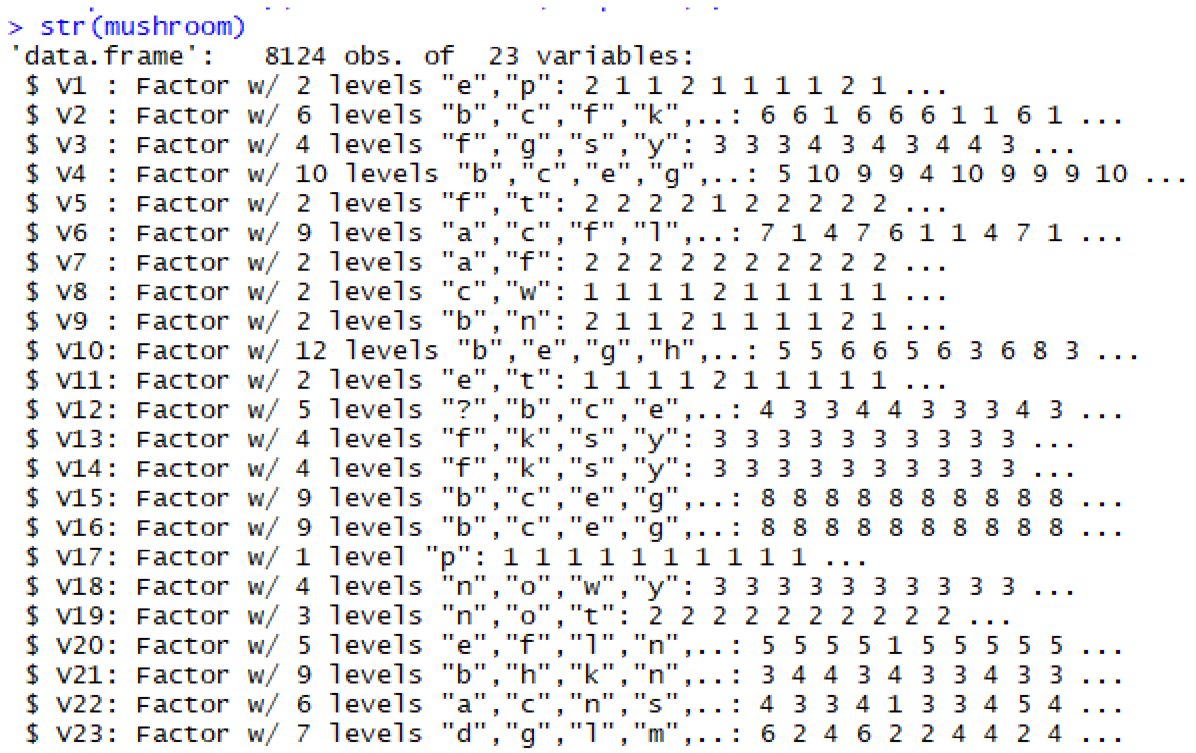
MACHINE LEARNING SVM vs. ANN Exploration

Dr. ALEXJANDRO DAVIANO

Today I want to experiment with an understand the differences between K-means and hierarchical clustering algorithms (HCAs). We will be able to do this based on the packages such as nnet, e1071, and caret that may help with the assignment. Other packages may be used as well. The package "gamlss" may be useful to help visualize the plots of the SVM and neural networks, while the "gamlss.add" package would also be helpful in being more visual for interpretation. We will use these packages to predict the accuracy of the packages as well as which kernels for the SVM may be better and how many levels may be better for the neural networks.

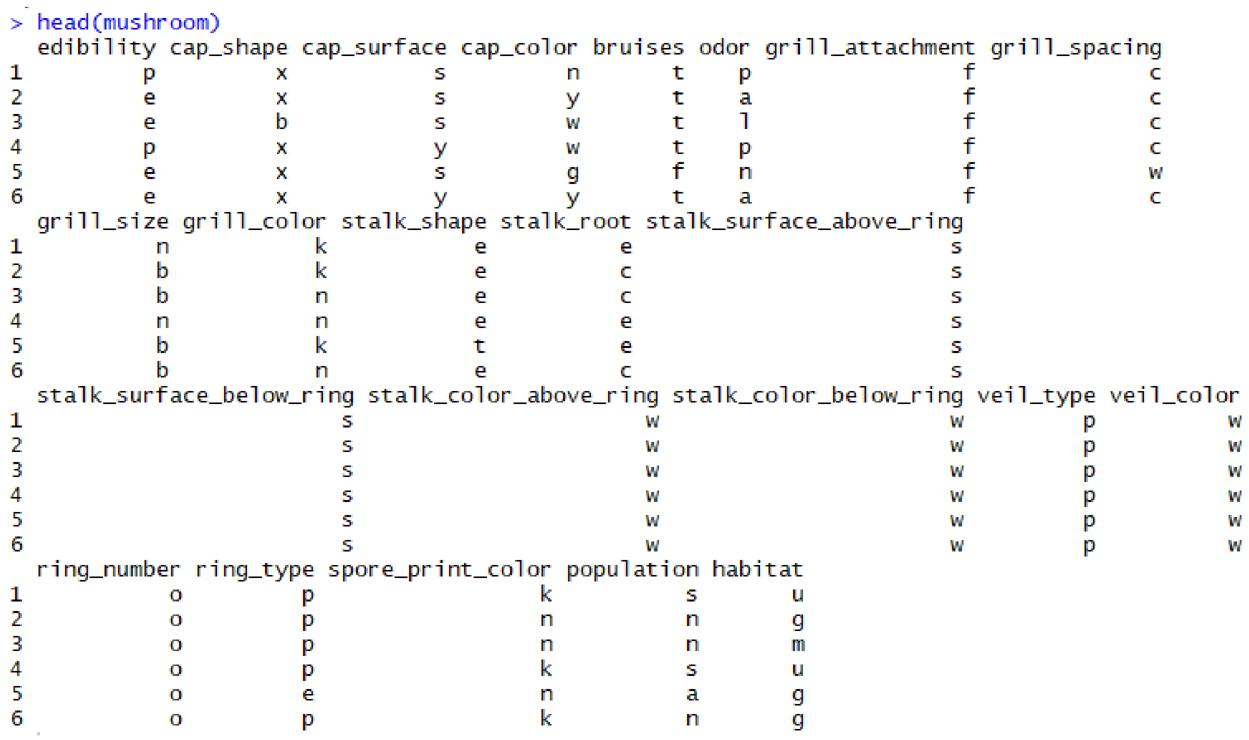
Methods and Results/interpretations

First, we will upload our data from the website directly in order to be more efficient with the download. We then want to look at the structure of the data in order to make sure it loaded correctly as well as to visualize what variables we may have to convert or impute. To do this I will also use the str(), table, hist, and mice functions to complete these tasks and to help me get a visualization of the data under each variable.



