Black Box Methods: Neural Networks and Support Vector Machines

Taylor Shrode

9/27/2020

Introduction

Black box methods refer to the "mechanism that transforms the input into the output which is obfuscated by an imaginary box (Lantz, 2015)." In other words, it is difficult to understand what a network is doing to return the output it gives. Support Vector Machines and Neural Networks are considered black box methods.

For this assignment, we will building a classifier using Support Vector Machines and Neural Networks while experimenting with different kernels and different parameters, respectively. The dataset we will be using contains descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms (Schlimmer, n.d.). Each species is identified as edible or poisonous. The attributes included in this dataset are described as follows (Schlimmer, n.d.):

1. class/type: edible=e, poisonous=p
2. cap-shape: bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s
3. cap-surface: fibrous=f, grooves=g, scaly=y, smooth=s
4. cap-color: brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y
5. bruises?: bruises=t, no=f
6. odor: almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s
7. gill-attachment: attached=a, descending=d, free=f, notched=n
8. gill-spacing: close=c, crowded=w, distant=d
9. gill-size: broad=b, narrow=n
10. gill-color: black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y
11. stalk-shape: enlarging=e, tapering=t
12. stalk-root: bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?
13. stalk-surface-above-ring: fibrous=f, scaly=y, silky=k, smooth=s
14. stalk-surface-below-ring: fibrous=f, scaly=y, silky=k, smooth=s
15. stalk-color-above-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y

16. stalk-color-below-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
17. veil-type: partial=p, universal=u
18. veil-color: brown=n, orange=o, white=w, yellow=y
19. ring-number: none=n, one=o, two=t
20. ring-type: cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z
21. spore-print-color: black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y
22. population: abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y
23. habitat: grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

We will be using the "class" variable as our outcome variable.

Libraries

Before we begin building our classifiers, we need to load the necessary libraries into R.

```r
library(DataExplorer)
library(caret)
library(e1071)
library("kernlab")
library(neuralnet)
```

The **DataExplorer** library allows us to perform data exploration analysis. The **caret** package allows to measure the accuracy of our predictions and to create dummy variables. The **e1071** package allows us to build our linear SVM model and the **kernlab** package allows us to build SVM models using other kernels. Finally, the **neuralnet** package is loaded to build our Neural Network classifier and experiment with different parameters.

Load Dataset

To load our data into R, we can load the data directly from the URL using the **read.csv()** function. We add the argument **sep = ","** to separate the columns in the dataset properly. Further, we set **header = FALSE** so we can specify the column names correctly. Otherwise, the first row of data is used at the column names.

```r
mushroom_df <- read.csv("http://archive.ics.uci.edu/ml/machine-learning-databases/mushroom/agaricus-lepiota.data", sep = ',', header = FALSE,
col.names = c("classes", "cap-shape", "cap-surface", "cap-color", "bruises",
"odor", "gill-attachment", "gill-spacing", "gill-size", "gill-color", "stalk-shape", "stalk-root", "stalk-surface-above-ring", "stalk-surface-below-ring",
"stalk-color-above-ring", "stalk-color-below-ring", "veil-type", "veil-color", "ring-number", "ring-type", "spore-print-color", "population",
"habitat"))
head(mushroom_df)
```

```
##   classes cap.shape cap.surface cap.color bruises odor gill.attachment
## 1       p         x           s         n       t    p               f
## 2       e         x           s         y       t    a               f
## 3       e         b           s         w       t    l               f
## 4       p         x           y         w       t    p               f
## 5       e         x           s         g       f    n               f
## 6       e         x           y         y       t    a               f
##   gill.spacing gill.size gill.color stalk.shape stalk.root
## 1            c         n          k           e          e
## 2            c         b          k           e          c
## 3            c         b          n           e          c
## 4            c         n          n           e          e
## 5            w         b          k           t          e
## 6            c         b          n           e          c
##   stalk.surface.above.ring stalk.surface.below.ring stalk.color.above.ring
## 1                        s                        s                      w
## 2                        s                        s                      w
## 3                        s                        s                      w
## 4                        s                        s                      w
## 5                        s                        s                      w
## 6                        s                        s                      w
##   stalk.color.below.ring veil.type veil.color ring.number ring.type
## 1                      w         p          w           o         p
## 2                      w         p          w           o         p
## 3                      w         p          w           o         p
## 4                      w         p          w           o         p
## 5                      w         p          w           o         e
## 6                      w         p          w           o         p
##   spore.print.color population habitat
## 1                 k          s       u
## 2                 n          n       g
## 3                 n          n       m
## 4                 k          s       u
## 5                 n          a       g
## 6                 k          n       g
```

Then, to ensure our column names are correct, we can compare the unique values to the attribute description above.

```
for (x in colnames(mushroom_df)){
  print(unique(mushroom_df[x]))
}
```

```
##   classes
## 1       p
## 2       e
##      cap.shape
## 1            x
## 3            b
## 16           s
## 17          f
## 4277        k
## 5127        c
##      cap.surface
## 1              s
## 4              y
## 15             f
## 5108           g
##      cap.color
## 1            n
## 2            y
## 3            w
## 5            g
## 940          e
## 2211         p
## 3985         b
## 4077         u
## 4166         c
## 4327         r
##   bruises
## 1       t
## 5       f
##      odor
## 1       p
## 2       a
## 3       l
## 5       n
## 1817    f
## 2211    c
## 4024    y
## 4330    s
## 6416    m
##      gill.attachment
## 1                  f
## 6039               a
##   gill.spacing
## 1            c
## 5            w
##   gill.size
## 1         n
## 2         b
##      gill.color
## 1             k
## 3             n
## 7             g
## 9             p
## 13            w
## 128           h
## 891           u
## 3985          e
## 4024          b
## 5141          r
## 6039          y
## 6041          o
##   stalk.shape
## 1           e
## 5           t
##      stalk.root
## 1             e
## 2             c
## 30            b
## 34            r
## 3985          ?
##      stalk.surface.above.ring
## 1                           s
## 81                          f
## 1817                        k
## 6913                        y
##      stalk.surface.below.ring
## 1                           s
## 15                          f
## 34                          y
## 1817                        k
##      stalk.color.above.ring
## 1                         w
## 306                       g
## 336                       p
## 1817                      n
## 2129                      b
## 3985                      e
## 6039                      o
## 6416                      c
## 6913                      y
##      stalk.color.below.ring
## 1                         w
## 306                       p
## 633                       g
## 1817                      b
## 2386                      n
## 4292                      e
## 4495                      y
## 6039                      o
## 6416                      c
##   veil.type
## 1         p
##      veil.color
## 1             w
## 6039          n
## 6376          o
## 6913          y
##      ring.number
## 1              o
## 3985           t
## 6416           n
##      ring.type
## 1            p
## 5            e
## 1817         l
## 4077         f
## 6416         n
##      spore.print.color
## 1                    k
## 2                    n
## 77                   u
## 1817                 h
## 3985                 w
## 4107                 r
## 6376                 o
## 6425                 y
## 6559                 b
##      population
## 1             s
## 2             n
## 5             a
## 9             v
## 16            y
## 3985          c
##      habitat
## 1          u
## 2          g
## 3          m
## 30         d
## 34         p
## 3985       w
## 4105       l
```

