chosen:

$$\begin{aligned} p(\alpha_i) &= Gamma(\alpha_i \mid a, b) \\ p(\sigma^2) &= Gamma(\sigma^2 \mid c, d) \end{aligned} \tag{6.2.4}$$

To make these priors non informative we might fix the parameters to small values e.g. a=b=c=d=10^{-5} . However in practice the RVM maximize the log product of the marginal likelihood (type II maximization) and the priors with respect to $\log(\mathbf{w})$ and $\log(\sigma^2)$. Thus practically we can use non informative *uniform* hyper priors over logarithmic scale and set these parameters to zero since in the log space the derivatives of those parameters disappear. Furthermore, setting those values to 10^{-5} in numerical experiments did not affect the convergence.

Using Gamma hyper prior for each α_i and integrating out the α_i generates a Student-t marginal prior for w_i .

$$p(w_i) = \int p(w_i \mid \alpha_i) p(\alpha_i) d\alpha_i = \frac{b^a \Gamma(a+0.5)}{(2\pi)^{0.5}\Gamma(a)}(b + w_i^2 / 2)^{-(a+0.5)} \tag{6.2.5}$$

So now for each w_i there is a student-t marginal prior and the prior for the $\mathbf{w}$ vector is a product of those student-t marginal priors. With such a prior the probability mass is concentrated in the origin and along the spines of the student-t distribution although most of the weights become zero [21]. Figure 6.2.1 illustrates the priors and the product of student-t in two dimensions example.

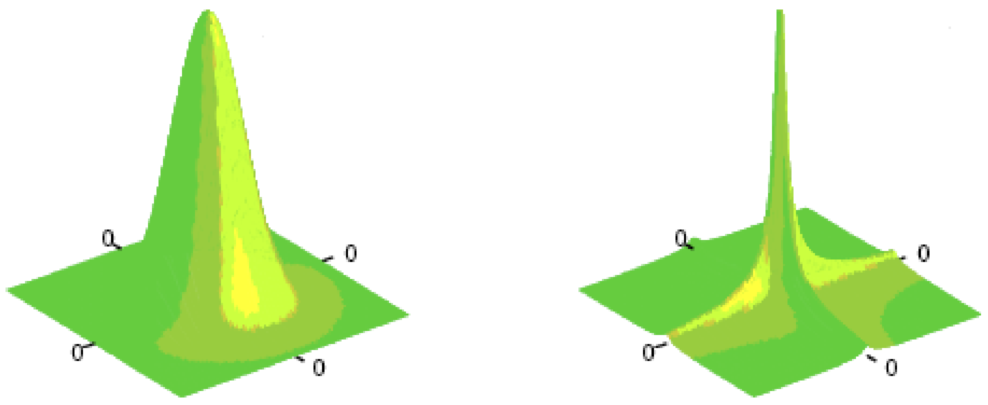

Figure 6.2.1 The left sketch is a two dimensional example of the Gaussian prior $p(w \mid \alpha)$ and the right sketch is the posterior $p(w)$ after the hyper parameters, α, have been integrated out to provide a product of student-t distributions. Figure adapted from [21].

Maximizing the likelihood function of data with respect to $\mathbf{w}$ and σ^2 (noise variance) using the priors described above causes many of the α_i to reach infinity. Thus, from (6.2.3) it is implies that the related w_i is zero. This procedure also known as Automatic Relevance Determination (ARD) prior which lead to sparse results and the results are a prototype of the entire distribution.

In the SVM, the choice of the kernel must satisfy Mercer's condition. The mercer condition simply says that g(x) has to satisfy the following condition for all square integrable functions:

$$\iint K(x,y)g(x)g(y)dxdy \geq 0 \qquad (6.2.6)$$

Theoretically, the RVM - unlike the SVM - is supposed to work with any desirable kernel function since the convergence of the model - its weights - is not related to the kernel function. However, we suspect that in practice some of the kernel functions may lead to a poor convergence, and limited sparsity. Furthermore, choosing improper basis functions may lead to numeric instability even in small data sets. Numeric instability occurs in the matrix inversion we introduced in formula (2.5.5) for the regression case and formula (5.1.1.2.1) for the classification case. More on numerical instability is available in appendix D.

Our assumption is that some of the kernels may lead to a poor convergence due to the smoothness (as defined by the number of derivatives) of the kernel function. The density of the weight vector has student-t "spines" (6.2.5) for each variable for the infinitely smooth Gaussian. A less smooth kernel function may lead to "extra spines" in the density (figure 6.2.1), which may slow down the convergence and may lead to poor sparsity. It is also interesting to test finite supported kernel functions in the RVM, since they generate sparse matrices. Using sparse matrix algebra in the RVM can potentially accelerate the algorithm, however, it is not clear if the RVM will converge for such a kernel. The RVM uses the kernel to construct a density function (the likelihood), and the finite supported kernel will generate zero density mass in some regions of the multi-dimensional space. The derivative of the density is zero in the regions of zero probability density. If the algorithm somehow arrives to a region with zero derivatives, it may not be able to use the derivative information for convergence.

