Least Squares Support Vector Machines

Rohan Shiloh Shah

April 29, 2005

Contents

1 Introduction

Support Vector Classification is a multi-class learning algorithm; it identifies each class by first projecting the data into a higher dimensional space and then using a hyperplane (or hyperplanes when there are more than 2 classes) to separate and classify the projected data[1]. The hyperplane that an SVM learns has a maximised margin amongst all hyperplanes that separate the data. The squared error is also minimised simultaneously while determining the optimal separating hyperplane. Intuitively we see that maximising the margin and minimising the squared error produces a decision boundary that is both accurate and generalises well to test data.

Support Vector Regression is similar to SVM Classification in that it learns a linear regression function in a higher dimensional space. The learnt function deviates the least from the training data amongst all such linear surfaces in the expanded space, according to some loss function. This regression surface is taken to be a weighted linear combination of a set (possibly infinite) of basis functions in the input space(see [1]), so the surface is non-linear in the input space and linear in the feature space.

1.1 *Vapnik's SVM Regression*

For training data $\{x_i, y_i\}_{i=1}^{N}$, a mapping m from the input space to what is called a feature space (some higher dimensional space) is defined:

$$(x_1, x_2, \cdots, x_N) \mapsto_m (\phi_1, \phi_2, \cdots, \phi_r) \circ (x_1, x_2, \cdots, x_N) \tag{1}$$

where r is the size of the feature vector (ie, the dimension of the feature space) and ϕ_i is the i^{th} feature. In this way every x_i can be projected into an r dimensional space with coordinates given by $(\phi_1, \phi_2, \cdots, \phi_r) \circ (x_i)$. The hope is that r is not too large, because a reduced feature space dimension will involve less computation, and more significantly, will generalize better. A linear function in the feature space can now be defined as:

$$y_{\mathbf{w}}(x_i) = \mathbf{w}^\top \phi(x_i) + \beta_0 \tag{2}$$

The loss function we will use to evaluate the accuracy of the regression surface is the empirical risk which is defined as

$$R_{emp} = \frac{1}{N} \sum_{i=1}^{N} |y_i - y_{\mathbf{w}}(x_i)|_\epsilon \tag{3}$$

so our primal optimisation task maximises the margin: $\min_{w,\xi} \frac{1}{2} w^T w + c \sum_{i=1}^{N} \xi_i$ subject to the correct classification of the data[2]: $|y_i - y_{\mathbf{w}}(x_i)| \leq \epsilon + \xi_i, i = 1, \cdots, N$ and $\xi_i \geq$

[1]This assumes the data is linearly separable. For the case where the data is not linearly separable, we define a slack variable ξ_i that measures the error associated with training example i, and find a hyperplane that minimizes $\sum_i \xi_i$. There is a key tradeoff between the complexity of our model and the sum of the slack variables; if we allow no 'slack', then we will have to map our features to a fairly high dimensional feature space where the training examples are perfectly linearly separable, resulting in an increased model complexity. If we allow a fair amount of 'slack', then we can lower the dimension of our model significantly.

[2]In the case of non-linear function approximation this corresponds to the case where all training points are within an ϵ-tube of accuracy. When small values of ϵ are chosen it might be the case that all

$0, i = 1, \cdots, N$ (see cite2). The lagrangian is:

$$L_{\alpha,\eta}(w, \beta_0, \xi) = \frac{1}{2}w^T w + \sum_{i=1}^{N}(c\xi_i - \eta_i\xi_i) - \sum_{i=1}^{N}\alpha_i(\epsilon + \xi_i + y_{\mathbf{w}}(x_i) - y_i) \tag{4}$$

The vectors $\phi(x)$ and w may be infinite dimensional and hence the need for an equivalent, yet computationally feasible, optimisation task. The saddle point is found by maximizing the lagrangian over the lagrange multipliers after minimising over the remaining variables; differentiating with respect to these remaining variables and setting the differential equal to zero gives us $w = \sum_{i=1}^{N} \alpha_i\phi(x_i)$, $\sum \alpha_i = 0$, and $c - \alpha_i - \eta_i = 0$; substituting these in the lagrangian and simplifying gives us the dual, a quadratic optimisation, which is of the form:

$$\max_{\alpha}\left[-\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j\phi(x_i)\phi(x_j) + \sum_{i=1}^{N}(y_i\alpha_i - \epsilon\alpha_i)\right] \text{such that } \sum_{i=1}^{N}\alpha_i = 0, \alpha_i \in [0, c] \tag{5}$$

