KTH Royal Institute of Technology

DD2434 SAP HT18-1

Machine Learning, Advanced Course

Group Project

January 20, 2019

Relevance Vector Machine
Sparse modeling

ABSTRACT

The aim of this paper is to present a Bayesian method, the Relevance Vector Machine (RVM), and to assess its performance on regression and classification tasks. It is a model of identical functional form to the well-known Support Vector Machine (SVM) but offers some advantages as, for instance, the sparsity of the solutions and its probabilistic interpretation. Indeed, using a Bayesian framework allows to encode some prior belief and to estimate the prediction uncertainty through the variance of the posterior distribution. We first describe the underlying theory of the RVM and, then, compare its efficiency with the SVM on toy and benchmark data sets. This gives an experimental proof that RVM allows to obtain accurate prediction models which use substantially less basis functions than a comparable SVM while offering the previously considered advantages of the Bayesian framework.

Authors:

Pierre Marza (marza@kth.se)
Pedro Santos (pedrops@kth.se)
Alexander Gaul (agaul@kth.se)
Sten Nygren (stenny@kth.se)

1 Introduction

In this paper, we want to validate the original results about the Relevance Vector Machine (RVM) in Tipping, 2001 [2] as well as apply the method to selected regression and classification problems. Furthermore, we'll do a comparison between the RVM and SVM, exploiting the advantages and disadvantages of both methods. On this section we provide a concise description of the RVM for both regression and classification tasks.

In supervised learning we want to learn a model of the dependency between the input vectors $\{\mathbf{x}_n\}_{n=1}^N$ and the target values $\{t_n\}_{n=1}^N$, the latter which can be real (possibly multidimensional) values in regression or class labels in classification. We follow a fully probabilistic approach and base our predictions on functions like (1), also implemented by SVMs,

$$y(\mathbf{x};\mathbf{w}) = \sum_{i=1}^{N} w_i K(\mathbf{x},\mathbf{x}_i) + w_0 \tag{1}$$

where $K(\mathbf{x},\mathbf{x}_i)$ is a kernel function defining one basis function for each training point and $\mathbf{w}$ are the parameters we wish to infer. The key feature of the RVM is that the inferred predictors are exceedingly sparse in that they contain relatively few non-zero w_i parameters. This is achieved by placing a prior over the weights $\mathbf{w}$ governed by a set of hyperparameters, one associated with each weight w_i, whose most probable values are iteratively estimated from the data. We find that the posterior distribution is sharply peaked around zero for most of the weights, thus allowing us to remove them from the model, while keeping only the non-zero ones which we call relevance vectors.

1.1 Regression

Following the standard probabilistic formulation, we assume that the targets are defined by:

$$t_n = y(\mathbf{x}_n;\mathbf{w}) + \epsilon_n \tag{2}$$

where ϵ_n are independent samples from a mean-zero Gaussian with variance σ^2, thus $p(t_n|\mathbf{x}) = \mathcal{N}(t_n|y(\mathbf{x}_n),\sigma^2)$. Due to the assumption of independence of t_n, the likelihood of the complete data set is the following:

$$p(\mathbf{t}|\mathbf{w},\sigma^2) = (2\pi\sigma^2)^{-N/2}\exp\left\{-\frac{1}{2\sigma^2}||\mathbf{t}-\mathbf{\Phi}\mathbf{w}||^2\right\} \tag{3}$$

where $\mathbf{t} = (t_1...t_N)^{\mathrm{T}}$, $\mathbf{w} = (w_0...w_N)^{\mathrm{T}}$ and $\mathbf{\Phi}$ is the 'design matrix' with $\mathbf{\Phi} = [\phi(\mathbf{x}_1)...\phi(\mathbf{x}_N)]^{\mathrm{T}}$ and $\phi(\mathbf{x}_n) = [1, K(\mathbf{x}_n,\mathbf{x}_1)...K(\mathbf{x}_n,\mathbf{x}_N)]^{\mathrm{T}}$. Then we encode our prior over the parameters $\mathbf{w}$ as:

$$p(\mathbf{w}|\boldsymbol{\alpha}) = \prod_{i=0}^{N}\mathcal{N}(w_i|0,\alpha_i^{-1}) \tag{4}$$

with $\boldsymbol{\alpha}$ being a vector containing one hyperparameter associated with each weight w_i. The assignment of an individual hyperparameter per weight is the key feature of the RVM, and is responsible ultimately for its sparsity properties. Now we proceed by computing the posterior distribution:

$$p(\mathbf{w},\boldsymbol{\alpha},\sigma^2|\mathbf{t}) = \frac{p(\mathbf{t}|\mathbf{w},\boldsymbol{\alpha},\sigma^2)p(\mathbf{w},\boldsymbol{\alpha},\sigma^2)}{p(\mathbf{t})} \tag{5}$$

We can't handle to continue to work with the posterior on this form so we decompose it into:

$$p(\mathbf{w},\boldsymbol{\alpha},\sigma^2|\mathbf{t}) = p(\mathbf{w}|\mathbf{t},\boldsymbol{\alpha},\sigma^2)p(\boldsymbol{\alpha},\sigma^2|\mathbf{t}) \tag{6}$$

The first term of (6) can be computed analytically and the posterior distribution over the weights is normally distributed accordingly to:

$$\mathbf{\Sigma} = \left(\sigma^{-2}\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} + \mathbf{A}\right)^{-1}, \qquad \boldsymbol{\mu} = \sigma^{-2}\mathbf{\Sigma}\mathbf{\Phi}^{\mathrm{T}}\mathbf{t} \tag{7}$$

with $\mathbf{A} = \text{diag}\ (\alpha_0 ... \alpha_N)$. One should notice that these results (7) are, except for the contents of $\mathbf{A}$, identical to the ones obtained in 'normal' Bayesian regression. Here the diagonal of $\mathbf{A}$ consists of $N+1$ different hyperparameters whereas in 'normal' Bayesian regression it's composed of one individual hyperparameter α. For the second term on (6) we need to use the type-II maximum likelihood method. We want to maximize $p(\boldsymbol{\alpha}, \sigma^2|\mathbf{t}) \propto p(\mathbf{t}|\boldsymbol{\alpha}, \sigma^2)p(\sigma^2)p(\boldsymbol{\alpha})$, with respect to $\boldsymbol{\alpha}$ and σ^2, which is equivalent[1] to maximize $p(\mathbf{t}|\boldsymbol{\alpha}, \sigma^2)$:

$$\begin{aligned} p(\mathbf{t}|\boldsymbol{\alpha}, \sigma^2) &= \int p(\mathbf{t}|\mathbf{w}, \sigma^2)p(\mathbf{w}|\boldsymbol{\alpha})d\mathbf{w} \\ &= (2\pi)^{-N/2} \left|\sigma^2\mathbf{I} + \boldsymbol{\Phi}\mathbf{A}^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\right|^{-1/2} \exp\left\{-\frac{1}{2}\mathbf{t}^{\mathrm{T}}\left(\sigma^2\mathbf{I} + \boldsymbol{\Phi}\mathbf{A}^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\right)^{-1}\mathbf{t}\right\} \end{aligned} \tag{8}$$

