Tree and Support Vector Machine Classification

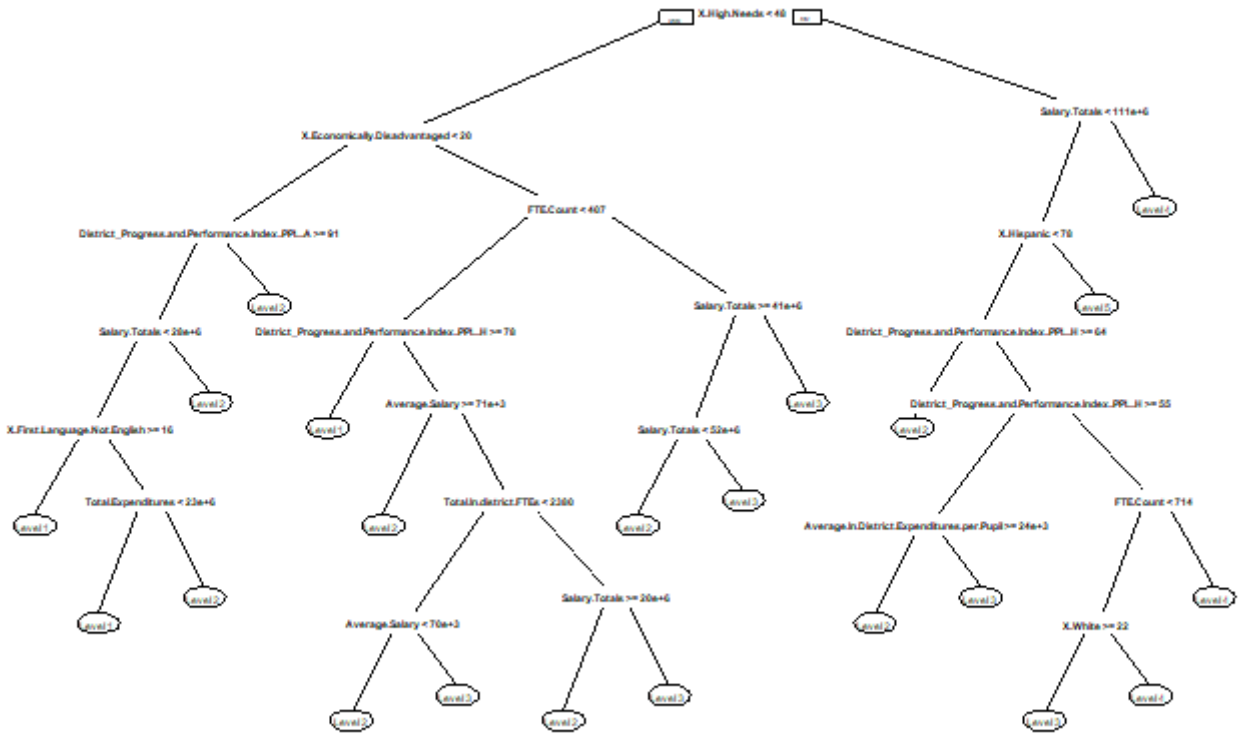
Ryan Wahl[1], Joseph Kim[2]

1 Summary

It is often difficult to determine what metrics contribute the most to school performance. We have gathered data from the Massachusetts Public Schools system including salary, class size, etc. in order to identify the variables that are most indicative of whether or not a school in Massachusetts is meeting the standards determined by the state. Massachusetts Public Schools rank each of its schools on a scale from 1 to 5. A score of 1 indicates that a school is meeting the standards. Any other score, $2-5$, indicates that a school is not demonstrating adequate performance with varying degrees of proficiency, 5 being the worst. These levels are strictly determined by standardized test scores. We would like to know if other variables actually correlate to achieving better test scores through tree and support vector machine classification methods.

2 Tree Classification

An obvious choice to determine the important variables that impact the rankings of the schools are by using a decision tree. Our response variable, the ranking of the school, is a categorical variable with levels 1 - 5. Our goal is to predict this ranking using the other variables included in our MPS data set [1]. To carry out this classification, we're going to use the *rpart* package in R to visualize the tree better. To begin, we create a full tree with the *minsplit* parameter set to 5 and run the function. The full tree can be visualized here:

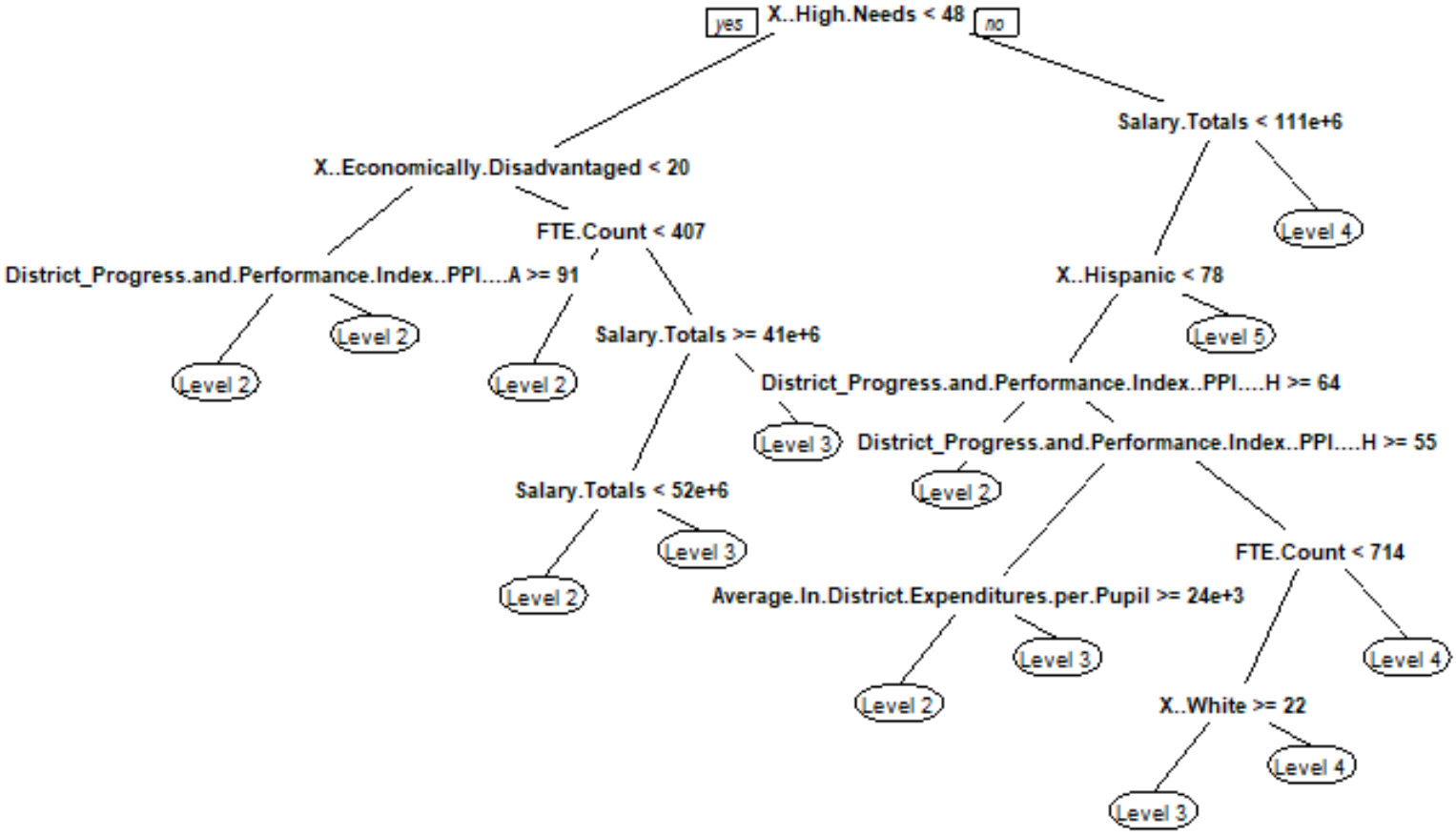


See Appendix A for Code

The full tree is large with many nodes. Lets inspect the variables that the model has included in the tree...

Variable name (full tree)	Variable name (full tree)
Average In District Expenditure per Pupil	Total In District FTEs
Average Salary	Number of Economically Disadvantaged Students
District Progress and Performance Index, All Students	Number of First Language Not English Students
District Progress and Performance Index, High Needs Students	Number of High Needs Students
FTE Count	Number of Hispanic Students
Salary Totals	Number of White Students
Total Expenditures	

There are a lot of variables in this model, leading us to believe that a pruned tree would only contain variables that are essential in classifying a school's ranking. It is also necessary to prune the tree to combat over-fitting our model and to reduce the chance that we are over-fitting the training data. The pruned tree looks much nicer:



See Appendix A for Code

Lets also look at the variables included in the pruned model...

Variable name (pruned tree)	Variable name (pruned tree)
Average In District Expenditures Per Pupil	Number of Economically Disadvantaged Students
District Progress and Performance Index, All Students	Number of High Needs Students
District Progress and Performance Index, High Needs Students	Number of Hispanic Students
FTE Count	Number of White Students
Salary Totals	

Here we can see that many of the nodes in the full tree have been pruned to create a tree that is much easier to interpret. There aren't many nodes that have a lot of children, so the hope here is that this pruned tree is going to predict out-of-sample data better than the full tree which is prone to overfitting. In total we created 5 trees to compare for our model. A full tree, a pruned full tree, and a tree created using cross validation. These trees were created using the standard R functions *tree*(), *snip.tree*(), and *cv.tree*(). The other 2 trees were created using the *rpart*() function and the *snip.rpart*() function. The trees were created using *rpart* mainly for their desirable visual output. The next step in tree classification is to assess how well each model predicts values for out-of-sample data points. We can do this by

predicting new values using each of the different models that we've created, and then compare these predictions to the true value of this new data.

Tree type	Out of sample misclassification error
Full tree	0.144
Pruned tree	0.234
CV Tree	0.144
Rpart Full tree	0.128
Rpart Pruned tree	0.157

See Appendix A for Code

The results from calculating the out of sample misclassification error for each tree model are intriguing. Our highest misclassification rate comes from the pruned tree using the default *snip.tree*() function in R. The pruned *rpart* tree had the second highest misclassification error with an error rate of 0.157, which is slightly better than the pruned tree that was not created using the *rpart* function. Whether or not we used the *rpart* function, both full trees had better predictive accuracy than their pruned tree counterparts. The full tree and the cross-validation determined tree predicted with the same error. This leads us to believe that the full trees were not actually overfitting our data but capturing splits that were beneficial in predicting the levels of the public school rankings. Each model was tested on data that the models had not seen before; the data was split into testing and training sets after it was cleaned. Our best model had misclassification error of just 0.128 which is excellent for data that the model has not seen yet.