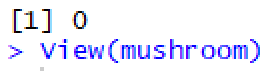
```
> str(mushroom)
'data.frame':   8124 obs. of  23 variables:
 $ V1 : Factor w/ 2 levels "e","p": 2 1 1 2 1 1 1 1 2 1 ...
 $ V2 : Factor w/ 6 levels "b","c","f","k",..: 6 6 1 6 6 6 1 1 6 1 ...
 $ V3 : Factor w/ 4 levels "f","g","s","y": 3 3 3 4 3 4 3 4 4 3 ...
 $ V4 : Factor w/ 10 levels "b","c","e","g",..: 5 10 9 9 4 10 9 9 9 10 ...
 $ V5 : Factor w/ 2 levels "f","t": 2 2 2 2 1 2 2 2 2 2 ...
 $ V6 : Factor w/ 9 levels "a","c","f","l",..: 7 1 4 7 6 1 1 4 7 1 ...
 $ V7 : Factor w/ 2 levels "a","f": 2 2 2 2 2 2 2 2 2 2 ...
 $ V8 : Factor w/ 2 levels "c","w": 1 1 1 1 2 1 1 1 1 1 ...
 $ V9 : Factor w/ 2 levels "b","n": 2 1 1 2 1 1 1 1 2 1 ...
 $ V10: Factor w/ 12 levels "b","e","g","h",..: 5 5 6 6 5 6 3 6 8 3 ...
 $ V11: Factor w/ 2 levels "e","t": 1 1 1 1 2 1 1 1 1 1 ...
 $ V12: Factor w/ 5 levels "?","b","c","e",..: 4 3 3 4 4 3 3 3 4 3 ...
 $ V13: Factor w/ 4 levels "f","k","s","y": 3 3 3 3 3 3 3 3 3 3 ...
 $ V14: Factor w/ 4 levels "f","k","s","y": 3 3 3 3 3 3 3 3 3 3 ...
 $ V15: Factor w/ 9 levels "b","c","e","g",..: 8 8 8 8 8 8 8 8 8 8 ...
 $ V16: Factor w/ 9 levels "b","c","e","g",..: 8 8 8 8 8 8 8 8 8 8 ...
 $ V17: Factor w/ 1 level "p": 1 1 1 1 1 1 1 1 1 1 ...
 $ V18: Factor w/ 4 levels "n","o","w","y": 3 3 3 3 3 3 3 3 3 3 ...
 $ V19: Factor w/ 3 levels "n","o","t": 2 2 2 2 2 2 2 2 2 2 ...
 $ V20: Factor w/ 5 levels "e","f","l","n",..: 5 5 5 5 1 5 5 5 5 5 ...
 $ V21: Factor w/ 9 levels "b","h","k","n",..: 3 4 4 3 4 3 3 4 3 3 ...
 $ V22: Factor w/ 6 levels "a","c","n","s",..: 4 3 3 4 1 3 3 4 5 4 ...
 $ V23: Factor w/ 7 levels "d","g","l","m",..: 6 2 4 6 2 2 4 4 2 4 ...
```

We then add names to the variables and check to see if they all took like they were supposed to using the head() function.



```
> head(mushroom)
  edibility cap_shape cap_surface cap_color bruises odor grill_attachment grill_spacing
1         p         x           s         n       t    p                f             c
2         e         x           s         y       t    a                f             c
3         e         b           s         w       t    l                f             c
4         p         x           y         w       t    p                f             c
5         e         x           s         g       f    n                f             w
6         e         x           y         y       t    a                f             c
  grill_size grill_color stalk_shape stalk_root stalk_surface_above_ring
1          n           k           e          e                        s
2          b           k           e          c                        s
3          b           n           e          c                        s
4          n           n           e          e                        s
5          b           k           t          e                        s
6          b           n           e          c                        s
  stalk_surface_below_ring stalk_color_above_ring stalk_color_below_ring veil_type veil_color
1                        s                      w                      w         p          w
2                        s                      w                      w         p          w
3                        s                      w                      w         p          w
4                        s                      w                      w         p          w
5                        s                      w                      w         p          w
6                        s                      w                      w         p          w
  ring_number ring_type spore_print_color population habitat
1           o         p                 k          s       u
2           o         p                 n          n       g
3           o         p                 n          n       m
4           o         p                 k          s       u
5           o         e                 n          a       g
6           o         p                 k          n       g
```