Those reasons motivated us to compare the results of the Gaussian kernel function to several variations of the Matérn kernel. As illustrated in figure 6.1.1.2.1 using the Matérn kernel we can control the smoothness of the function as well as the continuity of the function via the order for the Bessel function.

6.3 Comparative Experiments

The purpose of the following experiments is to check the sensitivity of the RVM to the kernel choice via comparative experiments between the Gaussian kernel and the Matérn kernel, since in the Matérn kernel it is possible to control the smoothness of the kernel function. Moreover, we will also test the finitely supported kernel function.

6.3.1 Experiment Description

We used the implementation of the working set RVM from chapter four. The size of the partition was determined via the heuristics described in chapter four, except for the Matern1 experiments where we used a partition size of 400 (since the solution was not sparse, and in such a case the regular heuristics does not provide any improvement).

The following types of the kernel functions were considered: Gaussian kernel, Matérn of order one (Matern1), Matérn of order two (Matern2), Matérn of order three (Matern3) and Matérn of order four (Matern4). Higher orders Matérn are not that different than the Gaussian.

The Training set was used for the learning phase, and the error was measured on the testing set. Each kernel was simulated for 10 different repetitions (the same randomizations were used for testing each kernel). The width parameter for each kernel was optimized manually via cross-validation experiments.

6.3.2 Data Sets and Evaluation Metrics

Instead of taking a set of unrelated benchmark data sets, we used the *Pumadyn* family of datasets (appendix A) in order to see if there is any difference among the kernels that is dependent on the type of the data sets (non linear, high noise). Table 6.3.2.1 present the details about the datasets and the split between the training set and the test set.

Table 6.3.2.1 Details of the Pumadyn family of datasets.

Name	**Size**	**# of Attributes**	**Level of noise**	**Level of non linearity**	**Training set /test set**
8fh	8192	8	high	fairly linear	6144/2048
8fm	8192	8	moderate	fairly linear	6144/2048
8nh	8192	8	high	non-linear	6144/2048
8nm	8192	8	moderate	non-linear	6144/2048
32fh	8192	32	high	fairly linear	6144/2048
32fm	8192	32	moderate	fairly linear	6144/2048
32nh	8192	32	high	non-linear	6144/2048
32nm	8192	32	moderate	non-linear	6144/2048

The measures we used in this experiment were; the number of RVs and the accuracy (RMSE).

6.3.3 Results and Conclusions

The following tables present the comparative RMSE and the comparative number of RVs respectively. The standard deviation of each measure is also presented in the table**.**

Table 6.3.3.1 Comparative RMSE

Name	Gauss	Matern1	Matern2	Matern3	Matern4
puma8fh	3.14±0.05	3.14±0.04	3.14±0.05	3.16±0.06	3.17±0.04
puma8fm	1.05±0.018	1.07±0.018	1.05±0.018	1.05±0.017	1.23±0.026
puma8nh	3.25±0.043	3.24±0.06	3.22±0.036	3.25±0.03	4.25±0.06
puma8nm	1.22±0.016	1.18±0.02	1.14±0.02	1.2±0.024	3.54±0.034
puma32fh	$0.02\pm3*10^{-4}$	$0.021\pm3*10^{-4}$	$0.02\pm3*10^{-4}$	$0.02\pm3*10^{-4}$	$0.02\pm3*10^{-5}$
puma32fm	$0.005\pm6*10^{-5}$	$0.005\pm7*10^{-5}$	$0.005\pm7*10^{-5}$	$0.005\pm9*10^{-5}$	$0.005\pm1*10^{-5}$
puma32nh	$0.033\pm5*10^{-4}$	$0.034\pm7*10^{-4}$	$0.033\pm5*10^{-4}$	$0.033\pm4*10^{-4}$	$0.033\pm3*10^{-5}$
puma32nm	$0.027\pm5*10^{-4}$	$0.028\pm5*10^{-4}$	$0.027\pm5*10^{-4}$	$0.027\pm3*10^{-4}$	$0.027\pm5*10^{-5}$

Table 6.3.3.2 Comparative number of RVs

Name	Gauss	Matern1	Matern2	Matern3	Matern4
puma8fh	38.5±2.9	46.7±5.3	36.5±2.5	34.2±1.7	12.3±1.4
puma8fm	70.6±3.1	136.1±10.3	79.8±3.22	57.2±2.6	16.1±1
puma8nh	142.4±8	160.7±8.5	93±2.47	45.85±5.8	19.3±1.41
puma8nm	82.8±2.5	530.4±85.4	173.3±6	82.76±3.9	17.7±1.25
puma32fh	79.1±3.7	240.8±37.1	41±11	38.6±3.4	11.3±2.5
puma32fm	140±13.4	265.1±14.3	64±7	34.5±4.3	17±4.6
puma32nh	9.9±1.7	268.3±19.5	28±21.5	7.3±1.6	7.3±1.4
puma32nm	67.7±5.2	324±10	31.1±12.5	16.9±8.6	8.77±2.6