3 Support Vector Machine (SVM) Background

3.1 Theory

The basic theory underlying SVM involves separating data through the use of hyperplanes. For our purposes, we will only be considering linear SVM, but this can be generalized to other forms of classification depending on the kernel function one chooses. The goal of linear SVM is to maximize the margin around the decision boundary to the closest data points in order to minimize the chance of misclassification of any future data points. In most real world applications the data is not completely separable. Thus, it is necessary to have a loss function that accounts for some misclassification around the decision boundary.

To begin to understand the maths behind this intuition, we begin by defining a plane in $\mathbf{R}^3$ as follows,

$$Ax + By + Cz + d = 0$$

refer to [8]

We can generalize this in terms of the dot product of a weight vector, $\mathbf{w}$, and a variable vector $\mathbf{x}$.

$$\mathbf{w}^t\mathbf{x} + b = 0 \tag{1}$$

$$w_i^t x_i + b = 0 \qquad i = 1, 2, 3, \ldots, p$$

refer to [8]

Note that the weight vector corresponds to A, B, and C while the variable vector corresponds to x, y, and z for $p = 3$, such that $w_i^t x_i + b = 0 \in \mathbf{R}^p$. For linear SVM $p = 2$, so the hyperplane is actually a line in $\mathbf{R}^2$. From here on, we will use the index i with the assumption that $p = 2$.

We now need to define a way of classifying points above and below the line according to some margin. First, we define the distance from a point,
(x_1, y_1, z_1) to a plane, $Ax + By + Cz + d = 0$ as

$$distance = \frac{|Ax_1 + By_1 + Cz_1 + d|}{\sqrt{A^2 + B^2 + C^2}}$$

refer to [3]

As before, we can transform this equation into its vector form

$$distance = \frac{|\mathbf{w}^t\mathbf{x} + b|}{\|\mathbf{w}\|} \tag{2}$$

see [7]

For a *hard margin*, it is convention to set the closest point to the boundary as 1, $|\mathbf{w}^t\mathbf{x} + b| = 1$. Thus, we have

$$margin = \frac{1}{\|\mathbf{w}\|}$$

Note that,

$$total\ margin = \frac{2}{\|\mathbf{w}\|}$$

since the absolute value involves the *margin* above and below the hyperplane

as shown in [7]

The intuition behind setting $|\mathbf{w}^t\mathbf{x} + b| = 1$ is that we need some way of defining how close we want the points on the boundary line to be away from the hyperplane. We could have chosen a different constraint, but it turns out that setting the expression to 1 is useful for scaling and solving large SVMs. Now that we have defined a *hard margin*, we can also define a classifying scheme.

First, we define $f(\mathbf{x})$ such that

$$f(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + b$$

as shown in [2]

By convention, if a point is above the hyperplane, it is labeled as 1. If a point is below the hyperplane, it labeld as -1. Thus, for the pair (x_i, y_i),

$$y_i = \left\{ \begin{array}{ll} 1, & \text{for } w^t x_i + b > 0 \\ -1, & \text{for } w^t x_i + b < 0 \end{array} \right\}$$

Given that the closest point to the boundary is defined as $|\mathbf{w}^t\mathbf{x}+b| = 1$, only the points above and below the *hard margin* will be classified. It follows,

$$y_i(w^t x_i + b) \geq 1 \tag{3}$$

as shown in [2]

Geometrically, for a hyperplane $\mathbf{w}^t\mathbf{x} + b = 0$, the data points must fall on or above $\mathbf{w}^t\mathbf{x} + b = 1$ and on or below $\mathbf{w}^t\mathbf{x} + b = -1$ in order to be correctly classified, according to the constraint in (3). However, this is rarely the case for real data which cannot be fully separated into two distinct groups. In order to account for points that may fall below the margin, on the hyperplane, or on the opposite side of the hyperplane, which would be a misclassification, we must incorporate a cost function that defines a *soft margin* [2]. Thus, we have

$$y_i(w^t x_i + b) \geq 1 - \xi_i \qquad \xi_i \geq 0 \tag{4}$$

as shown in [2]

Note that for $0 < \xi_i \leq 1$, would be classified correctly but would be some distance less than the margin on one side of the hyperplane. For $\xi_i > 1$, the point would be misclassified [2].

Having defined the margin we want to maximize, we can now define the optimization problem as follows,

$$\max_{\mathbf{w}} \frac{2}{\|\mathbf{w}\|} \Longleftrightarrow \min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^t\mathbf{w}$$

We convert the maximization problem into its equivalent minimization counterpart for notational simplicity. After incorporating the error and constraint,

$$\min \frac{1}{2}\mathbf{w}^t\mathbf{w} + C\sum_{i=1}^{m} \xi_i \tag{5}$$

$$s.t. \qquad y_i(\mathbf{w}^t x_i + b) \geq 1 - \xi_i \qquad \xi_i \geq 0$$

refer to [2]

C is simply a tuning parameter that adjusts the relative weighting between having a smaller margin and ensuring that most margins are at least 1[2].