Thus, the key aspect of the RVM becomes the search for the hyperparameters that maximize the hyperparameter posterior. The values of α and σ^2 that maximize (8) cannot be obtained in a closed form, so we must calculate them using iterative estimation. By differentiation of (8) and equalling to zero we get the following update equations which are used iteratively:

$$\alpha_i^{new} = \frac{\gamma_i}{\mu_i^2}, \qquad \gamma_i \equiv 1 - \alpha_i\Sigma_{ii}, \qquad (\sigma^2)^{new} = \frac{||\mathbf{t} - \boldsymbol{\Phi}\boldsymbol{\mu}||^2}{N - \sum_i \gamma_i}, \tag{9}$$

where μ_i is the i-th posterior mean weight from (7), Σ_{ii} is the i-th diagonal element of the posterior weight covariance from (7) computed with the current α and σ^2 values and N denotes the number of data examples. Thus, the learning algorithm consists of repeated application of (7) and (9) until convergence. During this process, many α values will tend to infinity meaning that we are certain that the corresponding weights are zero. Given this, the corresponding basis functions can be 'pruned'. After convergence, we compute the predictive distribution for a new data point $\mathbf{x}_*$ using:

$$p(t_*|\mathbf{t}, \hat{\boldsymbol{\alpha}}, \hat{\sigma}^2) = \int p(t_*|\mathbf{w}, \hat{\sigma}^2)p(\mathbf{w}|\mathbf{t}, \hat{\boldsymbol{\alpha}}, \hat{\sigma}^2)d\mathbf{w} \tag{10}$$

where $\hat{\boldsymbol{\alpha}}$ and $\hat{\sigma}^2$ are the maximizing parameters of (8). The predictive distribution is Gaussian with:

$$y_* = \boldsymbol{\mu}^{\mathrm{T}}\phi(\mathbf{x}_*), \qquad \sigma_*^2 = \hat{\sigma}^2 + \phi(\mathbf{x}_*)^{\mathrm{T}}\boldsymbol{\Sigma}\phi(\mathbf{x}_*) \tag{11}$$

1.2 Classification

With respect to classification tasks, the RVM follows a similar procedure with the one described for regression in the previous section. For two-class classification, we generalize the previously described linear model by applying the logistic regression function $\sigma(x) = 1/(1+e^{-x})$ to $y(\mathbf{x})$. The model now becomes $y(\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}))$. While using a Bernoulli distribution for $p(t|\mathbf{x})$, the likelihood becomes:

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} \sigma\left\{y(\mathbf{x}_n; \mathbf{w})\right\}^{t_n} \left[1 - \sigma\left\{y(\mathbf{x}_n; \mathbf{w})\right\}\right]^{1-t_n} \tag{12}$$

where $t_n \in \{0, 1\}$. In contrast to the regression model, we cannot integrate out the weights analytically, so we need to approximate the posterior distribution (Laplace method). We begin by initializing the hyperparameter $\boldsymbol{\alpha}$ vector. For this given value of $\boldsymbol{\alpha}$, we then build a Gaussian approximation to the posterior distribution and thereby obtain an approximation to the marginal likelihood. Maximization of this approximate marginal likelihood then leads to a re-estimated value for $\boldsymbol{\alpha}$, and the process is repeated until convergence. The mode of the posterior is obtained by maximizing:

$$\begin{aligned} \ln p(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha}) &= \ln\{p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\boldsymbol{\alpha})\} - \ln p(\mathbf{t}|\boldsymbol{\alpha}) \\ &= \sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\} - \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{A}\mathbf{w} + C \end{aligned} \tag{13}$$

[1] For the case of uniform hyperpriors. See Tipping, 2001 [2] for general case.

for a fixed set of $\boldsymbol{\alpha}$ values, where $\mathbf{A} = \text{diag}\ (\alpha_i)$. Then we can apply iterative re-weighted least squares, in order to maximize (13). Taking the first and second derivatives:

$$\nabla \ln p(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha}) = \boldsymbol{\Phi}^{\mathrm{T}}(\mathbf{t} - \mathbf{y}) - \mathbf{A}\mathbf{w}, \qquad \nabla\nabla \ln p(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha}) = -\left(\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{B}\boldsymbol{\Phi} + \mathbf{A}\right) \tag{14}$$

where $\mathbf{B} = \text{diag}\ (\beta_1 ... \beta_N)$ is a diagonal matrix with $\beta_n = \sigma\{y(\mathbf{x}_n)\}[1 - \sigma\{y(\mathbf{x}_n)\}]$. The mean and covariance of the approximation of the posterior distribution are the following:

$$\hat{\mathbf{w}} = \mathbf{A}^{-1}\boldsymbol{\Phi}^{\mathrm{T}}(\mathbf{t} - \mathbf{y}), \qquad \boldsymbol{\Sigma} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{B}\Phi + \mathbf{A}\right)^{-1} \tag{15}$$

