Assignment 1, Predict 454

From: John Hokkanen

To: Dr. Chad Bhatti

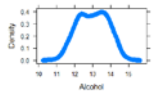
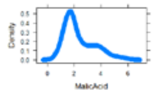
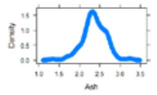
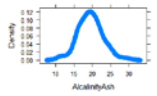
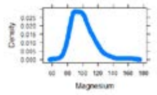
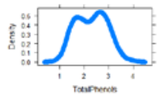
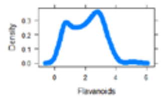
Date: November 1, 2017

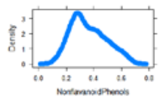
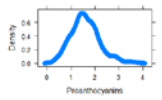
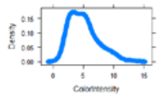
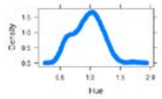
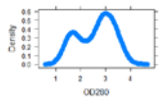
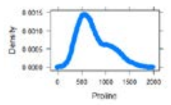
## Summary

The assignment is to create several multiclass models to predict a wine cultivar from a set of chemical features. Three non-parametric model types have been selected for the analysis: random forest, neural network, and support vector machine. As set forth below, any one of the various model types appear to be able to classify this dataset quite well, but for the reasons noted, the recommended model choice would be a tuned support vector machine followed by a random forest model.

## Data Check

The dataset is comprised of 178 records of 13 chemical constituents with one classifier variable of three wine cultivars. There are no missing data elements. Other than alcohol, which appears to be a percentage, the measurement units (e.g., parts per million) are unknown. They are all real numbers except the two integer variables noted and the factor response:

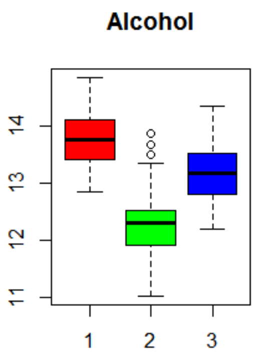
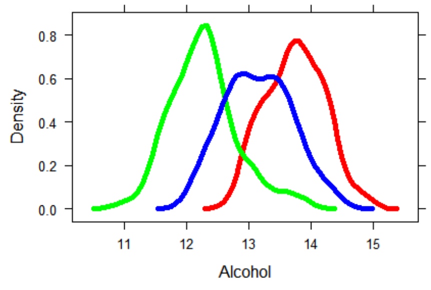
| Item | Range | Distribution |
|---|---|---|
| Cultivar (Response) | 1 to 3 (factor) | 1: 59; 2: 71; 3: 48 |
| Alcohol | 11.03 to 14.83 |  |
| MalicAcid (Malic Acid) | 0.74 to 5.800 |  |
| Ash | 1.36 to 3.230 |  |
| AlcalinityAsh (Alcalinity of Ash) | 10.6 to 30.00 |  |
| Magnesium | 70 to 162 (integer) |  |
| TotalPhenols (Total Phenols) | 0.98 to 3.880 |  |
| Flavanoids | 0.34 to 5.080 |  |

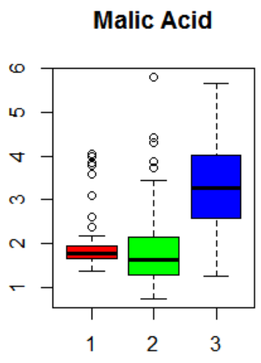
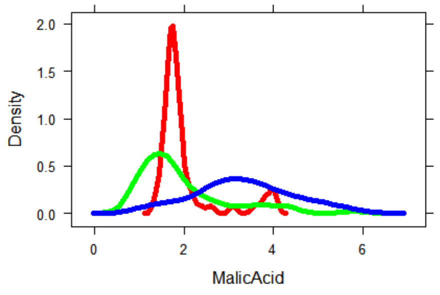
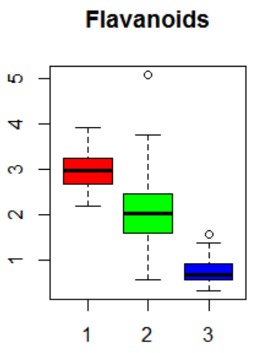
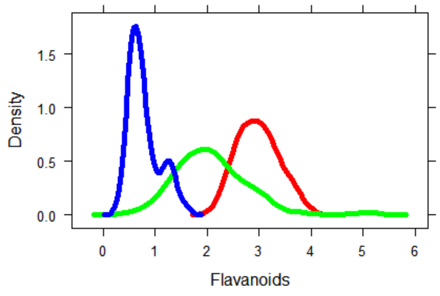
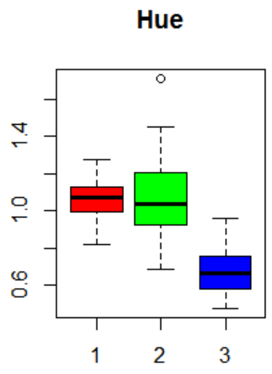
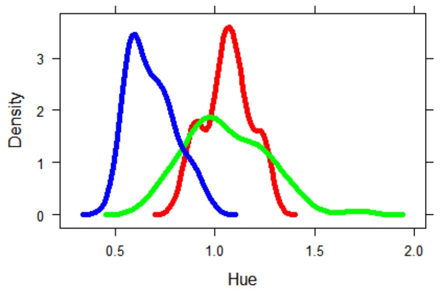
| | | |
|---|---|---|
| NonflavanoidPhenols (Non-flavanoid Phenols) | 0.13 to 0.6600 |  |
| Proanthocyanins | 0.41 to 3.580 |  |
| ColorIntensity (Color Intensity) | 1.28 to 13.000 |  |
| Hue | 0.48 to 1.7100 |  |
| OD280 (OD280/OD315 of diluted wines) | 1.27 to 4.000 |  |
| Proline | 278 to 1680 (integer) |  |

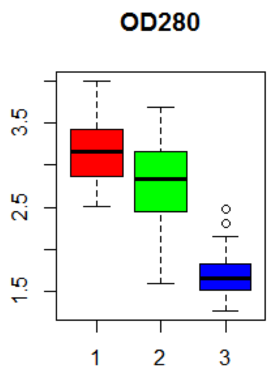
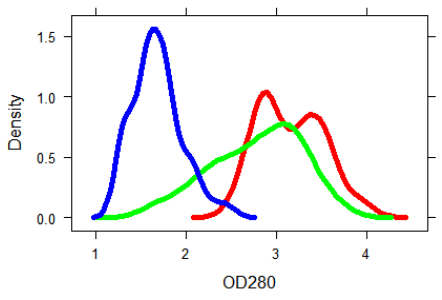
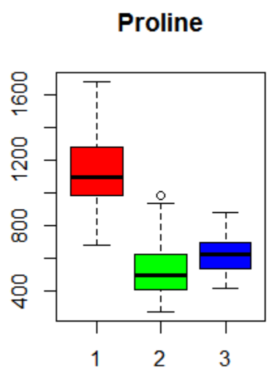
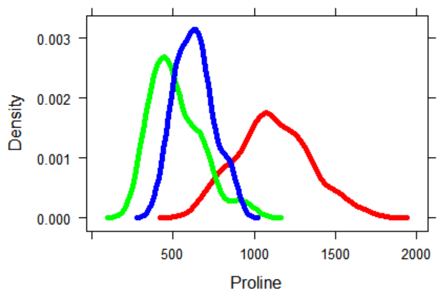
The class distributions are balanced. The ranges of the data are fairly narrow, and most of the features have less than a single order of magnitude in range. Some of the distributions have somewhat longer tails, but outliers are not likely to be an issue with these non-parametric model types. Some of the features appear to be bimodal (e.g., Alcohol, TotalPhenols, Flavanoids); this is somewhat expected with multiple classes because different classes are likely to have different distributions.

Exploratory Data Analysis

Because the project goal is a multi-class classifier model, the features are analyzed to examine what differences occur across different responses. Below are the conditional boxplots and density plots for the most important variables. The complete set may be found in Appendix A.

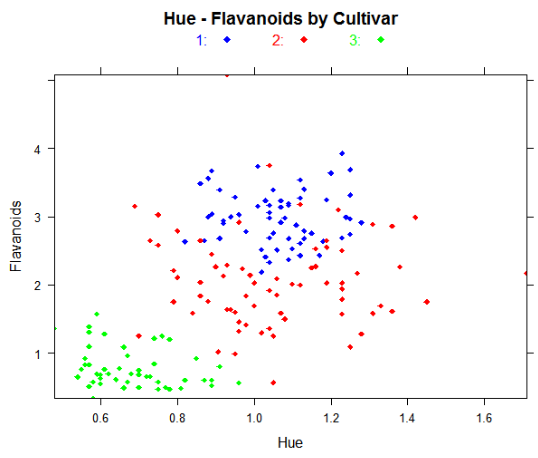
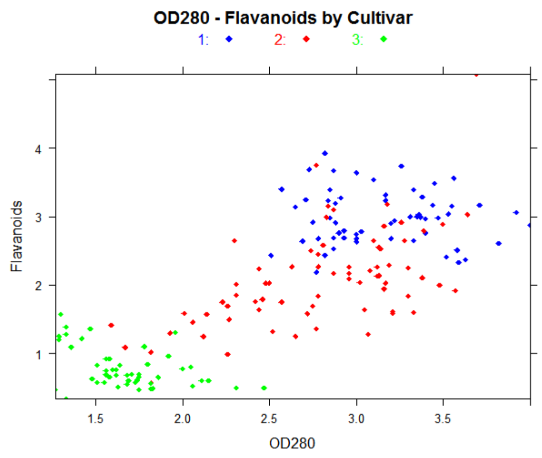
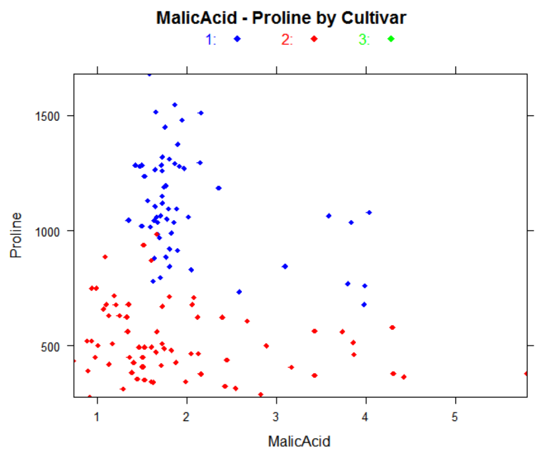
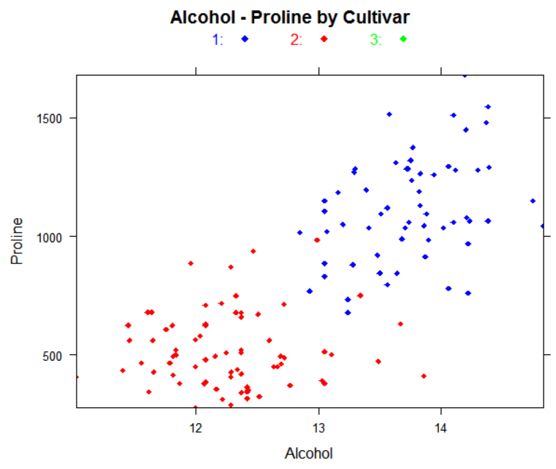
| **Boxplot** | **Density Plot** | **Notes** |
|---|---|---|
|  |  | The variability for alcohol across classes is no surprise as that chemical is determined as much by the wine making process as it is by the grapes. That said, there is not a lot of overlap between classes 1 and 2, and only a couple of outliers on class 2. |

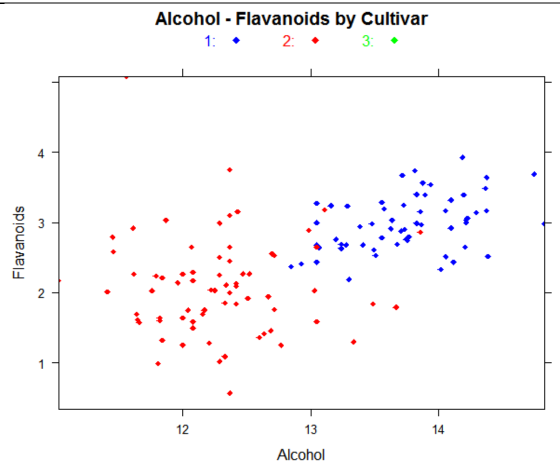
|  |  |  |
|---|---|---|
|  |  | There are outliers in classes one and two which are quite evident in the density plots. Due to the wide distribution of class 3 and the outliers, this feature would probably not be a great discriminator. |
|  |  | This is a fabulous discriminator for classes 1 and 3, and class 2 peaks where the others end. This should be a strong feature in the models. |
|  |  | There is only one outlier, but other than some possible value as a discriminator for class 3, especially vis-à-vis class 1, the variable has a lot of overlap across classes. |

| | | |
|---|---|---|
|  |  | Though this feature has some overlap between classes and 1 and 2, it is probably a good discriminator for class 3 (blue) for the same reasons the following item is a good discriminator for class 1. |
|  |  | This may be a useful discriminator for class 1 because ¾ of its values are beyond the values of the other classes, noting the one outlier. But, because there are other features that separate class 1 from 3, this feature's most significant value be class 1's difference from class 2. |

In examining these conditional plots, outliers do not appear to pose a major problem as the individual distributions look relatively well composed.  Even for distributions like Malic Acid and Flavanoids which have values beyond the 1.5 inner quartile range, there is little reason to believe that the values are erroneous.