This problem is not easy to solve due to the added constraint. However, we can incorporate the constraint into our objective function, (5), by taking the Lagrangian. This allows us to optimize only along the boundary defined by our constraint in terms of the Lagrange multipliers, greatly simplifying our problem. The Lagrangian is as follows,

$$\mathcal{L}(w, b, \xi, \alpha) = \frac{1}{2}\mathbf{w}^t\mathbf{w} + C\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}\alpha_i[1 - \xi_i - y_i(\mathbf{w}^t x_i + b)] \tag{6}$$

refer to [2]

Note that α_i represents the Lagrange multipliers. Now we can minimize over w, b, and ξ, while keeping α constant.

$$\min_{w,b,\xi} \mathcal{L}(w, b, \xi, \alpha) \tag{7}$$

refer to [2]

Setting the partial derivatives equal to 0,

$$\frac{\partial\mathcal{L}}{\partial w} = 0 \Longrightarrow \mathbf{w} - \sum_{i=1}^{m}\alpha_i y_i \mathbf{x_i} = 0 \Longrightarrow \mathbf{w} = \sum_{i=1}^{m}\alpha_i y_i \mathbf{x_i}$$
$$\frac{\partial\mathcal{L}}{\partial b} = 0 \Longrightarrow -\sum_{i=1}^{m}\alpha_i y_i = 0$$
$$\frac{\partial\mathcal{L}}{\partial \xi} = 0 \Longrightarrow \sum_{i=1}^{m} C - \alpha_i = 0$$

as shown in [6]

We can rearrange (6) such that it is clear how we can apply our partial derivative constraints.

$$\frac{1}{2}\mathbf{w}^t\mathbf{w} - \sum_{i=1}^{m}\alpha_i y_i(\mathbf{w}^t x_i) + \sum_{i=1}^{m}(C - \alpha_i)\xi_i + \sum_{i=1}^{m}\alpha_i - b\sum_{i=1}^{m}\alpha_i y_i \tag{8}$$

Applying our partial derivative constraints,

$$\frac{1}{2}\mathbf{w}^t\mathbf{w} - \sum_{i=1}^{m}\alpha_i y_i(\mathbf{w}^t x_i) + \sum_{i=1}^{m}(C - \alpha_i)\xi_i + \sum_{i=1}^{m}\alpha_i - b\sum_{i=1}^{m}\alpha_i y_i$$
$$\frac{1}{2}\mathbf{w}^t\mathbf{w} - \sum_{i=1}^{m}\alpha_i y_i(\mathbf{w}^t x_i) + \cancel{\sum_{i=1}^{m}(C - \alpha_i)\xi_i}^{\,0} + \sum_{i=1}^{m}\alpha_i - b\cancel{\sum_{i=1}^{m}\alpha_i y_i}^{\,0}$$
$$\frac{1}{2}\mathbf{w}^t\mathbf{w} - \sum_{i=1}^{m}\alpha_i y_i(\mathbf{w}^t x_i) + \sum_{i=1}^{m}\alpha_i$$
$$\frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j\langle\mathbf{x_i}\cdot\mathbf{x_j}\rangle - \sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j\langle\mathbf{x_i}\cdot\mathbf{x_j}\rangle + \sum_{i=1}^{m}\alpha_i$$
$$\sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j\langle\mathbf{x_i}\cdot\mathbf{x_j}\rangle$$

refer to [6]

Note that we must include an additional box constraint such that $\sum_{i=1}^{m}(C - \alpha_i)\xi_i$ cannot be negative; otherwise, the error would go to $-\infty$ [2].

Putting all of this together, we want to maximize α over Θ given

$$\Theta(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j\langle\mathbf{x_i}\cdot\mathbf{x_j}\rangle \tag{9}$$

$$s.t. \quad 0 \le \alpha_i \le C \tag{10}$$

$$\sum_{i=1}^{m}\alpha_i y_i = 0 \tag{11}$$

as shown in [2]

This is referred to as the Lagrangian *dual* form and requires us to also check the Karush-Kuhn-Tucker conditions.

$$\begin{aligned}\alpha_i = 0 &\implies y_i(\mathbf{w}^t x_i + b) \geq 1 \\ \alpha_i = C &\implies y_i(\mathbf{w}^t x_i + b) \leq 1 \\ 0 < \alpha_i < C &\implies y_i(\mathbf{w}^t x_i + b) = 1\end{aligned}$$

With the optimal α, we can solve for the optimal weights.

