HarvardX - PH125.9x your own project

Joe Mark Lippert

07/12/2020

Introduction

I selected the `abalone` dataset that can be downloaded at UCI Repository.

The objective of this assignment is to predict the age of abalone from physical measurements. The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings identified through a microscope. Other measurements, which are easier to obtain, are used to predict the age. The objective is a classification task.

The variables are:

1. sex (Male, Female, Infant)
2. length (measured in mm)
3. diameter (mm)
4. height (mm)
5. whole_weight (grams)
6. shucked_weight (grams)
7. viscera_weight (grams)
8. shell_weight (grams)
9. total_rings (indicates age)

Variables 1 - 8 are the features and, variable 9 is the outcome. The dataset contains 4177 items. In data science speak this number is referred to as n and the number of features p.

I shall use Machine Learning to predict the outcomes. Machine learning "deals, directly or indirectly, with estimating the regression function, also called the conditional mean."[Norm Matloff]

Methods/Analysis

A look in Let's begin by looking at the data. With any dataset it is always a good idea to take a look around. Identify the features and output, then strategise an approach to preprocessing.

```r
# look at the data type of the variables
glimpse(abalone)
```

```
## Rows: 4,177
## Columns: 9
## $ sex            <chr> "M", "M", "F", "M", "I", "I", "F", "F", "M", "F", "F...
## $ length         <dbl> 0.455, 0.350, 0.530, 0.440, 0.330, 0.425, 0.530, 0.5...
## $ diameter       <dbl> 0.365, 0.265, 0.420, 0.365, 0.255, 0.300, 0.415, 0.4...
## $ height         <dbl> 0.095, 0.090, 0.135, 0.125, 0.080, 0.095, 0.150, 0.1...
## $ whole_weight   <dbl> 0.5140, 0.2255, 0.6770, 0.5160, 0.2050, 0.3515, 0.77...
## $ shucked_weight <dbl> 0.2245, 0.0995, 0.2565, 0.2155, 0.0895, 0.1410, 0.23...
## $ viscera_weight <dbl> 0.1010, 0.0485, 0.1415, 0.1140, 0.0395, 0.0775, 0.14...
## $ shell_weight   <dbl> 0.150, 0.070, 0.210, 0.155, 0.055, 0.120, 0.330, 0.2...
## $ total_rings    <int> 15, 7, 9, 10, 7, 8, 20, 16, 9, 19, 14, 10, 11, 10, 1...
```

