Pomona College

Senior Thesis in Mathematics

SVM and the Application of Prediction Rules

Author:
Isaiah Boone

Advisor:
Dr. Johanna Hardin

Submitted to Pomona College in Partial Fulfillment
of the Degree of Bachelor of Arts

March 29, 2016

Abstract

This paper details SVM and how they are built. I apply different model-building algorithms based on Bradley Efron's paper, *Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation* (1983). In his paper, Efron examines the error rate different prediction rules. Though Efron applied this analysis of prediction rules to regression models, I apply these models to SVM and examine their performance and accuracy in measuring error rates.

Contents

Chapter 1

Introduction

The main focus of this thesis is SVM (SVMs). SVM are a method of classification. A classification algorithm is an algorithm in which given a certain amount of information, a model is built that is used to predict the classification of future data that is input into the model. An example of how a model of this nature might work might be that a doctor wants to predict whether or not somehow has heart disease based on two variables, their height and their weight. I know that these variables aren't necessarily the best predictor of heart disease in an individual but they will suffice for a simple example. To begin with we have two classes, in one class are the people with heart disease, in the other class are people without heart disease. We would take information on patients who we already know either have or don't have heart disease and use them for the model. An SVM would take the information on these patients, such as their height, weight, and whether or not they have heart disease, and train a model. The model would then use this prior knowledge to predict whether or not future individuals will have heart disease based on these different attributes of the two classes. For instance, if the model assesses that many of the people that did not have heart disease were above a certain height, then the program would be more likely to classify a future observation above that height as not having heart disease, though it would take into account weight as well.

What's the usefulness of this type of classification algorithm you might ask. Well when predicting observations we do not always know the classification of the individual and so a classification algorithm can provide valuable information. In the context of the example, this information might be useful for a doctor in deciding treatment for a patient especially in urgent situations.

The truly useful part about SVM is that we can use a lot more than just two variables to create very detailed models and the real world applications can provide a number of benefits.

Breaking down the following chapters, the second chapter of this thesis details SVMs and the how they are technically derived. The third chapter discusses the typical method of evaluating the error rate of SVMs. The fourth and fifth chapters addresses Efron's paper that was mentioned in the abstract, and focuses on the derivation and rationale behind the prediction algorithms he proposed in his paper. The sixth chapter is a summary of the results of the prediction algorithms applied to a real world data set. And finally the seventh chapter reviews the findings of the applications and reflects on whether Efron's prediction algorithms are suited for use with SVM.

Chapter 2

Support Vector Machines

For someone who has very little familiarity with classification algorithms there is not a clear point at which to begin. However, with SVMs the concept of hyperplanes is basis of the model and is where this chapter will start before eventually explaining how this ties into SVMs.

2.1 Hyperplanes

A hyperplane in a p-dimensional space is a flat subspace of $p-1$ dimensions. In a two dimensional space, the hyperplane is a line and is defined by the equation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0 \tag{2.1}$$

for parameters β_0, β_1, and β_2. In a three dimensional space, the hyperplane is a plane and is given by the equation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 = 0 \tag{2.2}$$

for parameters β_0, β_1, β_2, and β_3. Defining the hyperplane means that any $X = (X_1, X_2, X_3)^T$ that satisfies (2.2) is a point on the hyperplane. Equation 2.2 can be extended to p-dimensions. For p-dimensions, the equation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = 0 \tag{2.3}$$

defines the appropriate hyperplane. Similarly to how it was described earlier, defining the hyperplane means that if a point $X = (X_1, X_2, ..., X_p)^T$ satisfies

(2.3), then X lies on the hyperplane. If X, on the other hand, does not satisfy (2.3) and instead,

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p > 0 \tag{2.4}$$

then we know X lies to one side of the hyperplane. Alternatively, if

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p < 0, \tag{2.5}$$

then we know X lies on the other side of the hyperplane. The $p-1$ dimensional hyperplane can thus be viewed as dividing a p-dimensional space into two subspaces.

In terms of the application of hyperplanes to classification, suppose that there are n observations with p characteristics represented by the n x p matrix M.

$$M = \begin{pmatrix} x_1 \\ . \\ . \\ . \\ x_n \end{pmatrix} \textit{where} \quad x_1 = \begin{pmatrix} x_{11} \\ . \\ . \\ . \\ x_{1p} \end{pmatrix}^T, ..., x_n = \begin{pmatrix} x_{n1} \\ . \\ . \\ . \\ x_{np}, \end{pmatrix}^T \tag{2.6}$$

from which x_i for $i \in \{1, n\}$ belongs to one of two classes, $\{-1, 1\}$ or in other words, $y_1, ..., y_n \in \{-1, 1\}$ where -1 and 1 represent two distinct classes. These data are considered to be the training observations and our goal is to create a decision rule based on our training observations that correctly predicts the class of additional observations. One way to separate these training data is to create a hyperplane that separates the training observations according to their class, i.e. the x_i that are members of -1 are separated from x_i that are members of 1 as seen in the left subfigure of Figure 2.1. If we assume we are able to construct a separating hyperplane, it would look something like any one of the hyperplanes in the right figure of Figure 2.1. We can say that the solid dots to the left of the separating hyperplane belong to one of the classes, $y_i = -1$ and the open dots to the right belong to the other class such that $y_i = +1$.

Thus a separating hyperplane is such that

$$\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + ... + \beta_p X_{ip} > 0 \text{ if } y_i = 1, \tag{2.7}$$

and

$$\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + ... + \beta_p X_{ip} < 0 \text{ if } y_i = -1. \tag{2.8}$$

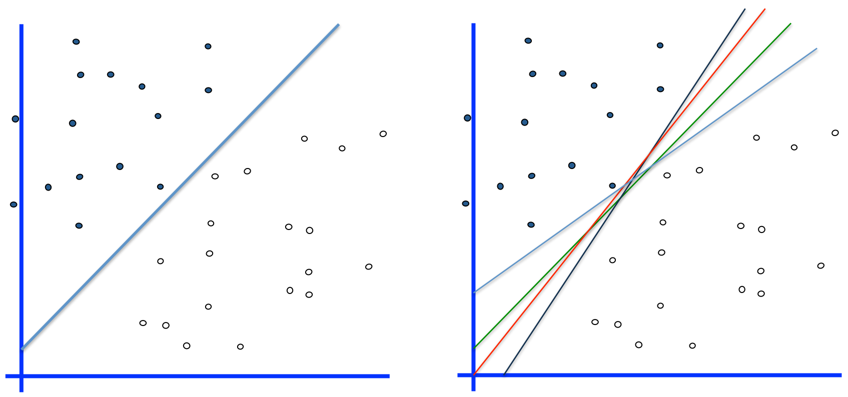


Figure 2.1: Left: Single Hyperplane Right: Multiple Hyperplanes Solid Point: Class 1 Open Point: Class 2

These two equations can also be rewritten in the form

$$y_i(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + ... + \beta_p X_{ip}) > 0 \tag{2.9}$$

for all $i = 1, ..., n$.

We want to define a function that will correctly classify our test observations. The idea behind this is that the observation will be assigned to a class depending the side of the hyperplane it falls on. Consider a new observation $x*$. The test observations are classified based on the function, $f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + ... + \beta_p x_p^*$. If the function returns a negative value for a given test observation, it is classified into -1. If the function returns a positive value then we assign to 1. The magnitude of $f(x^*)$ is also useful as well as the further $f(x^*)$ is from 0, the further x^* is from the the hyperplane and thus the more confident we can be about its classification. On the contrary, if $f(x^*)$ is close to 0, then we know that x^* is close to the separating hyperplane and therefore we are less certain about its classification.