We now make use of the *kernel trick* and substitute the dot product in the feature space $\phi(x_i)\phi(x_j)$ with $K(x_k, x_l)$, transforming it into a non-parametric problem since the size of the solution vector α varies with the amount of data N, whereas the primal was parametric as the solution vector in that case, w, varied with the dimension of the feature space and not the number of input data points. The regression surface is now defined as $y_{\alpha,\beta_0}(x_i) = \sum_{k=1}^{N} \alpha_k K(x_i, x_k) + \beta_0$, intuitively this corresponds to a linear combination of basis functions in the input space. One of the advantages of Vapnik's SVM is that we have sparseness, i.e. the quadratic optimisation sets many α_i equal to zero, and hence the regression surface can be redefined as $y_{\alpha,\beta_0}(x_i) = \sum_{k=1}^{S} \alpha_k K(x_i, x_k) + \beta_0$ where S is the number of non-zero α_i's; the corresponding $\phi(x_i)$ are called support vectors and hence S is the number of support vectors.

1.2 *The Least Squares Approach*

There are many aspects of Vapnik's SVM that make it attractive; for instance the sparseness of the solution vector α, the implicit redefining of the optimisation in the input space etc. It still remains computationally difficult as the dual increases in complexity with the size of the dataset. We would like to simplify the original formulation while retaining its many 'good' properties.

We now consider the primal:

$$\min_{w,\beta_0,e} \frac{1}{2}w^T w + \gamma\frac{1}{2}\sum_{i=1}^{N} e_i^2 \text{ such that } y_i = y_{\mathbf{w}}(x_i) + e_i, i = 1, \cdots, N \tag{6}$$

Notice that this is a slightly modified approach; an equality constraint is defined which aligns the regression surface in the feature space exactly with the target surface $\{y_i, i = 1, \cdots N\}$ [3] with the use of a slack variable e_i. Ideally we would like to see

the data does not fit within an epsilon tube of accuracy and hence a further slack variable ξ_i is defined. Furthermore, the sum of these slack variables in minimised.

[3] In classification the analagous interpretation is that overlaps in the target distribution are accomodated

misclassifications in this 'overlap' area minimised and so we optimise the sum of the square of this error variable (see [9]). The lagrangian is: $L_\alpha(w, \beta_0, e) = \frac{1}{2}w^T w + \gamma\frac{1}{2}\sum_{i=1}^N e_i^2 - \sum_{i=1}^N \alpha_i\{w^T\phi(x_i) + \beta_0 + e_i - y_i\}$ and differentiating gives us: $w = \sum_{i=1}^N \alpha_i\phi(x_i)$, $\sum_{i=1}^N \alpha_i = 0$, $\alpha_i = \gamma e_i, i = 1, \cdots, N$ and $w^T\phi(x_i) + \beta_0 + e_i - y_i = 0, i = 1, \cdots, N$. Substituting w and e in the lagrangian gives us the following KKT linear system:

$$\left[\begin{array}{cc} 0 & 1_N^T \\ 1_N & Z^T Z + I/y \end{array}\right]\left[\begin{array}{c} \beta_0 \\ \alpha \end{array}\right] = \left[\begin{array}{c} 0 \\ y \end{array}\right] \text{ where } Z^T = \left[\begin{array}{c} \phi(x_1)^T y_1 \\ \phi(x_2)^T y_2 \\ \cdots \\ \phi(x_N)^T y_N \end{array}\right] \tag{7}$$

The kernel trick can now be applied in forming the matrix $Z^T Z$: $K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$, implicitly transforming the linear system so that computation is performed in the input space instead of the feature space. The resulting regression surface is $y_{\alpha,\beta_0}(x) = \sum_{i=1}^N \alpha_i K(x, x_i) + \beta_0$. Notice that sparseness of the solution vector is not achieved here; post-learning pruning needs to be applied in this case. Tests performed in [9] (pages 95-97) show the least squares SVM to be comparable to Vapnik's SVM on several datasets using different kernels.

For large datasets there exist several efficient numerical methods, two are outlined briefly.

