Predicting Whether a Given Mushroom Is Edible

Ian Boulis and Caleb Hiltunen
MA4790
December 2020

Abstract:

Mushroom hunting and identification has been an important part of many cultures since the dawn of man. Cultural impacts ranging from essential spiritual journey to cooking ingredients and even into pop culture. Knowing which mushrooms are dangerous to you and which pose no threat is what gives mushrooms much of their cultural significance. Data on given mushroom characteristics have been collected and compiled via the *The Audubon Society Field Guide to North American Mushrooms* and our goal is to determine which physical characteristics are indicators of a mushroom being poisonous.The distributions and frequencies of each characteristic will be explored along with which characteristics we believe do not aid in classification. Both linear and non-linear models will be constructed using the training data, our top performing models will be used to predict on the testing set, and the final overall best performing model will be selected.

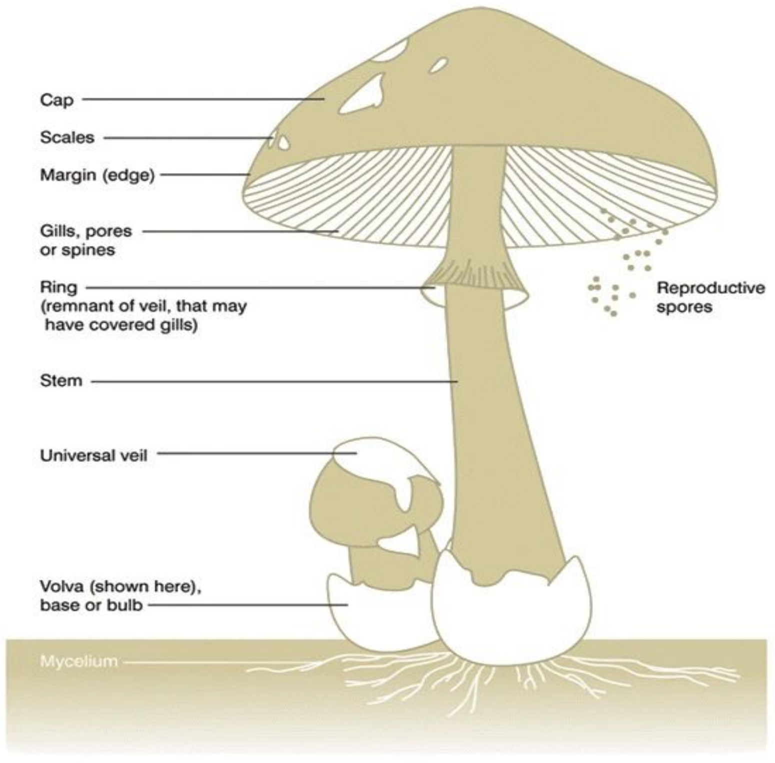


Table of Contents

Abstract…………………………………………………………………..…..1

1 Background………………………………………………………….......3

a. Variable Introduction and Definitions…………………………….....3

2 Preprocessing of the predictors……………………………………......7

a. Zero/Near-Zero Variance..10

b. Correlations...10

c. Center and Scale...12

3 Splitting of the Data...12

4 Model Fitting..13

a. Linear Models...13

b. Non-Linear Models..15

c. Optimization...16

5

Summary...17

6 Appendix 1: Linear Classification Models Outputs and Plots..........19

7 Appendix 2: Nonlinear Classification Models Outputs and Plots.....23

8 R Code..30

1 Background

a) Variable Introduction and Definitions:

Data has been collected from *The Audubon Society Field Guide to North American Mushrooms* and accessed via Kaggle. Our response variable is “class” categorized by whether a given mushroom is poisonous or not. Below is a list of our variables, different categories for each variable and a description for each variable. There were 22 predicting variables, all categorical, and 8124 samples.

Predictor Name	**Description**
● Cap shape	The shape of the top of the mushroom
○ b = bell	
○ c = conical	
○ x = convex	
○ f = flat	
○ k = knobbed	
○ s = sunken	
● Cap surface	The surface of the top of the mushroom
○ f = fibrous	
○ g = grooves	
○ y = scaly	
○ s = smooth	
● Cap color	The color of the top of the mushroom
○ n = brown	
○ b = buff	
○ c = cinnamon	
○ g = gray	
○ r = green	
○ p = pink	
○ u = purple	
○ e = red	
○ w = white	
○ y = yellow	
● Bruises	The presence of any discoloration on the mushroom
○ t = bruises	
○ f = no bruises	
● Odor	The odor the mushroom gives off

    - a = almond
    - l = anise
    - c = creosote
    - y = fishy
    - f = foul
    - m = musty
    - n = none
    - p = pungent
    - s = spicy
- Gill attachment — The attachment of the underside of the cap
    - a = attached
    - d = descending
    - f = free
    - n = notched
- Gill spacing — The spacing between gills under the cap
    - c = closed
    - w = crowded
    - d = distant
- Gill size — The shape of the gills under the cap
    - b = broad
    - n = narrow
- Gill color — The color of the gills under the cap
    - k = black
    - n = brown
    - b = buff
    - h = chocolate
    - g = gray
    - r = green
    - o = orange
    - p = pink
    - u = purple
    - e = red
    - w = white
    - y = yellow
- Stalk shape — The size of the vertical portion of the mushroom
    - e = enlarging
    - t = tapering
    - Stalk root
    - b = bulbous
    - c = club
    - u = cup
    - e = equal
    - z = rhizomorphs
    - r = rooted

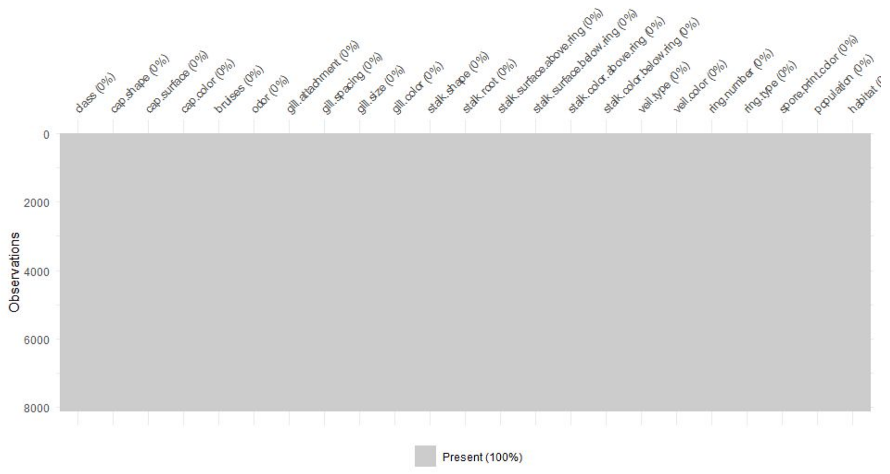
    - ? = missing
- Stalk surface above ring
  The surface between the ring and the cap
    - f = fibrous
    - y = scaly
    - k = silky
    - s = smooth
- Stalk surface below ring
  The surface of the mushroom bellow the ring
    - f = fibrous
    - y = scaly
    - k = silky
    - s = smooth
- Stalk color above ring
  The color between the ring and the cap
    - n = brown
    - b = buff
    - c = cinnamon
    - g = gray
    - o = orange
    - p = pink
    - e = red
    - w = white
    - y = yellow
- Stalk color below ring
  The color below the ring
    - n = brown
    - b = buff
    - c = cinnamon
    - g = gray
    - o = orange
    - p = pink
    - e = red
    - w = white
    - y = yellow
- Veil type
  Presence of a veil below or above the cap (mostly found in premature mushrooms)
    - p = partial
    - u = universal
- Veil color
  Color of veil if present
    - n = brown
    - o = orange
    - w = white
    - y = yellow
- Ring number
  Number of rings or veils
    - n = none
    - o = one
    - t = two
- Ring type
  Shape of ring or veil

    - c = cobwebby
    - e = evanescent
    - f = flaring
    - l = large
    - n = none
    - p = pendant
    - s = sheathing
    - z = zone
- Spore print color — Color produced when pressing spores to white paper
    - k = black
    - n = brown
    - b = buff
    - h = chocolate
    - r = green
    - o = orange
    - u = purple
    - w = white
    - y = yellow
- Population — Amount of mushrooms seen in a given area
    - a = abundant
    - c = clustered
    - n = numerous
    - s = scattered
    - v = several
    - y = solitary
- Habitat — Environment where mushroom was observed at
    - g = grasses
    - l = leaves
    - m = meadows
    - p = paths
    - u = urban
    - w = waste
    - d = woods

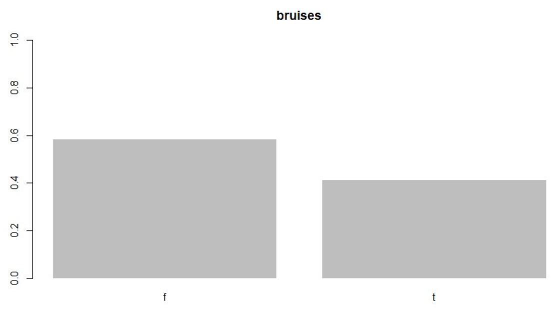
With these predictors the relationship between characteristics of each predictors will be analyzed to determine the most defining characteristic of an edible mushroom. The data will be preprocessed including missing variables, zero or near zero variances in the set, and collinearity between predictors. Then, both linear and non-linear models will be applied to predict defining characteristics of an edible mushroom.

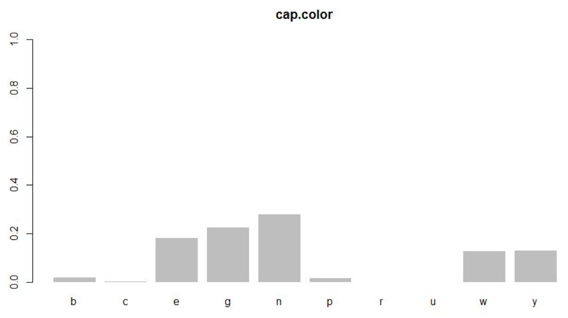
2 Preprocessing of the Predictors

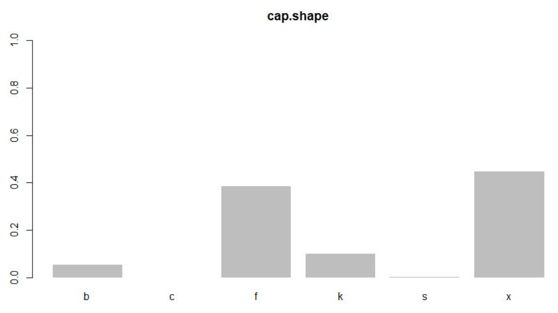
The first step in preprocessing our data is to find any missing variables in our set and impute new variables to obtain a complete data set.