To further explore the variables and examine the general ability to easily separate the classes using the data, some two-variable plots colored by the response variable (Cultivar) show that there is some fairly good class separation with only a few features:

| | |
|---|---|
|  With a single straight line at about 45 degrees, one could separate the entire third cultivar (green) with only one erroneous data item within the group. It is clear that the third cultivar will be fairly easy to classify. As noted earlier conidiitional density plots, the third cultivar can be separated from the first cultivar with just Flavanoids. |  If Hue-Flavanoids was not adequate, then OD280-Flavanoids is nearly as good for identifying the third cultivar. |
|  This plot removes the third (green) cultivar to focus on the separation between the first two cultivars. With a simple line, one can separate the first (blue) cultivar leaving only a few second cultivar data items in the classification. |  For Alcohol-Proline, a relatively simple curve would almost separate almost all of the first and second classes. |

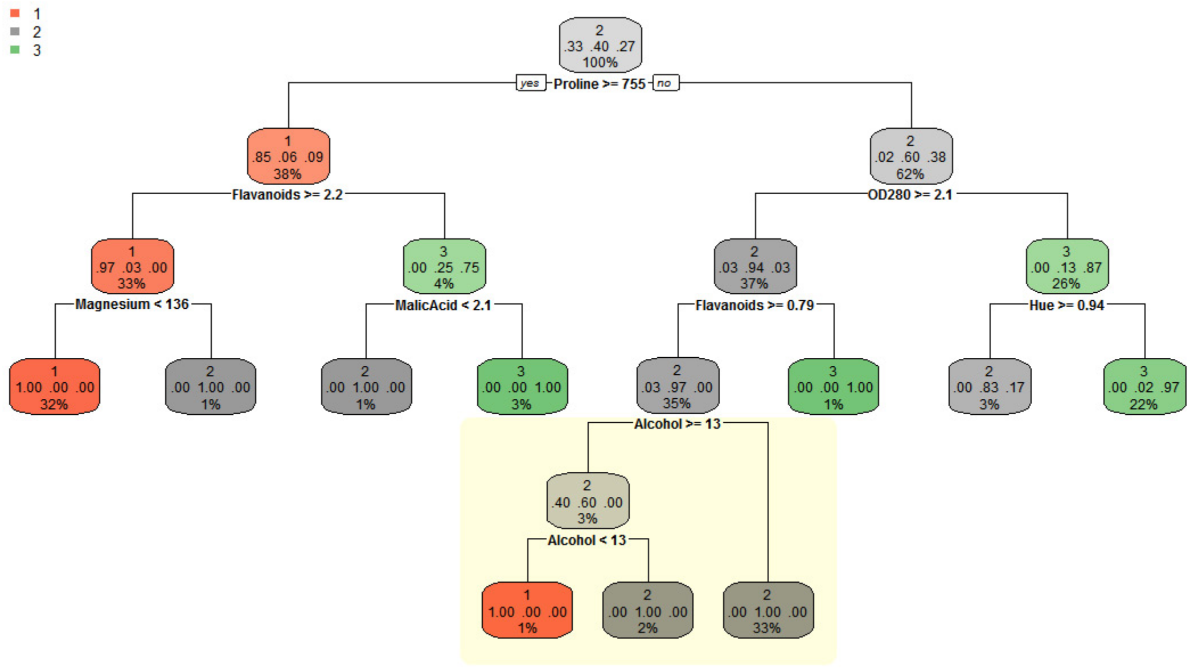


Finally the Flavanoids-Alcohol pairing are similar to the Proline-Alcohol pairing, with four red items mixed into the blue cultivar.

Intentionally
Left
Blank

<u>Exploration with Decision Tree</u>

Finally, the data set was explored by running several decision trees. The figure shown below has only two misclassifications:



The yellow highlighted portion was an expansion of the previous, simpler tree to resolve one additional case. The tree confirms the variable pairs previously identified in the joint plots immediately preceding this section. With only six variables and seven branches is able to classify correctly all but three of the data points, and with the extra Alcohol variable, only two data points are misclassified. This strong performance shows that simpler models can sometimes yield great performance.

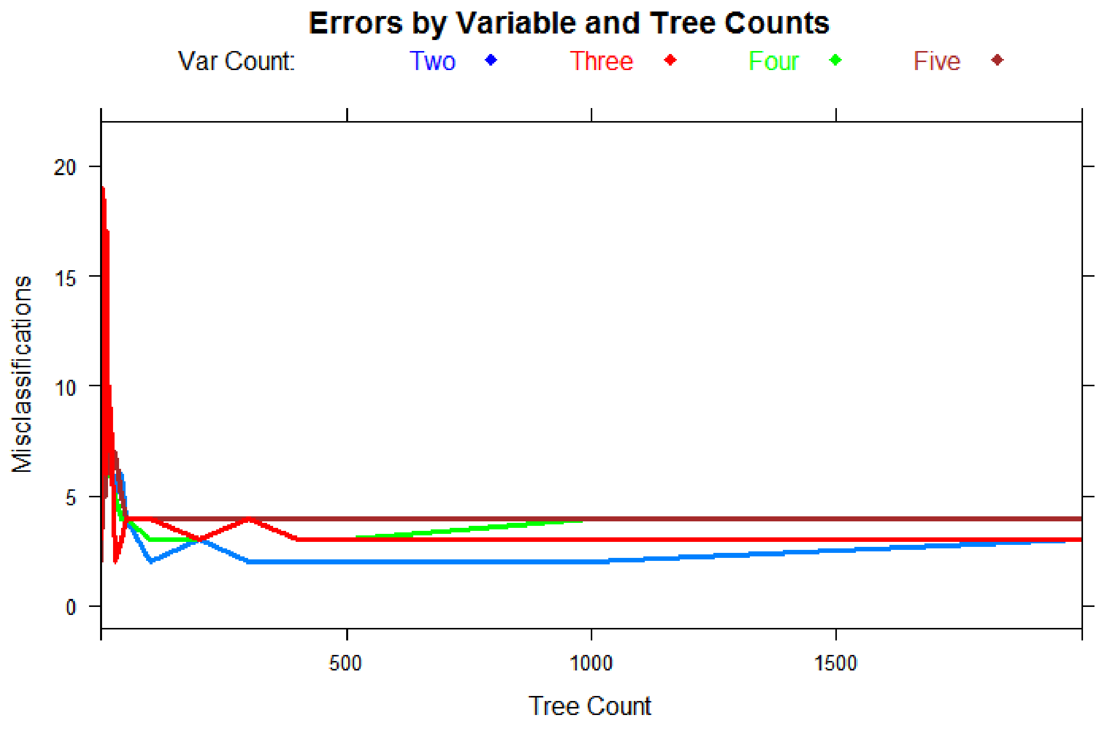
**Models**

Three modeling methods were selected for examination: random forest, neural nets, and support vector machine. Each is discussed in detail.

**Random Forest Model**

The random forest model builds upon the tree methodology by decorrelating the variables with multiple trees of random subsets of variables. This modeling method has quite a few tunable parameters including the depth of the trees to be used, the voting cutoff values, parameters for stratified samples, as well as other parameters; however, for this analysis, only the number of variables used for each tree and the number of trees used will be tuned.

The general rule of thumb recommends the square root of the number of predictors and at least a few hundred trees. Instead of using rules of thumb, the misclassification error rate generated by the method during training was plotted for both variable and tree size:



The plot shows that misclassification error collected from the out of bag statistics in the random forest method fell dramatically with just a few trees. In this case, the error rate using two variables was the smallest over many runs of these computations, and so it may be that using two variables has some beneficial properties. Note: Interestingly, univariate random forests (i.e., single variable forests) with at least 500 trees performed as well as the two-variable random forests.). The tree count selected is 1,000 trees. (It should be remembered that there are only $2^{13}$ or 8192 two-variable combinations and tree duplications will occur as the tree count rises.) The graph suggests that above 1,000 trees, some overfitting may occur for the two-variable combination, but that may also be just chance.

Using the two-variable, one thousand tree random forest model, the variable importance list sorted by decreasing Gini value is as follows:

| Feature | Mean Decrease Accuracy | Mean Decrease Gini |
|---|---|---|
| **Proline** | 0.10 | 16.8 |
| **ColorIntensity** | 0.10 | 16.7 |
| **Flavanoids** | 0.11 | 16.4 |
| **Alcohol** | 0.07 | 13.6 |
| **OD280** | 0.08 | 11.8 |
| **Hue** | 0.06 | 10.0 |
| **TotalPhenols** | 0.04 | 7.5 |
| **MalicAcid** | 0.02 | 5.6 |
| **AlcalinityAsh** | 0.02 | 4.9 |
| **Magnesium** | 0.02 | 4.6 |
| **Proanthocyanins** | 0.01 | 4.1 |
| **Ash** | 0.01 | 2.4 |
| **NonflavanoidPhenols** | 0.01 | 2.1 |

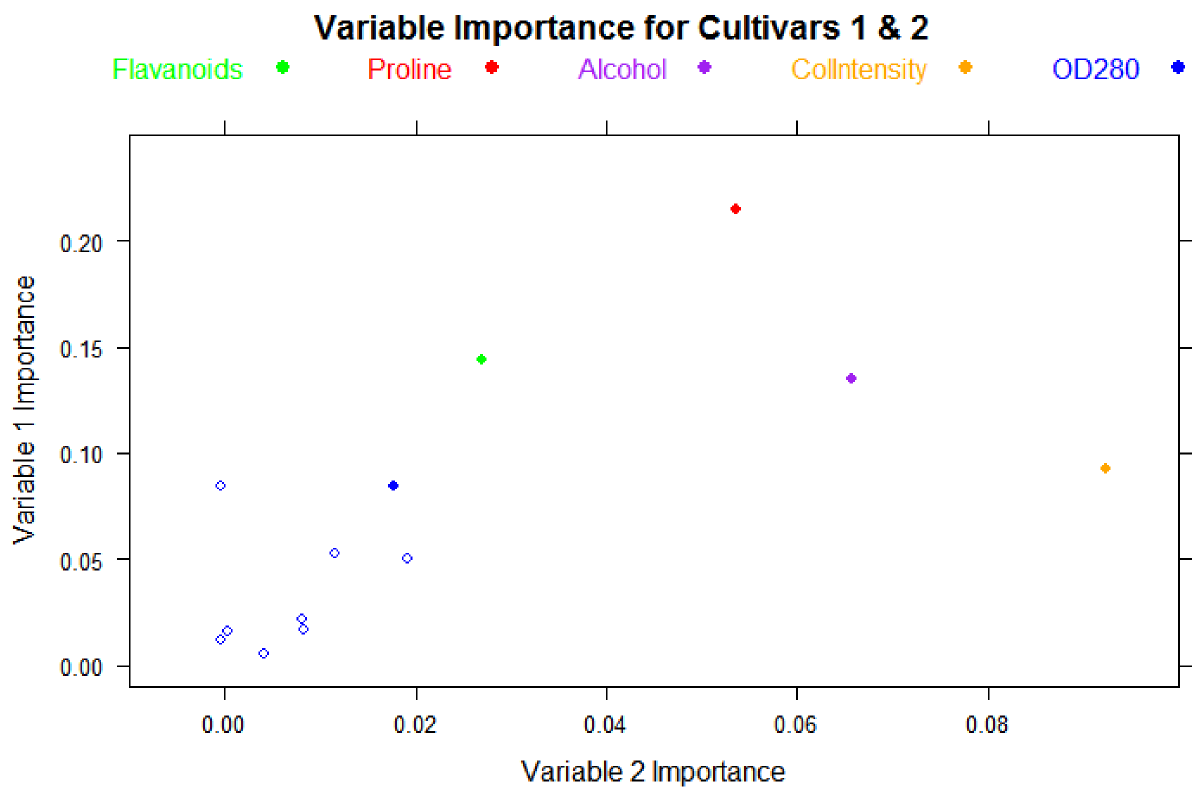
For this dataset, the gini metric and decrease in accuracy are, in large part, identical order. The two surprises in this data were the importance of Color Intensity (which did not show up in the exploratory tree model) and the unimportance of Malic Acid (which appeared to be important in both the plots as well as the exploratory tree model).

To assess the features to the individual classifications, the most important variables for Cultivar 3 are examined in isolation as was done earlier in exploratory data analysis:

| Feature | C3 |
|---|---|
| **Flavanoids** | 0.21 |
| **OD280** | 0.15 |
| **Hue** | 0.12 |
| **ColorIntensity** | 0.12 |
| **TotalPhenols** | 0.05 |
| **Proline** | 0.03 |
| **AlcalinityAsh** | 0.03 |

The table confirms the previous exploratory analysis: Flavanoids, OD280, and Hue largely identify Cultivar 3.  If one includes Color Intensity, the remaining variables are essentially irrelevant for this class.

For Cultivars 1 and 2, the variables will be examined jointly as was done with the exploratory analysis.  This may be visually presented by plotting the importance values as follows:



| Feature | C1 | C2 |
|---|---|---|
| **Proline** | 0.22 | 0.05 |
| **Flavanoids** | 0.14 | 0.03 |
| **Alcohol** | 0.14 | 0.07 |
| **ColorIntensity** | 0.09 | 0.09 |
| **TotalPhenols** | 0.08 | 0.00 |
| **OD280** | 0.08 | 0.02 |
| **Magnesium** | 0.05 | 0.01 |
| **Hue** | 0.05 | 0.02 |
| **AlcalinityAsh** | 0.02 | 0.01 |
| **MalicAcid** | 0.02 | 0.01 |
| **Proanthocyanins** | 0.02 | 0.00 |
| **NonflavanoidPhenols** | 0.01 | 0.00 |
| **Ash** | 0.01 | 0.00 |

(Table sorted by decreasing Cultivar 1 importance.)

As the scatter plot and values table show, Proline, Flavanoids, and Alcohol featured most prominently. Color Intensity was most important to Variable 2. The confusion matrix generated by the randomForest method from the OOB data reveals that the prediction errors occur from mistaking a few items from Cultivar 1 & 3 as Cultivar 2 data..

| 2 variables 1000 trees | Random Forest | | | |
|---|---|---|---|---|
| | Truth | | | |
| RF_Prediction | 1 | 2 | 3 | Classification error |
| 1 | 59 | 0 | 0 | 0 |
| 2 | 1 | 68 | 2 | .042 |
| 3 | 0 | 0 | 48 | 0 |

The random forest model was very efficient at separating the classes.

