A Numerical Comparison of Rule Ensemble Methods and Support Vector Machines

Juan Meza* **Mark Woods†**

December 18, 2009

Abstract

Machine or statistical learning is a growing field that encompasses many scientific problems including estimating parameters from data, identifying risk factors in health studies, image recognition, and finding clusters within datasets, to name just a few examples. Statistical learning can be described as "learning from data", with the goal of making a prediction of some outcome of interest. This prediction is usually made on the basis of a computer model that is built using data where the outcomes and a set of features have been previously matched. The computer model is called a *learner*, hence the name machine learning. In this paper, we present two such algorithms, a support vector machine method and a rule ensemble method. We compared their predictive power on three supernova type 1a data sets provided by the Nearby Supernova Factory and found that while both methods give accuracies of approximately 95%, the rule ensemble method gives much lower false negative rates.

1 Introduction

Machine or statistical learning is a growing field that encompasses many scientific problems including estimating parameters from data, identifying risk factors in health studies, image recognition, and finding clusters within datasets, to name just a few examples. In the recent book on statistical learning by Hastie, Tibshirani, and Friedman [9], they describe this field as "learning from data", with the goal of making a prediction of some outcome of interest. This prediction is usually made on the basis of a computer model that is built using data where the outcomes and a set of features have been previously matched. The computer model is called a *learner*, hence the name machine learning. In this paper, we present two such algorithms and compare their predictive power on three supernova type 1a data sets provided by the Nearby Supernova Factory [1].

The general form of a machine learning problem can be described as follows: given a set of training data $\{\mathbf{x}_i, y_i\}_1^N$ where each $\mathbf{x}_i \in \mathbb{R}^n$ is an observation with a set of n attributes, each y_i is the ith observation's response, and N is the number of observations, find a mapping that will predict the response y. That is, the goal is to produce a function $F(\mathbf{x})$ that minimizes the risk over all future predictions defined as:

$$R(F(\mathbf{x})) = E_{\mathbf{x}y} L(y, F(\mathbf{x})), \tag{1}$$

*Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Mail Stop 50B-4230, Berkeley, CA 94720, JCMeza@lbl.gov; Supported in part by the Director, Office for Advanced Scientific Computing Research, Division of Computational Science Research and Partnerships(SciDAC) of the U.S. Department of Energy, under contract number DE-AC02-05CH11231.

†Lawrence Berkeley National Laboratory, MWoods@lbl.gov

where $L(y, F(\mathbf{x}))$ is the loss associated with predicting a value $\hat{y} = F(\mathbf{x})$ when the actual value is y [8]. Thus, we wish to find the target function

$$F^*(\mathbf{x}) = \arg\min_{F(\mathbf{x})} E_{\mathbf{x}y} L(y, F(\mathbf{x})). \tag{2}$$

For a binary classification problem, the responses are usually denoted by $y_i \in \{-1, 1\}$, where y_i is the ith observation's class. Predictions are denoted by $\hat{y} = \text{sign}[F(\mathbf{x})]$.

The main goal of this report is to implement the rule based learning ensemble method proposed by Friedman and Popescu [8] and to compare its accuracy against another well known technique known as a support vector machine (SVM). Section 2 gives some background information on support vector machines and rule ensemble methods. In section 3, we describe the algorithms we used and the software we implemented. Each section is divided into two subsections: one for support vector machines and one for rule ensembles. At the end of this paper, numerical results are presented and compared.

2 Background

Before describing our implementation, we first present some background information on classification and support vector machines. Here we follow the derivation presented in the book on statistical learning by Hastie, Tibshirani, and Friedman [9]. More details on support vector machines can also be found in the literature, see for example, [4, 5].

2.1 Support Vector Machines

A support vector machine (SVM) is a technique for classifying data, that attempts to find an optimal separation between two classes [9] from a given labeled "training" data set, through the use of a loss function. One such example is the hinge loss function defined by:

$$L(y, F(\mathbf{x})) = \max[0, 1 - yF(\mathbf{x})]. \tag{3}$$

Once an SVM has been trained on the labeled data, it can then be used to predict the class of specific observations from a different unlabeled data set [7].

Although the training data can be mapped in an n-dimensional space, separating the classes can sometimes be difficult to parameterize, as shown in Fig. 1. A standard trick is to map the training data into a higher dimensional space where separation may be simpler. As shown in Fig. 2, separation can be achieved through the use of a hyperplane defined as

$$F(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + \beta, \tag{4}$$

where $\mathbf{w} \in \mathbb{R}^q$ $(q \geq n)$ is a normal vector to the hyperplane, $\phi(\mathbf{x}) \in \mathbb{R}^q$ is a function that maps the predictor variables to a higher dimensional space, and $\beta \in \mathbb{R}^1$ is an offset [2].

Figure 2 shows the ideal case where all observations can be separated into their respective classes. Let d_+ and d_- be the perpendicular distance from the separating hyperplane to the closest positive and negative observation, respectively. We can then define the *margin* of the hyperplane to be $d_+ + d_-$, where $d_+ = d_-$ (the hyperplane is equidistant from both points). The optimal separating hyperplane is the hyperplane with the largest margin [4]. The hyperplane, then, has the constraints

$$\mathbf{w}^T \phi(\mathbf{x}_i) + \beta \geq +1 \quad \text{if } y_i = +1, \tag{5}$$

$$\mathbf{w}^T \phi(\mathbf{x}_i) + \beta \leq -1 \quad \text{if } y_i = -1, \tag{6}$$

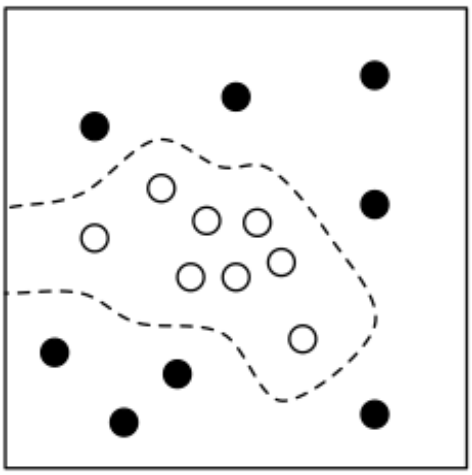

Figure 1: An example of how a good separation can be difficult to parameterize in the given n-dimensional space [2].

for $i = 1, 2, \ldots, N$. More succinctly, this can be expressed as

$$y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + \beta) - 1 \geq 0 \quad \forall i. \tag{7}$$

Notice that all the points that lie exactly on the hyperplane satisfy $\mathbf{w}^T\phi(\mathbf{x}) + \beta = 0$. Thus, the perpendicular distance to the origin is $|\beta|/||\mathbf{w}||$. Now, take a positive point on the margin (5) so that $\mathbf{w}^T\phi(\mathbf{x}_i) + \beta = +1$. The perpendicular distance from this point to the origin is $|1 - \beta|/||\mathbf{w}||$. Therefore, the distance between the hyperplane and the positive margin is $1/||\mathbf{w}||$. A similar argument can be made for the negative margin [4], making the total width of the margin $2/||\mathbf{w}||$. The optimal separating hyperplane is then defined to be the hyperplane that maximizes the margin, or equivalently, the hyperplane that minimizes $||\mathbf{w}||$. This can be formulated as

$$\min \frac{1}{2}||\mathbf{w}||^2 \quad \text{subject to} \quad y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + \beta) \geq 1 \quad \forall i. \tag{8}$$