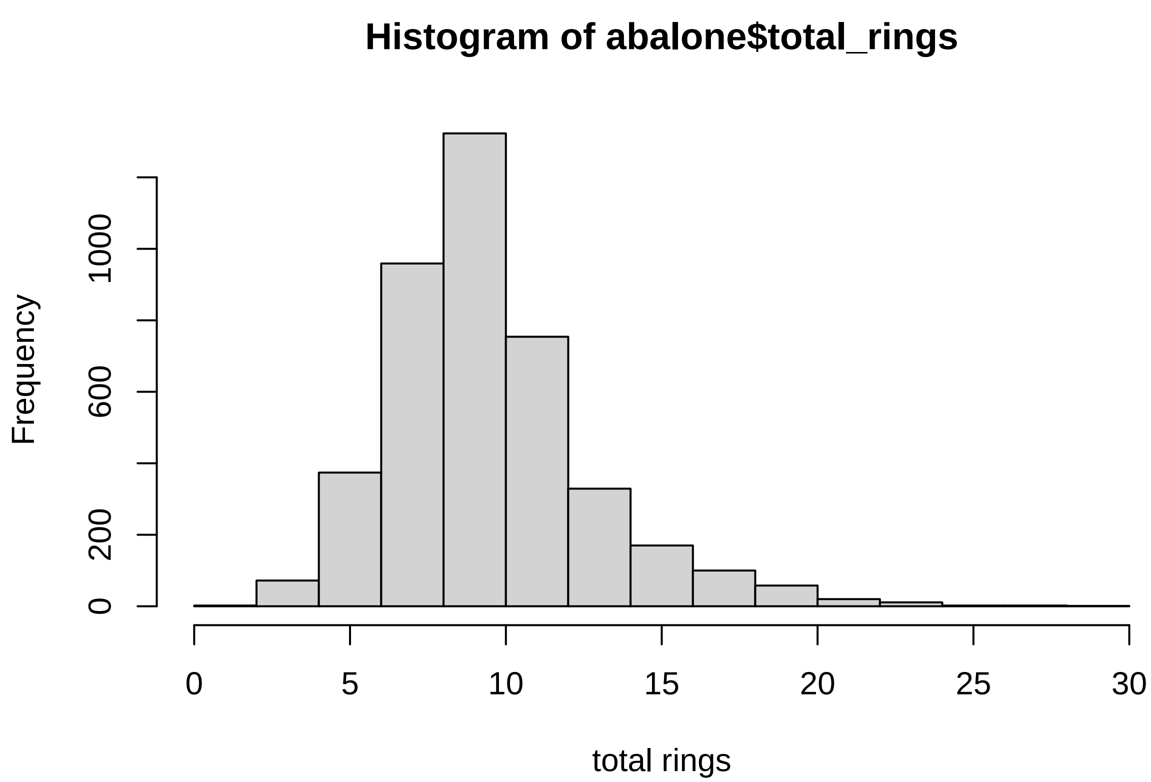
```
# now let's look at the first 10 items of the dataset
abalone %>% as_tibble()
```

```
## # A tibble: 4,177 x 9
##    sex   length diameter height whole_weight shucked_weight viscera_weight
##    <chr>  <dbl>    <dbl>  <dbl>        <dbl>          <dbl>          <dbl>
##  1 M      0.455    0.365  0.095        0.514         0.224          0.101
##  2 M      0.35     0.265  0.09         0.226         0.0995         0.0485
##  3 F      0.53     0.42   0.135        0.677         0.256          0.142
##  4 M      0.44     0.365  0.125        0.516         0.216          0.114
##  5 I      0.33     0.255  0.08         0.205         0.0895         0.0395
##  6 I      0.425    0.3    0.095        0.352         0.141          0.0775
##  7 F      0.53     0.415  0.15         0.778         0.237          0.142
##  8 F      0.545    0.425  0.125        0.768         0.294          0.150
##  9 M      0.475    0.37   0.125        0.509         0.216          0.112
## 10 F      0.55     0.44   0.15         0.894         0.314          0.151
## # ... with 4,167 more rows, and 2 more variables: shell_weight <dbl>,
## #   total_rings <int>
```

```
# then look at the distribution of the classes in the outcome vector
table(abalone$total_rings)
```

```
##
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
##   1   1  15  57 115 259 391 568 689 634 487 267 203 126 103  67  58  42  32  26
##  21  22  23  24  25  26  27  29
##  14   6   9   2   1   1   2   1
```

```
# and again in a histogram
hist(abalone$total_rings, xlab = "total rings")
```



```r
# lastly, (certainly not least), check for NAs
sum(is.na(abalone))
```

```
## [1] 0
```

First approach Let's begin with a common sense approach by predicting age (total_rings) from a single feature.

The feature I will use first is the one that is most closely correlated with total_rings. That feature is shell_weight. See below the correlation calculation.

```r
cor(abalone[2:9])
```

```
##                   length  diameter    height whole_weight shucked_weight
## length         1.0000000 0.9868116 0.8275536    0.9252612      0.8979137
## diameter       0.9868116 1.0000000 0.8336837    0.9254521      0.8931625
## height         0.8275536 0.8336837 1.0000000    0.8192208      0.7749723
## whole_weight   0.9252612 0.9254521 0.8192208    1.0000000      0.9694055
## shucked_weight 0.8979137 0.8931625 0.7749723    0.9694055      1.0000000
## viscera_weight 0.9030177 0.8997244 0.7983193    0.9663751      0.9319613
## shell_weight   0.8977056 0.9053298 0.8173380    0.9553554      0.8826171
## total_rings    0.5567196 0.5746599 0.5574673    0.5403897      0.4208837
##                viscera_weight shell_weight total_rings
## length              0.9030177    0.8977056   0.5567196
## diameter            0.8997244    0.9053298   0.5746599
## height              0.7983193    0.8173380   0.5574673
## whole_weight        0.9663751    0.9553554   0.5403897
## shucked_weight      0.9319613    0.8826171   0.4208837
## viscera_weight      1.0000000    0.9076563   0.5038192
## shell_weight        0.9076563    1.0000000   0.6275740
## total_rings         0.5038192    0.6275740   1.0000000
```

Now assume we foraged an abalone along the False Bay coast in South Africa with shell weight 497 grams (0.497kg), and we want to predict its age. How would we proceed? We'd most likely look at what the total rings are for a few abalone, in our dataset, with shell weight closest to 497 grams and calculate the average number of rings of the selected items.

