Laboratory #1 Classification

Table of Contents

The Online News Popularity data set from the University of California – Irvine Machine Learning Data Repository is provided for your use as well as two related journal articles, "Predicting and Evaluating the Popularity of Online News" by He Ren and Quan Yang, and "A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News" by Kelwin Fernandes, Pedro Vinagre and Paulo Cortez. The URL for the data set description is: http://archive.ics.uci.edu/ml/datasets/Online+News+Popularity#. Ultimately, your assignment is to use this one dataset and compare the results of these different machine learning techniques to evaluate which of those provides the best results. Because you are assessing the "market share" based on different measures of popularity, in this case the best results will be the results that provide the best understanding of what drives market share. This is different than determining exactly what the market share is.

The steps for you to follow to do this include:

1. Using the datasets provided run each of the different machine learning techniques as described in this laboratory documentation. This will provide you with familiarity of each technique. This counts for roughly 50% of the points available for this laboratory.
2. Next, understanding how each of these techniques works use each of the techniques to evaluate the online news popularity dataset. Note that this will involve some significant data "munging" or pre-processing to get the dataset in a form that will work. That is, each technique may require you to pre-process the dataset in a different way.
3. Generate your written laboratory report on your work. You should use a standard style for writing, APA is preferred.

Below is a matrix of the points available for this laboratory.

Solution by:	Original Example	Online News Popularity	Total
Tree-based classification	25	10	35
Support Vector Machines	25	10	35

Adding Regression to Trees	25	10	35
Solution for Online New Popularity		20	20
Overall paper quality by rubric		10	10
			100
Extra Points			+35

Examples for the Classification Methods (I've added the R commands to each of the methods below. Be sure to note that for two of the methods the commands used in the textbook no longer worked with the R packages designated. I've substituted working commands in these cases. (Try not to just copy and paste the commands into R. You'll understand this all better if you at least take the time to type the commands in yourself!)

Method #1. Tree-based classification

(MLR pg 128)

Step 1: Collecting the data

The data set used for this method is the credit.csv file. This data set is available for you on Moodle. In addition, you can check it out at the UCI Data Repository at http://archive.ics.uci.edu/ml/. Before you use a data set you should check on its description to see how big the data set is, what format it is in, what processing has already been done, and to determine if there is anything "quirky" about it that you need to know in order to conduct your analysis. For example, the class variable "default" is the 17th column, or variable, out of the 17 in this dataset. We'll use this knowledge when we create our model later. For now, use the following command to read it into R.

```
> credit <- read.csv("credit.csv")
```

Use the structure command to check what is in the credit object created to hold the data in R. The str command is shown below with the first few lines returned.

```
> str(credit)
'data.frame':  1000 obs. of  17 variables:
 $ checking_balance    : Factor w/ 4 levels "< 0 DM","> 200 DM",..: 1 3 4 1 1
4 4 3 4 3 ...
 $ months_loan_duration: int  6 48 12 42 24 36 24 36 12 30 ...
 $ credit_history      : Factor w/ 5 levels "critical","good",..: 1 2 1 2 4 2
2 2 2 1 ...
```

Step 2: Exploring the data

There are many ways you can further check the data in this object, e.g. summary or table. For example, you can use the summary() command to get information about loan amounts requested as follows (keep in mind that the monetary unit is Deutsch Marks (DM)). The table lists the number of loans that were defaulted or not.

```
> summary(credit$amount)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    250    1366    2320    3271    3972   18420
> table(credit$default)

 no yes
700 300
```

In order to develop the tree-based classification model you need to split the data into training and test records. But before we do that this particular data set has not been randomized so we can do a good analysis with it. Use the following to randomize the 1000 observations (records) in this data set.

```
> set.seed(12345)
> credit_rand <- credit[order(runif(1000)), ]
```

You can and should check to make sure that randomizing your data has not made any substantive changes to it, i.e. the means should still be the same and so on. You can check the summary on the original data to the summary using the new object you created to do this.

```
> summary(credit$amount)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    250    1366    2320    3271    3972   18420
> summary(credit_rand$amount)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    250    1366    2320    3271    3972   18420
```