1.2.1 Computational algorithm for LS-SVMs: Nystrom Method

Define the kernel matrix as $\Sigma_{(N,N)} = Z^T Z$ then from (7) we know we have to solve the linear system $\alpha = (\Sigma_{(N,N)} + I/\gamma)^{-1} y$. Using the Sherman-Morrison-Woodbury formula and the eigenvalue decomposition $\Sigma_{(N,N)} = \tilde{U}\tilde{\Lambda}\tilde{U}^T$ we have

$$\alpha = \gamma\left(y - \tilde{U}(\frac{1}{\gamma}I + \tilde{\Lambda}\tilde{U}^T\tilde{U})^{-1}\tilde{\Lambda}\tilde{U}^T y\right) \tag{8}$$

To reduce computation we now apply an approximation by defining $\Sigma_{(M,M)}$ as a random selection of M rows of $\Sigma_{(N,N)}$. We have $\Sigma_{(M,M)} = \bar{U}\bar{\Lambda}\bar{U}^T$ and use the following estimates:

$$\tilde{\lambda}_i = \frac{N}{M}\bar{\lambda}_i \text{ and } \tilde{u}_i = \sqrt{\frac{N}{M}}\frac{1}{\bar{\lambda}_i}\Sigma_{(N,M)}\bar{u}_i \tag{9}$$

where $\bar{\Lambda} = diag([\bar{\lambda}_1; \cdots; \bar{\lambda}_M])$. So we can now compute $\tilde{U}$ and $\tilde{\Lambda}$ for use in (8) from $\bar{U}$ and $\bar{\Lambda}$.

1.2.2 Computational algorithm for LS-SVMs: Incomplete Cholesky Factorization

In solving a quadratic form using an interior point method (applied to SVMs), the most computationally expensive operation is inverting or factorising (see [10]) the sum of the kernel matrix $\alpha_i\alpha_j K(x_i, x_j)$ and a diagonal matrix D to find a solution to $Mu = v$ where $M = D + \alpha_i\alpha_j K(x_i, x_j)$. It is numerically stable to use a product cholesky factorisation:

$$M = \overbrace{L^1 L^2 \cdots L^k}^{L}\Lambda\underbrace{(L^k)^T \cdots (L^2)^T(L^1)^T}_{L^T} \tag{10}$$

Next the following lower triangular systems $L^1u^1 = w, L^2u^2 = u^1, \cdots, L^K u^k = u^{k-1}$ are solved using forward substitution and the following upper triangular systems $(L^k)^T u^{k+1} = \Lambda^{-1}u^k, \cdots, (L^1)^T u = u^{2k-1}$ are solved using back substitution, to obtain the solution u. Solving for u takes $0(n^3)$ time. If $\alpha_i \alpha_j K(x_i, x_j)$ is low rank then each L^i has a special structure and the solution u can be obtained more efficiently in $O(k^2 n)$ time where k is the rank of a $n \times k$ matrix V such that $\alpha_i \alpha_j K(x_i, x_j) = VV^T$. Let v be any column vector from V then assuming $L\Lambda L^T$ is the Cholesky factorisation of D we have

$$D + vv^T = L\Lambda L^T + vv^T = I_n D(I_n)^T + Lpp^T L^T = I_n(\Lambda + pp^T)(I_n)^T = \tilde{L}\tilde{\Lambda}\tilde{L}^T \tag{11}$$

This amounts to updating a Cholesky factorisation by accomodating a rank-one matrix and is easily done in $O(n)$ time. Since $VV^T = \sum_{i=1}^{k} v_i(v_i)^T$, we simply have to apply this rank-one update k times to get the Cholesky factorisation of $D + VV^T$.

1.3 *Feature Selection*

Although our heuristic is to minimize the dimension of the feature space, the essential information contained within the original data should still be captured. Feature selection typically involves choosing the features ϕ_i that are relevant for a given classification task. In text classification one common feature selection technique is to use the *information gain* criteria which measures the decrease in the entropy or 'randomness' of our classifier, that is brought about by including ϕ_i in the feature vector. So a high value of $IG(\phi_i)$ says that ϕ_i is a necessary part of the feature vector and therefore the dimension that is associated with it in the feature space is 'critical' to computing a separating hyperplane or regression surface, specifically one that does not overfit the training data.

So one *manual* way of performing feature selection involves using the information gain to find irrelevant features that are then removed from the feature vector. We say a test feature ϕ_i is irrelevant if its information gain is below some threshold. This method is computationally infeasible because the feature vector might be of infinite dimension and because computing the information gain is a long and boring procedure (even for a computer).