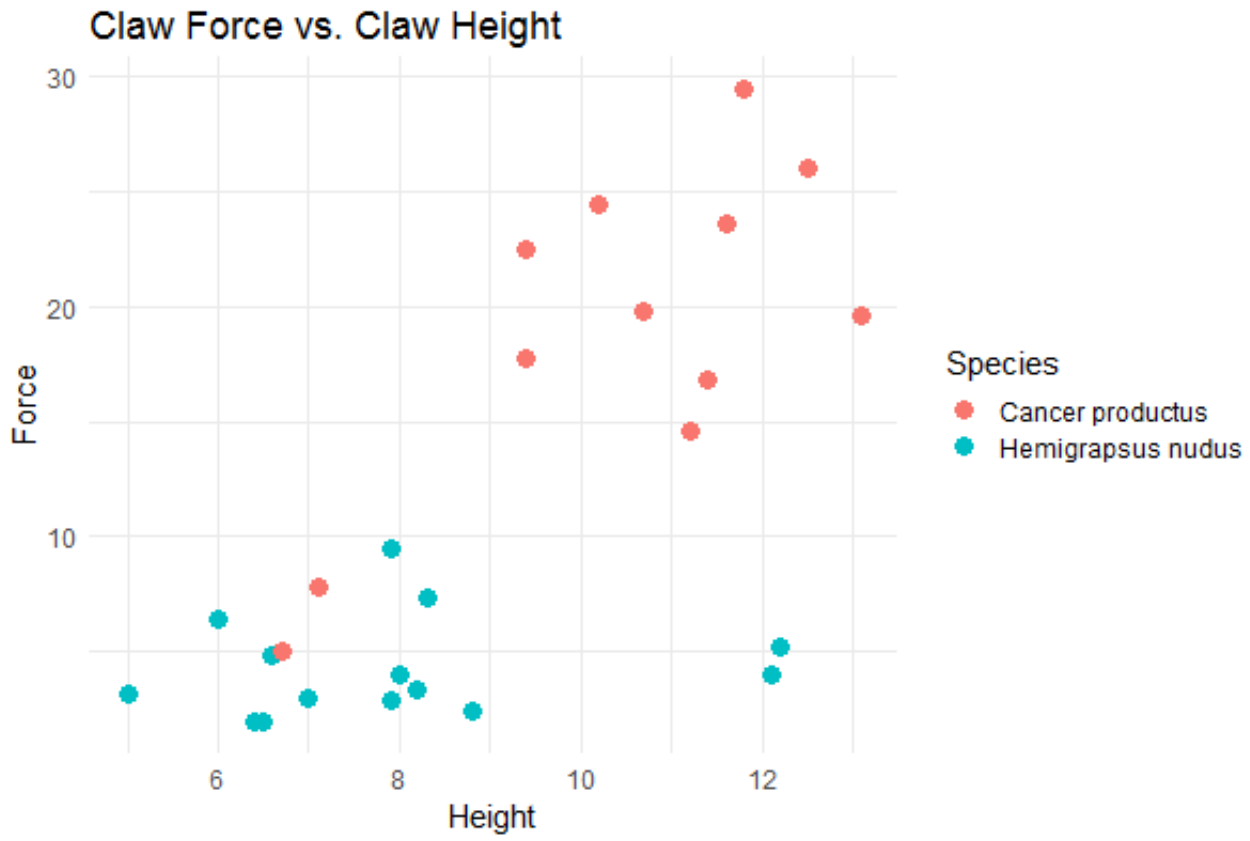
$$w = \sum_{i=1}^{m} \alpha_i x_i^t y_i \tag{12}$$

refer to [2]

There are of course other methods of formulating linear SVM instead of using the *dual* form. This is also just a brief overview of the ideas behind SVM from basic geometry with relatively simple optimization concepts to more complex optimization problems with added constraints. Solving for the optimal α can be achieved by the Sequential Minimal Optimization (SMO) Algorithm, which is used to solve the *dual* form. We will not go into the details of this algorithm for the sake of brevity. Instead, we will just apply the algorithm to an example dataset and solve for the optimal α, giving the optimal weights for the hyperplane in $\mathbf{R}^2$. More details of the SMO and the pseudo-code behind the algorithm can be found in the references section [4].

3.2 Example

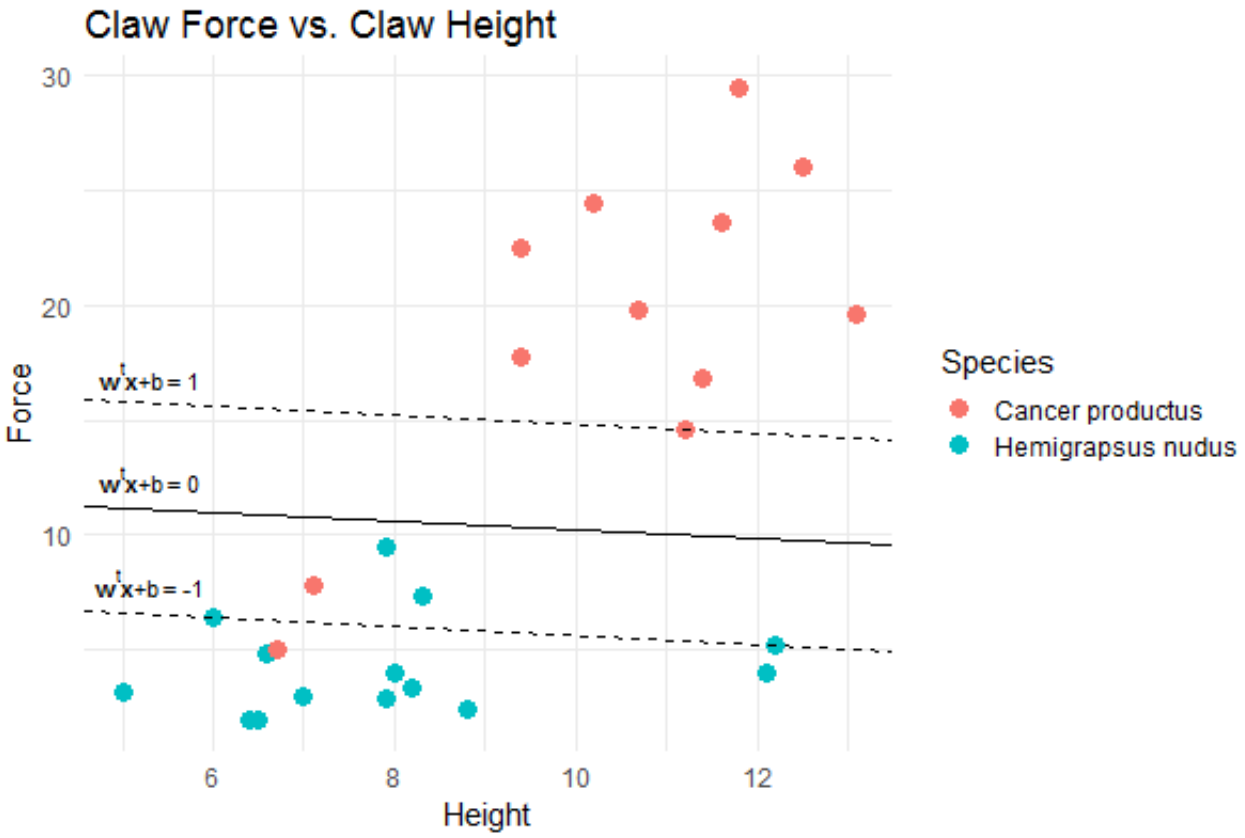
Below is a plot from the Sleuth3 ex0722 dataset.