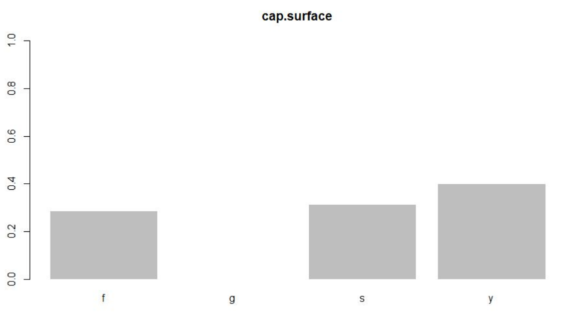


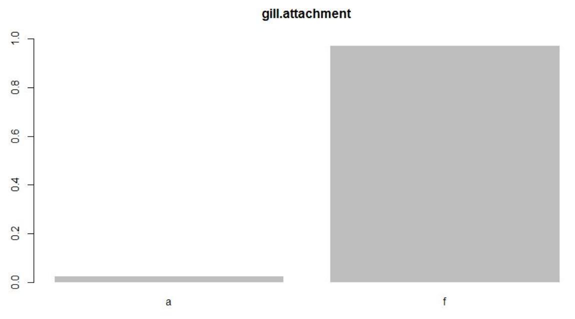
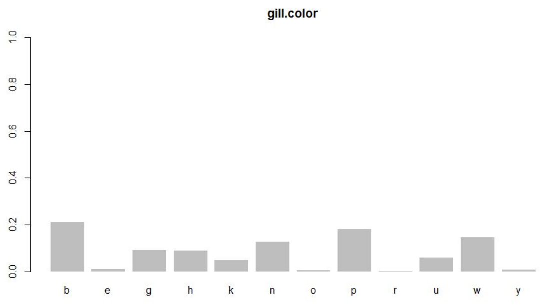
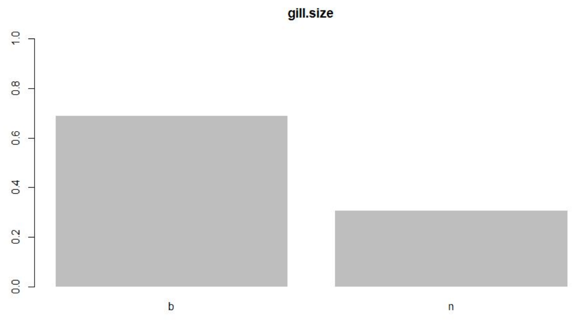
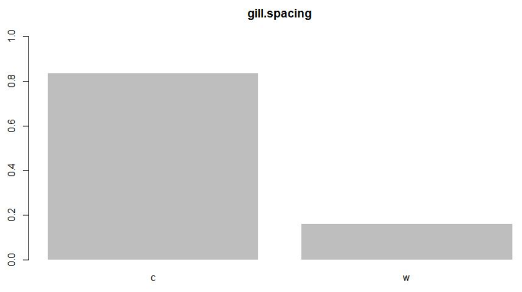
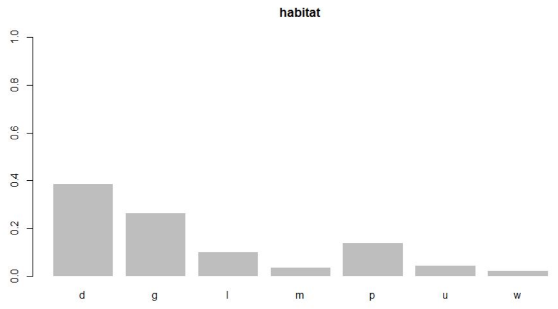
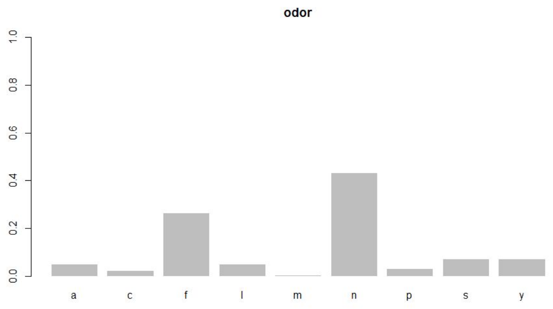
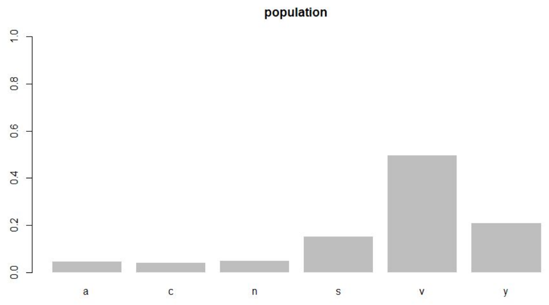
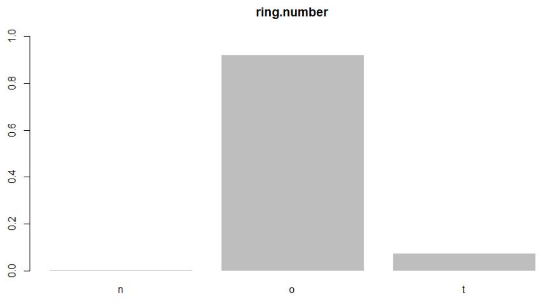
Our data set was complete in that there were no predictors with missing variables. Next we created dummy variables for every predictor. We created a total of 95 dummy variables. Then a check for unbalanced predictors was done.

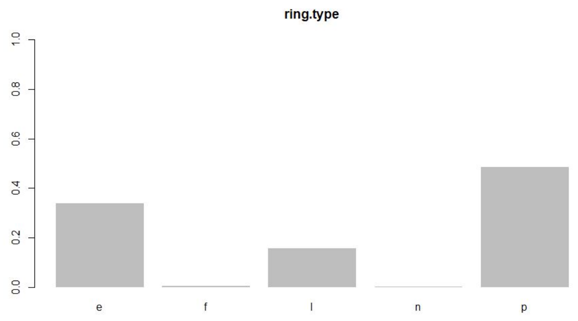


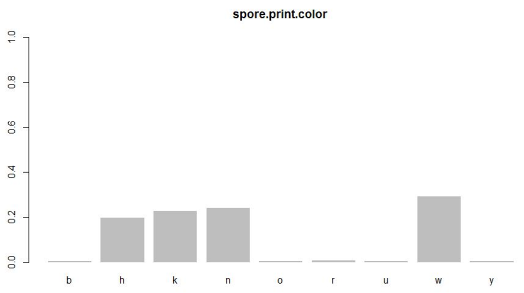


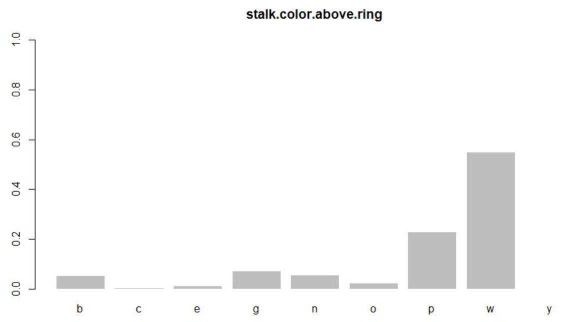


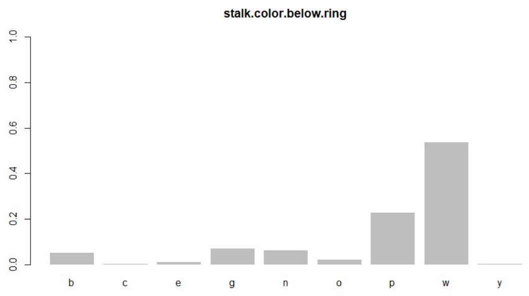


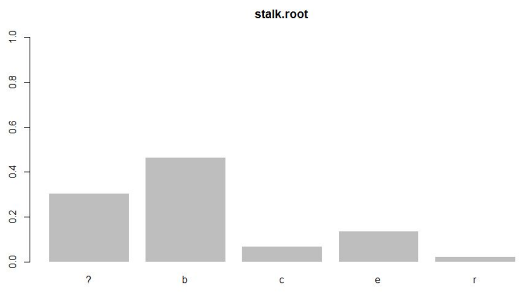

gill.attachment
1.0
0.8
0.6
0.4
0.2
0.0
a
f

gill.color
1.0
0.8
0.6
0.4
0.2
0.0
b
e
g
h
k
n
o
p
r
u
w
y

gill.size
1.0
0.8
0.6
0.4
0.2
0.0
b
n

gill.spacing
1.0
0.8
0.6
0.4
0.2
0.0
c
w

habitat
1.0
0.8
0.6
0.4
0.2
0.0
d
g
l
m
p
u
w

odor
1.0
0.8
0.6
0.4
0.2
0.0
a
c
f
l
m
n
p
s
y

population
1.0
0.8
0.6
0.4
0.2
0.0
a
c
n
s
v
y

ring.number
1.0
0.8
0.6
0.4
0.2
0.0
n
o
t

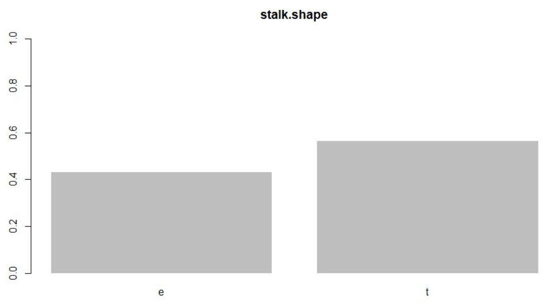

ring.type
1.0
0.8
0.6
0.4
0.2
0.0
e
f
l
n
p

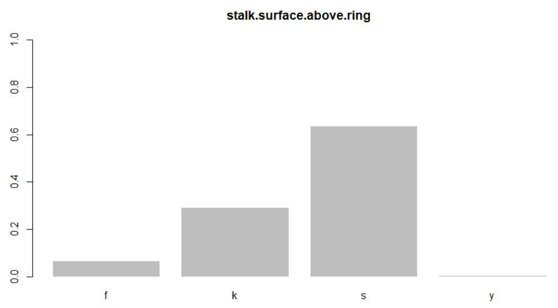

spore.print.color
1.0
0.8
0.6
0.4
0.2
0.0
b
h
k
n
o
r
u
w
y

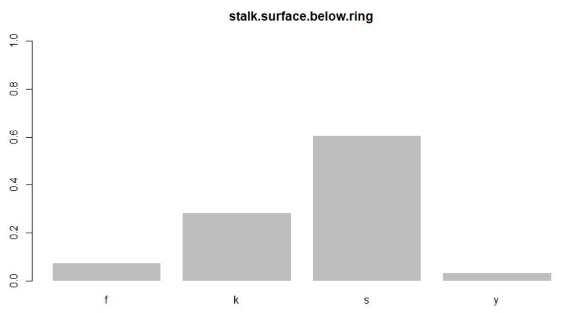

stalk.color.above.ring
1.0
0.8
0.6
0.4
0.2
0.0
b
c
e
g
n
o
p
w
y

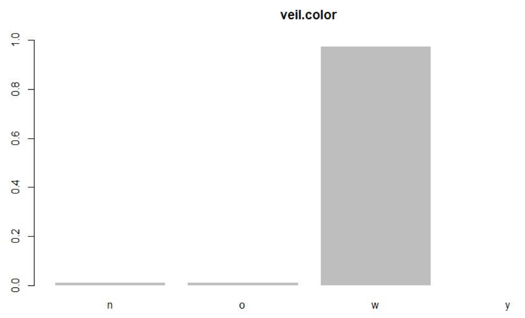

stalk.color.below.ring
1.0
0.8
0.6
0.4
0.2
0.0
b
c
e
g
n
o
p
w
y


stalk.root
1.0
0.8
0.6
0.4
0.2
0.0
?
b
c
e
r


stalk.shape
1.0
0.8
0.6
0.4
0.2
0.0
e
t







As you can see, quite a few of our predictors were unevenly distributed. This is partially because when collecting samples the field workers determined characteristics prior to actually going out into the field and gathering data. So there were some characteristics that they expected to observe, however, found none for a handful of categories. This will be fixed after checking for zero/near-zero variances, followed up with center and scaling.