2.2 Maximal Margin Hyperplane

In the case that the observations can be perfectly separated into two groups, there are an infinite number of separating hyperplanes, the right subfigure in Figure 2.1 shows just a handful. The next question that arises after learning that a separating hyperplane can be used to classify the test observations and

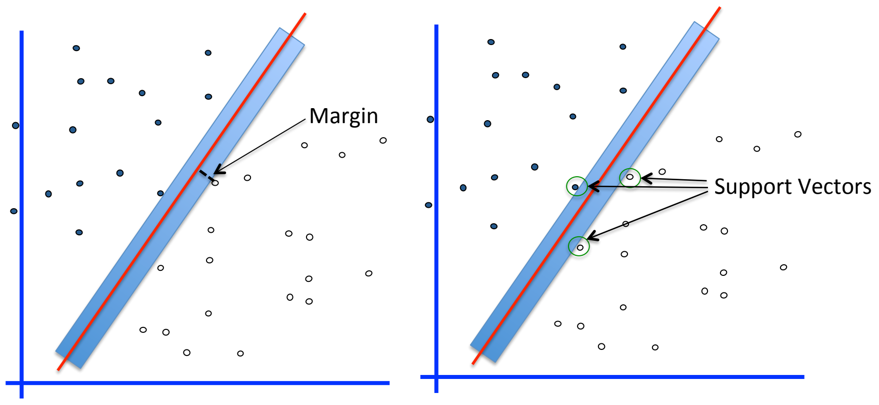


Figure 2.2: Left: Margin Diagram Right: Support Vectors Diagram

that there are an infinite number of them, is how to decide which hyperplane is the best. One of the previous facts we stated provides some of the intuition behind the method used to identify the best hyperplane. In the previous section we expressed that if $f(x^*)$ is far away from 0, we are more confident about the classification. Thus we would like to find the $f(\cdot)$ or the hyperplane that creates the greatest distance between the different classes.

This hyperplane is known as the maximal margin hyperplane, which is the separating hyperplane furthest from the training observations. The smallest (perpendicular) distance from the training observations to the hyperplane is known as the margin and the maximal margin hyperplane is the hyperplane which has the largest margin. Classifying the test observations based on the side to which they lie using the maximal margin hyperplane is known as the maximal margin classifier. When examining a maximal margin hyperplane, we see that there are test observations that lie along the margin, these observations are known as support vectors and are actually the only observations that factor in when creating the hyperplane.

After defining what the maximal margin classifier is, the next step is to construct it, which we do by solving the following optimization problem,

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{maximize} \; M \tag{2.10}$$

subject to

$$\sum_{j=1}^{p} \beta_j^2 = 1, \tag{2.11}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}) \geq \mathrm{M} \quad \forall \quad i = 1, ..., n. \tag{2.12}$$

where M is the margin.

The equations required for the optimization problem ensures certain aspects of the maximal margin classifier. Equation 2.10 ensures that the distance between the separating plane and the nearest observations is as large possible. Equation 2.12 makes sure that each observation falls on the correct side, while equation 2.11 makes sure the hyperplane is unique.

2.3 Lagrange Multipliers

The optimization problem in equations 2.10 - 2.12 can be solved using Lagrange multipliers. However when we go to solve this problem, we realize that the (2.11) constraint is actually rather difficult and not something that can easily be solved with any software that we have. Let $w = (\beta_1, \beta_2, ..., \beta_p)$. To simplify, the constraint 2.11 can be rewritten as

$$\|w\| = 1 \tag{2.13}$$

As we mentioned earlier, this is difficult to work with so we transform our conditions such that they guarantee the same properties but are more suitable for optimization. In order to do this, we select $\hat{M}$ such that $\frac{\hat{M}}{\|w\|} = M$. We can then write the optimization problem as:

$$\underset{\beta_0, \beta_1, ..., \beta_p}{maximize} \ \frac{\hat{M}}{\|w\|} \tag{2.14}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}) \geq \hat{M} \quad \forall \quad i = 1, ..., n. \tag{2.15}$$

Though we only have two constraints now, we still have a non-convex objective function in equation (2.14). Due to the properties of Lagrange multipliers and our conditions we can add an arbitrary scaling constraint on w such that the functional margin with respect to the training set must be 1:

$$\hat{M} = 1. \tag{2.16}$$

It then becomes clear that in order to maximize $\frac{1}{\|w\|}$ we must minimize $\|w\|^2$, which gives us the following modified optimization problem:

$$\underset{\beta_0,\beta_1,...,\beta_p}{minimize} \ \frac{1}{2}\|w\|^2 \tag{2.17}$$

$$\text{subject to } \ y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) \geq 1 \ \ \forall \ \ i = 1, ..., n. \tag{2.18}$$

With conditions (2.17) and (2.18), the optimization problem is now a lot more manageable. We will solve this optimization problem. The following is a simple example with two points and two classes. It is primarily drawn from Baxter (2004).

Example Suppose we have two points in two dimensions that fall into two distinct classes and we want to find the optimal separating plane which in this case is a line. Let these two points be:

$$\dot{x}_1 = A_1 = (a, b), \dot{x}_2 = B_1 = (c, d) \tag{2.19}$$

where a, b, c and d $\in \mathbb{R}$.

To start with, for any constraint-based optimization, the following conditions known as the Krush-Kuhn-Tucker conditions (KKT) must be met. The KKT Conditions are required when optimizing a nonlinear programming problem with inequality constraints. The Lagrange method doesn't accomplish this when the problem doesn't have strict equality constraints. The KKT conditions guarantee the solution we find is indeed optimal. In the example we are working out, the optimization problem doesn't actually have any inequality constraints, so the KKT conditions aren't necessary. However, it is still useful to show how they would be used since the process would be the same if we did have inequality constraints. Important to note is that in this case based on the way w is defined, $w = (\beta_1, \beta_2)$. But in cases in higher dimensions w will have additional parameters. The KKT conditions are as such:

$$\frac{\partial}{\partial w} L(w, \beta_0, \lambda) = w - \sum_i \lambda_i y_i \dot{x}_i = 0 \tag{2.20}$$

$$\frac{\partial}{\partial \beta_0} L(w, \beta_0, \lambda) = - \sum_i \lambda_i y_i = 0 \tag{2.21}$$

$$y_i[\langle w, (\dot{x}_1, \dot{x}_2) \rangle + \beta_0] - 1 \geq 0 \tag{2.22}$$

$$\lambda_i \geq 0 \tag{2.23}$$

$$\lambda_i(y_i[\langle w, (\dot{x}_1, \dot{x}_2)\rangle + \beta_0] - 1) = 0 \tag{2.24}$$

Using equations (2.17) and (2.18) from earlier, we see that the function f and the constraint g can be written as:

$$f(w) = \frac{1}{2}\|w\|^2 \tag{2.25}$$

$$g_i(w, \beta_0) = y_i[\langle w, \dot{x}_i\rangle + \beta_0] - 1) = 0 \tag{2.26}$$

With Lagrange multipliers we know $g_i(w, \beta_0)$ can be expanded to:

$$g_1(w, \beta_0) = [\langle w, \dot{x}_1\rangle + \beta_0] - 1 \geq 0 \tag{2.27}$$

$$g_2(w, \beta_0) = -[\langle w, \dot{x}_2\rangle + \beta_0] - 1 \geq 0 \tag{2.28}$$

As a results we can write the Lagrange as:

$$L(w, \beta_0, \lambda) = f(w) - \lambda_1 g_1(w, \beta_0) - \lambda_2 g_2(w, \beta_0) \tag{2.29}$$

Plugging in the equations for $f(w)$ and $g_i(w, \beta_0)$ we write the Lagrangian as:

$$\begin{aligned} L(w, \beta_0, \lambda) &= \frac{1}{2}\|w\|^2 - \lambda_1([\langle w, \dot{x}_1\rangle + \beta_0] - 1) - \lambda_2(-[\langle w, \dot{x}_2\rangle + \beta_0] - 1) \\ &= \frac{1}{2}\|w\|^2 - \lambda_1([\langle w, \dot{x}_1\rangle + \beta_0] - 1) + \lambda_2([\langle w, \dot{x}_2\rangle + \beta_0] - 1) \end{aligned} \tag{2.30}$$

The resulting gradient can be written in the form:

$$\nabla L(w, \beta_0, \lambda) = \nabla f(w) - \lambda_1 \nabla g_1(w, \beta_0) - \lambda_2 \nabla g_2(w, \beta_0) = 0 \tag{2.31}$$

