Relevance Vector Machine

Philip Robinson

Oregon Health Sciences University

June 23, 2017

1 Introduction

Due to my interest in support vector machines, I selected to study Michael Tipping's relevance vector machine. Relevance vector machines are sparse models that yield probabilistic predictions for supervised classification and regression problems. Support vector machines, identify a maximal margin hyperplane that separates your data (with some associated cost parameter). Alternatively, relevance vector machines fit a Gaussian about this decision boundary, whose variance are described by fitting relevance vectors. I provide the first relevance vector machine implementation in `julia`.

1.1 Relevance vector machine

Usually our predictions, modeled by

$$y(\vec{x};\vec{w}) = \sum_{i=1}^{m} w_i \Phi(\vec{x})$$

where m counts the general non-linear and fixed basis functions, described below.

$$\Phi(\vec{x}) = [\phi_1(\vec{x}), \ldots, \phi_m(\vec{x})]^\intercal$$

Our usual goal is to predict $\vec{w}$, our weights. Michael Tipping's paper provides a Bayesian approach for estimating $\vec{w}$ from this linear form.

Since the relevance vector machine is based on the support vector machine, we apply this method to

$$y(\vec{x},\vec{w}) = \sum_{i=1}^{n} w_i \cdot K(\vec{x},\vec{x_i}) + w_0$$

Which shares our same goal (of minimizing critical weights). We now fit $\vec{\alpha}$, which describe the inverse variance associated with each weight.

$$p(\vec{w}|\vec{\mu},\vec{\alpha}) = \prod \mathcal{N}(w_i|\mu,\alpha_i^{-1})$$

In both support vector machines and relevance vector machines, it is completely possible for every point $\{\vec{x}_n\}$ to be represented with a with a weight w_n, which should be interpreted as over-fitting.

In support vector machines, the use of margin/cost parameters dissuade this result. In relevance vector machines, we achieve the same effect by fitting to a Gaussian about these weights.

For a potential relevance vector $\phi(\vec{x_i})$ As α_i becomes large, α_i^{-1} becomes small, resulting in a non-informative associated Gaussian. These relevance vectors are down-selected, along with their associated hyper-parameters.

As this is a classification problem, Once the relevance vectors are identified and their associated weight Gaussian are fit, a logistic sigmoid function $\dot{\sigma}$ is applied to the predicted probability.

$$P(t \equiv 1|\vec{w}) = \prod_{i \in \{t_i \equiv 1\}} \dot{\sigma}(y(\vec{x}_n, \vec{w})) \prod_{i \in \{t_i \not\equiv 1\}} (1 - \dot{\sigma}(y(\vec{x}_n, \vec{w})))$$

2 Dataset

Since I was most interested in classification problems, I selected to analyze UCI wilt dataset. This dataset provides summary statistics of color channels from images of sick and healthy trees. This remote sensing problem is important to solve, as it would significantly reduce the costs of assessing the health of observed trees.

Table 1: wilt_training.csv

class	**GLCM_pan**	**Mean_Green**	**Mean_Red**	**Mean_NIR**	**SD_pan**
n	132.2993039	203.4074074	90	495.4074074	36.94377146
n	124.0184758	170.9272727	69.07272727	217.8363636	13.33725287
w	122.7978339	202.3428571	132.4857143	322.2857143	15.8353776
w	113.6561798	213.1071429	129.7857143	480.8571429	29.23776563
w	147.7711268	220.1111111	123.7222222	479.1666667	23.33703674
n	112.8245614	236.92	108.04	705.84	22.67065063
w	134.4980916	210.2121212	116.9090909	594.8484848	27.93768482
w	132.4047758	201.3055556	125.4074074	373.8055556	23.22029498
⋮					

The UCI wilt dataset ships with two files, separating test and training information. The training set has 74 `w`ilting trees and 4265 `n`ot. The test set has 187 `w`ilting trees and 313 `n`ot. The dataset is well described, and has no missing values.

3 Implementation Details

My code was written to most closely resemble the implementation described in Michael Tipping's paper, using the same variable names at all steps (for clarity and knowledge transfer).

To encourage eventual adoption into the `julia` base, I use some standard libraries as resources. Firstly, I use `MLBase` for encoding/decoding of target classes in a standardized way. Secondly, I use `MLKernels`[1] as my library of kernels used in machine learning algorithms.