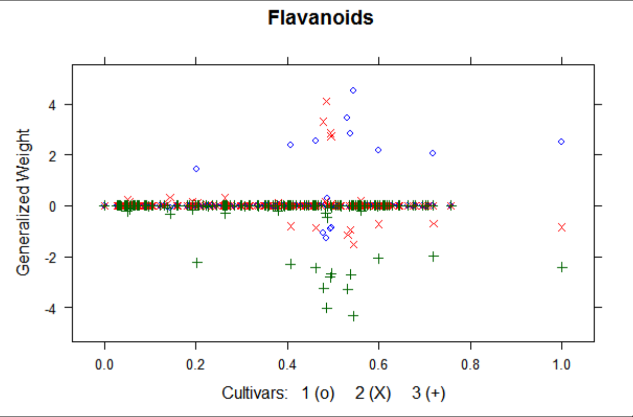
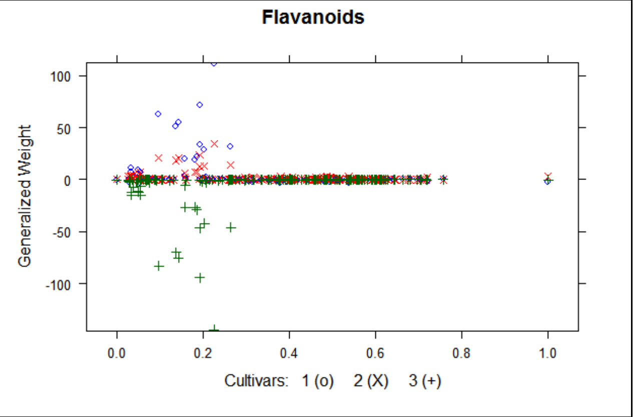
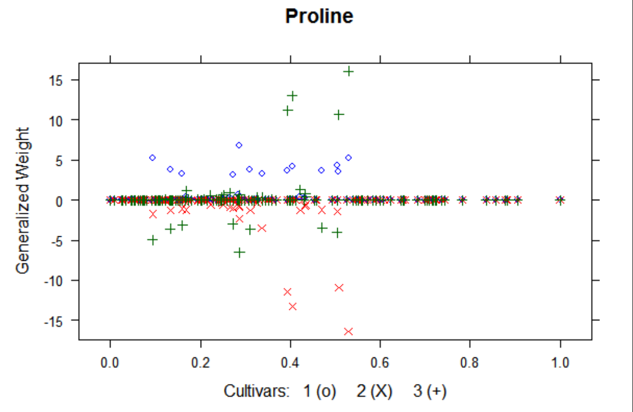
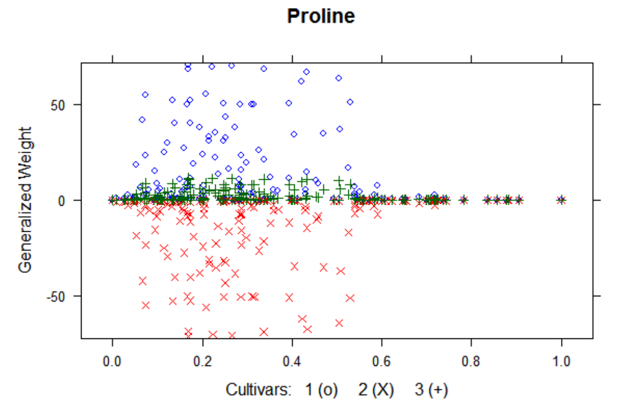
## Neural Network Model

The neural network model ("NN") has strong non-linear modeling capabilities. Given that the exploratory data analysis suggests that separation of these classes is relatively straightforward, the neural network model should have little difficulty in separating the classes.

Because of the use of gradient descent, neural networks typically perform best when the data is standardized so that the gradient descent can proceed efficiently. If one has data on markedly different scales, the data can form topological surfaces that have long narrow valleys, and it makes locating the global minima more difficult. Nevertheless, because the scales on the variables were not multiple orders of magnitude different, an NN with two hidden neurons using the raw data was modeled. The confusion matrix for this initial model reveals some problems:

| Hidden: 2 | Neural Net- unscaled data | | Error: 0 |
|---|---|---|---|
| | Truth | | |
| NN_Prediction | 1 | 2 | 3 |
| 1 | 59 | 11 | 5 |
| 2 | 0 | 60 | 0 |
| 3 | 0 | 0 | 43 |

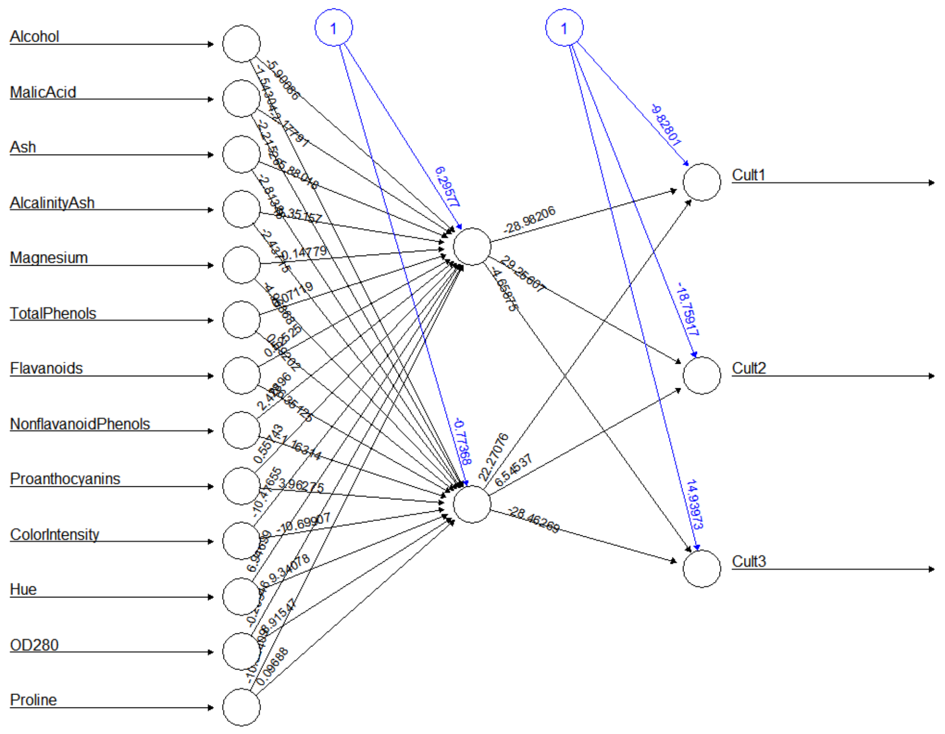
With sixteen prediction misclassifications on the very data on which it was trained, the conclusion was made that standardization was indeed required. Though that sounds simple enough, the kind of transformation that ought to be performed appeared to be an open question. For example, with z-score normalization, outliers can still cause problems. To make a decision based on some sort of evidence, the author created models using both z-score normalization as well as 0-1 normalization, and then examined the scatter plots of the generalized weights:

| Z-Score Standardization | 0-1 Standardization |
|---|---|
|  |  |
|  |  |

The generalized weights plots help provide insight into how a predictor is being used by the model. The two pairs of plots show two of the most important variables with the idea that one would expect the models to leverage them for all their predictive power. In looking at the plots of the generalized weights, the top pair of plots (Flavanoids) do not show much difference. However, for the lower pair (Proline), the 0-1 standardization generated a much wider range of weight values instead of having a considerable number of zero values. In other words, the neural net model leveraged the Proline variable to a much greater amount when it was standardized with 0-1 scaling. Though not shown, the pairs of plots of the other variables looked more like the lower pair than the upper pair.

It is not clear to me why the neural net would utilize the Proline variable less with the other scaling; it may be that the nodes were saturating and thus generating zero weights for some variables. Without further examination, it would be difficult to determine exactly what is happening, but the author felt that it was prudent to use 0-1 standardization method.

Using the 0-1 standardization, another two-neuron neural network model was generated and its network plot is as follows:



As the plot shows, the network has thirteen input nodes for the predictors, two activation nodes, three output nodes, and two bias nodes (blue), one for the activation nodes and the output nodes. The weights for the nodes are noted on the lines. The interactions between the two activation nodes create a great deal of complexity and obscure the relationships between the inputs and the outputs. For example, we know from previous analysis that Color Intensity is very important for Cultivar 2, but the weights for the two activation nodes are -10.5 and -10.7, and its importance is only possible when examined in combination with all of the other variables at the same time, and this is not easily comprehended. For this neural network, the sigmoid activation function was selected. The confusion matrix of predictions was then calculated for the data used to train the network model and, unsurprisingly, it showed no errors:

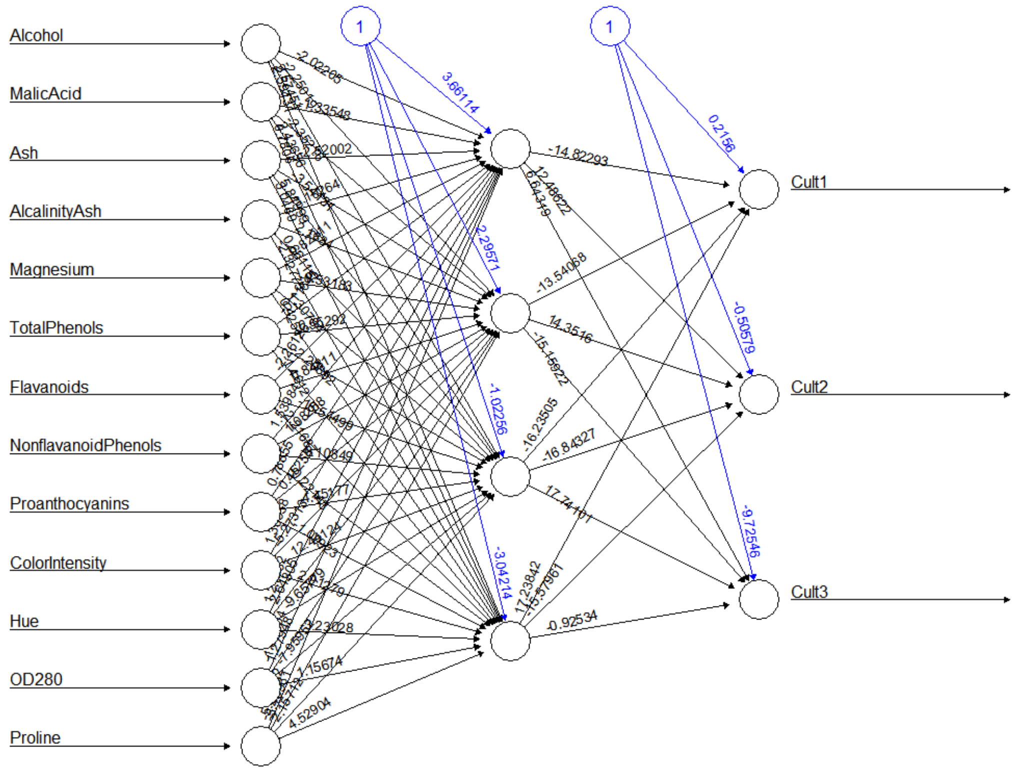
**Confusion Matrix – Training Data (2 Neurons)**

| Hidden: 2 | Neural Net | | Error: 0 |
|---|---|---|---|
| | Truth | | |
| NN_Prediction | 1 | 2 | 3 |
| 1 | 59 | 0 | 0 |
| 2 | 0 | 71 | 0 |
| 3 | 0 | 0 | 48 |

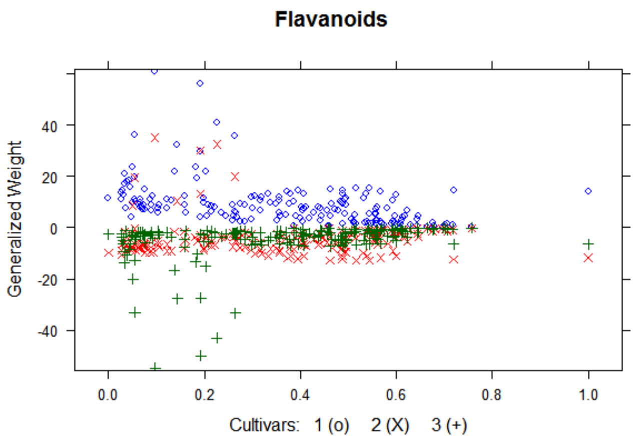
Because it is not making mistakes on its own training data (as did the unscaled data model), the author concluded that the neural network was working correctly.

<u>Increasing the Number of Neurons</u>

The number of neurons determines the amount of nonlinearity that the neural network can model. It was unknown how much nonlinearity in the model was required to effectively model the dataset, and, therefore, how many hidden neurons should be used. A plot of a four-neuron model reveals how these models become very complex very fast:

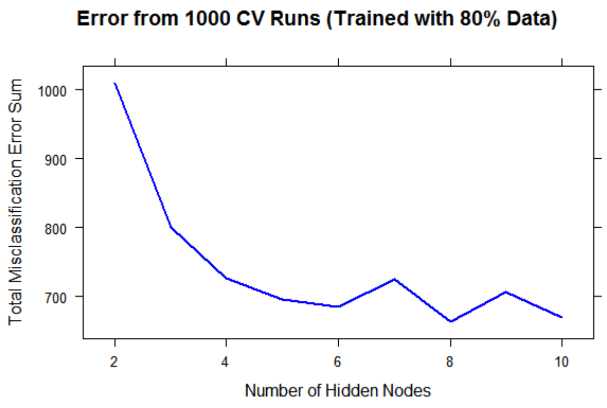


As one can see, the number of combinations of the input variables has increased dramatically and thus allows much greater fitting of the model to the data. By again looking at the general weights, one can see the effect of adding these two extra neurons. Previously the general weights for Flavanoid were nearly all zeroes in the two-neuron model. With the four-neuron model, the general weights for the Flavanoids variable are now as follows:

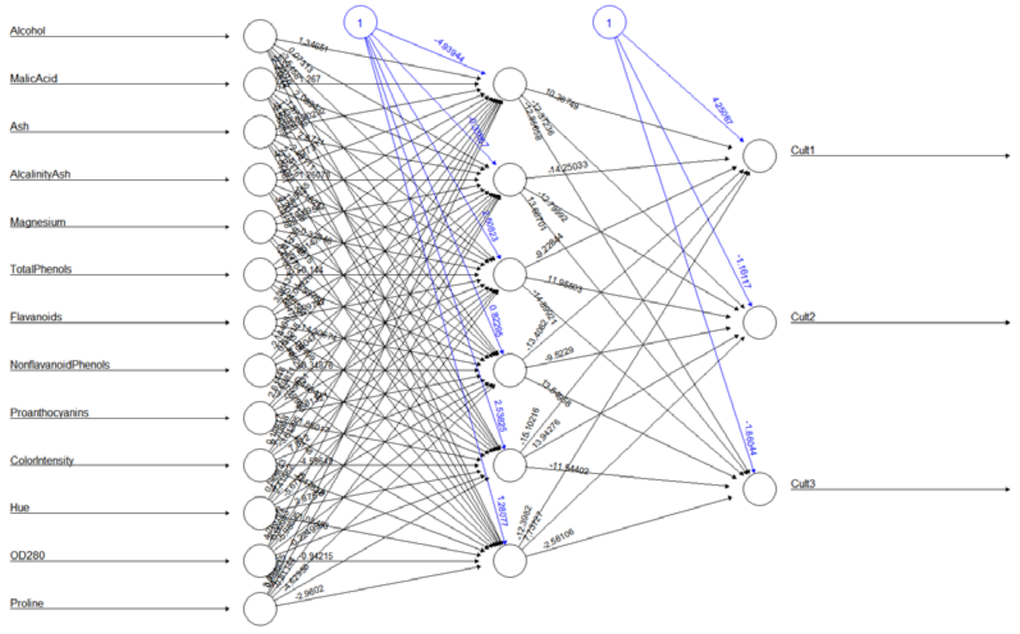


As the plot shows, the weights are almost entirely non-zero, meaning that this predictor is now affecting the outputs in many more of the cases. Since we already know that this is an important variable, this result would seem desirable (assuming that one does not get overfitting).