Now that we have confirmed that the column names have been specified correctly, we can move forward.

Remove Unnecessary Columns, Handle Missing Values, Convert Column Datatypes, and Data Exploration

First, we need to convert our target variable to a factor.

```
mushroom_df$classes <- factor(mushroom_df$classes, levels = c("e","p"),
labels=c("edible","poisonous"))
str(mushroom_df)
```

```
## 'data.frame':    8124 obs. of  23 variables:
##  $ classes                 : Factor w/ 2 levels "edible","poisonous": 2 1
```

```
1 2 1 1 1 1 2 1 ...
##  $ cap.shape               : chr  "x" "x" "b" "x" ...
##  $ cap.surface             : chr  "s" "s" "s" "y" ...
##  $ cap.color               : chr  "n" "y" "w" "w" ...
##  $ bruises                 : chr  "t" "t" "t" "t" ...
##  $ odor                    : chr  "p" "a" "l" "p" ...
##  $ gill.attachment         : chr  "f" "f" "f" "f" ...
##  $ gill.spacing            : chr  "c" "c" "c" "c" ...
##  $ gill.size               : chr  "n" "b" "b" "n" ...
##  $ gill.color              : chr  "k" "k" "n" "n" ...
##  $ stalk.shape             : chr  "e" "e" "e" "e" ...
##  $ stalk.root              : chr  "e" "c" "c" "e" ...
##  $ stalk.surface.above.ring: chr  "s" "s" "s" "s" ...
##  $ stalk.surface.below.ring: chr  "s" "s" "s" "s" ...
##  $ stalk.color.above.ring  : chr  "w" "w" "w" "w" ...
##  $ stalk.color.below.ring  : chr  "w" "w" "w" "w" ...
##  $ veil.type               : chr  "p" "p" "p" "p" ...
##  $ veil.color              : chr  "w" "w" "w" "w" ...
##  $ ring.number             : chr  "o" "o" "o" "o" ...
##  $ ring.type               : chr  "p" "p" "p" "p" ...
##  $ spore.print.color       : chr  "k" "n" "n" "k" ...
##  $ population              : chr  "s" "n" "n" "s" ...
##  $ habitat                 : chr  "u" "g" "m" "u" ...
```

Our outcome variable has been successfully converted to a factor. Notice, that the remainder of variables are categorical variables. This is not an issue when working with SVM's, but we will need to convert them to numeric values when we build our Neural Network classifier.

Now, we will remove the **veil.type** column because it only contains one unique value (level/constant).

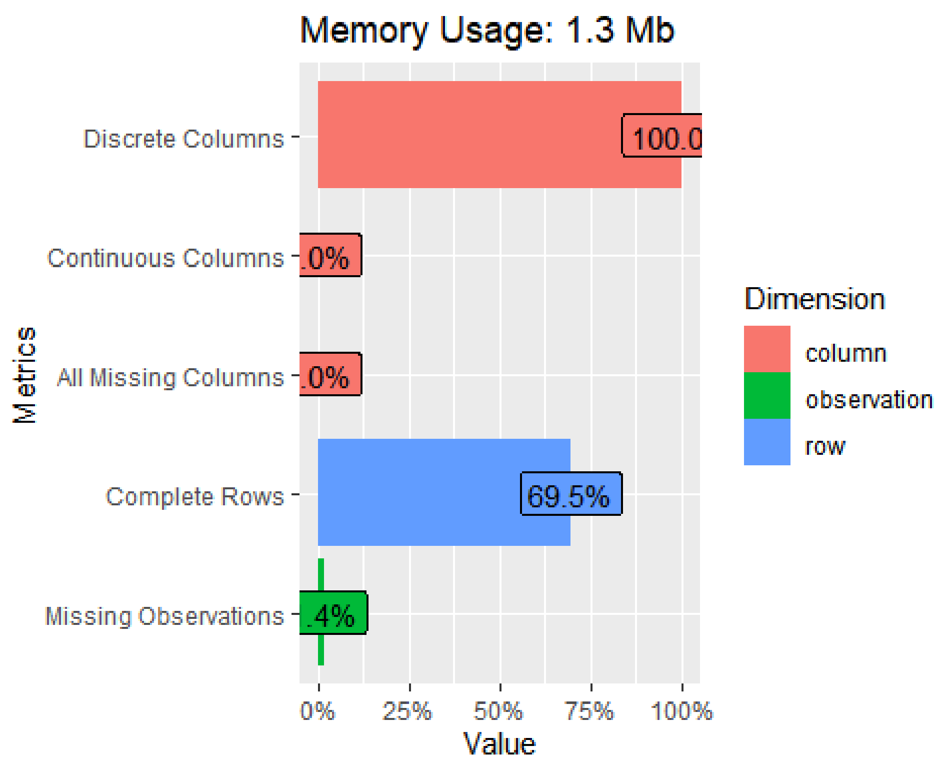
```
mushroom_df$veil.type <- NULL #remove veil.type column
```

As indicated in our attribute information, there are missing values in our data, which are represented by "?". First, we need to convert the "?" values to null values (NA) and then we can identify the location and amount of missing values (R - replace all 0 values to na, n.d.).