Analysis of the results indicates that:

- All the kernels beside the Matern4 achieved similar accuracy. In three datasets the Matern4 damage the accuracy in a significant way.
- Although further experiments are necessary, the Matern4 achieved the sparsest results over most of the data sets but its accuracy is less good than the others. The Matern3 achieved the sparser results than Matern1, Matern2 and the Gaussian. We already know that when the order of the Matérn is reaching ∞ the Matérn degenerates to the Gaussian kernel. It may be that a Matérn from higher order than four may achieve even sparser results, however, the accuracy can be damaged more as being observed in the Matern4.
- We also trained the finitely supported Matérn kernel of order 5 (due to the constraints on the order in (6.1.1.3.1)) with the RVM for the first four pumadyn datasets. The results were that none of the experiments with finitely supported kernels converged to a solution. (in the one case it did converge, the kernel was not sparse).
- Regarding which kernel is more suitable for a given problem (non-linearity, noise) it is hard to decide. However the Matérn of order 3 obtained the sparsest results compared to the Matern1, Matern2 and the Gaussian and better accuracy compared to the Matern4. Thus, it is recommended.

The main contribution of these experiments is that we found a kernel which is better than the Gaussian - we suggested using the Matérn of order 3.

6.4 Discussion

The problem of selecting the best kernel and tuning its parameters lies in the core of all the kernel based methods. In this chapter we investigated for the first time the kernel issues related to the RVM.

The finite support kernel, which is a compactly supported kernel, is able to produce a sparse matrix and as a consequence from that to reduce the computational complexity by using sparse linear algebra techniques. Surprisingly, when we trained the RVM using the finite support Matérn order 5, the results were very poor convergence if any, and a very long training time respectively to the ordinary Matérn kernel. Even when convergence was obtained, the kernel was not sparse, thus we could not use sparse linear algebra in those experiments. One possible explanation to the poor results could be the regions of zero derivatives of the finite support kernels which causes the RVM to slow down, or stop the convergence.

Further experiments are required for the finite supported Matérn kernel with different order values. We did not manage to answer the question if there is a kernel that is best suited for a given problem (non-linear, noise etc') or whether kernels for classification should be different than kernels for regression. However, this is the first time (to the best of our knowledge) that a kernel different than the Gaussian, or specifically the Matérn family, was used for the RVM. It turns out – unlike conjectured by [21] – that not every kernel is suitable for the RVM.

7. SUMMARY AND DISCUSSION

The purposes of this chapter are to overview this dissertation, its main contributions, and discuss some open issues.

7.1 The RVM

The relevance vector machine is a state-of-the-art machine learning algorithm. It is a probabilistic sparse kernel model which takes a place in a Bayesian framework. The RVM addresses two important problems in the machine learning, the first one is regression and the second is classification. The goal of the RVM is to accurately predict the next value for a new input observation via a generalized linear model (GLM) that performs a nonlinear projection of the input space into transformed space by means of a set of nonlinear basis functions. The final trained RVM model is given in (7.1.1). There are some minor modifications to the classification problem as described in sub-section 5.1.1.

$$y(x) = \sum_{n=1}^{N} w_n * k(x, x_n) + w_0 \tag{7.1.1}$$

Here $\mathbf{w}$ presents a vector of the regression coefficients. N is the number of the basis functions in the final model. $k(x, x_n)$ is a bivariate kernel function which transform the input space into the feature space. The goal of the RVM is to learn appropriate values for the $\mathbf{w}$ vector we introduce in (7.1.1). The algorithms starts with a prior over $\mathbf{w}$ introduced in (7.1.1) and governed them by a set of hyper priors produced from a gamma distribution, one for each weight.

7.1.1 Strengths of the RVM

Beside the ability of the RVM to produce very accurate GLMs, it benefits from some additional attractive properties.

- The RVM effectively performs Automatic Relevance Determination (ARD); during the learning phase of the model, many of the hyper parameters reached infinity and their corresponding weights are pruned. Thus, the final model is based only on a subset of the data and often it is a very small subset. The sparse solution does not depend on the size of data set.
- Most of the algorithms which produce GLMs are lack a probabilistic interpretation of its predictions. The RVM is a fully probabilistic model which produces not only

the mean of the distribution but also its standard deviation, thus we can compute a confidences interval.

- Machine learning models usually require an expensive cross validation in order to avoid over fitting and provide a probabilistic interpretation. The RVM does not need cross validation since it has a built in probabilistic interpretation.
- Theoretically, there is no restriction on the kernel function used in the RVM. However, as discovered in chapter six, not every kernel function can lead to convergence and sparsity.

7.1.2 Weaknesses of the RVM

Although the RVM benefits from several advantages comparable to other machine learning algorithm it still suffers from a number of disadvantages.

