The Relevance Vector Machine

DD2434 Group Project, group 5

Alex Hermansson
Alice Karnsund
Elin Samuelsson
Therese Stålhandske

January 2018

1. Abstract

The Relevance Vector Machine (RVM) is a method for classification and regression, similar to the Support Vector Machine (SVM). Though, the RVM has some major advantages over the SVM, since it approaches the problem with probabilities, and thus it does not suffer from the same disadvantages as the SVM. In this paper we introduce the techniques of the RVM, and compare the RVM with the SVM in the framework of "The Relevance Vector Machine" by Michael E. Tipping, 2000. For this comparison, a couple of different well known data sets are used.

1 Introduction

The Relevance Vector Machine (RVM) can be used for both classification and regression. It can be seen as the Bayesian version of the Support Vector Machine (SVM), which makes predictions according to

$$y(\mathbf{x}) = \sum_{n=1}^{N} w_n K(\mathbf{x}, \mathbf{x}_n) + w_0. \tag{1}$$

In the case of classification, the SVM aims to minimize the number of errors made on the training set, and simultaneously maximize the margin between the two classes. The boundary can be modeled using only a subset of training data, i.e. only a few kernel functions, resulting in a sparse model. However, the SVM method has some major flaws. First, it returns a decision rather than a probability, and thus disregards the uncertainty in the decision process. Second, to extend SVM to multi-class problems causes various problems, described in [1]. Further, the number of kernel functions needed by the SVM grows steeply with the size of the training set, drastically increasing the computational cost. There are also parameters regarding complexity, C, and insensitivity, ϵ, which are usually estimated with a costly cross-validation procedure. Finally, the SVM requires its kernel functions to be positive definite.

The majority of these problems can be prevented using a Bayesian approach, i.e. the RVM method. Furthermore, the RVM typically gives even sparser models than the SVM, which decreases the computational cost, while keeping a comparable generalization error. The RVM is therefore of great interest, and in the following sections both regression and classification will be discussed. The goal is to investigate the performance of the RVM compared to the SVM, in terms of compuational complexity, sparsity and prediciton error. An implementation of the RVM for multi-classification is also discussed, but not implemented.

1.1 RVM for regression

Under the Bayesian framework, our model defines a Gaussian conditional distribution for a real-valued target variable t, given an input vector $\mathbf{x}$ [3], $p(t|\mathbf{x}) \sim \mathcal{N}(t|y(\mathbf{x}), \sigma^2)$, where $y(\mathbf{x})$ is defined by equation (1). The likelihood over an i.i.d. dataset $(\mathbf{X}, \mathbf{t}) = (\{\mathbf{x}_n\}_{n=1}^N, \{t_n\}_{n=1}^N)$ then becomes

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \sigma^2) = (2\pi\sigma^2)^{-N/2} \exp\{-\frac{1}{2\sigma^2}|\mathbf{t} - \Phi\mathbf{w}|^2\}, \tag{2}$$

with a weight vector $\mathbf{w} = [w_0, ..., w_N]$ and a design matrix Φ. The rows of Φ define basis functions $\boldsymbol{\phi}(\mathbf{x}_n) = [1, K(\mathbf{x}_n, \mathbf{x}_1), ..., K(\mathbf{x}_n, \mathbf{x}_N)]$, which renders Φ to have size $N \times (N+1)$. $K(\mathbf{x}, \mathbf{x}')$ is a kernel function, and the first element in each row is set to one because of the bias term w_0 in equation (1).

Next, we introduce a prior over the weight vector with *a separate hyperparameter α_i for each w_i.*

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

[1] Bishop, Pattern Recognition And Machine Learning, Springer, 2006, Page 345

This is the key feature of the RVM that causes the desired sparsity of the model (in contrast to Bayesian linear regression, which uses the same hyperparameter α for all weights). Following the steps in Bishop chap. 3.3, we find that equation (2) and (3) generates the following posterior distribution:

$$p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \boldsymbol{\alpha}, \sigma^2) = \mathcal{N}(\mathbf{w}|\mu, \Sigma) \qquad \text{with } \Sigma = (\Phi^T \mathbf{B}\Phi + \mathbf{A})^{-1} \tag{4}$$

$$\mu = \Sigma \Phi^T \mathbf{B}\mathbf{t} \tag{5}$$

where $\mathbf{A} = diag(\alpha_0, ..., \alpha_N)$ and $\mathbf{B} = \sigma^{-2}\mathbf{I}_N$.