The hyperparameters $\boldsymbol{\alpha}$ estimation remains the same as in the regression case so we can apply the algorithm described in the previous section.

2 Methods

The implementation followed the description of the previous section. For the **regression** paradigm, as previously stated, the key difference between the RVM and normal Bayesian linear regression is the estimation of the hyperparameters $\boldsymbol{\alpha}$ (as well as the use of a kernel function). We begin by initializing σ^2 and the vector $\boldsymbol{\alpha}$. Then, we iteratively apply equations (9) while, at the same time, updating the posterior statistics $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ using equations (7), until some convergence criteria is satisfied. During this procedure, we prune the $\boldsymbol{\alpha}$'s that are higher than a certain threshold. After the termination of the learning process, we use equations (11) to make predictions for new data points. With respect to **classification** tasks, we adapt the target conditional distribution by applying the logistic sigmoid function making the RVM similar to the Bayesian logistic regression. However, the key difference is, once again, that this model uses a separate hyperparameter for each weight (ARD prior). We begin by initializing the vector $\boldsymbol{\alpha}$. Then, we iteratively apply equations (9) (only for $\boldsymbol{\alpha}$) while, at the same time, updating the posterior statistics $\boldsymbol{\Sigma}$ and $\hat{\boldsymbol{w}}$ using the Gaussian approximation given by equations (15), until some convergence criteria is satisfied. During this procedure, we prune the $\boldsymbol{\alpha}$'s that are higher than a certain threshold. After the termination of the learning process, we use $\hat{y} = \sigma\{\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_*)\}$ to make predictions for new data points.

For the implementation it was used a Python environment. Scipy was used to optimize the posterior distribution (13) for classification, Numpy for matrices operations, Scikit-learn for metrics and Matplotlib/Seaborn for plots.

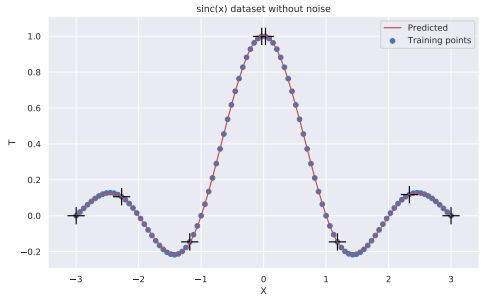
3 Results

This section presents some results using the RVM on regression and, then, on classification tasks. In both cases, it was firstly tested on some toy datasets. Afterwards, it was applied to some benchmark datasets and a comparison between the RVM and the SVM was done. We reproduced the same experiments as in Tipping, 2001 [2] by using the same kernel function for similar datasets.

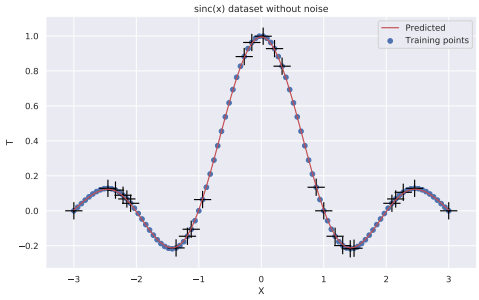
3.1 Regression

First of all, the RVM was tested on a simple artifical linear dataset. It fitted nicely the points while only using 2 relevance vectors, achieving a Root Mean Squared Error (RMSE) of 0.169. Afterwards, the RVM was tested with the $\text{sinc}(x)$ dataset. Figure 1 displays the results. For the **non-noisy** dataset, the RVM achieved a RMSE of 0.0032, while the SVM obtained 0.0042. More interesting to note, is the difference between the number of relevance vectors and the number of