- Updating the hyper parameters depend on computing the posterior weight covariance matrix. This computing requires an inverse matrix operation which in practice is implemented via Cholesky decomposition. The inverse matrix operation is of order $O(N^3)$. The kernel matrix is of full rank, thus, the required memory storage is $O(N^2)$. N denotes the size of the dataset in both cases. These complexity issues - the main trust of this dissertation - made the basic RVM impractical for average datasets (several thousands) and upwards.
- The RVM is a kernel based method. There is not yet a methodology for kernel selection.
- There is a need to tune the kernel parameters. Effectively running the RVM many times during the tuning.
- The algorithm stops when the likelihood is ceases to increase. It is possible that the maximum is local and not absolute, thus it can lead us to a sub-optimal model.
- The solution of the RVM is not unique. Although the relevance vectors are seems to be more like prototypes of the data it is possible to get different solutions for two consecutive trainings on the same data set if we permute the data. However, the accuracy will remain the same.

7.2 Novelty and Contributions of this Study

7.2.1 The First Contribution

The primary goal of this study was to make the RVM feasible for large data sets. In order to achieve the goal we obtained an order of magnitude complexity reduction using the partition algorithms we developed. We reduced the complexity of the algorithm for both the regression problem and the classification problem. The problem of the storage complexity can also be easily addressed.

For the regression problem (chapter four), we suggested three acceleration algorithms for training the RVM while achieving the accuracy and the sparseness of the origin solution. Complexity analysis of those algorithms indicate that the partition algorithms reduced the complexity of the RVM from $O(N^3)$ to $O(N^2)$. Simulations on regression benchmark datasets indicate that the three partitions heuristics indeed accelerate the RVM while not damaging the accuracy of the solution. We also proposed a heuristic for selecting a reasonable partition size for each data set. Simulation demonstrates the quality of the heuristics. Statistical analyses, via non-linear regression, for the working set runtime complexity and for the basic RVM runtime complexity also demonstrate the reduced complexity of the partitions algorithms. In fact the conclusion from the three approaches (complexity analysis, simulations experiments and statistical analysis) mutually supported each other.

Regarding the classification problem, chapter five suggests a version of the working set partition algorithm for classification. Complexity analysis indicate that the partition algorithm reduce the complexity of the RVM from $O(N^3)$ to $O(N^2)$. Measuring the acceleration rate on different sizes of synthetic dataset support the complexity reduction as well; statistical analysis via non-linear regression demonstrates significant reduction. Simulations on natural classification problems indicate that the working set achieved the same solution as the basic RVM while doing it in a much shorter time. Again the conclusions from the three approaches (complexity analysis, statistical analysis and simulations experiments) mutually support each other in classification case as well.

Regarding the best partition algorithm; since none of the partition algorithms demonstrate superiority over all the datasets and measures we cannot say for sure which one of the partition algorithms is the best. However, the working set algorithm is expected to provide a good solution even if it has to be prematurely aborted due to time constraints, furthermore the possibility that it is deleting 'good' basis functions is small.

7.2.2 The Second Contribution

The second goal of this study was to study the limits of the RVM regarding the type of data sets; non-linearity, noise and high dimensionality.

The results in sub-section 4.4.1 indicate that the RVM is capable of finding good solution for any type of data (non-linear, noisy, high number of dimensions).

7.2.3 The Third Contribution

The third contribution of this study was to examine for the first time the sensitivity of the RVM to the kernel choice, and specifically the Matern kernel. We wanted to know if the Matern kernel is a better choice than the Gaussian for the RVM. Another kernel family tested was the finite support Matern kernel – which can potentially introduce sparse linear algebra - and as consequence of that further reduction in the complexity of the RVM.

The simulations results in chapter six indicate that when we are using the Matern kernel as a kernel function for the RVM it is suggested to use Matern kernel of order 3. The Matern of order three achieved better results from the Gaussian thus it is recommended. In practice we noticed that creating the $\mathbf{\Phi}$ matrix using Matern kernel is more time consuming than creating the $\mathbf{\Phi}$ using the Gaussian kernel function. Although further experiments are needed; while training the RVM with the finite support Matern kernel from order one, the algorithm did not converge.

7.2.4 Practical Conclusions and Recommendations

- Using our partition algorithms, the RVM is now capable of handling large data sets. We recommend using the working set algorithm because if we exhaust the time or space constraints, we can stop and still obtain a decent solution.
- As a complementary part to working set algorithm we suggest using the partition size heuristics which was introduced in chapter five for the regression and in chapter six for the classification problem.
- The RVM is capable of handling all kind of problem; non-linearity, noisy data and datasets of high dimensionality.

- It is recommended to consider Matern kernel of order three as potential for further sparsity.
- The memory storage of the RVM remains $O(N^2)$ since we are using kernel function which centered on each of the N training data point, thus, we use a $N \times N$ matrix. For the partition algorithms it is not necessary to generate the full kernel matrix. We can generate the kernel matrix on the fly only for the active partition and as a consequence reduce the storage complexity to $O(P^2)$. In the update of the noise variance (formula 2.5.7) we compute the error for the full dataset. However, in practice we notice that calculating the error only for the active partition damages the accuracy only by little bit. If one has a memory constraint it is advisable to compute the error only on the partition in memory, therefore to produce the kernel matrix only for the active partition. Note that this suggestion might lead to an overfitted model.