Therefore, when training, the goal is to find optimal values for the parameters, which we denote $\boldsymbol{\alpha}_{MP}$ and σ^2_{MP}. The algorithm (see Method section) drives many of the $\{\alpha_i\}$'s towards infinity, which is equivalent to the corresponding weights $\{w_i\}$ having posterior distributions with both mean and variance approaching zero. Such zero-valued elements in $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ completely remove the contribution from those inputs $\{\mathbf{x}_i\}$. Thus when predicting a distribution for t_*, given a new input $\mathbf{x}_*$,the only inputs that contribute are those whose posterior weight distribution maintain nonzero mean and/or variance. Those inputs are referred to as *relevance vectors*: $\mathbf{X}_{RV} = \{\mathbf{x}_n\}_{RV}$. All other inputs can successively be removed and only the relevance vectors are used for predictions:

$$p(t_*|\mathbf{x}_*, \mathbf{X}, \mathbf{t}, \boldsymbol{\alpha}_{MP}, \sigma^2_{MP}) = \int p(t_*|\mathbf{x}_*, \mathbf{w}, \sigma^2_{MP}) p(\mathbf{w}|\mathbf{t}_{RV}, \mathbf{X}_{RV}, \boldsymbol{\alpha}_{MP}, \sigma^2_{MP}) d\mathbf{w} \tag{6}$$

$$= \mathcal{N}(t_*|\boldsymbol{\mu}^T_{RV}\boldsymbol{\phi}_{RV}(\mathbf{x}_*), \sigma^2_{MP} + \boldsymbol{\phi}_{RV}(\mathbf{x}_*)^T \Sigma \boldsymbol{\phi}_{RV}(\mathbf{x}_*)) \tag{7}$$

1.2 RVM for classification

To extend the RVM framework for classification, we need to find the posterior probability $p(C_k|\mathbf{x})$ of class membership for an input $\mathbf{x}$. For the binary classification problem, we generalize the linear model, following statistical convention, by applying the sigmoid function

$$\sigma(y(\mathbf{x})) = \frac{1}{1 + e^{-y(\mathbf{x})}} \tag{8}$$

to $y(\mathbf{x}) = \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x})$, defined by equation (1). For the binary case, we adopt a Bernoulli distribution for $p(t|\mathbf{x})$ and the likelihood is therefore,

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} \sigma\{y(\mathbf{x}_n; \mathbf{w})\}^{t_n} [1 - \sigma\{y(\mathbf{x}_n; \mathbf{w})\}]^{1-t_n}, \tag{9}$$

where $t_n \in \{0, 1\}$. For a problem with multiple classes, the RVM is extended by introducing the multinomial likelihood. In this case the posterior probabilities of each class are given by applying the softmax function to the transformation of linear functions of the feature variables, given by

$$p(\mathcal{C}_k|\mathbf{x}) = \text{softmax}(y_k(\mathbf{x})) = \frac{\exp(\mathbf{w}_k^T\boldsymbol{\phi}(\mathbf{x}))}{\sum_j \exp(\mathbf{w}_j^T\boldsymbol{\phi}(\mathbf{x}))} \tag{10}$$

In the same manner as for regression, we make predictions using only samples from the training data that have nonzero weights, i.e. the so called relevance vectors.