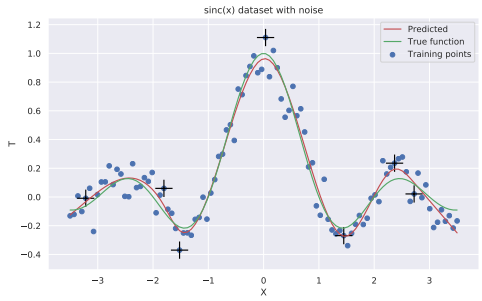
support vectors. The RVM used only 8 relevance vectors, whereas the SVM solution contains 26 support vectors. For the **noisy** dataset, the RVM achieved a RMSE of 0.0409, while the SVM obtained 0.0489. Once again, we should note that the number of relevance vectors is substantially lower than the number of support vectors. The RVM solution features only 7 relevance vectors while the SVM has 46 support vectors. To achieve a comparable RMSE, the RVM solution always contains less relevance vectors than the number of support vectors. Then, the RVM was tested with the $\text{sinc}(x_1) + 0.1x_2$ dataset. Figure 2 presents the used training data and original function alongside with the RVM function approximation. The generated data was sampled from a 21x21 grid with additive Gaussian noise of standard deviation 0.1. As seen in Table 1 the RVM achieved a lower RMSE than the SVM while using a much lower number of basis functions.
For these toy problems on regression, our results align with the ones from Tipping, 2001 [2].



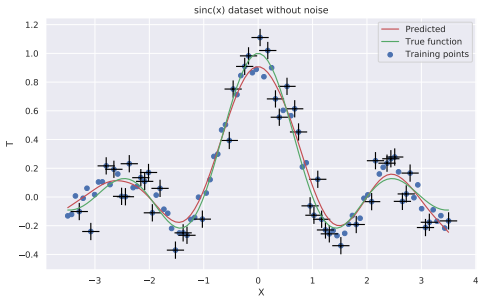
(a) RVM non-noisy $\text{sinc}(x)$.



(b) SVM non-noisy $\text{sinc}(x)$.

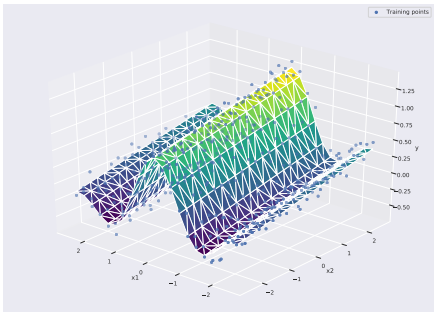


(c) RVM noisy $\text{sinc}(x)$.

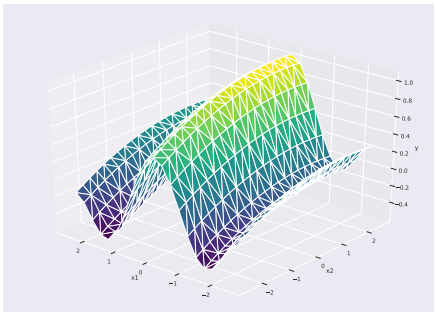


(d) SVM noisy $\text{sinc}(x)$.

Figure 1: RVM & SVM comparison for non-noisy & noisy $\text{sinc}(x)$ datasets.



(a) Original function & training data.



(b) RVM approximation.

Figure 2: RVM approximation of the 2-dimensional $\text{sinc}(x_1) + 0.1x_2$ function.

	Error	# functions
SVM	0.2789	338
RVM	0.0836	7

Table 1: RMSE and number of basis functions for the $\text{sinc}(x_1) + 0.1x_2$ function approximation.