7.3 Suggestions for Future Research

The choice of which kernel is suitable for a given problem and the optimization of its parameters is lying in the core of all the kernel based methods. Finding the most suitable kernel will increase the usage of those methods and will make them standard tools for the machine learning and data mining practitioners. Trying to do so it is recommended combine the kernel selection with the tuning of its parameters. May be it is sufficient to tune the kernel parameters only for a subset of the data.

As we explained before, computing the error for only the active partition reduces the storage complexity to $O(P^2)$ and may have only a minor affect (if any) on the accuracy of the model. This idea should be thoroughly investigated for the absence of over fitting. This could lead to further decrease not only in the storage complexity, but also in the runtime complexity as well, since the error calculation requires $O(N^2)$ operations.

In our study we simulate the finite support Matern kernel from order one and the results were that the RVM did not converge. It is important to find out if there exists finite supported Matern kernel or any other compactly supported kernel which can achieve convergence while not damaging the accuracy of the RVM model. It will open the possibility of using sparse linear algebra in the RVM and as a consequence from that obtaining additional complexity reduction of the algorithm.

REFERENCES

[1] A. Agarwal, B. Triggs (2004). 3D Human Pose from Silhouettes by Relevance Vector Regression. *IEEE International Conference on Computer Vision and Pattern Recognition*.

[2] B. E. Boser, I. M. Guyon, and V. N. Vapnik (1992). A training algorithm for optimal margin classifiers. *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144-152. ACM Press.

[3] J. Q. Candela, L. K. Hansen (2002). Time series prediction based on the relevance vector machine with adaptive kernels. *International Conference on Acoustics, Speech, and Signal Processing*.

[4] S. Chakraborty, M. Ghosh, B. K. Mallick. Bayesian Nonlinear Regression for Large p Small n Problems. Available at: www.stat.ufl.edu/~schakrab/svmregression.pdf

[5] N. Cristianini, J. Shawe-Taylor (2003). An Introduction to Support Vector Machines and other kernel-based learning methods. *Cambridge University Press*.

[6] N. Cristianini, J. Shawe-Taylor (2000). *An Introduction to Support Vector Machines. Cambridge. Cambridge University Press.*

[7] N. Cristianini, J. Shawe-Taylor, R. C. Williamson (2001). Introduction to the Special Issue on Kernel Methods. *Journal of Machine Learning Research 2*.

[8] T.A Down, T.J.P Hubbard (2003). Relevance Vector Machines for Classifying Points and Regions in Biological Sequences. *Welcome Trust, Sanger Institute.*

[9] A. D'Souza, S. Vijayakumar, S. Schaal, (2004). The Bayesian Backfitting Relevance Vector Machine. *Proceedings of the 21st International conference on Machine Learning*, July 4-8, Baniff, Canada.

[10] J. H. Friedman (1991). Multivariate adaptive regression splines. *Annals of Statistics*.

[11] M.G.Genton (2001). Classes of Kernels for Machine Learning: A Statistics Perspective. *Journal of Machine Learning Research 2*, pages 299-312.

[12] J.R.Gilbert, C. Moler, R. Schreiber (1992). Sparse matrices in MATLAB design and implementation. *SIAM Journal on Matrix Analysis*, 13(1), pages 333–356.

[13] A. B. A. Graf, F. A. Wichmann (2004). Insights from Machine Learning Applied to Human Visual Classification. *NIPS*, Vancouver, BC Canada.

[14] M. A. Hearst (1998). Support Vector Machines. *IEEE Intelligent Systems*. University of California, Berkeley.

[15] D. J. C. MacKay (1998). Introduction to Gaussian processes. *In C. M. Bishop, editor, Neural Networks and Machine Learning,* Springer, pages 133-165.

[16] B. Matern (1960*). Spatial Variation*. New York*, Springer.*

[17] S. Matthias (2004). *Gaussian Processes for Machine Learning*. Available at: http://www.cs.berkeley.edu/~mseeger/papers/bayesgp-tut.pdf

[18] C.E. Rasmussen, J.Q. Candela (2005). Healing the Relevance Vector Machine through Augmentation. *Proceedings of the 22nd International Conference on Machine Learning*, August 7-11, Bonn, Germany.

[19] M E.Tipping, C. M.Bishop (2000). Variational Relevance Vector Machine. *Microsoft research Cambridge UK*.

[20] M. E. Tipping (2000). The Relevance Vector Machine. *Advances in Neural Information Processing Systems 12*. Cambridge, Mass: MIT press.

[21] M. E. Tipping (2001). Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research 1*, pages 211-244.