2 Method

2.1 Regression

To find the optimal values σ^2_{MP} and $\boldsymbol{\alpha}_{MP}$, an EM approach as well as a type-2 maximum likelihood approach can be used. We chose the latter, since it has been shown to exhibit faster convergence [2]. The logarithm of the marginal likelihood function,

$$\ln p(\mathbf{t}|\mathbf{X}, \alpha, \sigma^2) = -\frac{1}{2}\{N \ln(2\pi) + \ln|C| + \mathbf{t}^T\mathbf{C}^{-1}\mathbf{t}\}, \tag{11}$$

with $\mathbf{C} = \mathbf{B}^{-1} + \Phi\mathbf{A}^{-1}\Phi^T$, is maximized by differentiating and equating to zero. The following re-estimation equations are generated,

$$\alpha_i^{new} = \frac{\gamma_i}{\mu_i^2} \qquad (\sigma^2)^{new} = \frac{|\mathbf{t} - \Phi\boldsymbol{\mu}|^2}{N - \sum_i \gamma_i} \qquad \gamma_i = 1 - \alpha_i\Sigma_{ii} \tag{12}$$

As initial values, we set all α_i to 1 and σ^2 to 0.01^2. Then, we iteratively calculated $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$, and re-estimated values for α_i and σ^2. The convergence criteria we used was that the number of relevance vectors did not change in 200 iterations. Further, we set the maximum number of iterations to 1000.

The training was sped up by pruning the parameters. As discussed in section 1.1, only the relevance vectors contribute to the prediction. An infinite alpha value α_i is equivalent to a γ_i value falling below machine precision, which in our case was, 2.22×10^{-16}. If this criteria is fulfilled, the corresponding elements in $\boldsymbol{\gamma}$, $\boldsymbol{\alpha}$, $\boldsymbol{\mu}$, $\mathbf{T}$, $\mathbf{X}$ and columns in Φ were deleted, shrinking the dimensions, and decreasing the computational cost.

The RVM regression (RVR) was explicitly programmed in Python. To evaluate its performance, we compared it to SVM regression (SVR) (described in [3]), which was implemented with the `sklearn` package in Python. The two methods were applied to multiple datasets, following the specifications in [2] and [4].

In many cases, optimization of parameters were needed. We used 5-fold cross-validation to perform a grid search, i.e. we assigned a set of possible values for each parameter and picked those who gave the smallest prediction error when cross-validating. In general, we did this in two steps. For a parameter value $P = c_1 10^{c_2}$, step one set $c_1 = 1$ and optimized c_2, and step two optimized c_1 given the best c_2-value.

2.2 Binary Classification

In contrast to the regression case, we cannot integrate out the weights analytically to get the marginal likelihood. Therefore, we adopt an approximate method based on Laplace's method, as described in [2]. It consists of two steps.

1. For the current value of $\boldsymbol{\alpha}$, the location of the mode, $\mathbf{w}_{MP}$, for the posterior distribution is found. Noting that the posterior $p(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha})$ is proportional to $p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\boldsymbol{\alpha})$, we can find $\mathbf{w}_{MP}$ by maximizing $p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\boldsymbol{\alpha})$ with respect to $\mathbf{w}$. Equivalently we can minimize

[2] Tipping, The Relevance Vector Machine, Microsoft Research, Cambridge U.K, 1999

[3] Bishop, Pattern Recognition And Machine Learning, Springer, 2006, Chapter 7.1.4, Page 339-343

[4] Tipping, Sparse Bayesian Learning and the Relevance Vector Machine, Journal of Machine Learning Research 1 (2001) pp.211-244

$$E = -\log[p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\boldsymbol{\alpha})] = -\sum_{n=1}^{N}[t_n \log y_n + (1-t_n)\log(1-y_n)] + \frac{1}{2}\mathbf{w}^T\mathbf{A}\mathbf{w}, \tag{13}$$

where $\mathbf{A} = \text{diag}(\alpha_i)$. To find the minimum of E, we use a second-order Newton method called iteratively-reweighted least-squares (IRLS) described in section 2.2.1.

2. The hessian of E, which is computed in the IRLS algorithm in step 1., is inverted to give the covariance matrix $\boldsymbol{\Sigma}$ of a Gaussian approximation of the weight posterior. After that, the value of $\boldsymbol{\alpha}$ is updated, using the statistics $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu} = \mathbf{w}_{MP}$, with the same rule as in the regression case, given by equation (14) and (16).

