Derek Hughes
Assignment #4
Predict 454 – Sec 55
Winter 2016

Introduction

Wine is considered a delicacy, a way of life, and a sign of sophistication for some people. Many people go to wine tastings and consider being connoisseurs of wine, claiming their palette is a master of identifying even the slightest variations between wines. But what components make up wine and can they be used to identify different types of wine in an objective manner? Fortunately, determining the chemical composition of liquids has become fairly routine in this age of technology, so obtaining the ingredients in wine and their quantities should be easy. But can they be used to classify the type of wine?

This study aims to discover the ingredient quantities in wine, any patterns between the ingredients, and if they can be used within three unique modeling methods to correctly classify them into their appropriate wine category.

The problem is approached by collecting thirteen different quantities of constituents found in each of three types of wine grown in three different cultivars in the same region of Italy. By using traditional statistical methods to find correlations between these attributes and each class of wine, and patterns within the attributes, the data reveal interesting findings that we can subsequently use to build models that predict if a collection of attributes is a particular category of wine.

One of the interesting findings is that every variable has at least one mean value that is significantly different between class 1, class 2, and/or class 3. This indicates that every independent variable will provide some information to distinguish between wine categories. Furthermore we learn that Proline is a great indicator of wine category 1, while flavonoids and malic acid are great indicators of wine category 3. Additionally, we learn that by using these ingredients a couple of our models achieve 100% accuracy!

By creating a model that can accurately identify wine categories, we create two important benefits for the wine community. First, we expedite the labeling and classifying of wine by making it a more automated process. Second, we objectify the identification of wine categories. Now, the latter may be to the chagrin of many so called wine connoisseurs, who often vehemently defend their “expertly” trained palettes. However, this type of drawback for some is often overridden by the long-term financial benefits of standardization and continuity in an industry.

Modeling Problem

The problem for this study is to develop a model that best predicts the category of wine based on the thirteen attributes measured. Along the way we will learn interesting relationships between the variables and between the variables and each category of wine. These inferences are used to create a larger understanding of the data and improve the value

of the study, but the main goal is to develop a predictive model that correctly identifies the wine category. Hence, this is a classification modeling problem.

The data available consist of continuous variables representing thirteen attributes found in the wines. The dependent response variable, class, is a three value integer value that, when we begin the modeling phase of the project, we convert to a factor with three levels. For clarity, we label the wine category of "1, 2, 3" as "class1, class2, class3," respectively.

The techniques we use in the study are implemented in a systematic fashion in order to initially understand the data and then, based on those findings, let the data guide us to developing the most predictive model. Initially, we check the quality of the data and clean it if necessary. This initial step includes investigating the structure of the data, if we have missing values and if they should be imputed, if the values in the data are valid, if the range of values is appropriate, the presence of any outliers, the distributions of the data within each variable, etc.

Second, we apply traditional EDA to look for relationships in the data based on this being a classification problem. We look to compare the means and distributions between variables using box and whisker plots. We confirm any potential visual differences in mean values with ANOVA and T-tests comparing the means of a variable when it is classified as a particular wine category. Furthermore we look for correlations between the continuous variables. We also run a naïve decision tree to guide our EDA investigation and support any of our previous findings about variable importance.

For creating models, we normally use the training paradigm to determine goodness of fit of our models. We do this by dividing the data into a training and test set based on a 70% to 30% division, respectively, and create our models on the training set and evaluate them on the test set. However, due to the relatively small size of this data set we will only an in-sample training set to train our models. Still, the performance of the model is based on the accuracy between the predicted wine category and the actual wine category of the training set.

When we are ready to build our models, we manually transform the response variable into a factor and attach the labels "class1, class2, class3" as appropriate. We then build three different models: support vector machine, random forest, and neural network.

We build multiple versions of each model by mainly using a full model and by varying the types of parameters available with each model. For instance, support vector machines use four different kernels and vary the appropriate parameters in each to achieve higher accuracy. Neural networks evaluate five versions and vary the number of hidden neurons, layers, threshold, activation and loss functions. Random forests evaluate three versions and vary the amount of trees, the variables used, and the amount of pruned trees to use. Random forest is the only model that uses different subsets of variables to achieve higher accuracy.

Finally, after each parameter adjustment is applied to each model, its accuracy on the in-sample training set is evaluated. Each model's accuracy values on the training set are used as the final performance metric for comparing between models.

Data Quality Check

The data have continuous variables and a multiclass response variable (three classes) and are made up 178 observations and 14 variables.

VARIABLE	CLASS
class	factor
alcohol	numeric
malicacid	numeric
ash	numeric
alcalinityofash	numeric
magnesium	integer
totalphenols	numeric
flavanoids	numeric
nonflavanoidphenols	numeric
proanthocyanins	numeric
colorintensity	numeric
hue	numeric
ods	numeric
proline	integer

Table 1 – Data Description (full table in appendix)

The numeric data is comprised of thirteen variables with two as integers. All are continuous except the response variable which is a three class integer which we change to a three-class factor for modeling. There are not any missing values and all values appear valid.

We gather more information from other descriptive statistics. In terms of range, the range of values appears acceptable for each variable. In terms of outliers, every variable has 2-5 outliers. This comprises 1.1% to 2.8% of the total observations respectively. Furthermore, none of the outliers are so extreme that we consider removing the observation.

Our response variable, class, has three classes: class1, class2, class3. Its class frequencies show frequencies of 59, 71, and 48 for class 1, 2, and 3 respectively. This is not a uniform distribution but it is close enough that we do not consider it as a large class imbalance.

VARIABLE	MIN	Q01	Q25	Q50	Q75	Q99	MAX	MEAN	SD	OUT
alcohol	11.03	11.441	12.362	13.05	13.678	14.473	14.83	13.001	0.812	4
malicacid	0.74	0.898	1.603	1.865	3.083	5.542	5.8	2.336	1.117	4
ash	1.36	1.7	2.21	2.36	2.558	2.989	3.23	2.367	0.274	3
alcalinityofash	10.6	11.354	17.2	19.5	21.5	28.5	30	19.495	3.34	3
magnesium	70	78	88	98	107	141.76	162	99.742	14.282	3
totalphenols	0.98	1.139	1.742	2.355	2.8	3.596	3.88	2.295	0.626	4
flavanoids	0.34	0.47	1.205	2.135	2.875	3.791	5.08	2.029	0.999	3
nonflavanoidphenols	0.13	0.14	0.27	0.34	0.438	0.63	0.66	0.362	0.124	2
proanthocyanins	0.41	0.42	1.25	1.555	1.95	3.034	3.58	1.591	0.572	5
colorintensity	1.28	1.863	3.22	4.69	6.2	11.018	13	5.058	2.318	4
hue	0.48	0.548	0.782	0.965	1.12	1.427	1.71	0.957	0.229	4

ods	1.27	1.29	1.938	2.78	3.17	3.843	4	2.612	0.71	3
proline	278	306.94	500.5	673.5	985	1522.36	1680	746.893	314.907	4

Table 2 – Numeric Data Descriptive Statistics Summary Chart

The distribution of some independent variables is right-skewed. These variables (malicacid, magnesium, colorintensity, nonflavanoidphenols, proline) will be considered for log transformations when we conduct the model building process (Figure 2). There are some variables with double peaked distributions. These variables will be considered for transformations as well.