This formulation works well for the case when the observations are separable. An alternative approach is required however when the classes have some overlap. One way to do this is to modify the constraints (5)-(6), to allow some observations to lie on the wrong side of the hyperplane. One technique is to define positive slack variables ξ_i, $i = 1, 2, \ldots, N$ and introduce them into equations (5)-(6), resulting in:

$$\mathbf{w}^T\phi(\mathbf{x}_i) + \beta \geq +1 - \xi_i \quad \text{if } y_i = +1, \tag{9}$$

$$\mathbf{w}^T\phi(\mathbf{x}_i) + \beta \leq -1 - \xi_i \quad \text{if } y_i = -1, \tag{10}$$

$$\xi_i \geq 0 \quad \forall i. \tag{11}$$

Now, an observation is on the wrong side of the hyperplane if its corresponding ξ_i is greater than 1 [4]. Hence, the objective function can be re-stated as

$$\min \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}\xi_i \quad \text{subject to} \begin{cases} y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + \beta) \geq (1 - \xi_i) & \forall i \\ \xi_i \geq 0,\ \sum_{i=1}^{N}\xi \leq 0 & \forall i \end{cases}, \tag{12}$$

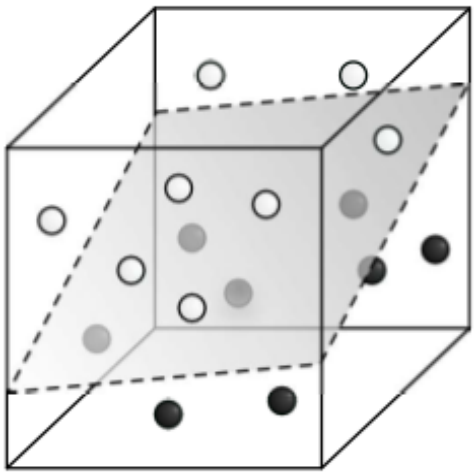

Figure 2: Mapping the training data in a higher dimensional space can lead to a simpler separation [2].

where $C > 0$ is a cost parameter specified by the user. A larger value of C penalizes the error more than a lower value of C [4].

For computational reasons, it is more convenient to express this problem in terms of Lagrange multipliers. Let us first define the Lagrangian primal function by

$$L_P = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(\mathbf{w}^T\phi(\mathbf{x}_i)+\beta) - (1-\xi_i)] - \sum_{i=1}^{N}\mu_i\xi_i, \tag{13}$$

where each α_i and μ_i is a positive Lagrange multiplier. Our goal is to minimize the primal function (13) with respect to $\mathbf{w}$, β, and ξ_i for $i = 1, 2, \ldots, N$. The Karush-Kuhn-Tucker optimality conditions can be written as:

$$\mathbf{w} - \sum_{i=1}^{N}\alpha_i y_i \phi(\mathbf{x}_i) = \mathbf{0}, \tag{14}$$

$$\sum_{i=1}^{N}\alpha_i y_i = 0, \tag{15}$$

$$C - \alpha_i - \mu_i = 0, \tag{16}$$

$$y_i(\mathbf{w}^T\phi(\mathbf{x}_i)+\beta) - (1-\xi_i) \geq 0, \tag{17}$$

$$\xi_i \geq 0, \tag{18}$$

$$\alpha_i \geq 0, \tag{19}$$

$$\mu_i \geq 0, \tag{20}$$

$$\alpha_i[y_i(\mathbf{w}^T\phi(\mathbf{x}_i)+\beta) - (1-\xi_i)] = 0, \tag{21}$$

$$\mu_i\xi_i = 0. \tag{22}$$

Substituting (14)-(16) into the primal function (13), we obtain the Lagrangian dual function

$$L_D = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\alpha_i\alpha_{i'}y_i y_{i'}\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_{i'}). \tag{23}$$

Details of this substitution are shown in Appendix A.

The dual function is maximized subject to $0 \leq \alpha_i \leq C$ for $i = 1, 2, \ldots, N$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$. All observations i where $\alpha_i \neq 0$ are called support vectors [4]. We can see from (14) that the hyperplane normal vector $\mathbf{w}$ can be represented in terms of only the support vectors. The solution can then be given by

$$\mathbf{w} = \sum_{i=1}^{N_S} \alpha_i y_i \phi(\mathbf{x}_i), \tag{24}$$

where N_S is the number of support vectors.

If we solve (16) for μ_i and substitute the result into (22), we can see that $\xi_i = 0$ if $\alpha_i < C$. Therefore, to find the offset β, we can take any observation i for which $0 < \alpha_i < C$ and solve (21) for β. Geometrically, this is a support vector not on its own margin. For numerical stability, one usually takes the mean value of β resulting from all such observations [4].

To implement this algorithm, we are now left with the interesting question of how to choose the mapping $\phi(\mathbf{x})$. Looking at the dual function (23), we see that the mapping only appears as a dot product. Thus, we can use a kernel function, which defines an inner product in a higher dimensional space $K(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u})^T \phi(\mathbf{v})$ [2]. There are several choices for a kernel function that can be found in the literature, including the radial basis function kernel and the polynomial kernel.

The final classification rule can now be computed as

$$\hat{y} = \text{sign}\left[\mathbf{w}^T \phi(\mathbf{x}) + \beta\right]. \tag{25}$$

2.2 Rule Ensembles

While support vector machines have many advantages, it can sometimes be difficult to choose an appropriate kernel that works well over a wide range of data. Another approach is to use simpler models and then take an average from the results produced by the individual models. This approach, called an ensemble method, can be used to describe any method that combines individual base learners into a single model $F(\mathbf{x})$ [8]. Here, the loss function used to determine the final model is the squared error ramp loss

$$L(y, F(\mathbf{x})) = [y - H(F(\mathbf{x}))]^2, \tag{26}$$

where

$$H(F(\mathbf{x})) = \max[-1, \min(1, F(\mathbf{x}))]. \tag{27}$$

When each of the base learners is given by a rule $r_m(\mathbf{x})$, the resulting method is known as a rule ensemble method. Each rule can be viewed as a parameterized function $r_m(\mathbf{x}) = r(\mathbf{x}, s_m)$ where each s_m characterizes the corresponding rule.