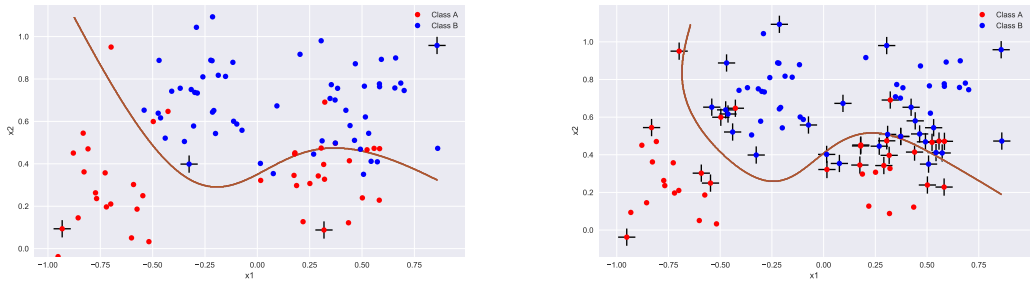
The RVM was then tested against the three Friedman-sets and the Boston housing set for benchmarking. For the Friedman-cases 240 points were generated using one random seed, and the resulting vector machine tested against 1000 points generated from a different random seed. For the Boston housing set 481 samples were used to train the vector machines, and the remaining 25 points used to test them. Table 2 shows the results.
As with the toy problems, our results mostly align with Tipping, 2001 [2], however slightly outperforming his results in some cases.

Data set	N	Test errors		Number of vectors	
		SVM	RVM	SVM	RVM
Friedman1	240	1.9615	1.384 71	31	8
Friedman2	240	310.1234	18.072 89	238	14
Friedman3	240	0.2071	1.108 22	168	35
Boston housing	481	6.2817	2.531 39	455	257

Table 2: RVM and SVM benchmarking for regression tasks.

3.2 Classification

The RVM was first tested on the simple Ripley's synthetic dataset. It is composed of points with 2 coordinates either belonging to a class A or B (binary classification). They were generated by Ripley using a mixture of 2 Gaussians. Figure 3 presents the training dataset composed of 100 points (randomly chosen from Ripley's original 250) we used and the results, i.e. decision boundary and relevance/support vectors, when fitting RVM and SVM to them.



(a) Trained RVM

(b) Trained SVM

Figure 3: Results after training RVM and SVM to Ripley's synthetic dataset. The brown curve is the obtained decision boundary and + markers denote the relevance/support vectors depending on the considered algorithm.

As for regression, the first main difference is between the number of relevance and support vectors. Indeed, RVM needs here only 4 relevance vectors whereas training the SVM leads to 47 support vectors. This difference in the complexity of both models can also be seen with the decision

boundaries : the one obtained with RVM is much smoother and, so, less dependent on the random initialization of the training data set. Then, the accuracy of both approaches was compared on Ripley's synthetic test data set composed of 1000 samples. The test error was 8.3% for RVM and 8.6% when predicting with the SVM. As a result, RVM and SVM perform almost similarly on new data, even if RVM seems to be a bit better here, but the major difference is the incredibly sparse parametrization allowed by RVM.
Afterwards, the RVM was tested against other datasets for benchmarking. Table 3 presents the results. As with the toy dataset, our results mostly align with Tipping, 2001 [2].

Data set	N	d	Test errors		Number of vectors	
			RVM	SVM	RVM	SVM
Pima Diabetes	200	8	20.8 %	21.1 %	3	116
Banana	400	2	11.3 %	10.7 %	11.2	150.5
Breast Cancer	200	9	28.8 %	27.5 %	11.2	128.2
Titanic	150	3	23.4 %	23.0 %	59.0	71.0
Waveform	400	2	11.1 %	11.0 %	20.0	167.4
German	400	21	23.6 %	23.7 %	46.8	425.2
Heart	170	13	20.6 %	16.8 %	15.4	97.0

Table 3: RVM and SVM benchmarking for classification tasks.