Let's look at the 5 'nearest' shell_weights and calculate the average of the selected items.

```r
options(scipen=999)
shell <- abalone$shell_weight
dists <- abs(shell - 0.497) # distances of shell_weight closest to 0.497
close5 <- order(dists)[1:5]

# The 5 closest distances to shell_weight of 0.497 are:
dists[close5]
```

```
## [1] 0.0000 0.0005 0.0010 0.0010 0.0010
```

```r
# and the 5 corresponding closest shell_weights are:
abalone$shell_weight[close5]
```

```
## [1] 0.4970 0.4975 0.4980 0.4980 0.4960
```

```r
# the total_rings for each of these 5 closest items are:
abalone$total_rings[close5]
```

```
## [1] 11 11 12 13 10
```

```r
# and the mean total_rings for the 5 closest shell weights is:
mean(abalone$total_rings[close5])
```

```
## [1] 11.4
```

When we decided to look at the 5 closest shell weights, the decision to select the 5 closest shell weights as opposed to 20, was arbitrary. In Machine Learning (ML) k denotes the arbitrary number (5) we chose. These 5 are called the **k** nearest neigbours. So how do we decide what the best **k** is? There is no sure way to choose the best **k**. Various methods are used in practice that work well. A rule of thumb derived from mathematical theory is:

$$k < \sqrt{n}$$

where n is the number of items (rows) in your dataset.

The regtools package/kNN() function Now let's predict total_rings with the regtools package function kNN() again assuming we foraged an abalone in the shallows and recorded all the features' measurements.

The kNN(**x**, **y**, **newx**, **k**) function takes the following basic arguments:

- **x**: the X matrix for the the training set. It has to be a matrix because 'nearest-neigbour' distances between rows must be computed.
- **y**: the Y vector for the training set.
- **newx**: a vector of feature values for a new case or a matrix of vectors.
- **k**: the number of nearest neighbours we wish to use.

But first, convert the sex variable to a dummy since this is a regtools package requirement.

```r
abalone <- abalone %>% mutate_if(is.character, as.factor)
aBalone <- factorsToDummies(abalone, omitLast = TRUE) # factorsToDummies() coerces the data.frame to a 

# Look at the column names after converting the 'sex' feature to dummy.
colnames(aBalone)
```

```
##  [1] "sex.F"          "sex.I"          "length"         "diameter"
##  [5] "height"         "whole_weight"   "shucked_weight" "viscera_weight"
##  [9] "shell_weight"   "total_rings"
```

```r
# Observe that the number of variables has increased from 9 to 10.
dim(aBalone)
```

```
## [1] 4177   10
```

```r
# Create the X matrix to be the training set.
abalone.x <- aBalone[, 1:9]

# And the Y vector for the training set.
age <- aBalone[, 10]

# Now let's begin using kNN() by predicting the rings for one abalone (some random abalone).
knnout <- kNN(abalone.x, age, c(0, 0, 0.35, 0.39, 0.09, 0.46, 0.2, 0.11, 0.12), 5)

# Here we see the 5 positions (row numbers) of the predictions.
knnout$whichClosest
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,] 2205 3154    1   12 3837
```

```r
# The output below shows us the average of the 5 predictions. It is the regression estimate.
knnout$regests
```

```
## [1] 10
```

```r
# Below we access the total_rings that corresponds with the indexes shown above.
aBalone[c(2205, 3154, 1, 12, 3837), 10]
```

```
## [1]  7  9 15 10  9
```

```r
# And confirm the average.
mean(aBalone[c(2205, 3154, 1, 12, 3837), 10])
```

```
## [1] 10
```