1.3.1 Information Gain

We can compute the information gain of a feature ϕ_t:

$$IG(\phi_t) = \underbrace{\sum_{i=1}^{2} H(c_i)}_{1} - \underbrace{P(\phi_t) \cdot \sum_{i=1}^{2} H(c_i|\phi_t)}_{2} - \underbrace{P(\overline{\phi_t}) \cdot \sum_{i=1}^{2} H(c_i|\overline{\phi_t})}_{3} \tag{12}$$

(1) is simply the entropy of our classifier. (2) is equal to the weighted (by the prior probability of ϕ_t) entropy of our classifier given ϕ_t.

Support Vector Machines have a more robust mechanism for dealing with feature selection; support vectors are those projected data points that lie on (or close to) the margin that defines the separating hyperplane[4]. Or since the separating decision boundary is de-

[4]For regression tasks we take support vectors to be those projected data points that lie on the edge of an ϵ-tube (i.e. $\pm\epsilon$) defined around the regression surface

fined ‘close’ to all examples that are difficult to classify; we call these examples support vectors because they ‘support’ the decision boundary. These *support vectors* essentially define the important features of our projected space and a basis function is associated with each of these support vectors. However, the number of support vector (or features) typically grows linearly with the size of the training set.

High-dimensional feature spaces are not a computational problem for SVM’s, since all computation is performed in the input space, but rather a problem of over-fitting the data. Traditionally, SVM’s have used some post-learning techniques to prune the number of features. Not only is this inefficient but it might destroy some of what has been learnt. *Is there a way to* **sparsify** *the number of features* **during learning**, *assigning each of these features some* **relevance** *in comparison to the other features, and producing a prediction based on the regression surface that has a probabilistic interpretation?*

2 Bayesian Approach to Learning

We start by finding appropriate values for the weights used to define the regression function which in vector notation is given by $y_{\mathbf{w}}(x) = \mathbf{w}^\top \phi(x)$ [5] or in matrix notation, $\mathbf{y_w} = \mathbf{w}^\top \Phi$, where Φ is the $n \times n$ design matrix where $\Phi = [\phi(x_1), \cdots, \phi(x_n)]$ and $\phi(x_i) = [1, K(x_i, x_1), \cdots, K(x_i, x_n)]$. The *penalised least-squares* approach minimises:

$$RSS(\mathbf{y}, \lambda, \mathbf{w}) = \sum_{i=1}^{n} (y(i) - y_w(x_i))^2 + \lambda \int_a^b (y_w(\mathbf{x})'')^2 dt \tag{13}$$

where a regularisation term has been added to control the complexity (more specifically the curvature) of the basis functions considered (see [4]). As λ varies from 0 to ∞ we vary the basis functions from very complex to a simple straight line. If a simpler regularisation term $\frac{1}{2}\sum w_i^2$ is used we can express the residual sum of squares in matrix notation as:

$$RSS(\mathbf{y}, \lambda, \mathbf{w}) = (\mathbf{y} - \mathbf{w}^\top \Phi)'(\mathbf{y} - \mathbf{w}^\top \Phi) + \lambda \mathbf{w}'\mathbf{w} \tag{14}$$

from which it follows after differentiating and setting equal to zero that $\hat{\mathbf{w}}_{PLS} = (\Phi'\Phi + \lambda I)^{-1}\Phi'\mathbf{y}$.

2.1 *Maximum Likelihood and the Hierarchichal Bayes Approach*

Now if we assume that there exists normally distributed noise in our model with mean 0 and variance σ^2 then for $t = y_w(x) + \epsilon$ we can write the likelihood (see [5]) as follows:

$$p(t|x, \mathbf{w}, \sigma^2) = \prod_{i=1}^{n} \frac{1}{(2\pi)^{1/2}\sigma} \exp\left[-\frac{\{t_i - y_w(x_i)\}^2}{2\sigma^2}\right] \tag{15}$$

The maximum likelihood (maximising $p(t|x, \mathbf{w}, \sigma^2)$) estimate is identical to the least-squares estimate (minimizing the residual sum of squares)[6]. We now define a gaussian

[5]Remember $\phi(x_i)$ is a vector - $[\phi_1(x_i), \cdots, \phi_n(x_i)]^\top$

[6]This can be seen by minimising the negative logarithm of the likelihood and comparing it to the least squares approach

prior distribution on the weights, to control the model complexity, with zero mean and variance parameter $1/\alpha$. So we prefer models with $w_i \approx 0$ because this maximises the prior $p(\mathbf{w}|\alpha)$ - essentially this is a regularisation mechanism because we are assigning a higher likelihood to simpler models. For each weight there is a separate hyperparameter α_i, and over each of these hyperparameters we define a separate Gamma distribution with shape parameters **a** and **b**. We also define a Gamma hyperprior over the noise variance σ^2 with shape parameters **c** and **d**. By setting the shape parameters to zero we obtain uniform hyperpriors on a logarithmic scale[7]. So using bayes rule we know that:

$$p(\mathbf{w}|\mathbf{t},\alpha,\sigma^2) = \frac{p(\mathbf{t}|\mathbf{w},\sigma^2)p(\mathbf{w}|\alpha)}{p(\mathbf{t}|\alpha,\sigma^2)} \tag{16}$$