See Appendix B.1 for Code

Intuitively, we can see a clear division between the two species, but there are two points of the "Cancer productus" class that are likely to be misclassified as they are grouped among most of the "Hemigrapsus nudus" points.

After running the SMO algorithm, we recovered the weights and intercept to give us the following plot with the linear decision boundary.



See Appendix B.1 for Code

The plot shows how the SMO algorithm attempts to maximize the number points above and below the $\mathbf{w}^t\mathbf{x} + b = 1$ and $\mathbf{w}^t\mathbf{x} + b = -1$ lines, respectively. Furthermore, the summary statistics reveal what we would expect concerning the misclassified points. Note that the code used to calculate the decision boundaries lines was adapted from here [5].

	Cancer productus (**Predicted**)	Hemigrapsus nudus (**Predicted**)
Cancer productus (**Actual**)	10	2
Hemigrapsus nudus (**Actual**)	0	14

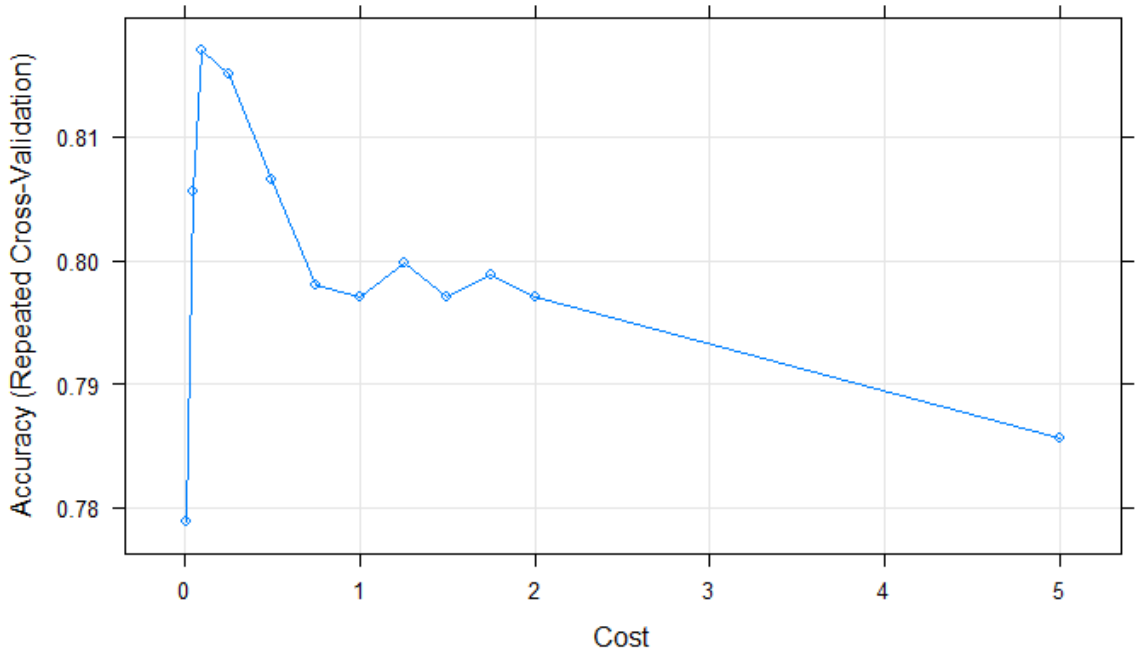
RMSE: 0.3698 | **Correct**: 92.3077 % | **Incorrect**: 7.6923 %

See Appendix B.1 for Code

The linear SVM model with the SMO optimization resulted in a misclassification of two points, predicted to be "Hemigrapsus nudus" but were actually "Cancer productus".

