Mushroom Classification

Support Vector Machines and Artificial Neural Nets

Sean O'Malley

Objective : Predict class of mushroom given characteristic metrics about the mushrooms.

Methodology 1 : Support Vector Machine

Methodology 2 : Artificial Neural Net

Ingest and Explore Data

```
shroom_names <- c("class","cap-shape","cap-surface","cap-color","bruises","odor", "grill-attachmen
t",  "grill-spacing","grill-size", "grill-color", "stalk-shape", "stalk-root", "stalk-surface-abov
e-ring", "stalk-surface-below-ring", "stalk-color-above-ring", "stalk-color-below-ring", "veil-typ
e", "veil-color", "ring-number", "ring-type", "spore-print-color", "population", "habitat")

shroom <- read.delim("http://archive.ics.uci.edu/ml/machine-learning-databases/mushroom/agaricus-l
epiota.data",
                   sep = ",",
                   header = F,
                   col.names = shroom_names,
                   na.strings = "NA")

# Remove NA's and Columns with only 1 factor
shroom <- na.omit(shroom)
shroom <- shroom %>% select(-veil.type)

summary(shroom)
```

```
##  class    cap.shape cap.surface   cap.color    bruises       odor
##  e:4208   b: 452    f:2320      n      :2284   f:4748   n      :3528
##  p:3916   c:   4    g:   4      g      :1840   t:3376   f      :2160
##           f:3152    s:2556      e      :1500            s      : 576
##           k: 828    y:3244      y      :1072            y      : 576
##           s:  32                w      :1040            a      : 400
##           x:3656                b      : 168            l      : 400
##                                 (Other): 220            (Other): 484
##  grill.attachment grill.spacing grill.size  grill.color   stalk.shape
##  a: 210           c:6812        b:5612     b      :1728   e:3516
##  f:7914           w:1312        n:2512     p      :1492   t:4608
##                                            w      :1202
##                                            n      :1048
##                                            g      : 752
##                                            h      : 732
##                                            (Other):1170
##  stalk.root stalk.surface.above.ring stalk.surface.below.ring
##  ?:2480     f: 552                   f: 600
##  b:3776     k:2372                   k:2304
##  c: 556     s:5176                   s:4936
##  e:1120     y:  24                   y: 284
##  r: 192
##
##
##  stalk.color.above.ring stalk.color.below.ring veil.color ring.number
##  w      :4464           w      :4384           n:  96     n:  36
##  p      :1872           p      :1872           o:  96     o:7488
##  g      : 576           g      : 576           w:7924     t: 600
##  n      : 448           n      : 512           y:   8
##  b      : 432           b      : 432
##  o      : 192           o      : 192
##  (Other): 140           (Other): 156
##  ring.type spore.print.color population habitat
##  e:2776    w      :2388      a: 384     d:3148
##  f:  48    n      :1968      c: 340     g:2148
##  l:1296    k      :1872      n: 400     l: 832
##  n:  36    h      :1632      s:1248     m: 292
##  p:3968    r      :  72      v:4040     p:1144
##            b      :  48      y:1712     u: 368
##            (Other): 144                 w: 192
```

Create Test and Training Datasets

To do so I created a function that splits the data into a 70/30 split and sets a seed. I then ran the wine dataset through my function.

```r
split_data <- function(x, p = 0.7, s = 777) {
  set.seed(s)
  index = sample(1:dim(x)[1])
  train = x[index[1:floor(dim(x)[1]*p)],]
  test = x[index[((ceiling(dim(x)[1]*p))+1):dim(x)[1]],]
  return(list(train = train, test = test))
}

shroom_split <- split_data(shroom)

shroom_test <- shroom_split$test

shroom_train <- shroom_split$train
```

Support Vector Machine

Support vector machines can be defined as a surface that defines a boundary between various points of data which represent examples plotted in multidimensional space according to their feature values. The goal of the SVM is to create a flat boundary, called a hyperplane, which leads to fairly homogeneous partitions of data on either side. In this way, SVM learning combines aspects of both the instance-based nearest neighbor learning, and linear regression modeling. The combination is extremely powerful, allowing SVM's to model highly complex relationships.