The denominator is a convolution of two gaussians which we have seen before:

$$\begin{aligned} p(\mathbf{t}|\alpha,\sigma^2) &= \int p(\mathbf{t}|\mathbf{w},\sigma^2)p(\mathbf{w}|\alpha)d\mathbf{w} \\ &= (2\pi)^{-n/2}|\sigma^2 I + \alpha^{-1}\Phi^\top\Phi|^{-1/2}\exp\left[-\frac{1}{2}\mathbf{t}^\top(\sigma^2 I + \alpha^{-1}\Phi^\top\Phi)^{-1}\mathbf{t}\right] \end{aligned} \tag{17}$$

which is a gaussian distribution over the data vector $\mathbf{t}$. After multiplying and dividing these 3 gaussian distributions and rearranging we see that $p(\mathbf{w}|\mathbf{t},\alpha,\sigma^2)$ is normally distributed with mean $\mu = (\Phi^\top\Phi + \sigma^2\alpha I)^{-1}\Phi^\top\mathbf{t}$ and covariance matrix $\Sigma = \sigma^2(\Phi^\top\Phi + \sigma^2\alpha I)^{-1}$ which is identical to the *penalised least squares* estimate $\hat{\mathbf{w}}_{PLS}$ with λ replaced by $\sigma^2\alpha_i$. [8] Is'nt that strange? We changed our regularisation technique (from minimising the sum of the weights to defining a prior distribution over the weights that assigns a priori more *significance* to smaller weights) and we changed our solution method (from penalised least squares to a bayesian approach that uses maximum likelihood) and yet the solution is unchanged. Ofcourse, the added advantage of using a bayesian approach is that we had a point estimate before $\hat{\mathbf{w}}_{PLS}$[9] and now we have a distribution $p(\mathbf{w}|\mathbf{t},\alpha,\sigma^2) \sim N(\mu,\Sigma)$ that can be used to define a *probabilistic* estimate.

2.2 *Bayesian Inference*

Since a decomposition, using bayes rule, of the posterior $p(\mathbf{w},\alpha,\sigma^2|\mathbf{t})$ leaves us with a normalising factor $p(\mathbf{t})$ that is analytically intractable we will start with the following decomposition:

$$p(\mathbf{w},\alpha,\sigma^2|\mathbf{t}) = p(\mathbf{w}|\mathbf{t},\alpha,\sigma^2)p(\alpha,\sigma^2|\mathbf{t}) \tag{18}$$

where we know that $p(\mathbf{w}|\mathbf{t},\alpha,\sigma^2) \sim N(\mu,\Sigma)$. For $p(\alpha,\sigma^2|\mathbf{t})$ we find the *most probable* values α_{MP} and σ^2_{MP}. *Type-II Maximum Likelihood* estimation assumes uniform priors for σ^2 and α on the logarithmic scale and then equivalently maximises $p(\mathbf{t}|\alpha,\sigma^2)$ to arrive at estimates for α_{MP} and σ^2_{MP}. So we have reduced our learning task to a search for

[7]I'm not sure why?

[8]Remember α is a vector $[\alpha_1,\cdots,\alpha_n]$, where each α_i is associated with w_i

[9]We could define a distribution around this point estimate with mean $\hat{\mathbf{w}}_{PLS}$ and variance $var(\hat{\mathbf{w}}_{PLS})$ but a change in λ, our preference for the complexity of the regression surface changes the entire distribution and requires a complete recomputation.