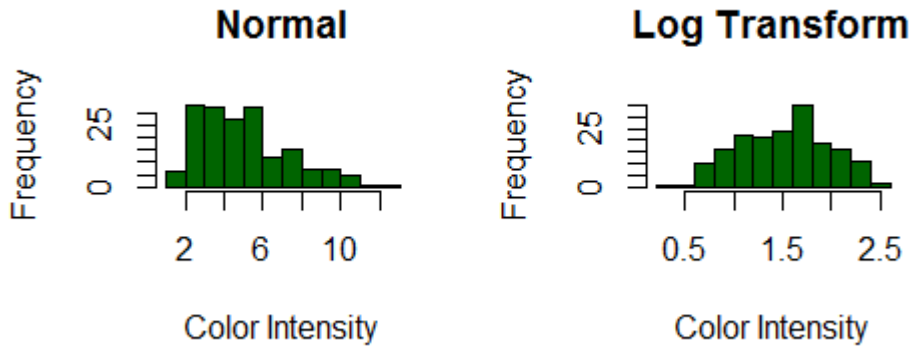

Figure 1 – Distribution of Color Intensity with Log Transformation

Exploratory Data Analysis

Exploratory data analysis is the process of looking for relationships in the data driven by the type of problem. In our case, we look for relationships related to the response variable, spam.

Using correlation to compare a multiclass factor response variable to continuous variables is not appropriate, so we use box and whisker plots to compare the means of a given variable to the three different classes of the response variable. We visually inspect the means and outliers to look for significant differences.

Here (Figure 2) we can see how the mean values of flavonoids, nonflavanoidphenols, and proanthocynanins are different for the three different classes.

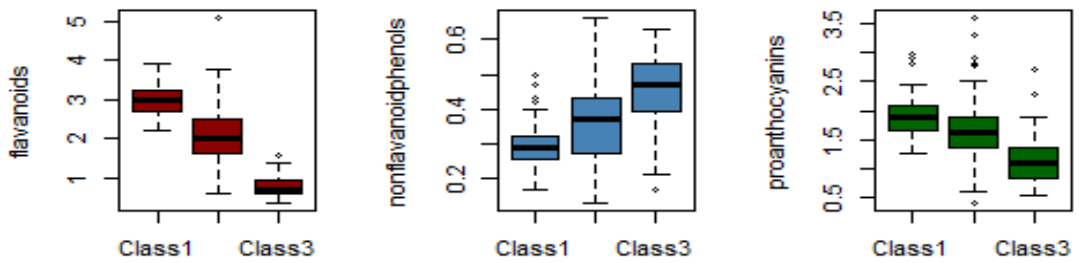

Figure 2 – Box Whisker Plot of Variables vs Wine Class Classification

Since the means look significantly different between the three classes for variable "flavonoid," we run an ANOVA to determine if there are any significant differences between means followed by a t-test to identify which means are significant (Table 3).

Tukey multiple comparisons of means
(95% family-wise confidence level)

Fit: aov(formula = flavanoids ~ class, data = wine)

	diff	lwr	upr	p adj
Class2-Class1	-0.9015278	-1.119784	-0.6832712	0
Class3-Class1	-2.2009145	-2.441737	-1.9600925	0
Class3-Class2	-1.2993867	-1.530899	-1.0678741	0

Table 3 – T-Test of Significant Mean Differences for Flavanoids on Wine Categories

The t-tests tell us every class mean is significantly different for flavonoids. We repeat this process and, interestingly, we find that at least one mean value is significantly different from the other two class means for every variable in the data set. This indicates that every variable can provide some type of predictive information to the model. Additionally, some variables such as proline, flavonoids, and malic acid are very good indicators of one specific class.

Between the continuous variables we see positive correlations greater than .60 for alcohol and proline, totalphenols and flavanoids, proanthocyanins, and ods (see Figure X), and flavonoids and proanthocyanins and ods. These highly correlated pairs are suspect for collinearity issues and possible variable reduction due to the potential of having overlapping information.

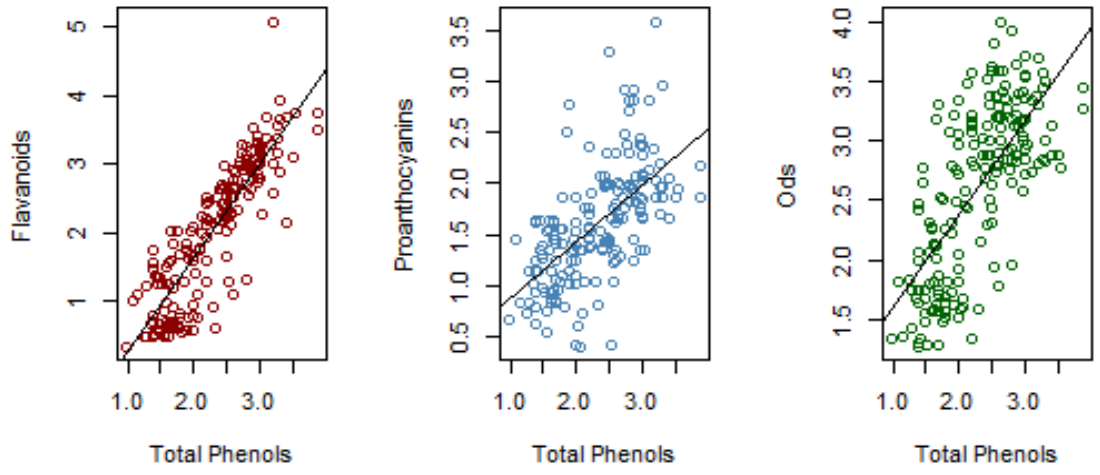

Figure 3 – Strong Correlations with Total Phenols

Finally we run a naïve decision tree on the wine categories (class) using all variables to see what variables are selected as significant. Not surprisingly considering the relationships we discovered with our box whisker plots, it uses flavonoids, proline, hue, and ods for the nodes of all splits. It is interesting to note that the decision tree accuracy rate was already 93.8% using this basic naïve approach.