In terms of determining some general limits on the number of hidden neurons to use, the general rule of thumb is not to exceed the number of inputs minus the number of outputs, which in this case would be a limit of ten. Consequently, to tune the number of neurons, training using 1000 samples of 80% of the data was conducted with the error results collected for 20% unused data. The error data for the various hidden node counts is plotted below.



Please note that the neural nets with two hidden nodes would fault at times, and so the reported result for two nodes is merely an estimate based on the runs that completed successfully. The plot shows that the neural nets with two or three hidden nodes are underfitting the data. It is unclear whether the neural nets for hidden nodes greater than six are overfitting the data or whether the results were due to chance. The fact that a six node network appears to have most of the error reduction and less risk of overfitting, the six-neuron model was selected. The network model for the six hidden node layer is as follows:



It is interesting to note that in this model, two out of every three output weights from the hidden nodes to the output nodes are negative; this makes sense given that negative values for the sigmoid function yield values towards zero and two of the three output values should be near zero for proper classification. Because the methodology above calculated the out-of-sample errors, these errors where accumulated for 100 samples to generate a representative confusion matrix for the out-of-sample data:

**CV Error on 3600 Data Points**
**Neural Net with 6 Hidden Nodes**

| | Truth | | |
|---|---|---|---|
| NN_Pred | 1 | 2 | 3 |
| 1 | 1195 | 24 | 0 |
| 2 | 10 | 1411 | 10 |
| 3 | 0 | 19 | 931 |

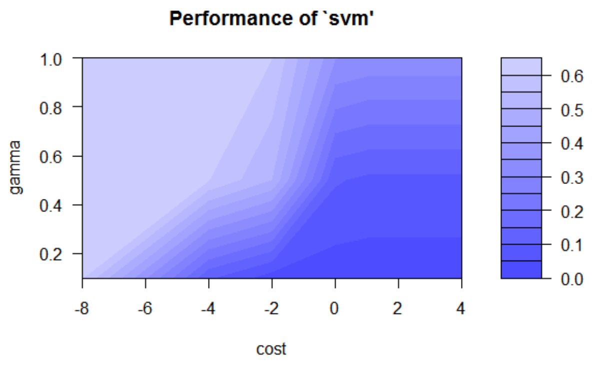
With little surprise, the model never made an error confusing Cultivars 1 & 3. The only confusion arose with respect to Cultivar 2. The total misclassification rate was 63/3600 or 1.75%. It should be remembered that the test error rate may be less than this because this is a small data set and important data may be randomly selected for the out-of-bag sample.

Support Vector Machine Model

The support vector machine is a robust classifier model which has both advantages and limitations.  Through its use of support vectors to define boundaries, outliers tend to be unimportant.  Though it may be a bit slow to train as it identifies the support vectors, it is fast to execute because the number of support vectors is usually a tiny fraction of the total number of data items.  In comparison, a KNN classifier may have no training time, but can be extremely slow in production as each novel prediction requires the identification of the nearby neighbors in an expanding data set.

The support vector machine model is a bit of a black box model, and for this model, the only tuning parameters considered were the cost and gamma values as these are extremely important to obtain good performance.  The cost value represents the cost associated with a support vector, and thus, if the cost is low, there will be more of them which means that individual support vector matter less and thus low cost values have higher bias.  A higher cost function means fewer support vectors and thus individual data points matter more and thus increase variance.

A low cost function with higher bias may underfit the data whereas a high cost function creates more variance and risks overfitting.  Thus, one can imagine that one wants a balance which means to reduce the classification error while minimizing the cost value.  The gamma value represents the overall impact of any particular support vector upon the other data elements, and may be thought of how tall and pointed or flat and broad of the distribution around each support vector.   Thus, one can see how these variables are interrelated, and their tuning needs to be considered simultaneously.  Using the tune function with cross validation, the support vector machine model with the radial kernel was tuned.  The author understands that the linear kernel is a subset of the radial kernel, and that performance should not be worse, and testing seemed to confirm it.  Using various values for both the cost and gamma values generating the tune function with the radial kernel generated the following plot:



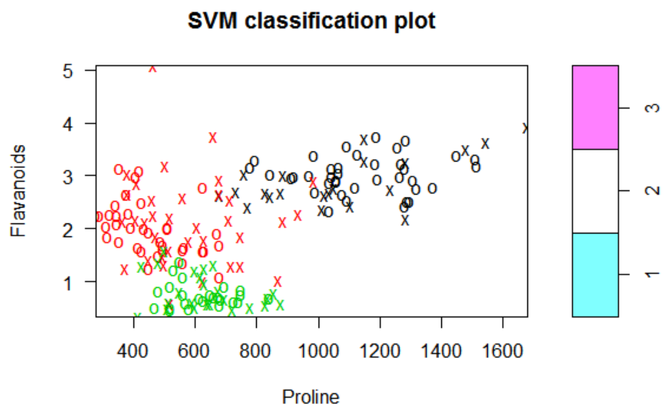
The function recommended optimal parameter settings of a cost of .25 with a gamma of .1. In the plot above, which was done with a log2 transform, this corresponds to {-2, .1}. It is important to note that in running the tuning multiple times, the random out-of-bag samples from this small dataset caused the tuning function to recommend cost values as high as 1 (less bias, more variance). Using these .25/.1 settings with higher bias to avoid overfitting, the support vector machine model was then used to predict the data set to obtain an idea of how well it might do in practice:

| Cost .25 | Gamma .1 | | Error: 2 |
|---|---|---|---|
| 104 SV | Truth | | |
| Prediction | 1 | 2 | 3 |
| 1 | 58 | 0 | 0 |
| 2 | 1 | 70 | 0 |
| 3 | 0 | 1 | 48 |

The table shows two prediction errors or approximately a 1.1% error rate on the training set. Setting the cost value of 1 decreases the support vector count to 80 and generates zero misclassifications when predicting the dataset, but such a setting might yield poor results on unseen test data. It should be noted that a cost setting of 50 only decreases the support vector count to 74, and most of the dropped items are within Cultivar 3.

Originally, the author intended to compute a confusion matrix using repeated training/testing samples as was done with the neural net model. This yielded extremely high error rates, with the author theorizing that removal of any of the data from the small dataset made it difficult for the method to determine the most appropriate data points. This would help explain the widely varying cost values returned by the tuning function.

The SVM model is somewhat difficult to visualize as it leverages a high dimensional space that is difficult to visualize. One method is to create the two-dimensional support vector plots:



The Xes designate support vector points, and the plots look good as the Xes appear on the periphery of the cluster of points colored by cultivar. (Note: The plotting function miscolored the bar to the right, with the green points being Cultivar 3 and the red/black ones being Cultivars 1 and 2.) Comparing the same plot for different cost values revealed that higher cost values mainly affected the Cultivar 3 support vectors. In isolation these plots do not provide much useful information. For example, the fact that support vectors for Cultivar 3 are at the bottom of the graph suggest that the two dimensional representation is woefully inadequate for showing the real relationships that are identified in the multi-dimensional context.

If the cost value is increased (as shown below in the following two confusion matrices), the number of support vectors is reduced, but error on the training set goes to zero with a cost value of 1.

| Cost .5 | Gamma .1 | | Error: 1 |
|---|---|---|---|
| 87 SV | Truth | | |
| Prediction | 1 | 2 | 3 |
| 1 | 59 | 0 | 0 |
| 2 | 0 | 70 | 0 |
| 3 | 0 | 1 | 48 |

| Cost: 1 | Gamma .1 | | Error: 0 |
|---|---|---|---|
| 80 SV | Truth | | |
| Prediction | 1 | 2 | 3 |
| 1 | 59 | 0 | 0 |
| 2 | 0 | 71 | 0 |
| 3 | 0 | 0 | 48 |

The varying cross validation results on the dataset bothered me for days, and finally, it occurred to me that maybe it was unwise to include all of the variables without any reflection as to their importance. Though the random forest model is designed to handle irrelevant features, the introduction of all the features into the support vector model might introduce more error than signal. Consequently, the insights identified from all the EDA and previous models were then applied to an attempt to create a superior support vector machine model that benefitted from a reduced set of variables.

To create this model, I selected the top features for discriminating between Cultivars 1 & 2 and the top features for identifying Cultivar 3. After numerous runs of the tuning function to identify the lowest possible cross validation error, the following variables were selected:

**Proline, Flavanoids, Alcohol, ColorIntensity, OD280, and Hue**

Unlike the model with all of the variables, the tuning function repeatedly returned a single set of values, cost=16, gamma=.1. With these values, a new support vector model was created, and one hundred 80% training and 20% testing samples were created and the errors recorded using the same approach as with the neural network evaluation. The following table shows the results:

|  | Truth |  |  |
|---|---|---|---|
| SVM_Pred | 1 | 2 | 3 |
| 1 | 1215 | 18 | 0 |
| 2 | 3 | 1329 | 1 |
| 3 | 0 | 23 | 1011 |

This table is a fairly dramatic improvement over the neural net tested in the same manner. The Cultivar 1 and 3 misclassifications of 3 and 1 were 10 and 10, respectively, for the neural net. The other two misclassifications were approximately the same at 18 and 23 for the SVM and 24 and 19 for the neural net.

By reducing the number of features to six, the support vector count for this revised model went down from 104 in the previous model with all features to thirty (30) support vectors. By removing irrelevant features and noise, the model is able to select a set of features that more pristinely defines the boundaries of these classes, and it results in fewer errors. Consequently, the final SVM model to be compared would be this reduced variable set SVM model.

## Model Comparison and Recommendation

All of the models did very well during their development phase. After much consideration, the author decided that an apples-to-apples approach was needed for comparison of the models because different approaches had been used during the development. To evaluate the models, each model was subjected to a Leave One Out Cross Validation Test. This would generate a confusion matrix of all 178 points and place them in their location in the matrix. Since all models would receive the same test, it was a fair comparison, and the only downside was the few minutes of computation time required to produce it. The results are as follows:

**Random Forest Model**

|  |  | Truth |  |
|---|---|---|---|
| **RF** | 1 | 2 | 3 |
| 1 | 59 | 1 | 0 |
| 2 | 0 | 69 | 0 |
| 3 | 0 | 1 | 48 |

**Neural Net Model**

|  |  | Truth |  |
|---|---|---|---|
| **NN** | 1 | 2 | 3 |
| 1 | 58 | 1 | 0 |
| 2 | 1 | 69 | 0 |
| 3 | 0 | 0 | 48 |

**SVM – Full Model**

|  |  | Truth |  |
|---|---|---|---|
| **SVM** | 1 | 2 | 3 |
| 1 | 58 | 0 | 0 |
| 2 | 1 | 70 | 1 |
| 3 | 0 | 1 | 47 |

**SVM – Reduced Feature Model**

|  |  | Truth |  |
|---|---|---|---|
| **SVM** | 1 | 2 | 3 |
| 1 | 59 | 1 | 0 |
| 2 | 0 | 70 | 0 |
| 3 | 0 | 0 | 48 |

None of the models had any problems differentiating the first and third cultivars; it was the second cultivar that caused difficulties. Assuming new data is comparable to the existing dataset, this author would probably exclude the neural network model from consideration. The model seems highly sensitive to data preparation and inclined to overfit. Its confusion matrix during testing encompassed all four error categories of the second cultivar errors. This suggests that it is either more balanced, which could be good, or very sensitive to slight changes in the data, which is more problematic. The author thinks the second is more likely than the first and on that basis would exclude this model from consideration.

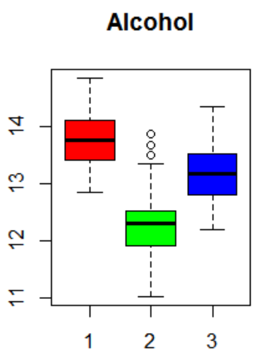
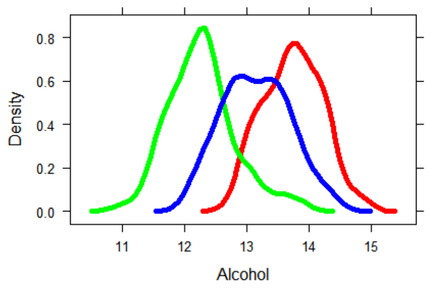
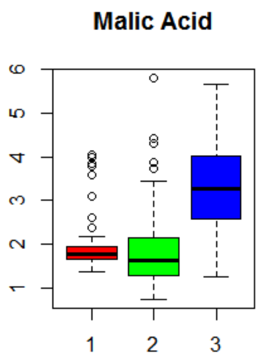
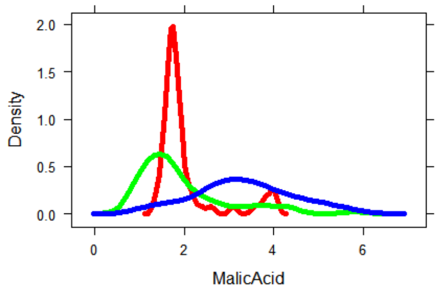
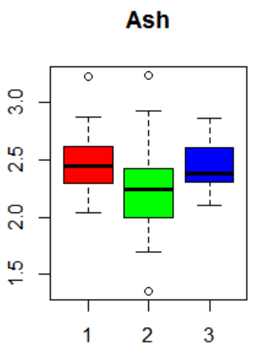
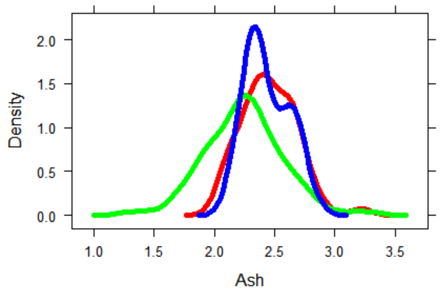
As between the random forest model and the SVM, this choice is somewhat of a toss-up. The random forest clearly has some advantages with its automated partitioning and decorrelation, and it performed better than the full variable set SVM on the leave one out cross validation test. Nevertheless, my intuition is that the tuned support vector machine model would likely beat the random forest model with an unseen test set, and this is suggested by the leave one out results in which the tuned SVM had the lowest error score. This conclusion is bolstered by the fact that the support vector model has now been tuned to achieve very high performance from a minimal (30) number of data points as support vectors. This small dataset does not play to the random forest's true strength which is to decorrelate large numbers of features over large solution spaces. Without the tuning of the support vector model in terms of feature selection, then the random forest model would be the safer choice.