One way to generate rules is to build a decision tree and let each interior and terminal node of that tree define a rule [8]. Each rule would take the form of the conjunctive rule

$$r_m(\mathbf{x}) = \prod_{j=1}^{n} I(x_j \in s_{jm}), \tag{28}$$

where the parameters s_{jm} define a subset of values for the corresponding x_j and $I(\cdot)$ is an indicator function whose value is in $\{0, 1\}$. To help illustrate this idea, consider the following example depicted by Fig. 3. Each rule can be written as:

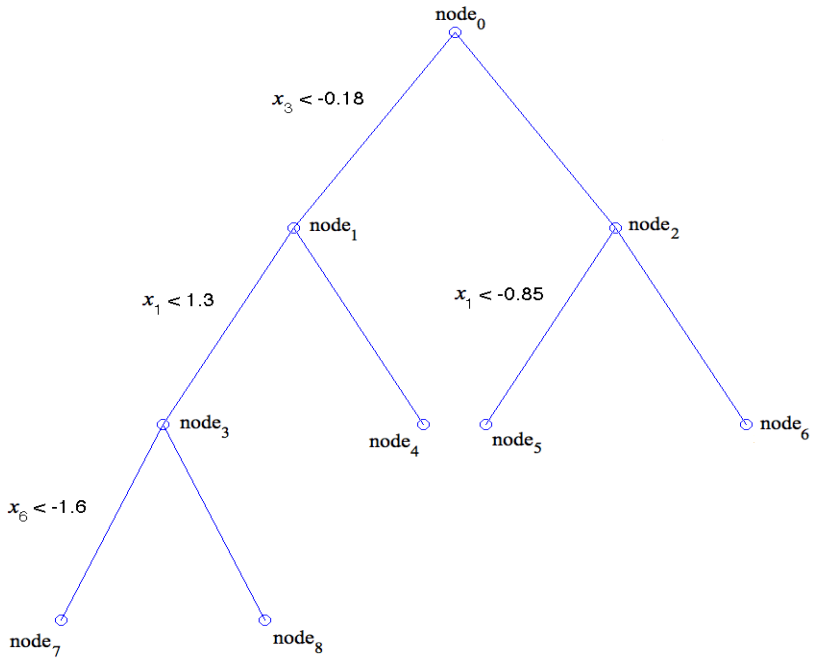

Figure 3: A decision tree.

$$
\begin{aligned}
r_1(\mathbf{x}) &= I(x_3 < -0.18), \\
r_2(\mathbf{x}) &= I(x_3 \geq -0.18), \\
r_3(\mathbf{x}) &= I(x_3 < -0.18) \cdot I(x_1 < 1.3), \\
r_4(\mathbf{x}) &= I(x_3 < -0.18) \cdot I(x_1 \geq 1.3), \\
r_5(\mathbf{x}) &= I(x_3 \geq -0.18) \cdot I(x_1 < -0.85), \\
r_6(\mathbf{x}) &= I(x_3 \geq -0.18) \cdot I(x_1 \geq -0.85), \\
r_7(\mathbf{x}) &= I(x_3 < -0.18) \cdot I(x_1 < 1.3) \cdot I(x_6 < -1.6), \\
r_8(\mathbf{x}) &= I(x_3 < -0.18) \cdot I(x_1 < 1.3) \cdot I(x_6 \geq -1.6).
\end{aligned}
$$

Thus, each rule takes on values $r_m(\mathrm{x}) \in \{0, 1\}$. Notice that there is no rule associated with the root node. Also notice that for a single (binary) tree, the total number of rules is $2(t-1)$ where t is the number of terminal nodes.

The prescription for generating a rule ensemble is described in Algorithm 1.

At each iteration, M trees are generated. As stated above, each tree contains $2(t_m - 1)$ rules, where t_m is the number of terminal nodes for the mth tree. The memory function $f_m(\mathbf{x})$ contains information about each previous rule. This is controlled by the value of the shrinkage parameter $0 \leq \upsilon \leq 1$, where $\upsilon = 0$ generates each rule without regard to the previous rules, $\upsilon = 1$ maximizes the effect of each previous rule, and $0 < \upsilon < 1$ is in between [8].

Line 3 of Algorithm 1 is an expression asking for the best set of rules in combination with the previously generated rules, where $S_m(\eta)$ is a subset of size $\eta < N$ randomly drawn without replace-

Algorithm 1 Rule Ensemble Generation [8]

1: $f_0(\mathbf{x}) = \arg\min_c \sum_{i=1}^{N} L(y_i, c)$
2: **for** $m = 1$ **to** M **do**
3: $\quad \mathbf{s}_m = \arg\min_{\mathbf{s}} \sum_{i \in S_m(\eta)} L(y_i, f_{m-1}(\mathbf{x}_i) + r(\mathbf{x}, \mathbf{s}))$
4: $\quad \mathbf{r}_m(\mathbf{x}) = r(\mathbf{x}, \mathbf{s}_m)$
5: $\quad f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \upsilon \sum_{k=1}^{2(t_m-1)} r_{km}(\mathbf{x})$
6: **end for**
7: Set ensemble $= \{\mathbf{r}_m(\mathbf{x})\}_1^M$

ment from the training data. This represents an optimization problem for which fast algorithms are difficult to obtain for the loss function given by (26)-(27).

One possible solution to this as suggested by [9] is to use a steepest descent method for tree induction. That is, calculate the negative gradient evaluated at $f_{m-1}(\mathbf{x})$ and fit a regression tree to the result. These are called the pseudo residuals p_m and are defined by:

$$p_m = \left\{ - \left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} \right]_{f(\mathbf{x}_i) = f_{m-1}(\mathbf{x}_i)} \right\}_{i=1}^{N}$$

After the rule ensemble generation is complete, the total number of rules will be

$$K = \sum_{m=1}^{M} 2(t_m - 1). \tag{29}$$

From Fig. 3 and (29), we can see that the number of rules is directly related to the size of each tree. Larger trees can capture higher order interaction effects, but may overfit the data. Smaller trees only capture low order interaction effects and may underfit the data. Since each target function (2) is different, it is difficult to know *a priori* how large to grow each tree.

A solution to this problem is to let the number of terminal nodes in each tree be randomly drawn from some distribution (for example an exponential distribution). Let $\bar{L}$ be the average number of terminal nodes for all M trees. Then

$$\Pr(\delta) = \frac{1}{\bar{L} - 2} \exp\left(\frac{-\delta}{\bar{L} - 2} \right), \tag{30}$$

so that

$$t_m = 2 + \lfloor \delta \rfloor. \tag{31}$$

Letting the number of terminal nodes for each tree be a random number from an exponential distribution has the effect of including high and low order interaction effects in the model. Most trees will have $t_m < \bar{L}$ terminal nodes, but some trees will have $t_m \simeq \bar{L}$ terminal nodes. A few trees will have $t_m \gg \bar{L}$ terminal nodes.

The next step in this procedure is to determine an approach for building a regression tree. The algorithm is presented below. Note that this algorithm is for the general case of building a regression tree. For the actual implementation, replace each y with p_m as stated above.

In Algorithm 2, n_s is a subset of attributes $n_s \subseteq n$ randomly selected for each split. Also, $N(t)$ is the number of observations in the parent node (the node being split) and $N(t_L)$ and $N(t_R)$ are the respective number of observations in the left and right child nodes. N in lines 14 and 15 is still the total number of observations in the entire data set.