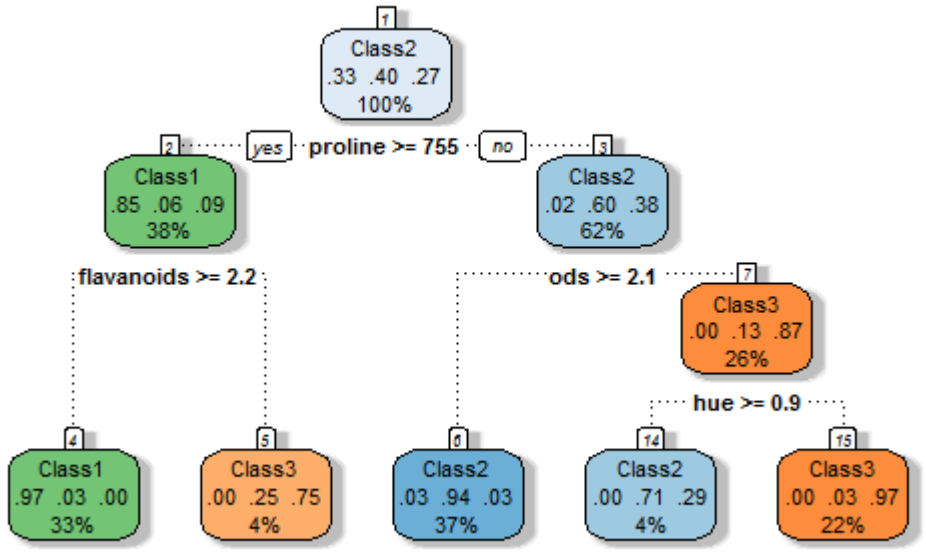

Figure 4 – Naïve Tree Diagram using All Variables

Modeling Methods and Results

Neural Network Models

Model	# Variables	Hidden Neurons	Layers	Threshold	Steps	Error	Accuracy
neural networks	Full - 16	12	1	0.01	3748	0.000293	1.0
neural networks	Full - 16	5	2	0.01	---does not converge ---		
neural networks	Full - 16	5	1	0.01	26	336.54	0.3989
neural networks	Full - 16	100	1	0.01	328	0.000151	1.0
neural networks	Full - 16	3	2	0.01	37	336.54	0.3989

Table 4 – Models Built by Neural Networks

We run five types of neural networks models to determine the effect of parameter changes in the model, specifically adjusting the number of hidden neurons and number of layers. Furthermore, we use an error function of cross-entropy and constant threshold value of .01.

Of our five models, only one, the model using 5 hidden neurons and 2 layers, never converges to the threshold of 0.01. Two models have the same accuracy, models using 5 hidden neurons with 1 layer and three hidden neurons with 2 layers. Two models achieve perfect accuracy, models using hidden neurons of 12 and 100.

	ACTUAL		
PREDICT	**Class1**	**Class2**	**Class3**
Class1	33.15%	0	0
Class2	0	39.89%	0
Class3	0	0	26.97%

Table 5 – Confusion Matrix of 12 Hidden Neurons for Neural Networks

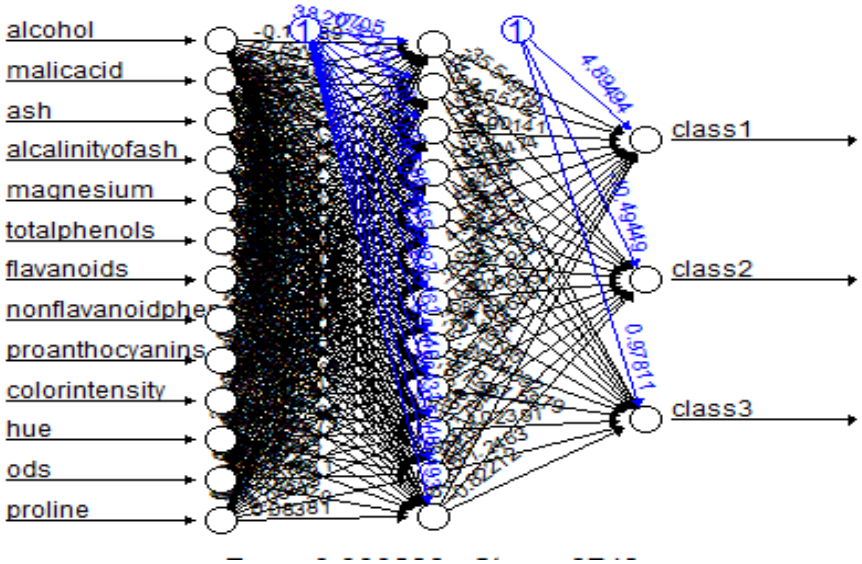

Figure 5 – Visualization of Neural Network with 12 Hidden Neurons

The model using 100 hidden neurons has the highest accuracy, but it only improves the error by .00015. Furthermore, more hidden neurons greatly increase the model complexity.

Next, we reduce the repeat the process with less and less hidden neurons until we find the minimum amount that still achieves 100% accuracy for selecting wine categories. This model has twelve hidden neurons, achieves 100% accuracy, an error of .000293, and improved parsimony over the 100 hidden neurons model. Hence, with some experimenting with the neural network parameters, we can optimize the model and achieve 100% accuracy with this data set.

Support Vector Machine Models

Model	# Variables	Formula	Parameters	Train Accuracy
support vector	Linear	Class ~.	Cost .01	0.994382
support vector	Polynomial	Class ~.	Degree 3, Coef0 2	0.994382
support vector	Radial	Class ~.	Gamma 0.1	1.0000000
support vector	Sigmoid	Class ~.	Gamma 0.1, CoefO 0.1	0.9606742

Table 6 – Models Built by Support Vector Machines

Support Vector Machine models use the full model in every model. The main difference is the kernel, degree, coef0, and gamma values applied.

For the linear kernel we use a soft margin penalty cost of .01 was found to be the best. Using the full model we achieved a training accuracy of 99.4%, almost perfect accuracy. The best polynomial kernel uses the full model only with 3 degrees and a coef0 of 2. It also achieves a training accuracy of 99.44% , matching the linear kernel.