4 Discussion & Conclusion

Throughout this report we presented a concise description of the Relevance Vector Machine and validated it on both toy and benchmark datasets. Our results were mostly aligned with the ones obtained in the original article [2]. The RVM showed a good generalization capacity. It achieved a similar performance when compared to the SVM (outperforming it for some problems), however the major improvement was on the sparsity of the obtained solutions. We now provide a concise comparison between the RVM and SVM:

Overall performance: When trying to optimize the hyper-parameters of both algorithms so that they both give their best accuracy on test data, we could notice that they performed almost similarly on regression and classification tasks, even if RVM was often a bit better.

Model sparsity: The main difference between RVM and SVM is the number of basis functions needed to predict on new input data. Indeed, through training, the RVM puts much more of its weights to 0 and, so, to achieve a similar accuracy, the RVM uses way less relevance vectors than support vectors for the SVM.

Generalization: As a result, the RVM ends up with simpler models. By reducing the complexity, the algorithm gains some generalization capabilities and, then, is less prone to overfitting than the SVM. Though, to reduce the SVM dependence on training data initialization, regularization techniques can be used. Then, the advantage of the RVM can be seen as the removal of the need to use such techniques.

Probabilistic output: The RVM outputs contain more information than the SVM because its predictions have a probabilistic meaning, whereas the SVM outputs a point estimate. On regression tasks, the RVM is able to quantify the uncertainty on the predicted values by the inclusion of error bars and, with respect to classification, predicts a class membership probability as opposed to the point estimate given by the SVM. This uncertainty estimation is crucial to increase the model interpretability making it more suitable to many applications where a point estimate is not enough. However, we should not always trust the RVM's predictive distribution. One should also note, that

there are extensions to the SVM that can output a class probability, although, the article authors [2] (and others) argue that the estimates are unreliable.

Computational cost/complexity & scalability: The SVM translates into a convex optimization problem while the RVM does not. The iterative procedure to estimate the hyperparameters $\boldsymbol{\alpha}$ that maximize (8) can get stuck in a local optimum because the mode of $p(\boldsymbol{\alpha}, \sigma^2|\mathbf{t})$ is not unique. Despite this, its predictive approximation is very effective in general. Differently, as mentioned before, the SVM optimization space is convex which means that the global solution will be obtained. When using SVMs we need to estimate some hyperparameters (C in classification and ϵ in regression). This usually entails a cross-validation procedure, which is wasteful both of computation and data. On the other hand, the RVM doesn't need to use this kind of procedures because its hyperparameters are automatically estimated from the data. For classification tasks we need to rely on the Laplace approximation to the posterior distribution. For this linearly parameterised model the posterior is unimodal so we expect good results but still we are denied to closed form solutions. With respect to the algorithm complexity, the inverse operation to compute the posterior is the most costly operation, bounding the method with $O(M^3)$ for time and $O(M^2)$ for space complexity where M denotes the number of basis-functions. Although, the pruning reduces M to a manageable number, in many problems, the RVM model at initialization will lead to extended training times making the algorithm non-scalable. There are, however, modifications to the RVM method described on this report that minimize this problem.
Finally, when considering scalability in terms of data storage, we cannot ignore one drawback of the SVM which is the necessity to store support vectors for prediction: by reducing the complexity of the model and, as a result, the needed storage size, the RVM can become more memory-scalable and increase prediction speed.

Basis functions: With respect to the RVM there is no constraint over the type of basis functions that may be used, although it can influence the computational efficiency of the algorithm. On the other hand, for the SVM, the used kernel function must satisfy Mercer's condition.

References

[1] M. E. Tipping. *The Relevance Vector Machine*. MIT Press, 2000.

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

[3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] Shi, Yi & Xiong, Fenfen & Xiu, Renqiang & Liu, Yu. *A comparative study of relevant vector machine and support vector machine in uncertainty analysis*. 2013.