The gradient of the Lagrange gives us the following system of equations when we take their partial derivates:

$$\frac{\partial}{\partial w} L(w, \beta_0, \lambda) = w - \lambda_1 \dot{x}_1 + \lambda_2 \dot{x}_2 = 0 \tag{2.32}$$

$$\frac{\partial}{\partial \beta_0} L(w, \beta_0, \lambda) = -\lambda_1 + \lambda_2 = 0 \tag{2.33}$$

$$\frac{\partial}{\partial \lambda_1} L(w, \beta_0, \lambda) = [\langle w, \dot{x}_1\rangle + \beta_0] - 1 = 0 \tag{2.34}$$

$$\frac{\partial}{\partial \lambda_2} L(w, \beta_0, \lambda) = [\langle w, \dot{x}_2 \rangle + \beta_0] + 1 = 0 \tag{2.35}$$

If we take equations (2.34) and (2.35) we can write that:

$$\begin{aligned} \langle w, \dot{x}_1 \rangle + \beta_0 - 1 &= \langle w, \dot{x}_2 \rangle + \beta_0 + 1 \\ \langle w, \dot{x}_1 \rangle - 1 &= \langle w, \dot{x}_2 \rangle + 1 \\ \langle w, \dot{x}_1 \rangle - \langle w, \dot{x}_2 \rangle &= 2 \\ \langle w, [\dot{x}_1 - \dot{x}_2] \rangle &= 2 \end{aligned} \tag{2.36}$$

Let the two points we mentioned earlier be:

$$\dot{x}_1 = A_1 = (0, 0), \dot{x}_2 = B_1 = (2, 2) \tag{2.37}$$

Plugging in the values we have for points $\dot{x}_1$ and $\dot{x}_2$, we can find the equation for the vector w.

$$\begin{aligned} \langle (w_1, w_2), [(0, 0) - (2, 2)] \rangle &= 2 \\ \langle (w_1, w_2), (-2, -2) \rangle &= 2 \\ -2w_1 - 2w_2 &= 2 \\ 2w_1 &= -(2w_2 + 2 \\ w_1 &= -(w_2 + 2) \end{aligned} \tag{2.38}$$

We can now combine equations (2.32) and (2.33) along with our points in order to solve for w. Equation (2.33) tells us that $\lambda_1 = \lambda_2$ so we can write equation (2.32) as:

$$(w_1, w_2) - \lambda_1(0, 0) + \lambda_2(2, 2) = 0 \tag{2.39}$$

$$(w_1, w_2) - \lambda_1(0, 0) + \lambda_1(2, 2) = 0 \tag{2.40}$$

$$(w_1, w_2) + \lambda_1(2, 2) = 0 \tag{2.41}$$

Thus we have that:

$$w_1 + 2\lambda_1 = 0 \tag{2.42}$$

and

$$w_2 + 2\lambda_1 = 0 \tag{2.43}$$

These two equations imply that:

$$w_1 = w_2 \tag{2.44}$$

Plugging this back into equation (2.38) we get that:

$$w_1 = w_2 = -1 \tag{2.45}$$

As a result, we can see from equations (2.42) and (2.43) that:

$$\lambda_1 = \lambda_2 = .5 \tag{2.46}$$

Recall equation (2.35), with the values that we have for w and the point $\dot{x}_1$ we can express is as:

$$\begin{aligned} [\langle -1, (0,0)\rangle + \beta_0] - 1 &= 0 \\ \beta_0 - 1 &= 0 \\ \beta_0 &= 1 \end{aligned} \tag{2.47}$$

We can see from our derivation that equations (2.20), (2.21), (2.22) and (2.24) from the KKT conditions hold. We can also see that equation (2.23) is satisfied since:

$$\lambda_1 = \lambda_2 = .5 \geq 0 \tag{2.48}$$

Accordingly we see that all the KKT conditions are met, which means that we know our solution is the optimal solution for the given problem. The equation of the hyperplane for our maximal margin classifier is the line $1 + \frac{1}{2}x_1 + \frac{1}{2}x_2 = 0$. This simple example demonstrate the basic method by which the Lagrangian is solved. The following section describes Kernel Function, which are incorporated into the Lagrangian for more complex problems.

2.4 Kernel Functions

The following description of kernel function and their uses in SVM was mainly drawn from Berwick (2008). In the previous section we discussed linear decision boundaries. Although we didn't discuss it explicitly, in some cases we might not be able to fit the data perfectly, but we can still fit the vast majority of the points using a linear decision boundary. However, there is also the possibility that a linear decision boundary isn't appropriate at all. As can be seen in the left figure in Figure 2.3, there is no line that will fit the observations relatively well. Using kernel functions we can find a non-linear decision boundary that fits the observations much better as you can see in the right figure in Figure 2.3.

Finding a non-linear decision boundary is where kernel functions come in handy. They help to dramatically simplify the process by mapping observations in the original feature space into a higher dimensional space. The intuition is that if we increase the number of dimensions by enough, aside from when there are observations with the same x-values and opposite classifications, there will always a be a plane that will be able to separate the observations. We then map the linear high dimensional hyperplane back into the original feature space and that resulting projection is our non-linear decision boundary.

Kernel functions have the form:

$$K(\vec{x_i}, \vec{x_j}) = \phi(\vec{x_i}) \cdot \phi(\vec{x_j}) \tag{2.49}$$

where $\phi(x)$ maps x into a higher dimensional space. Consider the following example:

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \vec{y} = \begin{pmatrix} y_1 \\ y_2, \end{pmatrix} \tag{2.50}$$

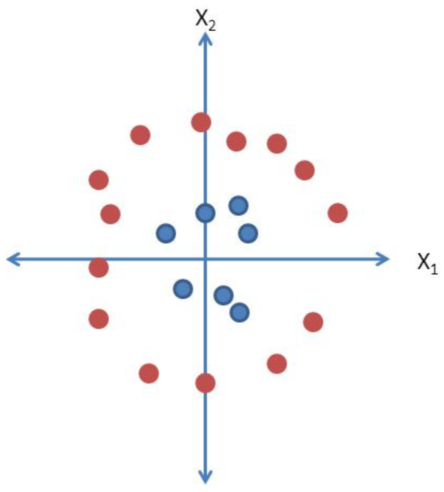
and

$$\phi(\vec{x}) = (x_1, x_2, x_1^2 + x_2^2) \tag{2.51}$$

We now have a function that will map our observations from their original 2-dimensional space to a 3-dimensional space. The the kernel function would then look like:

$$\begin{aligned} K(\vec{x}, \vec{y}) &= \phi(\vec{x}) \cdot \phi(\vec{y}) \\ &= (x_1, x_2, x_1^2 + x_2^2) \cdot (y_1, y_2, y_1^2 + y_2^2) \\ &= x_1 y_1 + x_2 y_2 + (x_1^2 + x_2^2)(y_1^2 + y_2^2) \\ &= x_1 y_1 + x_2 y_2 + x_1^2 y_1^2 + x_1^2 y_2^2 + x_2^2 y_1^2 + x_2^2 y_2^2 \end{aligned} \tag{2.52}$$

The amazing part about kernel functions is that we don't need to know ϕ, the equation (2.51) can be written as a function of the dot product of the data in the original space. In Figure 2.4 you can see that the 2-D figure on the left is not linearly separable. However, when mapped into three dimensions, which is shown in the figure on the right, you can find a plane that separates the data into the two appropriate classes. When mapped back down, the decision boundary will be circular and look very similar to the decision boundary presented in Figure 2.3These two figures demonstrate how the transformation described earlier might look visually.