	ACTUAL		
PREDICT	**Class1**	**Class2**	**Class3**
Class1	33%	0	0
Class2	0	40%	0
Class3	0	0	27%

Table 7 – Confusion Matrix of Radial Kernel Support Vector Machine

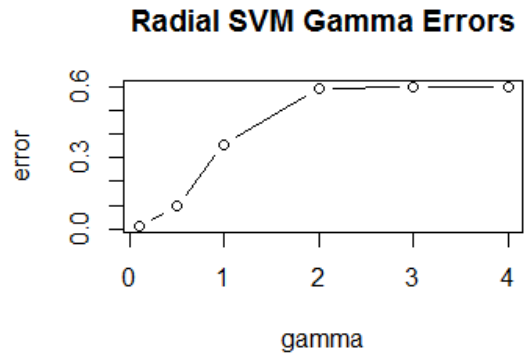

Figure 6 – Gamma Values and Errors for Radial SVM

When we move to radial kernel with a gamma of .1our accuracy increases to perfect accuracy! When we use the sigmoid kernel, the best gamma is 0.1 and coef0 of 0.1. Unfortunately, the data does not follow the sigmoid kernel as well and accuracy drops to the lowest of the kernels at 96.1%.

Random Forest Models

Model	# Variables	500 Trees Accuracy	Pruned # Trees	Pruned Accuracy
random forest	full-13	.9775	38	.9831461
random forest	EDA select logged -9	.9831	16	.9831461
random forest	EDA select - 9	.9831	37	.988764

Table 8 – Models Built by Random Forests

Random forest is run using three models: one full model, one model with EDA selected variables, and one model with log transformed EDA selected variables. Using the pruning technique, for each model we run an initial model using 1000 trees then another using the optimized "pruned" number of trees.

	ACTUAL		
PREDICT	**Class1**	**Class2**	**Class3**
Class1	33%	0	0
Class2	0	39%	0
Class3	0	1%	27%

Table 9 – Confusion Matrix of Random Forest EDA Selected Variables

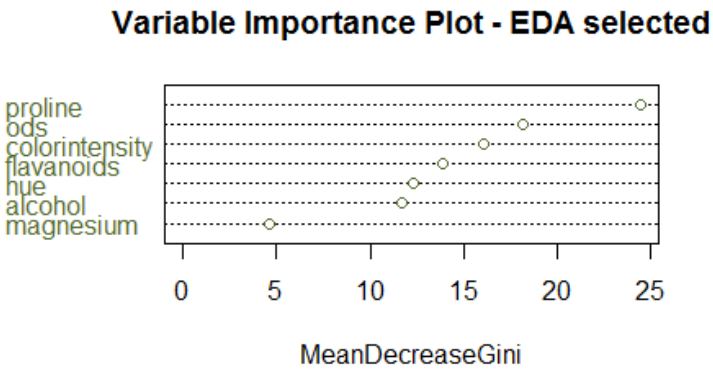

Figure 7 – Variable Importance for Random Forest EDA Selected Variables

The full model performs very well, achieving an initial accuracy rate of .9775 which improves to .9831 by pruning to 38 trees. We try to improve that model by using variables from our EDA that showed significant mean differences between wine categories. This model improves full trees accuracy to .9831 and using 37 trees, improves accuracy to a model best of 98.9%. With this model we see the variable "proline" is the most important.

Finally we try a model with log transformations of highly right skewed EDA selected variables. This model matches the accuracy of our best model when using 500 trees, but

does not improve accuracy when pruning the tree. Hence, by applying our EDA insights we improve the model over a "wood chipper" approach.

Model	# Variables	Parameters	Train Accuracy
neural network	full-13	100 hidden neurons	1.00
neural network	full-13	12 hidden neurons	1.00
neural network	full-13	5 hidden – 1 layer	0.39
neural network	full-13	3 hidden – 2 layers	0.39
SVM - linear	full-13	Cost .01	0.994382
SVM – polynomial	full-13	Degree 3, Coef0 2	0.994382
SVM - radial	full-13	Gamma 0.1	1.00
SVM – sigmoid	full-13	Gamma 0.1, CoefO 0.1	0.9606742
random forest	full-58	500 trees – 38 pruned	.9831461
random forest	EDA selected – 9	500 trees – 37 pruned	.988764
random forest	EDA log transf - 9	500 trees – 16 pruned	.9831461

Table 10 - Comparison of Model Performance

Conclusions

It is clear that, based on our exploratory data analysis and modeling methods, by using the ingredients found in wine an accurate wine classification can be achieved. Specifically, while every random forest model achieves 98%+ accuracy rates, the best accuracy values are found by using neural networks and support vector machines. Both achieve 100% accuracy and neural networks achieve this with two different versions of models. Support vector machines achieve 99.4% accuracy with two of its three best models, showing it fits the data well using the linear, polynomial, and radial kernels. Neural networks continue to perform better with more hidden layers, but at the expense of complexity and computational time. Thus, we recommend the use of a radial kernel support vector machine or a neural network using only twelve hidden neurons for future applications.

A recommendation to improve the random forest models is the more extensive use of boosting, as we believe that 100% accuracy can be achieved with more time with random forest methods. Additionally, it should be noted that by using our EDA methods we learn insights that we can apply and improve our models. Again, this demonstrates the importance of doing an effective EDA to maximize the performance of our models.

Finally, understanding the fervor associated with many individuals intimately involved in the wine industry either recreationally or professionally, our hope is that those wine connoisseurs do not threaten reproach after this project identified many models that can be used to replace their "expert" palettes with 100% accuracy.

Appendix

Hastie, T., James, G., Tibshirani, R., Witten, D. (2013). *An Introduction to Statistical Learning with Applications in R*. New York, NY: Springer.

Johnson, W. & Myatt, G. (2014). *Making Sense of Data I. A Practical Guide to Exploratory Data Analysis and Data Mining*. Hoboken, New Jersey: Wiley.

Lesmeister, C. (2015). *Mastering Machine Learning with R*. Birmingham, UK: Packt Publishing.

Mount, J. & Zumel, N. (2014). *Practical Data Science with R*. Shelter Island, NY: Springer.

TABLE 1

	VAR.NAME	VAR.CLASS
1	class	factor
2	alcohol	numeric
3	malicacid	numeric
4	ash	numeric
5	alcalinityofash	numeric
6	magnesium	integer
7	totalphenols	numeric
8	flavanoids	numeric
9	nonflavanoidphenols	numeric
10	proanthocyanins	numeric
11	colorintensity	numeric
12	hue	numeric
13	ods	numeric
14	proline	integer

