Erasmus University

Bachelor Thesis: Econometrics and Operations Research

SVM Partitioned Decision Trees

Author:
Oluwatosin Fayoyin
(368524)

Supervisor:
Prof. dr. P.J.F. (Patrick) Groenen
Second assesor:
Prof. dr. (Denis) Fok

July 2, 2017

Abstract

This paper investigates the use of decision trees to improve the performance of a linear support vector machine. Specifically, it creates a decision tree and uses the results of an SVM prediction as a branching rule for the tree. When tested on a suit of 8 data sets, this hybrid methodology was seen to outperform the usual linear SVM on 3 of the data sets tested and in the other situations the performance of the two models were similar.

Contents

1 Introduction

In the field of Machine Learning, Support Vector Machines (SVM) are a class of supervised learning algorithms developed for the prediction of binary outcome variables. In supervised learning, a $m-$dimensional multivariate observation x is associated with a class label c. The objective is to "learn" a mathematical function f, that can be evaluated on the input x to yield a prediction of its class, c (Hahne et al, 2010). Alongside SVM, Decision Trees (DT) are also a class of supervised learning algorithms that are used for binary prediction. The ultimate goal in the development of classification algorithms such as SVM and DT is to obtain the best possible prediction. There is a growing base of support in the literature for the use of model combining to provide improved prediction performance (Abbott 1999).

The core idea behind the concept of model combining is to train several models from the same data set, or from samples of the same data set and combine the output predictions, typically by the use of a majority vote (Abbott 1999). Applications of model combining have been so successful that scholars such as Friedman et al. (2008) have referred to boosting (a form of model combining), as one of the most important recent developments in classification methodology. In this paper, a novel method for improving the performance of SVM is developed. Rather than applying a model combining approach, this paper utilizes a hybrid decision tree methodology that uses SVM to determine the optimal binary partition of the data space at each node of the decision tree. Specifically, in this tree, the root node is the entire data set. After running an SVM on this data set and making in sample predictions, instances that are predicted to be of the first class are sent along one branch of the tree while the other instances are sent along the other branch. These two new sets of instances are thus the children of the root node. Following this process, the new level of the tree is created in the same manner by determining the children's children. This process continues until an appropriate stopping criterion is reached. The prediction of new instances is then made depending on how a given instance i flows through the tree.

That said, this paper attempts to answer the following research question: "Can a decision tree that uses SVM to determine the partition of the data space at each node, outperform a single SVM in terms of prediction accuracy?"

There are multiple ways to formulate the SVM optimization problem. Currently all formulations fall under, either the primal or dual formulations and whether they use linear or nonlinear optimization. Generally speaking, the primal formulation of the SVM problem is easier to grasp than the dual formulation (Groenen et al, 2008) and non-linear SVMs prove to be computationally demanding when the number of instances is large (Chang et al, 2008). For this reason, the specific SVM classifier that is employed in this paper uses the primal formulation of the SVM problem and makes use of a linear optimization

approach based on iterative majorization. The iterative majorization approach to SVMs is outlined in Groenen et al. (2008) and this paper can be seen as an attempt to improve the performance of that majorization algorithm through the use of decision trees.

The remainder of this paper is structured as follows: Section 2 provides a literature review of the various ways in which practitioners have attempted to improve the performance of base models and the applications of DT to SVM; Section 3 gives a detailed outline of the SVM model; Section 4 describes the iterative majorization approach to SVM; Section 5 introduces and motivates a SVM approach to partitioning the data space and creating and the DT-SVM classifier; Section 6 shows the results of a numerical analysis of the predictive performance of the DT-SVM in comparison to a single SVM classifier and Section 7 brings together the major findings of the research and outlines further paths of possible inquiry for the use of DT in an SVM context.

2 Literature Review

The two major paradigms for the purposes of improving the performance of base models are bagging and boosting. In bagging, multiple bootstrap samples (with replacement) of the data set are created. The model in question is then trained on all these new samples and the results of a prediction from all the classifiers, trained on the different samples, are then aggregated to result in a final prediction (Brieman 1996). In boosting, each instance i in the data set is assigned a particular weight, initially, each instance is assigned an equal weight. Yet, on each round, the weights of incorrectly classified instances are increased so that the classifier subsequently focuses on the hard examples in the training set (Freund and Schapire 1999). In bagging, one needs to specify the number of bootstrap samples that are created. There is no general methodology to do this and Rokach (2010) has indicated that the performance of bagging is highly dependent on this parameter. Thus, we opt for exploring this DT hybrid approach as opposed to bagging. Additionally, boosting has been applied to many contexts with SVM. Hence, for the sake of novelty it is also set aside in favor of the aforementioned hybrid approach.

When a predictive model is built by integrating and combining multiple models, as in bagging, this is referred to as an ensemble methodology. The idea behind ensemble methodology is to weigh several classifiers and combine them in order to obtain a classifier that outperforms each one of them (Rokach 2010). Ensemble methodology was introduced to supervised learning when Tukey (1977) suggested combining two linear regression models. The first being a classic regression model on the training data and the second being a regression on the residuals of the original model. Since then, Hansen and Salamon (1990) demonstrated that ensemble methodology outperforms other methods for improving performance and training of Neural Networks. In relation to SVM, Kim

et al. (2003) applied bagging and boosting to simulated data on handwritten recognition and fraud detection. Their results show that the SVM ensemble based on bagging and boosting outperforms a single SVM in terms of classification accuracy by a great margin. Additionally, given that real world data sets are often imbalanced, Aijun and Huang (2006) applied an integrated sampling technique. This technique combines oversampling the minority class and under sampling the majority class to develop an ensemble of SVMs. Their results showed that this integrated sampling technique outperforms a single SVM and several other state of the art classifiers.