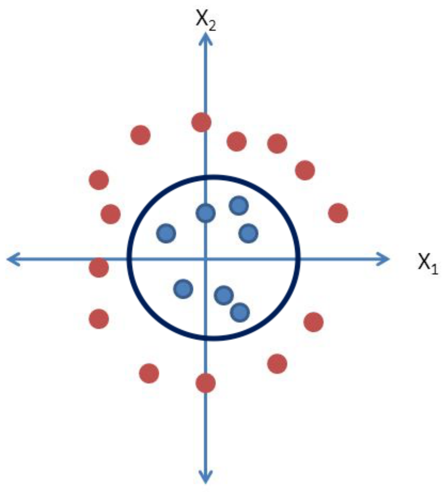
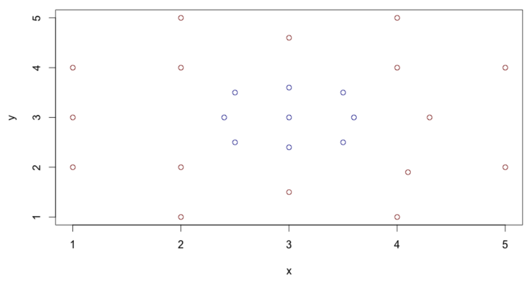


Figure 2.3: Non-Linearly Separable Observations Red: Class 1 Blue: Class 2



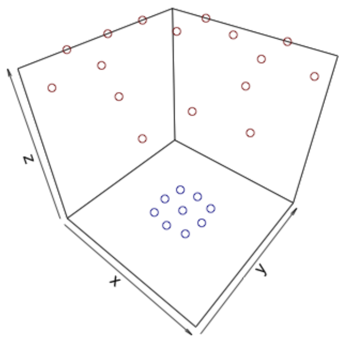


Figure 2.4: Non-Linearly Separable Observations Red: Class 1 Blue: Class 2

There are 3 main kernel functions that are used frequently.

Linear Kernel:

$$K(\vec{x},\vec{y}) = \sum_{j=1} x_{ij} \cdot y_{ij}, \tag{2.53}$$

Polynomial Kernel:

$$K(\vec{x},\vec{y}) = (1 + \sum_{j=1} x_{ij} \cdot y_{ij})^d \tag{2.54}$$

Radial Kernel:

$$K(\vec{x},\vec{y}) = exp(-\gamma \sum_{j=1} (x_{ij} - y_{ij})^2) \tag{2.55}$$

The kernel selected varies according to which one best fits the data, and in the most commonly selected one out of the three is the radial kernel.

When examining the kernel functions, it is clear that 2 of the 3 require parameters.The question that usually arises after examining this is how do you know which parameters give the best model. The answer is that it firsts depends on the data given and the process of finding the best parameter once you have the data is called tuning and is used to narrow down which parameters are the best for the model. To do this we use a technique called cross-validation (Which is discussed in the following chapter) in order to determine under which parameter does the model perform the best. The technique of cross-validation introduces a concept known as the cost variable for SVMs. The cost variable is the weight we assess creating a larger margin at the price of increasing training error, which ultimately means you don't have to perfectly separate the points. The larger cost we have, the fewer the number of misclassifications as they are penalized more heavily. Additionally, cross-validation is used in order to determine the parameters for the kernel functions as well. The following chapter will help to explain how this is done. Thus, through the process of cross-validation the parameters are fine tuned and the best model is put forward.

Ultimately, including the kernel function into the Lagrange for the optimization problem of fitting a hyper plane in a higher dimensional transformed space, we arrive at the following equation which is used in order to determine the optimal decision boundary for SVM.

$$L(w,\beta_0,\lambda) = \frac{1}{2}\|w\|^2 - w\sum_i \lambda_i y_i K(\dot{x}_i, \dot{x}_{i'}) = 0 \tag{2.56}$$

The resulting optimization problem is more complex than the small one that was walked through earlier. However, the intuition is the same. The same process is used in which the Lagrange Multiplier is derived and used to solve for the unknown parameter values for the hyperplane. The terms that are included in the optimization are no longer just the x_1 and x_2 that were used in the example in Chapter 2. Mapping the observation to a higher dimensional space might seem like an extensive task. The kernel trick saves us a lot of time in this process due to the fact that we don't have to know the equations that map the observation but instead can examine functions of the dot products. As a result, the kernel trick proves to be invaluable in the process of creating SVM and solving the optimization problem with equation 2.56 is the way in which a SVM is built.

Chapter 3

Cross-Validation

3.1 Error Rates

Error Rates in machine learning algorithms are very important as they inform the model builder of the expected accuracy of future predictions. The error rate is measured by: $\frac{(Number\ of\ Misclassifications)}{(Number\ of\ Test\ Observations)}$. A lower error rate results in a greater level of confidence in future predictions predicted correctly. In order to compute the error rate for a given set of observations you must know the response variable of these observations. In most cases the only observations for which we know the response variable are in our sample. However, we should not treat the error rate found using the training data as the actual test error rate due to the fact that the model will be inherently fit towards the training observations and return an error rate that is lower than the true future error rate.

3.2 Cross-Validation

Cross-Validation addresses the overfitting issue by designating some of the training data for model building and another portion for testing. In cross validation the training set of data is partitioned into K groups. The algorithm for K-Fold Cross-Validation can be written as such:
1. Partition the data set into K groups
2. Remove one of the groups from the data set and build the SVM on the remaining K-1 groups
3. Run the held-out observation through the model and compare the pre-

dicted classification to the actual classification and keep track of misclassifications.
4. Repeat Steps 2 and 3 for each of the K groups
5. Average the misclassification error for all of the K models that were built.

The resulting error rate can then be used as an estimation for the true error rate of the model.

3.3 Cross-Validation for Model Optimization

Cross-Validation isn't only limited to helping find the true error rate of a model, but can also be used to fine tune parameters for a given model. In the context of SVM, for most models, the cost parameter is tuned. This process is very similar to K-Fold Cross-Validation for estimating error rates, however for each of the K-Folds, we assign a different parameter value. Whichever parameter value returns the lowest error rate is selected for the model at hand. If we have a model with one parameter λ, the algorithm for that model can be written as such:
1. Partition the data set into K groups
2. Assign each of the groups a different for λ
3. Remove one of the groups from the data set and build the Support Vector Machine on the remaining K-1 groups with the λ value for the held out group as the parameter value
4. Predict the classification of the held out group by using the model built on the remaining K-1 groups and compare to the actual classifications for those observations. Keep track of misclassifications and divide by the number of observations in the left out sample.
5. Repeat Steps 3 and 4 for each of the K groups
6. Select the λ value that returned the model with lowest error rate.

Through this process, we can find the parameter value that is best fit for the model and data at hand. As was mentioned earlier this is very useful with kernel functions. As you can see, there are different parameters for both the polynomial and radial kernels, (2.54) and (2.55). When building a Support Vector Machine with either polynomial or radial kernels, we fine tune these parameters in order to find the best model. We then can use Cross-Validation again to find the estimated error rate of the model with the optimal parameters.

Chapter 4

Prediction Rules Using SVM

4.1 Support Vector Machine Bagging

When building decision rules, there are a number of factors that influence the effectiveness of the given model. Two of the most important factors are bias and variance. While bias is calculated by the average error between predicted values and the actual values of these points, the variance is the difference between predictions for the same point for different models. The variance of a model is very important to observe because it means that if we have a high variance we might get a very different model based on our samples. This is not ideal since it implies that even if the model performs well on training data, it might not make accurate predictions on other observations since it is fit so closely to the training data. One way to reduce this variance is to lower the cost variable, which was mentioned earlier, that is used in the model. Another option would be to create multiple models, and average across the models. By taking the average prediction, you would significantly reduce the variance as it would no longer be dependent on a single model. The following sections go into more detail on how this is done and expands on how it is used in regards to prediction rules for SVM.

4.1.1 Bootstrapping

When we sample from a population we are hopefully getting a group of observations that is representative of the entire population. The only thing that would be better than getting one sample from the population would be to get more than one sample from the population. With additional samples from the population, statisticians and researchers could build a better model based on a larger sample size. These models in essence would be better since there is a certain amount of variability from sample to sample, even with random sampling and taking multiple sample helps to reduce this variability. The only issue with resampling from the population is that in almost all cases, statisticians and researchers aren't able to gather additional samples from the population. In these situations, the next best option is to bootstrap.

Bootstrapping is the process of resampling from the sample that's been taken from the population. What this entails is that if we have a sample, let's call it W, of n observations, we randomly sample from W with replacement until we have n observations. The idea of resampling can be visualized as reaching into a bag of marbles, pulling out one marble, writing down the color of the marble, returning the marble to the bag and repeating the process until you have gathered enough observations. The bag of marbles would be your sample, and each marble you take out and take note of is an observation for the bootstrapped sample. The idea is that if the sample is representative of the population, by resampling from the sample, the new samples will also be representative of the actual population and by examining these additional samples, statisticians and researchers will be able to reduce the variability in their model.