R Code

```
#### FIGURE 1 ####
#histogram of numeric distributions
par(mfrow=c(1,2))
hist(wine$colorintensity, xlab="Color Intensity", main="Normal", col="darkgreen")
hist(log(wine$colorintensity), xlab="Color Intensity", main="Log Transform", col="darkgreen")


#### FIGURE 2 #######
#box whisker plots against response levels
par(mfrow=c(1,3))
boxplot(flavanoids~class, data=wine, ylab="flavanoids", col="darkred") # mostly no except, VVS1 and I1 lowest class size
boxplot(nonflavanoidphenols~class, data=wine, ylab="nonflavanoidphenols", col="steelblue")     # no apparent difference
boxplot(proanthocyanins~class, data=wine, ylab="proanthocyanins", col="darkgreen") # mall mean lowest class size


######### FIGURE 3 #############
par(mfrow=c(1,3))
# plot totalphenols vs flavanoids
plot(wine$totalphenols , wine$flavanoids, xlab="Total Phenols", ylab="Flavanoids")
abline(lm(wine$flavanoids~wine$totalphenols), col="darkred", lwd=2)

plot(wine$totalphenols , wine$proanthocyanins, xlab="Total Phenols", ylab="Proanthocyanins")
abline(lm(wine$proanthocyanins~wine$totalphenols), col="steelblue", lwd=2)

plot(wine$totalphenols , wine$ods, xlab="Total Phenols", ylab="Ods")
abline(lm(wine$ods~wine$totalphenols), col="darkgreen", lwd=2)



#######FIGURE 4########
# tree on entire data set no-modified training data set
wine.tree <- rpart(class ~ ., data=wine)
summary(wine.tree) # summarize findings
library(rattle)
fancyRpartPlot(wine.tree, sub="")text(spam.fit, pretty=0)



#######FIGURE 5########
# Neural Network Visual EDA Model
set.seed(71)
nn1 <-neuralnet(f, data=nnet_df[,2:17], hidden=12, err.fct="ce", linear.output=FALSE)
head(nn1$result.matrix)
plot(nn1) # neural network plot with hidden neurons



#######FIGURE 6##########
set.seed(71)
rbf.tune = tune.svm(class~., data=wine, kernel="radial",gamma=c(0.1,0.5,1,2,3,4))
summary(rbf.tune)
best.rbf.full = rbf.tune$best.model
plot(rbf.tune, data=wine, main="Radial SVM Gamma Errors",
```

```
    color = "blue", gcolor = par("fg"), lcolor = "black")



#####FIGURE 7#######
### Random Forest Variable Importance Plot ###
### RF EDA selected variables model ###
set.seed(71)
rf.bkwd2 = randomForest( class ~ alcohol + malicacid + colorintensity +
                         magnesium + flavanoids + proline + ods +
                         totalphenols + alcalinityofash,
                       data=wine, ntree=37)
print(rf.bkwd2)
varImpPlot(rf.full.2, n.var=7, main="Variable Importance Plot - EDA selected",
          color = "darkolivegreen", gcolor = par("fg"), lcolor = "black")







################################################################################
######TABLE 1########

###### data description table ######
VAR.NAME = colnames(wine)

variable.class <- function(df){
  var.class = rep(1,ncol(df),0)
  for(i in 1:ncol(df)){
    var.class[i] = class(df[,i])
  }
  return(var.class)
}

VAR.CLASS <- variable.class(wine)

data.descrip= data.frame(VAR.NAME, VAR.CLASS)

head(data.descrip)
###### end - data description table ######



########TABLE 2############

######my summary function#####
my.summary <- function(in.orig.df){

  # check and create df with only numeric columns for the summary function
  col.to.add = numeric()  # initialize column vector

  for (i in 1:ncol(in.orig.df)){
    if(is.numeric(in.orig.df[,i]) == 'TRUE'){
      col.to.add = append(col.to.add,i)  # add column number to vector
    }
  }
```

```
in.df <-in.orig.df[col.to.add]  # using '<<' adds object to global env for use outside function
VAR.NAME <- colnames(in.df);   # get variable names from df

# count not-NAs function
not.na <- function(x){
  out <- sum(-1*(is.na(x)-1));
  return(out);
}
NOT.NA <- apply(X=in.df,MARGIN=2,FUN='not.na');

# count NAs function
count.na <- function(x){
  out <- sum(is.na(x));
  return(out);
}

###### OUTLIER FUNCTION #####
find.outliers <- function(x){

  # count outliers in each variable based on outside Q01 or Q99 (these limits are adjustable)

  # create quantile of input
  quantile.99=quantile(x,c(0.99), na.rm=T)
  quantile.01=quantile(x,c(0.01), na.rm=T)

  # create min and max of input
  min= min(x)
  max= max(x)

  # initial outlier value for this column loop
  outlier = 0

  for(j in 1:length(x)){  # loop through each value in each column

    # 1) find observation value
    obs=x[j]

    # 2) find observation and outlier barrier ratios
    obs.min.ratio = obs/min
    obs.max.ratio = max/obs
    outlier.min.ratio = quantile.01/min
    outlier.max.ratio = max/quantile.99

    # 3) compare if observation ratio is less than outlier ratios
    result.min = obs.min.ratio < outlier.min.ratio #if observation/min ratio less than outlier/min ratio = outlier
    result.max = obs.max.ratio < outlier.max.ratio #if max/obs ratio less than max/outlier ratio = outlier

    # result.min
    if(result.min == 'TRUE' || result.max == 'TRUE'){
      outlier = outlier + 1
    }

  }

  return(outlier)
}
###### END OUTLIER SECTION #####


#### combine functions and apply() to the numeric data frame #####
```

```
  IS.NA <- apply(X=in.df,MARGIN=2,FUN='count.na');  # number of NAs
  PCT.NA <- round(IS.NA/(NOT.NA+IS.NA),digits=3);   # percent that are NAs
  # Note that we need to account for computations with NA values;

  p <- c(0.01,0.05,0.25,0.50,0.75,0.95,0.99); # set quantiles we want to use
  Q <- round(apply(X=in.df,MARGIN=2,FUN='quantile',p,na.rm=TRUE),digits=3); # quantiles
  tQ.df <- as.data.frame(t(Q));  # transpose quantile vector and turn into data frame to display properly later
  colnames(tQ.df) <- c('Q01','Q05','Q25','Q50','Q75','Q95','Q99');  # change col names in data frame to be more
readable

  MIN <- round(apply(X=in.df,MARGIN=2,FUN='min',na.rm=TRUE),digits=3); # min value
  MAX <- round(apply(X=in.df,MARGIN=2,FUN='max',na.rm=TRUE),digits=3); # max value
  MEAN <- round(apply(X=in.df,MARGIN=2,FUN='mean',na.rm=TRUE),digits=3); # mean value
  SD <- round(apply(X=in.df,MARGIN=2,FUN='sd',na.rm=TRUE),digits=3); # Standard deviation value
  OUT = apply(X=in.df,MARGIN=2,FUN='find.outliers')  # outlier values for each variable stored in vector


  # combine all results into one data frame for display
  wide.df <- cbind(VAR.NAME,NOT.NA,IS.NA,MIN,tQ.df,MAX,MEAN,SD,OUT);
  rownames(wide.df) <- seq(1,dim(wide.df)[1],1);   # rename rownames to numeric values

  # Write CSV in R
  write.csv(wide.df, file = "summary_numeric_data_wine.csv")

  return(wide.df);
}

my.summary.df = my.summary(in.orig.df=wine)  # test my.summary() function with diamonds data set
```