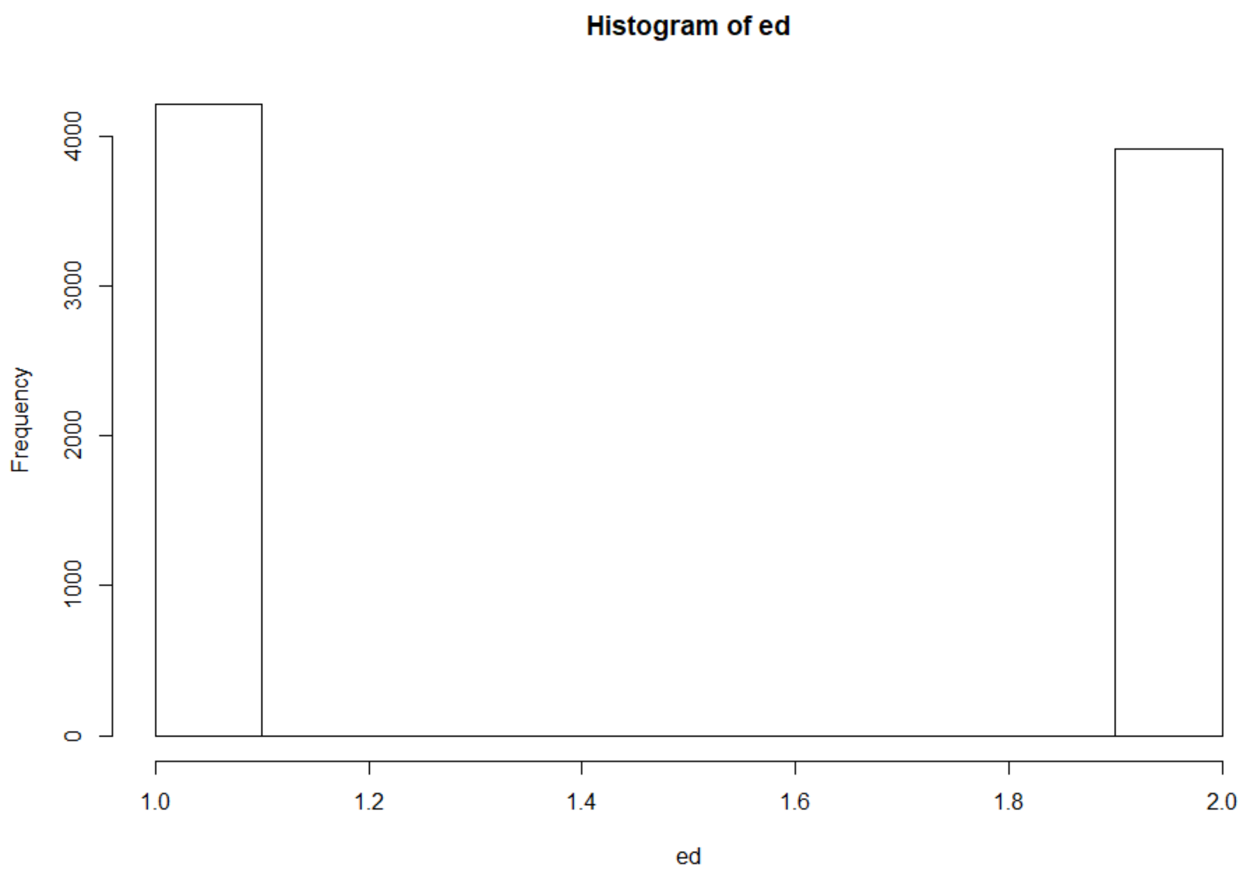
Once we look at the data, we see that, we will want to convert the input variables to a numeric variable to use with the SVM. We also look and find no missing values in this version of the data.



```
[1] 0
> View(mushroom)
```

We can see that there are 8124 observations and 23 variables. All of them are factors that will be converted to numeric except for the outcome variable in some cases of the analysis which will stay as a factor variable at times. We will remove the veil type variable because it only has one level.

We know that there are only 2 outcomes we want to consider in the set as a response variable, which are edible and poisonous. We will look at the distribution of the response variable to start understanding what the split is.



To use this variable in the analysis as the response variable we will want this variable as a leveled factor. Doing this will help us to better understand the ratings as they pertain to the variables. We can now look at the different classifications and the split in table form.

```
> table(ed)
ed
   1    2 
4208 3916 
```

Now we can look at what the observed edibility and poisonous percentages from the overall dataset to understand what we need to train the algorithms to get.

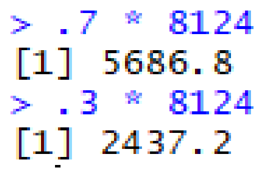
```
> 3916/8124
[1] 0.4820286
> 4208/8124
[1] 0.5179714
```

We can see here that overall in the mushroom data about 52% of the data is edible and 48% is poisonous.

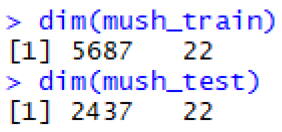
We want to make a classifier first with the SVM, we will want to experiment with different kernels and look the different accuracies. Now that we have the best types of data for analysis, we will have to split the data into a test and a training set to run the analysis with SVM. We now want to move on to create our decision tree. We will use the SVM kernel of linear first and them look at the radial kernel to see the difference and to compare the accuracy.

We will classify the mushrooms into 2 levels based on the different properties/variables in the rest of the dataset. We will do all of this using the "e1071" package first. This will help us to randomize the samples and to use the train and test sets we want to split the data into to complete our analysis. We will split our data in training set of 70% and a test set of 30% to try to give the most training to the models before use. This will help us to get the best results we can out of our algorithms.

