Accelerating the Relevance Vector Machine via Data Partitioning

David Ben-Shimon[1] and Armin Shmilovici[1]

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

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 partition the dataset into manageable chunks. Our experiments 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.

1 Introduction

The Relevance Vector Machine (RVM) [2] 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.

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

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 inferences based on Gaussian Processes (GP) - in its naïve implementation - lack 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 [6], 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 [2,6]. The goal of the RVM is to accurately predict the target function, while retaining as few basis functions as possible in ***w***. The 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 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, thus it will not be discussed here. 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 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 ***w*** are called Relevance Values, and their corresponding data-points are called Relevance Vectors (RV).

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

Approximate analytical solution for the RVM can be obtained by the Laplace method (Tipping, 2001): 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 and $p(t \mid x) = N(t \mid y(x), \sigma^2)$. The conditional probability of the targets given the parameters and the data can now be expressed as

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

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

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

The unknowns (μ,Σ) are computed as

$$\boldsymbol{\Sigma}=(\boldsymbol{\Phi}^{\mathbf{T}}\mathbf{B}\boldsymbol{\Phi}+\mathbf{A})^{-1} \quad (6)$$

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

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

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

The solution is derived via the following iterative type II maximization of the marginal likelihood (7) with respect to (α,σ^2)

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

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

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 [2]. For a discussion of convergence and sparseness look in [5]

The most computational intensive part in the algorithm is the matrix inversion operation in (6), which requires $O(N^3)$ operations. The computation of the error term in equation (8) requires $O(N^2)$ operations. The matrices Φ and Σ are full rank, thus require initially $O(N^2)$ space complexity. It is difficult to quantify the full computational complexity, since the size of the matrices may reduce from cycle to cycle. We would like to provide as little prior knowledge as possible about which basis functions (or kernel) to use. Unfortunately, it is common that the matrix inversion becomes ill-conditioned after several cycles, unless the parameters of the kernel function are optimized. These problems make the basic RVM algorithm unpractical for moderately sized problems.

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 first approach is considered in [1] and will not be considered here. The second approach is suggested by [4] - 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 [7], 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.

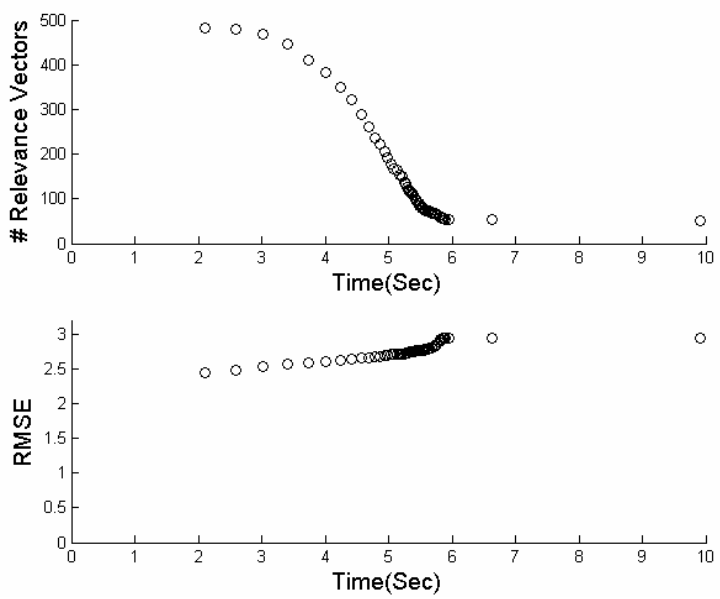

Fig. 1. The number of relevance vectors as a function of time and the RMSE as a function of time for the *Boston* data. Each circle marks the time when the RVM deleted a basis function

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 relevance vectors as a function of time for the Boston benchmark data: initially, the number of RV 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 testing 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. In this paper we focus specifically on these types of investigations. We propose three different sub-sampling algorithms, and compare their performance over benchmark datasets.
The rest of this paper is as follows: section 2 presents the three algorithms; section 3 presents the numerical experiments; and section 4 concludes with a discussion.

2 Three data partitioning RVM Algorithms

For a fair comparison of the algorithms the same training set of size N was partitioned into $M = \lceil \frac{N}{P} \rceil$ 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 basic two modifications:

The error was computed over the full dataset. The reasoning is that otherwise the RVM may overfit a single partition, thus, eliminate 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.