#######TABLE 3 #########

Pairwise T-test results
results = aov(flavanoids ~ class, data=wine)
TukeyHSD(results, conf.level = 0.95)

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = flavanoids ~ class, data = wine)

$class
                    diff       lwr        upr p adj
Class2-Class1 -0.9015278 -1.119784 -0.6832712     0
Class3-Class1 -2.2009145 -2.441737 -1.9600925     0
Class3-Class2 -1.2993867 -1.530899 -1.0678741     0
```

########################

#######TABLE 5 ##########
Neural Network confusion matrix
set.seed(71)
nn1 <-neuralnet(f, data=nnet_df[,2:17], hidden=12, err.fct="ce", linear.output=FALSE)
head(nn1$result.matrix)
plot(nn1) # neural network plot with hidden neurons

```
varNames <- c(1, 2, 3)
nnet_result2 <- data.frame(pred)
names(nnet_result2) <- varNames
nnet_result2$classID <- colnames(nnet_result2)[max.col(nnet_result2, ties.method="first")]
nnet_result2 <- nnet_result2[,4]
cm2 <- table(nnet_result2, nnet_df$class)
accuracy <- sum(diag(cm2))/length(nnet_result2)
cm2
round(prop.table(cm2)*(100), digits=2)



#######TABLE 7 ##########
### SVM confusion matrix ###

### SVM full radial model ###
### SVM full radial model ###
set.seed(71)
rbf.tune = tune.svm(class~., data=wine, kernel="radial",gamma=c(0.1,0.5,1,2,3,4))
summary(rbf.tune)
best.rbf.full = rbf.tune$best.model
tune.train.full = predict(best.rbf.full)
#tune.test.full = predict(best.rbf.full, newdata=spam.test)

conf.train = table(tune.train.full, wine$class)
#conf.test = table(tune.test.full, spam.test$spam)
conf.train.perc = round(conf.train/dim(wine)[1], digits=2)
#conf.test.perc = round(conf.test/dim(spam.test)[1], digits=2)

# Accuracy for train and test sets
accuracy.train=sum(diag(table(tune.train.full,wine$class)))/length(tune.train.full)
#accuracy.test=sum(diag(table(tune.test.full,spam.test$spam)))/length(tune.test.full)
# accuracy train = 1.0
# tune.train.full Class1 Class2 Class3
# Class1     59      0      0
# Class2      0     71      0
# Class3      0      0     48
# tune.train.full Class1 Class2 Class3
# Class1   0.33   0.00   0.00
# Class2   0.00   0.40   0.00
# Class3   0.00   0.00   0.27
# best parameters:
#   gamma - free parameter of gaussian radial basis function
#   a radial kernal "raises" some obs above the 2D plane so a
#   hyperplane can be used to separate them. Small gamma means the
#   obs raised have sharp peaks, which means low bias-high variance
#   High gamma means obs have flatter, broader peaks which menas
#   higher bias-lower variance.
#   0.1


#######TABLE 9 ##########
### Random Forest confusion matrix ###
set.seed(71)
rf.bkwd2 = randomForest( class ~ alcohol + malicacid + colorintensity +
                         magnesium + flavanoids + proline + ods +
                         totalphenols + alcalinityofash,
                       data=wine, ntree=37)
```

```
rf.bkwd.train = predict(rf.bkwd2, type="response")
#rf.bkwd.test = predict(rf.bkwd2, newdata=spam.test, type="response")

conf.train = table(rf.bkwd.train, wine$class)
#conf.test = table(rf.bkwd.test, spam.test$spam)
conf.train.perc = round(conf.train/dim(wine)[1], digits=2)
#conf.test.perc = round(conf.test/dim(spam.test)[1], digits=2)

# Accuracy for train and test sets
accuracy.train=sum(diag(table(rf.bkwd.train,wine$class)))/length(rf.bkwd.train)
#accuracy.test=sum(diag(table(rf.bkwd.test,spam.test$spam)))/length(rf.bkwd.test)
# accuracy train = 0.988764
# rf.bkwd.train Class1 Class2 Class3
# Class1     59      0      0
# Class2      0     69      0
# Class3      0      2     48
# rf.bkwd.train Class1 Class2 Class3
# Class1   0.33   0.00   0.00
# Class2   0.00   0.39   0.00
# Class3   0.00   0.01   0.27




########## Model Based EDA ###############

#
# DECISION TREE -  on entire data set no-modified training data set
wine.tree <- rpart(class ~ ., data=wine)
summary(wine.tree) # summarize findings

library(rattle)
fancyRpartPlot(wine.tree, sub="")
#
# check trained decision tree on test data and evaluate predictive accuracy
pred.wine.train <- predict(wine.tree, type="class")

conf.mat = table(pred.wine.train,wine$class)
accuracy.train=sum(diag(table(pred.wine.train,wine$class)))/length(pred.wine.train)
# predictive accuracy (class ~. ) = 0.9382022
# pred.wine.train Class1 Class2 Class3
# Class1     57      2      0
# Class2      2     66      4
# Class3      0      3     44
```

```
############################################################
############# MODELING METHODS ############################
############################################################


################ Support Vector Machine ####################


### SVM full linear model ###
set.seed(71)
linear.tune.full = tune.svm(class~., data=wine, kernel="linear", cost=c(0.001,0.01, 0.1, 1,5,10))
summary(linear.tune.full)
best.linear.full = linear.tune.full$best.model

### SVM full polynomial model ###
set.seed(71)
poly.tune.full = tune.svm(class~., data=wine, kernel="polynomial",degree=c(3,4,5), coef0=c(0.1,0.5,1,2,3,4))
summary(poly.tune.full)
best.poly.full = poly.tune.full$best.model

### SVM full radial model ###
set.seed(71)
rbf.tune = tune.svm(class~., data=wine, kernel="radial",gamma=c(0.1,0.5,1,2,3,4))
summary(rbf.tune)
best.rbf.full = rbf.tune$best.model
plot(rbf.tune, data=wine, main="Radial SVM Gamma Errors",
    color = "blue", gcolor = par("fg"), lcolor = "black")



### SVM full sigmoid model ###
set.seed(71)
sigmoid.tune = tune.svm(class~., data=wine, kernel="sigmoid",gamma=c(0.1,0.5,1,2,3,4),
coef0=c(0.1,0.5,1,2,3,4))
summary(sigmoid.tune)
best.sigmoid.full = sigmoid.tune$best.model