You'll need to subset the observations (records) to establish the training set and the test set. Sort of a rule of thumb is to use between 75% and 90% of the records for the training set. There are many ways to do this in R. For example, you can use:

```
> credit_train <- credit_rand[1:900, ]
> credit_test <- credit_rand[901:1000, ]
```

If the randomization went well then the percentages between splits should be close. Before we checked on the number of defaults using the table() command. You can check the percentages rather than actual numbers for both the training set and the test set using:

```
> prop.table(table(credit_train$default))

       no       yes
0.7022222 0.2977778
> prop.table(table(credit_test$default))

  no  yes
0.68 0.32
```

Step 3: Training a model on the data

You'll need the C5.0 algorithm to complete this step. If you haven't already you'll need to install the C50 package. To start you'll need to attach it to this session using:

```
> library(C50)
```

You only need one command line to create the decision tree model. However, now you need to set-up this command so that the algorithm knows that the class or response variable is in the 17th column. The R command and the model created is:

```
> credit_model

Call:
C5.0.default(x = credit_train[-17], y = credit_train$default)

Classification Tree
Number of samples: 900
Number of predictors: 16

Tree size: 67

Non-standard options: attempt to group attributes
```

To actually examine the decision tree itself use the summary() command:

```
> summary(credit_model)

Call:
C5.0.default(x = credit_train[-17], y = credit_train$default)


C5.0 [Release 2.07 GPL Edition]       Mon Aug 15 12:17:20 2016
-------------------------------

Class specified by attribute `outcome'
```

```
Read 900 cases (17 attributes) from undefined.data

Decision tree:

checking_balance = unknown: no (358/44)
checking_balance in {< 0 DM,> 200 DM,1 - 200 DM}:
:...credit_history in {perfect,very good}:
    :...dependents > 1: yes (10/1)
    :   dependents <= 1:
    :   :...savings_balance = < 100 DM: yes (39/11)
    :       savings_balance in {> 1000 DM,500 - 1000 DM,unknown}: no (8/1)
    :       savings_balance = 100 - 500 DM:
    :       :...checking_balance = < 0 DM: no (1)
    :           checking_balance in {> 200 DM,1 - 200 DM}: yes (5/1)
    credit_history in {critical,good,poor}:
    :...months_loan_duration <= 11: no (87/14)
        months_loan_duration > 11:
        :...savings_balance = > 1000 DM: no (13)
            savings_balance in {< 100 DM,100 - 500 DM,500 - 1000 DM,unknown}:
            :...checking_balance = > 200 DM:
                :...dependents > 1: yes (3)
                :   dependents <= 1:
                :   :...credit_history in {good,poor}: no (23/3)
                :       credit_history = critical:
                :       :...amount <= 2337: yes (3)
                :           amount > 2337: no (6)
                checking_balance = 1 - 200 DM:
                :...savings_balance = unknown: no (34/6)
                :   savings_balance in {< 100 DM,100 - 500 DM,500 - 1000 DM}:
                :   :...months_loan_duration > 45: yes (11/1)
                :       months_loan_duration <= 45:
                :       :...other_credit = store:
                :           :...age <= 35: yes (4)
                :           :   age > 35: no (2)
                :           other_credit = bank:
                :           :...years_at_residence <= 1: no (3)
                :           :   years_at_residence > 1:
                :           :   :...existing_loans_count <= 1: yes (5)
                :           :       existing_loans_count > 1:
                :           :       :...percent_of_income <= 2: no (4/1)
                :           :           percent_of_income > 2: yes (3)
                :           other_credit = none:
                :           :...job = unemployed: no (1)
                :               job = management:
                :               :...amount <= 7511: no (10/3)
                :               :   amount > 7511: yes (7)
                :               job = unskilled: [S1]
                :               job = skilled:
                :               :...dependents <= 1: no (55/15)
                :                   dependents > 1:
                :                   :...age <= 34: no (3)
                :                       age > 34: yes (4)
                checking_balance = < 0 DM:
                :...job = management: no (26/6)
                    job = unemployed: yes (4/1)