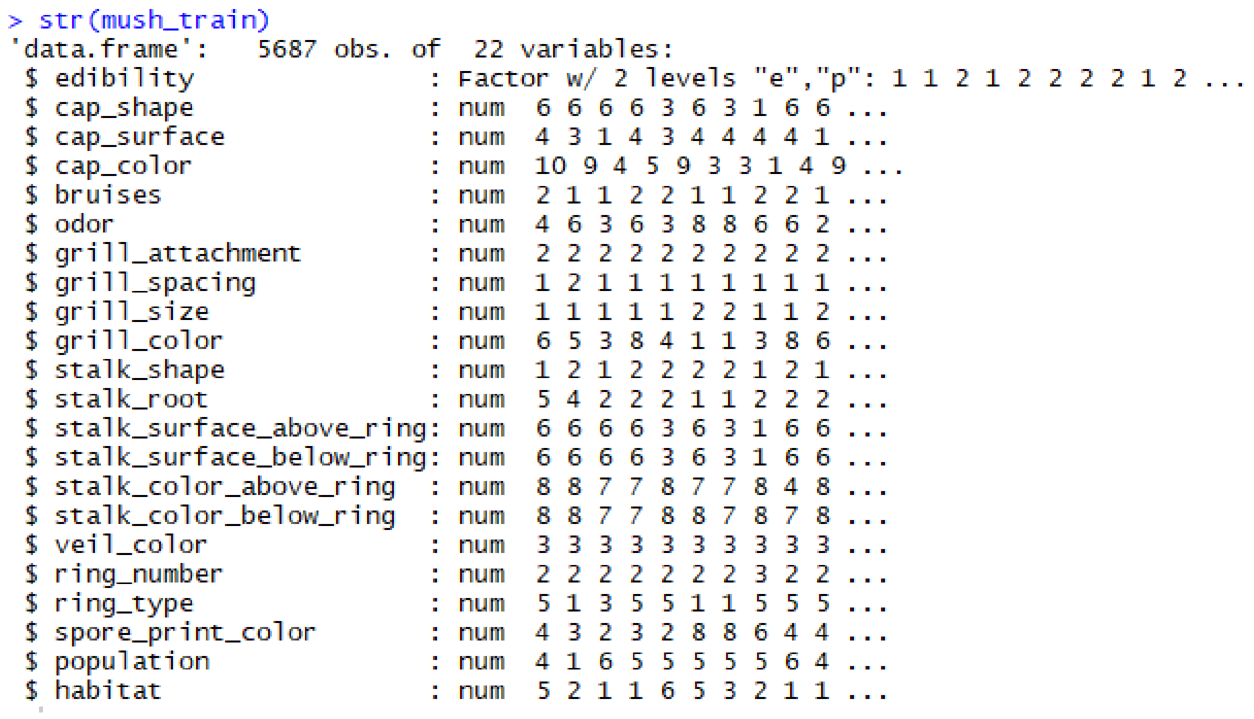
First, we figure out what an 70/30 split of our data will need to look like. Then we will create the data sets based on what the splits should be from the original.



```
> .7 * 8124
[1] 5686.8
> .3 * 8124
[1] 2437.2
```

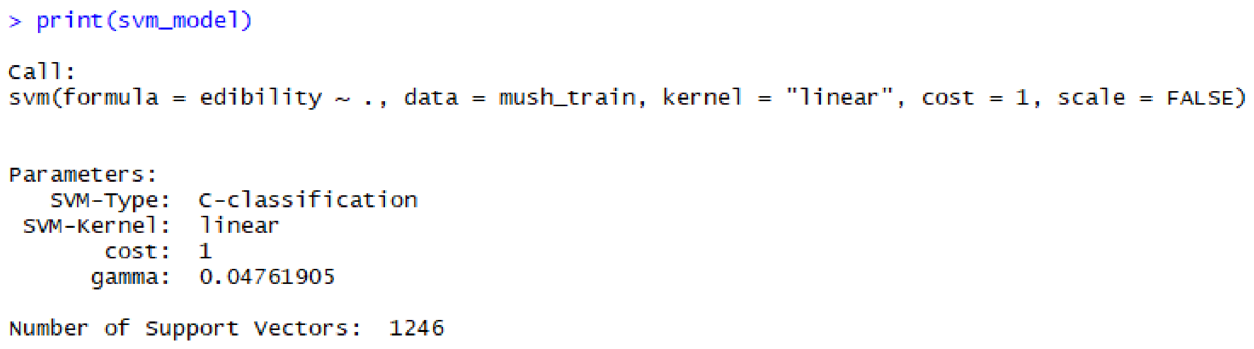


```
> dim(mush_train)
[1] 5687   22
> dim(mush_test)
[1] 2437   22
```



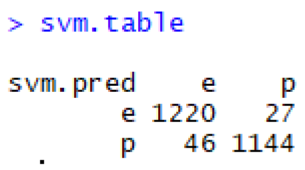
```
> str(mush_train)
'data.frame':	5687 obs. of  22 variables:
 $ edibility               : Factor w/ 2 levels "e","p": 1 1 2 1 2 2 2 2 1 2 ...
 $ cap_shape               : num  6 6 6 6 3 6 3 1 6 6 ...
 $ cap_surface             : num  4 3 1 4 3 4 4 4 4 1 ...
 $ cap_color               : num  10 9 4 5 9 3 3 1 4 9 ...
 $ bruises                 : num  2 1 1 2 2 1 1 2 2 1 ...
 $ odor                    : num  4 6 3 6 3 8 8 6 6 2 ...
 $ grill_attachment        : num  2 2 2 2 2 2 2 2 2 2 ...
 $ grill_spacing           : num  1 2 1 1 1 1 1 1 1 1 ...
 $ grill_size              : num  1 1 1 1 1 2 2 1 1 2 ...
 $ grill_color             : num  6 5 3 8 4 1 1 3 8 6 ...
 $ stalk_shape             : num  1 2 1 2 2 2 2 1 2 1 ...
 $ stalk_root              : num  5 4 2 2 2 1 1 2 2 2 ...
 $ stalk_surface_above_ring: num  6 6 6 6 3 6 3 1 6 6 ...
 $ stalk_surface_below_ring: num  6 6 6 6 3 6 3 1 6 6 ...
 $ stalk_color_above_ring  : num  8 8 7 7 8 7 7 8 4 8 ...
 $ stalk_color_below_ring  : num  8 8 7 7 8 8 7 8 7 8 ...
 $ veil_color              : num  3 3 3 3 3 3 3 3 3 3 ...
 $ ring_number             : num  2 2 2 2 2 2 2 3 2 2 ...
 $ ring_type               : num  5 1 3 5 5 1 1 5 5 5 ...
 $ spore_print_color       : num  4 3 2 3 2 8 8 6 4 4 ...
 $ population              : num  4 1 6 5 5 5 5 5 6 4 ...
 $ habitat                 : num  5 2 1 1 6 5 3 2 1 1 ...
```

We can see all went well and we now have the two randomized samples of the data that we specified we wanted earlier. We also look at the structure of the train data before we move to use it. We can now create our SVM model using the svm() function from the e1071 package. Once we build our model we will want to look at it. We could use another plotting package, however the best one to use for an SVM model is the "gamlss" or "gamlss.add" package or even just the plot() or print() function, which we will use to visualize our results next.