[22] M. E. Tipping, A. Faul (2003). Fast Marginal Likelihood Maximization for Sparse Bayesian Models. *Proceedings of the 9th International workshop on Artificial Intelligence and Statistics*. January 3-6, Key West, Florida

[23] V.N. Vapnik (1995). The Nature of Statistical Learning Theory. *2nd edn. New York, Springer.*

[24] V.N Vapnik, S.E. Golowich, A.J. Smola (1997). Support vector method for function approximation, regression estimation and signal processing. *Advances in Neural Information Processing Systems* 9. MIT Press.

[25] O. Williams, A. Blake, R. Cipolla (2003). A Sparse Probabilistic Learning Algorithm for Real-Time Tracking. *9th IEEE international conference on computer vision*.

[26] http://www.cs.toronto.edu/~roweis/notes/nipstut.pdf

[27] http://www.relevancevector.com

Papers based on this research:

[A] D. Ben-Shimon, A. Shmilovici (2005). Accelerating the relevance vector machine via data partitioning. *Proceedings of 1st ADBIS workshop on data mining and knowledge discovery ADMKD'2005*, September 15-16, Tallinn, Estonia, pages 25-34.

[B] D. Ben-Shimon, A. Shmilovici (2006). Accelerating the relevance vector machine via data partitioning. *Accepted to the Journal of Foundations of Computing and Decision Sciences*.

[C] D. Ben-Shimon, A. Shmilovici. A Working Set Algorithm for Accelerating the Relevance Vector Machine (2006). *Submitted to the 11th conference of information processing and management of uncertainty in knowledge based-systems*. July 2006 Paris, France.

[D] D. Ben-Shimon, A. Shmilovici. Kernels for the Relevance Vector Machine – An Empirical Study (2006). *Accepted to ATDM.* Israel, Beer Sheva, June 5-7 2006. Forthcoming in Springer

APPENDICES

Appendix A. Benchmark Data Sets

The purpose of this appendix it to provide additional details regarding the dataset we used in this study.

Regression

- **Boston Housing** data set obtained from the statistical library archive at http://lib.stat.cmu.edu/datasets/boston, it contains 506 examples with 14 variables. The results are given for the task of predicting the median house value (variable 14) from the remaining 13 variables.

- **Abalone** data set obtained from the UCI machine learning repository at http://www.ics.uci.edu/~mlearn/MLSummary.html. It comprises 4177 examples with 8 variables and the results are given for the task of predicting the age of abalone (variable 8) from the remaining 7 variables.

- **Sinc** data set is generated synthetically from the function $y(x) = \frac{\sin|x|}{|x|}$ at 1000 equally-space x-values in [-10, 10] with added Gaussian noise of variance 0.1.

- **Friedman number 1** data set is also generated from synthetic function taken from Friedman (1991), the data was generate from the function: $y(x) = 10\sin(\pi x_1 x_2) + 20(x_3 - 1/2)^2 + 10x_4 + 5x_5 + \sum_{k=6}^{10} 0.x_k$, and inputs were generated at random from the 10-dimensional unit hypercube and Gaussian noise with standard deviation of 1 added to $y(x)$.

- **Pumadyn** is a family of datasets synthetically generated from a realistic simulation of the dynamics Puma 560 robot arm. These data sets obtain from: http://www.cs.toronto.edu/~delve/data/pumadyn/desc.html and there are eight datasets in this family. The task in these datasets is to predict the angular acceleration of one of the robot arm's links. Each and every one of the eight datasets have its own characteristics regarding three properties:
    - Number of inputs (8 or 32).
    - Degree of non-linearity (fairly linear or non-linear).

    - Amount of noise in the output (moderate or high)

Classification

- **Ripley's -** This data set is a synthetic data set for two class classification problem. Both classes were generated from mixtures of two Gaussians by Ripley (1996) with class overlapping to the extent that the Bayes error is 8%. The data set is two dimensional plus class attribute and contains 1250 examples. Available at: http://www.dcs.ex.ac.uk/studyRes/COM6401/
- **Breast cancer-** This data set was obtained from the University Medical Centre Institute of Oncology, Ljubljana, Yugoslavia. Thanks go to M. Zwitter and M. Soklic for providing the data. The data contains 277 examples with 9 attributes plus class attribute. Class value 1 is interpreted as "tested positive for breast cancer". The data is only for academic use thus it's restricted however via the following address it is possible to ask for it: http://www.ics.uci.edu/~mlearn/MLRestricted.html.
- **Pima Indian diabetes-** This medical data set contains 768 examples with 8 attributes plus class attribute. Class value 1 is interpreted as "tested positive for diabetes". Available at: http://www.ics.uci.edu/~mlearn/MLSummary.html
- **German-** German credit dataset classifies people described by a set of attributes as good or bad credit risks. It comprises 1000 examples with 20 attributes plus class attribute available at : http://www.ics.uci.edu/~mlearn/MLSummary.html
- **Titanic-**The titanic dataset gives the values of four categorical attributes for each one of the 2201 people on board the Titanic when it struck an iceberg and sank. The data set contains 2201 examples with 3 attributes plus class attribute which indicates whether the person survived or not. The dataset Available at: http://www.cs.toronto.edu/~delve/data/titanic/desc.html
- **Banana –** this data set comprises 5300 examples with 2 attributes plus class attribute. Available at: http://ida.first.fraunhofer.de/projects/bench/benchmarks.htm

Appendix B. Statistical Analysis of sub-section 4.3

The purpose of this appendix is to provide additional details regarding the statistical analysis described in sub-section 4.3. The following details present the output of the SPSS statistical software for the basic RVM and for the Working Set respectively.