                    job = unskilled:
```

```
                    :...employment_duration in {4 - 7 years,
                    :   :                       unemployed}: no (4)
                    :   employment_duration = < 1 year:
                    :   :...other_credit = bank: no (1)
                    :   :   other_credit in {none,store}: yes (11/2)
                    :   employment_duration = > 7 years:
                    :   :...other_credit in {bank,none}: no (5/1)
                    :   :   other_credit = store: yes (2)
                    :   employment_duration = 1 - 4 years:
                    :   :...age <= 39: no (14/3)
                    :       age > 39:
                    :       :...credit_history in {critical,good}: yes (3)
                    :           credit_history = poor: no (1)
                    job = skilled:
                    :...credit_history = poor:
                        :...savings_balance in {< 100 DM,100 - 500 DM,
                        :   :                   500 - 1000 DM}: yes (8)
                        :   savings_balance = unknown: no (1)
                        credit_history = critical:
                        :...other_credit = store: no (0)
                        :   other_credit = bank: yes (4)
                        :   other_credit = none:
                        :   :...savings_balance in {100 - 500 DM,
                        :       :                   unknown}: no (1)
                        :       savings_balance = 500 - 1000 DM: yes (1)
                        :       savings_balance = < 100 DM:
                        :       :...months_loan_duration <= 13:
                        :           :...percent_of_income <= 3: yes (3)
                        :           :   percent_of_income > 3: no (3/1)
                        :           months_loan_duration > 13:
                        :           :...amount <= 5293: no (10/1)
                        :               amount > 5293: yes (2)
                        credit_history = good:
                        :...existing_loans_count > 1: yes (5)
                            existing_loans_count <= 1:
                            :...other_credit = store: no (2)
                                other_credit = bank:
                                :...percent_of_income <= 2: yes (2)
                                :   percent_of_income > 2: no (6/1)
                                other_credit = none: [S2]

SubTree [S1]

employment_duration in {< 1 year,1 - 4 years}: yes (11/3)
employment_duration in {> 7 years,4 - 7 years,unemployed}: no (8)

SubTree [S2]

savings_balance = 100 - 500 DM: yes (3)
savings_balance = 500 - 1000 DM: no (1)
savings_balance = unknown:
:...phone = no: yes (9/1)
:   phone = yes: no (3/1)
savings_balance = < 100 DM:
:...percent_of_income <= 1: no (4)
    percent_of_income > 1:
    :...phone = yes: yes (10/1)
```

```
        phone = no:
        :...purpose in {business,car0,education,renovations}: yes (3)
            purpose = car:
            :...percent_of_income <= 3: no (2)
            :   percent_of_income > 3: yes (6/1)
            purpose = furniture/appliances:
            :...years_at_residence <= 1: no (4)
                years_at_residence > 1:
                :...housing = other: no (1)
                    housing = rent: yes (2)
                    housing = own:
                    :...amount <= 1778: no (3)
                        amount > 1778:
                        :...years_at_residence <= 3: yes (6)
                            years_at_residence > 3: no (3/1)


Evaluation on training data (900 cases):

          Decision Tree
        ----------------
        Size      Errors

          66  125(13.9%)   <<


         (a)   (b)    <-classified as
        ----  ----
         609    23    (a): class no
         102   166    (b): class yes


        Attribute usage:

        100.00% checking_balance
         60.22% credit_history
         53.22% months_loan_duration
         49.44% savings_balance
         30.89% job
         25.89% other_credit
         17.78% dependents
          9.67% existing_loans_count
          7.22% percent_of_income
          6.67% employment_duration
          5.78% phone
          5.56% amount
          3.78% years_at_residence
          3.44% age
          3.33% purpose
          1.67% housing


Time: 0.0 secs
```

Based on the evaluation above for this output the algorithm properly classified 609 records as (a), 166 records as (b) and the remaining 125 records create a 13.9% (misclassification) error in training.