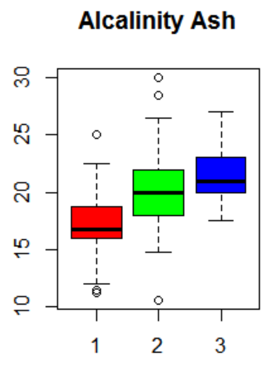
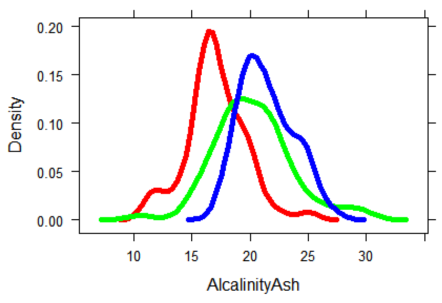
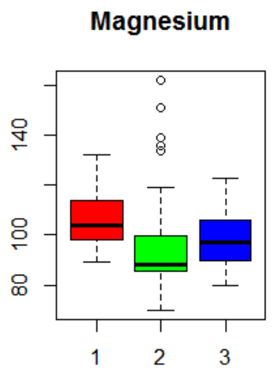
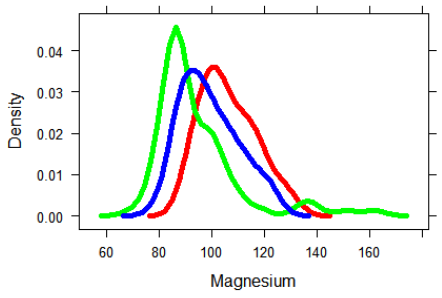
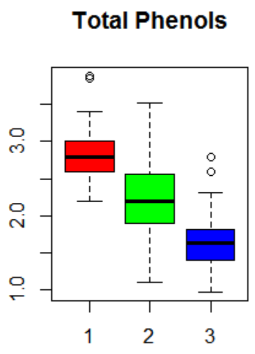
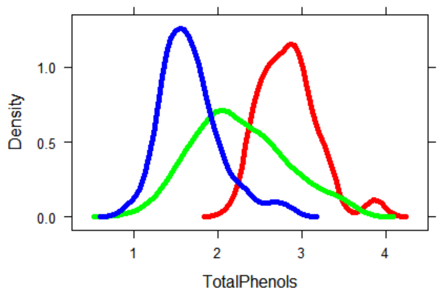
Thus, given the unique characteristics of this problem, the reduced feature support vector machine model would be selected as the champion model with the random forest model as a challenger. If the data set was both broader and deeper, the conclusion might well be the opposite.

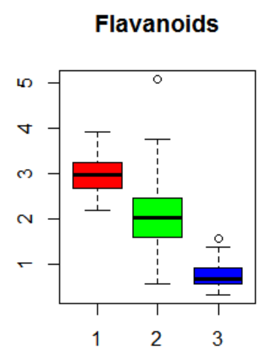
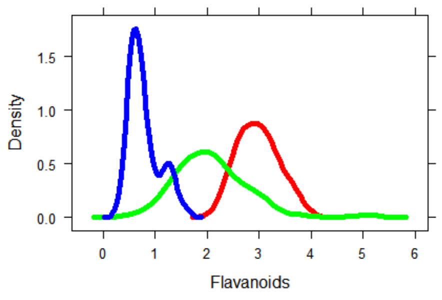
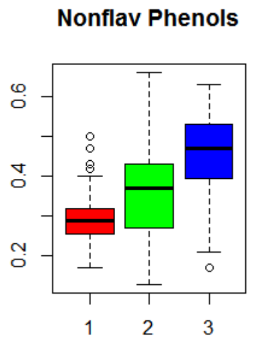
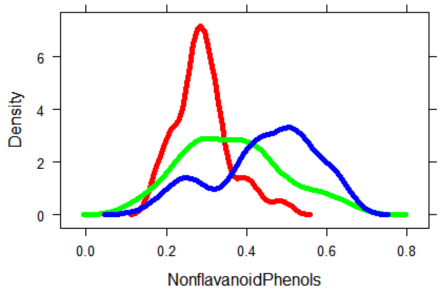
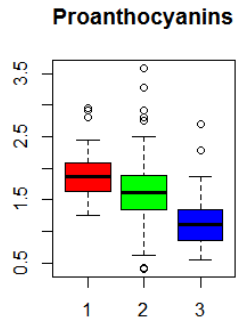
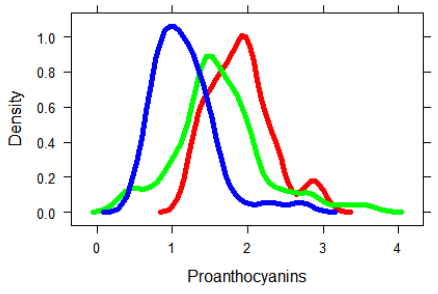
## Final Comments

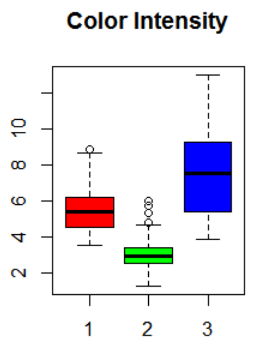
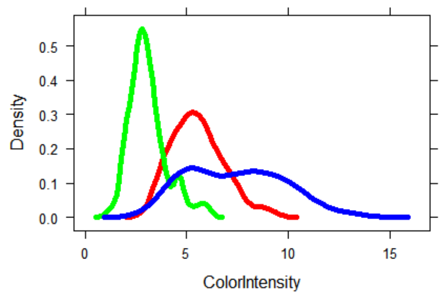
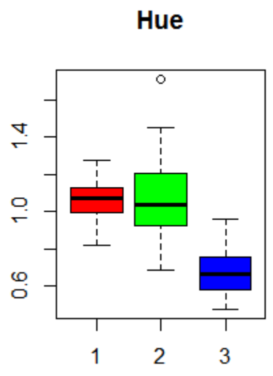
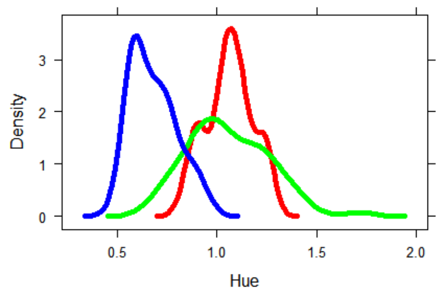
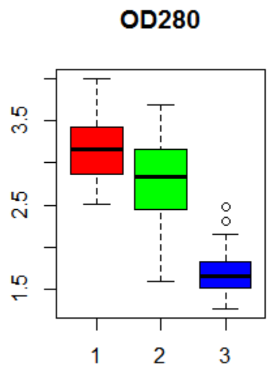
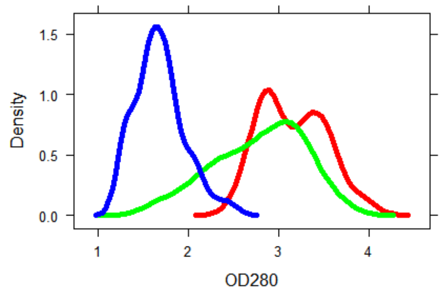
During the course of data exploration, this problem repeatedly looked like a problem that would benefit from creating a two-staged model. Cultivar 3 is nearly separable with a linear plane involving just a few variables. Consequently, one could easily run a high-performing binary logistic regression model to identify all Cultivar 3 data items and remove them. For the remaining data, then one of the other approaches analyzed here could leverage their non-linear abilities and yield even better results by not being burdened by trying to balance the influences of the Cultivar 3 data points.

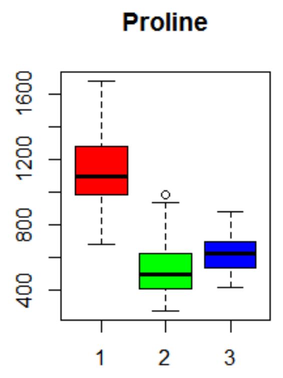
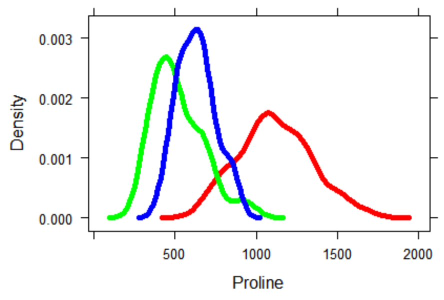
## Appendix A
## Feature Plots

| Boxplot | Density Plot | Notes |
|---|---|---|
|  |  | The variability for alcohol across classes is no surprise as that chemical is determined as much by the wine making process as it is by the grapes. That said, there is not a lot of overlap between classes 1 and 2, and only a couple of outliers on class 2. |
|  |  | There are outliers in classes one and two which are quite evident in the density plots. Due to the wide distribution of class 3 and the outliers, this feature would probably not be a great discriminator. |
|  |  | This feature may be the worst discriminator due to the near-identical distributions for classes 1 & 3 (red and blue) and the substantial overlap with class 2. |

| | | |
|---|---|---|
|  |  | Alcalinity of ash improves upon ash as a class discriminator, but is still not a great one. |
|  |  | Magnesium is probably the second-worst class discriminator due to the overlap of all three distributions. The small benefit of class 2 being shifted to the left is offset by its far right outliers. |
|  |  | Though not quite as good as flavonoids (which follows) in separating classes 1 and 3, it is nevertheless pretty strong in that regard. Not very useful for class 2 as it overlaps both of the other classes. |

| | | |
|---|---|---|
|  |  | This is a fabulous discriminator for classes 1 and 3, and class 2 peaks where the others end. This should be a strong feature in the models. |
|  |  | Class 2 (green) has a lot of overlap, but classes 1 and 3 are largely separated. |
|  |  | This variable has outliers in all three classes. It even has outliers on both sides of the distribution of class 2. This feature is similar to flavonoids, but the latter feature is much better. |

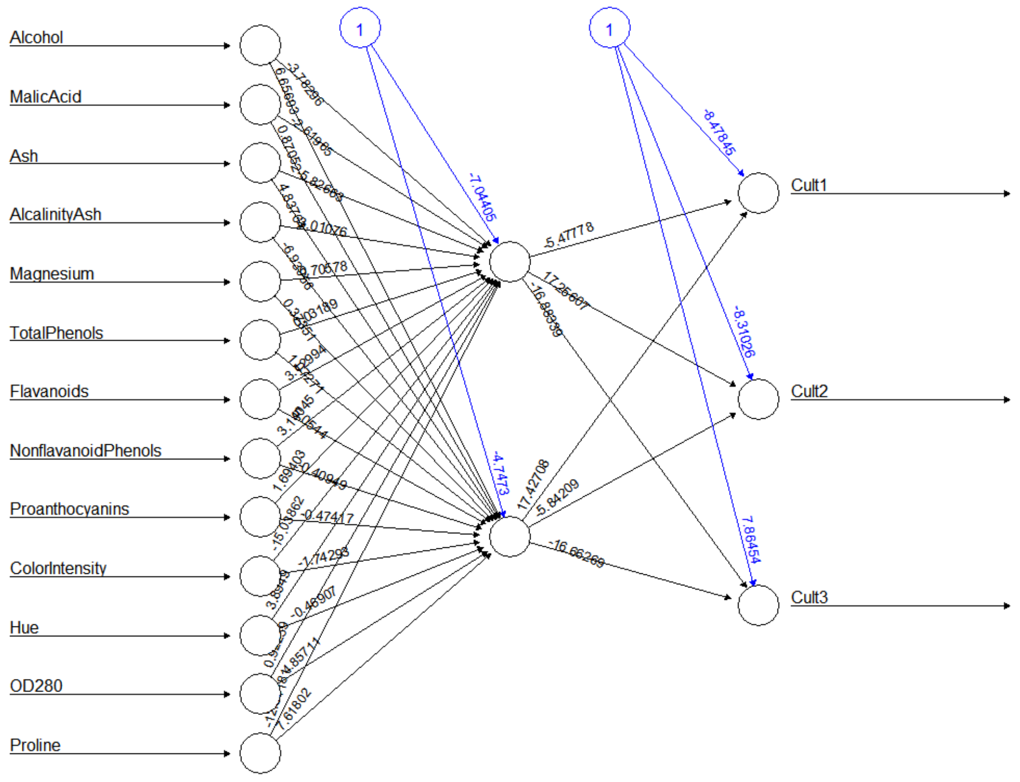
| | | |
|---|---|---|
|  |  | Color intensity might be useful for identifying class 2, though there are a few outliers. |
|  |  | There is only one outlier, but other than some possible value as a discriminator for class 3, especially vis-à-vis class 1, the variable has a lot of overlap across classes. |
|  |  | Though this feature has some overlap between classes and 1 and 2, it is probably a good discriminator for class 3 (blue) for the same reasons the following item is a good discriminator for class 1. |

| | | |
|---|---|---|
|  |  | This may be a useful discriminator for class 1 because ¾ of its values are beyond the values of the other classes, noting the one outlier. But, because there are other features that separate class 1 from 3, this feature's most significant value be class 1's difference from class 2. |