Basic RVM

Regression Curve Estimation. Method: POWER.

Statistical Measures

Multiple R	.96222
R Square	.92588
Adjusted R Square	.91661
Standard Error	.40028

ANOVA

	DF	Sum of Squares	Mean Square
Regression	1	16.01	16.01
Residuals	8	1.28	0.16
F=99.92		Signif F=.0000	

Variables in the Equation

Variable	Value	SE B	Beta	T	Sig T
Power	2.53	0.74	0.99	33.85	.0
Coefficient	$2.7*10^{-7}$	$1.44*10^{-7}$		1.83	.1

Working Set

Regression Curve Estimation. Method: POWER

Statistical Measures

Multiple R	.978
R Square	.956
Adjusted R Square	.951
Standard Error	.325

ANOVA

	DF	Sum of Squares	Mean Square
Regression	1	18.78	18.78
Residuals	8	0.84	0.1
F=177.05		Signif F=.0000	

Variables in the Equation

Variable	Value	SE B	Beta	T	Sig T
Power	1.51	0.113	0.99	33.85	.0
Coefficient	$4.5*10^{-5}$	$3.7*10^{-5}$		1.2	.263

Appendix C. Statistical Analysis of sub-section 5.3.3

The purpose of this appendix is to provide additional details regarding the statistical analysis described in sub-section 5.3.3. The following details present the output of the statistical software SPSS for the basic RVM and for the Working Set respectively.

Basic RVM

Regression Curve Estimation. Method: POWER

Statistical Measures

Multiple R	.999
R Square	.999
Adjusted R Square	.999
Standard Error	.044

ANOVA

	DF	Sum of Squares	Mean Square
Regression	1	36.3	36.3
Residuals	8	0.015	0.0019
F=18592.48		Signif F=.0000	

Variables in the Equation

Variable	Value	SE B	Beta	T	Sig T
Power	2.74	0.02	0.99	136.35	.0
Coefficient	$6.74*10^{-7}$	$9.2*10^{-8}$		7.26	.0001

Working Set

Regression Curve Estimation. Method: POWER

Statistical Measures

Multiple R	.999
R Square	.998
Adjusted R Square	.9981
Standard Error	.048

ANOVA

	DF	Sum of Squares	Mean Square
Regression	1	14.4	14.14
Residuals	8	0.018	0.02
F=6119.3		Signif F=.0000	

Variables in the Equation

Variable	Value	SE B	Beta	T	Sig T
Power	1.71	0.021	0.99	78.22	.0
Coefficient	0.0002	$2.8*10^{-5}$		6.682	0.002

Appendix D. Numerical Stability

The purpose of this appendix is to overview the RVM numerical instability problems. Fairly often, the RVM algorithm encounters ill conditioning of matrices and simply quits after issuing an error message of 'non-positive definite matrix'.

Before we describe the numerical instability problems in the RVM we first define several terms.

Positive Definite Matrix - In order to perform inverse operation on a matrix, the matrix has to be positive definite. A positive definite matrix is a matrix where all the eigenvalues are positive. The eigenvalue problem is to determine the solutions of the equation $Ax = \lambda x$ where A is an n-by-n matrix, x is a length n column vector, and λ is a scalar. The n values of λ that satisfy the equation is the eigenvalues and the corresponding values of x are the right eigenvectors. In the context of inverting a matrix, if matrix A has a matrix B such that A*B=I then A is invert able.

Hessian - The matrix of the derivatives $df / dx_1, df / dx_2, ... df / dx_n$, of a function $f(x_1, x_2, ... x_n)$ with respect to $x_1, x_2, ... x_n$ is called the Hessian of x. In the RVM we meet the Hessian in formula (2.5.5), $\mathbf{\Sigma} = (\mathbf{\Phi}^{\mathbf{T}}\mathbf{B}\mathbf{\Phi} + \mathbf{A})^{-1}$ where the Hessian matrix is $\mathbf{\Phi}^{\mathbf{T}}\mathbf{B}\mathbf{\Phi} + \mathbf{A}$.

The covariance matrix $\mathbf{\Sigma}$ as indicated in (2.5.5) is computed via the inversion of the Hessian.