Step 4: Evaluating Model Performance

We still need to use our test set to evaluate/validate the model's overall performance. To do this we'll use the predict() command as follows:

```
> cred_pred <- predict(credit_model, credit_test)
```

Last, we'll use the gmodels package to create a confusion table looking at the predicted and actual values using the training and test sets for our credit data. So, first make sure gmodels is installed and attached using library(gmodels). Then use the CrossTable() command as follows:

```
> CrossTable(credit_test$default, cred_pred, prop.chisq = FALSE, prop.c = FAL
SE, prop.r = FALSE, dnn = c('actual default', 'predicted default'))
```

```
   Cell Contents
|-------------------------|
|                       N |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  100


                | predicted default
actual default |        no |       yes | Row Total |
---------------|-----------|-----------|-----------|
            no |        57 |        11 |        68 |
               |     0.570 |     0.110 |           |
---------------|-----------|-----------|-----------|
           yes |        16 |        16 |        32 |
               |     0.160 |     0.160 |           |
---------------|-----------|-----------|-----------|
  Column Total |        73 |        27 |       100 |
---------------|-----------|-----------|-----------|
```

The table indicates that for the 100 records in our test set 11 defaults were misclassified, i.e. false negatives or a Type II error, and 16 actual defaults were misclassified as not defaulting, i.e. false positives or a Type I error.

We could go one step further and work on improving our model performance but will defer that until after our discussion of "Boosting" later in the course.

Method #2. Support vector machines

(MLR pg 233)

Step 1: Collecting the Data

We'll look at support vector machines from the perspective of optical character recognition (OCR). We'll use the letterdata.csv file also from the UCI Data Repository. We can simply read it in and look at the structure as follows:

```
> letters <- read.csv("letterdata.csv")
> str(letters)
'data.frame':  20000 obs. of  17 variables:
 $ letter: Factor w/ 26 levels "A","B","C","D",..: 20 9 4 14 7 19 2 1 10 13 .
..
 $ xbox  : int  2 5 4 7 2 4 4 1 2 11 ...
 $ ybox  : int  8 12 11 11 1 11 2 1 2 15 ...
 $ width : int  3 3 6 6 3 5 5 3 4 13 ...
 $ height: int  5 7 8 6 1 8 4 2 4 9 ...
 $ onpix : int  1 2 6 3 1 3 4 1 2 7 ...
 $ xbar  : int  8 10 10 5 8 8 8 8 10 13 ...
 $ ybar  : int  13 5 6 9 6 8 7 2 6 2 ...
 $ x2bar : int  0 5 2 4 6 6 6 2 2 6 ...
 $ y2bar : int  6 4 6 6 6 9 6 2 6 2 ...
 $ xybar : int  6 13 10 4 6 5 7 8 12 12 ...
 $ x2ybar: int  10 3 3 4 5 6 6 2 4 1 ...
 $ xy2bar: int  8 9 7 10 9 6 6 8 8 9 ...
 $ xedge : int  0 2 3 6 1 0 2 1 1 8 ...
 $ xedgey: int  8 8 7 10 7 8 8 6 6 1 ...
 $ yedge : int  0 4 3 2 5 9 7 2 1 1 ...
 $ yedgex: int  8 10 9 8 10 7 10 7 7 8 ...
```

Step 2: Preparing the Data

The only thing we need to do to prepare this data set is to subset it into our training and test data sets. There are 20,000 observations so we have considerable flexibility in deciding how many to put in our training set and how many to keep for testing. In class we decided to use 18,000. We'll stick with that number now.

```
> letters_train <- letters[1:18000, ]
> letters_test <- letters[18001:20000, ]
```

Step 3: Training a Model on the Data

Again, we'll need to install and attach a package to complete this analysis, i.e. the kernlab package. The actual command is the ksvm() command in this package. It requires us to set up a syntax like the formulas we've used before fn(y ~ x, …). We'll also use the "vanilladot" kernel for this analysis. For more information on the ksvm() command and what the "vanilladot"

kernel is use the help("ksvm") command in RStudio. It may take some time for the algorithm to run. You'll see a red "stop sign" icon at the top of the console frame in RStudio while it is running. The complete command is:

```
> letter_classifier <- ksvm(letter ~ ., data = letters_train, kernel = "vanil
ladot")
```

As an initial indication of how the algorithm performed simply enter the object name you used for the output from the ksvm() command at the command prompt, i.e.:

```
> letter_classifier
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Linear (vanilla) kernel function.

Number of Support Vectors : 7886

Objective Function Value : -15.3458 -21.3403 -25.7672 -6.8685 -8.8812 -35.955
5 …
… -151.3409 -90.0329 -27.9491 -42.4688 -12.5118 -26.4828 -2.0045 -62.195 -9.1
662 -178.4616 -1.9406 -1.9871 -11.3982 -0.5214 -29.6136 -35.0449 -6.7569
Training error : 0.1335
```

That is, we have an initial training error of a bit over 13%.

