RVM++: Relevance Vector Machine in C++

XIAO Jianxiong

csxjx@cse.ust.hk

1 Introduction

This project is to implement the Relevance Vector Machine [8, 3, 9, 1] in C++ for both classification and regression.

2 Object Oriented Design

The class diagram is listed in Fig. 1. The class RVMmachine represented the model for training and testing. RVMclassifier and RVMregressor inherit from RVMmachine by implementing different hyper-parameters estimation function. A data point is represented by an object of RVMpoint, while a RVMdataset object contains pointers to multiple RVMpoint objects. A RVMdataset object is used as the training dataset for a RVMmachine instant. By calling RVMmachine::train() function, this training dataset is used to train the model and the results are a set of weights stored in each RVMpoints in the training dataset. Then RVMmachine::predictDataset(RVMdataset*) and RVMmachine::predictPoint(RVMpoint*) can be called to predict the new label for a dataset or a single data point. For each RVMmachine instance, a pointer to a RVMkernel object is maintained. The RVMkernel object represents the kernel function that is used for both training and testing. RVMkernel contains all parameters including bias, and uses a function pointer to call the respected kernel function according to its name.

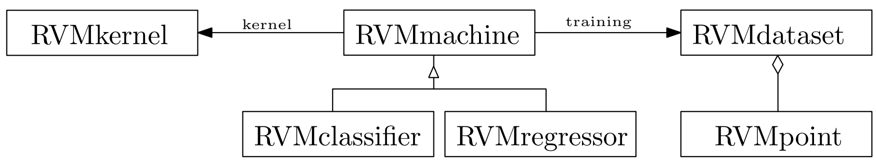

Figure 1: Class Relationships

3 Empirical Comparison with SVM classification

To justify the classification quality and sparsity of RVM, we compare our implementation with LibSVM v2.86 [2] on three real-world datasets from UC Irvine Machine Learning Repository [5].

3.1 Data Scaling

Scaling the elements in feature vectors before applying RVM or SVM is very important. The main advantage is to avoid attributes in greater numeric ranges dominate those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the calculation. Because kernel values usually depend on the inner products of feature vectors, e.g. the linear kernel and the polynomial kernel, large attribute values might cause numerical problems. We recommend linearly scaling each attribute to the range $[-1, +1]$.

3.2 Testing Dataset

Iris Dataset There are 3 classes and 150 instance in the Iris dataset. In the experiment, only the Iris Setosa class and Iris Versicolour class are used. And the total number of instance is 100, 50 for each class. 20 instances are randomly picked for testing and the remaining 80 instances are used for training. For both data sets, the first column is the class label, binary {0,1} representing {Setosa,Versicolour}. The 3rd to 5th column are representing sepal length in cm, sepal width in cm, petal length in cm, petal width in cm respectively.

Wine Dataset These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. In the experiment, only class 1 and class 2 are used. 21 instances are randomly selected for testing and remaining 110 instances for training. For both data set, the first column is the class label, binary {0,1} representing {class 2, class 1}. For both RVM and SVM, We scale each attribute by shifting the mean to zero and dividing by the standard deviation.

Image Segmentation Dataset The instances were drawn randomly from a database of 7 outdoor images. The images were hand segmented to create a classification for every pixel. The Image data are described by 18 high-level numeric-valued attributes, 7 classes. In the experiment, only the grass class and the sky class are used. The training data set is segmentation_data.txt, with 60 instances. The testing data set is segmentation_test.txt, with 600 instances. For both data set, the first column is the class label, binary {0,1} representing {sky, grass}. For both RVM and SVM, We scale each attribute by shifting the mean to zero and dividing by the standard deviation.

		RVM++		LIBSVM	
Dataset	Kernel	# Vectors	Testing Err	# Vectors	Testing Err
Iris	+gauss	4	0	10	0
Wine	+poly1	2	0	41	0
Image Segmentation	+poly1	1	0	18	0.33%

Table 1: Empirical Comparison with SVM classification on UCI Datasets

3.3 Classification Results

The experiment results are shown in Tab. 1. Notice that for both RVM++ and LIBSVM, we use the same kernel with all default parameters. Specifically, for Iris dataset, RBF kernel is used, while linear kernel (=degree 1 polynomial kernel) is used for the other two dataset. The classification performance is very good for both RVM and SVM. One observation is that the result of RVM is very sparse comparing with SVM. More interestingly, sometimes there is no relevance vectors for some class. For example, in the image segmentation dataset, RVM++ only relies on one relevance vectors which is impossible for two class classification of SVM. We interpret this as that since RVM is originally designed for regression, for some class, even without a relevance vector, its decision boundaries can still recovered by the relevance vectors from other class.

