ACCELERATING THE RELEVANCE VECTOR MACHINE VIA DATA PARTITIONING

David Ben-Shimon and Armin Shmilovici[1]

Abstract. The Relevance Vector Machine (RVM) is a method for training sparse generalized linear models, and its accuracy is comparably to other machine learning techniques. For a dataset of size N 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. We suggest three different algorithms which reduce the runtime complexity to $O(N^2)$ via partitioning the dataset into small chunks of size P. A heuristic is presented for selecting the chunk size. Extensive experiments with benchmark datasets indicate that the partition algorithms can significantly reduce the complexity of the RVM while retaining the attractive attributes of the original solution.

Keywords: Machine Learning, Data Mining, The Relevance Vector Machine

1. Introduction

The Relevance Vector Machine (RVM) [5] is a method for training a generalized linear model (GLM) such as (1), where $k(x, x_i)$ is a bivariate kernel function centered on each one of the N training data points x_i, $w = [w_1, \ldots, w_N]^T$ is a vector of regression coefficients, and ε is the noise.

$$\mathbf{y(x, w)} = \sum_{i=1} w_i * k(\mathbf{x}, x_i) + \varepsilon \tag{1}$$

The processing of large real-world data still poses significant challenge to state-of-the-art supervised learning algorithms, even in linear settings. While traditional statistical techniques (such as partial least squares regression) are often efficient, they lack a probabilistic interpretation and can not easily provide predictive distributions. On the other hand, Bayesian inference based on Gaussian Processes (GP) - in its naïve implementation -

[1] Department of Information Systems Engineering, Ben-Gurion University, P.O.Box 653, 84105 Beer-Sheva, Israel{dudibs,armin}@bgumail.bgu.ac.il

lacks the computational efficiency needed for processing data of large size (more than few thousands).

Sparsity is generally considered a desirable feature of a machine learning algorithm. The RVM is a sparse method for training GLMs. This means that it will select a subset (often a small subset) of the provided basis functions to use in the final model. The RVM is named by analogy to the known Support Vector Machine (SVM) method [1], which is also a sparse GLM trainer. However, the SVM can only be applied to training GLMs with restrictions on the suitable kernel functions, while the RVM can train a GLM with any collection of basis functions. Furthermore, using the same kernel the RVM can produce sparser and more accurate solutions than the SVM [1, 5].

The goal of the RVM is to accurately predict the target function, while retaining as few basis functions as possible in $\boldsymbol{w}$. Sparseness is achieved via the framework of sparse Bayesian learning and the introduction of an additional vector of hyper parameters α_i that controls the width of a Normal prior distribution over the precision of each element of w_i .

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

To complete the Bayesian specification, a Gamma distribution[2] is used as a prior over the hyper parameters α_i - a standard choice for non-informative priors over Gaussian width parameters. This form of prior structure results in the marginal distribution over $\boldsymbol{w}$ being a product of Student-t distributions, and thus favors sparse solutions that lie along the hyper-spines of the distribution [3]. A large 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 (3). The non-zero elements of $\boldsymbol{w}$ are called Relevance Values, and their corresponding data-points are called Relevance Vectors (RVs).

$$p(\mathbf{w} \mid \boldsymbol{\alpha}) = \prod_{i=0}^{N} N(w_i \mid 0, {\alpha_i}^{-1}) \tag{3}$$

Approximate analytical solution for the RVM can be obtained by the Laplace method [5]: consider a dataset of inputs-target pairs, $\{x_i, t_i\}_{i=1}^{N}$ where targets are assumed, *jointly* Normal distributed - $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, and $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ 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 so $p(\mathbf{t} \mid \mathbf{x}) = N(\mathbf{t} \mid \mathbf{y}(\mathbf{x}), \sigma^2)$. The conditional probability of the targets given the parameters and the data can now be expressed as