kNN() and prediction using the original dataset Let's continue to demonstrate usage of the kNN() function by predicting the rings for the original dataset. To begin, let's set **k** = 5.

```r
knnout <- kNNallK(abalone.x, age, abalone.x, 5, allK = TRUE)

# Here we see the structure of the new object which is a list.
str(knnout)
```

```
## List of 8
##  $ whichClosest  : int [1:4177, 1:5] 1 2 3 4 5 6 7 8 9 10 ...
##  $ regests       : num [1:5, 1:4177] 15 11.5 11 10.2 9.8 ...
##  $ mhdists       : num [1:4177] 5.42 6.36 4.43 5.62 3.81 ...
##  $ scaleX        : logi TRUE
##  $ xcntr         : Named num [1:9] 0.313 0.321 0.524 0.408 0.14 ...
##   ..- attr(*, "names")= chr [1:9] "sex.F" "sex.I" "length" "diameter" ...
##  $ xscl          : Named num [1:9] 0.4637 0.467 0.1201 0.0992 0.0418 ...
##   ..- attr(*, "names")= chr [1:9] "sex.F" "sex.I" "length" "diameter" ...
##  $ leave1out     : logi FALSE
##  $ startAt1adjust: num 0
##  - attr(*, "class")= chr "kNNallk"
```

```r
# The matrix below shows that we generate 60 sets of 4177 predictions. Each row has 60 nearest neighbou
# Below we show the nearest neigbour predictions for only the first 3 rows of the prediction dataset.
knnout$whichClosest[1:3, 1:5]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1 2575 2144 2099   36
## [2,]    2 3174 2249  607 3316
## [3,]    3 2465 2333  460 2024
```

```r
# See the predicted values (Y) using the original data. These are the regression estimates.
# Row 1 has all the predicted values for k=1, row 2 shows the predicted values for k = 2, etc.
knnout$regests[1:3, 1:5]
```

```
##      [,1]     [,2]      [,3]      [,4]     [,5]
## [1,] 15.0 7.000000  9.000000 10.000000 7.000000
## [2,] 11.5 6.500000 10.000000  8.500000 6.500000
## [3,] 11.0 6.333333  9.666667  8.666667 6.666667
```

```r
# The first 5 real Y values are shown below
age[1:5] # the predicted values for k=1 is the same as this
```

```
## [1] 15  7  9 10  7
```

Notice that the entries of the first row of the regression estimates (regests) are identical to the real values (age[1:5]). Because we set both **x** and **newx** to aBalone.x, we observe that the first element in the first column 'says' that the closest row in aBalone.x to the first row in aBalone.x is the first row in aBalone.x. So the closest data point is itself. This is called overfitting. And overfitting should be avoided like the plague. How do we achieve this? We leave out **k** = 1. The kNN() function has an argument that permits us to do that. The argument is called leave1out. See its use below.

Now use k = 60 We finally settle on a value for k that is less than $\sqrt{n}$. 60 is a little less than $\sqrt{n}$.