A premature convergence flag was added, so that the algorithm could stop when the number of RV drops below $\frac{P}{2}$. The reasoning is that in intermediate processing of partitions, there may not be any benefit in waiting for the full and lengthy convergence (consider Figure 1), if the resulting RV is immediately integrated with another set of data. Only on the last stage of the algorithm, the basic RVM is allowed to run until convergence.

The key idea in the following partition 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 RV resulting from a single partition with the rest of the data. In the following we describe each heuristics and its reasoning.

2.1 The Split and Merge RVM

The key idea here was taken from the merge sort algorithm: The resulting RV from each two partitions are merged together into a new partition (but this time its size may be larger than P), and the process is recursively repeated:

1. Run the basic RVM on each of the original M partitions and store each set of resulting RV.
2. Merge each two sets of resulting RV 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 RV. 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: The number of merge operations is on the order of $\log_2\left(\frac{N}{P}\right)$. Because of the premature stopping of the basic RVM on each dataset for a sufficiently large P - as long as sparseness can be obtained in that specific problem - the complexity of processing each dataset should be $O(P^3)$, except possibly the last step. In a multi-processor system, each partition can be processed on a different processor.

2.2 The Incremental RVM

The idea here is to merge the RV 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.

1. Run the basic RVM on the first partition and store the resulting RV.
2. Merge the resulting RV with the next partition. Run the basic RVM on the data and store the resulting RV.
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.

The number of merge operations is on the order of $\left\lceil \frac{N}{P} \right\rceil$. Because of the premature stopping of the basic RVM on each dataset when the number of RV drops to half, the size of the dataset gradually increases: $p, \frac{p+2p}{2}, \frac{p+2p+4p}{4}, \frac{p+2p+4p+8p}{8} \ldots \to 2p$

The complexity of processing the last dataset is $O\left((2P)^3\right)$.

2.3 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 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.

1. Run the basic RVM on the first partition and store the resulting RV.
2. Construct a model from the last RV, 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 RV with the newly generated partition. Run the basic RVM on the data and store the resulting RV.
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.

The complexity analysis of this algorithm is the same as that of the incremental algorithm, however, the final RV are identified earlier than in incremental process.

3. Comparative Experiments

For the analysis of the heuristics, we conducted the following experiments on the benchmark datasets described in Table 1 [2]. 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 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). The partition used was $P=50$ 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 RV, 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.

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

Name	Size	# Attributes	Training set /test set
Cpu	209	7	189/20
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 RV 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 Friedman1 datasets. This is due to its number of RV which is much larger than P due to its non-linearity. This important issue is discussed in section 3.1

Regarding which heuristic is better, it is hard to decide with this small set of experiments, since no heuristic demonstrates dominance over all datasets and for the three quality measures.

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 the 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” RV also drops very fast. Similar behavior was detected for the larger

Abalon dataset. Thus, for very large datasets (that run too slow), premature stopping of the heuristic will indeed provide a decent solution.

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

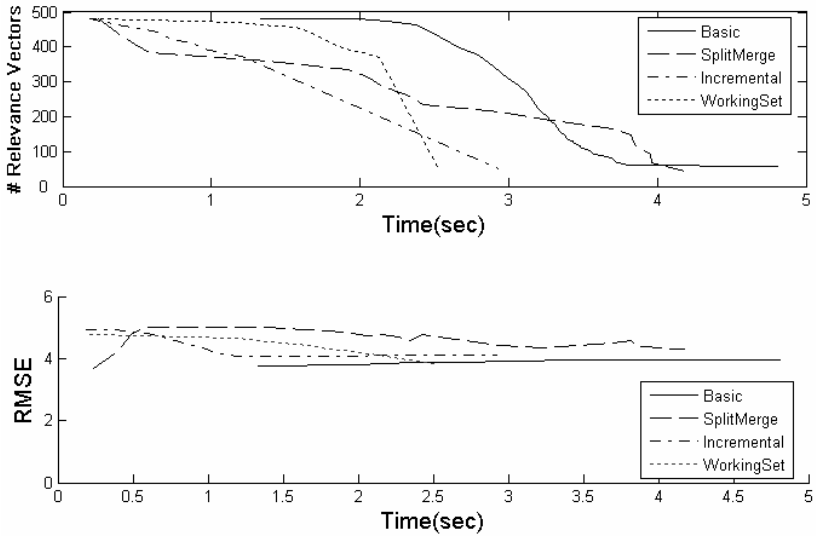

Table 2. Comparative RMSE

	BasicRVM	SplitMerge	Incremental	WorkingSet
Cpu	83.1 ± 86.7	86.2 ± 69.0	88.9 ± 88.8	84.4 ± 83.3
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 ± 0.02	.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 RV