SVMs can be adapted for use with nearly any type of learning task, including both classification and numeric prediction. Many of the algorithm's key successes have come in pattern recognition. The task of the SVM algorithm is to identify a line that separates the two classes. The SVM searches for the Maximum Margin Hyperplane, that creates the greates separation between two classes. Doing so makes it more likely that the future data is properly classified with some level of flexibility.

The support vectors are points from each class that are the closest to the MMH; each class must have at least one support vector, but it is possible to have more than one. Support vectors provide a compact way to store a classification model, even if the number of features is extremely large.

Radial Basis SVM using e1071 Package

I am building a model here that uses radial basis nonlinear mapping with a cost of 1, representing the cost of violating constraints (the more large the values the narrower the result).

Build Model

```r
svm1 <- svm(class~., data = shroom_train, kernel = "radial", cost = 1, gamma =
1/ncol(shroom_train))
```

Use to Predict Results

```r
predict1 <- predict(svm1, shroom_test, type = "response")
```

View Predicted Results

```r
confusionMatrix(predict1, shroom_test$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    e    p
##          e 1249    0
##          p    0 1188
##
##                Accuracy : 1
##                  95% CI : (0.9985, 1)
##     No Information Rate : 0.5125
##     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 : 0.5125
##          Detection Rate : 0.5125
##    Detection Prevalence : 0.5125
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : e
##
```

I am showing absolute perfection with the confusion matrix, which is mildly confusing. The p value also impossibly good, but being that I used the test set to train set to test the train set and am still getting this level of accuracy does not suggest overfitting.

This makes sense based on how the data works well with a radial basis function (RBF), which is a real-valued function whose value depends only on the distance from the origin. So if the classes are of significant distance from eachother, the class radials will more easily pick up all the classified datapoints.

Hyperbolic Tangent Sigmoid SVM using kernal Package

I am building a model here that uses hyperbolic tangentsigmoid kernal mapping with a cost of 1, representing the cost of violating constraints (the more large the values the narrower the result).

Build Model

```
svm2 <- ksvm(class~., data = shroom_train, kernel = "tanhdot", C = 1)
```

```
##  Setting default kernel parameters
```

Use to Predict Results

```
predict2 <- predict(svm2, shroom_test, type = "response")
```

View Predicted Results

```
confusionMatrix(predict2, shroom_test$class)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    e    p
##          e  152    1
##          p 1097 1187
##                                          
##                Accuracy : 0.5494          
##                  95% CI : (0.5294, 0.5693)
##     No Information Rate : 0.5125          
##     P-Value [Acc > NIR] : 0.0001413       
##                                          
##                   Kappa : 0.1182          
##  Mcnemar's Test P-Value : < 2.2e-16       
##                                          
##             Sensitivity : 0.12170         
##             Specificity : 0.99916         
##          Pos Pred Value : 0.99346         
##          Neg Pred Value : 0.51970         
##              Prevalence : 0.51252         
##          Detection Rate : 0.06237         
##    Detection Prevalence : 0.06278         
##       Balanced Accuracy : 0.56043         
##                                          
##        'Positive' Class : e               
## 
```

The model is understandably worse than the radial model given how the sigmoid function works. If the multidimensionality does not fall into the the sigmoid function's characteristic "S"-shaped curve or sigmoid curve, then the classification would be wrong.

Linear SVM using kernal Package

I am building a model here that uses linear kernal mapping with a cost of 1, representing the cost of violating constraints (the more large the values the narrower the result).