4.1.2 Out of Bag Error Rate

When bootstrapping, on average approximately one third of the observations that were in the original sample do not make it into the bootstrapped sample. These observations are known as out of bag observations and actually turn out to be very helpful. Since these observation are left out, the sample and model that is built using these samples are only built using roughly two thirds of the available observations. Thus we are able to use the out of bag observations to test the accuracy of our model. The resulting error rate that is computed by summing the total number of misclassified predictions and dividing by the total number of predictions is known as the out of bag error rate.

4.1.3 SVM Bagging

An additional benefit of multiple different ”samples” is the fact that different models can be built using these samples. By building models for each of the

bootstrapped samples and taking the average or aggregate response, one can build a classification algorithm with a lot less variance than an algorithm which relies on just one sample. In the case of Support Vector Machine Bagging, the models would all be SVMs and the predicted outcome for the classification would be the majority response variable across the various models. The procedural points for this method would be to
1. Bootstrap sample B times
2. For each of the B bootstrap samples, build a Support Vector Machine
3. Take the majority response from the B models and label as the classification for the bagging model

Once the model is built, the OOB error rate is used to approximate the true error rate of the model.

4.2 Support Vector Machine Randomized Bootstrap Bagging

Relating very closely to SVM Bagging, Support Vector Machine Randomized Bootstrap Bagging (SVM RBB) is almost identical expect that there is a small modification in the sampling procedure. The intuition behind SVM RBB is that when resampling from our sample, the class of an observation is certain. If we let $\bar{y}_i$ be the complement of y_i, then this can be written as $\pi_i(y_i, \mathbf{x}) = 1, \pi_i(\bar{y}_i, \mathbf{x}) = 0$. However, there is always the possibility that due to some error the classifications we have aren't completely accurate. SVM RBB assigns some of the probability mass to the complementary response to account for potential uncertainty. Efron (1983) uses $\pi_i(y_i, \mathbf{x}) = .9, \pi_i(\bar{y}_i, \mathbf{x}) = .1$ Thus each time an observation is selected during the process of bootstrapping, its classification is determined using a Bernoulli(.9) distribution, where the new classification remains the same as the old classification if a 1 is returned and the new classification become the complementary classification if a 0 is returned. This is done for all observations that are sampled during the bootstrapping process. Then just like SVM Bagging, the B samples are used to build B models that are used to create the algorithm. The OOB Error Rate is used to determine the approximate error rate of the model.

Chapter 5

The .632 Estimator

5.1 The .632 Estimator

The .632 Estimator is different than the prediction rules described in Chapter 4. While the prediction rules included in Chapter 4 detail the algorithm for creating a learning method, the .632 Estimator provides an estimate of the error rate for a bagging algorithm. Most of the following information is taken from Efron (1997). The .632 estimator differs from the previous algorithms in that it itself is not an actual classification method. What it does instead is provide an estimate of what the true error rate will be for an algorithm which uses bootstrapping. While other methods of predicting error rate have low bias, they can have high variance which is not desirable when estimating error rates. The .632 Estimator tries to factor this in by including Leave-One-Out Cross Validation (LOOCV) in its calculation of the error rate.

The formula for producing the .632 Estimator is:

$$Err^{(.632)} = .368 \times err + .632 \times Err^{(1)}. \tag{5.1}$$

Where

$$Err^{(1)} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{|C^{-1}|} \sum_{b \in C^{-1}} L(y_i, f^b(x_i)) \tag{5.2}$$

and err is the training error rate. $Err^{(1)}$ represents the extra-sample prediction error. L in this context is a loss function and not be confused with a Lagrangian. The function measures whether the bth model correctly predicts the ith predicted value given the ith observation. The calculation is derived from generating B bootstraps. One observation is selected and all of the models from bootstrapped samples that do not include that specific observation are used to test the left out observation. The samples without the given observation are identified by $|C^{-1}|$. What this means is that $|C^{-1}|$ is the set of samples that do not contain that observations. The intuition behind the .632 estimator is that the LOOCV error greatly reduces the amount of variance, however this estimator still has issues with bias due to the repetition of observations in the bootstrapped samples. To attempt to balance out the variance problems with training error and the bias problem with leave-one-out bootstrap error, the percentages of .368 and .632 are assigned respectively. The .632 number comes from the fact that bootstrap sample contains close to .632 of the points of the original sample.

Though the definition of the .632 Estimator provided in this chapter is relatively succinct, it encompasses the essential components of the estimator. By taking the LOOCV error rate and weighing it along with the training error rate, we expect to calculate a more accurate error rate. In Efron (1983) Efron explains that this estimator is the most accurate estimation of the true error rate of the bagged model, and I will examine if this is true with SVMs as well.

Chapter 6

Applications

In this section, I apply the prediction rules and the .632 Estimator described in Chapters 4 and Chapter 5 to two data sets. In Efron (1983), Efron examines how the estimated error rates perform with Linear Regression Models. I, however, use SVMs as the base of the algorithms and seek to see how they perform in regard to the actual error rate. I run the analysis a number of times to get an idea of how much variability we might see when performing this type of analysis. By conducting this application, I want to see if there are any interesting trends or findings and examine how the results might compare to what Efron found in his paper.

6.1 Gapminder

6.1.1 Data Set

One of the data sets used for the application portion was downloaded from GapMinder. GapMinder data is an online collection of various information on countries across the world. GapMinder data is tracked over a period of years and can be used to assess changes in economical and social aspects of countries.The data that I downloaded for this application includes the following variables:

- CO2 emission levels
- Residential electricity consumption
- Income Share Held by the Highest 10 percent

- Adult Literacy Rate of Females above 15 years old
- Billionaires per million inhabitants
- Percentage of Unemployed People above 15 years Old
- Agricultural Land as Percentage of Total Land Area
- Agricultural Value Added as Percentage of GDP
- Total Number of Agriculture Workers
- Children out of Primary School
- Aid Received as Percentage of GNI
- Tuberculosis Mortality as Percentage of Child Mortality per 100,000 Population per Year
- Female Children out of Primary School

I downloaded the data into RStudio using the googlesheets package. After downloading the data using the GoogleSheets API I joined all the sheets and selected only data from the most recent year for each respective category. Its important to note that this is not the same year for each category because some data stopped being collected prior to others. I did this with the hope that with improvements in technology and communication, there would be a higher chance that the data would be available. One problem with Gap-Minder data is that since the observational units are countries, a lot of the data is not available and thus are listed as NA. To address this, I removed variables that had either all NAs for all countries or a large number of NAs, which I categorized as more than half. For the remaining variables I set all the NAs equal to 0, which I know isn't the best solution, but was the most reasonable solution given the data set. The predicted variable was whether a country is a developed country or not. This variable was not included in the Gapminder data set and so it was added to the model. Whether or not a country was classified as developed was based on how the term is defined by the CIA. The CIA provides a list of countries regarded as developed and this list was used to determine which countries in the Gapminder data set were classified into developed and undeveloped.

For the process of analysis, I randomly sampled 60 observations out of the 180 available observations to use as my sample. I purposely chose a number that was significantly less than the available 180 observations to reflect that when sampling we often are not getting the majority of the population. By sampling in this way, we could use the remainder of the observations to calculate the actual error rate. Once I had this sample, examined it with 2 Prediction Rules and used the .632 Estimator to approximate what the error rates would be. I repeated this analysis 50 times in order to capture potential variability depending on sampling.

6.1.2 Results

As mentioned earlier, I simulated the analysis 50 times in order to get a more complete range of potential values for the error rates. I also calculated the actual error rate to assess how close they are to our estimated error rate. The following results summarize what I found.

For SVM Bagging,the error rate ranged from 0.06649 to 0.26540 with the 1st and 3rd Quartiles at 0.14410 and 0.19460, respectively. The mean of the bagging error rates was 0.16820 and the median was close to this at 0.16920.

For SVM Randomized Bootstrap Bagging, the error rate ranged from 0.3586 to 0.4844 with the 1st and 3rd Quartiles at 0.4127 and 0.4527, respectively. The mean of the bagging error rates was 0.4326 and the median was close to this at 0.4315.