Step 4: Evaluating Model Performance

Again, we'll develop a confusion table to determine how well our model performed overall. Since we are looking at letters of the alphabet we'll have a considerable number of columns and rows to our table. You might want to adjust the size of the console frame accordingly. The commands to run the evaluation and output the table are:

```
> letter_predictions <- predict(letter_classifier, letters_test)
> table(letter_predictions, letters_test$letter)

letter_predictions  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z
                 A 73  0  0  0  0  0  0  0  0  1  0  0  0  0  3  0  4  0  0  1  2  0  1  0  0  0
                 B  0 61  0  3  2  0  1  1  0  0  1  1  0  0  0  2  0  1  3  0  0  0  0  0  0  0
                 C  0  0 64  0  2  0  4  2  1  0  1  2  0  0  1  0  0  0  0  0  0  0  0  0  0  0
                 D  2  1  0 67  0  0  1  3  3  2  1  2  0  3  4  2  1  2  0  0  0  0  0  0  1  0
                 E  0  0  1  0 64  1  1  0  0  0  2  2  0  0  0  0  2  0  6  0  0  0  0  1  0  0
                 F  0  0  0  0  0 70  1  1  4  0  0  0  0  0  0  5  1  0  2  0  0  1  0  0  2  0
                 G  1  1  2  1  3  2 68  1  0  0  0  1  0  0  0  0  4  1  3  2  0  0  0  0  0  0
                 H  0  0  0  1  0  1  0 46  0  2  3  1  1  1  9  0  0  5  0  3  0  2  0  0  1  0
                 I  0  0  0  0  0  0  0  0 65  3  0  0  0  0  0  0  0  0  2  0  0  0  0  2  1  0
                 J  0  1  0  0  0  1  0  0  3 61  0  0  0  0  1  0  0  0  1  0  0  0  0  1  0  4
                 K  0  1  4  0  0  0  0  5  0  0 56  0  0  2  0  0  0  4  0  1  2  0  0  4  0  0
                 L  0  0  0  0  1  0  0  1  0  0  0 63  0  0  0  0  0  0  0  0  0  0  0  0  0  0
                 M  0  0  1  0  0  0  1  0  0  0  0  0 70  2  0  0  0  0  0  0  1  0  6  0  0  0
                 N  0  0  0  0  0  0  0  0  0  0  0  0  0 77  0  0  0  1  0  0  1  0  2  0  0  0
                 O  0  0  1  1  0  0  0  1  0  1  0  0  0  0 49  1  2  0  0  0  1  0  0  0  0  0
                 P  0  0  0  0  0  3  0  0  0  0  0  0  0  0  2 69  0  0  0  0  0  0  0  0  1  0
                 Q  0  0  0  0  0  0  3  1  0  0  0  2  0  0  2  1 52  0  1  0  0  0  0  0  0  0
                 R  0  4  0  0  1  0  0  3  0  0  3  0  0  0  1  0  0 64  0  1  0  1  0  0  0  0
                 S  0  1  0  0  1  1  1  0  1  1  0  0  0  0  0  0  6  0 47  1  0  0  0  1  0  6
                 T  0  0  0  0  1  1  0  0  0  0  1  0  0  0  0  0  0  0  1 83  1  0  0  0  2  2
                 U  0  0  2  1  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0 83  0  0  0  0  0
                 V  0  0  0  0  0  0  0  0  0  0  0  0  1  0  1  0  0  0  0  0  0 64  1  0  1  0
                 W  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  3 59  0  0  0
                 X  0  1  0  0  1  0  0  1  0  0  1  4  0  0  0  0  0  1  0  0  0  0  0 76  1  0
                 Y  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0  0  0  1  0  0  0  1 58  0
                 Z  1  0  0  0  2  0  0  0  0  2  0  0  0  0  0  0  0  0  5  1  0  0  0  0  0 70
```

As seen in the table, most of the predictions matched the actual letters. We can get the evaluation data more succinctly by the following:

```
> agreement <- letter_predictions == letters_test$letter
> table(agreement)
agreement
FALSE  TRUE
  321  1679
```

This tells us that the classification was correct in 1,679 out of our 2000 test records. As before, we'll defer improving model performance until later in the course.