Build Model

```
svm3 <- ksvm(class~., data = shroom_train, kernel = "vanilladot", C = 1)
```

```
##  Setting default kernel parameters
```

Use to Predict Results

```
predict3 <- predict(svm3, shroom_test, type = "response")
```

View Predicted Results

```
confusionMatrix(predict3, shroom_test$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    e    p
##          e 1249    0
##          p    0 1188
##
##                Accuracy : 1
##                  95% CI : (0.9985, 1)
##     No Information Rate : 0.5125
##     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 : 0.5125
##          Detection Rate : 0.5125
##    Detection Prevalence : 0.5125
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : e
##
```

Once again we have perfect classification of the test set using the training dataset. Given the linear mapping an a binary classification, this makes sense. However, I have yet to become comfortable with having perfect classification, guess I'll have to get used to it for this exercise.

Subset SVM Features and Re-Run Models

I would like to systematically approach the reduction of features in our model via methods in feature importance, and from there culling the data down. Next, we can put this new data through the same models from above, predict, and interpret our results.

What is unique about reducing features is that SVMs have built-in regularization because they tend to penalize large weights of predictors which amounts to favor simpler models. They're often used with recursive feature elimination

Feature Importance

```
library(rpart)

shroom_rp <- rpart(class~., data = shroom_train)

variable_importance <- data.frame(shroom_rp$variable.importance)

print(variable_importance)
```

```
##                            shroom_rp.variable.importance
## odor                                            2659.898
## spore.print.color                               1969.177
## grill.color                                     1600.439
## stalk.surface.below.ring                        1437.919
## stalk.surface.above.ring                        1419.756
## ring.type                                       1387.466
```

Reduce Features

Based on the feature importance data frame above, lets reduce our independent variable feature list from 20 down to 6, then use our 3 SVM's to predict again and see what we get.

```
trimmed_shroom <- shroom %>% select(class, odor, spore.print.color, grill.color, stalk.surface.bel
ow.ring, stalk.surface.above.ring, ring.type)
```

Partition into Test and Training

```
trimmed_shroom_split <- split_data(trimmed_shroom)

trimmed_shroom_test <- trimmed_shroom_split$test

trimmed_shroom_train <- trimmed_shroom_split$train
```

Radial Basis SVM using e1071 Package

Using the trimmed_shroom training data, I am building a model that uses radial basis nonlinear mapping with a cost of 1, representing the cost of violating constraints (the more large the values the narrower the result).

Build Model

```
svm1t <- svm(class~., data = trimmed_shroom_train,
            kernel = "radial", cost = 1,
            gamma = 1/ncol(trimmed_shroom_train))
```

Use to Predict Results

```
predict1t <- predict(svm1t, trimmed_shroom_test, type = "response")
```

View Predicted Results

```
confusionMatrix(predict1t, trimmed_shroom_test$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    e    p
##          e 1249    2
##          p    0 1186
##
##                Accuracy : 0.9992
##                  95% CI : (0.997, 0.9999)
##     No Information Rate : 0.5125
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9984
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 1.0000
##             Specificity : 0.9983
##          Pos Pred Value : 0.9984
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5125
##          Detection Rate : 0.5125
##    Detection Prevalence : 0.5133
##       Balanced Accuracy : 0.9992
##
##        'Positive' Class : e
##
```

Pretty amazing, using only about 1/3 the features we see are still able to predict with 98% accuracy whether a mushroom is poisenous or not. Even so, we our only false prediction is a false positive, which means not eating the mushroom vs. getting sick (the preferrable misprediction if I may say so myself). We see an equally fantastic p value and only a slight drop in specificity from our previous model, understandably so since we took away most of the features.

Hyperbolic Tangent Sigmoid SVM using kernal Package

Using the trimmed_shroom training data, I am going to again use the hyperbolic tangentsigmoid kernal mapping with a cost of 1, representing the cost of violating constraints.