On the question of how to select and weigh the models that will feature in the ensemble, the literature indicates that diverse individual models make better ensemble predictions (Kittler, Matef and Matas 1998). Diversity means difference and models are different when their misclassified observations do not overlap. The intuition here being that when the models make the same errors, weighting the predictions of the individual models would add no benefit to the prediction accuracy because all the models will make the same wrong prediction.

In addition to bagging and boosting, there are other techniques for improving the performance of base models that focus on selecting the different explanatory variables to be used in the training phase. Specifically, these methods create multiple models by using different subsets of the total number of explanatory variables available. Salcedo and Whitley (1999) proposed a simple genetic algorithm for feature selection which involves exploring the space of all possible feature subsets and creating an ensemble based on them. Furthermore, Oliveira, Morita and Sabourin (2006) showed that in cases where the general error rates for the individual models are low, their feature selection method provided sizable improvements to the prediction accuracy of a single model. When compared to bagging and boosting, both feature selection methods where shown to perform better than these methods when classifiers have to work with very low error rates. The majorization approach to SVM has a problem with dealing with a large number of explanatory variable. As m grows each iteration of the majorization algorithm gets slower (Groenen et al, 2008). This makes feature selection methods infeasible for SVM-maj because feature selection methods work best when there is a large number of explanatory variables from which to construct the different classifiers.

Decision trees have also been applied to SVMs. However, they are mainly applied for multi-class SVM classification. The original SVM was created for the purpose of binary classification and its extension to a multi-class context is not straight forward (Dejan and Chorbev 2008). Hence, in order to extend it to multi-class classification researchers often opt for using an ensemble of binary classifiers. One-against-one and one against all are the two most popular strategies for multi-class SVM (Milgram, Cheriet and Sabourin 2006). One-against-one trains binary a SVM for each pair of classes that exist for the specific multi-class problem and one-against-all trains an SVM for each class, against

all the other classes (Milgram, Cheriet and Sabourin 2006). Another method has been implemented which uses binary decision trees. Here a decision tree is constructed by training a SVM hierarchically where each binary classification can contain multiple classes (Milgram, Cheriet and Sabourin 2006). Additionally, Chang et al. (2010) also developed a multi-class SVM based on a decision tree. However, their tree works by a binary split based on certain values of the explanatory variables in the feature space and was developed to improve the training time on large scale SVM problems and not to improve prediction accuracy. (Kumar and Gopal 2010) also developed a hybrid between SVM and DT. However their methodology was more DT based than SVM based in the sense that a DT is used for classifying most instances and an SVM is used only for classifying those crucial data points lying near the decision boundary. Given that we now have an overview of the current climate for improving the predictive performance of SVM. We now proceed by outlining the the theory that underlies SVM.

3 The SVM Loss Function

For the SVM classification problem, the data points are a set of instance-label pairs (x_i, y_i) where $i = 1...n$, $y_i \in \{1, -1\}$ and $X_i \in \mathbf{R}^m$. The matrix $\mathbf{X}$ holds the predictor variables and the vector $\mathbf{y}$ contains the class labels. The aim is to find a hyperplane $\mathbf{w}$ and an intercept c that separates the positive and negative instances with maximum margin, whilst penalizing the points inside the margin linearly (Pontil et al, 1998). Given an intercept c and hyperplane $\mathbf{w}$, the prediction for an instance i is given by:

$$q_i = c + \mathbf{x}_i^{'}\mathbf{w} \tag{1}$$

where $\mathbf{x}_i^{'}$ is a row of the matrix $\mathbf{X}$. To illustrate the idea of finding the maximal separating hyperplane, consider Figure 1 below where there are two predictor variables. The x-axis represents the first of our predictor variables while the y-axis represents the second. The pluses (+) in the figure denote positive instances while the circles (o) denote negative instances. Given certain values for w_1 and w_2, the points in the scatter plot can be projected onto a line, with the exact values of w_1 and w_2 determining the direction of that line. In the Figure 1a below, the dashed line is the projection line. The separating line is orthogonal to the projection line and the intercept c determines exactly where along the projection line that this separating line will be (Groenen et al, 2008).

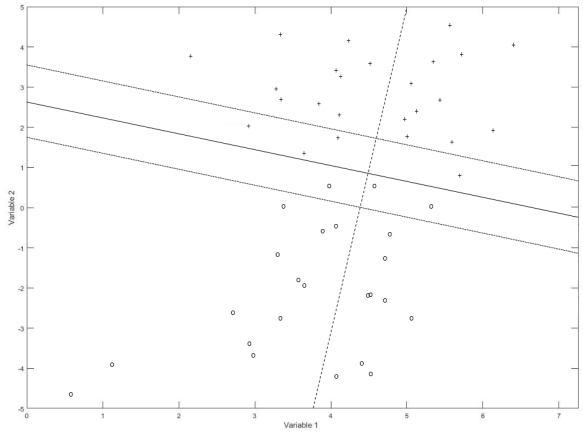

Figure 1: Projections of the positive and negative instances onto the line given by w_1 and w_2.