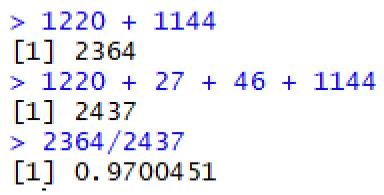
```
> print(svm_model)

Call:
svm(formula = edibility ~ ., data = mush_train, kernel = "linear", cost = 1, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.04761905

Number of Support Vectors:  1246
```

The SVM shows us that we have about over 1200 vectors identified while using the linear kernel. We will also look at the accuracy of the different kernels next.



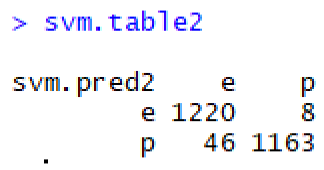
```
> svm.table

svm.pred    e    p
       e 1220   27
       p   46 1144
```

From the table we can then calculate the overall accuracy of the model.



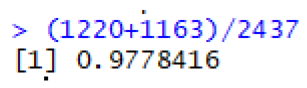
```
> 1220 + 1144
[1] 2364
> 1220 + 27 + 46 + 1144
[1] 2437
> 2364/2437
[1] 0.9700451
```

We can see above that the accuracy of the model was at 97%. So now we want to try to experiment a bit with the results and the model. We know that if we increase the cost of the model it will be more discriminate and therefore improve accuracy. So, lets try an increase to 100 on the cost and see if that helps the model. Once we do that we will look at the table and calculate the accuracy to see if it improved.



```
> svm.table2

svm.pred2    e    p
        e 1220    8
        p   46 1163
```

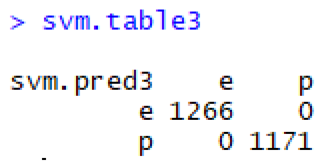
The new table is above and now we will go ahead to calculate the accuracy.



```
> (1220+1163)/2437
[1] 0.9778416
```

We can see here that changing the cost does in fact improve the accuracy of the model with the model's overall accuracy increase going to approximately 98%.

Now we want to see if it helps to change the kernel. We know that the use of a nonlinear kernel helps with accuracy when the data is not necessarily linear. So, we can try to use a different kernel and we will impose a nonlinear one by using the radial kernel next.

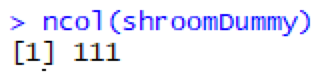


```
> svm.table3

svm.pred3    e    p
        e 1266    0
        p    0 1171
```

We can already see that changing the kernel was the key to this algorithm. We see that the radial SVM kernel actually predicted the edibility and the poisonous labels of the mushroom dataset with 100% accuracy. Since we can't improve on 100% accuracy we will now move to building and assessing the neural network to compare to this.

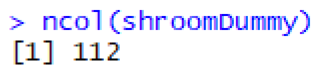
Neural Network

Now we want to ensure we reload the data and take out the veil variable due to it only having one level again. We will also do two versions where we impute the data for one model in the stalk_root variable due to it having too many missing values, while we will also take it completely out for another one since that is what we would do in real life. This set will be the main set we work while the imputed set will have a 2 behind it for clarity. We will then compare those methods as well as comparing the Neural net to the SVM. However, with both sets it is much easier to create dummy variables for the factors in order to much more easily create the test and train sets for the data. And we will only look at the set with the excluded stock_root variable here for the rest of this analysis.