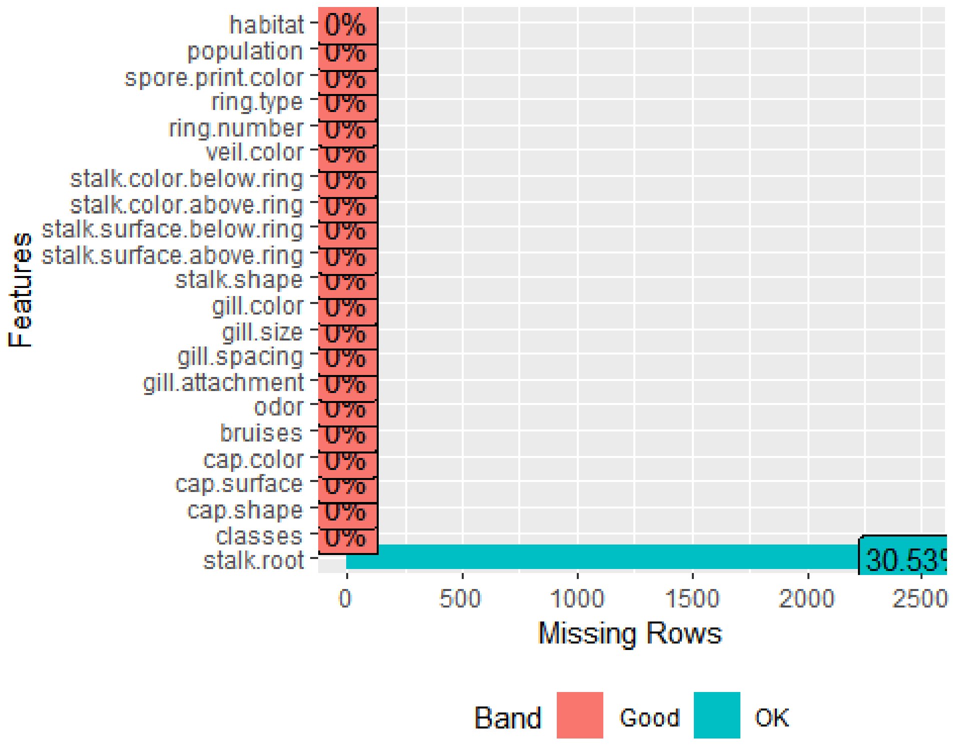
```
mushroom_df[mushroom_df == "?"] <- NA
introduce(mushroom_df)

##   rows columns discrete_columns continuous_columns all_missing_columns
## 1 8124      22               22                  0                   0
##   total_missing_values complete_rows total_observations memory_usage
## 1                 2480          5644             178728      1409752
```

We can plot our **intro** information above and we can also plot our missing values.

```
plot_intro(mushroom_df)
```

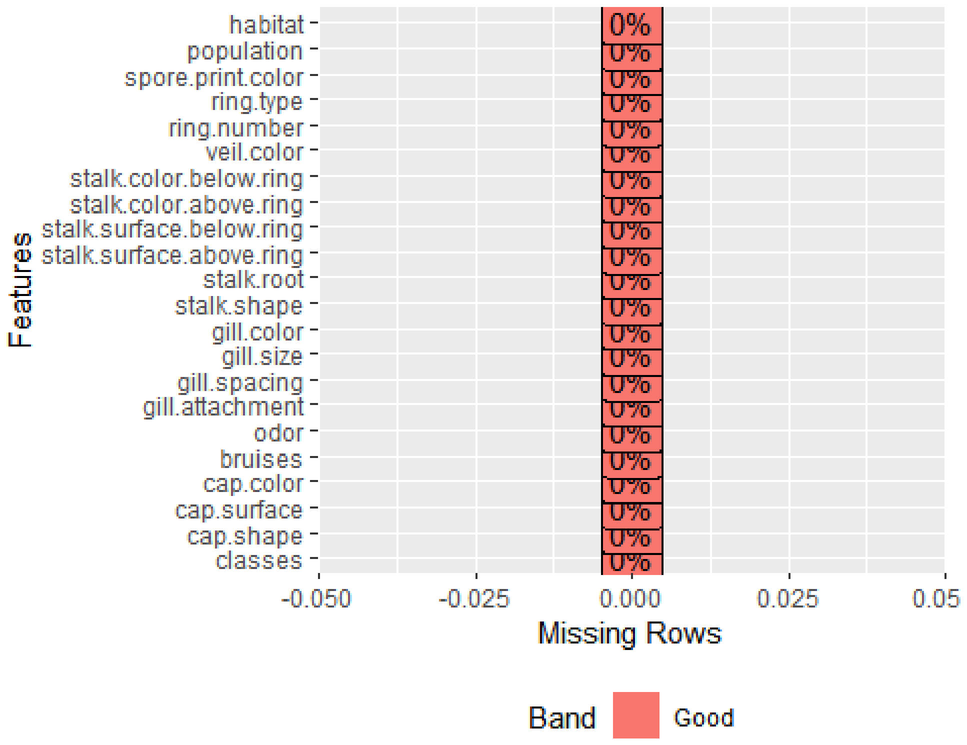


```r
sum(is.na(mushroom_df))

## [1] 2480

plot_missing(mushroom_df)
```