Hence, the vector $\mathbf{w}$ determines the projection line and the constant c determines the separation point of the projection line. The core of SVM is to choose these two values in such a way that the positive instances are well separated from the negative instances. The two lines that are parallel to the separation line are called the margin lines. They indicate all points that are projected onto the line at $q_i = 1$ and $q_i = -1$. All the points inside the margin are incorrectly predicted and are the data points that contribute to our total loss. The length between the margin lines is $2/\|\mathbf{w}\|$.

From this, we see that the central tenet of SVM lies in the way the loss function is constructed. Instances that are correctly predicted do not contribute to the total error whilst those that are incorrectly predicted contribute according to the magnitude of the error. So if i is a positive instance and $q_i \geq 1$, this yields a zero error. However, if $q_i < 1$ then the error is linear with $1 - q_i$. Similarly, when instance i corresponds to a negative class and $q_i \leq -1$, this also yields a zero error. Yet, if $q_i > -1$ then this instance contributes linearly to the total error with $1 + q_1$. Figure 2 shows the error functions for the two classes, given that the contribution of each error to the total loss is the magnitude of the error, this error function is called the absolute hinge error.

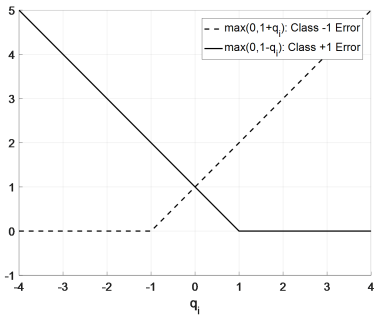

Figure 2: the absolute hinge error for positive (dashed) and negative (solid) instances

As $\mathbf{w}$ controls how close the margin lines are to each other, and instances within the margin lines are the incorrectly predicted instances, there is a bias in to model toward making $\|\mathbf{w}\|$ as small as possible. To account for this, a penalty term dependent on $\|\mathbf{w}\|$ is added to the loss function.

Let G_1 and G_{-1} denote the set of class 1 and -1 instances. Then the SVM loss function is given by:

$$L_{\text{SVM}} = \sum_{i \in G_1} \max(0, 1 - q_i) + \sum_{i \in G_{-1}} \max(0, 1 + q_i) + \lambda \mathbf{w}^{'} \mathbf{w} \tag{2}$$

where $\lambda > 0$ determines the strength of the penalty term. Note, equation (2) can also be expressed in the following form:

$$L_{\text{SVM}} = \sum_{i=1}^{n} \max(0, 1 - y_i q_i) + \lambda \mathbf{w}^{'} \mathbf{w}. \tag{3}$$

The minimum of L_{SVM} is a global solution due to the function being convex in c and $\mathbf{w}$, being coercive and being bounded below by zero (Groenen et al, 2008). The instances i that contribute to the total loss are those that project on, or onto the wrong side of the margin. Such instances are called support vectors as they form the fundamental components of the SVM solution. Note, that the support vectors are not known in advance and all n instances need to be present when analysis is conducted.

In addition to the Absolute hinge error given here, Groenen et al. (2008) also proposed two alternative error functions that can be applied in the context of SVMs. These are the quadratic and huber hinge errors. The quadratic hinge error simply squares the absolute hinge error, yielding the loss function:

$$L_{\text{SVM-Q}} = \sum_{i \in G_1} \max(0, 1 - q_i)^2 + \sum_{i \in G_{-1}} \max(0, 1 + q_i)^2 + \lambda \mathbf{w}^{'} \mathbf{w}. \tag{4}$$

The Huber hinge on the other hand is characterized by a linearly increasing error if the error is large, a smooth quadratic error for errors between zero and the linear part and a zero error for objects that are correctly predicted. The smoothness is governed by a value $k \geq -1$ and as K approaches -1, the huber hinge behaves like the absolute hinge error whilst, when k approaches 1 the huber hinge behaves like the quadratic hinge error. Thus, the huber hinge can be seen as a compromise between the absolute and quadratic hinges (Groenen et al, 2008). The huber hinge loss function is given by:

$$L_{\text{SVM-H}} = \sum_{i \in G_1} h_1(q_i) + \sum_{i \in G_{-1}} h_{-1}(q_i) + \lambda \mathbf{w}^{'} \mathbf{w} \tag{5}$$

where

$$h_{-1}(q_i) = \begin{cases} (1/2)(k+1)^{-1} \max(0, q_i + 1)^2 & \text{if } q_i \leq k \\ q_i + 1 - (k+1)/2 & \text{if } q_i \geq k \end{cases}$$

$$h_1(q_i) = \begin{cases} 1 - q_i - (k+1)/2 & \text{if } q_i \leq -k \\ (1/2)(k+1)^{-1} \max(0, 1 - q_i)^2 & \text{if } q_i \geq k \end{cases}.$$

Now that the idea underlying SVM has been presented, we proceed by outlining the SVM-maj algorithm developed to find the minimum of the SVM loss function through an iterative majorization framework.

4 Iterative Majorization

The principle behind majorization is to iteratively minimize a complex function $f(\mathbf{q})$ via a simpler majorization function $g(\mathbf{q}, \bar{\mathbf{q}})$ that is dependent on $\mathbf{q}$ and the previously known estimate $\bar{\mathbf{q}}$. The majorization function has to satisfy three essential requirements. First, at the supporting point $\bar{\mathbf{q}}$, $f(\bar{\mathbf{q}}) = g(\bar{\mathbf{q}}, \bar{\mathbf{q}})$. Second, the majorizing function g should never be below f, that is $f(\bar{\mathbf{q}}) \leq g(\bar{\mathbf{q}}, \bar{\mathbf{q}})$. Finally, $g(\mathbf{q}, \bar{\mathbf{q}})$ should be simple, preferably linear or quadratic in $\mathbf{q}$.