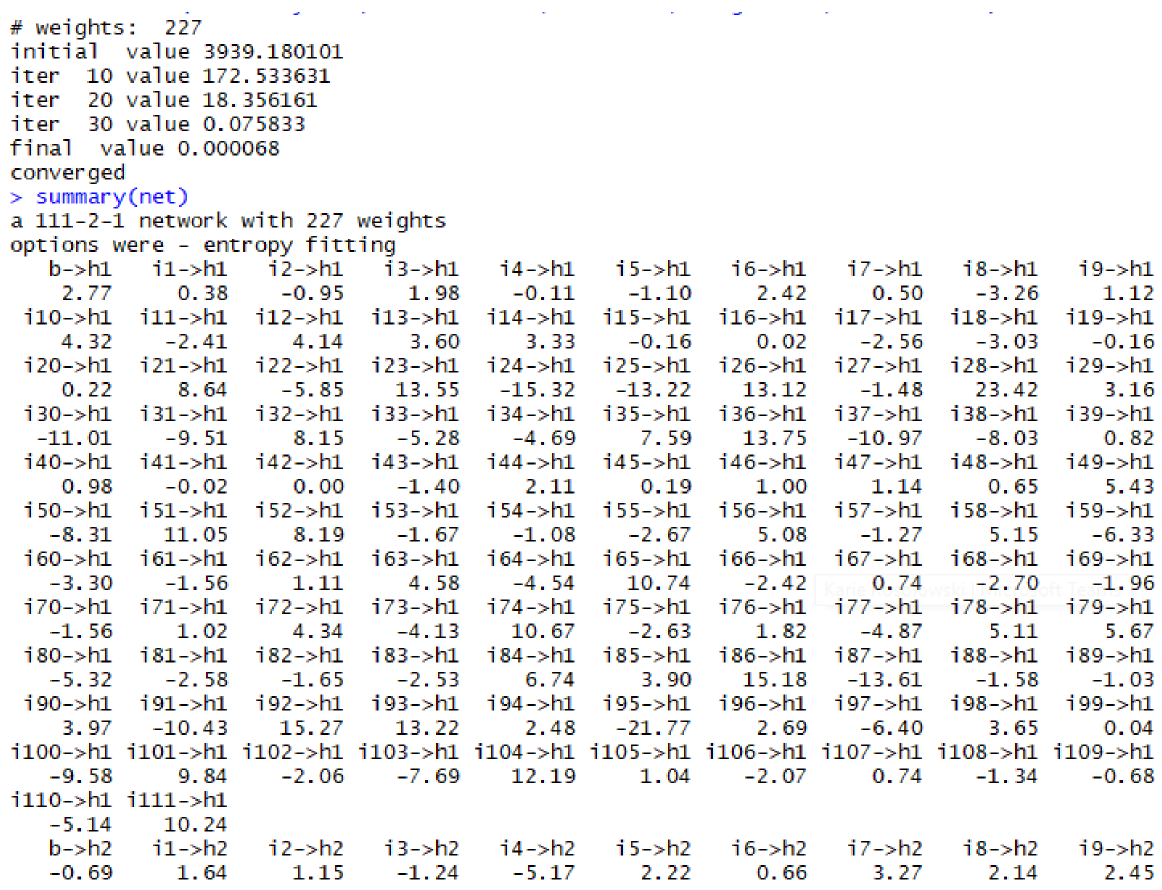
```
> ncol(shroomDummy)
[1] 111
```

We then add back in the edibility variable because we took it out for the dummy conversion.



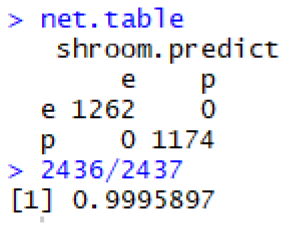
```
> ncol(shroomDummy)
[1] 112
```

We then build our neural net model to use in the analysis. We will then look at the summary of our neural net to make sure it took properly. We can see the weights and cut offs from the iterations that we set as well as the full fitting.

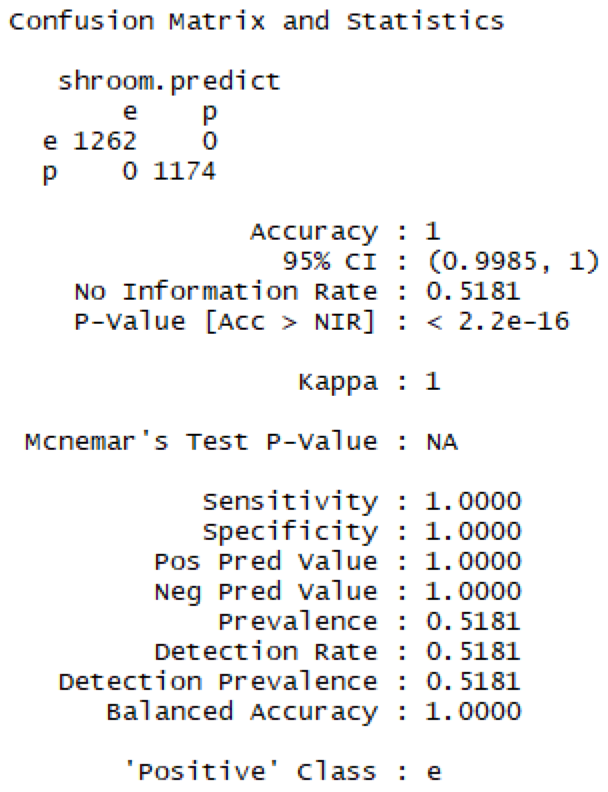


```
# weights:  227
initial  value 3939.180101
iter  10 value 172.533631
iter  20 value 18.356161
iter  30 value 0.075833
final  value 0.000068
converged
> summary(net)
a 111-2-1 network with 227 weights
options were - entropy fitting
   b->h1   i1->h1   i2->h1   i3->h1   i4->h1   i5->h1   i6->h1   i7->h1   i8->h1   i9->h1
    2.77     0.38    -0.95     1.98    -0.11    -1.10     2.42     0.50    -3.26     1.12
  i10->h1  i11->h1  i12->h1  i13->h1  i14->h1  i15->h1  i16->h1  i17->h1  i18->h1  i19->h1
    4.32    -2.41     4.14     3.60     3.33    -0.16     0.02    -2.56    -3.03    -0.16
  i20->h1  i21->h1  i22->h1  i23->h1  i24->h1  i25->h1  i26->h1  i27->h1  i28->h1  i29->h1
    0.22     8.64    -5.85    13.55   -15.32   -13.22    13.12    -1.48    23.42     3.16
  i30->h1  i31->h1  i32->h1  i33->h1  i34->h1  i35->h1  i36->h1  i37->h1  i38->h1  i39->h1
  -11.01    -9.51     8.15    -5.28    -4.69     7.59    13.75   -10.97    -8.03     0.82
  i40->h1  i41->h1  i42->h1  i43->h1  i44->h1  i45->h1  i46->h1  i47->h1  i48->h1  i49->h1
    0.98    -0.02     0.00    -1.40     2.11     0.19     1.00     1.14     0.65     5.43
  i50->h1  i51->h1  i52->h1  i53->h1  i54->h1  i55->h1  i56->h1  i57->h1  i58->h1  i59->h1
   -8.31    11.05     8.19    -1.67    -1.08    -2.67     5.08    -1.27     5.15    -6.33
  i60->h1  i61->h1  i62->h1  i63->h1  i64->h1  i65->h1  i66->h1  i67->h1  i68->h1  i69->h1
   -3.30    -1.56     1.11     4.58    -4.54    10.74    -2.42     0.74    -2.70    -1.96
  i70->h1  i71->h1  i72->h1  i73->h1  i74->h1  i75->h1  i76->h1  i77->h1  i78->h1  i79->h1
   -1.56     1.02     4.34    -4.13    10.67    -2.63     1.82    -4.87     5.11     5.67
  i80->h1  i81->h1  i82->h1  i83->h1  i84->h1  i85->h1  i86->h1  i87->h1  i88->h1  i89->h1
   -5.32    -2.58    -1.65    -2.53     6.74     3.90    15.18   -13.61    -1.58    -1.03
  i90->h1  i91->h1  i92->h1  i93->h1  i94->h1  i95->h1  i96->h1  i97->h1  i98->h1  i99->h1
    3.97   -10.43    15.27    13.22     2.48   -21.77     2.69    -6.40     3.65     0.04
 i100->h1 i101->h1 i102->h1 i103->h1 i104->h1 i105->h1 i106->h1 i107->h1 i108->h1 i109->h1
   -9.58     9.84    -2.06    -7.69    12.19     1.04    -2.07     0.74    -1.34    -0.68
 i110->h1 i111->h1
   -5.14    10.24
   b->h2   i1->h2   i2->h2   i3->h2   i4->h2   i5->h2   i6->h2   i7->h2   i8->h2   i9->h2
   -0.69     1.64     1.15    -1.24    -5.17     2.22     0.66     3.27     2.14     2.45
```