	BasicRVM	SplitMerge	Incremental	WorkingSet
Cpu	126.2 ± 5.4	28.1 ± 2.9	49.3 ± 7.5	73.5 ± 11.4
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
Cpu	96.5 ± 47.5	1.4 ± 0.6	2.6 ± 1.8	10.4 ± 5.7
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

3.1 Estimating the Partition Size P

Table 6 demonstrates the effect of the partition size P on the run time, sparseness, and accuracy for the Working Set algorithm on the Abalone dataset. The partition size P has a strong impact on the run-time of the RVM. A too large partition is inefficient in terms of computational complexity and space complexity. However, a too small partition by the error computation in formula which is (much smaller than the number of RV) has its own limits: the error computation requires $O(N^2)$ operations, there is an overhead in manipulating many partitions, and the convergence of the basic RVM for a small partition may be slow. 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.

Table 5. The impact of the partition size on the convergence time as measured on the Abalone dataset using the Working Set algorithm.

	P=12	P=25	P=50	P=100	P=200	P=400	P=800	P=1000
Time	113.1	38.9	19.1	20.4	26.9	39.7	71.9	87.6
RMSE	2.08	2.07	2.07	2.08	2.08	2.08	2.07	2.08
#RV	17	19	22	21	22	23	24	26

Table 6. Comparative measures, basic RVM and Working Set using the partition size heuristic

	RMSE		#RV		Time(sec)	
	Basic RVM	Working Set	Basic RVM	Working Set	Basic RVM	Working Set
Cpu	83.1±87	94.4±103	126.2±5.4	64.8±7.8	96.5±47.5	7.6±5.4
Boston	3.8±0.9	3.8±1.0	53.6±3.2	49.3±1.8	4.1±1.8	1.7±0.6
Sinc	.01±.003	.011±.002	7.8±1.6	8.5±1.9	11.12±1.1	3.6±0.9
Fried#1	.56±.02	.55±.01	172.8±7.7	160±7.2	193.7±32	59±15
Abalone	2.1±0.1	2.1±0.1	24.6±2.0	23.5±1.9	412.7±19	28.9±2.0

From other experiments we performed, it seems that the best results are obtained when P is about twice the number of RV. The number of RV 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. Obviously the number of RV can not be determined in advance. From empirical and computational considerations (which will

not be explained here), we found out that setting the partition size around $P = 4 * \sqrt{N}$, results in significant gain in the convergence time without reducing neither the accuracy nor the sparseness of the final solution. Table 6 compares the results of using this partition selection heuristic (for the Working Set RVM) to the basic RVM. It is a reasonable compromise compared to the experiments in tables 2,3,4.

4. 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 splitting the dataset for consecutive applications of the basic RVM and for merging the solutions. Preliminary 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. One of the heuristics – Working Set RVM – is possibly superior to the other two heuristics. However, further extensive experimentation on larger datasets is needed, as well as the optimization of various parameters, such as the partition size *P*.
The RVM algorithm is new and some essential feature – 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. D'Souza A., Vijayakumar S., Schaal S.: The Bayesian Backfitting Relevance Vector Machine, Proc. Of the 21st International conference on Machine Learning, July 4-8, Baniff, Canada, (2004)
2. Tipping M. E.: Sparse Bayesian Learning and the Relevance Vector Machine. Journal of Machine Learning Research 1, (2001) 211-244.
3. Down T.A., Hubbard T.J.P.: Relevance Vector Machines for Classifying Points and Regions in Biological Sequences, Wellcome Trust, Sanger Institute (2003)
4. 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)
5. Wipf D., Palmer J., Rao B.: Perspectives on Sparse Bayesian Learning. Advances in Neural Information Processing systems 16. Cambridge, Massachussettes, MIT Press. (2004)
6. Cristianini N., Shawe-Taylor J.: An Introduction to Support Vector Machines and other kernel-based learning methods, Cambridge University Press, New York, NY, (1999)
7. 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)