Build Model

```
svm2t <- ksvm(class~., data = trimmed_shroom_train, kernel = "tanhdot", C = 1)
```

```
##  Setting default kernel parameters
```

Use to Predict Results

```
predict2t <- predict(svm2t, trimmed_shroom_test, type = "response")
```

View Predicted Results

```
confusionMatrix(predict2t, trimmed_shroom_test$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   e   p
##          e 979 205
##          p 270 983
##
##                Accuracy : 0.8051
##                  95% CI : (0.7888, 0.8206)
##     No Information Rate : 0.5125
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6105
##  Mcnemar's Test P-Value : 0.003319
##
##             Sensitivity : 0.7838
##             Specificity : 0.8274
##          Pos Pred Value : 0.8269
##          Neg Pred Value : 0.7845
##              Prevalence : 0.5125
##          Detection Rate : 0.4017
##    Detection Prevalence : 0.4858
##       Balanced Accuracy : 0.8056
##
##        'Positive' Class : e
##
```

Super interesting results, our hyperbolic tangent sigmoid SVM is actually 30% more accurate by reducing the features from 20 to 6. This would suggest that this methodology of multidimensional mapping, at least in this case, is prone to overfitting. Let me remind you that if the multidimensionality does not fall into the the sigmoid function's characteristic "S"-shaped curve or sigmoid curve, then the classification will be less accurate.

This model is still less accurate than the linear or radial SVM's but the improvement highlights a fascinating reality about hyperbolic tangent sigmoids.

Linear SVM using kernal Package

Using the trimmed_shroom training data, I am building a model here that uses linear kernal mapping with a cost of 1, representing the cost of violating constraints.

Build Model

```
svm3t <- ksvm(class~., data = trimmed_shroom_train, kernel = "vanilladot", C = 1)
```

```
##  Setting default kernel parameters
```

Use to Predict Results

```
predict3t <- predict(svm3t, trimmed_shroom_test, type = "response")
```

View Predicted Results

```
confusionMatrix(predict3t, trimmed_shroom_test$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    e    p
##          e 1249    2
##          p    0 1186
##
##                Accuracy : 0.9992
##                  95% CI : (0.997, 0.9999)
##     No Information Rate : 0.5125
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9984
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 1.0000
##             Specificity : 0.9983
##          Pos Pred Value : 0.9984
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5125
##          Detection Rate : 0.5125
##    Detection Prevalence : 0.5133
##       Balanced Accuracy : 0.9992
##
##        'Positive' Class : e
##
```

We see amazing accuracy once again with the linearly mapped multidimensional SVM. Interestingly enough, it has the same accuracy as the radially mapped model. Suggesting once again, that even with reduced features, some significant space exists between the classes in the SVM's multidimensional space.

Artificial Neural Net

Artificial neural networks are the relationships between a set of input signals and an output signal using a model derived from our understanding of how a biological brain responds to stimuli from sensory inputs. ANN's use a network of artificial neurons or nodes to solve learning problems this way. Typically ANN's usually only require several hundred artificial neurons.

When understanding the structure of a neuron, know that an ANN node is a relationship between the input signals received by the (x) variable and the output signal, (y) variable. Each signal is weighted (w) according to its importance. The input signals are summed by the cell body and the signal is passed according to an activation function noted by (f).

To model the relationship between the class (poisonous or not), we will use a multilayer feedforward neural network. It offers a function to plot the network topology.

Data Manipulation and Preparation for ANN

The datatype needs to be numeric to be run through the dataframe. Per the simplicity of the factor data at hand I converted all the values to integers, then re-ran through my split function, and more easily organized / concatenated the variable names to insert into the model becaues the neuralnet function is a little touchy on variable input.

```
# Convert to Numeric for Model ANN

shroom <- as.data.frame(mapply(as.integer, shroom))

# Run initial Split of data for ANN

shroom_split <- split_data(shroom)

shroom_test <- shroom_split$test

shroom_train <- shroom_split$train

# Prepare for ease in Model Usage

shroom_names <- colnames(shroom)
shroom_predictors <- shroom_names[!shroom_names%in%"class"]
shroom_predictors <- paste(shroom_predictors,collapse = "+")
shroom_form <- as.formula(paste("class~", shroom_predictors, collapse = "+"))
```