Algorithm 2 Build a Regression Tree [3]

Find a terminal node that can be split
for $j = 1$ **to** n_s **do**
  for $i = 1$ **to** $N(t)$ **do**
    Determine each split $\mathcal{S} = \{x_{ij} + (x_{(i+1)j} - x_{ij})/2\}_{i=1}^{N(t)-1}$
  end for
  for all splits $\mathcal{S}$ **do**
    if $x_{ij} < \mathcal{S}$ **then**
      Go to the left child node t_L
    else
      Go to the right child node t_R
    end if
    $F_L(\mathbf{x}) = 1/(N(t_L)) \sum_{i \in t_L} y_{i'}$
    $F_R(\mathbf{x}) = 1/(N(t_R)) \sum_{i \in t_R} y_{i'}$
    $R(t_L) = 1/N \sum_{i \in t_L} [y_i - F_L(\mathbf{x})]^2$
    $R(t_R) = 1/N \sum_{i \in t_R} [y_i - F_R(\mathbf{x})]^2$
    $\tilde{R} = R(t_L) + R(t_R)$
  end for
end for
Choose the split S and attribute j that yields the minimum $\tilde{R}$

Algorithm 2 is repeated until there are t_m terminal nodes (31) in the tree or there is a specified number of observations in each terminal node, whichever comes first. For instance, the user may not want to split a node with fewer than five observations in it. In this case, any node with five or fewer observations would not be split.

Now, we need a way to combine the rules together to obtain the final model $F(\mathbf{x})$. One of the most popular ways to combine base learners is through a linear combination

$$F(\mathbf{x}) = a_0 + \sum_{k=1}^{K} a_k r_k(\mathbf{x}), \tag{32}$$

where each a_k is a weighting coefficient for its corresponding rule $r_k(\mathbf{x})$ and a_0 is an offset [7]. To find each coefficient $\{a_k\}_1^K$ and the offset a_0, one could solve

$$\{\hat{a}_k\}_0^K = \arg \min_{\{a_k\}_0^K} \frac{1}{N} \sum_{i=1}^{N} L\left(y_i, a_0 + \sum_{k=1}^{K} a_k r_k(\mathbf{x}_i)\right). \tag{33}$$

However, it is known [7] that this often provides a poor estimate to the optimal values

$$\{a_k^*\}_0^K = \arg \min_{\{a_k\}_0^K} E_{\mathbf{x}y} L\left(y, a_0 + \sum_{k=1}^{K} a_k r_k(\mathbf{x})\right), \tag{34}$$

especially when the number of observations N is not large compared to the number of rules K.

One way to remedy this is to regularize (33) by adding a lasso penalty. This is formulated as

$$\{\hat{a}_k\}_0^K = \arg \min_{\{a_k\}_0^K} \frac{1}{N} \sum_{i=1}^{N} L\left(y_i, a_0 + \sum_{k=1}^{K} a_k r_k(\mathbf{x}_i)\right) + \lambda \sum_{k=1}^{K} |a_k|. \tag{35}$$

The "lasso" has the effect of penalizing larger values of a_k. A larger value of λ produces more regularization, and often times, the majority of the coefficients are set to zero. Of course, (35) represents a more difficult optimization problem and the time required to solve this problem is computationally prohibitive for most problems.

One solution to this is to use a gradient descent approach starting with the point corresponding to $\lambda = \infty$. The algorithm is summarized below.

Algorithm 3 Gradient Directed Regularization [7]

1: $a_0 = \arg\min_a \sum_{i=1}^N L(y_i, a), \{a_k = 0\}_1^K$
2: **while** $||\mathbf{g}(\ell)|| \neq 0$ **do**
3: $F(\mathbf{x}) = a_0 + \sum_{k=1}^K a_k r_k(\mathbf{x})$
4: $\mathbf{g}(\ell) = -\frac{d}{d\mathbf{a}} \frac{1}{N} \sum_{i=1}^N L(y_i, F(\mathbf{x}_i))\Big|_{\mathbf{a}=\mathbf{a}(\ell)}$
5: $k^* = \arg\max_{0 \leq k \leq K} |g_k(\ell)|$
6: $G(\ell) = \{k : |g_k(\ell)| \geq |g_{k^*}(\ell)|\}$
7: $h_k(\ell) = \begin{cases} g_k(\ell) & \text{for } k \in G(\ell) \\ 0 & \text{otherwise} \end{cases}$
8: $\mathbf{a}(\ell + \Delta\ell) = \mathbf{a}(\ell) + \Delta\ell \cdot \mathbf{h}(\ell)$
9: **end while**

The step size value $\Delta\ell$ should be small (usually, $\Delta\ell = 0.01$ is sufficient) The final coefficient values $\{a_k\}_0^K$ are selected so that the risk (1) is minimized on a separate sample not used in Algorithm 3 [7]. To accomplish this, calculate the prediction risk based on an independent test sample of size N_T at each iteration. This can be done with

$$R_T(\ell) = \frac{1}{N_T} \sum_{i=1}^{N_T} L(y_i, F(\mathbf{x}_i)). \tag{36}$$

Once Algorithm 3 terminates, one can pick the coefficient values $\{a_k\}_0^K$ that correspond to the smallest test risk $R_T(\ell)$.

Notice that λ is not used in Algorithm 3. This is because Algorithm 3 gives an approximation of the solution to (35). It has been shown, however, that the two solutions are quite similar [9].

2.3 Computational Details

In order to increase computational efficiency, a few modifications can be applied to Algorithm 3. For the first calculation of the negative gradient (step 4 of Algorithm 3), one can use

$$g_k(\ell) = \tilde{c}(y, r_k(\mathbf{x}), \ell) - \sum_{k'=0}^K a_{k'}(\ell) \tilde{c}(r_k(\mathbf{x}), r_{k'}(\mathbf{x}), \ell), \tag{37}$$

where $\{r_{i0}(\mathbf{x}) = 1\}_1^N$ and $\tilde{c}(\cdot)$ is an approximate covariance matrix with elements

$$\tilde{c}(u, v, \ell) = \frac{1}{N} \sum_{i=1}^N u_i v_i I(|F(\mathbf{x}_i)| < 1). \tag{38}$$

After the first calculation, one can update the negative gradient with

$$g_k(\ell + \Delta\ell) = g_k(\ell) - \Delta\ell \sum_{k' \in G(\ell)} h_{k'}(\ell) \tilde{c}(r_k(\mathbf{x}), r_{k'}(\mathbf{x}), \ell), \tag{39}$$

at step 4 of Algorithm 3.