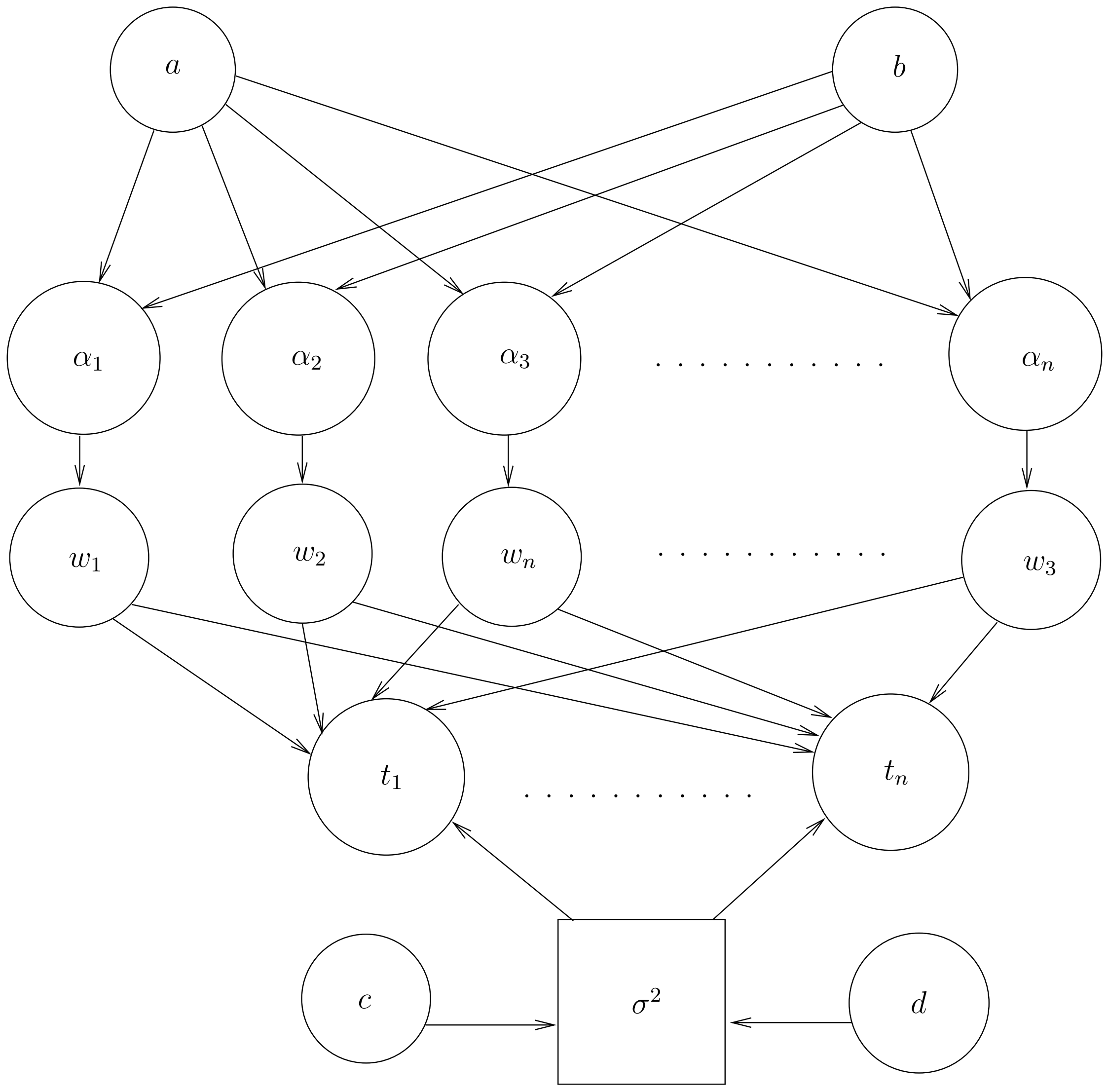

Figure 1: Heirarchical Bayesian Decomposition

maximum likelihood parameters α_{MP} and σ^2_{MP}.

If we assume that the priors are not uniform then it is not possible to obtain a closed form for $p(\mathbf{t}|\alpha, \sigma^2)$ and so we use a *iterative re-estimation* procedure to estimate the parameters α_i and σ_i^2. We first find maximum likelihood estimates for first α_i and then σ^2 then update our posterior statistics μ and Σ and then repeat the procedure, estimate α_i etc. Other *expectation maximisation* like techniques for parameter estimation are also possible.