In majorization, we start with a supporting point $\bar{\mathbf{q}}$ and then construct a majorizing function $g(\mathbf{q}, \bar{\mathbf{q}})$ for the function f at the point $\bar{\mathbf{q}}$ and then minimize the majorizing function to find the new supporting point $\mathbf{q}^*$. The new point $\mathbf{q}^*$ is the minimum of $g(\mathbf{q}, \bar{\mathbf{q}})$ and serves as the new supporting point for a new majorization function $g(\mathbf{q}, \bar{\mathbf{q}})$ where $\bar{\mathbf{q}} = \mathbf{q}^*$. This intuition is captured in equation (6) below:

$$f(\mathbf{q}^*) \leq g(\mathbf{q}^*, \bar{\mathbf{q}}) \leq g(\bar{\mathbf{q}}, \bar{\mathbf{q}}) = f(\bar{\mathbf{q}}). \tag{6}$$

This shows that iteratively, the majorization function never increases f and usually decreases it. By repeating these iterations, we obtain a monotonically non-increasing sequence of function values. For convex f, after a sufficient number of iterations, the IM algorithm stops at a global minimum (Groenen et al,

2008).

For the problem of minimizing our respective SVM loss functions, the majorizing function necessary to minimize the absolute, quadratic and huber hinges have been derived in Groenen et al. (2008) and are given below for the two classes.

$$\begin{aligned} g_{-1}(q_i) &= a_{-1i} - 2b_{-1i} + c_{-1i} \\ g_1(q_i) &= a_{1i} - 2b_{1i} + c_{1i} \end{aligned} \tag{7}$$

The variables are defined as follows:

$$a_i = \begin{cases} max(\delta, a_{-1i}) & \text{for } i \in G_{-1} \\ max(\delta, a_{1i}) & \text{for } i \in G_1 \end{cases}, b_i = \begin{cases} b_{-1i} & \text{for } i \in G_{-1} \\ b_{1i} & \text{for } i \in G_1 \end{cases}, c_i = \begin{cases} c_{-1i} & \text{for } i \in G_{-1} \\ c_{1i} & \text{for } i \in G_1 \end{cases}$$

where δ is a small positive constant that is smaller than the convergence criterion ϵ. It replaces a_{-1i} or a_{1i} for the cases where $\bar{q} = -1$ or $\bar{q} = 1$ respectively. When we combine the two classes, this leads to the following majorization inequality:

$$L_{\text{SVM}}(c, \mathbf{w}) \leq \sum_{i=1}^{n} a_i q_i^2 - 2\sum_{i=1}^{n} b_i q_i + \lambda \sum_{i=1}^{m} w_j^2. \tag{8}$$

Recall that, $q_i = c + \mathbf{x}_i^{'}\mathbf{w}$. We can deal with the optimization using one variable by adding an extra column of ones to the matrix $\mathbf{X}$. Let $\mathbf{v} = [c, \mathbf{w}]$ so that $\mathbf{q} = \mathbf{X}\mathbf{v}$. Now (2) can be majorized as:

$$\begin{aligned} L_{\text{SVM}}(c, \mathbf{w}) &\leq \sum_{i=1}^{n} (\mathbf{x}_i^{'}\mathbf{v})^2 - 2\sum_{i=1}^{n} b_i \mathbf{x}_i^{'}\mathbf{v} + \lambda \sum_{j=2}^{m} v_j^2 \\ &= \mathbf{v}^{'}\mathbf{X}^{'}\mathbf{A}\mathbf{X}\mathbf{v} - 2\mathbf{v}\mathbf{X}^{'}\mathbf{b} + c_m + \lambda \mathbf{v}^{'}\mathbf{P}\mathbf{v} \\ &= \mathbf{v}^{'}(\mathbf{X}^{'}\mathbf{A}\mathbf{X} + \lambda\mathbf{P})\mathbf{v} - 2\mathbf{v}^{'}\mathbf{X}^{'}\mathbf{b} + c_m \end{aligned} \tag{9}$$

where $\mathbf{A}$ is a diagonal matrix with the elements a_i on the diagonal, $\mathbf{b}$ is a vector with elements b_i, $c_m = \sum_{i=1}^{n} c_i$ and $\mathbf{P}$ is the identity matrix except for $p_{11} = 0$. Differentiating the last line of the above equation with respect to $\mathbf{v}$ yields a system of equations linear in $\mathbf{v}$:

$$(\mathbf{X}^{'}\mathbf{A}\mathbf{X} + \lambda\mathbf{P})\mathbf{v} = \mathbf{X}^{'}\mathbf{b}. \tag{10}$$

The update $\mathbf{v}^{+}$ solves this set of linear inequalities. Extra computational efficiency can be obtained for the quadratic and huber hinge errors for which $a_{-1i} = a_{1i} = a$ for all i and this a does not depend on $\bar{\mathbf{q}}$. In these cases the update simplifies to:

$$\mathbf{v}^{+} = (a\mathbf{X}^{'}\mathbf{X} + \lambda\mathbf{P})^{-1}\mathbf{X}^{'}\mathbf{b}. \tag{11}$$

Thus, the $m \times n$ matrix $\mathbf{S} = (a\mathbf{X}^{'}\mathbf{X} + \lambda\mathbf{P})^{-1}\mathbf{X}$ can be computed once and stored in memory so that the update simply amounts to: $\mathbf{v}^{+} = \mathbf{S}\mathbf{b}$.