[1]depends on `julia 0.4.7`

The optimization uses iterative reweighted least squares, first solving for $\vec{\mu}$ and Σ for $\vec{w}$ with fixed $\vec{\alpha}$ (by fitting the posterior probability), then using those values to update $\vec{\alpha}$ and repeating, until $\vec{\alpha}$ converges.

3.1 Coding standards

The when solving for $\vec{\mu}$ and $\Sigma_{M,M}$ is performed using Newtons Method. All operations were broken out into simple segmented functions to simplify debugging and increase readability (exampled below).

```julia
function _newton_method(X₀::AbstractVector, F::Function, # Initial, Function
                        ∇::Function, ∇∇::Function)      # Gradient, Hessian
  while true
    # @show F(X₀) # uncomment to watch converge
    X₁ = X₀ - inv(∇∇(X₀)) * ∇(X₀)

    # convergence
    ΔX = maximum(abs(X₁ .- X₀))
    X₀ = X₁[1:end]
    if ΔX < 1e-3
      break
    end
  end
  X₀
end
```

3.2 Use and workflow

To use the system, a `RVMSpec` is defined by parameters

- kernel, selected from the `MLKernels` module
- maximum iterations, to prevent infinite loops
- normalization technique, like standard score transformation
- tolerance, a sufficient delta for α convergence
- threshold, a sufficient cutoff for assumed infinite inverse variance

Next `fit` on their `RVMSpec` and data to generate an `RVMFit`, which can then be used for predictions. The *normalization technique* is applied to the prediction data, exactly as it was in the fitting process (ie. standard score uses the training mean and variance for every feature vector).

4 Evaluation

I report the confusion matrix and classification rate and number of relevance vectors on the UCI wilt dataset. For this evaluation I trained against `linear` and `gaussian` kernels. I also varied the training sample count of 100, 1000, 2300, 4300. Finally, I include training times for each test.

4.1 Language Note

In my original proposal, I was interested in comparing runtimes of `Python` to `julia`. This was abandonded because `Python` has swappable linear algebra libraries that made it very difficult to make this useful analysis. I also found that thresholds were a greater indicator of runtime.

4.2 `linear`

We can clearly see from the table below that the data isn't linearly separable.

Table 2: `linear`

size	**time (s)**	**relevance vectors**	**classification**	**matrix**
100	1.898	6	.367	$\begin{array}{c\|cc} & 1 & 2 \\ \hline 1 & 187 & 312 \\ 2 & 0 & 1 \end{array}$
1000	2.057	1	.626	$\begin{array}{c\|cc} & 1 & 2 \\ \hline 1 & 0 & 0 \\ 2 & 187 & 313 \end{array}$
2300	21.162	1	.626	$\begin{array}{c\|cc} & 1 & 2 \\ \hline 1 & 0 & 0 \\ 2 & 187 & 313 \end{array}$
4300	231.225	1	.626	$\begin{array}{c\|cc} & 1 & 2 \\ \hline 1 & 0 & 0 \\ 2 & 187 & 313 \end{array}$

4.3 `gaussian`

We can see that a `gaussian` kernel performs much better than `linear`.

Table 3: `gaussian`

size	**time (s)**	**relevance vectors**	**classification**	**matrix**
100	.09	5	.822	$\begin{array}{ccc} & 1 & 2 \\ \hline 1 & 181 & 83 \\ 2 & 6 & 230 \end{array}$
1000	4.253	11	.824	$\begin{array}{ccc} & 1 & 2 \\ \hline 1 & 123 & 24 \\ 2 & 64 & 289 \end{array}$
2300	36.53	12	.81	$\begin{array}{ccc} & 1 & 2 \\ \hline 1 & 112 & 20 \\ 2 & 75 & 293 \end{array}$
4300	257.77	15	.784	$\begin{array}{ccc} & 1 & 2 \\ \hline 1 & 99 & 20 \\ 2 & 88 & 293 \end{array}$

4.4 Conclusion

After my analysis I found that all the wilted training samples were at the head of the training file. It would be nice to repeat this with proportional sampling from these two groups.

5 Limitations

There are a few simplifications that were made to complete this project in time. This implementation only supports non-informative priors on the fit Gaussian. I also heavily depend on matrix inversion, which aren't expected to scale. Finally, I only provide solutions to binary classification. In future work I would like to add support for multiple classifications and regression.