4 Implementation

4.1 Usage

The basic usage of the classifier and regressor is:

RVMClassifier Training.file Testing.file Out.file

RVMRegressor Training.file Testing.file Out.file

To have more control of the initial parameters and maximal number of iterations, you can use:

RVMClassifier Training.file Testing.file Out.file [Kernel=+gauss Width=.5 InitAlpha=(1/N)^2 MaxIts=500]

RVMRegressor Training.file Testing.file Out.file [Kernel=+gauss Width=.5 InitAlpha=(1/N)^2 InitBeta=STD/10 MaxIts=500]

The set of kernel functions provided in this implementation is listed in Tab. 2. As in [10], let the name of the kernel prefix with '+' to add bias. For the (homogeneous) polynomial kernels, the exponent is regarded in the kernel name. For example, "poly4.0" means the exponent is 4.0 for polynomial kernels.

The file format for both training data and testing data is the same: each row represents a data point with the first column as its label, second column indicating the dimension of the feature vectors, and the corresponding feature vector starting from the third column:

label t_i dimension d feature x_0 feature x_1 ... feature x_d

Notice that there should not be any comment, new line or any visible or invisible character in the end of the file.

Choices	Kernel
gauss	Gaussian
laplace	Laplacian
poly	Polynomial
hpoly	Homogeneous Polynomial
cauchy	Cauchy (heavy tailed) in distance
cubic	Cube of distance
r	Distance
tps	'Thin-plate' spline
bubble	Neighbourhood indicator

Table 2: Kernel Choices

4.2 Source Code

This implementation tries to follow the notations used in [1, 10], and also tries to make the algorithm as close to [1, 10] as possible for easy understanding and comparison. The pseudo-code for both classification and regression are shown in Alg. 1 and 2. Comparing with the pseudo-code provided in [4], Alg. 1 and 2 cover every step in greater details. On trick step is that, in classification, instead of maximizing

$$\ln p(\mathbf{w}\,|\mathbf{t},\boldsymbol{\alpha}) = \sum_{n=1}^{N} \{t_n \ln y_n + (1-t_n)\ln(1-y_n)\} - \frac{1}{2}\mathbf{w}^T\mathbf{A}\mathbf{w} + \text{const},$$

its inverse $J = -\ln p(\mathbf{w}\,|\mathbf{t},\boldsymbol{\alpha})$ is minimized by Iterative Reweighted Least Squares (IRLS).

The codes of Gauss-Jordan elimination, Cholesky decomposition and PostScript plotting are modified and based on the codes offered in the book [6]. The kernel functions are in the file "RVMfunc.h" and "RVMfunc.cpp". To let the RVM support different types of features such as Graph kernels, you may change them and modified the definition of the class RVMpoint correspondingly. And except the initial $\boldsymbol{\alpha}$, β and maximal iterations allowed, other default parameters are all located in the file "RVMconstant.h" which can be easily changed.

Algorithm 1 Pseudo-code for RVM Classification Training

Require: A data matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$, $\mathbf{t} \in \{0,1\}^m$ and initial value $\boldsymbol{\alpha}$
Require: The maximum number of iterations $i_{\max}$ and $k_{\max}$
Require: A tolerance for pruning $\mathrm{TOL} \in \mathbb{R}^+$, and minimal $\ln \alpha$ change $\delta_{\min}$