5 SVM Partitioning

DT are a means to express a classification rule for a given instance i using the values of the predictor variables for that instance (Quinlan 1986). The instance i contains the vector $\mathbf{x}_i^{'}$ of predictor variables and what the decision tree does is show how to classify this instance based on rules that need to be successively applied to the predictor variables of this instance. As an example, consider a simple case with only two predictor variables and two decision rules. A decision tree corresponding to this situation is given in Figure 3 below.

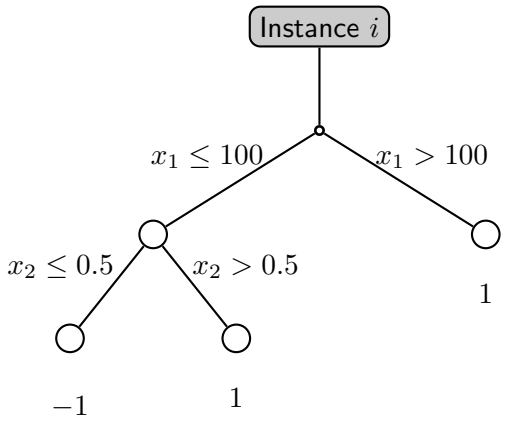

Figure 3: Illustration of how a given test instance flows through a normal decision tree

What this tree says, is that for a new instance i, we first ask if the variable x_1 is less than or equal to 100. If that is not the case we label i as a class 1 instance. If it is less than or equal to 100 then we ask whether the variable x_2 is less than or equal to 0.5. If it is, then i is assigned to class -1 and otherwise it is assigned to class 1. This is how the decision tree provides us with a classification rule. In this situation, the data space is partitioned as shown in Figure 4 below. In Figure 4, the horizontal axis corresponds to the variable x_1 and the vertical axis corresponds to the variable x_2.

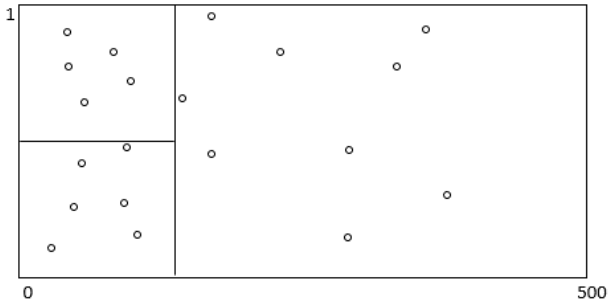

Figure 4: An illustration of how the data space is partitioned using a normal decision tree

In Figure 4 above, a circle represents an instance and every circle to the right of the vertical line is assigned a class 1 label. For those instances to the left of the vertical line, those that are below the horizontal line are also assigned a class -1 label whilst those that are above it are assigned a label of class 1. Thus, in a normal decision tree application, each node represents an attribute (predictor variable) based test. So in general, the partitioning procedure for the tree is based on the the values of one predictor variable at a time. This results in a data space that can only be partitioned according to block formations. Which is rather restrictive in the sense that there could be more complex relationships that could be used to discriminate between classes at a given node.

The SVM partitioning procedure works in a slightly different way. As opposed to the partitioning rule being based on an attribute based test, the partitioning rule is based on an SVM based test. Each node will contain different values for the parameter vector $\mathbf{v}$. At a given node N a prediction q is made at that node, which determines whether the instance i should be of class 1 or class -1. The instance i then moves along the branch for which it was predicted, onto the successive node that corresponds to that branch. This node will then have a different set of parameter estimates $\mathbf{v}$ that allow us to make another prediction for the instance i. Given the results of this second prediction, the instance i then follows along the branch for which it was predicted. Onto yet another node with its own set of parameter estimates. This process continues until a leaf node is reached and no more predictions can be made. The corresponding class for i is then the class to which it was allocated at the node before the leaf. This procedure is illustrated in Figure 5 below which has the same structure as Figure 5 with the only difference lying in the branching rule.

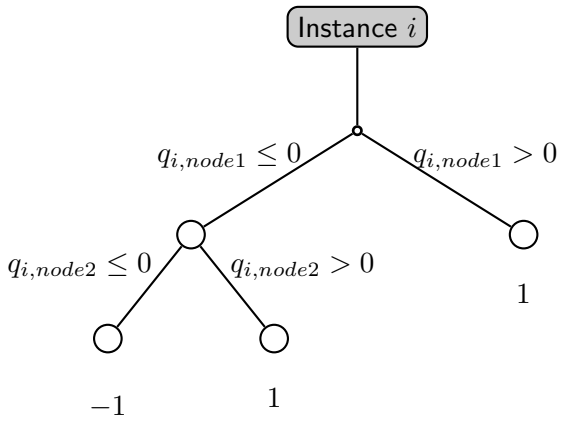

Figure 5: Illustration of how a given test instance flows through an SVM partition based decision tree

In Figure 5 above, the estimates q are denoted with the subscript indicating which node the prediction belongs to as a means to emphasize that each node contains a different set of parameter estimates $\mathbf{v}$ which lead to different predictions. This has the effect of removing the restriction that the data space has to be partitioned according to blocks and essentially frees the shape of the partitioned data points, allowing this shape to be anything. The resulting shape is determined by the values of the parameter estimates $\mathbf{v}$ for each node. Figure 6 below provides an illustration of this. Recall from Figure 1a that the values of w_1 and w_2 determined the direction of the projection line and the intercept c which is orthogonal to it, shows us how to partition the data space. Here, each estimate $\mathbf{v}$ can be seen to do just that. They each provide their own partitioning line of the data space, which we can see in Figure 6. Notice here, that the shape of these partitions is not limited to block formations. This allows us to capture more complex relationships between the predictor variables and class labels, allowing us to better discriminate between classes at each node.