###### SVM Predictions - train/test data set ######

### SVM full linear model ###
tune.train.full = predict(best.linear.full, type="response")
#tune.test.full = predict(best.linear.full, newdata=spam.test, type="response")

conf.train = table(tune.train.full, wine$class)
#conf.test = table(tune.test.full, spam.test$spam)
conf.train.perc = round(conf.train/dim(wine.train)[1], digits=2)
#conf.test.perc = round(conf.test/dim(spam.test)[1], digits=2)

# Accuracy for train and test sets
accuracy.train=sum(diag(table(tune.train.full,wine$class)))/length(tune.train.full)
#accuracy.test=sum(diag(table(tune.test.full,wine.test$spam)))/length(tune.test.full)
# accuracy train = 0.994382  FP = 83, .03
# tune.train.full Class1 Class2 Class3
# Class1     59      0      0
# Class2      0     70      0
# Class3      0      1     48
# tune.train.full Class1 Class2 Class3
# Class1   0.47   0.00   0.00
```

```
# Class2   0.00   0.56   0.00
# Class3   0.00   0.01   0.38
# - best parameters:
#   cost - .01 - the soft margin that trades error for stability
#   large C gives low bias-high variance, bc penalizes
#   misclassificaiton a lot
#   10 - making this lower may improve FP (misclassifying non-spam as spam)

### SVM full poly model ###
tune.train.full = predict(best.poly.full, type="response")
#tune.test.full = predict(best.poly.full, newdata=spam.test, type="response")

conf.train = table(tune.train.full, wine$class)
conf.train.perc = round(conf.train/dim(wine)[1], digits=2)
#conf.test.perc = round(conf.test/dim(spam.test)[1], digits=2)

# Accuracy for train and test sets
accuracy.train=sum(diag(table(tune.train.full,wine$class)))/length(tune.train.full)
#accuracy.test=sum(diag(table(tune.test.full,spam.test$spam)))/length(tune.test.full)
# accuracy train = [1] 0.994382
# tune.train.full Class1 Class2 Class3
# Class1     58      0      0
# Class2      1     71      0
# Class3      0      0     48
# tune.train.full Class1 Class2 Class3
# Class1   0.33   0.00   0.00
# Class2   0.01   0.40   0.00
# Class3   0.00   0.00   0.27
# - best parameters:
#   degree coef0
#       3   0.1
# Degree is the degree of the polynomial
# Coef0 is a limit on the coeficent so doesn't go to infinity
# or continue to smaller numbers approaching zero?


### SVM full radial model ###
tune.train.full = predict(best.rbf.full)
#tune.test.full = predict(best.rbf.full, newdata=spam.test)

conf.train = table(tune.train.full, wine$class)
#conf.test = table(tune.test.full, spam.test$spam)
conf.train.perc = round(conf.train/dim(wine)[1], digits=2)
#conf.test.perc = round(conf.test/dim(spam.test)[1], digits=2)

# Accuracy for train and test sets
accuracy.train=sum(diag(table(tune.train.full,wine$class)))/length(tune.train.full)
#accuracy.test=sum(diag(table(tune.test.full,spam.test$spam)))/length(tune.test.full)
# accuracy train = 1.0
# tune.train.full Class1 Class2 Class3
# Class1     59      0      0
# Class2      0     71      0
# Class3      0      0     48
# tune.train.full Class1 Class2 Class3
# Class1   0.33   0.00   0.00
# Class2   0.00   0.40   0.00
# Class3   0.00   0.00   0.27
# best parameters:
#   gamma - free parameter of gaussian radial basis function
#   a radial kernal "raises" some obs above the 2D plane so a
```

```
#   hyperplane can be used to separate them. Small gamma means the
#   obs raised have sharp peaks, which means low bias-high variance
#   High gamma means obs have flatter, broader peaks which menas
#   higher bias-lower variance.
#   0.1

### SVM full sigmoid model ###
tune.train.full = predict(best.sigmoid.full)
#tune.test.full = predict(best.sigmoid.full, newdata=spam.test)

conf.train = table(tune.train.full, wine$class)
#conf.test = table(tune.test.full, spam.test$spam)
conf.train.perc = round(conf.train/dim(wine)[1], digits=2)
#conf.test.perc = round(conf.test/dim(spam.test)[1], digits=2)

# Accuracy for train and test sets
accuracy.train=sum(diag(table(tune.train.full,wine$class)))/length(tune.train.full)
#accuracy.test=sum(diag(table(tune.test.full,spam.test$spam)))/length(tune.test.full)
# accuracy train = 0.9606742
# tune.train.full Class1 Class2 Class3
# Class1     59      4      0
# Class2      0     65      1
# Class3      0      2     47
# tune.train.full Class1 Class2 Class3
# Class1   0.33   0.02   0.00
# Class2   0.00   0.37   0.01
# Class3   0.00   0.01   0.26
# best parameters:
#   gamma coef0
#   0.1   0.1

### end - Support Vector Machine predictions ######

################ end - Support Vector Machine###############




################ Random Forest ####################

### RF full model ###
set.seed(71)
rf.full = randomForest(class~., data=wine)
print(rf.full)

#determine how many trees for min error rate
plot(rf.full)
which.min(rf.full$err.rate[,1])

#retry model with min number of trees
set.seed(71)
rf.full.2 = randomForest(class~., data=wine, ntree=38)
print(rf.full.2)
varImpPlot(rf.full.2, n.var=10, main="Variable Importance Plot - Full Model",
          color = "darkolivegreen", gcolor = par("fg"), lcolor = "black")


### RF bkwd vars model ###
set.seed(71)
```

```
rf.bkwd = randomForest( class ~ alcohol + malicacid + colorintensity +
                    magnesium + flavanoids + proline + ods +
                    totalphenols + alcalinityofash,
                   data=wine)
print(rf.bkwd)

#determine how many trees for min error rate
plot(rf.bkwd)
which.min(rf.bkwd$err.rate[,1])

#retry model with min number of trees
set.seed(71)
rf.bkwd2 = randomForest( class ~ alcohol + malicacid + colorintensity +
                     magnesium + flavanoids + proline + ods +
                     totalphenols + alcalinityofash,
                    data=wine, ntree=37)
print(rf.bkwd2)
varImpPlot(rf.full.2, n.var=7, main="Variable Importance Plot - EDA selected",
         color = "darkolivegreen", gcolor = par("fg"), lcolor = "black")



### RF log vars model ###
set.seed(71)
rf.bkwd = randomForest( class ~ alcohol + log(wine$malicacid) + log(wine$colorintensity) +
                    log(wine$magnesium) + flavanoids + log(wine$proline) + ods +
                    totalphenols + alcalinityofash,
                   data=wine)
print(rf.bkwd)

#determine how many trees for min error rate
plot(rf.bkwd)
which.min(rf.bkwd$err.rate[,1])