[2] The choice of its parameters is of little importance as long as it is sufficiently broad [5]

$$p\left(\mathbf{t} \mid \mathbf{w}, \sigma^2\right) = \left(2\pi\sigma^2\right)^{\frac{N}{2}} \exp\left\{-\frac{1}{2\sigma^2}\left\|\mathbf{t} - \mathbf{\Phi w}\right\|\right\} \tag{4}$$

where the data is hidden the *NxN* kernel function matrix $\mathbf{\Phi}$ representing all the pairs $\phi_{i,j} = k\left(x_i, x_j\right),\ i, j \in [1, \ldots, N]$. ($\mathbf{\Phi}$ could be extended to include a possible bias term). With the prior (3) and the likelihood (4) the posterior distribution over the weights can be expressed with Bayes rule:

$$p(\mathbf{w} \mid \mathbf{t}, \boldsymbol{\alpha}, \sigma^2) = \frac{p(\mathbf{t} \mid \mathbf{w}, \sigma^2)\, p(\mathbf{w} \mid \boldsymbol{\alpha})}{p(\mathbf{t} \mid \boldsymbol{\alpha}, \sigma^2)} \sim \frac{\left|\mathbf{\Sigma}\right|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}\left(\mathbf{w} - \boldsymbol{\mu}\right)^T \mathbf{\Sigma^{-1}} \left(\mathbf{w} - \boldsymbol{\mu}\right)\right]}{\left(2\pi\right)^{\frac{N+1}{2}}} \tag{5}$$

The unknowns $\left(\boldsymbol{\mu}, \mathbf{\Sigma}\right)$ are computed as

$$\mathbf{\Sigma} = \left(\mathbf{\Phi^T B \Phi} + \mathbf{A}\right)^{-1} \tag{6}$$

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

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

$$p(\mathbf{t} \mid \boldsymbol{\alpha}, \sigma^2) = \frac{\left|\mathbf{B}^{-1} + \mathbf{\Phi A^{-1} \Phi}^T\right|^{-\frac{1}{2}}}{\left(2\pi\right)^{\frac{N}{2}}} \exp\left\{-\frac{1}{2}\mathbf{t}^T \left(\mathbf{B}^{-1} + \mathbf{\Phi A^{-1} \Phi}^T\right)^{-1} \mathbf{t}\right\} \tag{7}$$

The solution is derived via the following iterative type II maximization of the marginal likelihood (7) with respect to $\left(\boldsymbol{\alpha}, \sigma^2\right)$

$$\alpha_i^{new} = \frac{1 - \alpha_i \Sigma_{ii}}{\mu_i^2} \tag{8}$$

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

The basic RVM algorithm cycles between (8) and (6), reducing the dimensionality of the problem when any α_i larger than the threshold. The algorithm stops when the likelihood (7) ceases to increase. For further details of the basic RVM look in [5]. For a discussion of convergence and sparseness look in [7].

An important step in GLM learning is to find a feature space - a projection of the data on highly dimensional space - where the data is linear for regression problems and linearly separable for classification problems. The choice of projection (the kernel function) is important for the accuracy and the convergence of the RVM.

It is difficult to accurately quantify the computational complexity of the RVM, since the size of the matrices in (6) may reduce from cycle to cycle. The matrix inversion operation in (6), which requires $O(N^3)$ operations is the computationally intensive part of the

algorithm. The computation of the error term in equation (8) requires $O(N^2)$ operations. The matrices $\mathbf{\Phi}$ and $\mathbf{\Sigma}$ are full rank[3], thus require initially $O(N^2)$ space complexity. Furthermore, it is common that the inversion of a large matrix becomes ill-conditioned after several cycles even for positive definite matrices unless the parameters of the kernel function are optimized. These problems limit the practicality of the basic RVM algorithm for moderately sized problems.

Two possible strategies for complexity reduction were suggested in the literature:

a) Performing approximate, rather than exact matrix inversion at a cost of $O(N^2)$.

b) Iterative sub-sampling of the data, thus reducing N.

The first approach is considered in [4] and will not be considered here. The second approach is suggested by [6] - 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 [3], 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.