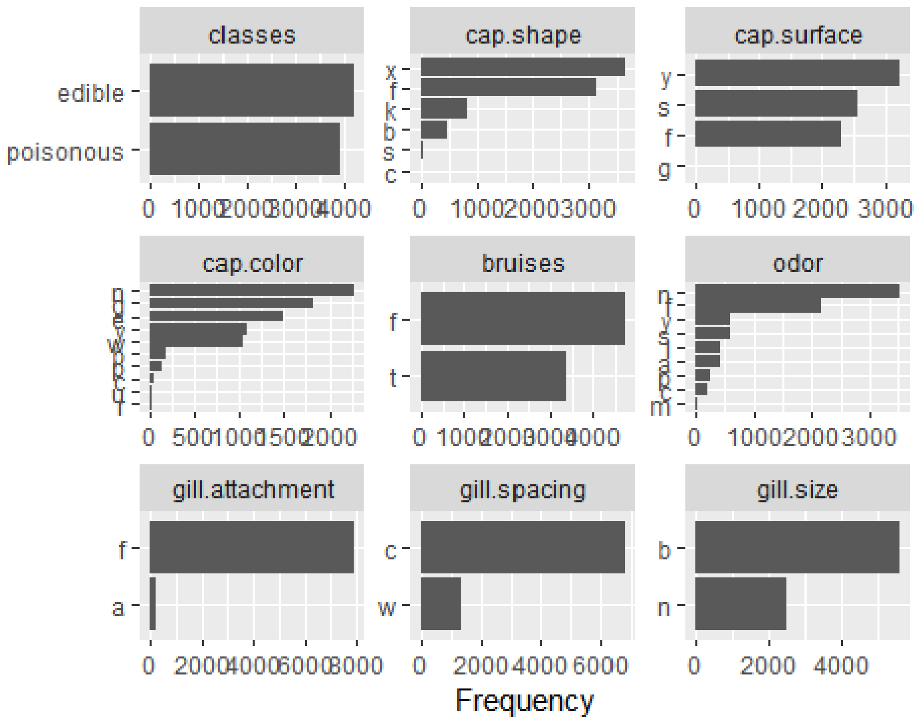
From our outputs above, we can see that all of the missing values are in our **stalk.root** column. We will replace these values with a **"m"** value, which is short for missing, and will act as another level (How do I replace NA values with zeros in an R dataframe?, n.d.).

```
mushroom_df$stalk.root[is.na(mushroom_df$stalk.root)] <- "m"
sum(is.na(mushroom_df))

## [1] 0

plot_missing(mushroom_df)
```



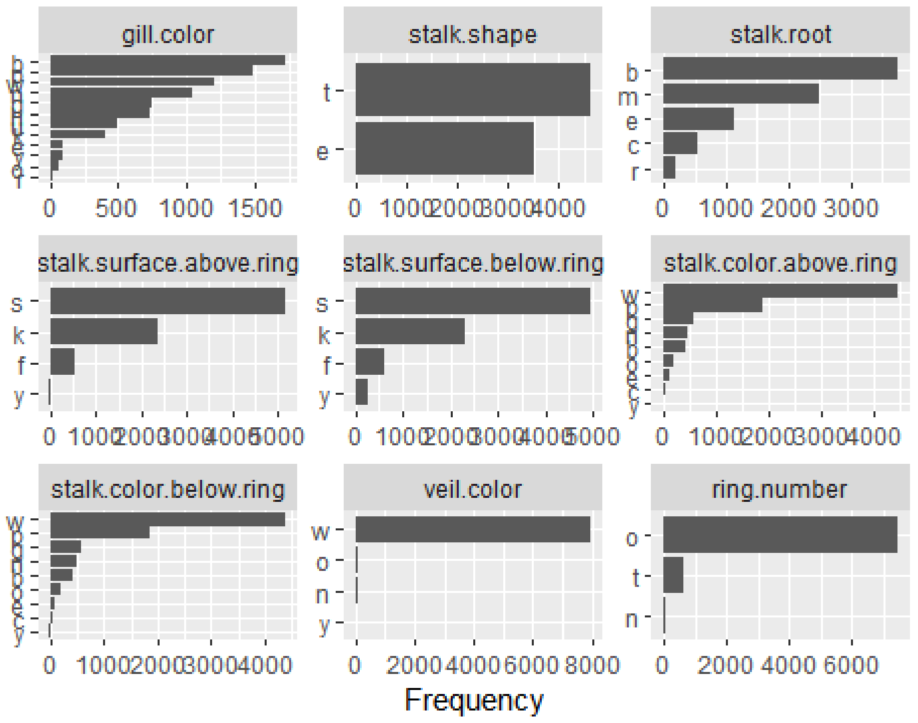
Now, we will view the distributions of our columns.

```r
plot_bar(mushroom_df)
```



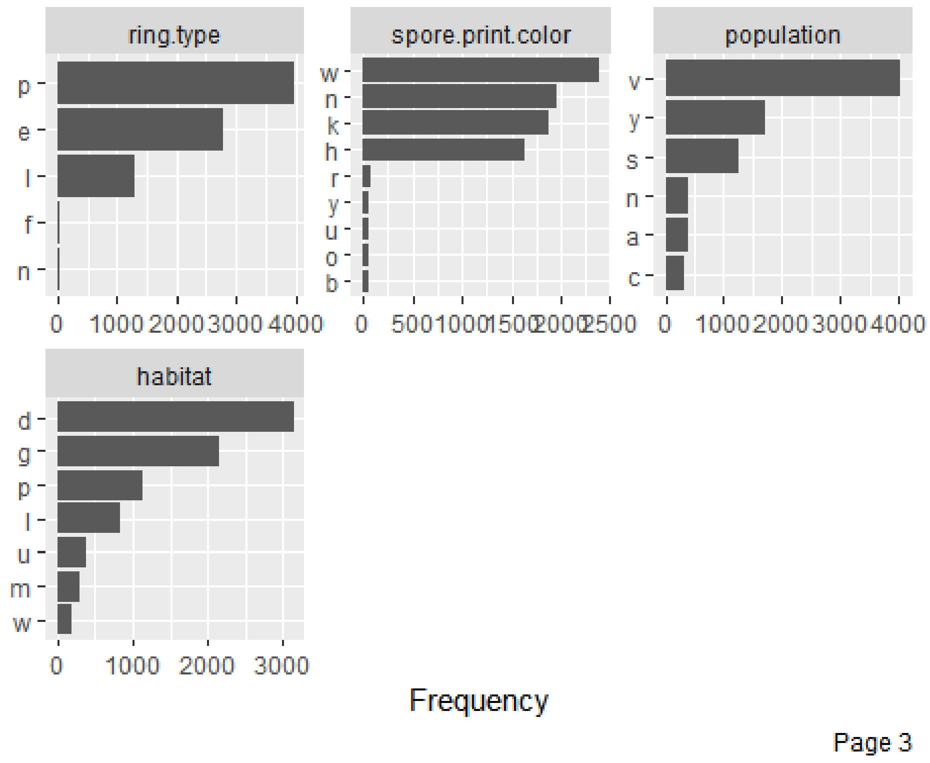
classes
edible
poisonous
cap.shape
cap.surface
cap.color
bruises
odor
gill.attachment
gill.spacing
gill.size
Frequency

Page 1



gill.color
stalk.shape
stalk.root
stalk.surface.above.ring
stalk.surface.below.ring
stalk.color.above.ring
stalk.color.below.ring
veil.color
ring.number
Frequency

Page 2



Support Vector Machines (SVM)

Support Vector Machines (SVM) "can be imagined as a surface that creates a boundary between points of data plotted in multidimensional that represent examples and their feature values (Lantz, 2015)." The goal of a SVM is to create a hyperplane, which is a flat boundary that divides the space to create homogeneous partitions on either side. In this way, the SVM learning combines aspects from the nearest neighbor learning method and the linear regression modeling (Lantz, 2015). SVMs are mostly used for binary classification, which is the method we will be using.