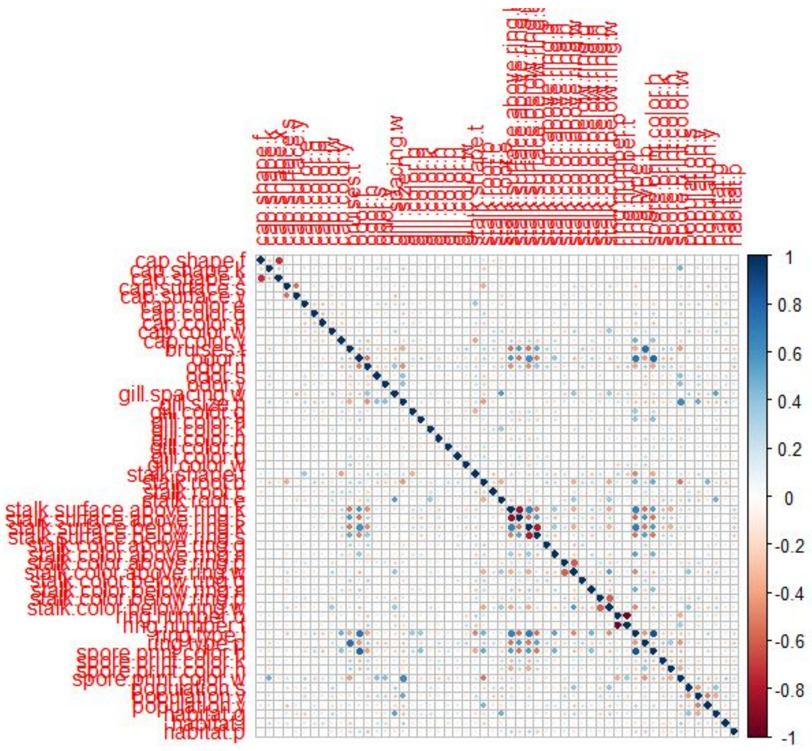
a. Zero/Near-Zero Variances

Using a cutoff of 95/5, the nearZeroVar function suggested that we remove nearly half of our predictors. R suggested that we remove so many because of the reasons mentioned earlier, that there were zero observations of certain characteristics. After removing the 41 near zero variance variables our set is left with 54 variables.

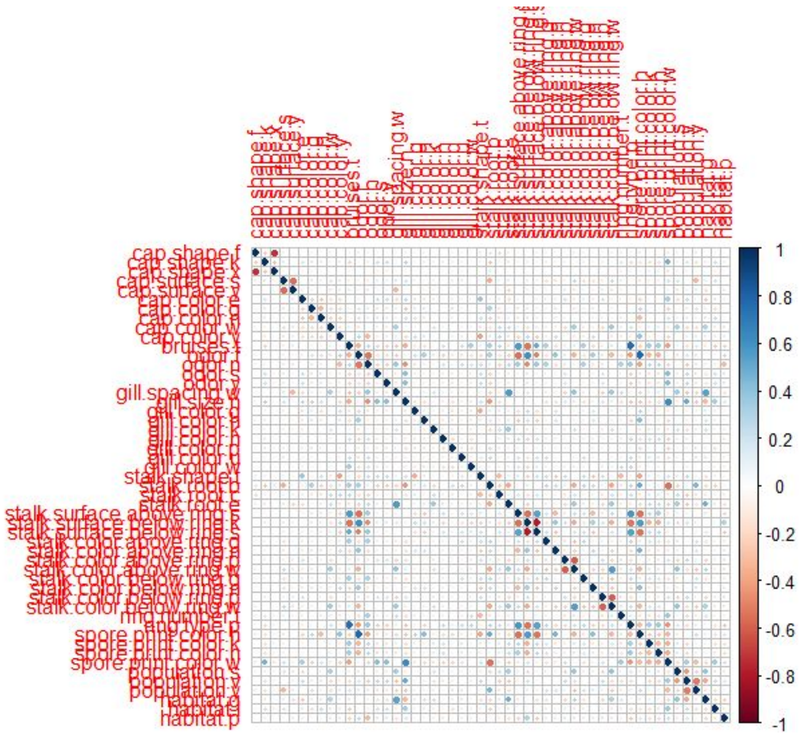
b. Collinearity

Nearly all of our data has a very weak correlation to each other. Using a cutoff value of 80% only three predictors were found to be highly correlated with one another.

Before highCorr() with a cutoff of 80%:



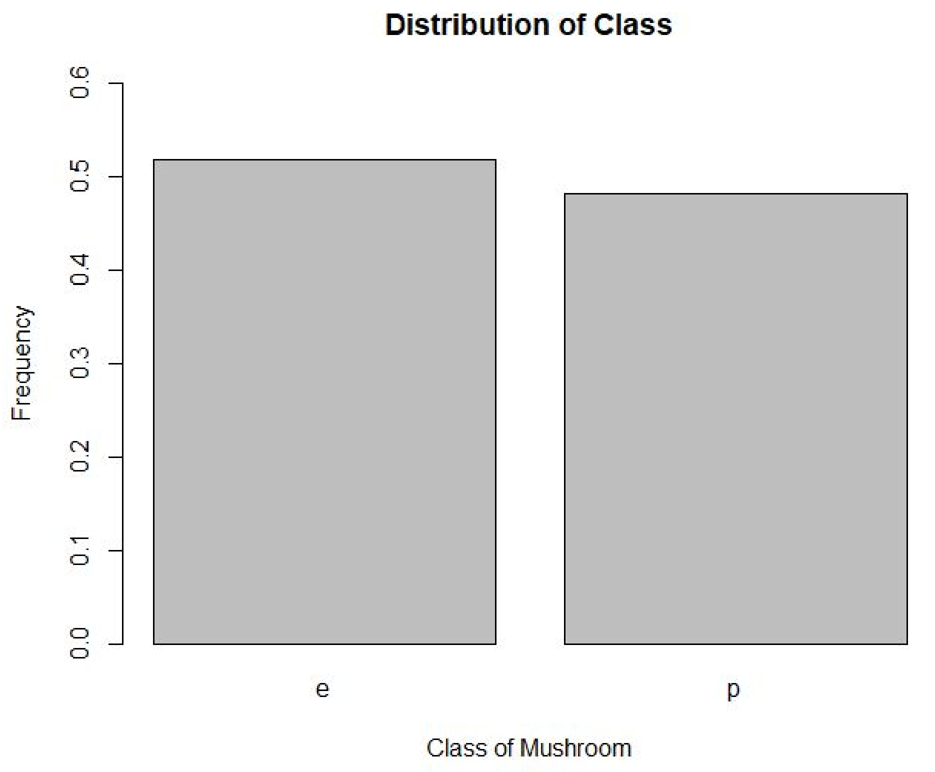
After highCorr() with a cutoff of 80%:



After this cutoff was implemented, we were left with 51 predictors.

c. Center and Scaling
Center and scaling was performed due to multiple models requiring center and scaling done. Following this process of centering and scaling the predictors, the dataset preprocessing is completed. We are left with 8124 observations and 51 dummy variables for predictors.

3 Splitting of the Data
To split the data, we decided to go with an 80/20 split training/testing using the stratified sampling method. The following figure is a distribution of the response variable, the variable we performed the split on.



While this class itself was not very unbalanced, performing a stratified split using this class gave the most equal representation of all other variables present within the dataset. Normal random would occasionally create huge disparities between the ‘1’ and ‘0’ values for other predictors, and even a stratified split performed predictors would cause the same issue. Stratified on the response variable ensured representation for each predictor in the training and testing sets. The split resulted in 6500 samples in the training set and 1624 in the testing set.

To help ensure overfitting would not be an issue within our model building, we also went with 10-fold-cross-validation. Our dataset is large enough where bootstrapping and

LOOCV would be too computationally intensive and unnecessary, so 10-fold-cross-validation felt the most appropriate.

4 Model Fitting

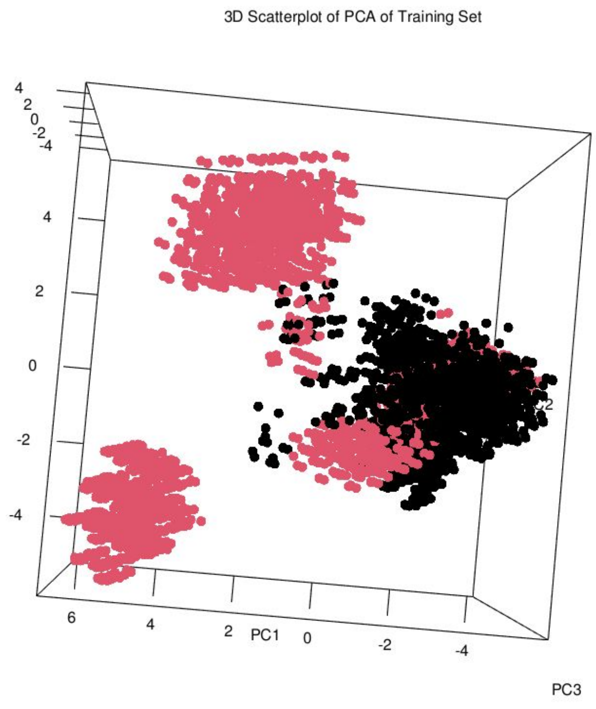
We chose to optimize the models for specificity, as the false positive rate is calculated from 1 - (specificity). It is important to have a good true positive rate in this study as a false positive in this context means that the mushroom is poisonous while being described as edible by the model, potentially resulting in a fatal consumption by the user.

A. Linear Models

Starting with the linear models:

Model	Tuning Parameters	Specificity
Logistic Regression	N/A	1.000
Linear Discriminant Analysis	N/A	0.9818
Partial Least Squares	Components = 29	0.9818
General Linear Model	Alpha = 0.4, Lambda = 0.01	0.9780
Shrunken Centroid	Threshold = 18.7	0.9697

Our linear models performed very well on this data set with most of them giving a near perfect value for specificity. The Logistic Regression model did output a specificity value of 1, however, it is a little misleading. Whilst running the Logistic Regression model R output an error message stating “fitted probabilities numerically 0 or 1 occurred”, and after researching what could cause that error message to appear we discovered that the predictor classes are perfectly separable. Looking at a 3D scatterplot of our data:



There is a very clear distinction between the edible and poisonous mushrooms which caused some errors when building said model. For this reason the Logistic Regression model will not be considered. For more insights into the linear classification models, their outputs, and their tuning plots, please refer to Appendix 1.

B. Non-Linear Models

While our linear models performed exceptionally well, we did observe some more appropriate models in the non-linear category.

Model	Tuning Parameters	Specificity	Kappa

Regular Discriminant Analysis	Gamma = 0.1
	Lamdba = 0.1
		0.9965	0.9948

Mixture Discriminant Analysis	Subclasses = 11	1.000	1.000

Flexible Discriminant Analysis	Degree = 2
	Nprune = 15
		1.000	0.9975

K-Nearest Neighbors	K = 1	1.000	1.000
Naive Bayes	N/A	0.8529	0.8394

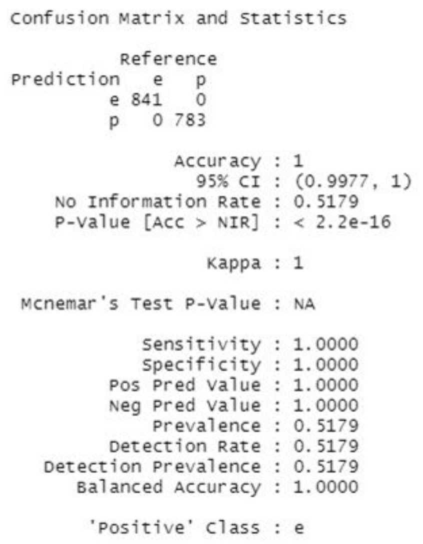
Support Vector Machine (Radial)	Sigma = 0.00736
	C = 4
		1.000	1.000

Neural Network	Size = 3
	Decay = 1
		1.000	1.000

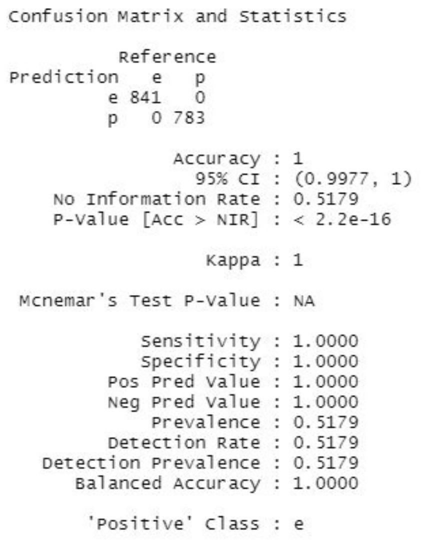
Several of our non-linear models gave perfect values for specificity with the other models (RDA and Naive Bayes) also giving very good values. The top two models are highlighted in the table and were then used to predict on the test set for a final model selection. We chose these two models because they had specificity values of 1 and they have the least amount of tuning parameters meaning they are simpler models to build and compute. Again, we need a model with a high specificity value for minimizing false positives. For further insights into the nonlinear classification models, their outputs, and their tuning plots, please refer to Appendix 2.