First, let us define a milestone to be every 100 iterations. At every milestone, one can calculate the indicators

$$\{\rho(\ell) = I(|F(\mathbf{x}_i)| < 1)\}_1^N \tag{40}$$

and

$$z_i(\ell_0, \ell) = \begin{cases} -1 & \text{if } \rho_i(\ell_0) = 1 \ \& \ \rho_i(\ell) = 0 \\ 0 & \text{if } \rho_i(\ell_0) = \rho_i(\ell) \\ +1 & \text{if } \rho_i(\ell_0) = 0 \ \& \ \rho_i(\ell) = 1 \end{cases} \tag{41}$$

to update the approximate covariances

$$\tilde{c}(u, v, \ell) = \tilde{c}(u, v, \ell_0) + \frac{1}{N} \sum_{z_i(\ell_0,\ell)\neq 0} u_i v_i z_i(\ell_0, \ell), \tag{42}$$

where ℓ_0 indexes the previous milestone. Now, one can use the previous milestones's values for the approximate covariances $\tilde{c}$ in the updating step (39). We note that a column of $\tilde{c}$ only needs to be calculated (38) and updated (42) when its corresponding coefficient a_k is nonzero ($a_k \neq 0$) [7].

In addition, instead of calculating the empirical risk (36) at each iteration, one can calculate it only at every milestone in conjunction with an early stopping strategy. Define a threshold parameter $\zeta > 1$, and terminate calculation at the point $\ell = \tilde{\ell}$ for which

$$R_T(\tilde{\ell}) > \zeta \cdot \min_{\ell<\tilde{\ell}} R_T(\ell). \tag{43}$$

Again, use the coefficient values $\{a_k\}_0^K$ that correspond to the smallest $R_T(\ell)$. Usually, $\zeta = 1.1$ is sufficient [7].

Once the calculation is complete, we have everything we need to make a prediction. A prediction is simply $\hat{y} = \text{sign}[F(\mathbf{x})]$ with $F(\mathbf{x})$ given by (32).

It is well known that a decision tree has trouble approximating a linear function [9]. Since each base learner in Algorithm 1 is derived from a tree, the resulting model (32) will also have trouble approximating a linear function. A simple solution is to add linear terms into the model.

However, one first needs to guard against outliers by winsorizing the data[1]. This is accomplished through the function

$$l(x_j) = \min(\psi_j^+, \max(\psi_j^-, x_j)), \tag{44}$$

where ψ_j^- and ψ_j^+ are the τ and $1 - \tau$ quantiles, respectively for each attribute $j = 1, 2, \ldots, n$. Typically, $\tau = 0.025$ is sufficient, but it depends on the data [8].

A common practice is to standardize the data so that each attribute has the same influence. This means that the data $l(x_j)$ should be transformed so that $\text{var}(l(x_j)) = 1$ for $j = 1, 2, \ldots, n$. The transformation is given by

$$\left\{ l(x_j) \leftarrow 0.4 \, \frac{l(x_j)}{\text{std}(l(x_j))} \right\}_1^n, \tag{45}$$

with 0.4 included so that each attribute takes on the average value of the rules $\{r_k(\mathbf{x})\}_1^K$.

We thus obtain the new model

$$F(\mathbf{x}) = a_0 + \sum_{k=1}^{K} a_k r_k(\mathbf{x}) + \sum_{j=1}^{n} b_j l(x_j), \tag{46}$$

[1]Named after Charles P. Winsor for his idea to replace extreme observations with more reasonable ones [6] – http://jeff560.tripod.com/w.html (accessed on September 16, 2009)

using

$$(\{\hat{a}_k\}_0^K, \{\hat{b}_j\}_1^n) = \arg \min_{\{a_k\}_0^K, \{b_j\}_1^n} \frac{1}{N} \sum_{i=1}^{N} L\left(y_i, a_0 + \sum_{k=1}^{K} a_k r_k(\mathbf{x}_i) + \sum_{j=1}^{n} b_j l(x_{ij})\right) + \lambda \left(\sum_{k=1}^{K} |a_k| + \sum_{j=1}^{n} |b_j|\right). \tag{47}$$

This quantity can be calculated as before using Algorithm 3 and the suggested computational improvements (37)–(43). For more information on the rule ensemble procedure, see [7, 8, 9].

3 Experimental Setup

This section describes the procedures used for testing both the support vector machines and the rule ensemble method along with the software used for each.

3.1 SVM Procedure

The SVM software used for testing was LIBSVM written by Chang and Lin [5]. Data sets were provided by the Nearby Supernova Factory [1]. Two different SVMs were tested along with two different kernel functions. The two SVMs were the C-support vector classification (C-SVC) machine and the ν-support vector classification (ν-SVC) machine. The two kernel functions were the radial basis function (RBF) kernel and the polynomial kernel.

The SVMs and the kernel functions all have parameters that can be adjusted. For C-SVC, the adjustable parameter is C, which is the upper bound for α in (23). For ν-SVC, the adjustable parameter is ν, which is an upper bound on the fraction of training errors and a lower bound on the number of support vectors. For the RBF kernel, the adjustable parameter is γ (a constant in the kernel function), and for the polynomial kernel, the adjustable parameters are the polynomial degree and the polynomial offset. For each test, one or two of these were sequentially adjusted to find an optimal accuracy.

Three supernova data sets were used. These data sets all contained 24 attributes. The first two are sets that each contain 5,000 supernovae or other objects of interest (positive examples) and 5,000 other celestial bodies of no interest (negative examples). The third data set contains 20,000 positive examples and 200,000 negative examples. The first two data sets were used to test all the different parameters and combinations of parameters. The third data set was only used to test the final set of parameters and validate the previous tests.

Each test was done by randomly choosing observations for training and testing and then averaging the accuracies and run times over 100 trials. The point selection was done by arbitrarily choosing a ratio of positive examples to negative examples for training $(N_+/N_-)_{\text{train}}$ and a ratio of positive examples to negative examples for predicting $(N_+/N_-)_{\text{predict}}$.

To illustrate, suppose a training ratio of 0.5 was chosen. Then, $(N_+/N_-)_{\text{train}}$ could contain 50 positive examples and 100 negative examples, 150 positive examples and 300 negative examples, etc. One might wonder which one to choose. Testing results showed that it didn't matter as long as the training data was large enough to capture all of the main effects. Five hundred total observations for training worked quite well. If the ratio $(N_+/N_-)_{\text{train}}$ was held constant, then the accuracies were well within one standard deviation of each other. The major difference between two different numbers of total training observations were run times. The run times increased with N^2,

which discouraged choosing an overly large number of training observations. For the testing data, 250 total observations were randomly selected. The main justification for only 250 total testing observations is that each test was performed 100 times and the average was taken. So even if a few tests were bad (a poor selection of random points was chosen), there were many others to make up for it.

In each test, a variety of prediction ratios $(N_+/N_-)_{\text{predict}}$ were selected and compared to each other. It was soon discovered that prediction accuracy vastly increased when $(N_+/N_-)_{\text{predict}}$ decreased (the opposite is true as well). Accuracies were typically above 95% (sometimes above 97%) when $(N_+/N_-)_{\text{predict}}$ was less than 0.1. Unfortunately, it is difficult to say what $(N_+/N_-)_{\text{predict}}$ is for unknown data. The only estimate came from the third data set. This set is called the "validation" data set and has $(N_+/N_-)_{\text{predict}} = 0.1$, so it was assumed that this would be similar to an actual data set. The next step is to then find which set of SVM parameters performs the best for this data set.