For The .632 Estimator, the resulting error rate ranged from 0.04137 to 0.17700 with the 1st and 3rd Quartiles at 0.08911 and 0.12420, respectively. The mean of the bagging error rates was 0.10740 and the median was close to this at 0.10480.

The Actual Error Rate ranged from 0.1167 to 0.2167 with the 1st and 3rd Quartiles at 0.1583 and 0.1750, respectively. The mean of the bagging error rates was 0.1652 and the median was close to this at 0.1667.

6.2 OJ Dataset

6.2.1 Data Set

In addition to using data that I downloaded and cleaned from Gapminder, I also used data available in the textbook *An Introduction to Statistical Learn-*

ing. I specifically focused on the OJ data set which is a data set of 1070 observation on 18 different variables. This data was used to predict whether or not a given customer purchased Citrus Hill or Minute Maid Orange Juice. These are the variables with brief descriptions in parentheses:

- WeekofPurchase (Week of purchase)
- StoreID (Store ID)
- PriceCH (Price charged for CH)
- PriceMM (Price charged for MM)
- DiscCH (Discount offered for CH)
- DiscMM (Discount offered for MM)
- SpecialCH (Indicator of special on CH)
- SpecialMM (Indicator of special on MM)
- LoyalCH (Customer brand loyalty for CH)
- SalePriceMM (Sale price for MM)
- SalePriceCH (Sale price for CH)
- PriceDiff (Sale price of MM less sale price of CH)
- Store7 (A factor with levels No and Yes indicating whether the sale is at Store 7)
- PctDiscMM (Percentage discount for MM)
- PctDiscCH (Percentage discount for CH)
- ListPriceDiff (List price of MM less list price of CH)
- STORE (Which of 5 possible stores the sale occurred at)

I did not alter this data set as it didn't have missing entries and I included all variables in the model. I applied the same algorithms to it that I applied to the Gapminder data. The only difference being that I randomly sampled 200 out of 1070 available observations. The following section details the results of this application.

6.2.2 Results

Similar to the OJ set, I simulated the analysis 50 times in order to get a more complete range of potential values for the error rates. I also calculated the actual error rate to assess how close they are to the estimated error rates. Below are the summary statistics for each statistic.

For SVM Bagging, the error rate ranged from 0.2987 to 0.5364 with the 1st and 3rd Quartiles at 0.3708 and 0.4330, respectively. The mean of the bagging error rates was 0.4043 and the median was 0.4058.

For SVM Randomized Bootstrap Bagging, the resulting error rate ranged from 0.4580 to 0.5034 with the 1st and 3rd Quartiles at 0.4852 and 0.4975, respectively. The mean of the bagging error rates was 0.4902 and the median was close to this at 0.4912.

For The .632 Estimator, the error rate ranged from 0.1890 to 0.3362 with the 1st and 3rd Quartiles at 0.2357 and 0.2776, respectively. The mean of the bagging error rates was 0.2571 and the median was close to this at 0.2596.

The Actual Error Rate ranged from 0.3724 to 0.5172 with the 1st and 3rd Quartiles at 0.3851 and 0.4135, respectively. The mean of the bagging error rates was 0.4037 and the median was close to this at 0.3983.

Chapter 7

Conclusions

When examining estimated error rates it is very useful to know how close the estimated error rates are to the true error rate. This information can be used to assess the confidence one should have when predicting future observations. Efron found in his 1983 paper that among the 3 estimators I examined that the Randomized Bootstrap Algorithm returned the lowest error rate while the .632 Estimator was the most accurate measure of the actual error rate. In Chapter 6, I treated the Gapminder and OJ data sets as populations. I took small samples from them and used the remainder of the population to predict the actual error rate for that given SVM. I applied these algorithm to SVMs and tried to see if I would see similar results to what Efron found.

For the Gapminder data set we found that aside from the SVM RBB, the SVM Bagging and .632 Estimator error estimates had similar ranges to the actual error rate of the data set. In terms of mean and median, the SVM Bagging error rate was the only estimator that had a mean and median close to that of the actual error rate. When looking closely it is evident that the SVM RBB error rate tended to overestimate the error rate while the .632 Estimator underestimated the error rate. The SVM RBB error rate range actually doesn't even overlap the range of the Actual Error rate, which suggests that at least with this data set, SVM RBB isn't a good predictor.

For the ISLR OJ data set we found that none of the estimators have ranges similar to that of the Actual Error rate. The closest among the 3 is the SVM Bagging estimator. In addition to having the closest range, the mean and median of the SVM Bagging estimator are very close to the mean and median of the Actual Error Rate. The .632 Estimator underestimates the Actual Error rate while the SVM RBB overestimates the Actual Error rate, which is similar to what we found with the Gapminder data. From these factors, it seems to follow that the SVM Bagging error rate gives the error rate closest to the actual error rates out of the 3 estimators I examined.

Though the findings from Chapter 6 seem to imply the SVM RBB estimator and .632 Estimator don't perform well, there were limitations in the way I conducted the application that hinder the comparison that can be made between the estimators and the actual error rate. One of the largest factors to note is that the error rates for different algorithms were not grouped. What this means is that the smallest error rate for SVM Bagging didn't necessarily come from the same sample that had the smallest actual error rate. Thus we can not assess how much the estimators varied from sample to sample. This is an important assessment to make because even if the estimator and the Actual Error rate have similar summary statistics, if the error rate for the estimator is high when the Actual Error rate is low and vice-versa, the estimator would not be a good estimator of the true error rate at all.

There are additional factors that might have led to the result we found and weaken the argument that Efron's findings can't be applied to SVMs. First, the data set that was used from Gapminder included a lot of NAs that were then turned into 0s so that the model could run appropriately. This easily could've skewed the models and error rates associated with them since in actuality these values aren't 0. At the same time, the classification breakdown for the data set was 148 to 32. This type of breakdown could easily lead to a bias in the models to just classify more observations as undeveloped. Another factor that might have contributed to the results we saw was that Efron applied the algorithms to linear regressions while I applied them to SVMs. While this might not have made a large difference, the nature of the two models is quite different and as such, it wouldn't be too surprising to see different results when algorithms are applied to them. The final point that needs to be addressed is that the simulation was ran only 50 times. Though this might seem like a good amount, with a larger simulation the results could change substantially and that should be recognized as well.

Ultimately, there might have been additional features that contributed to this result and if I were to run this type of experiment again I would definitely try to find a data set that had less NA's. I would also group the estimators by sample so that it would be easier to see how much the estimators varied and run the simulation as many times as possible. In all, it was still very interesting to apply different learning algorithms to these data sets to see how they would perform. Though these algorithms didn't perform exactly as was expected, they shed some light onto how we might expect them to perform in similar situations and suggest that additional work in this area might yield valuable information about the application of these estimators on a large scale.