C. Optimization

The two models we chose to predict with the testing set were MDA and KNN as they performed the best in terms of specificity and we're the easiest to interpret. Their testing outputs are as follows, with KNN on the left and MDA on the right.

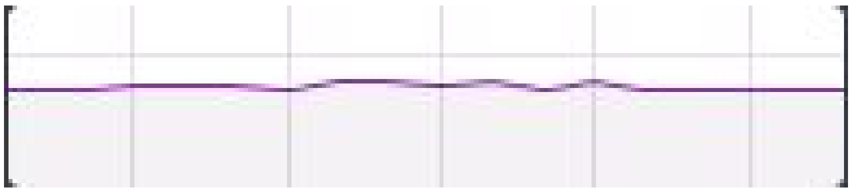


```
Confusion Matrix and Statistics

          Reference
Prediction   e   p
         e 841   0
         p   0 783

               Accuracy : 1
                 95% CI : (0.9977, 1)
    No Information Rate : 0.5179
    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 : 0.5179
         Detection Rate : 0.5179
   Detection Prevalence : 0.5179
      Balanced Accuracy : 1.0000

       'Positive' Class : e
```

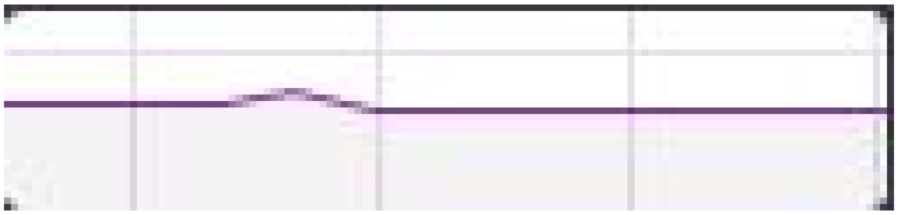


```
Confusion Matrix and Statistics

          Reference
Prediction   e   p
         e 841   0
         p   0 783

               Accuracy : 1
                 95% CI : (0.9977, 1)
    No Information Rate : 0.5179
    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 : 0.5179
         Detection Rate : 0.5179
   Detection Prevalence : 0.5179
      Balanced Accuracy : 1.0000

       'Positive' Class : e
```

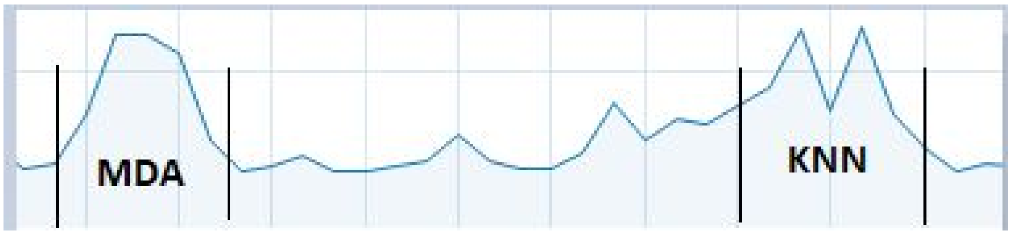
Since the two models we chose to continue testing on (MDA and KNN) gave nearly the exact same output, we had to use a different metric to compare between these two models. Since all of our models performed very well our final model choice came down to which model was the most optimal to build and test on with respect to computational time/stress. First we looked at memory allocation between the two:



Here is the amount of memory required to train our MDA model. The flatline on either end of the graph is the normal amount of memory required by your computer to be able to open up web pages and other applications, we are only concerned with the central part of the graph. We can see that the amount of memory required by the MDA model has several peaks and troughs, when comparing that to the graph for KNN's memory allocation:



We can see that KNN had a single spike in memory usage. This implies that KNN requires less memory to run. Next, we looked at CPU utilization:



While this is a crude graph, we can still extract important information from it. This graph is a graph of CPU usage, or how much stress the models put on the computer when building. On the left is the range of CPU usage that MDA required, while on the right is KNN. Both models peaked near the same CPU usage, however MDA appeared to spike then plateau at that height. KNN on the other hand spiked, immediately dropped down to normal usage, then spiked again. That drop to normal CPU usage while running KNN was a major contributor in decreasing its gross CPU utilization.

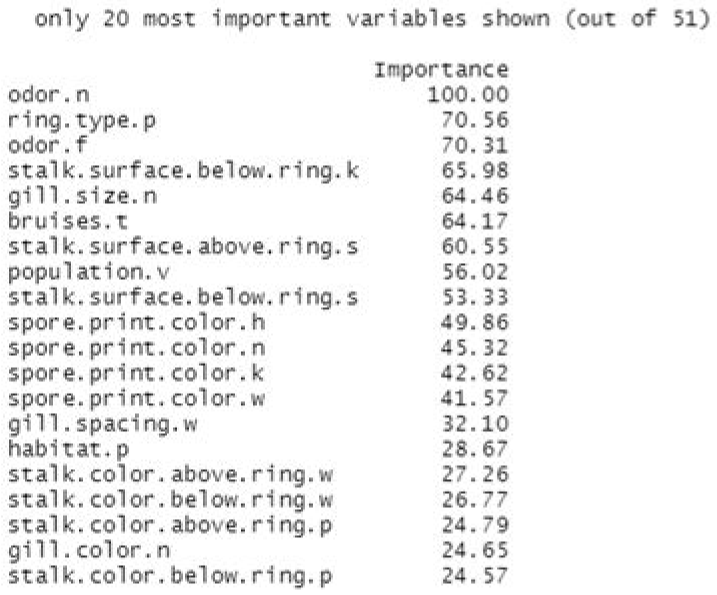
Finally, when deciding between these two models we looked at the time it took to build and predict on the models. Using the tictoc function in R we can see how long each model took to run down to .01 of a second. The MDA model took a combined time of 7.54 seconds to build and predict on, while the KNN model took a combined time of 4.01 seconds. The KNN model did take longer on the prediction side, however building the KNN model took less than half the time than building on the MDA model.

For these reasons we have decided to continue with KNN as our final model. Something to note is that these observations are all empirical evidence. KNN may not be the more optimal model theoretically or when expanded onto a much larger set, say in the millions of data entries. However, this is what we observed so these are the assumptions we must go under.

5 Summary

As mentioned in the previous section, we have chosen K-Nearest Neighbors for our final model. KNN returned a specificity value of 1.000 along with a Kappa value of 1 with a k value of 1. It was the simplest model that still gave us very good results both on the training and testing sets. Boasting the kappa value of 1 and specificity of 1 can help reassure us that our model isn't just randomly guessing these mushroom types and it outputs data that is potentially trustable.

While it's nice to know this model has a 1.0 specificity and 100% accuracy rate, it would also be nice to know what predictors are most prominent in the predictions. The following figure displays the top 20 important predictors for the KNN model, indicating that not having a smell is the most important predictor.

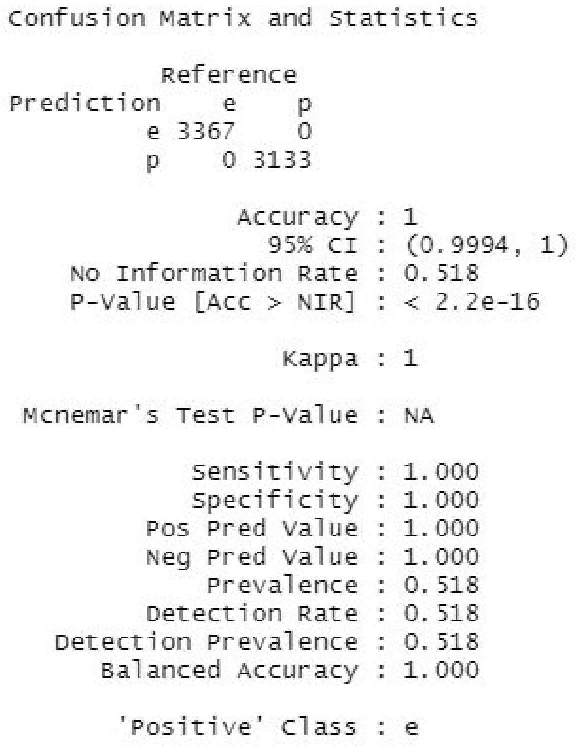


```
  only 20 most important variables shown (out of 51)

                             Importance
odor.n                          100.00
ring.type.p                      70.56
odor.f                           70.31
stalk.surface.below.ring.k       65.98
gill.size.n                      64.46
bruises.t                        64.17
stalk.surface.above.ring.s       60.55
population.v                     56.02
stalk.surface.below.ring.s       53.33
spore.print.color.h              49.86
spore.print.color.n              45.32
spore.print.color.k              42.62
spore.print.color.w              41.57
gill.spacing.w                   32.10
habitat.p                        28.67
stalk.color.above.ring.w         27.26
stalk.color.below.ring.w         26.77
stalk.color.above.ring.p         24.79
gill.color.n                     24.65
stalk.color.below.ring.p         24.57
```

This model could surely help mushroom foragers decide whether a mushroom is worth the risk of trying out or not. For the tuning plot and outputs of the KNN model, please refer to Appendix 2.

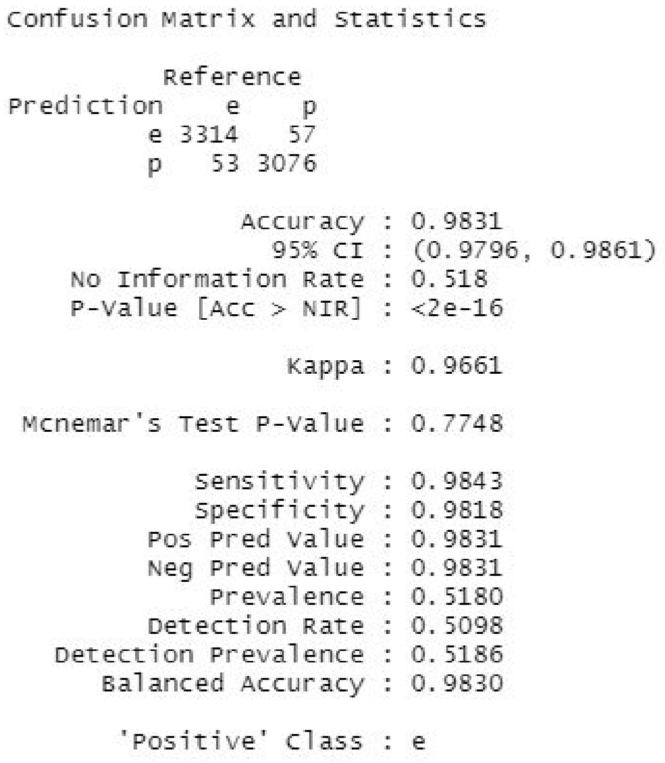
6 Appendix 1: Linear Classification Model Outputs/Plots