What is now left is to give an account of how to develop this tree. The resulting tree will simply hold the parameter estimates $\mathbf{v}$ at each node allowing us to classify new instances. Using the training data, the tree is developed in a top down manner. It allows two child nodes to grow from each node that is not a leaf. Each node N contains a set of instances I, if N is not a leaf then two child nodes are grown from N that contain a binary split of the instances I in N. Thus, the two child nodes contain their own set of instances $I_{\text{leftChild}}$ and $I_{\text{rightChild}}$ such that $I = I_{\text{leftChild}} \cup I_{\text{rightChild}}$. The root node contains the full training data set. The vector of parameter estimates $\mathbf{v}$ for this node is the result of training an SVM on this full training data set. The class -1 predictions go the the left child and the class 1 predictions go to the right child. A SVM is then trained on the instances $I_{\text{leftChild}}$ and $I_{\text{rightChild}}$ resulting in the parameter

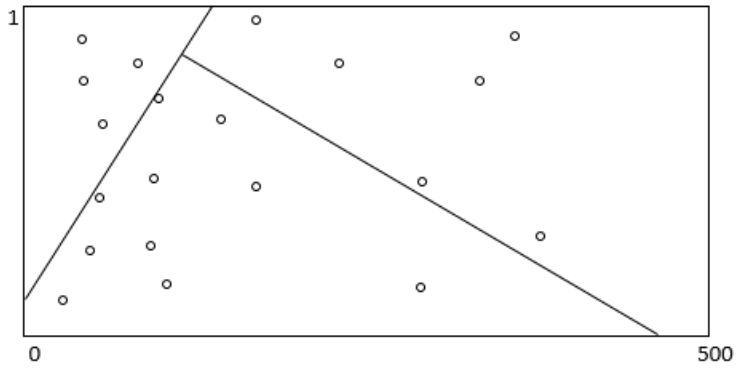

Figure 6: An illustration of how the data space is partitioned using a SVM partitioning rule

estimates $\mathbf{v}_{\text{leftChild}}$ and $\mathbf{v}_{\text{rightChild}}$. A further prediction is then made for all the instances in the left and right child nodes. The binary split for the left and right child nodes then make up the instances that flow to the grandchildren of the root node. This process continues until no more nodes can be further divided, that is, an appropriate stopping criterion is reached. Here, it is important to note that for each training of the SVM at a given node, 5-fold cross-validation is used to determine the optimal λ for the specific set of data which corresponds to that node. This ensures that we obtain better parameter estimates for the node an thus allows us to make better splits. To perform the cross validation, we run the SVM model on a grid of λ values ($\lambda = 2^p$ for $p = -15, -14.5, -14, ..., 7.5, 8$ where $2^{-15} = 0.0030518$ and $2^8 = 15$) and choose the λ that results in the highest prediction accuracy.

We stop splitting a node N when one of the following criteria is reached: (1) the number of instances i that flow to the node is equal to the number of explanatory variables, (2) all the instances i that are present in the node are assigned the same class label. This means all instances in the node are predicted to be either of class 1 or class -1 with no instance assigned to a different class label from the rest. Thus we have a situation of homogeneous predictions.

The first stopping criterion is motivated by the fact that, at a node where the number of instances is less than the number of explanatory variables, parameter estimates $\mathbf{v}$ will still need to be determined for that node. If the number of instances n is less than the number of explanatory variables m this means we are using a smaller number of data points to determine a larger number of estimates. Generally speaking, this often leads to poor estimates, especially when dealing with new cases. Thus, to ensure that each SVM partition in the tree is useful and informative this stopping criterion is developed. The second stopping criterion is motivated by the fact that, when a node contains observations of a

single class, then all subsequent predictions of the majorization algorithm will correspond to that class. This implies that we cannot have a binary split but more importantly, it shows those instances in the node most likely belong to the given class and hence no performance gains can be made by assigning any of the instances i in the given node N to a different class.

Thus, all in all, this DT-SVM partitioning procedure has two phases. First, it has to grow the decision tree. This tree is grown from the training data and the resulting nodes of this tree will each contain a vector of parameter estimates $\mathbf{v}$ that determine how a given test instance i flows through the tree. Secondly, there is the process of taking test instances and making predictions for them. This is done by an algorithm that determines how a given instance will flow through the tree from root to leaf, where the leaf determines the corresponding predicted class label. The pseudo code for such an algorithm is given in Algorithm 1 below.

Algorithm 1: Pseudo code for classifying new instances using the constructed decision tree

```
Data: the vector x_i' of predictor variables for a test instance i
Result: the last value of prediction
initialization:
prediction = 0;
currentNode = root;
canSplit = 1;
while canSplit equals 1 do
    CurrentP = make prediction using parameter estimates of this node;
    prediction = currentP;
    go to left or right child node depending on prediction;
    if we can branch at this node then
        return to top of while loop;
    else
        canSplit = 0;
    end
end
```

6 Numerical Analysis

To investigate the performance of this DT-SVM partitioning methodology, the prediction accuracy of the tree classifier is analyzed in comparison to a single SVM classification. The data is analyzed using data gathered from the UCI repository and the homepage of the LIBSVM software. The data sets are chosen according to: the number of observations in the data sets; the ratio of class labels; the degree of sparsity of the data and the ratio of instances to attributes. The precise information about each data set is found in Table 1 below.