$\mathbf{w} \leftarrow \mathbf{0}$
$\boldsymbol{\Phi} \leftarrow \kappa(\mathbf{X}, \mathbf{X})$ ▷ The design matrix $\Phi_{ni} := \phi_i(\mathbf{x}_n)$
for $i = 1, \ldots, i_{\max}$ **do**
  $\mathbf{n} = \langle j \in \{1, \ldots, n\} \,|\alpha_j < \mathrm{TOL}\rangle$ ▷ All non-pruned indices
  $\boldsymbol{\Phi}_\mathbf{n} \in \mathbb{R}^{m \times |\mathbf{n}|} := \boldsymbol{\Phi}(\mathbf{n})$ contains the $|\mathbf{n}|$ columns from $\boldsymbol{\Phi}$ indexed by $\mathbf{n}$
  $\boldsymbol{\alpha}_\mathbf{n} := \boldsymbol{\alpha}(\mathbf{n}), \mathbf{w}_\mathbf{n} := \mathbf{w}(\mathbf{n})$
  $\mathbf{A} \leftarrow \mathrm{diag}(\boldsymbol{\alpha}_\mathbf{n})$, $\mathbf{y} \leftarrow \mathrm{sigmoid}(\boldsymbol{\Phi}_\mathbf{n}\mathbf{w}_\mathbf{n})$
  $J \leftarrow -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\} + \frac{1}{2}\mathbf{w}_\mathbf{n}{}^T\mathbf{A}\mathbf{w}_\mathbf{n}$
  for $k = 1, \ldots, k_{\max}$ **do** ▷ Iterative Reweighted Least Squares
    $\mathbf{e} \leftarrow \mathbf{t} - \mathbf{y}$, $b_j := y_j(1-y_j)$, $\mathbf{B} \leftarrow \mathrm{diag}(\mathbf{b})$
    $\mathbf{g} \leftarrow -\nabla J = \boldsymbol{\Phi}_\mathbf{n}^T\mathbf{e} - \mathbf{A}\mathbf{w}_\mathbf{n}$ ▷ Negative Gradient of J
    if $\frac{\|\mathbf{g}\|}{|\mathbf{n}|} < \epsilon$ **then**
      break
    end if
    $\mathbf{H} \leftarrow \nabla\nabla J = \boldsymbol{\Phi}_\mathbf{n}^T\mathbf{B}\boldsymbol{\Phi}_\mathbf{n} + \mathbf{A}$ ▷ Hessian of J
    Cholesky Decomposition $\mathbf{H} = \mathbf{U}^T\mathbf{U}$
    $\Delta \leftarrow \mathbf{U}^{-1}\left(\mathbf{U}^{-T}\mathbf{g}\right)$ ▷ $\Delta := \mathbf{H}^{-1}\mathbf{g}$
    $\lambda \leftarrow 1$
    repeat
      $\widetilde{\mathbf{w}} \leftarrow \mathbf{w}_\mathbf{n} + \lambda\Delta$
      $\mathbf{y} \leftarrow \mathrm{sigmoid}(\boldsymbol{\Phi}_\mathbf{n}\widetilde{\mathbf{w}})$
      $\widetilde{J} \leftarrow -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\} + \frac{1}{2}\widetilde{\mathbf{w}}^T\mathbf{A}\widetilde{\mathbf{w}}$
      $\lambda \leftarrow \frac{\lambda}{2}$
    until $\widetilde{J} < J$ ▷ To minimize J and to maximize $\ln p(\mathbf{w}_\mathbf{n}|\mathbf{t}, \boldsymbol{\alpha}_\mathbf{n})$
    $\mathbf{w}_\mathbf{n} \leftarrow \widetilde{\mathbf{w}}, J \leftarrow \widetilde{J}$
  end for
  $\boldsymbol{\Sigma} \leftarrow -(\nabla\nabla J)^{-1} = \mathbf{H}^{-1}$
  for all $j \in \mathbf{n}$ **do**
    $\gamma_j \leftarrow 1 - \alpha_j\Sigma_{jj}$, $\alpha_j^{old} \leftarrow \alpha_j$
    if $i < i_{\max}/2$ **then**
      $\alpha_j \leftarrow \gamma_j / w_j^2$ ▷ MacKay-style update[8]
    else
      $\alpha_j \leftarrow \gamma_j\left(w_j^2/\gamma_j - \Sigma_{jj}\right)^{-1}$ ▷ Speed up by hybrid update[3]
    end if
    $\delta_j \leftarrow \left|\ln(\alpha_j) - \ln\left(\alpha_j^{old}\right)\right|$
  end for
  if $\max_j\{\delta_j\} < \delta_{\min}$ **then**
    break
  end if
end for