1) Logistic Regression



```
Confusion Matrix and Statistics

          Reference
Prediction    e    p
         e 3367    0
         p    0 3133

               Accuracy : 1
                 95% CI : (0.9994, 1)
    No Information Rate : 0.518
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.000
            Specificity : 1.000
         Pos Pred Value : 1.000
         Neg Pred Value : 1.000
             Prevalence : 0.518
         Detection Rate : 0.518
   Detection Prevalence : 0.518
      Balanced Accuracy : 1.000

       'Positive' Class : e
```

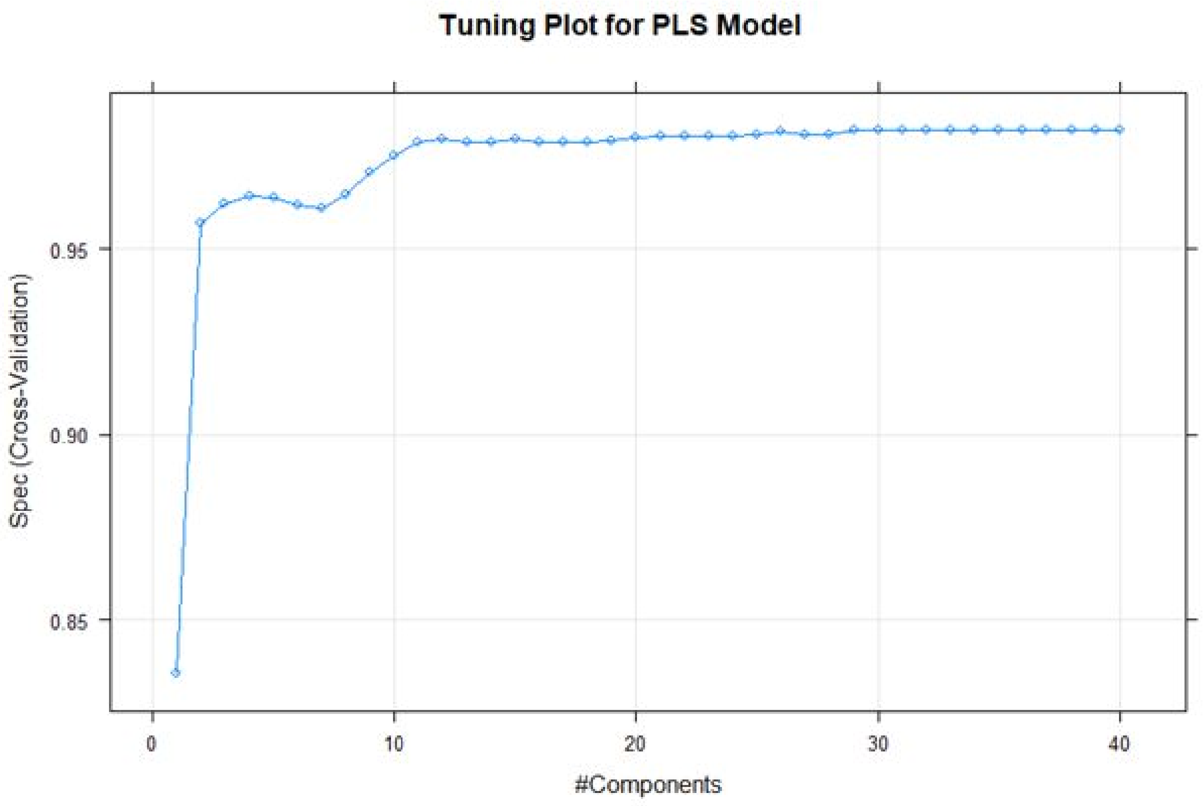
2) Linear Discriminant Analysis



```
Confusion Matrix and Statistics

          Reference
Prediction    e    p
         e 3314   57
         p   53 3076

               Accuracy : 0.9831
                 95% CI : (0.9796, 0.9861)
    No Information Rate : 0.518
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9661

 Mcnemar's Test P-Value : 0.7748

            Sensitivity : 0.9843
            Specificity : 0.9818
         Pos Pred Value : 0.9831
         Neg Pred Value : 0.9831
             Prevalence : 0.5180
         Detection Rate : 0.5098
   Detection Prevalence : 0.5186
      Balanced Accuracy : 0.9830

       'Positive' Class : e
```

3) Partial Least Squares

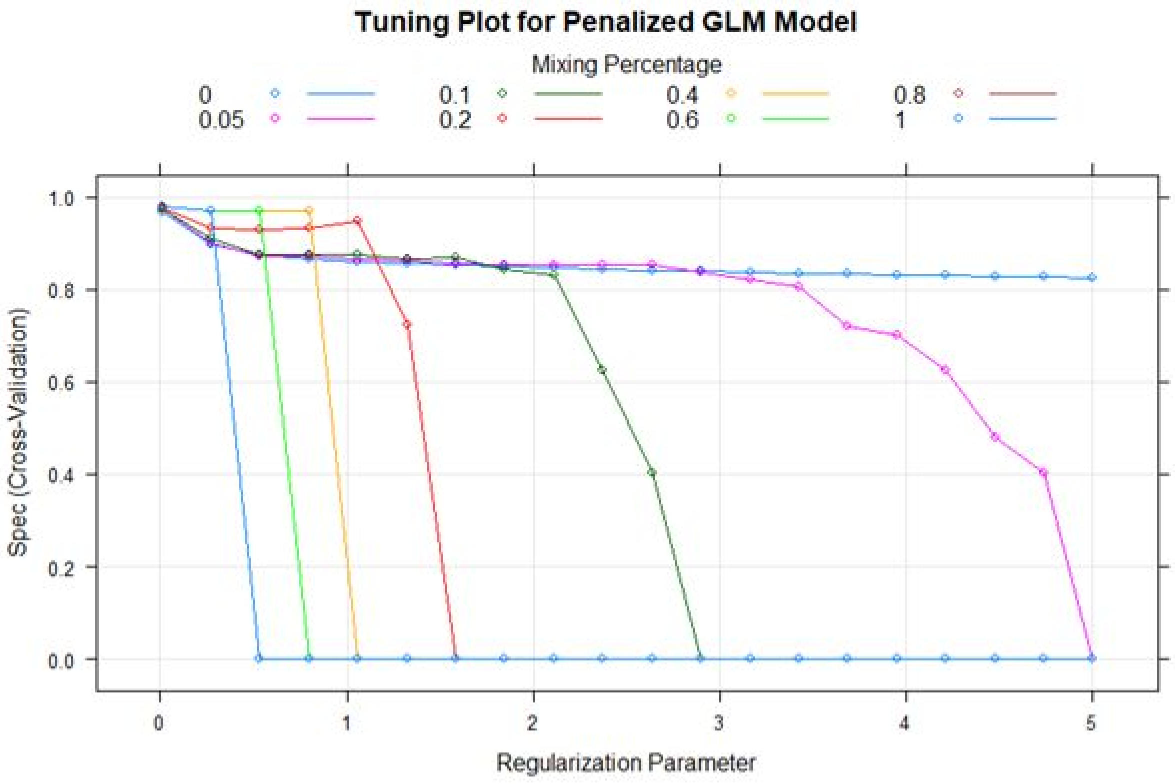
```
Confusion Matrix and Statistics

          Reference
Prediction    e    p
         e 3313   57
         p   54 3076

               Accuracy : 0.9829
                 95% CI : (0.9795, 0.9859)
    No Information Rate : 0.518
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9658

 Mcnemar's Test P-Value : 0.8494

            Sensitivity : 0.9840
            Specificity : 0.9818
         Pos Pred Value : 0.9831
         Neg Pred Value : 0.9827
             Prevalence : 0.5180
         Detection Rate : 0.5097
   Detection Prevalence : 0.5185
      Balanced Accuracy : 0.9829

       'Positive' Class : e
```



```
Spec was used to select the optimal model using the largest value.
The final value used for the model was ncomp = 29.
```

4) Penalized GLM

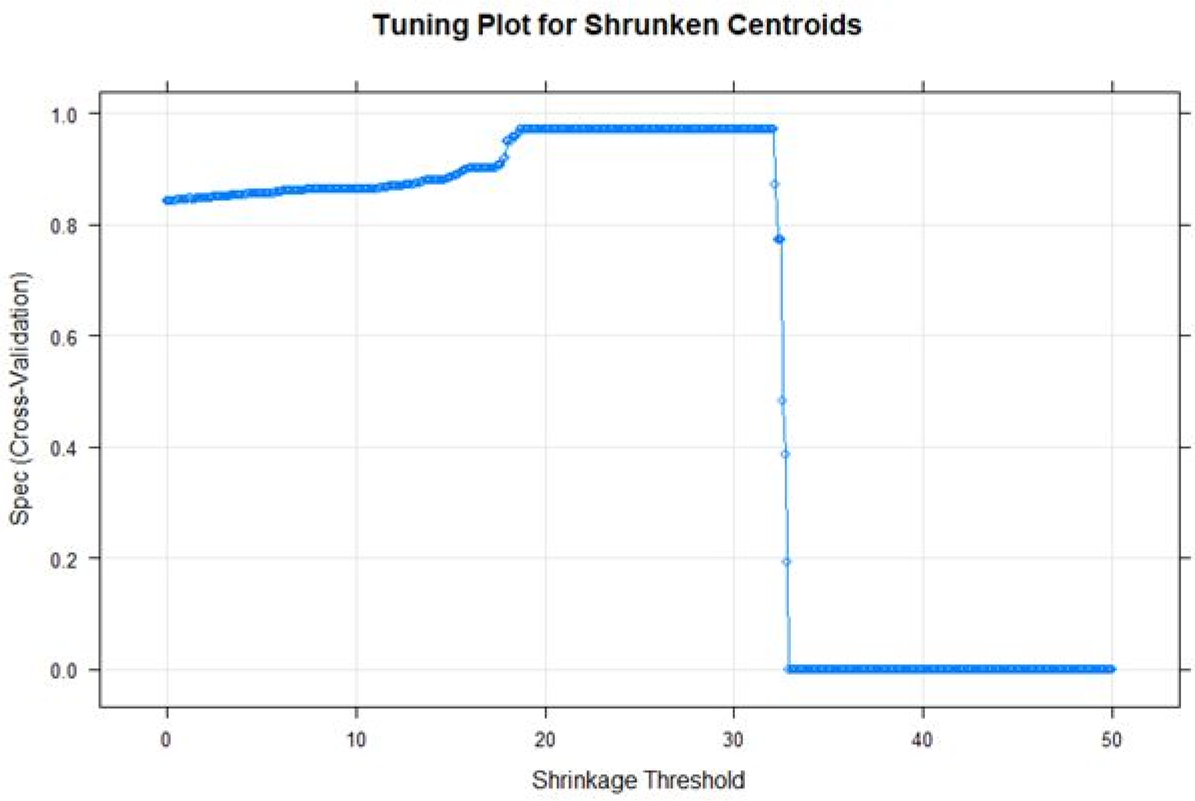
```
Confusion Matrix and Statistics

          Reference
Prediction    e    p
         e 3336   69
         p   31 3064

               Accuracy : 0.9846
                 95% CI : (0.9813, 0.9875)
    No Information Rate : 0.518
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9692

 Mcnemar's Test P-Value : 0.0002156

            Sensitivity : 0.9908
            Specificity : 0.9780
         Pos Pred Value : 0.9797
         Neg Pred Value : 0.9900
             Prevalence : 0.5180
         Detection Rate : 0.5132
   Detection Prevalence : 0.5238
      Balanced Accuracy : 0.9844

       'Positive' Class : e
```



```
Spec was used to select the optimal model using the largest value.
The final values used for the model were alpha = 0.4 and lambda = 0.01.
```