APPENDIX B

Neural Net Plots

NN Model with z-score standardization



Appendix C
R Code

```
# functions
########################
mystandardize = function(dftostandardize,referencedf){   #z-score standardization
  meanvals <- apply(referencedf, 2, mean)
  stddevs <- apply(referencedf, 2, sd)
  t((t(dftostandardize)-meanvals)/stddevs)
}
scalar0to1 = function(myvar){   #calculate a scalar to put a column between 0 and 1
  (myvar - min(myvar))/(max(myvar) - min(myvar))
  }
mystandardize01 = function(dftostandardize){     #0to1 standardization
  data.frame(lapply(dftostandardize,scalar0to1))
}
#_______________________________________________________________
#
#        Exploratory tree
#
#_______________________________________________________________
treedf = df.orig
treedf$Cultivar = as.factor(treedf$Cultivar)
tmodel <- rpart(Cultivar~., data =treedf, control=rpart.control(minsplit=2))
rpart.plot(tmodel,cex=.75,col=c("black"),digits=2)
a = predict(tmodel,newdata=treedf)
a

?rpart

#zero error model
tmodel <- train(Cultivar~., data = treedf, method = "rpart",
                control=rpart.control(minsplit=2),
                trControl = trainControl(method = "cv", number = 10),
                tuneLength=20)
#names(tmodel)
rpart.plot(tmodel$finalModel,digits=2,cex=.75,col=c("black"))
tmodel$results
tmodel$finalModel
tmodel$coefnames
?rpart.plot

Non-linearites
ListOfTreeVars =names(df)[-c(1,14)]
```

```
for (i in ListOfTreeVars){
  myformula = paste("resp_spam","~",i,sep="")

model.tree=rpart(as.formula(myformula),treedf,method="anova",control=rpart.control(minspli
t = 10, cp=.005))
  print(model.tree)
  rpart.plot(model.tree,digits=3,cex=.75)
}

#####################################################################
#_________________________________________________________
#
#         Neural Nets - Original Data
#
#_________________________________________________________
# Neural Net 0: Original variables with 0/1 as responses.
nnmod1 <-
neuralnet(Cult1+Cult2+Cult3~Alcohol+MalicAcid+Ash+AlcalinityAsh+Magnesium+TotalPhenols+
Flavanoids+NonflavanoidPhenols+Proanthocyanins+ColorIntensity+Hue+OD280+Proline,
          data=nn0.non, hidden=4, err.fct="ce",linear.output=FALSE)
?neuralnet
nnmod1    # basic output
names(nnmod1)   # reults options
nnmod1$result.matrix     # result matrix
nnmod1$generalized.weights
df.gwmod1=data.frame(nnmod1$generalized.weights)
names(df.gwmod1) <-
list("Alcohol1","MalicAcid1","Ash1","AlcalinityAsh1","Magnesium1","TotalPhenols1","Flavanoid
s1","NonflavanoidPhenols1","Proanthocyanins1","ColorIntensity1","Hue1","OD2801","Proline1
","Alcohol2","MalicAcid2","Ash2","AlcalinityAsh2","Magnesium2","TotalPhenols2","Flavanoids2
","NonflavanoidPhenols2","Proanthocyanins2","ColorIntensity2","Hue2","OD2802","Proline2","
Alcohol3","MalicAcid3","Ash3","AlcalinityAsh3","Magnesium3","TotalPhenols3","Flavanoids3","
NonflavanoidPhenols3","Proanthocyanins3","ColorIntensity3","Hue3","OD2803","Proline3")
# The given data is saved in nn$covariate and nn$response as well as in nn$data for the whole
data set inclusive
# non-used variables. The output of the # neural network, i.e. the fitted values o(x), is provided
# by nn$net.result:
outmod1 <- cbind(nnmod1$covariate,nnmod1$net.result[[1]])
dimnames(outmod1) <- list(NULL,
c("Alcohol","MalicAcid","Ash","AlcalinityAsh","Magnesium","TotalPhenols","Flavanoids","Nonfl
avanoidPhenols","Proanthocyanins","ColorIntensity","Hue","OD280","Proline","Cult1","Cult2","
Cult3"))

names(nn0.non)
```

```
nnmod1.pred = data.frame(compute(nnmod1,nn0.non[,-c(14:16)], rep=1))
predoutmod1 =
data.frame(Cult1=nnmod1.pred$net.result.1,Cult2=nnmod1.pred$net.result.2,Cult3=nnmod1.p
red$net.result.3)
pred_singlemod1 =
ifelse(predoutmod1$Cult1>predoutmod1$Cult2,ifelse(predoutmod1$Cult1>predoutmod1$Cult
3,1,3),ifelse(predoutmod1$Cult2>predoutmod1$Cult3,2,3))
errorsmod1 = ifelse(pred_singlemod1==as.numeric(Response_Num),0,1)
sum(errorsmod1)  #16 errors using original dataset
table(NN_Prediction=pred_singlemod1,Truth=as.numeric(Response_Num))
#
plot(nnmod1)    # network diagram
### Plots
head(nnmod1$generalized.weights[[1]])
plota = gwplot(nnmod1,selected.covariate=1,selected.response=1)  # Could use gwplot, but it
only plots one response at a time.
plotb = gwplot(nnmod1,selected.covariate=1,selected.response=2)
plotc = gwplot(nnmod1,selected.covariate=1,selected.response=3)
nplot = 2
nplot = 13
for (i in 1:nplot){
  iname = names(nn0.non)[i]
  miny = min(df.gwmod1[[i]],df.gwmod1[[13+i]],df.gwmod1[[26+i]])-1
  maxy = max(df.gwmod1[[i]],df.gwmod1[[13+i]],df.gwmod1[[26+i]])+1
  plot1 = xyplot(df.gwmod1[[i]]~nn0.non[[i]],lwd=2,main=iname,pch=c(1),
col=c("blue"),ylim=c(miny, maxy),
                xlab = "Cultivars:   1 (o)     2 (X)     3 (+)",
                ylab = "Generalized Weight (Standardized)")
  plot2 = xyplot(df.gwmod1[[13+i]]~nn0.non[[i]],lwd=2,main=iname,pch=c(4),
col=c("red"),#ylim=c(.60, 1),
                xlab = "Value",
                ylab = "Generalized Weight (Standardized)")
  plot3 =
xyplot(df.gwmod1[[26+i]]~nn0.non[[i]],lwd=2,main=iname,pch=c(3),col=c("darkgreen"),#ylim=c
(.60, 1),
                xlab = "Value",
                ylab = "Generalized Weight (Standardized)")

  print(plot1 + as.layer(plot2)+ as.layer(plot3))
}
#..........................................................
#..........................................................
#.....    Original Data, Standardized with Std Dev ...........
#..........................................................
```

```
#.........................................................
nnmod3 <-
neuralnet(Cult1+Cult2+Cult3~Alcohol+MalicAcid+Ash+AlcalinityAsh+Magnesium+TotalPhenols+
Flavanoids+NonflavanoidPhenols+Proanthocyanins+ColorIntensity+Hue+OD280+Proline,
                data=nn0.stdsd, hidden=2, err.fct="ce",linear.output=FALSE)

nnmod3    # basic output
names(nnmod3)   # reults options
nnmod3$result.matrix     # result matrix
nnmod3$generalized.weights
df.gwmod3=data.frame(nnmod3$generalized.weights)
names(df.gwmod3) <-
list("Alcohol1","MalicAcid1","Ash1","AlcalinityAsh1","Magnesium1","TotalPhenols1","Flavanoid
s1","NonflavanoidPhenols1","Proanthocyanins1","ColorIntensity1","Hue1","OD2801","Proline1
","Alcohol2","MalicAcid2","Ash2","AlcalinityAsh2","Magnesium2","TotalPhenols2","Flavanoids2
","NonflavanoidPhenols2","Proanthocyanins2","ColorIntensity2","Hue2","OD2802","Proline2","
Alcohol3","MalicAcid3","Ash3","AlcalinityAsh3","Magnesium3","TotalPhenols3","Flavanoids3","
NonflavanoidPhenols3","Proanthocyanins3","ColorIntensity3","Hue3","OD2803","Proline3")
outmod3 <- cbind(nnmod3$covariate,nnmod3$net.result[[1]])
dimnames(outmod3) <- list(NULL,
c("Alcohol","MalicAcid","Ash","AlcalinityAsh","Magnesium","TotalPhenols","Flavanoids","Nonfl
avanoidPhenols","Proanthocyanins","ColorIntensity","Hue","OD280","Proline","Cult1","Cult2","
Cult3"))

names(nn0.std)
nnmod3.pred = data.frame(compute(nnmod3,nn0.stdsd[,-c(14:16)], rep=1))
predout =
data.frame(Cult1=nnmod3.pred$net.result.1,Cult2=nnmod3.pred$net.result.2,Cult3=nnmod3.p
red$net.result.3)
pred_singlemod3 =
ifelse(predout$Cult1>predout$Cult2,ifelse(predout$Cult1>predout$Cult3,1,3),ifelse(predout$C
ult2>predout$Cult3,2,3))
errorsmod3 = ifelse(pred_singlemod3==as.numeric(Response_Num),0,1)
sum(errorsmod3)     #0 errors using the standardized version
#
plot(nnmod3)    # network diagram
### Plots
head(nnmod3$generalized.weights[[1]])
plota = gwplot(nnmod3,selected.covariate=1,selected.response=1)  # Could use gwplot, but it
only plots one response at a time.
plotb = gwplot(nnmod3,selected.covariate=1,selected.response=2)
plotc = gwplot(nnmod3,selected.covariate=1,selected.response=3)
nplot=13
for (i in 1:nplot){
  iname = names(nn0.std)[i]
  miny = min(df.gwmod3[[i]],df.gwmod3[[13+i]],df.gwmod3[[26+i]])-1
  maxy = max(df.gwmod3[[i]],df.gwmod3[[13+i]],df.gwmod3[[26+i]])+1
  plot1 = xyplot(df.gwmod3[[i]]~nn0.std[[i]],lwd=2,main=iname,pch=c(1),
col=c("blue"),ylim=c(miny, maxy),
        xlab = "Cultivars:   1 (o)     2 (X)     3 (+)",
        ylab = "Generalized Weight")
  plot2 = xyplot(df.gwmod3[[13+i]]~nn0.std[[i]],lwd=2,main=iname,pch=c(4),
col=c("red"),#ylim=c(.60, 1),
        xlab = "Value",
        ylab = "Generalized Weight (Standardized)")
  plot3 =
xyplot(df.gwmod3[[26+i]]~nn0.std[[i]],lwd=2,main=iname,pch=c(3),col=c("darkgreen"),#ylim=c(
.60, 1),
        xlab = "Value",
        ylab = "Generalized Weight (Standardized)")

  print(plot1 + as.layer(plot2)+ as.layer(plot3))
}
#..........................................................
#..........................................................
#............ Now, Original Data, but Standardized.0-1..........
#..........................................................
#..........................................................
nnmod2 <-
neuralnet(Cult1+Cult2+Cult3~Alcohol+MalicAcid+Ash+AlcalinityAsh+Magnesium+TotalPhenols+
Flavanoids+NonflavanoidPhenols+Proanthocyanins+ColorIntensity+Hue+OD280+Proline,
        data=nn0.std, hidden=2, err.fct="ce",linear.output=FALSE)

nnmod2    # basic output
names(nnmod2)   # reults options
nnmod2$result.matrix     # result matrix
nnmod2$generalized.weights
mean(df.gwmod2$Alcohol1) #,df.gwmod2$Alcohol2,df.gwmod2$Alcohol3)
mean(df.gwmod2$Alcohol2)
mean(df.gwmod2$Alcohol3)

df.gwmod2=data.frame(nnmod2$generalized.weights)
names(df.gwmod2) <-
list("Alcohol1","MalicAcid1","Ash1","AlcalinityAsh1","Magnesium1","TotalPhenols1","Flavanoid
s1","NonflavanoidPhenols1","Proanthocyanins1","ColorIntensity1","Hue1","OD2801","Proline1
","Alcohol2","MalicAcid2","Ash2","AlcalinityAsh2","Magnesium2","TotalPhenols2","Flavanoids2
","NonflavanoidPhenols2","Proanthocyanins2","ColorIntensity2","Hue2","OD2802","Proline2","
```

```
Alcohol3","MalicAcid3","Ash3","AlcalinityAsh3","Magnesium3","TotalPhenols3","Flavanoids3","
NonflavanoidPhenols3","Proanthocyanins3","ColorIntensity3","Hue3","OD2803","Proline3")
outmod2 <- cbind(nnmod2$covariate,nnmod2$net.result[[1]])
dimnames(outmod2) <- list(NULL,
c("Alcohol","MalicAcid","Ash","AlcalinityAsh","Magnesium","TotalPhenols","Flavanoids","Nonfl
avanoidPhenols","Proanthocyanins","ColorIntensity","Hue","OD280","Proline","Cult1","Cult2","
Cult3"))

names(nn0.std)
nnmod2.pred = data.frame(compute(nnmod2,nn0.std[,-c(14:16)], rep=1))
predout =
data.frame(Cult1=nnmod2.pred$net.result.1,Cult2=nnmod2.pred$net.result.2,Cult3=nnmod2.p
red$net.result.3)
pred_singlemod2 =
ifelse(predout$Cult1>predout$Cult2,ifelse(predout$Cult1>predout$Cult3,1,3),ifelse(predout$C
ult2>predout$Cult3,2,3))
errorsmod2 = ifelse(pred_singlemod2==as.numeric(Response_Num),0,1)
sum(errorsmod2)    #0 errors using the standardized version
table(NN_Prediction=pred_singlemod2,Truth=as.numeric(Response_Num))
#
plot(nnmod2)    # network diagram
### Plots
head(nnmod2$generalized.weights[[1]])
plota = gwplot(nnmod2,selected.covariate=1,selected.response=1)  # Could use gwplot, but it
only plots one response at a time.
plotb = gwplot(nnmod2,selected.covariate=1,selected.response=2)
plotc = gwplot(nnmod2,selected.covariate=1,selected.response=3)
nplot=13
for (i in 1:nplot){
  iname = names(nn0.std)[i]
  miny = min(df.gwmod2[[i]],df.gwmod2[[13+i]],df.gwmod2[[26+i]])-1
  maxy = max(df.gwmod2[[i]],df.gwmod2[[13+i]],df.gwmod2[[26+i]])+1
  plot1 = xyplot(df.gwmod2[[i]]~nn0.std[[i]],lwd=2,main=iname,pch=c(1),
col=c("blue"),ylim=c(miny, maxy),
              xlab = "Cultivars:   1 (o)     2 (X)     3 (+)",
              ylab = "Generalized Weight")
  plot2 = xyplot(df.gwmod2[[13+i]]~nn0.std[[i]],lwd=2,main=iname,pch=c(4),
col=c("red"),#ylim=c(.60, 1),
              xlab = "Value",
              ylab = "Generalized Weight (Standardized)")
  plot3 =
xyplot(df.gwmod2[[26+i]]~nn0.std[[i]],lwd=2,main=iname,pch=c(3),col=c("darkgreen"),#ylim=c(
.60, 1),
              xlab = "Value",