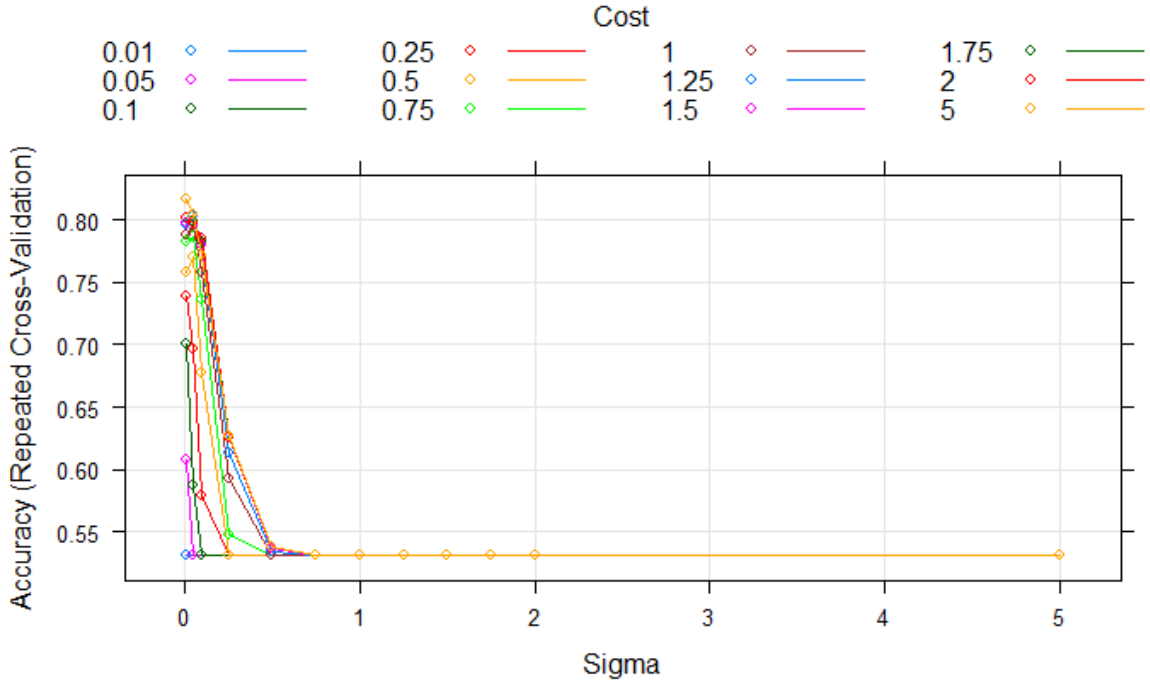
4 Support Vector Machine (SVM) Classification

Another way to classify the different rankings of the school is to use SVM classification. To carry out the SVM classification we are going to use the *caret* library in R. Using the *train*() function in the library we are able to start by fitting a linear SVM model to the data. We want to make sure we're accounting for many different tuning parameters as the data is not linearly seperable because it is so complex. We pass in a grid of different tuning parameters to find the optimal value. The function calculates the optimal hyperplane to separate the data and runs through the different tuning parameters to find the optimal value.



See Appendix B.2 for Code

Repeated cross-validation was used to determine the accuracy of the different tuning parameters. The final C value used for the model was 0.1 as shown from the graph. When this linear SVM model was used to predict our out-of-sample testing data, it achieved a respectable misclassification rate of .177. This is the second to last performer of all of our models, but it is not a bad model. We now want to try nonlinear SVM on our data to see if that classification method will create a better model. The difference with the nonlinear SVM model is the addition of another tuning parameter, sigma. In the same way that we passed in different values of C for the linear case to find the optimal tuning parameters, we are going to do the same with C and sigma.



See Appendix B.2 for Code

As can be seen from the graph, the model that had the highest accuracy was the model with parameters $C = 5$ and $\sigma = .01$. The nonlinear svm model had a misclassification rate of .166 which is slightly better than the linear SVM classification but still ranked second to worst out of all of the classification models built for this project. One of the main advantages CART has over SVM is the ability to interpret the splits, leafs, and nodes in the output tree. In high dimensions this does not become any more difficult. A huge drawback to trees is the potential to overfit the data.

However, we did not see much overfitting as our full trees had better accuracy than the pruned trees. Using SVM, there is not a great way to visualize the hyperplane that is created in such high dimensional space. It is much easier to just tune C and σ and create graphs of the different accuracies of the tuning parameters. It would be great to see how the algorithm was able to separate the data but unfortunately that is not very feasible. Our intuition as to why CART performed better than SVM is that the data is just not easily linearly separable. There are data points that are inside the soft margin that are being misclassified and is lowering our predictive accuracy. If the data was able to be more linearly separated than we would expect SVM's accuracy to improve.

5 Conclusion

From performing Tree and SVM classification, we observed that other secondary factors do affect the overall ranking of a public school in Massachusetts in terms of academic performance. The number of high needs students was the most influential variable in determining the level of a particular school. Furthermore, higher numbers of economically disadvantaged students and Hispanic students had a tendency to be categorized in the lower level schools. The tree classification also revealed that teacher salary totals did not play a huge role in determining the ranking of a school. In some cases, if salary totals were too high, then the level of a school had a tendency to be lower. This seems to be correlated with the full-time employee count. Higher numbers of full-time employees corresponded to lower level schools, indicating that larger schools have a tendency to do worse than smaller schools. Our results suggest, students with more burdens, whether mental or economic, who are attending larger schools, tend to perform worse on their standardized test. It was difficult to gain much insight through SVM, since there were many dimensions in the data, and it doesn't have a visual guide for anything more than three dimensions. Further research would need to be conducted to determine if the key variables we have identified are actually causing lower test scores, thereby lowering the level of the school, or if they are simply just correlations in the data.

References

[1] N. Dalziel, *Massachusetts public schools data.* URL: https://www.kaggle.com/ndalziel/massachusetts-public-schools-data.

[2] M. I. Jordan, *UC Berkeley Statistics, Lecture Notes: SoftMargin SVM*, 2004. URL: https://people.eecs.berkeley.edu/ jordan/courses/281B-spring04/lectures/lec6.pdf.

[3] D. Q. Nykamp, *Distance from point to plane. From Math Insight.* URL: https://mathinsight.org/distance_point_plane.

[4] J. Platt, *Sequential minimal optimization: A fast algorithm for training support vector machines*, (1998).

[5] quantsignals, *Learning kernels svm.* URL: https://www.r-bloggers.com/learning-kernels-svm/.

[6] D. Sontag, *MIT Computer Science, Lecture Notes: Support Vector Machines*, 2014. URL: http://people.csail.mit.edu/dsontag/courses/ml14/slides/lecture2.pdf.