As mentioned above, SVMs use hyperplanes to partition data. If the data can be separated perfectly by a straight line or flat surface, they are said to be *linearly separable* (Lantz, 2015). If the data cannot be separated linearly (*nonlinearly separable*), the solution is to create a soft margin which allows some points to fall on the incorrect side of the margin (Lantz, 2015). This is know as the *slack variable*.

In many real-world applications, the relationships between variables are nonlinear. A SVM can still be trained on nonlinearly separable data with use of a slack variable, but this allows for misclassified data. A key feature of SVMs is that they have the ability to map the problem into a higher dimension space using a process called the *kernel trick* (Lantz, 2015). Using the kernel trick, we can view nonlinear data in a higher dimension and the relationship will appear linear. Kernel functions have been developed to map the data into another space. The general kernel function applies some transformation to the feature vectors x_i and x_j and combines them using the dot product (Lantz, 2015):

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j).$$

The most commonly used kernel functions include (Lantz, 2015):

1. Linear Kernel: Does not transform the data at all.
2. Polynomial Kernel: Adds a simple nonlinear transformation of the data of some degree d;
$$K(x_i, x_j) = (x_i \cdot x_j + 1)^d$$
3. Sigmoid Kernel: The greek letters kappa and delta are used as kernel parameters. This kernel results in a SVM model somewhat analogous to a neural network using a sigmoid activation function;
$$K(x_i, x_j) = tanh(\kappa x_i \cdot x_j - \delta)^d$$
4. Guassian RBF Kernel: Similar to a RBF Neural network. The RBF kernel performs well and is used as a starting point for many learning tasks;
$$K(x_i, x_j) = e\frac{-||x_i - x_j||^2}{2\sigma^2}$$

There is typically no reliable rule to match a kernel to a learning task, just that the fit depends heavily on the concept to be learned as well as the amount of training data and the relationship between variables (Lantz, 2015). The choice of the kernel is arbitrary, since performance varies slightly (Lantz, 2015). Thus, we will experiment with different kernels below.

Create Training/Testing Dataset and Labels

Before we build our SVM classifiers, we need to create our training and testing datasets. We will use the **createPartition()** function in the **caret** library.

```
set.seed(789)
index <- createDataPartition(mushroom_df$classes, p =0.7, list = FALSE)
mushroomTrain <- mushroom_df[index,] #index for training set
train_labels <- mushroom_df[1][index,]
dim(mushroomTrain)

## [1] 5688   22

mushroomTest <- mushroom_df[-index,] #-index for test set
test_labels <- mushroom_df[-index,1]
dim(mushroomTest)

## [1] 2436   22
```

Now that our data is ready, we can build our classifiers.

Linear Support Vector Machines

First, we will use the linear kernel. The linear kernal provides a baseline measure of SVM performance since this kernel does not map any values to higher dimensions (Lantz, 2015). To build the classifier, we will use the **svm()** function in the **e1071** package.

```
 # Linear SVM - e1071
svm_linear_model = svm(classes~., data = mushroomTrain, kernel = "linear",
scale = FALSE)
summary(svm_linear_model)

##
## Call:
## svm(formula = classes ~ ., data = mushroomTrain, kernel = "linear",
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  252
##
##  ( 141 111 )
##
##
## Number of Classes:  2
##
## Levels:
##  edible poisonous
```

To make predictions, we can use the **predict()** function and then to calculate the accuracy, we can create a confusion matrix.

```
linear_pred <- predict(svm_linear_model, mushroomTest)
confusionMatrix(linear_pred, test_labels)

## Confusion Matrix and Statistics
##
##            Reference
## Prediction  edible poisonous
##   edible      1262         0
##   poisonous      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 : edible
##
```

The diagonal values of 1262 and 1174 indicate the total number of records where the predicted class matched the true value. Notice there are no incorrect predictions, so our model has a 100% accuracy.

Now, we will use the linear kernel combined with **ksvm()** function, which is in the **kernlab** package. We will be using the **ksvm()** function to experiment with other kernels also.

```
# Linear SVM - kernlab
svm_linear_model_2  <- ksvm(classes ~ ., data = mushroomTrain, kernel =
"vanilladot")

##  Setting default kernel parameters

svm_linear_model_2

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 252
##
## Objective Function Value : -10.1822
## Training error : 0
```

The information above does not indicate how well the model will perform on our testing dataset. Thus, to examine the performance on the testing set, we will use the **predict()** function and create a confusion matrix.

```
linear_pred_2 <- predict(svm_linear_model_2, mushroomTest)
#table(linear_pred_2, test_labels)
confusionMatrix(linear_pred_2, test_labels)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  edible poisonous
##   edible      1262         0
##   poisonous      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 : edible
##
```

Similar to the results using the linear kernel with the **svm()** function, we see that our model correctly predicted all records in our training dataset.