According to [10], a reasonable way to begin is to start with the C-SVC machine and select pairs of values for C and γ. They suggest picking $C = 2^{-5}, 2^{-3}, \ldots, 2^{15}$ and $\gamma = 2^{-15}, 2^{-13}, \ldots, 2^3$ and using every combination of these two sets with cross validation to find the best pair with which to train the machine. For this test, five-fold cross validation was used, and the resulting accuracies were plotted on a contour map. The grid was visually searched for the area with the highest accuracies. In the event of equal (or close to equal) accuracies, the area with the lowest run time was used. This produced a rectangle with which to create another set of pairs of C and γ to test. This process was repeated with the resulting grid until a single pair of C and γ was found. This pair was then used to train the machine and testing was performed. The resulting accuracies were plotted against $(N_+/N_-)_{\text{predict}}$ to test the robustness of this pair.

The parameters C and γ were also adjusted independently of each other. One parameter was held constant at its default value while the other was altered. The default values are $C = 1$ and $\gamma = 1/n$ (recall that n is the number of attributes). This was done in order to see the individual effects and if a more robust pair could be found.

Once a sufficient number of pairs of C and γ were tested, attention was directed toward ν-SVC and the parameter ν was manipulated. This turns out to be much simpler than the aforementioned process. Both C and γ are infinite in range of values, but $\nu \in (0, 1]$. Thus, a more exhaustive search is possible. The results of these tests were compared with the results of previous tests.

After this, both support vector machines were tested by varying the kernel function. For the polynomial kernel, its intercept and degree could also be varied. All of these were tested and the accompanying accuracies and run times were recorded.

This process was carried out to determine whether a robust set of parameters could be found. It was never assumed that this would be the best set of parameters for all data sets, just these particular supernova data sets.

After the final set of parameters was found, they were tested on the entire validation data set. Again, this set consisted of 20,000 positive examples and 200,000 negative examples. This would serve to validate the previous tests.

3.2 Rule Ensemble Procedure

Our version of the rule ensemble procedure as described in Section 2.2 was written in Matlab version 7.4.0.287 (R2007a). The only exception was the code for building regression trees. That code was developed by W. L. Martinez and A. R. Martinez [11] and modified to work with our code.

Five parameters were adjusted to find the optimal values for the supernova data sets. These parameters were adjusted one at a time to find the optimal accuracy for each one. The adjusted

parameters were the shrinkage parameter υ, the total number of trees in the ensemble M, the size of the subset of attributes used to determine each split n_s, the average number of terminal nodes in each tree $\bar{L}$, and the sample size used to build each tree η.

Each parameter adjustment used 1,000 total observations selected at random for training and 1,000 different observations selected at random for testing. This process was repeated 25 times and the average accuracy was taken. The reason for only averaging over 25 trials (as opposed to 100 used for the SVMs) was due to the increased run times. Each run would take between one and ten minutes (it varied due to the nature of the algorithm).

As before, after the final set of parameters was found for the training data set, they were tested on the entire validation data set.

4 Numerical Results

All tests were run on an iMac with a 2.93 GHz Intel Core 2 Duo processor, and a MacBook Pro with a 2.4 GHz Intel Core 2 Duo processor.

4.1 SVM Results

Throughout all of the tests, the kernel function $K(\mathbf{u}, \mathbf{v})$ used for the SVMs had little effect on performance. There was never any observed difference between accuracies and run times for the RBF kernel and the polynomial kernel. For this reason, the polynomial kernel was arbitrarily chosen for the rest of the tests.

The polynomial kernel is of the form

$$K(\mathbf{u}, \mathbf{v}) = \left(\gamma \mathbf{u}^T \mathbf{v} + r_0\right)^d, \tag{48}$$

where d is the degree and r_0 is the intercept. The tested degrees were $d = 2, 3, 4$. The tested intercepts were $r_0 \in [-100, 100]$. For both support vector machines, the degree did not change the accuracies or run times. The default value for the degree is 3, so it was left at this value for the remainder of the tests.

The intercept had a substantial effect on accuracy for C-SVC. A typical plot of accuracy vs. r_0 is shown in Fig. 4. For each value of $(N_+/N_-)_{\text{predict}}$, the corresponding plot showed that the maximum accuracy was achieved when r_0 was zero. Unfortunately, the default value for r_0 is zero, so no increase in accuracy or robustness was achieved. However, from this test, it was known to keep r_0 at zero for the remainder of the tests.

The grid search for a C and γ pair as described in the previous section was the only test where the results were not the average of 100 trials. Rather, these were averaged over 25 trials. The reason for this was because using cross validation greatly increased the run times. For some pairs of C and γ, the run times would be over a minute. Each test would contain somewhere between 110 and 225 pairs of C and γ, so running this 100 times was too costly. However, the vast majority of run times were on the order of 10^{-1} seconds. The run times that were over a minute resulted from the largest values of C and γ (recall that these were around $C = 2^{15}$ and $\gamma = 2^3$). The final pair chosen from this was $C = 0.22$ and $\gamma = 0.58$.

To see what the individual effects were, C and γ were adjusted independently in two separate tests. Varying γ produced a surprising result. Smaller, nonzero magnitudes yielded higher accuracies and run times. There was no downside to choosing a small value for γ. However, no change was observed for γ smaller than the default value $(1/n)$. Thus, it was decided to leave γ at its default value.

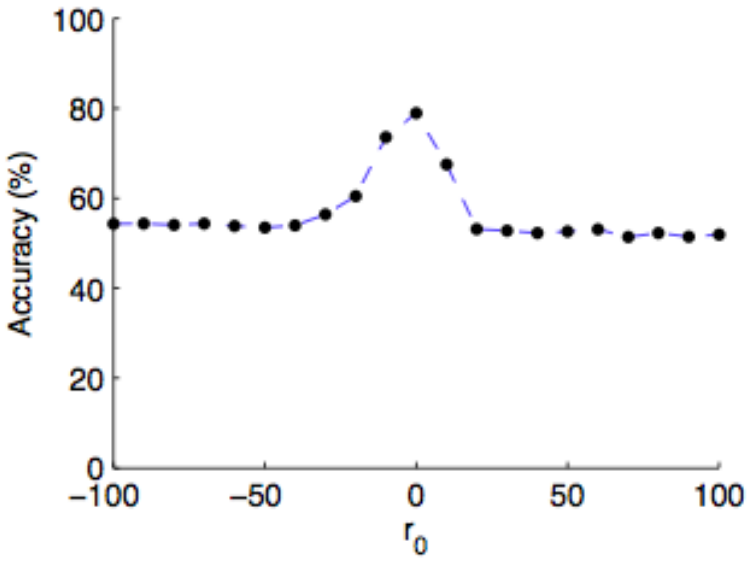

Figure 4: Plot of accuracy vs. r_0 for C-SVC with $(N_+/N_-)_{\text{predict}} = 1$.

Adjusting the parameter C showed that the maximum accuracy was attained usually when $C \in [14, 27]$. For higher values of $(N_+/N_-)_{\text{predict}}$, this was typically the case, but when $(N_+/N_-)_{\text{predict}} > 1$, the accuracy could be as low as 74%. For the lowest values of $(N_+/N_-)_{\text{predict}}$, $C \in [14, 27]$ produced accuracies above 95%.