Dataset	Source	n	n1	n-1	m	Sparsity
Australian	UCI	690	307	383	14	20.04
Heart_stalog	UCI	270	120	150	13	0.00
Hepatitis	UCI	155	123	31	19	39.68
Sonar	UCI	208	97	111	60	0.07
Voting	UCI	434	167	267	16	45.32
Diabetes	LIBSVM	768	500	268	8	0.15
Breat_cancer_w	UCI	699	458	383	14	20.04

Table 1: characteristics of the data sets used for the numerical analysis

The experiments were applied using an absolute and quadratic hinge. The Huber hinge is disregarded because according to Groenen et al. (2008), the Huber hinge is a compromise between the absolute and quadratic hinge in terms of the way the loss function is constructed, hence we expect that the performance of the tree created using the Huber hinge will not be drastically different from one created using either the absolute or quadratic hinge. Consequently, it was excluded from the analysis. All experiments were implemented using MATLAB 2016a, on a 3.4GHz AMD processor with 4GB of RAM under windows 7.

In these experiments, the performance indicator of interest is the prediction accuracy of the tree in comparison to the single SVM. To do this, half of the data set is used to train the SVM and the other half is used for testing. Specifically, the odd numbered instances are used for training while the even numbered instances are used for testing. Run times are not of interest because its is obvious that by creating the tree and running more than one SVM, the run time of the tree classifier will be longer than the single SVM classifier. Furthermore, because this is a tree, the loss of the tree cannot be compared to the loss of a single SVM because the tree contains multiple SVM models This would be akin to comparing a vector of losses to a single loss. Finally, the major goal of all classification models is to predict accurately, thus it makes sense that only prediction accuracy is considered in this analysis. Table 2 below provides the percentage of correctly predicted instances for each data set when the tree is applied using a quadratic hinge, whilst Table 3 provides the same results for the

absolute hinge error.

Dataset	Tree	Single SVM	Number of nodes
Australia	85.8	84.6	4
Heart_stalog	82.9	80.0	7
Hepatitis	59.7	67.5	6
Sonar	71.1	71.1	3
Voting	89.9	89.9	4
Diabetes	72.4	78.4	26
Breast_cancer_w	84.5	85.7	9

Table 2: Prediction accuracies for the quadratic hinge on out of sample forecasts

Dataset	Tree	Single SVM	Number of nodes
Australia	82.3	85.8	7
Heart_stalog	82.3	82.3	3
Hepatitis	85.3	85.3	3
Sonar	76.9	76.9	3
Voting	94.4	94.4	3
Diabetes	69.3	77.4	7
Breast_cancer_w	94.5	94.5	3

Table 3: Prediction accuracies for the absolute hinge on out of sample forecasts

Tables 2 and 3 above provide some interesting findings. First of all, we notice that the tree implemented using the absolute hinge error never outperforms the single SVM. Yet, the tree created using the quadratic hinge error outperforms the single SVM on some occasions. Additionally, we also observe that the number of active nodes in the quadratic tree are often more than those of the tree created using the absolute hinge error. As a matter of fact, it is often the case that the tree created using the absolute hinge error has exactly 3 nodes. This means, only one split was made from the root node resulting in only 3 active nodes in the tree. This also explains why the results for this tree is often the exact same as those from a single SVM, this is because only one prediction is made in the tree and this prediction corresponds to the one that will be made if the entire data set is used. Upon investigation, it was observed that the reason why this often happens is because, when an additional SVM is run on all the instances that were predicted to be of a certain class in the previous node, the predictions for these instances doesn't change, prompting the tree then, to stop at that point. For the results of the tree constructed using the quadratic hinge. The results show that this tree outperforms the single SVM on two occasions, it under performs on 3 occasions and there is a tie in 2 situations. As in the absolute hinge, the reason for the draws is due to the fact that the trees constructed using this data set end up with 3 active nodes. This means only one

prediction is made in such trees and this prediction corresponds to that which would be made if the entire data set was used. Voting is the only data set where the performance is the same but the tree had more than 3 active nodes. In the other situations, we observe that the tree can outperform the single SVM, yet it does not do this consistently. Table 2 also shows that, in the situations where the tree does under perform, the margin of this difference is larger than in the situations where the tree outperforms. This means, when the tree performs bad, it performs really bad, but when it performs well it performs moderately well. This indicates that there might be certain types of data sets that are not properly suited for this type of methodology.

Another possible reason for the results observed might have to do with the number of instances in each data set. All the data sets used in the experiments so far have less than 800 instances. This leaves less than 400 instances to be used to build the tree. As the tree also partitions the data set even more and runs multiple SVMs on these reduced data sets we can see how this will affect the parameter estimates further down the tree. To investigate this, a larger data set is experimented on. The results of this experiment are given in Table 4 below where the tree is constructed using the quadratic hinge error.