Method #3. Adding regression to trees

(MLR pg 190)

Step 1: Collecting the Data

Our last method will use the whitewines.csv file available for you on Moodle. There is nothing quirky about reading this data into R. The read command and structure of the wine object are shown below.

```
> wine <- read.csv("whitewines.csv")
> str(wine)
'data.frame':  4898 obs. of  12 variables:
 $ fixed.acidity       : num  6.7 5.7 5.9 5.3 6.4 7 7.9 6.6 7 6.5 ...
 $ volatile.acidity    : num  0.62 0.22 0.19 0.47 0.29 0.14 0.12 0.38 0.16 0.
37 ...
 $ citric.acid          : num  0.24 0.2 0.26 0.1 0.21 0.41 0.49 0.28 0.3 0.33
...
 $ residual.sugar      : num  1.1 16 7.4 1.3 9.65 0.9 5.2 2.8 2.6 3.9 ...
 $ chlorides           : num  0.039 0.044 0.034 0.036 0.041 0.037 0.049 0.043
0.043 0.027 ...
 $ free.sulfur.dioxide : num  6 41 33 11 36 22 33 17 34 40 ...
 $ total.sulfur.dioxide: num  62 113 123 74 119 95 152 67 90 130 ...
 $ density             : num  0.993 0.999 0.995 0.991 0.993 ...
 $ pH                  : num  3.41 3.22 3.49 3.48 2.99 3.25 3.18 3.21 2.88 3.
28 ...
 $ sulphates           : num  0.32 0.46 0.42 0.54 0.34 0.43 0.47 0.47 0.47 0.
39 ...
 $ alcohol             : num  10.4 8.9 10.1 11.2 10.9 ...
 $ quality             : int  5 6 6 4 6 6 6 6 6 7 ...
```

Again, there isn't anything particularly quirky about this data on the surface. However, because we will use regression we should check to see if the class variable, quality, follows a normal distribution or is nearly normal. We can do that with the hist() function as follows:

```
> hist(wine$quality)
```

This command will automatically generate a plot in "Plots". I will leave it up to you to answer the question of whether or not this data are normal.

Step 2: Exploring and Preparing the Data

Essentially all we need to do this time is to subset our data into training and test sets. Let's use an approximately 75% for training and 25% for testing split of the observations.

```
> wine_train <- wine[1:3750, ]
> wine_test <- wine[3751:4898, ]
```

Step 3: Training a Model on the Data

To create this model we'll use the rpart package. "rpart" stands for recursive partitioning. Using the rpart function we'll set-up our command with the formula syntax and the class variable quality as follows:

```
> m.rpart <- rpart(quality ~ ., data=wine_train)
```

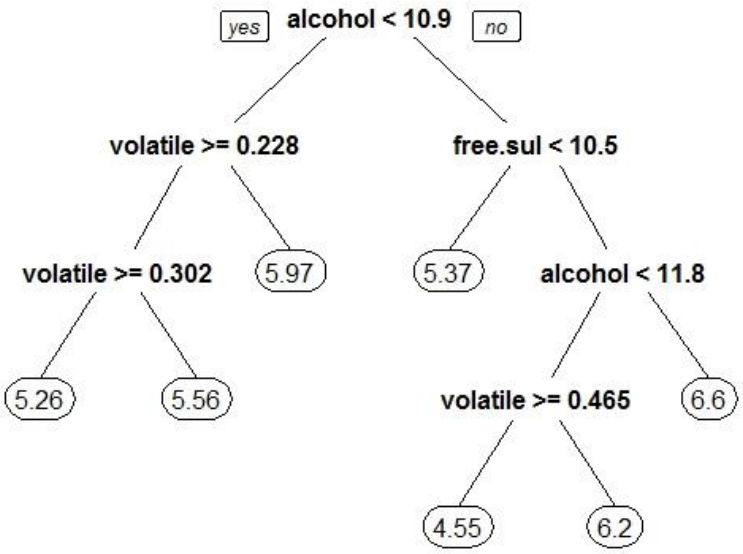
To get the basic information about the tree just type in the object name m.rpart at the command prompt:

```
> m.rpart
n= 3750

node), split, n, deviance, yval
      * denotes terminal node

 1) root 3750 2945.53200 5.870933
   2) alcohol< 10.85 2372 1418.86100 5.604975
     4) volatile.acidity>=0.2275 1611  821.30730 5.432030
       8) volatile.acidity>=0.3025 688  278.97670 5.255814 *
       9) volatile.acidity< 0.3025 923  505.04230 5.563380 *
     5) volatile.acidity< 0.2275 761  447.36400 5.971091 *
   3) alcohol>=10.85 1378 1070.08200 6.328737
     6) free.sulfur.dioxide< 10.5 84   95.55952 5.369048 *
     7) free.sulfur.dioxide>=10.5 1294  892.13600 6.391036
      14) alcohol< 11.76667 629  430.11130 6.173291
        28) volatile.acidity>=0.465 11   10.72727 4.545455 *
        29) volatile.acidity< 0.465 618  389.71680 6.202265 *
      15) alcohol>=11.76667 665  403.99400 6.596992 *
```