[7] O. Veksler, *Western University Computer Science, Lecture Notes: Pattern Recognition*, 2004. URL: http://www.csd.uwo.ca/ olga/Courses/CS434a$_5$41a/$Lecture$11.pdf.

[8] E. W. Weisstein, *Plane. From MathWorld–A Wolfram Web Resource.* URL: http://mathworld.wolfram.com/Plane.html.

A Tree Classification Code

```
##Creating the full tree
MPStree <- tree(District_Accountability.and.Assistance.Level ~ ., data = train, mincut = 5)
plot(MPStree, lwd = 2)
text(MPStree, cex = .5)
summary(MPStree)
(Same idea, just using the rpart library, code included in file)
#Pruning the tree
MPS_snip <- snip.tree(MPStree, c(4, 12))
plot(MPS_snip, lwd = 2)
text(MPS_snip, cex = .7)
summary(MPS_snip)
(same idea, just using the rpart library, code included in file)
#Using CV to get the best tree
MPS_tree_cv = cv.tree(MPStree, K = 90)
plot(MPS_tree_cv)
MPS_tree_prune <- prune.tree(MPStree, best = 16)
plot(MPS_tree_prune)
text(MPS_tree_prune)
summary(MPS_tree_prune)
#Comparing all of the models...
#Full tree
fullPredict <- predict(MPStree, newdata=test, type="class")
fullFalse <- fullPredict != test$District_Accountability.and.Assistance.Level
tableFullFalse <- mean(fullFalse)
#Pruned tree
prunePredict <- predict(MPS_snip, newdata=test, type="class")
pruneFalse <- prunePredict != test$District_Accountability.and.Assistance.Level
tablePruneFalse <- mean(pruneFalse)
#CV tree
cvPredict <- predict(MPS_tree_prune, newdata=test, type="class")
cvFalse <- cvPredict != test$District_Accountability.and.Assistance.Level
tableCVFalse <- mean(cvFalse)
(Same idea, just using the rpart trees, code included in file)
#Table of prediction error for various trees
titles = c("Full_tree", "Pruned_Tree", "CV_Tree", "Rpart_Full_Tree", "Rpart_Pruned_Tree")
error = c(tableFullFalse, tablePruneFalse, tableCVFalse, tableFullRPartFalse,
tablePrunedRPartFalse)
pandoc.table(cbind("Tree_type" = titles,"Out_of_sample_misclassification_error" = error))
```

B SVM Classification Code

B.1 Example SVM

```
# create plot
plot <- ggplot(df, aes(Height, Force, col=Species)) + geom_point(size=4)
+ theme_minimal(base_size = 15) + ggtitle("Claw_Force_vs._Claw_Height")
plot
# apply SMO algorithm
x <- cbind(df$Height, df$Force)
y <- df$Species
smo_fit <- ksvm(x, y,type="C-svc",C = 100, kernel="vanilladot",scaled=c())
smo_stat <- SMO(Species ~., data = df)
# recover parameters from alpha values
w <- colSums(coef(smo_fit)[[1]] * x[SVindex(smo_fit),])
b <- b(smo_fit)
# plot svm decision boundary
svm_plot <- ggplot(df, aes(Height, Force, col=Species)) + geom_point(size=4)
+ theme_minimal(base_size = 15) + ggtitle("Claw_Force_vs._Claw_Height")
+ geom_abline(slope = -w[1]/w[2], intercept = b/w[2])
+ geom_abline(slope = -w[1]/w[2], intercept = (b+1)/w[2], linetype = 2)
+ geom_abline(slope = -w[1]/w[2], intercept = (b-1)/w[2], linetype = 2)
+ annotate("text", x = 5.3, y = 17, label = TeX("$\\mathbf{w}^t\\mathbf{x}+b_=_1$"))
```

```
    + annotate("text", x = 5.3, y = 12.5, label = TeX("$\\mathbf{w}^t\\mathbf{x}+b = 0$"))
    + annotate("text", x = 5.3, y = 8, label = TeX("$\\mathbf{w}^t\\mathbf{x}+b = -1$"))
    svm_plot
    # get summary statistics
    summary(smo_stat)
    # get confusion matrix statistics
    confusion_matrix <- data.frame("Cancer productus (Predicted)" = c(10, 0),
    "Hemigrapsus nudus (Predicted)" = c(2, 14))
    rownames(confusion_matrix) <- c("Cancer productus (Actual)", "Hemigrapsus nudus (Actual)")
    kable(confusion_matrix, "latex")
```

B.2 Full SVM

```
svm_Linear <- train(District_Accountability.and.Assistance.Level ~ ., data = train, method = "
                    trControl=trctrl,
                    tuneLength = 5)
test_Pred <- predict(svm_Linear, newdata = test)
confusionMatrix(test_Pred, test$District_Accountability.and.Assistance.Level)
svm_Linear_Grid <- train(District_Accountability.and.Assistance.Level ~ ., data = train, metho
                         trControl=trctrl,
                         tuneGrid = grid,
                         tuneLength = 10)
test_pred_grid <- predict(svm_Linear_Grid, newdata = test)
confusionMatrix(test_pred_grid, test$District_Accountability.and.Assistance.Level)
svm_Radial <- train(District_Accountability.and.Assistance.Level ~ ., data = train, method = "
  trControl=trctrl,
  tuneGrid = grid,
  tuneLength = 10)
test_pred_Radial <- predict(svm_Radial, newdata = test)
confusionMatrix(test_pred_Radial, test$District_Accountability.and.Assistance.Level)
```