The two steps are iterated until convergence of $\mathbf{w}_{MP}$ and $\boldsymbol{\alpha}$. Also, during these iterations some values α_i approach infinity, which implies that the corresponding weight w_i is infinitely peaked at zero. Thus, the corresponding kernel will not be utilized for prediction and can be pruned away, leading to a sparse model. In the implementation of the model, the pruning is done when values of α_i exceeds a threshold of 10^{12}, chosen according to [2].

2.2.1 The IRLS Algorithm

The iterative-reweighted least-squares algorithm is a method to solve certain optimization problems, such as the one defined by minimizing equation (13). The algorithm uses an update rule for $\mathbf{w}$ according to,

$$\mathbf{w}^{new} = \mathbf{w}^{old} - \mathbf{H}^{-1}(\mathbf{w}^{old})\mathbf{g}(\mathbf{w}^{old}), \quad \text{with} \tag{14}$$

$$\mathbf{g}(\mathbf{w}) = \nabla E = \boldsymbol{\Phi}^T(\mathbf{y} - \mathbf{t}) + \mathbf{A}\mathbf{w} \tag{15}$$

$$\mathbf{H}(\mathbf{w}) = \nabla^2 E = \boldsymbol{\Phi}^T\mathbf{B}\boldsymbol{\Phi} + \mathbf{A}, \tag{16}$$

where $\mathbf{B}$ is a diagonal matrix with $B_{nn} = y_n(1-y_n)$ and $\mathbf{A} = \text{diag}(\alpha_i)$.

2.3 Multiclass Classification

The number of input points, active kernels and classes are denoted N,M and K respectively. The target vectors are represented by using a 1-of-K coding scheme, thus $\mathbf{t}_i$ is a bit vector, in which the k'th bit turns on iff $\mathbf{t}_i = k$. This gives a target vector of shape $N \times K$ with elements t_{nk}. The multinomial log-likelihood function is then written as [5],

$$\log p(\mathbf{T}|\mathbf{w_1}, ..., \mathbf{w_K}) = \log \prod_{n=1}^{N}\prod_{k=1}^{K} y_{nk}^{t_{nk}} = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \log y_{nk} \tag{17}$$

Here $\mathbf{w}_k$ denotes the class-specific weights for class k, with shape $M \times 1$. In a similar way, we denote the class-specific α's as $\mathbf{A}_k$, more specific $\mathbf{A}_k = \text{diag}(\alpha_{j=1,...N})$ for $k = 1, .., K$. The error function is then be given by,

$$E(\mathbf{w}_1, ..., \mathbf{w}_K) = -\log(p(\mathbf{T}|w_1, ..., w_K)p(\mathbf{w}|\alpha) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \log y_{nk} + \frac{1}{2}\sum_{k=1}^{K}\mathbf{w}_k\mathbf{A}_k\mathbf{w}_k \tag{18}$$

[5] Bishop, Pattern Recognition And Machine Learning, Springer, 2006, Page 356

The gradient with respect to w_j of the error function is given by,

$$\mathbf{g} = \nabla_{\mathbf{w}_j} E(\mathbf{w}_1, ..., \mathbf{w}_K) = \sum (y_{nj} - t_{nk})\boldsymbol{\phi}_n + \sum_{k=1}^{K} \mathbf{A_k}\mathbf{w}_k, \quad j \in 1, \ldots, K \tag{19}$$

The hessian now becomes a matrix of size $MK \times MK$ which comprises of blocks of size $M \times M$ where the block j, k is given by,

$$\mathbf{H}_{jk} = \nabla_{\mathbf{w}_k} \nabla_{\mathbf{w}_j} E(\mathbf{w}_1, ..., \mathbf{w}_K) = -\sum_{n}^{N} y_{nj}(I_{kj} - y_{nk})\boldsymbol{\phi}_n \boldsymbol{\phi}_n^T + \delta_{kj}\mathbf{A}_k, \quad j \in 1, \ldots, K \tag{20}$$

where I_{kj} is the identity matrix of size N. The IRLS process is identical to the one described in the two-class problem, along with the MAP estimation of the alpha-values. The pruning is performed over the class-specific weights $\mathbf{w}_k$ and class-specific relevance vectors are derived under the same conditions as for the binary case. This implies an unequal number of relevance vectors for each class.