5) Nearest Shrunken Centroids

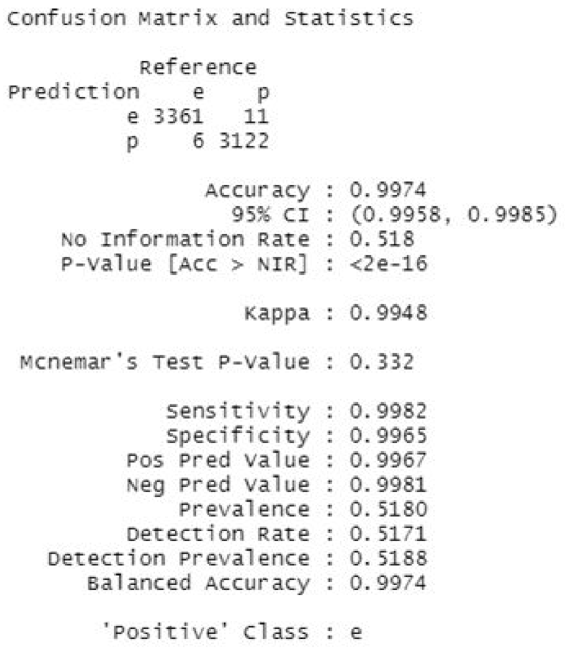
```
                    Reference
          Prediction    e    p
                   e 3289   95
                   p   78 3038

                        Accuracy : 0.9734
                          95% CI : (0.9692, 0.9772)
             No Information Rate : 0.518
             P-Value [Acc > NIR] : <2e-16

                           Kappa : 0.9467

          Mcnemar's Test P-Value : 0.2238

                     Sensitivity : 0.9768
                     Specificity : 0.9697
                  Pos Pred Value : 0.9719
                  Neg Pred Value : 0.9750
                      Prevalence : 0.5180
                  Detection Rate : 0.5060
            Detection Prevalence : 0.5206
               Balanced Accuracy : 0.9733

                'Positive' Class : e
```



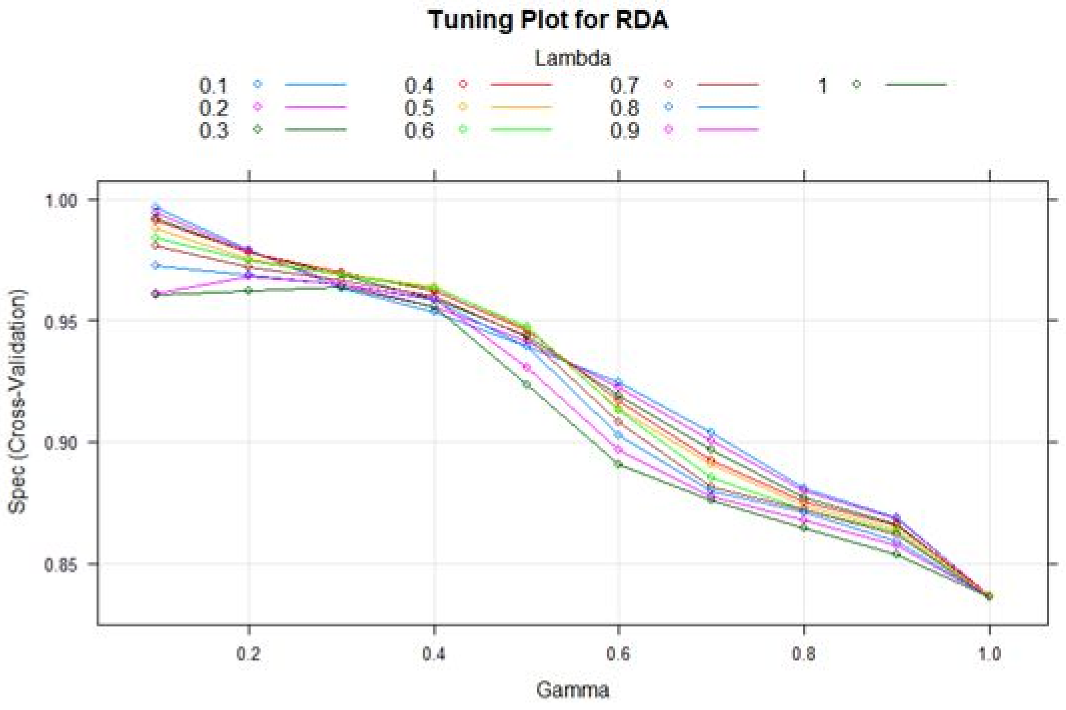
```
Spec was used to select the optimal model using the largest value.
The final value used for the model was threshold = 18.7.
```

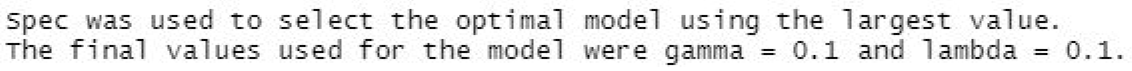
7 Appendix 2: Nonlinear Classification Model Outputs/Plots

1) Regular Discriminant Analysis



```
Confusion Matrix and Statistics

          Reference
Prediction    e    p
         e 3361   11
         p    6 3122

               Accuracy : 0.9974
                 95% CI : (0.9958, 0.9985)
    No Information Rate : 0.518
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9948

 Mcnemar's Test P-Value : 0.332

            Sensitivity : 0.9982
            Specificity : 0.9965
         Pos Pred Value : 0.9967
         Neg Pred Value : 0.9981
             Prevalence : 0.5180
         Detection Rate : 0.5171
   Detection Prevalence : 0.5188
      Balanced Accuracy : 0.9974

       'Positive' Class : e
```

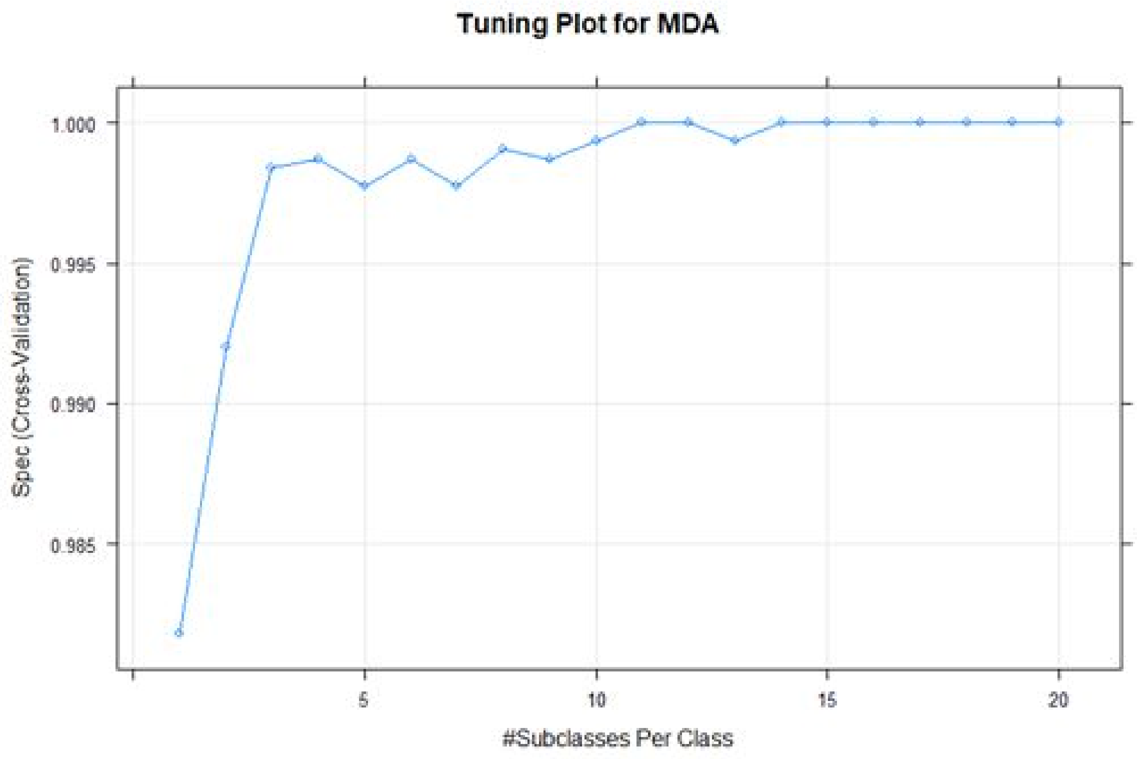




```
Spec was used to select the optimal model using the largest value.
The final values used for the model were gamma = 0.1 and lambda = 0.1.
```

2) Mixture Discriminant Analysis

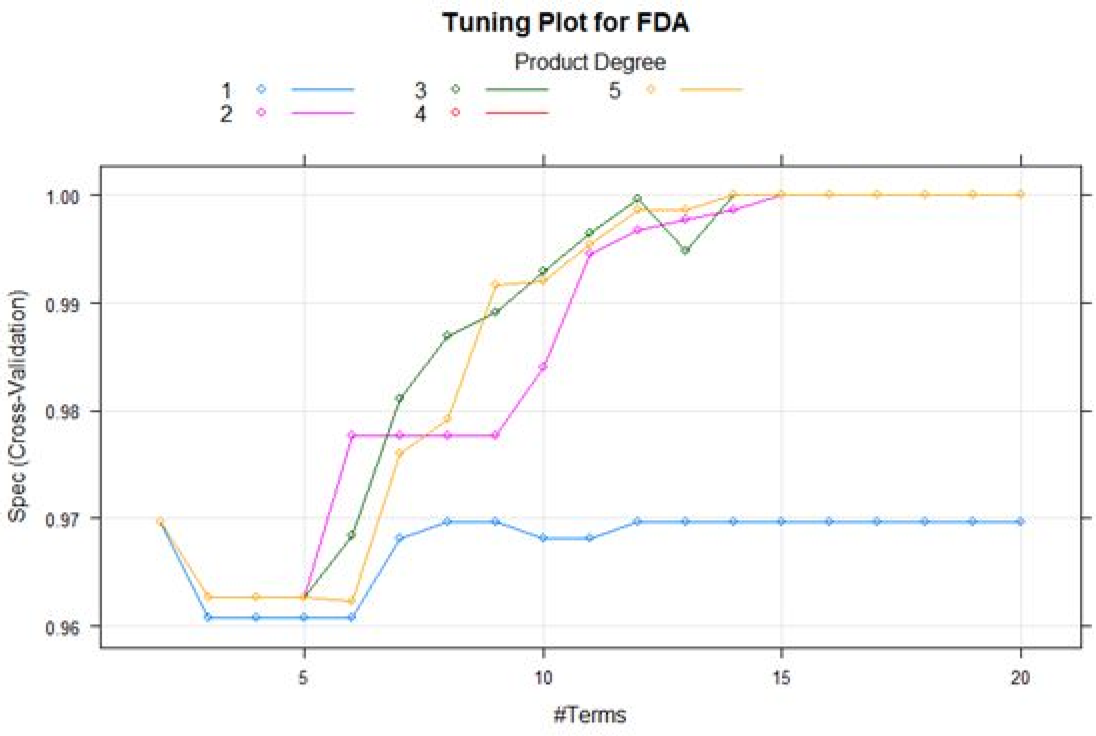
```
Confusion Matrix and Statistics

          Reference
Prediction    e    p
         e 3367    0
         p    0 3133

               Accuracy : 1
                 95% CI : (0.9994, 1)
    No Information Rate : 0.518
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.000
            Specificity : 1.000
         Pos Pred Value : 1.000
         Neg Pred Value : 1.000
             Prevalence : 0.518
         Detection Rate : 0.518
   Detection Prevalence : 0.518
      Balanced Accuracy : 1.000

       'Positive' Class : e
```



```
Spec was used to select the optimal model using the largest value.
The final value used for the model was subclasses = 11.
```