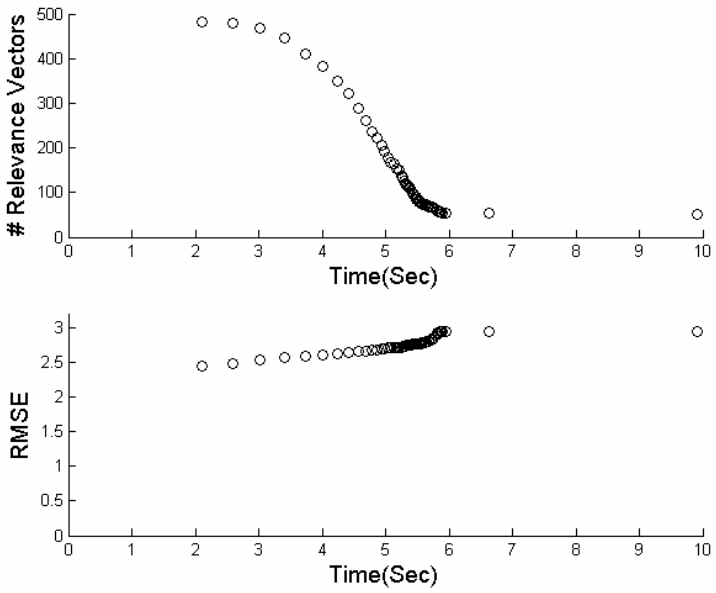

Figure 1. The number of relevance vectors as a function of time and the RMSE as a function of time for the Boston data. Each circle marks the time when the RVM deleted a basis function

Another possible computational inefficiency results from waiting for convergence of the basic RVM before considering the inclusion of a new basis function. For example, look in Figure 1 which presents a typical convergence of the accuracy and the number of

[3] Some kernel functions can produce a sparse representation

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, [3] suggested not waiting for the convergence of the basic algorithm - every time the number of relevance vectors drops below a certain low "water mark" the number of basis functions is increased to a "high water mark".

While those methods demonstrated a significant speedup over the basic RVM with no accuracy loss, there was no attempt to investigate many features of the algorithm, such as the best number of basis functions to have on each step, the selection of basis functions, and the best convergence conditions for the intermediate steps. Furthermore, in each step of the algorithm, the error is computed over the full training set – which may require unacceptable space complexity for very large data. The only solution for large data is to partition the problem into subsets of data or basis functions, which will be processed in sequence.

This paper focuses specifically on these types of investigations. We propose three different sub-sampling algorithms, and compare their performance over benchmark datasets. Also we give a reasonable heuristic for the size of partitions that these sub sampling algorithms should used.

The rest of this paper is as follows: section 2 presents the three algorithms and numerical experiments; section 3 presents a heuristic for selecting the size of the partition *P*; section 4 compares the three heuristics; and section 5 concludes with a discussion.

2. Three Data Partitioning RVM Algorithms

The key idea in the following partitioning heuristics is to benefit from the reduced complexity $O(P^3)$ of computing the basic RVM for a partition, with a possible premature convergence. Each algorithm considers differently how to merge the RVs resulting from a single partition with the rest of the data.

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 (6) and (8) was implemented with the MATLAB scripting language, with the following two modifications:

- The error $\|\mathbf{t} - \mathbf{\Phi}\boldsymbol{\mu}\|$ in (8) was computed over the *full* training set. The reason is that otherwise the RVM may overfit the date of any single partition, eliminating possibly good basis functions. Note that the complexity of the error computation is $O(N^2)$, 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.
- A premature convergence flag was added, so that the algorithm could stop when the number of RVs drops below $\frac{P}{2}$. The reason is that in intermediate processing of partitions, there may not be any benefit in waiting for the full and lengthy convergence (consider Figure 1), when the resulting RVs is to be integrated with the next data partition. Only on the last stage of the algorithm, the basic RVM is expected to run until convergence.

In the following sections we describe each heuristic and its motivation.

2.1 The Split and Merge RVM

The key idea here was taken from the merge sort algorithm. The RVs resulting from processing any two partitions are merged together into a new partition (whose size may be larger than P), and the merge process is repeated recursively. Figure 3 presents a sketch of the following algorithm for N=4000 and P=1000:

1. Run the basic RVM on each one of the initial M partitions and store each set of resulting RVs.
2. Merge each two sets of resulting RVs into $\lceil \frac{M}{2} \rceil$ 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 convergence and obtain the solution.