2.3 *Predictions*

Once our iterative procedure has converged, given some new data $\mathbf{x}_*$ we would like to make predictions. Given the estimates for α and σ^2 we can approximate the predictive distribution as:

$$p(t_*|\mathbf{t}, \alpha_{MP}, \sigma^2_{MP}) \approx \int p(t_*|\mathbf{w}, \sigma^2_{MP}) p(\mathbf{w}|\mathbf{t}, \alpha_{MP}, \sigma^2_{MP}) d\mathbf{w} \tag{19}$$

which is normally distributed with mean $\mu_* = y_\mu(x_*) = \sum \mu_i \phi_i(x_*)$ and variance $\sigma^2_* = \sigma^2_{MP} + \mathbf{f}^\top \Sigma \mathbf{f}$ where $\mathbf{f} = [\phi_1(x_*), \cdots, \phi_n(x_*)]^\top$. Intuitively, we see that the new mean μ_* is simply the model function $y_\mathbf{w}(x)$ evaluated at the posterior mean μ, i.e. the weights assigned to the basis expanded new data $\phi(x_*)$ are a measure of relevance for each of the features in the feature space. *How do these measures of relevance perform feature selection?*

3 Bayesian Approach to Feature Selection - Relevance Vectors

During learning a large proportion of the α_i end up at infinitely large values leading $p(w_i|\mathbf{t}, \alpha, \sigma^2)$ to be highly peaked around zero since α_i^{-1} is the variance of w_i. The vectors for which w_i are not zero are called *relevance vectors*. This replaces all our previous methods of selecting features including the use of *support vectors*. In practise Relevance Vector Machines are able to approximate functions of complexity comparable with those approximated by Support Vector Machines, *but with a fewer number of features.*

4 Results

I used code provided by Michael E. Tipping to run RVM tests on 2 data sets; noisy 'sinc' data for RVM Regression and Ripley's synthetic data for RVM Classification.

4.1 *Output for RVM Regression*

A total of 500 iterations were required to estimate α_i and σ^2 for noisy data generated from the function $sinc(x) = sin(x)/x$. Only gaussian kernels were considered in defining the basis functions and looking at Figure 2 we see that a total of 6 relevance vectors were required to approximate the function with test error 0.0424558. A Support Vector Regression used 36 support vectors and had a higher test error, so we see that RVMs clearly perform better for this dataset.

```
*
Evaluating kernel ...
*
Created RVM ...
kernel: '+gauss'
scale:  3.000000
PHI:    100 x 101
*
Calling hyperparameter estimation routine ...
  100> L = 172.115       Gamma = 5.87 (nz = 14)  s=0.093
  200> L = 172.204       Gamma = 5.83 (nz = 10)  s=0.093
  300> L = 172.208       Gamma = 5.84 (nz = 7)   s=0.093
  400> L = 172.211       Gamma = 5.83 (nz = 7)   s=0.093
  500> L = 172.211       Gamma = 5.83 (nz = 7)   s=0.093
Terminating: max log(alpha) change is 0.000230163 (<0.001).
  503> L = 172.211       Gamma = 5.83 (nz = 6)   s=0.093
*
Hyperparameter estimation complete
non-zero parameters:    6
log10-alpha min/max:    -0.13/1.91

RVM regression test error (RMS): 0.0424558
        estimated noise level: 0.0933 (true: 0.1000)
```

4.2 *Output for RVM Classification*

A total of 252 iterations were required to estimate α_i and σ^2 for Ripley's data. Only gaussian kernels were considered in defining the basis functions and looking at Figure 3 we see that a total of 4 relevance vectors were required to approximate the function (3 positive relevance vectors and 1 negative relevance vector) with test error 8.6%. A Support Vector Regression used 38 support vectors and had a higher test error at 10.8%, so we see that RVMs clearly perform better for this dataset as well.

```
*
Created RVM ...
kernel: '+gauss'
scale:  0.500000
PHI:    100 x 101
*
Calling hyperparameter estimation routine ...
   50> L = -7.630        Gamma = 3.61 (nz = 23)
```