Algorithm 2 Pseudo-code for RVM Regression Training

Require: A data matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$, $\mathbf{t} \in \mathbb{R}^{m}$ and initial value $\boldsymbol{\alpha}$, β
Require: The maximum number of iterations $i_{\max}$
Require: A tolerance for pruning $\mathrm{TOL} \in \mathbb{R}^{+}$, and minimal $\ln \alpha$ change $\delta_{\min}$

$\mathbf{w} \leftarrow \mathbf{0}$
$\boldsymbol{\Phi} \leftarrow \kappa(\mathbf{X}, \mathbf{X})$ ▷ The design matrix $\Phi_{ni} := \phi_i(\mathbf{x}_n)$
for $i = 1, \ldots, i_{\max}$ **do**
    $\mathbf{n} = \langle j \in \{1, \ldots, n\} \,|\, \alpha_j < \mathrm{TOL} \rangle$ ▷ All non-pruned indices
    $\boldsymbol{\Phi}_{\mathbf{n}} \in \mathbb{R}^{m \times |\mathbf{n}|} := \boldsymbol{\Phi}(\mathbf{n})$ contains the $|\mathbf{n}|$ columns from $\boldsymbol{\Phi}$ indexed by $\mathbf{n}$
    $\boldsymbol{\alpha}_{\mathbf{n}} := \boldsymbol{\alpha}(\mathbf{n}), \mathbf{w}_{\mathbf{n}} := \mathbf{w}(\mathbf{n})$
    $\mathbf{A} \leftarrow \mathrm{diag}(\boldsymbol{\alpha}_{\mathbf{n}})$, $\mathbf{y} \leftarrow \boldsymbol{\Phi}_{\mathbf{n}} \mathbf{w}_{\mathbf{n}}$
    $\mathbf{H} \leftarrow \beta \boldsymbol{\Phi}_{\mathbf{n}}^{T} \boldsymbol{\Phi}_{\mathbf{n}} + \mathbf{A}$
    Cholesky Decomposition $\mathbf{H} = \mathbf{U}^{T}\mathbf{U}$
    $\boldsymbol{\Sigma} \leftarrow \mathbf{U}^{-1}\mathbf{U}^{-T}$ ▷ $\boldsymbol{\Sigma} = \mathbf{H}^{-1}$
    $\mathbf{w}_{\mathbf{n}} \leftarrow \beta \boldsymbol{\Sigma} \boldsymbol{\Phi}_{\mathbf{n}}^{T} \mathbf{t}$
    $e \leftarrow \|\mathbf{t} - \boldsymbol{\Phi}_{\mathbf{n}} \mathbf{w}_{\mathbf{n}}\|^{2}$
    for all $j \in \mathbf{n}$ **do**
        $\gamma_j \leftarrow 1 - \alpha_j \Sigma_{jj}$, $\alpha_j^{old} \leftarrow \alpha_j$
        if $i < i_{\max}/2$ **then**
            $\alpha_j \leftarrow \gamma_j / w_j^{2}$ ▷ MacKay-style update[8]
        else
            $\alpha_j \leftarrow \gamma_j \left(w_j^{2} / \gamma_j - \Sigma_{jj} \right)^{-1}$ ▷ Speed up by hybrid update[3]
        end if
        $\delta_j \leftarrow \left| \ln(\alpha_j) - \ln\left(\alpha_j^{old}\right) \right|$
    end for
    $\beta \leftarrow \left(N - \sum_i \gamma_i\right) / e$
    if $\max_j \{\delta_j\} < \delta_{\min}$ **then**
        break
    end if
end for

5 Demo: 2D Classification

This demo is to train a classifier on Ripley's synthetic data[1] from [7]. You can run the program in "bin" directory. The PostScript plot named "classification2d_demo_plot.ps" will be stored in the same directory at the end of the program. If you are in Windows environment, GSView in "C:\Program Files\Ghostgum\gsview\gsview32.exe" will automatically open to show the plotted figure as in 2. The decision boundaries, 0.25 and 0.75 confidence contour are all plotted. And the two class data points are plotted as triangle and circle ball, while relevant vectors are braced by a square.