Complexity analysis: Considering that a sparse solution exists[4], 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. 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.

[4] Otherwise, the kernel function is probably not optimal, and numerical instability might be expected

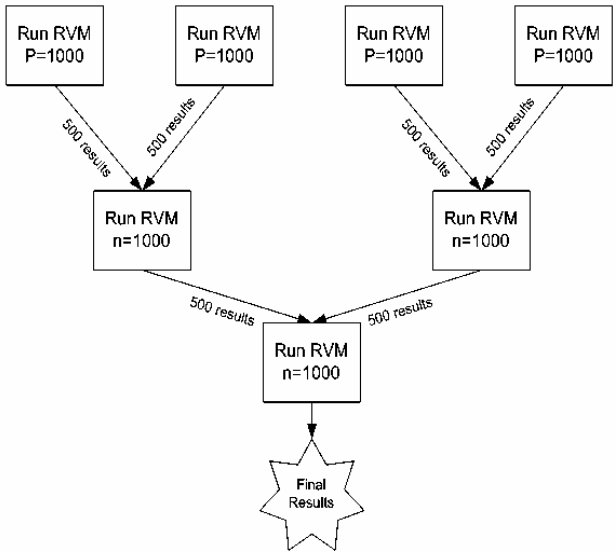

Figure 3. The split and merge heuristics

2.2 The Incremental RVM

The key 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 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. Figure 4 presents a sketch of the following algorithm for *N=4000* and *P=1000*

1. Run the basic RVM on the first partition and store the resulting RVs.
2. Merge the RVs of the previous partition with the data in the subsequent 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 convergence and obtain the final 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 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 approximately $\frac{2N}{P}$ partitions to be processed. The overall complexity is of the order of $8NP^2$.

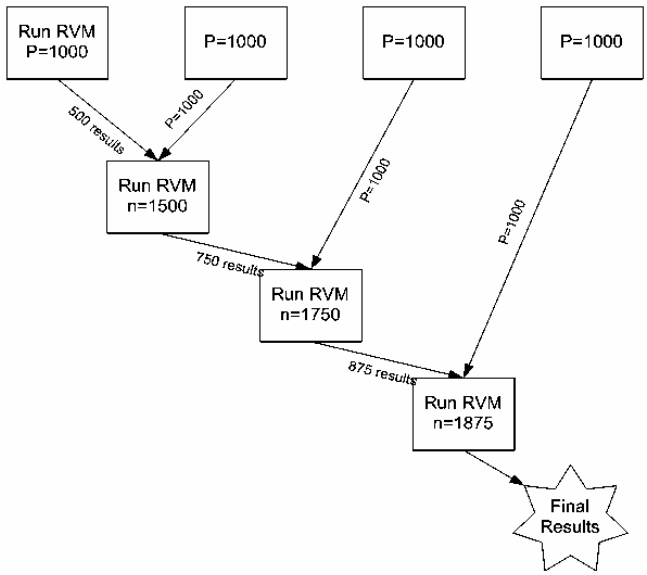

Figure 4. The incremental heuristics

2.3 The Working Set RVM

Here we expand the idea of the incremental RVM: instead of merging the RVs with the data of next partition – which is an arbitrary partition decided in advance – we merge with more informative partition that is generated on the fly by utilizing the current RVM model: the prediction error (which is computed anyway during the basic RVM algorithm) is used to identify the data with the worst prediction error – which is possibly the most informative for improving the current model. The next merge operation is performed with the partition of the most informative basis functions of size P. There is a gradual increase in the complexity 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.

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 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 heuristics does may enhance the convergence.

2.4 Comparative Experiments

For comparative analysis of the above three heuristics, we conducted the following experiments on the benchmark datasets described in Table 1 [5]. A training set was used for the learning phase, and the error was measured on the testing set. Since the algorithms may be influenced by the order of the data, each algorithm was simulated for 100 random repetitions. Before each repetition the data was randomized and then the split between the test set and the training set was done. The largest dataset was simulated only 10 times. The same randomizations were used for comparing the three heuristics versus the basic RVM. The Gaussian kernel was utilized in all the experiments in this paper and its width parameter was optimized once for each dataset via cross validation. In all the experiments in this section, a partition of size $P=50$ was used to simulate the most interesting case - where P is much smaller than N. Tables 2, 3, 4 present respectively the comparative RMSE, the comparative number of RVs, and the comparative run time (on a computer equipped with a Pentium IV 2.42 GHz processor and 768 MB physical memory) for each benchmark dataset and each algorithm. The standard deviation of each measure is also presented in the table.