3) Flexible Discriminant Analysis

```
Confusion Matrix and Statistics

          Reference
Prediction    e    p
         e 3359    0
         p    8 3133

               Accuracy : 0.9988
                 95% CI : (0.9976, 0.9995)
    No Information Rate : 0.518
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.9975

 Mcnemar's Test P-Value : 0.01333

            Sensitivity : 0.9976
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9975
             Prevalence : 0.5180
         Detection Rate : 0.5168
   Detection Prevalence : 0.5168
      Balanced Accuracy : 0.9988

       'Positive' Class : e
```

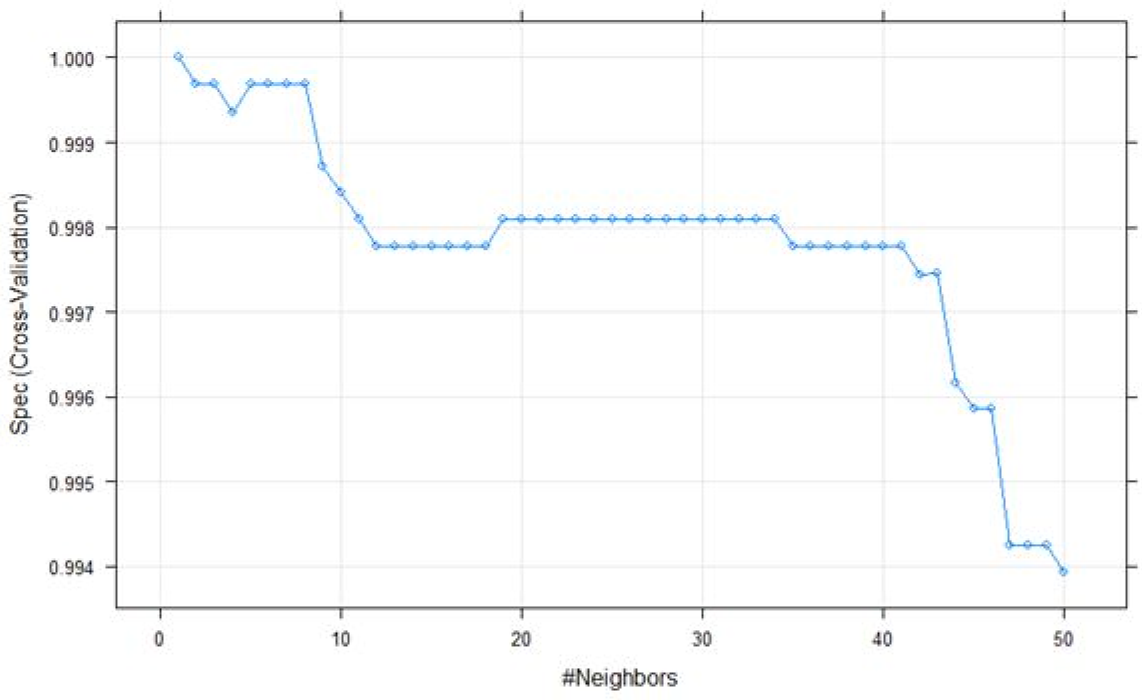


```
Spec was used to select the optimal model using the largest value.
The final values used for the model were degree = 2 and nprune = 15.
```

4) K-Nearest-Neighbor

```
Confusion Matrix and Statistics

          Reference
Prediction    e    p
         e 3367    0
         p    0 3133

               Accuracy : 1
                 95% CI : (0.9994, 1)
    No Information Rate : 0.518
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.000
            Specificity : 1.000
         Pos Pred Value : 1.000
         Neg Pred Value : 1.000
             Prevalence : 0.518
         Detection Rate : 0.518
   Detection Prevalence : 0.518
      Balanced Accuracy : 1.000

       'Positive' Class : e
```

Tuning Plot for KNN

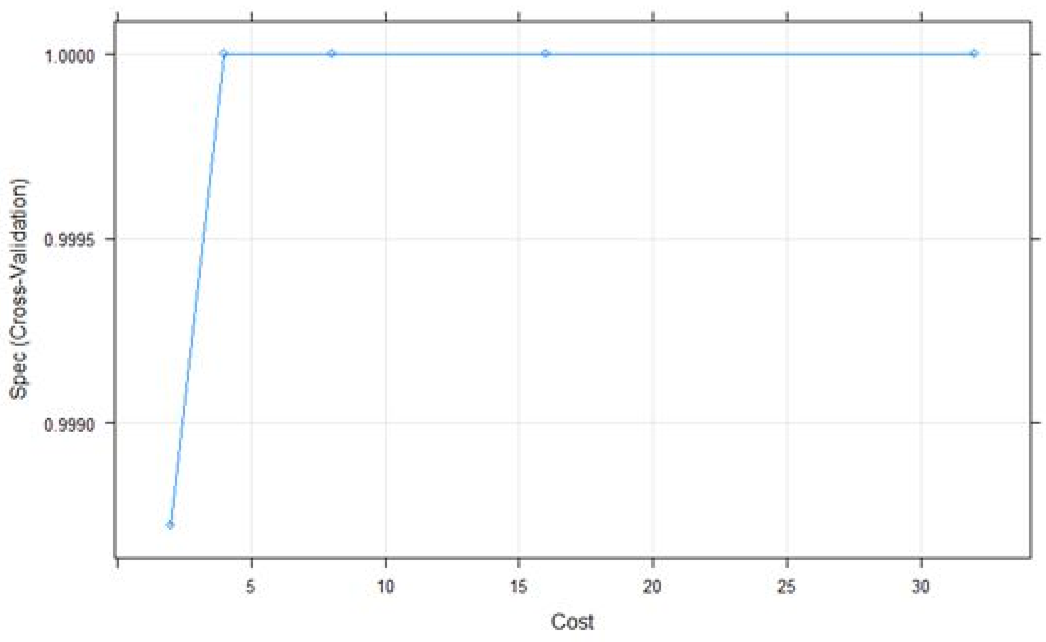


```
Spec was used to select the optimal model using the largest value.
The final value used for the model was k = 1.
```

5) Support Vector Machine (Radial)

```
Confusion Matrix and Statistics

          Reference
Prediction    e    p
         e 3367    0
         p    0 3133

               Accuracy : 1
                 95% CI : (0.9994, 1)
    No Information Rate : 0.518
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.000
            Specificity : 1.000
         Pos Pred Value : 1.000
         Neg Pred Value : 1.000
             Prevalence : 0.518
         Detection Rate : 0.518
   Detection Prevalence : 0.518
      Balanced Accuracy : 1.000

       'Positive' Class : e
```

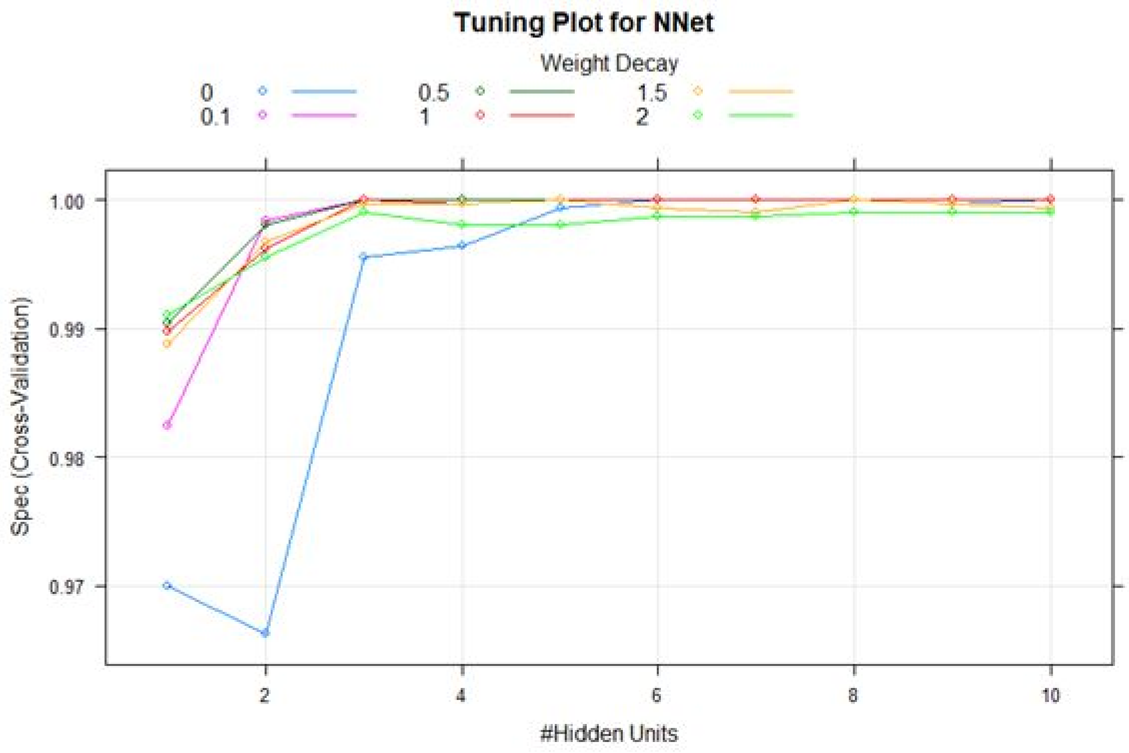
Tuning Plot for SVM



```
Tuning parameter 'sigma' was held constant at a value of 0.007310723
Spec was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.007310723 and C = 4.
```

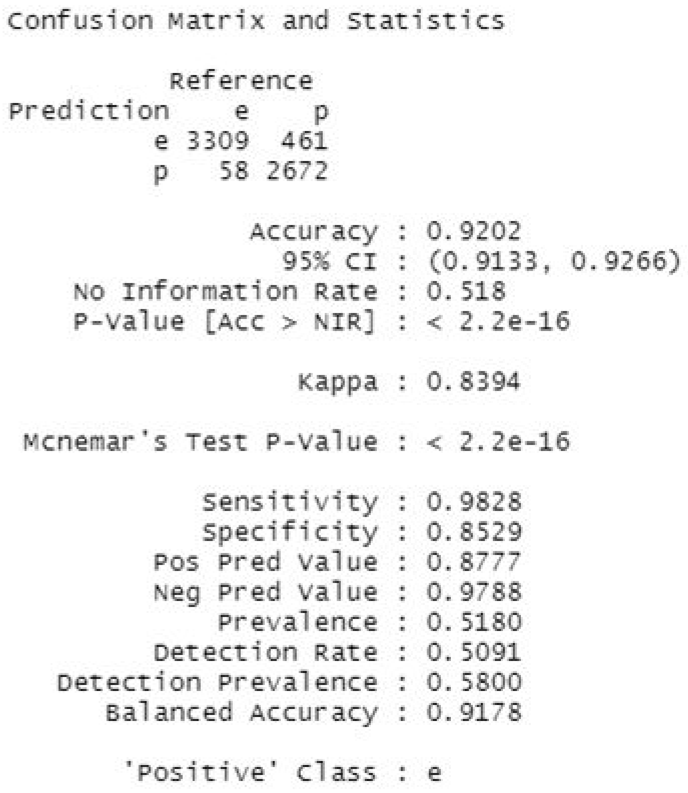
6) Neural Network

```
Confusion Matrix and Statistics

          Reference
Prediction    e    p
         e 3367    0
         p    0 3133

               Accuracy : 1
                 95% CI : (0.9994, 1)
    No Information Rate : 0.518
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.000
            Specificity : 1.000
         Pos Pred Value : 1.000
         Neg Pred Value : 1.000
             Prevalence : 0.518
         Detection Rate : 0.518
   Detection Prevalence : 0.518
      Balanced Accuracy : 1.000

       'Positive' Class : e
```



```
Spec was used to select the optimal model using the largest value.
The final values used for the model were size = 3 and decay = 1.
```