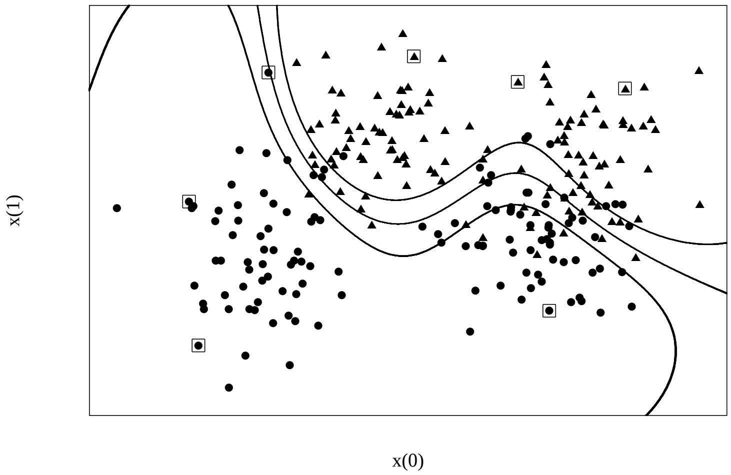

Figure 2: 2D Classification Demo. The middle curve is the decision boundary for separating the two classes. And the other two curves are the contour corresponding to 25% and 75% pobability. And the two class data points are plotted as triangle and circle ball, while relevant vectors are braced by a square.

6 Demo: 1D Regression

This demo is to train a regressor on a noisy 'sinc' data with 100 data points. You can run the program in "bin" directory. The PostScript plot named "regression1d_demo_plot.ps" will be stored in the same directory at the end of the program. If you are in Windows environment, GSView in "C:\Program Files\Ghostgum\gsview\gsview32.exe" will automatically open to show the plotted figure as in 3. The regression curve generated by the RVMregressor is plotted with the input noise data point, while the relevant vectors are braced by a square.

[1] Available for download from `http://www.stats.ox.ac.uk/pub/PRNN/`

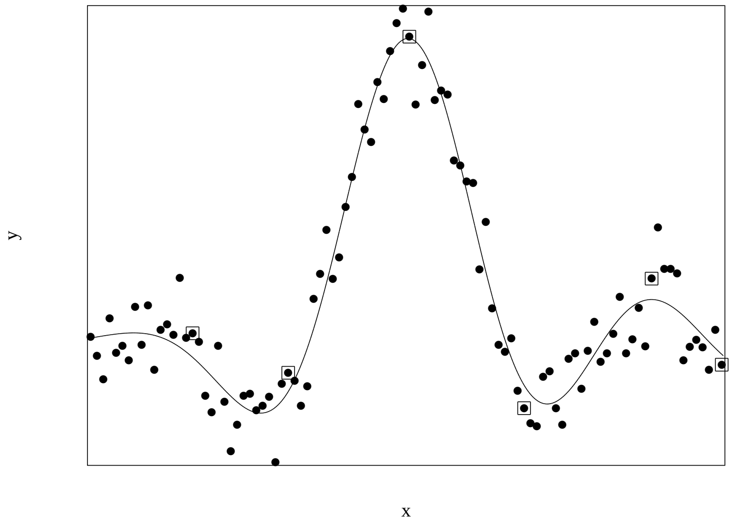

Figure 3: 1D Regression Demo.

7 Conclusion

This report details several aspects of the implementation of the Relevance Vector Machine in C++. An empirical comparison with SVM for classification is also presented. Since all matrix operations are just coded in very intuitive way, the speed of the implementation is quite slow. As a future work, the matrix operations will be replaced by calling some highly optimized matrix libraries such as BLAS, LAPACK, ATLAS or GotoBLAS, which are extensively used in Matlab$^{\mathrm{TM}}$.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

[3] Anita C. Faul and Michael E. Tipping. Analysis of sparse bayesian learning. In *In Proceedings of Neural Information Processing Systems*, 2001.

[4] Ralf Herbrich. *Learning Kernel Classifiers:Theory and Algorithms*. The MIT Press, 2002.

[5] UC Irvine. Uc irvine machine learning repository. `http://archive.ics.uci.edu/ml/`.

[6] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, third edition edition, 2007.

[7] B.D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[8] Michael E. Tipping. The relevance vector machine. In In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, volume 12, pages 652–658. The MIT Press, 2000.

[9] Michael E. Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

[10] Michael E. Tipping. SparseBayes: A matlab implementation of sparse bayesian learning. `http://www.miketipping.com/index.php?page=rvm`, 2002.