Non-Linear Support Vector Machines

We can use more complex kernel functions like the Gaussian RBF, Polynomial, or Sigmoid kernel to map the data into a higher dimensional space and potentially obtain a better model fit (Lantz, 2015). First, we will use the Gaussian RBF kernel since this is the most popular kernel to begin with. To utilize the more complex kernels, we need to use the **ksvm()** function.

```
# Gaussian RBF (Radial) SVM - Kernlab
svm_rbf_model <- ksvm(classes ~ ., data = mushroomTrain, kernel = "rbfdot")
svm_rbf_model

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0708333333333333
```

```
##
## Number of Support Vectors : 591
##
## Objective Function Value : -59.4227
## Training error : 0
```

Similary to above, the summary of our model does not provide any insight to model predictions. Thus, we need to make predictions like we did above.

```
rbf_pred <- predict(svm_rbf_model, mushroomTest)
table(rbf_pred, test_labels)
```

```
##            test_labels
## rbf_pred    edible poisonous
##   edible      1262         0
##   poisonous      0      1174
```

```
confusionMatrix(rbf_pred, test_labels)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  edible poisonous
##   edible      1262         0
##   poisonous      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 : edible
##
```

Like the linear models above, our Gaussian RBF model has made 100% correct predictions. Now, we will use the Polynomial kernel. Using similar steps above,

```
# Polynomial SVM - Kernlab

svm_poly_model <- ksvm(classes ~ ., data = mushroomTrain, kernel = "polydot")

##  Setting default kernel parameters

#svm_poly_model
poly_pred <- predict(svm_poly_model, mushroomTest)
#table(poly_pred, test_labels)
confusionMatrix(poly_pred, test_labels)

## Confusion Matrix and Statistics
##
##            Reference
## Prediction  edible poisonous
##   edible      1262         0
##   poisonous      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 : edible
##
```

We have also obtained a 100% accuracy using the Polynomial kernel. Finally, we will use the Sigmoid Kernel.

```
  # Sigmoid SVM - Kernlab
svm_sigmoid_model <- ksvm(classes ~ ., data = mushroomTrain, kernel =
"tanhdot")

##  Setting default kernel parameters

svm_sigmoid_model
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Hyperbolic Tangent kernel function.
##  Hyperparameters : scale =  1  offset =  1
##
## Number of Support Vectors : 5384
##
## Objective Function Value : -5102.511
## Training error : 0.452707
```

```
sigmoid_pred <- predict(svm_sigmoid_model, mushroomTest)
#table(sigmoid_pred, test_labels)
confusionMatrix(sigmoid_pred, test_labels)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  edible poisonous
##   edible       211        23
##   poisonous   1051      1151
##
##                Accuracy : 0.5591
##                  95% CI : (0.5391, 0.579)
##     No Information Rate : 0.5181
##     P-Value [Acc > NIR] : 2.668e-05
##
##                   Kappa : 0.1432
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.16719
##             Specificity : 0.98041
##          Pos Pred Value : 0.90171
##          Neg Pred Value : 0.52271
##              Prevalence : 0.51806
##          Detection Rate : 0.08662
##    Detection Prevalence : 0.09606
##       Balanced Accuracy : 0.57380
##
##        'Positive' Class : edible
##
```

Using the Sigmoid kernel has caused us to go from a 100% accurate model to a 56% accurate model. We can see that the diagonal values of 211 and 1151 indicate the total number of records where the predicted class matched the true value and the values 1051 and 23 indicate the total numbers of records where the predicted class did not match the true values.

Artificial Neural Networks (ANN)

Now, we will build our Artificial Neural Network (ANN) classifier. An ANN models the relationship between a set of input signals and an output signal using a model derived from our understanding of how a brain responds to stimuli (Lantz, 2015). ANN uses a network of artificial neurons (typically several hundred), or nodes, to solve learning problems (Lantz, 2015). ANNs are best applied to problems where the input data and output data are well-defined or at least fairly simple (Lantz, 2015). Although there are several variants of neural networks, each can be defined in terms of the following characteristics:

1. An activation function, which transforms a neuron's combined input signals into a single output signal to be broadcasted further in the network.
2. A network topology (or architecture), which describes the number of neurons in the model as well as the number of layers and manner in which they are connected.
   - Number of layers: input and output nodes are arranged in groups
   - Number of nodes in each layer: input and output nodes are predetermined by the number of features and by the number of outcomes, respectively
3. The training algorithm that specifies how connection weights are set in order to inhibit or excite neurons in proportion to the input signal.

Training a neural network is done by using the backpropagation algorithm, which uses a strategy of back-propagating errors (Lantz, 2015). Over time, the network uses the information sent backward to reduce the total error of the network and determines how much a node's weight should be changed. This is done by using a technique called gradient descent. The back propagation uses the derivative of each neuron's activation function to identify the gradient in the direction of each of the incoming weights (Lantz, 2015). The gradient suggests how steeply the error will be reduced or increased from the change in the weight of each node (Lantz, 2015). The algorithm attempts to change the weights which result in the greatest reduction error by an amount, known as the learning rate (Lantz, 2015).

Convert Categorical Variables to Numeric Values

In order to utilize an ANN classifier, we first need to convert our categorical variables to numeric values by using one-hot-encoding or with the **dummyVars()** function. Below, we use the **dummyVars()** function to convert our variables to numeric values, excluding our outcome variable (**class**). Recall, the **dummyVars()** function breaks out unique values from a column into individual columns (Amunategui, n.d.). In other words, if you have 5 unique values in a column, dummying them will add 5 news columns to a dataset. To "dummify" our variables, we will first subset our **mushroom_df** dataframe to include only the variables we want to convert to dummy variables.

```
mushroom_ann_df <- subset(mushroom_df[2:22])
class <- subset(mushroom_df[1])
dmy <- dummyVars(~.,mushroom_ann_df, fullRank=T)
trsf <- data.frame(predict(dmy, newdata = mushroom_ann_df))
```

Now, we can combine our outcome variable with our dummy dataframe using the **cbind()** function.

```
#combine class and dummy variables
final_mushroom_ann <- cbind(trsf, class)
dim(final_mushroom_ann)

## [1] 8124   96
```

Notice, we have the same amount of rows in our **final_mushroom_ann** dataframe, but now we have 96 columns versus the 23 in our original dataframe.

Create Training and Testing Datasets

Now, we need to split our data into training and testing datasets. To do this, we will use the **sample()** function.

```
  #create train and test data sets
ind = sample(2, nrow(final_mushroom_ann),replace=TRUE,prob=c(0.7,0.3))
trainset = final_mushroom_ann[ind == 1,]
dim(trainset)

## [1] 5712   96

testset = final_mushroom_ann[ind==2,]
dim(testset)

## [1] 2412   96
```

Now, we can build our ANN classifier.