3 Results

3.1 Regression

First we studied the SVM and the RVM of the $\text{sinc}(x) = |x|^{-1} \sin |x|$ function. Input data for both training and testing was 100 equidistant points $x \in [-10, 10]$. We used training output data both with and without Gaussian noise $\mathcal{N}(0, 0.2)$. A linear spline kernel was used [4]. Since SVR needed a positive definite kernel, we modified it by replacing all x with $|x|$.

For noise-free training data, SVM used $\epsilon = 0.01$, and C was optimized with 5-fold cross-validation (as described in the Method section). Similarly, for RVM σ^2 was fixed to 0.01^2, and $\boldsymbol{\alpha}$ alone was re-estimated. For noisy training data, SVR optimized both ϵ and C, and RVM re-estimated both σ^2 and $\boldsymbol{\alpha}$. Numerical results can be found in Table 1, and the RVM plots are found in Figure 1.

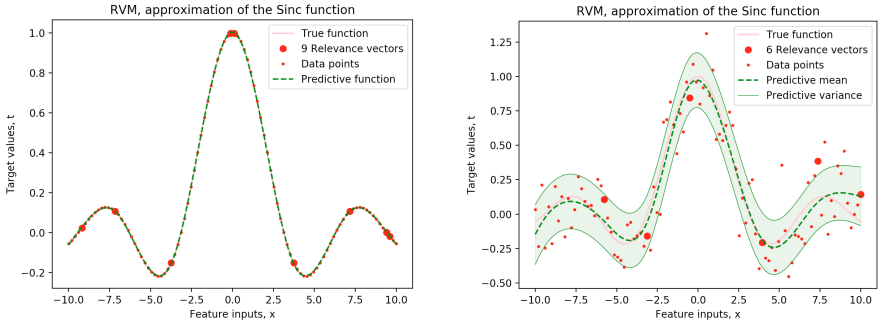

Figure 1: Results of RVM on the $\text{sinc}(x)$ function. Left: noise-free data. Right: noisy data.

Data set	*max error*		*RMS error*		*vectors*	
	SVM	**RVM**	**SVM**	**RVM**	**SVM**	**RVM**
Sinc (no noise)	0.01038	0.00705	0.00719	0.00290	18	9
Sinc (Gaussian 0.2)	0.17913	0.18351	0.08372	0.05469	23	6

Table 1: Results of SVM and RVM on the sinc(x) function. Max error = the single largest error between a predicted value t_* and its actual function value sinc(x_*)

Next, we applied RVM and SVM on 6 datasets. Specifications for training - and test sets can be found in [4]. We made use of the Gaussian kernel $K(\mathbf{x}_m, \mathbf{x}_n) = \exp(-\gamma|\mathbf{x}_m - \mathbf{x}_n|^2)$. In RVM, γ was optimized, and in SVM, a combined optimization of C, ϵ and γ was made.

The data set Friedman #1 is special in the sense that the output only is a function of 5 of the 10 input features. Because of this, an alternative kernel was used, $K(\mathbf{x}_m, \mathbf{x}_n) = \exp(-\eta_1(x_{m,1} - x_{n,1})^2 - \eta_2(x_{m,2} - x_{n,2})^2 ... - \eta_{10}(x_{m,10} - x_{n,10})^2)$. Optimization was made for two possible values of $\eta_i \in \{10^{-2}, 10^{-10}\}$, resulting in $\boldsymbol{\eta} = [10^{-2}, 10^{-2}, 10^{-2}, 10^{-2}, 10^{-2}, 10^{-10}, 10^{-10}, 10^{-10}, 10^{-10}, 10^{-10}]$, i.e. making the last five inputs irrelevant.