Interestingly enough, ν-SVC was not affected by kernel parameter changes. Altering r_0 and γ did not produce any change in accuracy or run time. For each $(N_+/N_-)_{\text{predict}}$, this held true. So, it must be concluded that ν-SVC is not sensitive to kernel parameter changes.

The parameter that did affect ν-SVC was ν. Altering ν produced some promising results. The highest accuracies of all of the tests were attained when $\nu = 0.8$ and $(N_+/N_-)_{\text{predict}} \lesssim 0.15$. For the smallest values of $(N_+/N_-)_{\text{predict}}$, the accuracies could be greater than 97%. Unfortunately, for higher values of $(N_+/N_-)_{\text{predict}}$, the accuracies fell.

The interesting choice for ν is 0.2. This produced the most robust result of any of the tests. For all values of $(N_+/N_-)_{\text{predict}}$, the accuracy stayed between 89% and 91%. These accuracies may not be as high as one might like, but knowing the accuracy in advance may be useful.

The results for the four best SVMs are shown in Fig. 5. Clearly, the worst SVM of these four corresponds to $C = 0.22$ and $\gamma = 0.58$ (the result of the grid search). Nowhere does it have the greatest accuracy and its accuracy drops off quicker than the others.

The other three SVMs all have the greatest accuracy for a portion of $(N_+/N_-)_{\text{predict}}$ values. As mentioned above, the ν-SVC machine with $\nu = 0.8$ gives the greatest overall accuracy. If one has strong reasons to believe that $(N_+/N_-)_{\text{predict}}$ is less than 0.15, this should be used over the other SVMs. If one has no indication of what $(N_+/N_-)_{\text{predict}}$ is, then one should consider using the ν-SVC machine with $\nu = 0.2$. In doing so, the analyst would know that the accuracy will be approximately 90%.

The core issue is finding a way to estimate the value of $(N_+/N_-)_{\text{predict}}$. It is unlikely that this ratio will be very high because a type 1a supernova occurs only about once or twice every millennium in a given galaxy [2].

The validation test set that was provided contains 20,000 positive examples and 200,000 negative

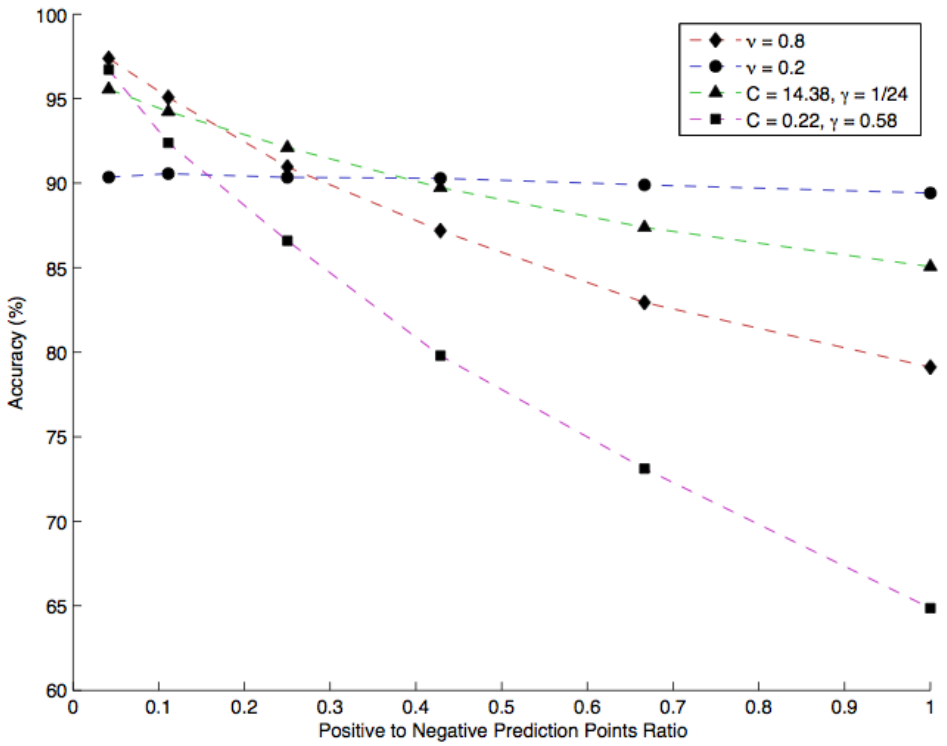

Figure 5: Accuracy vs. $(N_+/N_-)_{\text{predict}}$ with a polynomial kernel of degree 3 (each point is the average of 100 trials).

examples, yielding $(N_+/N_-)_{\text{predict}} = 0.1$. Thus, it was decided to test this data set with the ν-SVC machine with $\nu = 0.8$. Based on the previous tests, it was expected that the accuracy would be close to 95%. In fact, the accuracy was computed to be 95.4950%. The false positive rate was 0.916%, but the false negative rate was 40.395%. The other SVMs all had worse false negative rates and worse accuracies. This produces a stellar false positive rate, but a poor false negative rate. The decision boundary (25) is predicting almost all of the observations to be negatives (not a supernova type 1a).

It was later suggested that not every attribute in the data set might contribute to creating a good model for prediction. Out of the 24 given attributes from the data set, 19 were suggested that should improve performance [12]. In other words, 5 attributes were deleted from the data sets for training and predicting. This conjecture was tested on the validation test with the ν-SVC using $\nu = 0.8$ (the same conditions that yielded 95.4950% above). The simulations indicated that the accuracy decreased to 89.7845%, while the false positive rate increased to 1.3735% and the false negative rate increased to 98.635%. Clearly, using all attributes in the data set yielded better results for these data sets.

4.2 Rule Ensemble Results

The shrinkage parameter υ was varied between 0 and 1 to determine the optimal value. It had a significant effect on the accuracy of the resulting model. For the larger values of υ, accuracies were as low as 79%. For the lower, nonzero values of υ, accuracies were as high as 93%. In [8], it is stated that $\upsilon = 0.01$ performs best for most situations. Thus, it was decided to set υ to 0.01 for

the remainder of the tests.

The average number of terminal nodes $\bar{L}$ also had an effect on the model accuracy. However, it did not yield any improvements to accuracy. Ensembles using stumps for each tree lowered the overall accuracy to about 90%. The accuracies were best when six terminal nodes were used as averaged over all trees.

The other parameters did not affect the model accuracy much. However, they did affect the total run times. Taking into account the run times and marginal gains in accuracy ($\sim 1\%$), the final parameter values chosen were $M = 200$ trees, $n_s = 12$ sampled attributes at each split, and $\eta = 250$ observations for each tree.

For the rule ensemble program, the same validation test set was used with $\upsilon = 0.01$, $M = 200$, $n_s = 12$, $\bar{L} = 6$, and $\eta = 250$ as stated in section 4.2. Due to the inherent randomness in the algorithm, a different result was obtained each time. Each test required about fifteen minutes. The results are given in Table 1.