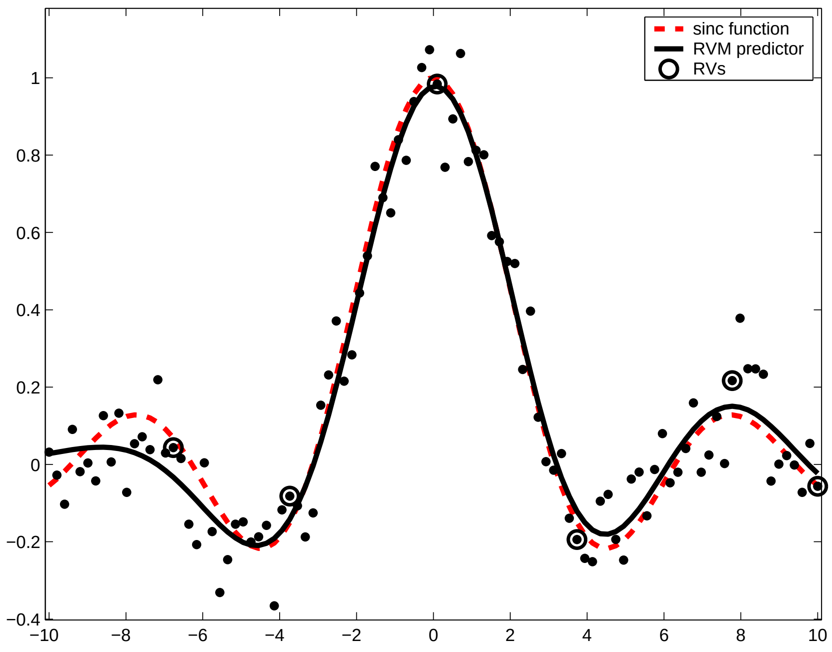

Figure 2: RVM Regression with noisy 'sinc' data

```
  100> L = -7.630        Gamma = 3.60 (nz = 13)
  150> L = -7.630        Gamma = 3.60 (nz = 8)
  200> L = -7.630        Gamma = 3.60 (nz = 6)
  250> L = -7.630        Gamma = 3.60 (nz = 5)
Terminating: max log(alpha) change is 2.67016e-07 (<0.001).
  252> L = -7.630        Gamma = 3.60 (nz = 4)
*
Hyperparameter estimation complete
non-zero parameters:    4
log10-alpha min/max:    -1.86/-0.99

RVM CLASSIFICATION test error: 8.60%
```

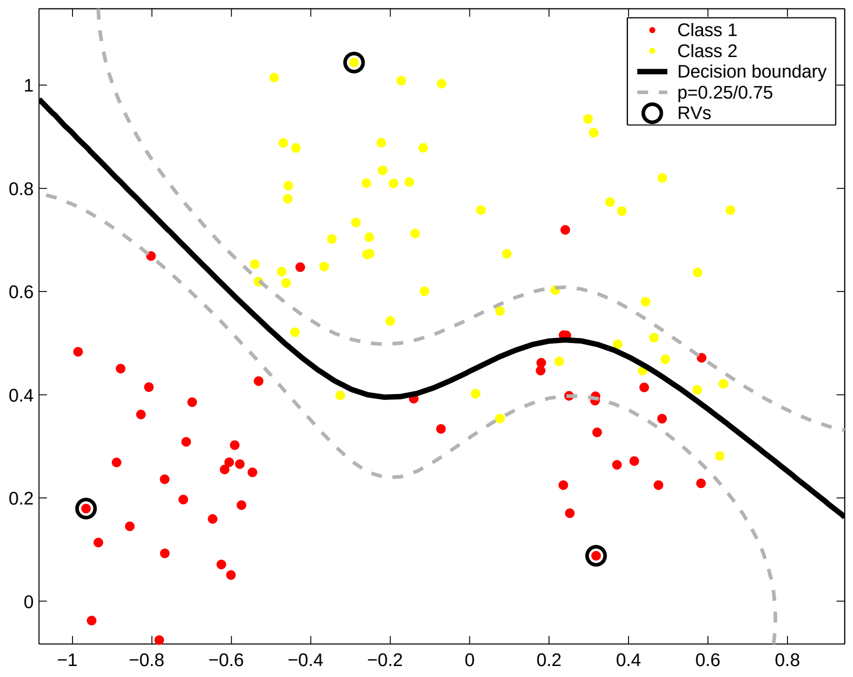

Figure 3: RVM Classification of Ripley's synthetic data

References

[1] Christopher J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition", 1998

[2] Nello Cristianini, John Shawe-Taylor, "An Introduction to Support Vector Machines", 2000

[3] Radford Neal, "Bayesian Learning for Neural Networks", 1996

[4] George Seber, Alan Lee, "Linear Regression Analysis", 2003

[5] Michael E. Tipping, "Sparse Bayesian Learning and the Relevance Vector Machine", 2001

[6] Nello Cristianini, Lecture Notes from `www.cs.berkeley.edu/ñello`, 2002

[7] Thorsten Joachims, SVM-*light*, `svmlight.joachims.org`

[8] Trevor Hastie, 'The Elements of Statistical Learning', 2001

[9] Suykens, Van Gestel, De Brabanter, De Moor, Vandewalle, 'Least Squares Support Vector Machines', 2002

[10] Schienberg, Fine, 'Efficient SVM training using Low-Rank SVM Kernel Representations', 2001