Build ANN Models

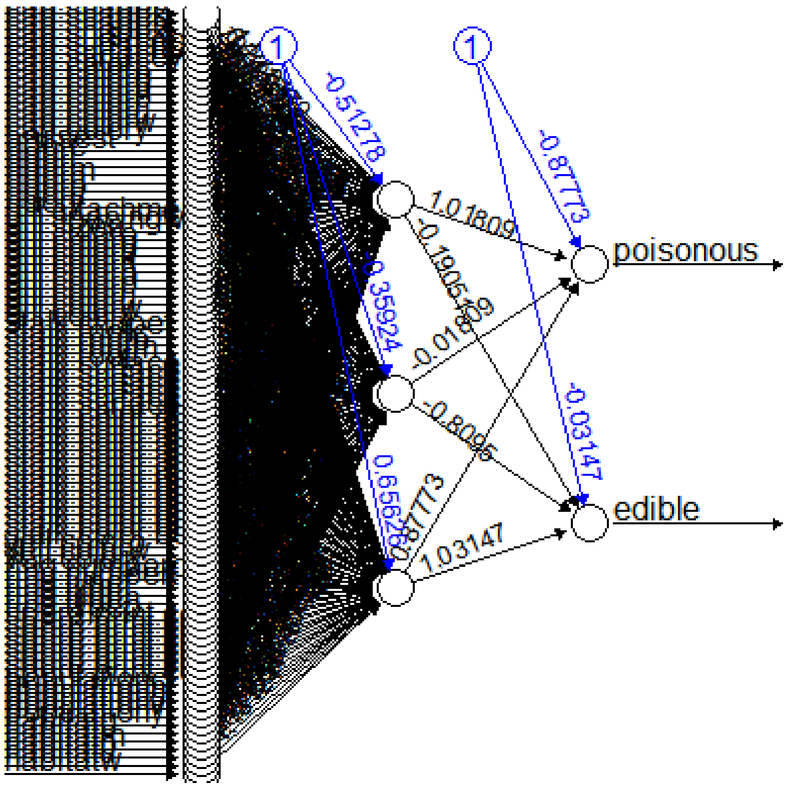
To avoid typing each variable one at a time in the **neuralnet()** function, we can get the columns from our training dataset, excluding the outcome variable, and then define the formula we want entered in the **neuralnet()** function. To do this, we use the **as.formula()** function, which allows us to create a formula from a string. Then, we use the **paste** function to concatenate the strings with a specified value.

```
  # Train model
train_columns <- colnames(trainset[-96])
nn_form <- as.formula(paste("classes~", paste(train_columns, collapse =
"+")))
nn_form

## classes ~ cap.shapec + cap.shapef + cap.shapek + cap.shapes +
##     cap.shapex + cap.surfaceg + cap.surfaces + cap.surfacey +
##     cap.colorc + cap.colore + cap.colorg + cap.colorn + cap.colorp +
##     cap.colorr + cap.coloru + cap.colorw + cap.colory + bruisest +
##     odorc + odorf + odorl + odorm + odorn + odorp + odors + odory +
##     gill.attachmentf + gill.spacingw + gill.sizen + gill.colore +
##     gill.colorg + gill.colorh + gill.colork + gill.colorn + gill.coloro +
##     gill.colorp + gill.colorr + gill.coloru + gill.colorw + gill.colory +
##     stalk.shapet + stalk.rootc + stalk.roote + stalk.rootm +
##     stalk.rootr + stalk.surface.above.ringk + stalk.surface.above.rings +
##     stalk.surface.above.ringy + stalk.surface.below.ringk +
stalk.surface.below.rings +
##     stalk.surface.below.ringy + stalk.color.above.ringc +
stalk.color.above.ringe +
##     stalk.color.above.ringg + stalk.color.above.ringn +
stalk.color.above.ringo +
##     stalk.color.above.ringp + stalk.color.above.ringw +
stalk.color.above.ringy +
##     stalk.color.below.ringc + stalk.color.below.ringe +
stalk.color.below.ringg +
##     stalk.color.below.ringn + stalk.color.below.ringo +
stalk.color.below.ringp +
##     stalk.color.below.ringw + stalk.color.below.ringy + veil.coloro +
##     veil.colorw + veil.colory + ring.numbero + ring.numbert +
##     ring.typef + ring.typel + ring.typen + ring.typep + spore.print.colorh
+
##     spore.print.colork + spore.print.colorn + spore.print.coloro +
##     spore.print.colorr + spore.print.coloru + spore.print.colorw +
##     spore.print.colory + populationc + populationn + populations +
##     populationv + populationy + habitatg + habitatl + habitatm +
##     habitatp + habitatu + habitatw
```

Now, we can train our model with the formula above. We will use the following arguments (Neuralnet function | r documentation, n.d.):

- hidden: Specifies the number of hidden neurons
- threshold: Specifies the threshold for the partial derivatives of the error function as stopping criteria
- learningrate: specifying the learning rate used by traditional backpropagation

```
nn_model <- neuralnet(nn_form, data = trainset, hidden = 3,
threshold=0.001,learningrate=0.02)
head(nn_model$result.matrix) #summary: short output

##                                   [,1]
## error                     5.350155e-09
## reached.threshold         8.740715e-04
## steps                     3.350000e+02
## Intercept.to.1layhid1    -5.127825e-01
## cap.shapec.to.1layhid1   -1.478179e+01
## cap.shapef.to.1layhid1    2.313586e-01

plot(nn_model, rep="best")
```



To evaluate the model performance, we will use the **compute()** function. This generates a prediction probability matrix based on a trained neural network and testing dataset (Yu-Wei, 2015). Then, we obtain other possible labels by finding the column with the greatest probability. Using the **table()** function, we can generate a classification table based on the predicted labels and labels of the testing set (Yu-Wei, 2015).

```
 #predictions
nn_pred = compute(nn_model, testset[-96])$net.result #remove "classes" column
nn_predicition = c("edible","poisonous")[apply(nn_pred, 1, which.max)]
#obtain other possible labels by finding the column with the greatest
probability
pred_table = table(testset$classes, nn_predicition)#generate classification
table
pred_table

##             nn_predicition
##              edible poisonous
##   edible       1247         0
##   poisonous       0      1165
```

Now, we can build a confusion matrix.