During the RVM the Hessian can appear to be non-positive definite even if analytically it is positive definite, thus, cause numerical stability problems. During the optimizations of the hyper parameters $\boldsymbol{\alpha}$ in the RVM, many of the α_i becomes infinite. In that case, a very high numerical range results between the smallest α_i and the largest α_i in a given vector. Ill conditioning of the Hessian appears when the ratio between the smallest α_i to the largest α_i is in the order of the machine precision (a large condition number – in numerical analysis terms) [21]. An easy solution could be to prune the related basis function from the model as practically happens since the related α_i reached some threshold value and as consequence of that it's pruned. However we encounter other situations where the Hessian becomes ill condition even if we choose threshold value smaller than the machine precision. The reason for that is that inversion of matrices via numerical methods is inherently unstable, due to the accumulation of numerical errors. The larger the matrix is, the higher the probability of having a large condition number, resulting in instability. The standard recommendation in numerical analysis textbooks is to replace the direct matrix inversion with inversion via Cholesky decomposition – which is more stable. We did follow this recommendation in the implementation, since it somewhat also reduces the time. However we did not follow the second standard recommendation – of computing in quadruple precision, since this step will significantly slow down the computations.

This problem of ill conditioning of the Hessian occurs only in big matrices (more than 1000). In the partition algorithms no such a case was found. Furthermore, the numerical instability is typically associated with non-optimal kernel function. In that case it is advisable to further optimize the kernel function shape parameters.

Appendix E. Reintroduction of Deleted Basis Functions

The purpose of this appendix is to provide a description of an unsuccessful experiment where we try to reintroduce deleted basis functions in the partition algorithms.

The partition algorithms are somewhat 'greedy' strategies for constructing the final model of the RVM. In theory it is possible that 'good' basis functions may be eliminated during the individual processing of each partition, thus we may not achieving the best possible model. In this experiment we attempted to add another step to the partition algorithm such that no 'good' basis function will be permanently deleted.

A sequential version of the RVM algorithm was proposed by [22]. In that algorithm instead of starting the training with all the candidate basis functions they started the training with a *single* basis function. Whenever a basis function is considered to maximize the Likelihood it was added to the model, and if not contributing to the likelihood, it was deleted. It is a highly effective method which avoids the calculation of the full Hessian and the inversion of it. It is somewhat like our partition algorithms with partition size of 1. Obviously there is overhead with deletion insertion operations, however they demonstrated a significant speed up while comparing this algorithm to the basic RVM. In this algorithm they define two measures for each basis function; its 'sparsity' factor and its 'quality' factor. The 'sparsity' measure indicates whether there is already a basis function in the model which the current basis function is overlapped with. The 'quality' measure is the amount of accuracy that the current basis function adds to the model. If the quality measure is larger than the sparsity measure then we update the corresponding weight, otherwise, we set its related weight to infinite and as consequence that basis function is considered irrelevant to the model. This iterative procedure continues for every basis function until some convergence criterion is achieved.

Implementing this kind of mechanism in our partition algorithm will remove the doubt behind the possibility that we are deleting 'good' basis function during the training of each partition. The idea is to add a step at the end of the creation of the final model. This step will compute the sparsity measure and the quality measure for every basis function which is not include in the final model. Using those measures we will give a 'second chance' to the basis functions which were not included in the last partition to be included in a further training of the last partition.

The procedure was as follow:

1. Compute the following 'sparsity' measure and 'quality measure for every i^{th} irrelevant basis function.
    a. $s_i = \mathbf{\Phi}_i^T \cdot \mathbf{\Sigma} \cdot \mathbf{\Phi} +_i \mathbf{\Phi}_i^T \cdot \mathbf{\Phi}_i$
    b. $q_i = \sigma^2 \cdot \Phi_i^T (t - y(x))$
2. If $q_i^2 > s_i$ than insert the corresponding basis function to the final model and compute the corresponding weight as: $\alpha_i = s_i^2 / (q_i^2 - s_i)$.

During estimation of $\mathbf{s}$ and $\mathbf{q}$ for the working set algorithm we found that the $\mathbf{s}$ values were very small - almost all of them around 0.005 - and the $\mathbf{q}$ values were between 3 to 2 (on a Sinc dataset comprises 1000 observations). These results are clearly pointing to the fact that there is problem with the procedure while using it on the partition algorithms. If we follow the recommendations of this procedure, we will have to re insert almost all the vectors to the solution which is of course not making any sense.

We do not really know why this effect happened, and if it depends on the type of used data. Except for [22], we do not know of other successful attempts to implement that algorithm.

Appendix F. Kernels Width Parameters

The following table presents the width parameters for all the kernel functions and datasets that have been used in this study.

Dataset name	**Gauss**	**Matern1**	**Matern2**	**Matern3**	**Matern4**
PD8fh	2	3	4	8	18
PD8fm	2	5	4	8	25
PD8nh	2	7	8	16	25
PD8nm	10	12	14	18	32
PD32fh	25	50	50	50	90
PD32fm	25	20	25	125	175
PD832nh	135	180	190	135	135
PD832nm	60	135	140	150	160
Sinc	3				
Boston	112				
Friedman#1	3				
Abalone	3				
Ripley	0.5				
Breast cancer	8				
Pima Indian Diabetes	180				
German	10				
Titanic	2.8				
Banana	0.8				