Appendix A

R Code

A.1 Gapminder

```
load("GapminderData")
check <- function(x,y){
  match1 <- match(x,y)
  match2 <- sum(!is.na(match1))
  if(match2 > 0){
    final <- 1
  } else {
    final <- 0
  }
  return(final)
}

countrylist <- read.delim(file= '~/Developed Countries', header=FALSE,
stringsAsFactors = FALSE)
countrylist1 <- unlist(countrylist)
for(i in 1:nrow(g1)){
  g1$developed[i] <- check(g1$Country[i],countrylist1)
}
totallist  <- read.delim(file= '~/Country List', header=FALSE, stringsAsFactors =
FALSE)
totallist1 <- unlist(totallist)

#remove any observations that aren't actually countries

for(i in 1:nrow(g1)){
  g1$keep[i] <- check(g1$Country[i],totallist1)
}

g2 <- g1 %>% filter(keep==1)

g3 <- g2 %>% select(-`Income.share.held.by.highest.10. X2007`,-
`Total.15..unemployment.... X2005`,
              -`Agricultural.land....of.land.area. X2011`, -`Children.out.of.school..primary
X2011`,
              -`Children.out.of.school..primary X2011`, -
`Children.out.of.school..primary..female X2011`,
              -keep)

g3[is.na(g3)] <- 0


#build an svm
countrylist <- g3$Country
for(i in 1:nrow(g3)){
  g3$id[i] <- i
}
```

```
g3$developed <- as.factor(g3$developed)

#take random sample
bagerrC <- c()
ranbagerrC <- c()
estimatorerrC <- c()
acterrC <- c()
set.seed(5)
for(z in 1:50){
trainC=sample(1:nrow(g3), 60)
leftoutC <- g3[-trainC,]
C.t <- g3[trainC,]
country.train <- g3[trainC,]
svmfit <- svm(developed~.-id-Country, data=country.train, scale=FALSE)
svmfit
tune1 <- tune(svm, developed~.-id-Country, data=country.train, kernel="linear",
        ranges=list(cost=c(0.01, 0.1, 1, 5, 10)))
summary(tune1)

#Best performance is with aa cost parameter of 1. We find that the cross-validation
error rate is 0.1343773
#cross validation error rate
#bagging error rate
#svm bagging

vals <- sample(1:nrow(country.train),replace=TRUE,12000)
sam = matrix(vals,nrow=60,ncol=200)
sam[1,]

#bootstrap samples
samps <- list()
for(i in 1:200){
  samps[[i]] <- country.train[sam[,i],]
}
#build an svm for each sample

samsvmC <- list()
for(i in 1:200){
  data <- samps[[i]]
  if(!is.na(match(1,data$developed))){
  samsvmC[[i]] <- svm(developed~.-id-Country, data=data, scale=FALSE)
  }else{
    samsvmC[[i]] <- NA
  }
}
```

```
#find out of bag observations
oob <- list()
for(i in 1:200){
  oob[[i]] <- country.train[-sam[,i],]
}

#predict
results1 <- c()
for(i in 1:200){
  if(!is.na(samsvmC[[i]])){
  pre <- predict(samsvmC[[i]],oob[[i]])
  t <- table(pre,oob[[i]]$developed)
  results1[i] <- 1 - (t[1,1]+t[2,2])/nrow(oob[[i]])
  }else{
    results1[i] <- NA
  }
}
#bag error
BaggingErrorC <- mean(results1,na.rm=TRUE)

#Predicts 30% error with SVM
#randomized bootstrap bagging
#same as before except apply randomize to all bootstraps before running

ranC <- function(x){
  for(i in 1:60){
  toss <- sample(0:1,size = 1, prob = c(0.9,1))
  if(toss == 1){
    x[i,]$developed <- x[i,]$developed
  }
  else{
    if(x[i,]$developed == 1){
      x[i,]$developed <- 0
    }
    else{
      x[i,]$developed <- 1
    }
  }

  }
  return(x)
}


#set up randomization
```

```
randsamps <- list()
for(i in 1:200){
  randsamps[[i]] <- ranC(samps[[i]])
}


ransamsvm <- list()
for(i in 1:200){
  data <- randsamps[[i]]
  if(!is.na(match(1,data$developed))){
  ransamsvm[[i]] <- svm(developed~.-id-Country, data=data, scale=FALSE)
  }else{
    ransamsvm[[i]] <- NA
  }
}
ranresults <- c()
for(i in 1:200){
  if(!is.na(ransamsvm[[i]])){
  pre <- predict(ransamsvm[[i]],oob[[i]])
  t <- table(pre,oob[[i]]$developed)
  ranresults[i] <- 1 - (t[1,1]+t[2,2])/nrow(oob[[i]])
  }else{
    ranresults[i] <- NA
  }
}
RandomBagErrorC <- mean(ranresults,na.rm=TRUE)

#error rate


#.632 estimator
#Cycle through ids


tracker <- list()
for(i in 1:59){
  id <- country.train$id[i]
  counter <- 0
  error <- 0
  for(j in 1:200){
    if(is.na(match(id,samps[[j]]$id))){
      data <- samps[[j]]
      if(!is.na(match(1,data$developed))){
      counter = counter+1
      svmm <- svm(developed~.-id-Country, data=data, scale=FALSE)
      pre <- predict(svmm,country.train[i:(i+1),])
    t <- table(pre[1],country.train$developed[i])
    results <- (t[1,2]+t[2,1])
    error = error + results
    }
   }
   tracker[[i]] <- c(error,counter)
 }
}
#Add last value

 id <- country.train$id[60]
 counter <- 0
 error <- 0
 for(j in 1:200){
   if(is.na(match(id,samps[[j]]$id))){
    counter = counter+1
    data <- samps[[j]]
    svmm <- svm(developed~.-id-Country, data=data, scale=FALSE)
    pre <- predict(svmm,country.train[59:60,])
    t <- table(pre[2],country.train$developed[i])
    results <- (t[1,2]+t[2,1])
    error = error + results
   }
   tracker[[60]] <- c(error,counter)
 }

for(i in 1:60){
 error <- error + tracker[[i]][1]
 counter <- counter + tracker[[i]][2]
}
ErrC <- error/counter

svmmod <- svm(developed~.-id-Country, data=country.train, scale=FALSE)
pre <- predict(svmmod,country.train)
t <- table(pre,country.train$developed)
error2 <- 1 - (t[1,1]+t[2,2])/nrow(country.train)

svmbagerC <- function(x,y){
 add <- 0
 for(i in 1:200){
   results <- predict(x[[i]],y[[i]])
   tfin <- table(results, y[[i]]$developed)
   error <- (tfin[1,2]+tfin[2,1])
   add <- add + error
 }
 totalerror <- (add)/(50*200)
```

```
  return(totalerror)
}

EstimatorErrorC <- .368 * error2 + .632 * ErrC




remain <- g3[-trainC,]
prefinal <- predict(svmmod, remain)
tfinal <- table(prefinal,remain$developed)
ErrAct <- 1 - (tfinal[1,1]+tfinal[2,2])/nrow(remain)
ErrAct




acterrorC <- function(x,w){
  outcome <- c()
  feeder <- c()
  for(i in 1:(nrow(w)-1)){
    ob <- w[i:(i+1),]
    count <- 0
    for(j in 1:200){
      if(!is.na(x[[j]])){
      p <- predict(x[[j]], ob)[1]
      if(p==1){
        count = count + 1
      }
      }
    }
    feeder <- c(feeder,count)
    if(count >= 100){
      outcome[i] <- 1
    } else{
      outcome[i] <- 0
    }
  }
  i <- nrow(w)
  ob <- w[(i-1):i,]
  count <- 0
  for(j in 1:200){
    if(!is.na(x[[j]])){
    p <- predict(x[[j]], ob)[2]
    if(p==1){
      count = count + 1
    }
    }
  }
  if(count >= 100){
    outcome[i] <- 1
  } else{
    outcome[i] <- 0
  }
  feeder <- c(feeder,count)
  return(feeder)
}

predC <- acterrorC(samsvmC,leftoutC)
outcomeCtable <- table(predC,leftoutC$developed)
if(nrow(outcomeCtable)==1){
  ActualErrorC <- outcomeCtable[1,2]/120
} else{
  ActualErrorC <- (outcomeCtable[1,2] + outcomeCtable[2,1])/120
}


bagerrC <- c(bagerrC, BaggingErrorC)
ranbagerrC <- c(ranbagerrC, RandomBagErrorC)
estimatorerrC <- c(estimatorerrC,EstimatorErrorC)
acterrC <- c(acterrC,ActualErrorC)
}

#Bagging Summary
summary(bagerrC)
#Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#0.06649 0.14410 0.16920 0.16820 0.19460 0.26540
#Random Bagging
summary(ranbagerrC)
#Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#0.3586  0.4127  0.4315  0.4326  0.4527  0.4844
#Estimator Summary
summary(estimatorerrC)
#Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#0.04137 0.08911 0.10480 0.10740 0.12420 0.17700

#Actual Error
summary(acterrC)
#Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#0.1167  0.1583  0.1667  0.1652  0.1750  0.2167
```