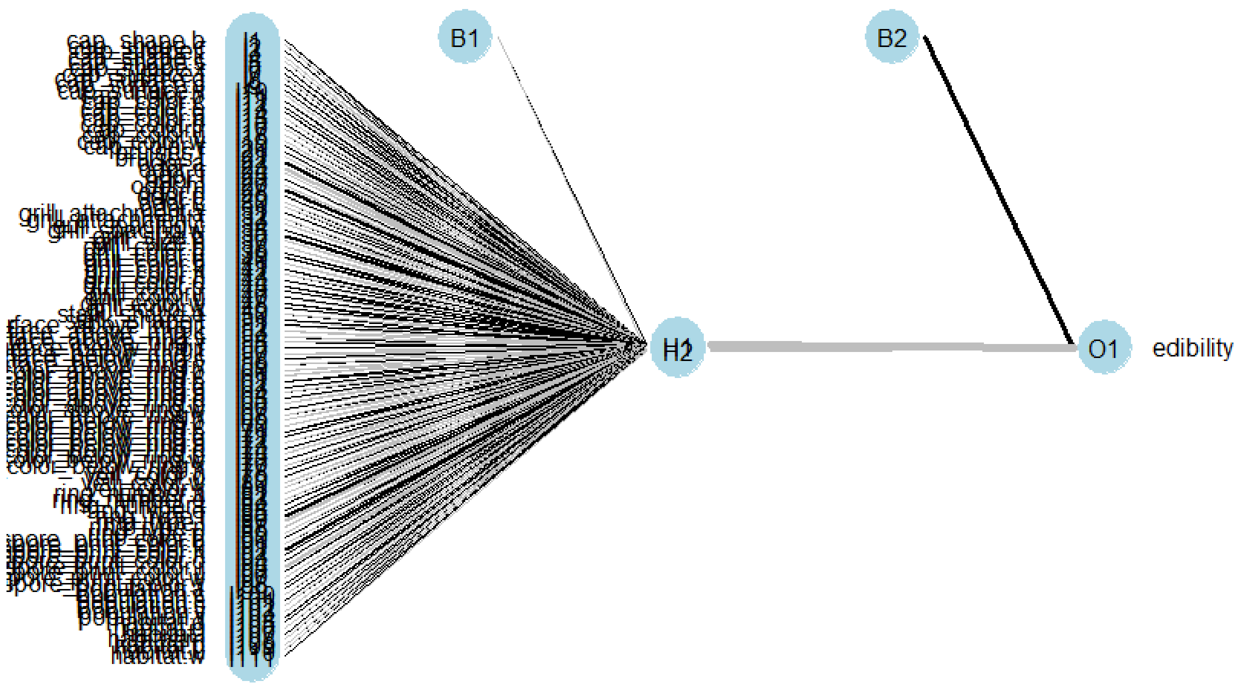
We then look at the table and do the math and find that the matrix actually misclassifies one of the values, but it is rounded into a 100% approximation. The actual fitting is 99.85% for the ANN without the missing values and 98.23% for the model with imutation.



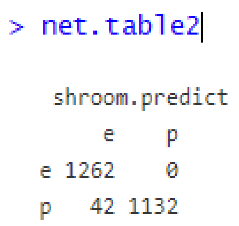
```
> net.table
   shroom.predict
       e    p
  e 1262    0
  p    0 1174
> 2436/2437
[1] 0.9995897
```



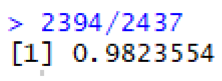
```
Confusion Matrix and Statistics

   shroom.predict
       e    p
  e 1262    0
  p    0 1174

               Accuracy : 1
                 95% CI : (0.9985, 1)
    No Information Rate : 0.5181
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.5181
         Detection Rate : 0.5181
   Detection Prevalence : 0.5181
      Balanced Accuracy : 1.0000

       'Positive' Class : e
```



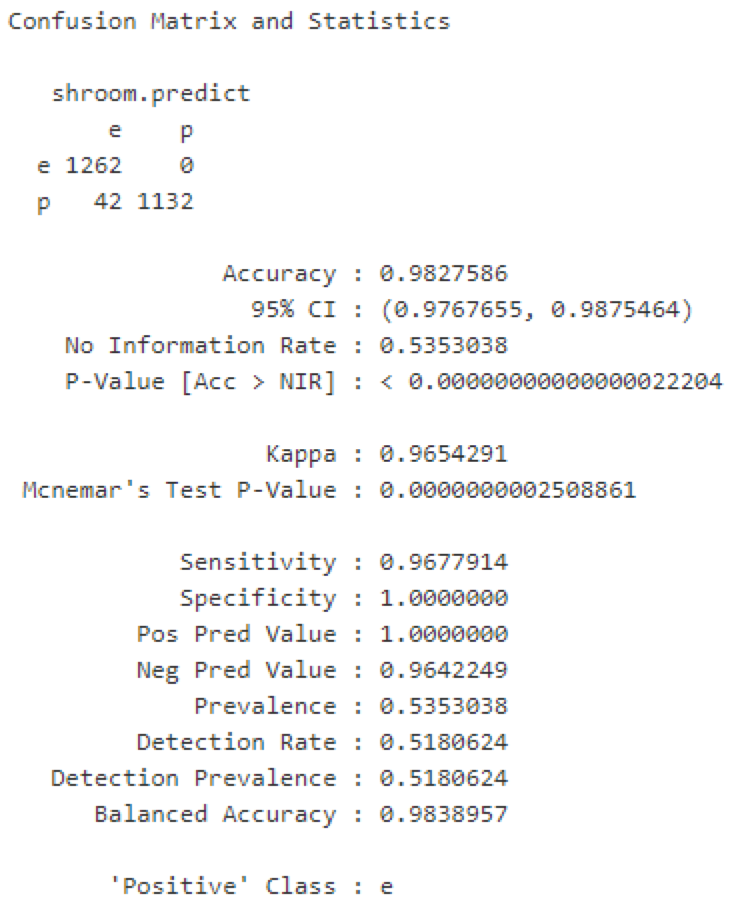
We can see the plot above of the neural net as well to help with our visualization. We also see from our imputed model we did parallel to this one that it was less accurate as well