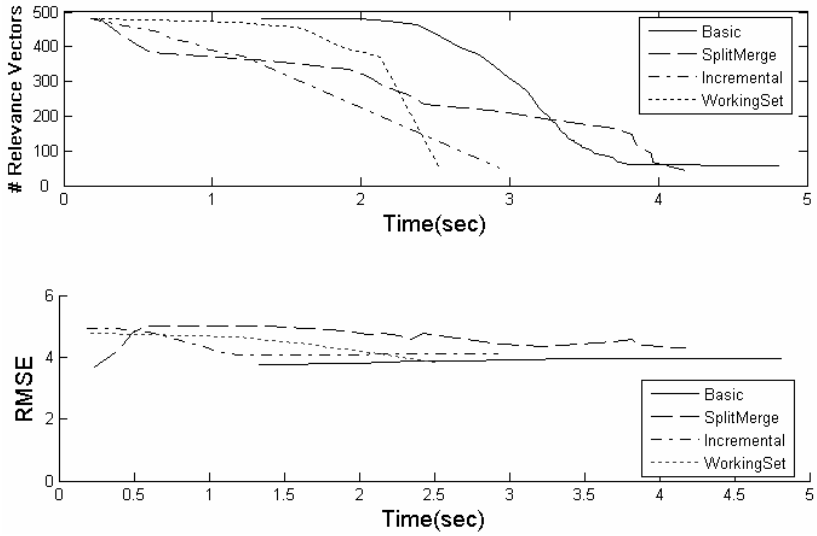

Figure 2. Comparative accuracy (on the training set) and the number of RVs over the run time of the heuristics for the Boston dataset.

Table 1. Details of the benchmark datasets used for the comparative experiments

Name	Size	# Attributes	Training set /test set
Boston	506	13	481/25
Sinc	1000	1	1000/1000
Friedman#1	2400	10	2400/1000
Abalone	4177	8	3341/836

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 2) thus they are correct.
- The number of RVs is also similar (Table 3) and possibly smaller than that of the basic RVM, thus retaining the sparseness of the solution.
- The time to convergence (Table 4) is much shorter in most cases; except the Friedman#1 datasets. This is due to its number of RVs which is much larger than P due to its severe non-linearity. This important issue is discussed in section 3.
- Section 4 is dedicated to comparing the three heuristics.
- One motivation for the working set algorithm is that it is possible to stop it prematurely and still receive a decent solution. Figure 2 demonstrates a typical behavior of this heuristics for the Boston dataset during its run time. As expected, RMSE of the working set heuristics 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 heuristic will indeed provide a decent solution.

Table 2. Comparative RMSE

	BasicRVM	SplitMerge	Incremental	WorkingSet
Boston	3.8 ± 0.9	4.2 ± 1.0	4.0 ± 1.0	3.9 ± 0.9
Sinc	.01 ± .003	.034 ± .002	.01 ± .002	.001 ± .002
Friedman#1	.56 ± .02	.69 ± .06	1.0 ± .01	.60 ± .02
Abalone	2.1 ± 0.1	2.1 ± 0.1	2.1 ± 0.1	2.1 ± 0.1

Table 3. 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. 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

3. Estimating the Best Partition Size P

As exemplified with the Friedman#1 dataset, in Table 4 there is no saving in the run time. Note that in this specific dataset, the number of RVs is about three times 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 $\frac{P}{2}$, 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. This overhead in this growth process may slow the heuristics even more than the basic RVM, as exemplified with the Friedman#1 dataset, in Table 4.

The partition size *P* has a strong impact on the run-time of the three techniques described in section 2. 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 of error calculations in (8). The error computations 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. Following, we suggest a heuristics for guessing a decent P.