Note that the rpart function automatically determined that the most important predictor was the variable "alcohol". The root split between <10.85 and >=10.85 as shown above. Nodes with an * are terminal or leaf nodes.

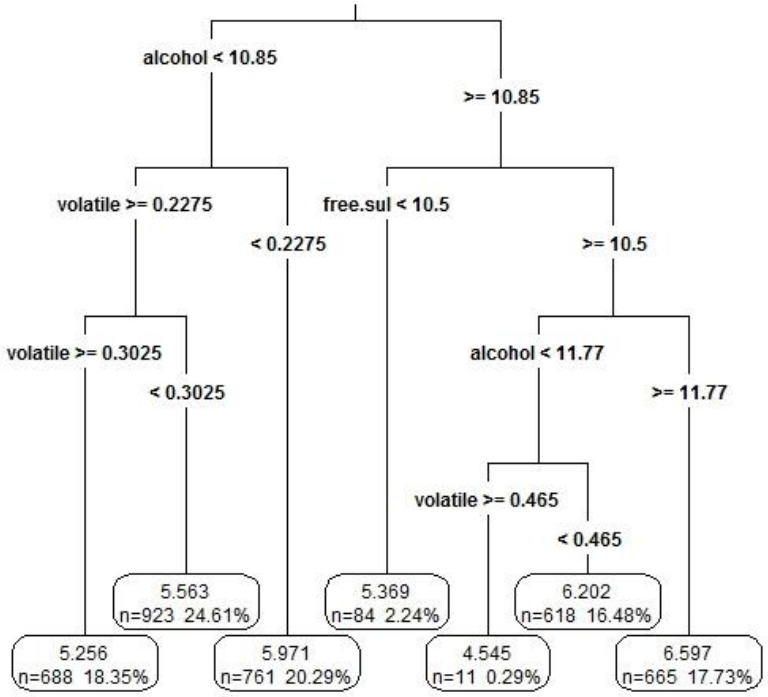
If we want to actually plot the tree we need to install and attach the rpart.plot package. Once that is finished the command is (I've also pasted a picture of the tree below the command):

```
> rpart.plot(m.rpart, digits=3)
```



Another, very different way to visualize the tree is using the command:

```
> rpart.plot(m.rpart, digits=4, fallen.leaves = TRUE, type = 3, extra = 101)
```



Step 4: Evaluating Model Performance

In this case we'll look at summary statistics to evaluate our model's performance.

```
> summary(p.rpart)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.545   5.563   5.971   5.893   6.202   6.597
> summary(wine_test$quality)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   5.000   6.000   5.901   6.000   9.000
```

The fact that the model's range is much smaller than the data's actual range suggests that the model is not adequately capturing the extremes of the range of quality, i.e. the very good or very bad wines. We can also check the correlation:

```
> cor(p.rpart, wine_test$quality)
[1] 0.5369525
```

A 54% correlation is ok, but not great. Again, we'll defer our discussion of improving model performance until later in the term.