A.2 ISLR OJ

```
bagerrOJ <- c()
ranbagerrOJ <- c()
estimatorerrOJ <- c()
acterrOJ <- c()
set.seed(4)
for(i in 1:50){
trainO=sample(1:1070, 200)
leftoutOJ <- OJ[-trainO,]
OJ.t <- OJ[trainO,]
for(i in 1:nrow(OJ.t)){
  OJ.t$id[i] <- i
}
traint <- sample(1:200,50, replace=FALSE)
OJ.test <- OJ.t[traint,]
OJ.train <- OJ.t[-traint,]

svmfit <- svm(Purchase~., data=OJ.train, scale=FALSE)
svmfit
tune1 <- tune(svm, Purchase~., data=OJ.train,
              ranges=list(cost=c(0.01, 0.1, 1, 5, 10)))

vals <- sample(1:nrow(OJ.train),replace=TRUE,12000)
sam = matrix(vals,nrow=150,ncol=200)


#bootstrap samples
sampsOJ <- list()
for(i in 1:200){
  sampsOJ[[i]] <- OJ.train[sam[,i],]
}
#build an svm for each sample
samsvmOJ <- list()
for(i in 1:200){
  data <- sampsOJ[[i]]
  samsvmOJ[[i]] <- svm(Purchase~., data=data, scale=FALSE)
}

oobOJ <- list()
for(i in 1:200){
  oobOJ[[i]] <- OJ.train[-sam[,i],]
}

#predict
results1 <- c()
for(i in 1:200){
  pre <- predict(samsvmOJ[[i]],oobOJ[[i]])
  t <- table(pre,oobOJ[[i]]$Purchase)
  results1[i] <- 1 - (t[1,1]+t[2,2])/nrow(oobOJ[[i]])
}
#bag error
BaggingErrorOJ<- mean(results1)

#Predicts 30% error with SVM
#randomized bootstrap bagging
#same as before except apply randomize to all bootstraps before running

ranOJ <- function(x){
  for(i in 1:150){
    toss <- sample(0:1,size = 1, prob = c(0.9,1))
    if(toss == 1){
      x[i,]$Purchase <- x[i,]$Purchase
    }
    else{
      if(x[i,]$Purchase == 'MM'){
        x[i,]$Purchase <- 'CH'
      }
      else{
        x[i,]$Purchase <- 'MM'
      }
    }

  }
  return(x)
}


#set up randomization
randsampsOJ <- list()
for(i in 1:200){
  randsampsOJ[[i]] <- ranOJ(sampsOJ[[i]])
}

ransamsvmOJ <- list()
for(i in 1:200){
  data <- randsampsOJ[[i]]
  ransamsvmOJ[[i]] <- svm(Purchase~., data=data, scale=FALSE)
}
ranresultsOJ <- c()
for(i in 1:200){
  pre <- predict(ransamsvmOJ[[i]],oobOJ[[i]])
  t <- table(pre,oobOJ[[i]]$Purchase)
  ranresultsOJ[i] <- 1 - (t[1,1]+t[2,2])/nrow(oobOJ[[i]])
```

```
}
RandomBagErrorOJ <- mean(ranresultsOJ)


#.632 estimator
#Cycle through ids




tracker <- list()
for(i in 1:149){
  id <- OJ.train$id[i]
  counter <- 0
  error <- 0
  for(j in 1:200){
    if(is.na(match(id,sampsOJ[[j]]$id))){
      counter = counter+1
      data <- sampsOJ[[j]]
      svmm <- svm(Purchase~., data=data, scale=FALSE)
      pre <- predict(svmm,OJ.train[i:(i+1),])
      t <- table(pre[1],OJ.train$Purchase[i])
      results <- (t[1,2]+t[2,1])
      error = error + results
    }
    tracker[[i]] <- c(error,counter)
  }
}
#Add last value

id <- OJ.train$id[150]
counter <- 0
error <- 0
for(j in 1:200){
  if(is.na(match(id,sampsOJ[[j]]$id))){
    counter = counter+1
    data <- sampsOJ[[j]]
    svmm <- svm(Purchase~., data=data, scale=FALSE)
    pre <- predict(svmm,OJ.train[149:150,])
    t <- table(pre[2],OJ.train$Purchase[i])
    results <- (t[1,2]+t[2,1])
    error = error + results
  }

  tracker[[150]] <- c(error,counter)
}
```

```
for(i in 1:150){
  error <- error + tracker[[i]][1]
  counter <- counter + tracker[[i]][2]
}
ErrOJ <- error/counter

svmmodOJ <- svm(Purchase~., data=OJ.train, scale=FALSE)
pre <- predict(svmmodOJ,OJ.train)
t <- table(pre,OJ.train$Purchase)
error2 <- 1 - (t[1,1]+t[2,2])/nrow(OJ.train)

svmbagerOJ <- function(x,y){
  add <- 0
  for(i in 1:200){
    results <- predict(x[[i]],y[[i]])
    tfin <- table(results, y[[i]]$Purchase)
    error <- (tfin[1,2]+tfin[2,1])
    add <- add + error
  }
  totalerror <- (add)/(200*200)
  return(totalerror)
}

EstimatorErrorOJ <- .368 * svmbagerOJ(samsvmOJ,sampsOJ) + .632 * ErrOJ

#Actual Error

remainOJ <- OJ[-trainO,]
remainOJ <- remainOJ %>% select(-id)
OJ.train <- OJ.train %>% select(-id)
svmmodOJ <- svm(Purchase~., data=OJ.train, scale=FALSE)
prefinal <- predict(svmmodOJ, remainOJ)
tfinal <- table(prefinal,remainOJ$Purchase)
ErrActOJ <- (tfinal[1,2]+tfinal[2,1])/nrow(remainOJ)




acterrorOJ <- function(x,w){
  outcome <- c()
  for(i in 1:nrow(w)){
  ob <- w[i,]
  count <- 0
  for(j in 1:200){
    p <- predict(x[[j]], ob)
    if(p=="MM"){
      count = count + 1
    }
  }
  if(count >= 70){
    outcome[i] <- "MM"
  } else{
    outcome[i] <- "CH"
  }
  }
  return(outcome)
}

pred <- acterrorOJ(samsvmOJ,leftoutOJ)

outcomeOJtable <- table(pred,leftoutOJ$Purchase)
if(nrow(outcomeOJtable)==1){
  ActualErrorOJ <- outcomeOJtable[1,2]/870
} else{
ActualErrorOJ <- (outcomeOJtable[1,2] + outcomeOJtable[2,1])/870
}

bagerrOJ <- c(bagerrOJ, BaggingErrorOJ)
ranbagerrOJ <- c(ranbagerrOJ, RandomBagErrorOJ)
estimatorerrOJ <- c(estimatorerrOJ,EstimatorErrorOJ)
acterrOJ <- c(acterrOJ,ActualErrorOJ)
}

#Bagging Stats
summary(bagerrOJ)
#Min. 1st Qu.  Median    Mean 3rd Qu.
#0.2987  0.3708  0.4058  0.4043  0.4330
#Max.
#0.5364

#Random Bagging Stats
summary(ranbagerrOJ)
#Min. 1st Qu.  Median    Mean 3rd Qu.
#0.4580  0.4852  0.4912  0.4902  0.4975
#Max.
#0.5034

#Estimator Stats
summary(estimatorerrOJ)
#Min. 1st Qu.  Median    Mean 3rd Qu.
```

```
#0.1890  0.2357  0.2596  0.2571  0.2776
#Max.
#0.3362

#Actual Error Stats
summary(acterrOJ)
#Min. 1st Qu.  Median    Mean 3rd Qu.
#0.3724  0.3851  0.3983  0.4037  0.4135
#Max.
#0.5172
```

Bibliography

[1] Baxter Tyson, Smith; "*Lagrange Multipliers Tutorial in the Context of SVM*", St. John's, Newfoundland, Canada 2004.

[2] Bradley, Efron, Robert, Tibshirani; "*Improvements on Cross-Validation: The .632+ Bootstrap Method*", Journal of the American Statistical Association, Vol. 92, No. 438, Jun.,1997, pp. 548-560

[3] Bradley, Efron; "*Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation*", Journal of the American Statistical Association, Vol. 78, No. 382, Jun.,1983, pp. 316-331

[4] Andrew, Ng; CS229 Lecture notes, SVM, Stanford

[5] Laurent, El Ghaoui; "*Convex Optimization and Applications*", 2012

[6] Robert, Berwick; "*An Idiot's guide to SVM (SVMs)*", http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf

[7] James, Gareth, et al. ; "*An Introduction to Statistical Learning*", SVM, Chapter 9, Spring, New York, 2013