#retry model with min number of trees
set.seed(71)
rf.bkwd2 = randomForest( class ~ alcohol + log(wine$malicacid) + log(wine$colorintensity) +
                     log(wine$magnesium) + flavanoids + log(wine$proline) + ods +
                     totalphenols + alcalinityofash,
                    data=wine, ntree=16)
print(rf.bkwd2)
varImpPlot(rf.bkwd2, main="Variable Importance Plot - Bkwd Model")



###### Random Forest Predictions - train/test data set ######

### RF full model ###
rf.full.train = predict(rf.full.2, type="response")
#rf.full.test = predict(rf.full.2, newdata=spam.test, type="response")

conf.train = table(rf.full.train, wine$class)
#conf.test = table(rf.full.test, spam.test$spam)
conf.train.perc = round(conf.train/dim(wine)[1], digits=2)
#conf.test.perc = round(conf.test/dim(spam.test)[1], digits=2)

# Accuracy for train and test sets
accuracy.train=sum(diag(table(rf.full.train,wine$class)))/length(rf.full.train)
```

```
#accuracy.test=sum(diag(table(rf.full.test,spam.test$spam)))/length(rf.full.test)
# accuracy train = 0.9831461
# rf.full.train Class1 Class2 Class3
# Class1     58      1      0
# Class2      1     69      0
# Class3      0      1     48
# rf.full.train Class1 Class2 Class3
# Class1   0.33   0.01   0.00
# Class2   0.01   0.39   0.00
# Class3   0.00   0.01   0.27


### RF EDA model ###
rf.bkwd.train = predict(rf.bkwd2, type="response")
#rf.bkwd.test = predict(rf.bkwd2, newdata=spam.test, type="response")

conf.train = table(rf.bkwd.train, wine$class)
#conf.test = table(rf.bkwd.test, spam.test$spam)
conf.train.perc = round(conf.train/dim(wine)[1], digits=2)
#conf.test.perc = round(conf.test/dim(spam.test)[1], digits=2)

# Accuracy for train and test sets
accuracy.train=sum(diag(table(rf.bkwd.train,wine$class)))/length(rf.bkwd.train)
#accuracy.test=sum(diag(table(rf.bkwd.test,spam.test$spam)))/length(rf.bkwd.test)
# accuracy train = 0.988764
# rf.bkwd.train Class1 Class2 Class3
# Class1     59      0      0
# Class2      0     69      0
# Class3      0      2     48
# rf.bkwd.train Class1 Class2 Class3
# Class1   0.33   0.00   0.00
# Class2   0.00   0.39   0.00
# Class3   0.00   0.01   0.27



### RF EDA logged transformed model ###
rf.bkwd.train = predict(rf.bkwd2, type="response")
#rf.bkwd.test = predict(rf.bkwd2, newdata=spam.test, type="response")

conf.train = table(rf.bkwd.train, wine$class)
#conf.test = table(rf.bkwd.test, spam.test$spam)
conf.train.perc = round(conf.train/dim(wine)[1], digits=2)
#conf.test.perc = round(conf.test/dim(spam.test)[1], digits=2)

# Accuracy for train and test sets
accuracy.train=sum(diag(table(rf.bkwd.train,wine$class)))/length(rf.bkwd.train)
#accuracy.test=sum(diag(table(rf.bkwd.test,spam.test$spam)))/length(rf.bkwd.test)
# accuracy train = 0.988764
# rf.bkwd.train Class1 Class2 Class3
# Class1     59      0      0
# Class2      0     69      0
# Class3      0      2     48
# rf.bkwd.train Class1 Class2 Class3
# Class1   0.33   0.00   0.00
# Class2   0.00   0.39   0.00
# Class3   0.00   0.01   0.27

### end - Random Forest predictions ######
################ end - Random Forest ###############
```

```
########## NEURAL NETWORKS ######################

#Store data in a new data frame to manipulate for NN
colnames=c("class", "alcohol", "malicacid", "ash", "alcalinityofash", "magnesium",
        "totalphenols", "flavanoids","nonflavanoidphenols","proanthocyanins",
        "colorintensity","hue","ods","proline")
names(mydata) = colnames

nnet_df <- mydata

#Dummy variables for each class
nnet_df <- cbind(nnet_df, mydata$class == '1')
nnet_df <- cbind(nnet_df, mydata$class == '2')
nnet_df <- cbind(nnet_df, mydata$class == '3')

# change name of the dummies just created
names(nnet_df)[15] <- 'class1'
names(nnet_df)[16] <- 'class2'
names(nnet_df)[17] <- 'class3'

# create a text formula
f <- as.formula(paste("class1+class2+class3~alcohol+malicacid+ash+alcalinityofash+
                magnesium+totalphenols+flavanoids+nonflavanoidphenols+
                proanthocyanins+colorintensity+hue+ods+proline"))

# or ...

n <- names(wine)
f2 <- as.formula(paste("class1+class2+class3~", paste(n[!n %in% "class"], collapse = " +
")))


# Model
set.seed(71)
nn1 <-neuralnet(f, data=nnet_df[,2:17], hidden=12, err.fct="ce", linear.output=FALSE)
head(nn1$result.matrix)
plot(nn1) # neural network plot with hidden neurons
# head(nn1$generalized.weights[[1]])
# ls(nn1)
# nn1$response
# nn1$net.result
ls(nn1)
nn1$err.fct


# head(nnet_df[,1:13])
# nnet_df[,1] is class, [,15:17] are dummy vars for class1, class2, class3
# so will use [,2:14]
res = compute(nn1, nnet_df[,2:14])
pred = res$net.result

#pred = ifelse(pred>=0.5,1,0) if binomail response
#pred
#table(pred, nnet_df$class)


varNames <- c(1, 2, 3)
# Store result as data frame
# nnet_result <- data.frame(nn1$net.result)
nnet_result2 <- data.frame(pred)
```

```
# Change result columns to respective category name
# names(nnet_result) <- varNames
names(nnet_result2) <- varNames
# Create prediction bases on highest value of result columns
# nnet_result$classID <- colnames(nnet_result)[max.col(nnet_result, ties.method="first")]
nnet_result2$classID <- colnames(nnet_result2)[max.col(nnet_result2, ties.method="first")]
# Separate prediction column and store as data frame
# nnet_result <- nnet_result[,4]
nnet_result2 <- nnet_result2[,4]
# Compute confusion matrix on actual and predicted data frames
# cm1 <- table(nnet_result, nnet_df$class)
cm2 <- table(nnet_result2, nnet_df$class)
# Return %
# prop.table(cm1)*(100)
accuracy <- sum(diag(cm2))/length(nnet_result2)
cm2
round(prop.table(cm2)*(100), digits=2) end - Random Forest ###############
```