Simple ANN with 1 hidden neuron Using nerualnet Package

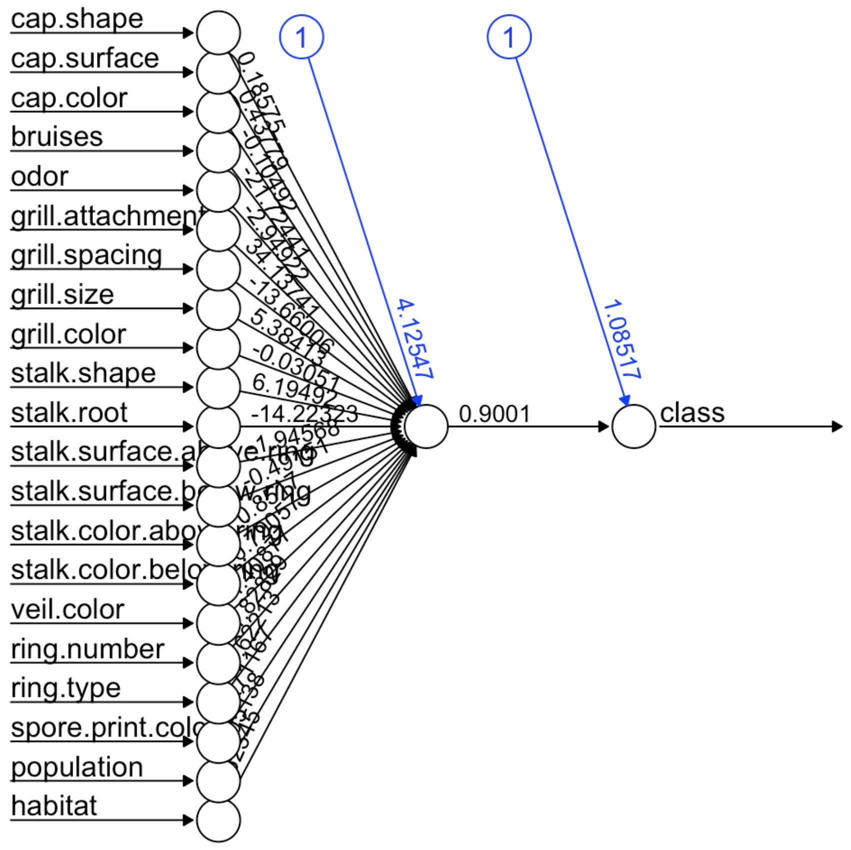
Build Model

```
shroom_ann1 <-  neuralnet(formula = shroom_form, hidden = 1, data = shroom_train)
```

View Model Output

You can se that in this model, there is our one input node for each of the eight features, followed by a single hidden node and a single output node that predicts whether a mushroom is poisonous or not. The weights of the conenctions are also depicted, the bias terms, and the sum of squared errors.

```
plot(shroom_ann1, rep = "best")
```



Predict With Model

The compute function returns a list with neurons and net results, which stores the predicted values.

```
#  Compute Function

ann_predict1 <- compute(shroom_ann1, shroom_test[2:22])

# Predicted Strength

ann_predict_strength1 <- ann_predict1$net.result
```