3.1 The Partition Size Algorithm

The sparseness of the RVM solution depends on the number of basis functions, the dimensionality of the data, and the complexity of the data set. The main idea of the heuristic 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 should be at least as large as the size of the subset. Following are the details of the heuristics we tested in the next section.

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 dimensionality of the data.
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 6 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, 1) were chosen based on extensive experimentation (which will be detailed elsewhere). 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.

Complexity analysis: The partition P in this heuristic is on the order of $N^{0.5}$. Thus, the overall runtime complexity of the three partition heuristics is $O(N^2)$.

3.2 Experiments with the Partition Size P

In order to validate the partition heuristic defined in the previous subsection, we used four data sets from the *pumadyn*[5] family of datasets. The datasets are all variations of the same system; a realistic simulation of the dynamics of a Puma 560 robot arm. The task in these datasets is to predict the angular acceleration of one of the robot arm's links. This family contains 8 data sets, each one of them has its own characteristics regarding three properties: the level of non-linearity, the noise in the data and its dimensionality. Table 5 describes the datasets which were used for validating the partition size heuristic and for the experiments in section 4. Table 6 describes experiments with the working set algorithm for various sizes of P. The results in Table 6 (and in section 4) were obtained from 20 repetitions (same approach as in section 2.4) on a 3.1 GHz Pentium IV processor equipped with 2 GB of physical memory.

[5] http://www.cs.toronto.edu/~delve/data/datasets.htm

Table 5. Details of the Pumadyn datasets used to validate the heuristic and the experiments in chapter 4.

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

The results in Table 6 indicate 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 best time is obtained when the partition size is somewhat close to double the number of RVs in the final solution.
- The partition size *P* has a small effect on the accuracy, and some effect on the number of the RVs.

Table 6. The impact of the partition size on the results as measured on Pumadyn datasets using the Working Set technique and the comparative results to the heuristic.

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

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 2, we already know that the run time complexity of the split and merge technique is less than the run time 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. That may affect the effective complexity of the algorithms.

Instead of taking a set of unrelated benchmark data sets, we decided to use the *pumadyn* family of datasets (as described in Table 5) in order to see if there is any difference among the techniques that is 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. Training set was used for the learning phase, and the error was measured on the testing set. The heuristics of section 3 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). Tables 7, 8, 9 present respectively the comparative RMSE, the comparative number of RVs, and the comparative run time. The standard deviation of each measure is also presented in the table.

Analysis of the results indicates that

- The three heuristics obtained a similar accuracy and a similar sparseness (tables 7, 8).
- None of the heuristic 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 different behavior than the other two.
- Although further experiments are necessary, it seems that the RVM capable of solving problems with high degree of non-linearity, noise, and high dimensionality.

Table 7. 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 8. 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 9. 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

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 paper we suggested three heuristics for accelerating the basic RVM via splitting the dataset for consecutive applications of the basic RVM and merging the solutions. Simulation experiments on benchmark datasets indicate that the three heuristics indeed accelerate the RVM, while retaining the accuracy and the sparseness of the basic RVM. We also proposed a heuristic for selecting good partition size. Complexity analysis indicate that the partition 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.

Another open research question is the acceleration of the classification RVM. The classification RVM [5], includes in addition to the matrix inversion of (6), 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. This is currently a subject of active research by the authors.

The RVM algorithm is new and some of its essential features – such as proper selection of basis functions – are not yet known. This paper 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.

References

[1] Cristianini N., Shawe-Taylor J., *An Introduction to Support Vector Machines and other kernel-based learning methods*, Cambridge University Press, New York, NY, 1999.

[2] Down T.A., Hubbard T.J.P., Relevance Vector Machines for Classifying Points and Regions in Biological Sequences, Wellcome Trust, Sanger Institute, 2003.

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

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

[5] Tipping M. E., Sparse Bayesian Learning and the Relevance Vector Machine. Journal of Machine Learning Research **1**, 2001, 211-244.

[6] Tipping M. E., Faul A., 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, 2003.

[7] Wipf D., Palmer J., Rao B., Perspectives on Sparse Bayesian Learning. *Advances in Neural Information Processing systems*, **16**. Cambridge, Massachussettes, MIT Press, 2004.