Data set	*RMS error*			*vectors*		
	SVM	**RVM**	**η-RVM**	**SVM**	**RVM**	**η-RVM**
Sinc (Gaussian 0.1)	0.0399	0.0236		75.05	8.3	
Sinc (Uniform 0.1)	0.0208	0.0156		39.7	6.9	
Friedman #1	1.233	1.675	1.3499	213.6	49.05	16.55
Friedman #2	305.01	307.46		211.1	7.75	
Friedman #3	0.248	0.313		110.2	7.8	
Boston Housing	4.184	3.283		334.95	19.8	

Table 2: Average performance over 20 iterations.

3.2 Classification

The average classification results over 50 iterations of RVM and SVM are shown in Table 3. For all the dataset we made use of the Gaussian kernel, given in equation (25). The training and test sets were split into sets with sizes 70% and 30% of the total amount of data respectively.

Data set			*Accuracy* (%)		*Kernels*	
	N	**d**	**SVM**	**RVM**	**SVM**	**RVM**
Pima Indians Diabetes	767	8	75.6	75.7	300.1 $\pm$ 17.7	7.6 $\pm$ 1.4
Breast Cancer	569	30	94.4	93.2	71.9 $\pm$ 29.7	7.9 $\pm$ 1.7

Table 3: Average performance for SVM and RVM over two binary datasets. N gives the total number of data points in the set and d is the number of features for each dataset.

An implementation of the extension to multi-class classification was started. However, due to time constraints, we never managed to finish it completely for thorough testing.

4 Discussion

4.1 Regression

Our results strongly support the claims made by Tipping in 2 and 4. We calculated the RMS-error over the test data set, and then took the average over 20 iterations. These errors do not agree with Tipping's, but it is not stated explicitly how those values were calculated. Our values seem reasonable. For instance, the sinc-curve has a maximum value of 1 and we added noise of width 0.1 or 0.2, i.e. Tipping's error of ~ 0.3 is larger than the noise itself. From studying the plots, our values of ~ 0.03 seems reasonable for an RMS error.

Table 1 shows the improvement in RMS error, when using RVM instead of SVM. Note that the SVM max error $\approx \epsilon = 0.01$ for noise-free data. This is supported by theory, since ϵ is a measure of the margin.

In table 2, the error improvement is not really validated. However, there are no large deteriorations either, while the large differences in number of support-/relevance vectors are clearly shown. In other words, the RVM makes predictions that are better or approximatly equally good as SVM, needing fewer vectors, and thereby fewer kernel evaluations. The specific errors seem to strongly depend on the parameter choice. Our optimization method was discrete and the minimums we found were most likely just local ones.

η-RVM highlights the importance of both parameters and kernels, and did improve both error and number of support vectors. However, to optimize that many parameters requires many computations. In our small example, even though we restricted ourselves to two possible η_i-values, we still ended up with $2^{10} = 1024$ $\boldsymbol{\eta}$-vectors to investigate.

4.2 Classification

The results of the accuracy for the RVM classifier are comparable to those produced with an SVM classifier. We see in section 3.2 that RVM even exceeds SVM in classification accuracy on the Pima dataset. Furthermore, like in the regression case, the big difference of the two methods is in the number of kernels used to make predictions. The number of kernels is a whole order of magnitude smaller for RVM compared to SVM, which greatly reduces the computational cost of making predictions. The SVM does not extend naturally to the multi-class classification problem. The most common approach is to use a one-versus-all strategy, which increases the number of kernels by a factor of K (the number of classes). For RVM in the multi-class problem, the full hessian matrix is of size $MK \times MK$ which gives an additional computational cost of K^3, when inverting it, compared to the binary case. This increases the training time, but with the benefit of avoiding cross-validation to optimize hyperparameters. Also the complexity for making predictions is greatly reduced compared to SVM, just as in the binary classification problem.