              ylab = "Generalized Weight (Standardized)")

  print(plot1 + as.layer(plot2)+ as.layer(plot3))
}
#-------------------------------------------------------------------------------
#
#       INCREASING THE NUMBER OF HIDDEN NEURONS to 4
#
#-------------------------------------------------------------------------------

nnmod2 <-
neuralnet(Cult1+Cult2+Cult3~Alcohol+MalicAcid+Ash+AlcalinityAsh+Magnesium+TotalPhenols+
Flavanoids+NonflavanoidPhenols+Proanthocyanins+ColorIntensity+Hue+OD280+Proline,
                data=nn0.std, hidden=4, err.fct="ce",linear.output=FALSE)

nnmod2    # basic output
names(nnmod2)   # reults options
nnmod2$result.matrix     # result matrix
nnmod2$generalized.weights
mean(df.gwmod2$Alcohol1) #,df.gwmod2$Alcohol2,df.gwmod2$Alcohol3)
mean(df.gwmod2$Alcohol2)
mean(df.gwmod2$Alcohol3)

df.gwmod2=data.frame(nnmod2$generalized.weights)
names(df.gwmod2) <-
list("Alcohol1","MalicAcid1","Ash1","AlcalinityAsh1","Magnesium1","TotalPhenols1","Flavanoid
s1","NonflavanoidPhenols1","Proanthocyanins1","ColorIntensity1","Hue1","OD2801","Proline1
","Alcohol2","MalicAcid2","Ash2","AlcalinityAsh2","Magnesium2","TotalPhenols2","Flavanoids2
","NonflavanoidPhenols2","Proanthocyanins2","ColorIntensity2","Hue2","OD2802","Proline2","
Alcohol3","MalicAcid3","Ash3","AlcalinityAsh3","Magnesium3","TotalPhenols3","Flavanoids3","
NonflavanoidPhenols3","Proanthocyanins3","ColorIntensity3","Hue3","OD2803","Proline3")
outmod2 <- cbind(nnmod2$covariate,nnmod2$net.result[[1]])
dimnames(outmod2) <- list(NULL,
c("Alcohol","MalicAcid","Ash","AlcalinityAsh","Magnesium","TotalPhenols","Flavanoids","Nonfl
avanoidPhenols","Proanthocyanins","ColorIntensity","Hue","OD280","Proline","Cult1","Cult2","
Cult3"))

names(nn0.std)
nnmod2.pred = data.frame(compute(nnmod2,nn0.std[,-c(14:16)], rep=1))
predout =
data.frame(Cult1=nnmod2.pred$net.result.1,Cult2=nnmod2.pred$net.result.2,Cult3=nnmod2.p
red$net.result.3)
```

```
pred_singlemod2 =
ifelse(predout$Cult1>predout$Cult2,ifelse(predout$Cult1>predout$Cult3,1,3),ifelse(predout$C
ult2>predout$Cult3,2,3))
errorsmod2 = ifelse(pred_singlemod2==as.numeric(Response_Num),0,1)
sum(errorsmod2)    #0 errors using the standardized version
table(NN_Prediction=pred_singlemod2,Truth=as.numeric(Response_Num))
#
plot(nnmod2)    # network diagram
### Plots
head(nnmod2$generalized.weights[[1]])
plota = gwplot(nnmod2,selected.covariate=1,selected.response=1)  # Could use gwplot, but it
only plots one response at a time.
plotb = gwplot(nnmod2,selected.covariate=1,selected.response=2)
plotc = gwplot(nnmod2,selected.covariate=1,selected.response=3)
nplot=13
for (i in 1:nplot){
  iname = names(nn0.std)[i]
  miny = min(df.gwmod2[[i]],df.gwmod2[[13+i]],df.gwmod2[[26+i]])-1
  maxy = max(df.gwmod2[[i]],df.gwmod2[[13+i]],df.gwmod2[[26+i]])+1
  plot1 = xyplot(df.gwmod2[[i]]~nn0.std[[i]],lwd=2,main=iname,pch=c(1),
col=c("blue"),ylim=c(miny, maxy),
              xlab = "Cultivars:   1 (o)     2 (X)     3 (+)",
              ylab = "Generalized Weight")
  plot2 = xyplot(df.gwmod2[[13+i]]~nn0.std[[i]],lwd=2,main=iname,pch=c(4),
col=c("red"),#ylim=c(.60, 1),
              xlab = "Value",
              ylab = "Generalized Weight (Standardized)")
  plot3 =
xyplot(df.gwmod2[[26+i]]~nn0.std[[i]],lwd=2,main=iname,pch=c(3),col=c("darkgreen"),#ylim=c(
.60, 1),
              xlab = "Value",
              ylab = "Generalized Weight (Standardized)")

  print(plot1 + as.layer(plot2)+ as.layer(plot3))
}
########################## MANUAL CROSS VALIDATION of NEURAL NET
set.seed(16)
k <- 100
for (hiddennodes in 3:10){
  totalerr = 0
  for(i in 1:k){
    nnmod2 = NULL
    nnmod2.pred=NULL
    index <- sample(1:nrow(nn0.std),round(0.8*nrow(nn0.std)))
    train.cv <- nn0.std[index,]
    test.cv <- nn0.std[-index,]

    nnmod2 <-
neuralnet(Cult1+Cult2+Cult3~Alcohol+MalicAcid+Ash+AlcalinityAsh+Magnesium+TotalPhenols+
Flavanoids+NonflavanoidPhenols+Proanthocyanins+ColorIntensity+Hue+OD280+Proline,
                    data=train.cv, hidden=hiddennodes, err.fct="ce",linear.output=FALSE)

    nnmod2.pred = data.frame(compute(nnmod2,test.cv[,-c(14:16)], rep=1))
    predout =
data.frame(Cult1=nnmod2.pred$net.result.1,Cult2=nnmod2.pred$net.result.2,Cult3=nnmod2.p
red$net.result.3)
    pred_singlemod2 =
ifelse(predout$Cult1>predout$Cult2,ifelse(predout$Cult1>predout$Cult3,1,3),ifelse(predout$C
ult2>predout$Cult3,2,3))
    errorsmod2 = ifelse(pred_singlemod2==as.numeric(Response_Num[-index]),0,1)

    # print(table(NN_Pred=pred_singlemod2,Truth=as.numeric(Response_Num[-index])))

    totalerr = totalerr + sum(errorsmod2)    #0 errors using the standardized version
  }
  outstr = paste("Hidden: ",hiddennodes," TotalErr:",totalerr,sep="")
  print(outstr)
}
#For 20% x 10 runs, 2 nodes: sum(8,24,3,6,9,8,11,11,4,9) = sum(93+109)*5
#For 20% x 10 runs, 3 nodes: sum(13,1,8,8,12,8,12,5,3,11) =81
#For 20% x 10 runs, 4 nodes: sum(10,9,8,6,7,9,4,7,6,7) = 73
#For 20% x 10 runs, 5 nodes: sum(9,11,8,14,4,5,12,7,6,11) = 87
#For 20% x 10 runs, 5 nodes: sum(10,8,6,10,10,6,9,5,8,7) = 79
#1000 Samples
#2:1010
#3:800
#4:727
#5:696
#6:684
#7:725
#8:663
#9:707
#10:669
dfResults = data.frame(c(2,3,4,5,6,7,8,9,10),c(1010,800,727,696,684,725,663,707,669))
names(dfResults)[1]="HiddenNodes"
names(dfResults)[2]="ErrSum"
plot1 = xyplot(dfResults$ErrSum~dfResults$HiddenNodes,type="a",lwd=2,pch=c(18),
col=c("blue"),
            main = "Error from 1000 CV Runs (Trained with 80% Data)",
            xlab = "Number of Hidden Nodes",
            ylab = "Total Misclassification Error Sum")
#-----------------------------------------------------------------------------
#
#       Support Vector Machine
#
#-----------------------------------------------------------------------------
# For multiclass-classification with k levels, k>2, libsvm uses the 'one-against-one'-approach, in
which k(k-1)/2 binary
# classifiers are trained; the appropriate class is found by a voting scheme.
dfsvm=svm.non
names(dfsvm)
#
set.seed(1)
#ctrl <- trainControl(method = "cv", number=50)
#svmfit <- train(Cultivar~., data=dfsvm, method = "svmRadial", trControl = ctrl)

summary(svmfit$finalModel)
names(svmfit)
svmfit$bestTune

svmfit$coefnames
svmfit$modelInfo


set.seed(1) # This proves that the tune call standardizes the variables or at least the result is the
same.
svm_tune <- tune(svm, Cultivar~.,data=svm.stdsd, kernel="radial", ranges=list(cost = 2^c(-8,-4,-
2,0,1,2,3,4), gamma=c(.1,.5, 1)))
print(svm_tune)
plot(svm_tune, transform.x = log2)

set.seed(1)
svm_tune <- tune(svm, Cultivar~.,data=dfsvm, kernel="radial", ranges=list(cost = 2^c(-8,-4,-
2,0,1,2,3,4), gamma=c(.1,.5, 1)))
print(svm_tune)
plot(svm_tune, transform.x = log2)
#plot(svm_tune, type = "perspective", theta = 20, phi = 25)
# reportedly best: cost:.25   gamma: 0.1
# But this still has too much bias as shown here:
svmfit=svm(Cultivar~., data=dfsvm, method="C-classification", kernel="radial",
cost=.25,gamma=.1,scale=TRUE)
pred_SVM=as.numeric(predict(svmfit,dfsvm))
```

```
table(Prediction=pred_SVM,Truth=dfsvm$Cultivar)
svmfit
# With the standardization, it makes 2 errors.  (Without it, it guesses category 2 for every
record!!)
# Increasing cost to .5 reduces 1 error:
svmfit=svm(Cultivar~., data=dfsvm, method="C-classification", kernel="radial",
cost=.5,gamma=.1,scale=TRUE)
pred_SVM=as.numeric(predict(svmfit,dfsvm))
table(Prediction=pred_SVM,Truth=dfsvm$Cultivar)
svmfit
# Increasign cost to 1, reduces errors to 0.  BUT THIS DOESN"T TELL US WHAT HAPPENS ON
TEST SET!!! WE HAVE NO TEST SET!!
svmfit=svm(Cultivar~., data=dfsvm, method="C-classification", kernel="radial",
cost=1,gamma=.1,scale=TRUE)
pred_SVM=as.numeric(predict(svmfit,dfsvm))
table(Prediction=pred_SVM,Truth=dfsvm$Cultivar)
svmfit

# Increasign cost to 1, reduces errors to 0.  BUT THIS DOESN"T TELL US WHAT HAPPENS ON
TEST SET!!! WE HAVE NO TEST SET!!
svmfit=svm(Cultivar~., data=dfsvm, method="C-classification", kernel="radial",
cost=2,gamma=.1,scale=TRUE)
pred_SVM=as.numeric(predict(svmfit,dfsvm))
table(Prediction=pred_SVM,Truth=dfsvm$Cultivar)
svmfit

#
# high cost = few support vectors as each is influential. Thus, more variance because fewer
supports, but lower bias.
# small cost = lot more supports, so less variance but higher bias
# gamma is how to maximize gain from projection.  Large gamma shrinks the influence of the
support vector,
# small cost = high bias, small variance - big cost tolerates more mistakes, so more stable.
# small gamma = small bias, but high variance
#Increasing the cost while holding the gamma steady corrects the situation:
svmfit=svm(Cultivar~., data=dfsvm, method="C-classification", kernel="radial",
cost=1,gamma=.1,scale=TRUE)
pred_SVM=as.numeric(predict(svmfit,dfsvm))
table(Prediction=pred_SVM,Truth=dfsvm$Cultivar)
?svm
?plot
svmfit
plot(svmfit,dfsvm,ColorIntensity~Proline)
plot(svmfit,dfsvm,MalicAcid~Proline)
plot(svmfit,dfsvm,Flavanoids~Proline)
plot(svmfit,dfsvm,Flavanoids~Hue)
plot(svmfit,dfsvm,Flavanoids~OD280)
names(svmfit)
svmfit$tot.nSV
svmfit$SV

#-----------------------------------------------------------------------------
#
#       Random Forest
#
#-----------------------------------------------------------------------------
#
dfrf = df.orig
dfrf$Cultivar = as.factor(dfrf$Cultivar)
classerror = function(mydf,truth){
  sum(ifelse(mydf!=truth,1,0))
}
ctrlist = NULL;
varslist = NULL;
errorlist = NULL;
for (j in 2:5){
  for (i in c(1:15,20,30,40,50)){
    set.seed(1)
    rfmdl=randomForest(Cultivar~.,data=dfrf,mtry=j,importance=TRUE,ntree=i)
    predict.trn <- as.numeric((predict(rfmdl, newdata=dfrf, se.fit = TRUE)))
    ctrlist = c(ctrlist,i)
    varslist = c(varslist,j)
    errorlist = c(errorlist,classerror(predict.trn,dfrf$Cultivar))
  }
}

Perfdata = data.frame(ctrlist,varslist,errorlist)

plot2 =
xyplot(Perfdata$errorlist[varslist==2]~Perfdata$ctrlist[varslist==2],type='a',lwd=3,ylim=c(-1,12),
xlim=c(0,20),
                main = "Errors by Variable and Tree Counts",
                key=list(columns=5,
                        text=list(lab=c("Var
Count:","Two","Three","Four","Five"),col=c("black","blue","red","green","brown")),
                        points=list(pch=c(NA,
18,18,18,18),col=c("black","blue","red","green","brown"))),
                xlab = "Tree Count",
            ylab = "Misclassifications")
plot3 =
xyplot(Perfdata$errorlist[varslist==3]~Perfdata$ctrlist[varslist==3],type='a',lwd=3,ylim=c(-
1,12),col=c("red"))
plot4 =
xyplot(Perfdata$errorlist[varslist==4]~Perfdata$ctrlist[varslist==4],type='a',lwd=3,ylim=c(-
1,12),col=c("green"))
plot5 =
xyplot(Perfdata$errorlist[varslist==5]~Perfdata$ctrlist[varslist==5],type='a',lwd=3,ylim=c(-
1,12),col=c("brown"))

plot2+as.layer(plot4)+as.layer(plot5)+as.layer(plot3)

###  RF MODEL
#
rfmdl=randomForest(Cultivar~.,data=dfrf,mtry=3,importance=TRUE,ntree=100)
predict.trn <- as.numeric((predict(rfmdl, newdata=dfrf, se.fit = TRUE)))
classerror(predict.trn,dfrf$Cultivar)
table(RF_Predict=predict.trn,Truth=dfrf$Cultivar)

names(rfmdl)
impvalues = data.frame(rfmdl$importance)

plot1 = xyplot(impvalues$X1~impvalues$X2,lwd=3,ylim=c(-.01,.3), xlim=c(-
.01,.11),pch=c(1),col=c("blue"),
          main = "Variable Importance for Cultivars 1 & 2",
          key=list(columns=7,

text=list(lab=c(NA,"Flavanoids","Proline","Alcohol","ColIntensity","OD280"),col=c(NA,"green","r
ed","purple","orange","blue")),

points=list(pch=c(NA,19,19,19,19,19,NA),col=c(NA,"green","red","purple","orange","blue",NA))
),
          xlab = "Variable 2 Importance",
          ylab = "Variable 1 Importance")
plot2 = xyplot(impvalues$X1[13]~impvalues$X2[13],lwd=3,ylim=c(-.01,.3), xlim=c(-
.01,.11),pch=c(19),col=c("red"))
plot3 = xyplot(impvalues$X1[10]~impvalues$X2[10],lwd=3,ylim=c(-.01,.3), xlim=c(-
.01,.11),pch=c(19),col=c("orange"))