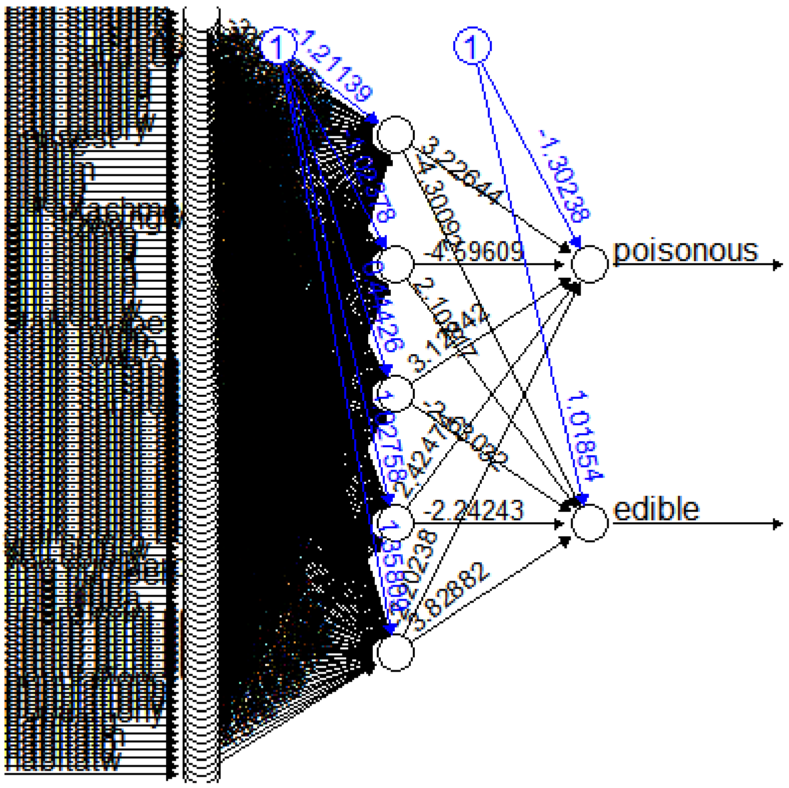
```
#classAgreement(pred_table)
confusionMatrix(pred_table)

## Confusion Matrix and Statistics
##
```

```
##            nn_predicition
##             edible poisonous
##   edible      1247         0
##   poisonous      0      1165
##
##                Accuracy : 1
##                  95% CI : (0.9985, 1)
##     No Information Rate : 0.517
##     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 : 0.517
##          Detection Rate : 0.517
##    Detection Prevalence : 0.517
##       Balanced Accuracy : 1.000
##
##        'Positive' Class : edible
##
```

Our confusion matrix above indicates that, using the parameters above, gives us a model accuracy of 100%. Now, let’s change a few parameters in our model. Instead of the parameters above, we will use:

- startweights: vector containing starting values for the weights. Set to NULL for random initialization
- hidden: Use 5 instead of 3
- err.fct = differentiable function that is used for the calculation of the error. Alternatively, the strings ‘sse’ and ‘ce’ which stand for the sum of squared errors and the cross-entropy can be used.
- act.fct: differentiable function that is used for smoothing the result of the cross product of the covariate or neurons and the weights. Additionally the strings, ‘logistic’ and ‘tanh’ are possible for the logistic function and tangent hyperbolicus.
- linear.output: If act.fct should not be applied to the output neurons set linear output to TRUE, otherwise to FALSE.

```
nn_model_2 <- neuralnet(nn_form , data = trainset , startweights = NULL,
hidden=5, err.fct="sse", act.fct="logistic", linear.output = FALSE)
plot(nn_model_2, rep="best")
```



```
nn_pred_2 = compute(nn_model_2, testset[-96])$net.result #remove "classes"
column
nn_predicition_2 = c("edible","poisonous")[apply(nn_pred_2, 1, which.max)]
pred_table_2 = table(testset$classes, nn_predicition_2)
confusionMatrix(pred_table)

## Confusion Matrix and Statistics
##
##              nn_predicition
##               edible poisonous
##   edible        1247         0
##   poisonous        0      1165
##
##                Accuracy : 1
##                  95% CI : (0.9985, 1)
##     No Information Rate : 0.517
##     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 : 0.517
##          Detection Rate : 0.517
##    Detection Prevalence : 0.517
##       Balanced Accuracy : 1.000
##
##        'Positive' Class : edible
##
```

Notice, with the parameters indicated above, there is no change to the model's accuracy.

Conclusion

For this assignment, we explored two machine learning methods: Support Vector Machines and Neural Networks. While both methods can be extremely useful, they are often overlooked due to their complexity. We applied SVM and ANN classifiers to our mushroom species dataset to determine whether a mushroom was poisonous or edible based on particular characteristics. During our exploration of the SVM classifier, we experimented with different kernels and found that the Sigmoid kernel cut the accuracy of our model in half. While the Gaussian RBF and Polynomial kernels returned a model that was 100% accurate, the Linear kernel would be the most appropriate for the dataset as there was no need to map the data into higher dimensions. For the ANN classifier, we experimented with different parameters and found no changes between the models. For the future, more drastic changes to parameters, and more parameters used, would result in changes to the model. These changes and use of parameters should be explored more. Comparing the two classifiers: the SVM classifier is much simpler to understand and seems to be easier to implement than the ANN classifier.

Resources

Amunategui, M. (n.d.). Data exploration & machine learning, hands-on. Retrieved September 27, 2020, from https://amunategui.github.io/dummyVar-Walkthrough/

How do I replace NA values with zeros in an R dataframe? (n.d.). Stack Overflow. Retrieved September 27, 2020, from https://stackoverflow.com/questions/8161836/how-do-i-replace-na-values-with-zeros-in-an-r-dataframe

Lantz, Brett. Machine Learning with R : Expert Techniques for Predictive Modeling to Solve All Your Data Analysis Problems. Vol. Second edition, Packt Publishing, 2015.

Neuralnet function | r documentation. (n.d.). Retrieved September 28, 2020, from https://www.rdocumentation.org/packages/neuralnet/versions/1.44.2/topics/neuralnet

R - replace all 0 values to na. (n.d.). Stack Overflow. Retrieved September 27, 2020, from https://stackoverflow.com/questions/11036989/replace-all-0-values-to-na

Schlimmer, J. (n.d.). Uci machine learning repository: Mushroom data set. Retrieved September 28, 2020, from http://archive.ics.uci.edu/ml/datasets/Mushroom

Yu-Wei, Chiu. Machine Learning with R Cookbook. Packt Publishing, 2015.