View Predicted Results

Below you'll see that we I take the predicted results and actual results after which I round the prediction results. To test the predictive power you can use the correllation number, but because we are predicting a binary indicator, the confusion matrix will work best.

```
results1 <- data.frame(actual = shroom_test$class, prediction = ann_predict1$net.result)

results1$prediction <- round(results1$prediction)

# Now test prediction strength with a correllation between the actual and predicted values

cor(results1$prediction, results1$actual)
```

```
## [1] 0.9049696203
```

```
confusionMatrix(results1$prediction, results1$actual)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    2
##          1 1238  108
##          2   11 1080
##
##                Accuracy : 0.9511695
##                  95% CI : (0.94185, 0.9593843)
##     No Information Rate : 0.5125154
##     P-Value [Acc > NIR] : < 0.00000000000000022204
##
##                   Kappa : 0.9020825
##  Mcnemar's Test P-Value : < 0.00000000000000022204
##
##             Sensitivity : 0.9911930
##             Specificity : 0.9090909
##          Pos Pred Value : 0.9197623
##          Neg Pred Value : 0.9899175
##              Prevalence : 0.5125154
##          Detection Rate : 0.5080016
##    Detection Prevalence : 0.5523184
##       Balanced Accuracy : 0.9501419
##
##        'Positive' Class : 1
##
```

Interpret Model Results

Looking in totality at our most simple ANN model, we see that we do have some significant predicting power. With only one node and one hidden node, we were able to predict with 95% accuracy whether a mushroom was poisonous or not. Once again, similar to our SVM's, we see a great p value and that most of our false predictions are false positives, which is okay because the reciprocations aren't quite as bad as a false negative (eating a poisonous mushroom).

Linear ANN with 2 hidden layers of 4 and 2 neurons using nerualnet Package

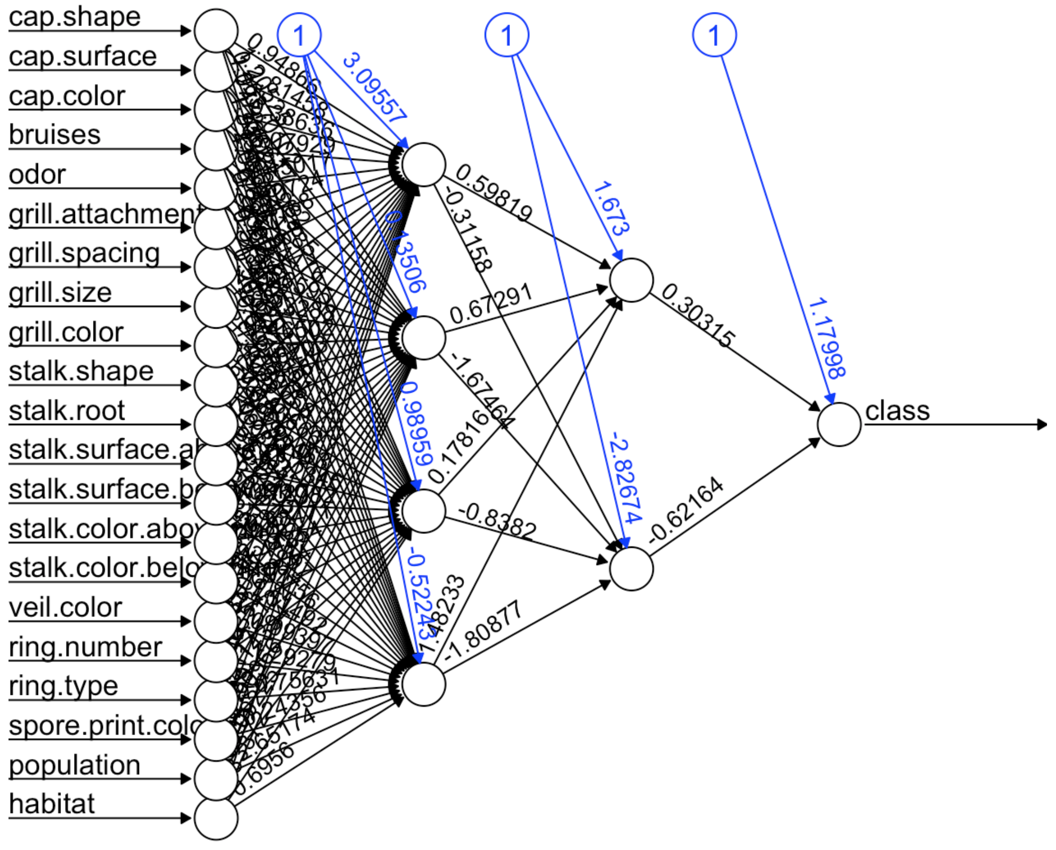
Build Model

```
shroom_ann2 <- neuralnet(formula = shroom_form, hidden = c(4,2), linear.output = T, data = shroom_
train)
```