plot4 = xyplot(impvalues$X1[7]~impvalues$X2[7],lwd=3,ylim=c(-.01,.3), xlim=c(-
.01,.11),pch=c(19),col=c("green"))
plot5 = xyplot(impvalues$X1[1]~impvalues$X2[1],lwd=3,ylim=c(-.01,.3), xlim=c(-
.01,.11),pch=c(19),col=c("purple"))
```

```
plot6 = xyplot(impvalues$X1[12]~impvalues$X2[12],lwd=3,ylim=c(-.01,.3), xlim=c(-
.01,.11),pch=c(19),col=c("blue"))
plot1+as.layer(plot2)+as.layer(plot3)+as.layer(plot4)+as.layer(plot5)+as.layer(plot6)


#*************************************************************
#
#      GRAPHICS
#
#*************************************************************
ylist =
c("Proline","Proline","Proline","Proline","Proline","Proline","Flavanoids","Flavanoids","Flavanoi
ds","Flavanoids","Flavanoids","OD280","OD280","MalicAcid")
xlist =
c("Flavanoids","OD280","MalicAcid","Magnesium","Hue","Alcohol","OD280","MalicAcid","Mag
nesium","Hue","Alcohol","Hue","Alcohol","Magnesium")

for (i in 1:length(ylist)){
  y=ylist[i]
  x=xlist[i]
  miny = min(df.orig[[y]])
  maxy = max(df.orig[[y]])
  minx = min(df.orig[[x]])
  maxx = max(df.orig[[x]])
  iname="Proline-Alcohol"
  title = paste(x," - ",y," by Cultivar",sep="")
  plot1 =
xyplot(df.orig[[y]][df.orig$Cultivar==1]~df.orig[[x]][df.orig$Cultivar==1],lwd=2,main=title,pch=c(
18), col=c("blue"),
              xlim=c(minx,maxx),ylim=c(miny,maxy),
              key=list(columns=3,
                     text=list(lab=c("1:","2:","3:"),col=c("blue","red","green")),
                     points=list(pch=c(18,18,18), col=c("blue","red","green"))),
              xlab = x,
              ylab = y)
  plot2 =
xyplot(df.orig[[y]][df.orig$Cultivar==2]~df.orig[[x]][df.orig$Cultivar==2],lwd=2,main=title,pch=c(
18), col=c("red"),
              xlim=c(minx,maxx),ylim=c(miny,maxy))
  plot3 =
xyplot(df.orig[[y]][df.orig$Cultivar==3]~df.orig[[x]][df.orig$Cultivar==3],lwd=2,main=title,pch=c(
18), col=c("green"),
              xlim=c(minx,maxx),ylim=c(miny,maxy))
  print(plot1 + as.layer(plot2)+ as.layer(plot3))
  print(plot1 + as.layer(plot2))

}

#plot1 = cloud(Proline ~
Alcohol+OD280,data=df.orig[df.orig$Cultivar==1,],lwd=2,main=title,pch=c(18),
col=c("blue"),screen=list(z=45, x =45, y=45))
#plot2 = cloud(Proline ~
Alcohol+OD280,data=df.orig[df.orig$Cultivar==2,],lwd=2,main=title,pch=c(18),
col=c("red"),screen=list(z=45, x =45, y=45))
#print(plot1 + as.layer(plot2))
#
############################################
############################################
############################################
###### Graphics for Exploratory data
############################################
#
densityplot(~Alcohol, Wine, plot.points=FALSE,lwd=10)
densityplot(~MalicAcid, Wine, plot.points=FALSE,lwd=10)
densityplot(~Ash, Wine, plot.points=FALSE,lwd=10)
densityplot(~AlcalinityAsh, Wine, plot.points=FALSE,lwd=10)
densityplot(~Magnesium, Wine, plot.points=FALSE,lwd=10)
densityplot(~TotalPhenols, Wine, plot.points=FALSE,lwd=10)
densityplot(~Flavanoids, Wine, plot.points=FALSE,lwd=10)
densityplot(~NonflavanoidPhenols, Wine, plot.points=FALSE,lwd=10)
densityplot(~Proanthocyanins, Wine, plot.points=FALSE,lwd=10)
densityplot(~ColorIntensity, Wine, plot.points=FALSE,lwd=10)
densityplot(~Hue, Wine, plot.points=FALSE,lwd=10)
densityplot(~OD280, Wine, plot.points=FALSE,lwd=10)
densityplot(~Proline, Wine, plot.points=FALSE,lwd=10)

key=list(space="right",
        lines=list(col=c("red","darkgreen","blue"), lty=c(3,2), lwd=6),
        text=list(c("Purple Line"," Dark-green Line"))
)
densityplot(~Alcohol, Wine, plot.points=FALSE,groups=Wine$Cultivar,
col=c("red","green","blue"), lwd=5)
densityplot(~MalicAcid, Wine, plot.points=FALSE,groups=Wine$Cultivar,lwd=5,
col=c("red","green","blue"))
densityplot(~Ash, Wine, plot.points=FALSE,groups=Wine$Cultivar,lwd=5,
col=c("red","green","blue"))
```

```
densityplot(~AlcalinityAsh, Wine, plot.points=FALSE,groups=Wine$Cultivar,lwd=5,
col=c("red","green","blue"))
densityplot(~Magnesium, Wine, plot.points=FALSE,groups=Wine$Cultivar,lwd=5,
col=c("red","green","blue"))
densityplot(~TotalPhenols, Wine, plot.points=FALSE,groups=Wine$Cultivar,lwd=5,
col=c("red","green","blue"))
densityplot(~Flavanoids, Wine, plot.points=FALSE,groups=Wine$Cultivar,lwd=5,
col=c("red","green","blue"))
densityplot(~NonflavanoidPhenols, Wine, plot.points=FALSE,groups=Wine$Cultivar,lwd=5,
col=c("red","green","blue"))
densityplot(~Proanthocyanins, Wine, plot.points=FALSE,groups=Wine$Cultivar,lwd=5,
col=c("red","green","blue"))
densityplot(~ColorIntensity, Wine, plot.points=FALSE,groups=Wine$Cultivar,lwd=5,
col=c("red","green","blue"))
densityplot(~Hue, Wine, plot.points=FALSE,groups=Wine$Cultivar,lwd=5,
col=c("red","green","blue"))
densityplot(~OD280, Wine, plot.points=FALSE,groups=Wine$Cultivar,lwd=5,
col=c("red","green","blue"))
densityplot(~Proline, Wine, plot.points=FALSE,groups=Wine$Cultivar,lwd=5,
col=c("red","green","blue"))

boxplot(Alcohol~Cultivar,data=Wine,main="Alcohol",col=c("red","green","blue"))
boxplot(MalicAcid~Cultivar,data=Wine,main="Malic Acid",col=c("red","green","blue"))
boxplot(Ash~Cultivar,data=Wine,main="Ash",col=c("red","green","blue"))
boxplot(AlcalinityAsh~Cultivar,data=Wine,main="Alcalinity Ash",col=c("red","green","blue"))
boxplot(Magnesium~Cultivar,data=Wine,main="Magnesium",col=c("red","green","blue"))
boxplot(TotalPhenols~Cultivar,data=Wine,main="Total Phenols",col=c("red","green","blue"))
boxplot(Flavanoids~Cultivar,data=Wine,main="Flavanoids",col=c("red","green","blue"))
boxplot(NonflavanoidPhenols~Cultivar,data=Wine,main="Nonflav
Phenols",col=c("red","green","blue"))
boxplot(Proanthocyanins~Cultivar,data=Wine,main="Proanthocyanins",col=c("red","green","bl
ue"))
boxplot(ColorIntensity~Cultivar,data=Wine,main="Color Intensity",col=c("red","green","blue"))
boxplot(Hue~Cultivar,data=Wine,main="Hue",col=c("red","green","blue"))
boxplot(OD280~Cultivar,data=Wine,main="OD280",col=c("red","green","blue"))
boxplot(Proline~Cultivar,data=Wine,main="Proline",col=c("red","green","blue"))
################################################################################
####################### LEAVE ONE OUT FINAL RESULTS ############################
################################################################################
########################## Leave One Out - All Variables Neural Net
set.seed(1)
k <- nrow(nn0.std)
mycm = data.frame(T1=c(0,0,0),T2=c(0,0,0),T3=c(0,0,0))
totalerr = 0
```

```
for(i in 1:k){
  nnmod2 = NULL
  nnmod2.pred=NULL
  index <- sample(1:nrow(nn0.std),round(0.8*nrow(nn0.std)))
  train.cv <- nn0.std[-i,]
  test.cv <- nn0.std[i,]

  nnmod2 <-
neuralnet(Cult1+Cult2+Cult3~Alcohol+MalicAcid+Ash+AlcalinityAsh+Magnesium+TotalPhenols+
Flavanoids+NonflavanoidPhenols+Proanthocyanins+ColorIntensity+Hue+OD280+Proline,
                  data=train.cv, hidden=6, err.fct="ce",linear.output=FALSE)
  nnmod2.pred = data.frame(compute(nnmod2,test.cv[,-c(14:16)], rep=1))
  predout =
data.frame(Cult1=nnmod2.pred$net.result.1,Cult2=nnmod2.pred$net.result.2,Cult3=nnmod2.p
red$net.result.3)
  pred_singlemod2 =
ifelse(predout$Cult1>predout$Cult2,ifelse(predout$Cult1>predout$Cult3,1,3),ifelse(predout$C
ult2>predout$Cult3,2,3))
  errorsmod2 = ifelse(pred_singlemod2==as.numeric(Response_Num[i]),0,1)

mycm[pred_singlemod2,as.numeric(Response_Num[i])]=mycm[pred_singlemod2,as.numeric(R
esponse_Num[i])]+1  #create confusion matrix
  totalerr = totalerr + sum(errorsmod2)    #0 errors using the standardized version
}
outstr = paste("6 Hidden Nodes TotalErr:",totalerr,sep="")
print("Neural Net - Full Model")
print(outstr)
print(mycm)


########################## Leave One Out - Revised Neural Net
nn0r.std = nn0.std[,-c(2:6,8:9)]
names(nn0r.std)
set.seed(1)
k <- nrow(nn0r.std)
mycm = data.frame(T1=c(0,0,0),T2=c(0,0,0),T3=c(0,0,0))
totalerr = 0
  for(i in 1:k){
    nnmod2 = NULL
    nnmod2.pred=NULL
    index <- sample(1:nrow(nn0.std),round(0.8*nrow(nn0r.std)))
    train.cv <- nn0r.std[-i,]
    test.cv <- nn0r.std[i,]
    nnmod2 <-
neuralnet(Cult1+Cult2+Cult3~Alcohol+Flavanoids+ColorIntensity+Hue+OD280+Proline,
                    data=train.cv, hidden=4, err.fct="ce",linear.output=FALSE)

    nnmod2.pred = data.frame(compute(nnmod2,test.cv[,-c(7:9)], rep=1))
    predout =
data.frame(Cult1=nnmod2.pred$net.result.1,Cult2=nnmod2.pred$net.result.2,Cult3=nnmod2.p
red$net.result.3)
    pred_singlemod2 =
ifelse(predout$Cult1>predout$Cult2,ifelse(predout$Cult1>predout$Cult3,1,3),ifelse(predout$C
ult2>predout$Cult3,2,3))
    errorsmod2 = ifelse(pred_singlemod2==as.numeric(Response_Num[i]),0,1)

mycm[pred_singlemod2,as.numeric(Response_Num[i])]=mycm[pred_singlemod2,as.numeric(R
esponse_Num[i])]+1  #create confusion matrix
    totalerr = totalerr + sum(errorsmod2)    #0 errors using the standardized version
  }
  outstr = paste("3 Hidden Nodes TotalErr:",totalerr,sep="")
  print("Neural Net - Reduced Model")
  print(outstr)
  print(mycm)

#-----------------------------------------------------------------------------
#
#       Support Vector Machine  - Leave One Out Score - Simplified Model
#
#-----------------------------------------------------------------------------
dfsvm=svm.non
k <- nrow(dfsvm)
mycm = data.frame(T1=c(0,0,0),T2=c(0,0,0),T3=c(0,0,0))
totalerr = 0
  for(i in 1:k){
    train.cv <- dfsvm[-i,]
    test.cv <- dfsvm[i,]
    svmfit=svm(Cultivar~Proline+Flavanoids+Alcohol+ColorIntensity+OD280+Hue, data=train.cv,
method="C-classification", kernel="radial", cost=16,gamma=.1,scale=TRUE)
    pred_SVM=as.numeric(predict(svmfit,test.cv))
    errorsmod2 = ifelse(pred_SVM==as.numeric(Response_Num[i]),0,1)

mycm[pred_SVM,as.numeric(Response_Num[i])]=mycm[pred_SVM,as.numeric(Response_Num
[i])]+1  #create confusion matrix
    totalerr = totalerr + sum(errorsmod2)    #0 errors using the standardized version
  }
  outstr = paste(k,"total recs, TotalErr:",totalerr,sep="")
  print("Support Vector Machine - Reduced Model")
  print(outstr)
  print(mycm)
  #-----------------------------------------------------------------------------
  #
  #       Support Vector Machine  - Leave One Out Score - Comprehensive Model
  #
  #-----------------------------------------------------------------------------
  dfsvm=svm.non
  k <- nrow(dfsvm)
  totalerr = 0
  mycm = data.frame(T1=c(0,0,0),T2=c(0,0,0),T3=c(0,0,0))
  for(i in 1:k){
    train.cv <- dfsvm[-i,]
    test.cv <- dfsvm[i,]
    svmfit=svm(Cultivar~., data=train.cv, method="C-classification", kernel="radial",
cost=1,gamma=.1,scale=TRUE)
    pred_SVM=as.numeric(predict(svmfit,test.cv))
    errorsmod2 = ifelse(pred_SVM==as.numeric(Response_Num[i]),0,1)

mycm[pred_SVM,as.numeric(Response_Num[i])]=mycm[pred_SVM,as.numeric(Response_Num
[i])]+1  #create confusion matrix
    totalerr = totalerr + sum(errorsmod2)    #0 errors using the standardized version
  }
  outstr = paste(k,"total recs, TotalErr:",totalerr,sep="")
  print("Support Vector Machine - Full Model")
  print(outstr)
  print(mycm)

#-----------------------------------------------------------------------------
#
#       Random Forest  - Leave One Out Score
#
#-----------------------------------------------------------------------------
#
dfrf = df.orig
dfrf$Cultivar = as.factor(dfrf$Cultivar)
mycm = data.frame(T1=c(0,0,0),T2=c(0,0,0),T3=c(0,0,0))
k <- nrow(dfrf)
totalerr = 0
for(i in 1:k){
  train.cv <- dfrf[-i,]
  test.cv <- dfrf[i,]
  rfmdl=randomForest(Cultivar~.,data=train.cv,mtry=2,importance=TRUE,ntree=1000)
  predict_rf <- as.numeric((predict(rfmdl, newdata=test.cv, se.fit = TRUE)))
  errorsmod2 = ifelse(predict_rf==as.numeric(Response_Num[i]),0,1)
  totalerr = totalerr + sum(errorsmod2)    #0 errors using the standardized version

mycm[predict_rf,as.numeric(Response_Num[i])]=mycm[predict_rf,as.numeric(Response_Num[i
])]+1  #create confusion matrix
}

outstr = paste(k,"total recs, TotalErr:",totalerr,sep="")
print("Random Forest - Full Model")
print(outstr)
print(mycm)
```