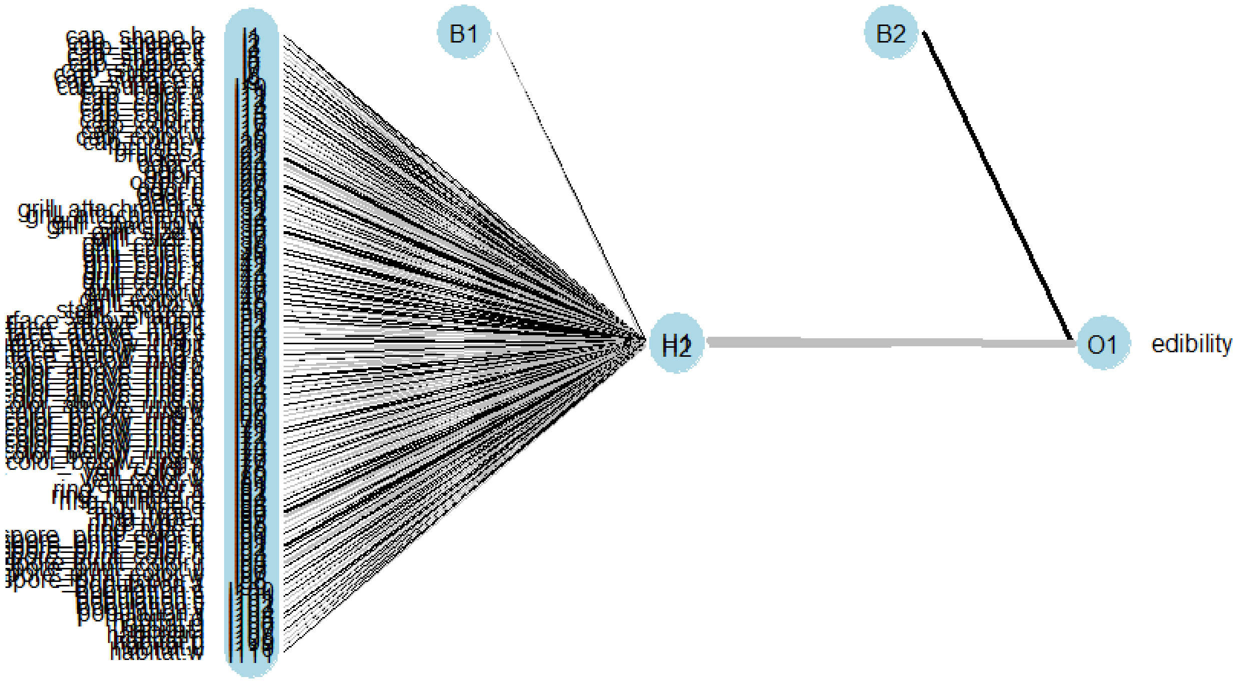
```
> net.table2

   shroom.predict
       e    p
  e 1262    0
  p   42 1132
```



```
> 2394/2437
[1] 0.9823554
```



```
Confusion Matrix and Statistics

   shroom.predict
       e    p
  e 1262    0
  p   42 1132

               Accuracy : 0.9827586
                 95% CI : (0.9767655, 0.9875464)
    No Information Rate : 0.5353038
    P-Value [Acc > NIR] : < 0.00000000000000022204

                  Kappa : 0.9654291
 Mcnemar's Test P-Value : 0.0000000002508861

            Sensitivity : 0.9677914
            Specificity : 1.0000000
         Pos Pred Value : 1.0000000
         Neg Pred Value : 0.9642249
             Prevalence : 0.5353038
         Detection Rate : 0.5180624
   Detection Prevalence : 0.5180624
      Balanced Accuracy : 0.9838957

       'Positive' Class : e
```



This lets us know if the ANN is actually not as accurate as the SVM model was.

So, overall, I would personally go with the different kernels of the SVM because this would allow me to adjust for the linearity of the data I am considering. I think that I like the reliability and the changeability of the SVM as opposed to the ANN. We see that in our first analysis we had multiple ways to adjust our fitting, kernels, and the ability to improve accuracy over different models. I have concerns that the ANN would not allow me to do this if I were trying to get more accurate in an analysis. We also see that the over fitting can happen. It is seen that yes it is accurate, however, there is a chance to be 100% accurate in a model from the SVM algorithms with the kernel tricks as well. It just seems this would be a better method to use when linearity and the data is in question.

Conclusion

Today we looked at the predictions that SVMs and ANNs can provide when using them in machine learning environments. Overall, we can see that it is probably more accurate and efficient to use an SVM depending on the data and linearity of the set.

References

Chiu, Y. (2015). Machine Learning with R cookbook. Packt Publishing (Chapter 6). ONLINE Book available via http://lumen.regis.edu/record=b1659405~S3.

Lantz, Brett "Machine learning with R: discover how to build machine learning algorithms, prepare data, and dig deep into data prediction techniques with R" http://lumen.regis.edu/record=b1734246~S3

Ren, J. (2012). ANN vs. SVM: Which one performs better in classification of MCCs in mammogram imaging. *Knowledge-Based Systems*, *26*, 144-153.