View Model Output

Wow, we have a much different look in our plotted output to our previous model. We see our hidden layers of 4 and 2, the SSE's as you move from right to left, all to determine the class of the mushroom as poisonous or not. Lets see if this complexity improves out model performance!

```
plot(shroom_ann2, rep = "best")
```



Predict With Model

The compute function returns a list with neurons and net results, which stores the predicted values.

```
#  Compute Function

ann_predict2 <- compute(shroom_ann2, shroom_test[2:22])

# Predicted Strength

ann_predict_strength2 <- ann_predict2$net.result
```

View Predicted Results

Below you'll see that we I take the predicted results and actual results after which I round the prediction results. To test the predictive power you can use the correllation number, but because we are predicting a binary indicator, the confusion matrix will work best.

```
results2 <- data.frame(actual = shroom_test$class, prediction = ann_predict2$net.result)

results2$prediction <- round(results2$prediction)

# Now test prediction strength with a correllation between the actual and predicted values

cor(results2$prediction, results2$actual)
```

```
## Warning in cor(results2$prediction, results2$actual): the standard
## deviation is zero
```

```
## [1] NA
```

```
confusionMatrix(results2$prediction, results2$actual)
```

```
## Warning in confusionMatrix.default(results2$prediction, results2$actual):
## Levels are not in the same order for reference and data. Refactoring data
## to match.
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    2
##          1 1249 1188
##          2    0    0
##
##                Accuracy : 0.5125154
##                  95% CI : (0.4924601, 0.5325406)
##     No Information Rate : 0.5125154
##     P-Value [Acc > NIR] : 0.5081504
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : < 0.00000000000000022
##
##             Sensitivity : 1.0000000
##             Specificity : 0.0000000
##          Pos Pred Value : 0.5125154
##          Neg Pred Value :       NaN
##              Prevalence : 0.5125154
##          Detection Rate : 0.5125154
##    Detection Prevalence : 1.0000000
##       Balanced Accuracy : 0.5000000
##
##        'Positive' Class : 1
##
```

Interpret Model Results

In our linear ANN model with 4 hidden layers of two nodes, we see a slight improvement upon our previous model in predictive power. We were able to predict with 95.5% accuracy whether a mushroom was poisonous or not. We also see another great p value, and an improvement in specificity, while maintaining solid sensitivity. Not bad!

ANN with resilient backpropagation without backtracking with 5 neurons using nerualnet Package

Resilient backpropagation is often faster than training with back propagation alone. Also, it doesn't require you to specify any free parameter values, as opposed to back propagation which needs values for the learning rate (and usually an optional momentum term). The main disadvantage of resilient backpropogation is that it's a more complex algorithm to implement than back propagation.