Dataset	Source	n	n1	n-1	m	Tree	Single SVM
Occupancy	UCI	8143	1688	6455	6	98.9	98.6

Table 4: Comparison on large data set

For this data set the results show that the tree outperforms the single SVM yet, once again, the margin of difference is low. Specifically, the tree predicted 4024 out of 4071 instances correctly while the single SVM made 4015 correct predictions. However, in this case it is not so surprising that the margin of difference is small. This is because here, the general hit rate is already very close to 100%. Although, the fact that the tree performs better even when there is barely any room for improvement is a good finding, highlighting great room for potential with this methodology.

7 Conclusion

The research conducted had one goal. Namely, to determine whether a hybrid DT-SVM methodology, which uses a SVM based partitioning rule, could outperform a single SVM in terms of prediction accuracy. When evaluated on the basis of that goal, the findings were mixed. Firstly, it was observed that in general, this decision tree methodology works best for the quadratic hinge error. This is because, the absolute hinge error suffers from the problem of making homogeneous class predictions very quickly. So from the first prediction, running an SVM again on the instances in one branch does not change the class predictions already made in the previous node. Hence, the trees that are created usually

only have one level and in situations where there is more than one level, the performance of the tree is poor. For the quadratic hinge error the results are more hopeful. Out of all the 8 data sets that were tested on, the tree outperforms the single SVM on 3 of them, ties on 2 of them and under performs on 3 data sets.

In sum, these results are interesting because they indicate that this DT-SVM partitioning methodology can be used to provide performance gains over the standard SVM. The main issue however is that it does not consistently do so. More research will have to be conducted as to which situations it is that this DT-SVM methodology performs better. Specifically, one may investigate the issue of large data sets and imbalances in the class distribution to assess how these may affect the performance of this methodology. Another possible area of further inquiry may be the issue of over fitting. The Hepatitis and Diabetes data sets have a relatively large number of active nodes in comparison to the number of instances in the data set and to other data sets with the same number of instances. So, although the Australia data set has close to the same number of instances as that of Diabetes, the number of active nodes it has is 4 whilst Diabetes has 26. The tree methodology also outperforms in the Australia data set but under performs in the Diabetes. This indicates that the tree in the Diabetes probably over fitted the data in the training phase and hence performed worse in the testing, which was done out of sample. In general, over fitting is a danger with all decision trees and it seems to happen here. Hence, for further research, one may consider developing trees that have a relatively moderate amount of active nodes in comparison to the the number of instances. So for example, consider that the Diabetes data set has more active nodes than the Occupancy data set which has only 6 active nodes yet far more instances. I look forward to seeing the results of such exploratory research.

8 References

Abbott, D. W. (1999, July). Combining models to improve classifier accuracy and robustness. In Proceedings of Second International Conference on Information Fusion, Fusion'99 (Vol. 1, pp. 289-295).

Breiman, L. (1996). Bagging predictors. Machine learning, 24(2), 123-140.

Chang, F., Guo, C. Y., Lin, X. R., & Lu, C. J. (2010). Tree decomposition for large-scale SVM problems. Journal of Machine Learning Research, 11(Oct), 2935-2972.

Freund, Y., Schapire, R., & Abe, N. (1999). A short introduction to boosting. Journal-Japanese Society For Artificial Intelligence, 14(771-780), 1612.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors).

The annals of statistics, 28(2), 337-407.

Groenen, P. J., Nalbantov, G., & Bioch, J. C. (2008). SVM-Maj: a majorization approach to linear support vector machines with different hinge errors. Advances in data analysis and classification, 2(1), 17-43.

Guerra-Salcedo, C., & Whitley, D. (1999, July). Genetic approach to feature selection for ensemble creation. In Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1 (pp. 236-243). Morgan Kaufmann Publishers Inc.

Hahne, F., Huber, W., Gentleman, R., & Falcon, S. (2010). Bioconductor case studies. Springer Science & Business Media.

Hansen, L. K., & Salamon, P. (1990). Neural network ensembles. IEEE transactions on pattern analysis and machine intelligence, 12(10), 993-1001.

Kim, Hyun-Chul, Shaoning Pang, Hong-Mo Je, Daijin Kim, and Sung Yang Bang. "Constructing support vector machine ensemble." Pattern recognition 36, no. 12 (2003): 2757-2767.

Kittler, J., Hatef, M., Duin, R. P., & Matas, J. (1998). On combining classifiers. IEEE transactions on pattern analysis and machine intelligence, 20(3), 226-239.

Kumar, M. A., & Gopal, M. (2010). A hybrid SVM based decision tree. Pattern Recognition, 43(12), 3977-3987.

Liu, Y., An, A., & Huang, X. (2006, April). Boosting Prediction Accuracy on Imbalanced Datasets with SVM Ensembles. In PAKDD (Vol. 6, pp. 107-118).

Madzarov, G., Gjorgjevikj, D., & Chorbev, I. (2008). Multi-Class classification using support vector machines in binary tree architecture. In International Scientific Conference (pp. 413-418).

Milgram, J., Cheriet, M., & Sabourin, R. (2006, October). "One against one" or "one against all": Which one is better for handwriting recognition with SVMs?. In Tenth international workshop on frontiers in handwriting recognition. Suvisoft.

Oliveira, L. S., Morita, M., & Sabourin, R. (2006). Feature selection for ensembles applied to handwriting recognition. International Journal on Document Analysis and Recognition, 8(4), 262-279.

Quinlan, J. R. (1986). Induction of decision trees. Machine learning, 1(1),

81-106.

Rokach, L. (2010). Ensemble-based classifiers. Artificial Intelligence Review, 33(1), 1-39.

Tukey, J. W. (1977). Exploratory data analysis.