Table 1: Rule Ensemble Results

	Accuracy	False Positive Rate	False Negative Rate
Trial 1	94.44%	5.16%	9.58%
Trial 2	94.60%	4.71%	12.28%
Trial 3	92.74%	7.32%	6.64%
Trial 4	94.51%	5.10%	9.40%
Trial 5	87.75%	12.99%	4.88%

5 Conclusions and Future Work

Although the accuracies from the rule ensemble method are slightly lower than those obtained from the SVMs, false positive and false negative rates are more balanced. That is, they are both reasonably low. If one had to choose between using SVMs and the rule ensemble models, the rule ensemble models would be the better choice. They can give accuracies in the mid 90% range and give much better false negative rates. The mere fact that the rule ensemble program gives varying results poses a dilemma. How is one to know *a priori* which model is better? One solution to this problem could be to take out the uncontrolled randomness in the program and replace it with a controlled randomness. This should cause the program to give the same results every time. Also, for future work, it may be beneficial to combine the two methods. That is, create a program that generates an ensemble of SVMs. Post processing (47) could be performed on the resulting ensemble in the same way as with the rules and linear terms.

Acknowledgements

Helpful discussions with and suggestions from David Bailey and Daniela Ushizima are greatly appreciated.

Appendix A – Primal to Dual

We want to obtain the dual function

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_{i'})$$

by substituting

$$\begin{aligned} \mathbf{0} &= \mathbf{w} - \sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x}_i), \\ 0 &= \sum_{i=1}^{N} \alpha_i y_i, \\ 0 &= C - \alpha_i - \mu_i \end{aligned}$$

into the primal function

$$L_P = \frac{1}{2}||\mathbf{w}||^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i [y_i(\mathbf{w}^T \phi(\mathbf{x}_i) + \beta) - (1 - \xi_i)] - \sum_{i=1}^{N} \mu_i \xi_i.$$

To begin, let us focus on the first term in the primal function.

$$\begin{aligned} A_1 &= \frac{1}{2}||\mathbf{w}||^2 \\ A_1 &= \frac{1}{2} \sum_{j=1}^{q} w_j^2 \\ \mathbf{w} &= \sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x}_i) \\ w_j &= \sum_{i=1}^{N} \alpha_i y_i \phi(x_{ij}) \\ A_1 &= \frac{1}{2} \sum_{j=1}^{q} \left(\sum_{i=1}^{N} \alpha_i y_i \phi(x_{ij}) \right)^2 \\ A_1 &= \frac{1}{2} \sum_{j=1}^{q} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \phi(x_{ij}) \phi(x_{i'j}) \\ A_1 &= \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_{i'}) \end{aligned}$$

Next, we move on to the second fourth terms in the primal function.

$$
\begin{aligned}
A_2 &= C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\mu_i\xi_i \\
A_2 &= \sum_{i=1}^{N}(C\xi_i - \mu_i\xi_i) \\
C &= \alpha_i + \mu_i \\
A_2 &= \sum_{i=1}^{N}[(\alpha_i + \mu_i)\xi_i - \mu_i\xi_i] \\
A_2 &= \sum_{i=1}^{N}(\alpha_i\xi_i + \mu_i\xi_i - \mu_i\xi_i) \\
A_2 &= \sum_{i=1}^{N}\alpha_i\xi_i
\end{aligned}
$$

Finally, we have the third term of the primal function.

$$
\begin{aligned}
A_3 &= -\sum_{i=1}^{N}\alpha_i[y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + \beta) - (1 - \xi_i)] \\
A_3 &= -\sum_{i=1}^{N}(\alpha_i y_i \mathbf{w}^T\phi(\mathbf{x}_i) + \alpha_i y_i\beta - \alpha_i + \alpha_i\xi_i) \\
A_3 &= -\left(\sum_{i=1}^{N}\alpha_i y_i \mathbf{w}^T\phi(\mathbf{x}_i) + \sum_{i=1}^{N}\alpha_i y_i\beta - \sum_{i=1}^{N}\alpha_i + \sum_{i=1}^{N}\alpha_i\xi_i\right) \\
0 &= \sum_{i=1}^{N}\alpha_i y_i \\
A_3 &= -\left(\sum_{i=1}^{N}\alpha_i y_i \mathbf{w}^T\phi(\mathbf{x}_i) - \sum_{i=1}^{N}\alpha_i + \sum_{i=1}^{N}\alpha_i\xi_i\right) \\
\mathbf{w} &= \sum_{i=1}^{N}\alpha_i y_i\phi(\mathbf{x}_i) \\
A_3 &= -\left(\sum_{i=1}^{N}\sum_{i'=1}^{N}\alpha_i\alpha_{i'}y_i y_{i'}\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_{i'}) - \sum_{i=1}^{N}\alpha_i + \sum_{i=1}^{N}\alpha_i\xi_i\right)
\end{aligned}
$$

Now, if we combine these three results $A_1 + A_2 + A_3$, we will get

$$
\begin{aligned}
A_1 + A_2 + A_3 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\alpha_i\alpha_{i'}y_i y_{i'}\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_{i'}) + \sum_{i=1}^{N}\alpha_i\xi_i \\
&\quad - \left(\sum_{i=1}^{N}\sum_{i'=1}^{N}\alpha_i\alpha_{i'}y_i y_{i'}\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_{i'}) - \sum_{i=1}^{N}\alpha_i + \sum_{i=1}^{N}\alpha_i\xi_i\right) \\
A_1 + A_2 + A_3 &= \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\alpha_i\alpha_{i'}y_i y_{i'}\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_{i'}),
\end{aligned}
$$

which is identical to the dual function.

References

[1] G. Aldering. The nearby supernova factory, May 2009. http://snfactory.lbl.gov/index.html.

[2] S. Bailey, C. Aragon, R. Romano, R. C. Thomas, B. A. Weaver, and D. Wong. How to find more supernovae with less work: Object classification techniques for difference imaging. *Astrophysical Journal*, 665:1246–1253, 2007.

[3] L. Brieman, J. H. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Chapman & Hall, 1984.

[4] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[5] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*. Computer Science Dept., National Taiwan University, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[6] W. J. Dixon. Simplified estimation from censored normal samples. *The Annals of Mathematical Statistics*, 31:385–391, 1960.

[7] J. H. Friedman and B. E. Popescu. Gradient directed regularization for linear regression and classification. Technical report, Statistics Dept., Stanford University, 2004.

[8] J. H. Friedman and B. E. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3):916–954, 2008.

[9] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, chapter 9, 10, 12, pages 316–331, 371–379. Springer Science, 2001.

[10] C. Hsu, C. Chang, and C. Lin. A practical guide to support vector classification. Technical report, Computer Science Dept., National Taiwan University, 2009.

[11] W. L. Martinez and A. R. Martinez. *Computational Statistics Handbook with Matlab*. Chapman & Hall, 2002.

[12] C. Wickham. Classifying the supernova data, 2007. Private communication (PowerPoint presentation).