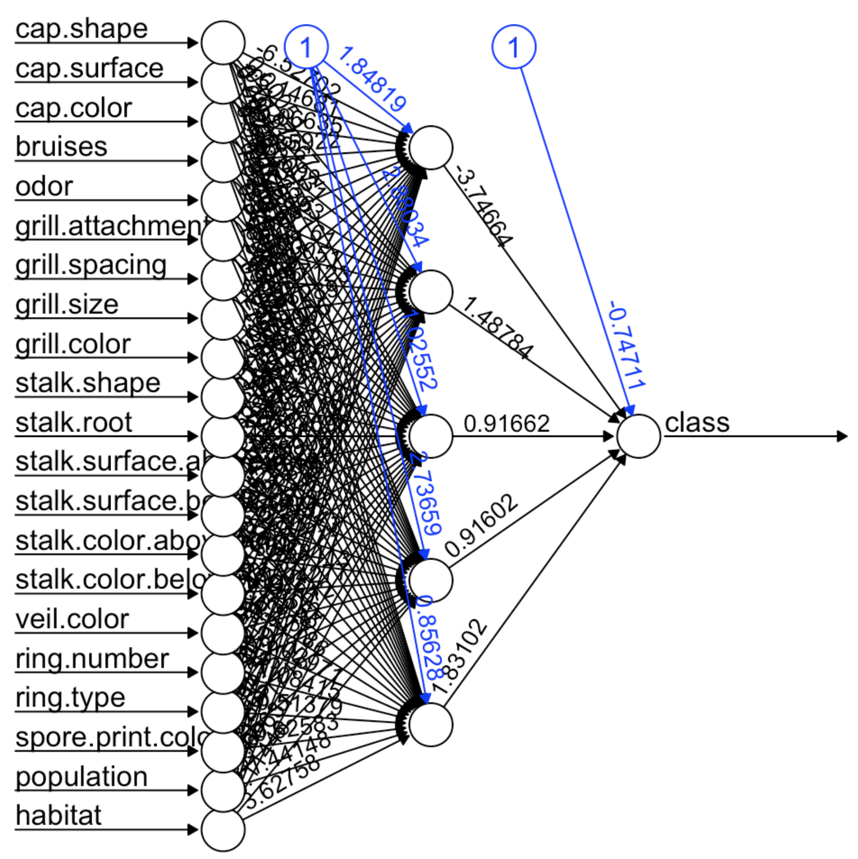
Build Model

```
shroom_ann3 <- neuralnet(formula = shroom_form, hidden = 5, algorithm = "rprop-", data = shroom_tr
ain)
```

View Model Output

We see our hidden layers of 5, their weights, and the SSE's as you move from right to left, all to determine the class of the mushroom as poisonous or not. Lets see if this alteration of complexity improves out model performance!

```
plot(shroom_ann3, rep = "best")
```



Predict With Model

The compute function returns a list with neurons and net results, which stores the predicted values.

```
#  Compute Function

ann_predict3 <- compute(shroom_ann3, shroom_test[2:22])

# Predicted Strength

ann_predict_strength3 <- ann_predict3$net.result
```

View Predicted Results

Below you'll see that we I take the predicted results and actual results after which I round the prediction results. To test the predictive power you can use the correllation number, but because we are predicting a binary indicator, the confusion matrix will work best.

```
results3 <- data.frame(actual = shroom_test$class, prediction = ann_predict3$net.result)

results3$prediction <- round(results3$prediction)

# Now test prediction strength with a correllation between the actual and predicted values

confusionMatrix(results3$prediction, results3$actual)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    2
##          1 1249  105
##          2    0 1083
##
##                Accuracy : 0.9569142
##                  95% CI : (0.9480804, 0.9646273)
##     No Information Rate : 0.5125154
##     P-Value [Acc > NIR] : < 0.000000000000000222204
##
##                   Kappa : 0.913588
##  Mcnemar's Test P-Value : < 0.000000000000000222204
##
##             Sensitivity : 1.0000000
##             Specificity : 0.9116162
##          Pos Pred Value : 0.9224520
##          Neg Pred Value : 1.0000000
##              Prevalence : 0.5125154
##          Detection Rate : 0.5125154
##    Detection Prevalence : 0.5556011
##       Balanced Accuracy : 0.9558081
##
##        'Positive' Class : 1
##
```

Interpret Model Results

Holy Smokes! Perfection!! As we see, our most complex ANN built yet, has the best predictive power. Essentially perfect metrics down throughout the confusion matrix. This really is due to the more complex weighting of the resilient backpropogation that occurs.