7) Naive Bayes



```
Confusion Matrix and Statistics

          Reference
Prediction    e    p
         e 3309  461
         p   58 2672

               Accuracy : 0.9202
                 95% CI : (0.9133, 0.9266)
    No Information Rate : 0.518
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8394

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9828
            Specificity : 0.8529
         Pos Pred Value : 0.8777
         Neg Pred Value : 0.9788
             Prevalence : 0.5180
         Detection Rate : 0.5091
   Detection Prevalence : 0.5800
      Balanced Accuracy : 0.9178

       'Positive' Class : e
```

8 R Code

```
library("caret")
library('corrplot')
library(AppliedPredictiveModeling)
library("e1071")
library("lattice")
library("ggplot2")
library("naniar")
library('plyr')
library('pamr')
library('rms')
library('mda')
library('MASS')
library('klaR')
library(nnet)
library('pROC')
library('kernlab')
library('rgl')
library('ggplot2')
library('tictoc')

#  Authors: Caleb Hiltunen, Ian Boulis
#  Date: 9/14/20
#  Description: Creating training/testing sets, detecting collinearity, creating models

mushrooms <- read.csv('mushrooms.csv')

#  Making everything workable
mushrooms$class <- as.factor(mushrooms$class)
mushrooms$cap.shape <- as.factor(mushrooms$cap.shape)
mushrooms$cap.surface <- as.factor(mushrooms$cap.surface)
mushrooms$cap.color <- as.factor(mushrooms$cap.color)
mushrooms$bruises <- as.factor(mushrooms$bruises)
mushrooms$odor <- as.factor(mushrooms$odor)
mushrooms$gill.attachment <- as.factor(mushrooms$gill.attachment)
mushrooms$gill.spacing <- as.factor(mushrooms$gill.spacing)
mushrooms$gill.size <- as.factor(mushrooms$gill.size)
mushrooms$gill.color <- as.factor(mushrooms$gill.color)
mushrooms$stalk.shape <- as.factor(mushrooms$stalk.shape)
mushrooms$stalk.root <- as.factor(mushrooms$stalk.root)
```

```
mushrooms$stalk.surface.above.ring <- as.factor(mushrooms$stalk.surface.above.ring)
mushrooms$stalk.surface.below.ring <- as.factor(mushrooms$stalk.surface.below.ring)
mushrooms$stalk.color.above.ring <- as.factor(mushrooms$stalk.color.above.ring)
mushrooms$stalk.color.below.ring <- as.factor(mushrooms$stalk.color.below.ring)
mushrooms$veil.color <- as.factor(mushrooms$veil.color)
mushrooms$ring.number <- as.factor(mushrooms$ring.number)
mushrooms$ring.type <- as.factor(mushrooms$ring.type)
mushrooms$spore.print.color <- as.factor(mushrooms$spore.print.color)
mushrooms$population <- as.factor(mushrooms$population)
mushrooms$habitat <- as.factor(mushrooms$habitat)
mushrooms$veil.type <- as.factor(mushrooms$veil.type)

# Checking for missing data
vis_miss(mushrooms)

# Distribution graphs
colnames <- colnames(mushrooms)
for(i in 1:23){
 barplot(prop.table(table(mushrooms[i])),
    main = colnames[i],
      col = "gray",
     border = "white",
      ylim = c(0,1))
}

# Dummy variables
dummRes <-
dummyVars("~+cap.shape+cap.surface+cap.color+bruises+odor+gill.spacing+gill.size+
gill.color+veil.color+gill.attachment+stalk.shape+stalk.root+stalk.surface.above.ring+stal
k.surface.below.ring+stalk.color.above.ring+stalk.color.below.ring+ring.number+ring.typ
e+spore.print.color+population+habitat", data = mushrooms, fullRank = TRUE)
Add_dumm <- data.frame(predict(dummRes,newdata = mushrooms))

# Near 0 Variance
varmush <- nearZeroVar(Add_dumm,95/5)
varmush
filter_Add_dum <- Add_dumm[,-varmush]

# Correlation
mush2 <- cor(filter_Add_dum)
```

```
corrplot(mush2)
highCorr <- findCorrelation(mush2,0.8)
length(highCorr)
highCorr
filter_Add_dum_2 <- filter_Add_dum[,-highCorr]
mush3 <- cor(filter_Add_dum_2)
corrplot(mush3)

# Center and Scaling
mush4 <- scale(filter_Add_dum_2, center = TRUE, scale = TRUE)
mush5 <- data.frame(mush4)

# Splitting the data
set.seed(121)
trainingSetHold <- createDataPartition(mushrooms$class, p = 0.80, list = FALSE)
trainingSetX <- mush5[ trainingSetHold, ]
trainingSetY <- mushrooms[ trainingSetHold, 1 ]
testingSetX <- mush5[-trainingSetHold,]
testingSetY <- mushrooms[-trainingSetHold, 1]

# PCA plots
plot3d(x = pca2$x[,c(1,3,2)], col = as.numeric(trainingSetY),size = 10, main = '3D
Scatterplot of PCA of Training Set')
scatterplot3d::scatterplot3d(x = pca$x[,c(1,3,2)], color = as.numeric(mushrooms$class),
main = '3D Scatterplot of PCA of Entire Dataset')
scatterplot3d::scatterplot3d(x = pca2$x[,c(2,1,3)], color = as.numeric(trainingSetY),
main = '3D Scatterplot of PCA of Training Dataset')

# Resampling method
ctrl <- trainControl(method = 'cv',
                 number = 10,
                 classProbs = TRUE,
                 summaryFunction = twoClassSummary,
                 savePredictions = TRUE)

## LINEAR CLASSIFICATION MODELS ##
# Logistic Regression
set.seed(111)
lr <- train(x = trainingSetX,
         y = trainingSetY,
            method = 'glm',
            metric = 'Spec',
            trControl = ctrl,
            maxit = 250)
lr
confusionMatrix(lr$pred$pred,
               reference = lr$pred$obs)

# LDA
set.seed(111)
lda <- train(x = trainingSetX,
             y = trainingSetY,
             method = 'lda',
             metric = 'Spec',
             trControl = ctrl)
lda
confusionMatrix(lda$pred$pred,
               reference = lr$pred$obs)

# PLS
plsGrid <- expand.grid(.ncomp = 1:40)
set.seed(111)
pls <- train(x = trainingSetX,
             y = trainingSetY,
             method = 'pls',
             metric = 'Spec',
             trControl = ctrl,
             tuneGrid = expand.grid(.ncomp = 29))
pls
plot(pls,
     main = 'Tuning Plot for PLS Model')
confusionMatrix(pls$pred$pred,
               reference = pls$pred$obs)

# Penalized
glmnGrid <- expand.grid(.alpha = c(0, 0.05, .1, .2, .4, .6, .8, 1),
                       .lambda = seq(.01, 5, length = 20))
set.seed(111)
glm <- train(x = trainingSetX,
             y = trainingSetY,
            method = 'glmnet',
            metric = 'Spec',
            trControl = ctrl,
            tuneGrid = expand.grid(.alpha = 0.4, .lambda = 0.01))
glm
plot(glm,
    main = 'Tuning Plot for Penalized GLM Model')
confusionMatrix(glm$pred$pred,
              reference = glm$pred$obs)

# Shrunken Centroids
nscGrid <- data.frame(.threshold = seq(0, 50, by = 0.1))
set.seed(111)
cent <- train(x = trainingSetX,
             y = trainingSetY,
             method = 'pam',
             trControl = ctrl,
             tuneGrid = nscGrid,
             metric = 'Spec')
cent
plot(cent,
    main = 'Tuning Plot for Shrunken Centroids')
confusionMatrix(cent$pred$pred,
              reference = cent$pred$obs)

## NONLINEAR CLASSIFICATION MODELS ##
# RDA
rdaGrid <- expand.grid(.gamma = seq(0.1, 1, by = .1),
                     .lambda = seq(0.1, 1, by = .1))
set.seed(111)
rda1 <- train(x = trainingSetX,
             y = trainingSetY,
             method = 'rda',
             trControl = ctrl,
             metric = 'Spec',
             tuneGrid = expand.grid(.gamma = 0.1, .lambda = 0.1),
             na.action = na.pass)
rda1
plot(rda1,
    main = 'Tuning Plot for RDA')
```

```
confusionMatrix(rda1$pred$pred,
             reference = rda1$pred$obs)

# MDA
set.seed(111)
tic()
mda1 <- train(x = trainingSetX,
            y = trainingSetY,
            method = 'mda',
            metric = 'Spec',
            trControl = ctrl,
            tuneGrid = expand.grid(.subclasses = 11))
toc()
mda1
plot(mda1,
    main = 'Tuning Plot for MDA')
confusionMatrix(mda1$pred$pred,
             reference = mda1$pred$obs)

# Neural Network
nnetGrid <- expand.grid(.size = 1:10, .decay = c(0, 0.1, 0.5, 1, 1.5, 2))
maxSize <- max(nnetGrid$.size)
mxWts <- (maxSize * (51 + 1) + (maxSize + 1)* 2)
set.seed(111)
nnet1 <- train(x = trainingSetX,
             y = trainingSetY,
             method = 'nnet',
             metric = 'Spec',
             tuneGrid = expand.grid(.size = 3, .decay = 1),
             maxit = 2000,
             MaxNWts = mxWts,
             trControl = ctrl,
             trace = FALSE)
nnet1
plot(nnet1,
    main = 'Tuning Plot for NNet')
confusionMatrix(nnet1$pred$pred,
             reference = nnet1$pred$obs)

# FDA
```

```
marsGrid <- expand.grid(.degree = 1:5, .nprune = 2:20)
set.seed(111)
fda <- train(x = trainingSetX,
            y = trainingSetY,
            method = 'fda',
            metric = 'Spec',
            tuneGrid = expand.grid(.degree = 2, .nprune = 15),
            trControl = ctrl)
fda
plot(fda,
    main = 'Tuning Plot for FDA')
confusionMatrix(fda$pred$pred,
               reference = fda$pred$obs)

# SVM
sigmaRange <- sigest(as.matrix(trainingSetX))
svmGrid <- expand.grid(.sigma = sigmaRange[1],
                      .C = 4)
set.seed(111)
svm1 <- train(x = trainingSetX,
             y = trainingSetY,
             method = 'svmRadial',
             metric = 'Spec',
             tuneGrid = svmGrid,
             trControl = ctrl)
svm1
plot(svm1,
    main = 'Tuning Plot for SVM')
confusionMatrix(svm1$pred$pred,
               reference = svm1$pred$obs)

# KNN
set.seed(111)
tic()
knn1 <- train(x = trainingSetX,
             y = trainingSetY,
             method = 'knn',
             metric = 'Spec',
             tuneGrid = data.frame(.k = 1),
             trControl = ctrl)
```

```
toc()
knn1
plot(knn1,
    main = 'Tuning Plot for KNN')
confusionMatrix(knn1$pred$pred,
            reference = knn1$pred$obs)

# Naive bayes
set.seed(111)
bayes1 <- train(x = trainingSetX,
            y = trainingSetY,
            method = 'nb',
            metric = 'Spec',
            tuneGrid = expand.grid(.fL = 10, .adjust = TRUE, .usekernel = TRUE),
            trControl = ctrl)
bayes1
plot(bayes1,
    main = 'Tuning Plot for Naive Bayes')
confusionMatrix(bayes1$pred$pred,
            reference = bayes1$pred$obs)

# Testing on two best ones
# KNN
tic()
knnPred <- predict(knn1, newdata = testingSetX)
toc()
postResample(knnPred, testingSetY)
confusionMatrix(knnPred,
            reference = testingSetY)

# MDA
tic()
MDAPred <- predict(mda1, newdata = testingSetX)
toc()
postResample(MDAPred, testingSetY)
confusionMatrix(MDAPred,
            reference = testingSetY)

# Important Predictors
varImp(knn1, metric = 'Spec')
```