We also use the function that evaluates the prediction accuracy. The function is findOverallLoss(). For each value in Y (age), we take the absolute difference between the actual value and predicted values, then compute the average for those absolute differences to get the Mean Absolute Prediction Error (MAPE).

```r
# Here we run the function but exclude k = 1
knnout <- kNNallK(abalone.x, age, abalone.x, 60, allK = TRUE, leave1out = TRUE)

# Now see the predicted row positions for first 3 items.
knnout$whichClosest[1:3, 1:60]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,] 2575 2144 2099   36 2513 1571 4157  952 2572   112  3846  3138  3090    61
## [2,] 3174 2249  607 3316 2452 2421 4121 2376  714   630   638   768   138   610
## [3,] 2465 2333  460 2024 3178 2066  404  115 3661  3120  3957  3490   280  3121
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
## [1,]  2151   725  2134  3276  2055   407  2054  2012  2145  1094  3154  2205
## [2,]  3206  2430    21  2393   212   304   636  2230  3399   644  3326  3906
## [3,]  2133  2384  1313  1304    14   640   108   738  2320  3313   203  3330
##      [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]
## [1,]  3955   318  3879  1464  2646    12   740  3539   118   147  2126  2064
## [2,]   715  3922  2392   303   621   140  3365   325   707   124  3835   641
## [3,]   197  3099  3275   564   474  3951    56  2301  3868  3451   471  4169
##      [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]
## [1,]    66  2887  3817   798    57  2053  2019   837  1461    20  2208  2186
## [2,]   635   609   461   300  2482  3372  3405  2104   618  2169    19  2132
## [3,]   597  2319  3875  2187   472  2756  3094    60  2429  3188  2127   109
##      [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60]
## [1,]  2259  2463  2292   114  3092  3551  2298  2739    64  1775
## [2,]   637  3379   712   530  3179   329    40  4163  3924   519
## [3,]     8   962   423   787    11  3291   984  2374   123  3352
```

```r
# An see the regression estimates for the first 3 items
knnout$regests[1:3, 1:60]
```

```
##          [,1] [,2]     [,3]     [,4] [,5]     [,6] [,7]     [,8]      [,9]
## [1,] 8.000000    6 11.00000 7.000000  6.0 8.000000 10.0 13.00000 10.000000
## [2,] 9.000000    6 10.00000 8.000000  6.5 7.500000 11.5 18.00000  9.500000
## [3,] 8.666667    7 10.33333 8.666667  6.0 7.333333 12.0 15.66667  9.666667
##         [,10]    [,11] [,12]    [,13]    [,14]    [,15] [,16]    [,17] [,18]
## [1,]  9.00000 16.00000   8.0 15.00000 11.00000  8.00000    10 8.000000    11
## [2,] 12.50000 14.50000   8.5 14.50000 12.50000  8.50000    11 7.500000    10
## [3,] 14.33333 12.66667   8.0 13.33333 13.33333 11.33333    12 7.333333     9
##         [,19]    [,20] [,21]    [,22]    [,23] [,24]    [,25] [,26] [,27]
## [1,] 9.000000 7.000000   7.0 7.000000  9.00000     8 8.000000  17.0    16
## [2,] 9.000000 7.000000   7.5 7.000000  9.00000     9 9.000000  13.5    12
## [3,] 9.333333 9.333333   7.0 6.666667 12.33333     9 8.666667  12.0    13
##         [,28]    [,29]    [,30] [,31]    [,32]    [,33]    [,34]    [,35] [,36]
## [1,] 11.00000 10.00000 8.000000    13 15.00000 11.00000 12.00000 12.00000   8.0
## [2,] 10.50000 12.00000 8.500000    15 14.00000 10.50000 14.00000 11.50000   8.5
## [3,] 10.33333 11.33333 8.333333    13 15.33333 10.33333 12.66667 11.33333   8.0
```

```
##         [,37] [,38]    [,39] [,40] [,41] [,42] [,43] [,44] [,45]    [,46] [,47]
## [1,]  8.00000   9.0 8.000000     7   9.0  10.0     5   5.0     6 7.000000    10
## [2,] 11.00000   7.5 8.000000     6   9.5  10.5     5   4.5     5 7.000000    10
## [3,] 10.33333   8.0 8.333333     7  10.0  10.0     5   4.0     5 7.333333    10
##      [,48] [,49] [,50]    [,51]     [,52]    [,53]     [,54]    [,55]    [,56]
## [1,]  10.0   6.0    14 9.000000  9.000000 6.000000 10.000000 9.000000  9.00000
## [2,]   9.5   6.5    12 8.500000 10.000000 7.000000  9.000000 8.000000  9.50000
## [3,]   9.0   7.0    12 8.666667  9.666667 7.666667  8.666667 7.666667 11.33333
##          [,57] [,58] [,59] [,60]
## [1,]  9.000000     8   6.0    10
## [2,] 10.500000     9   5.5     9
## [3,]  9.666667     9   5.0    10
```

```
# Below see the loss error for each k value. The Mean Absolute Prediction Error (MAPE)
findOverallLoss(knnout$regests, age)
```

```
##  [1] 2.027771 1.803926 1.708962 1.655973 1.609385 1.588620 1.573925 1.558535
##  [9] 1.549810 1.554226 1.550177 1.546205 1.541500 1.534013 1.529120 1.528848
## [17] 1.529975 1.529287 1.529214 1.530045 1.531619 1.529665 1.529234 1.529417
## [25] 1.530285 1.531408 1.529850 1.531097 1.530483 1.531211 1.531451 1.531991
## [33] 1.534057 1.534510 1.537098 1.537720 1.537836 1.538979 1.540303 1.541788
## [41] 1.544433 1.545538 1.546080 1.547479 1.547453 1.547267 1.548067 1.548270
## [49] 1.550107 1.552119 1.553620 1.554295 1.556105 1.557653 1.558339 1.558466
## [57] 1.559711 1.560368 1.561517 1.562505
```

```
# The optimal k is:
which.min(findOverallLoss(knnout$regests, age))
```

```
## [1] 16
```

```
# And the minimum loss error is:
min(findOverallLoss(knnout$regests, age))
```

```
## [1] 1.528848
```

The partykit package Here we will predict the number of rings using decision trees (DT). DT are basically flow charts. Like kNN, they look at the neighbourhood of the point to be predicted, only in a more sophisticated way.

So, to reiterate, the DT method sets up the prediction process as a flow chart. At the top of the tree we split the data into 2 parts. Then we split each of those parts into 2 further parts, etc,. An alternative name for this process is *recursive partitioning*.

First we dummify the data. The argument fullRank is used. The end result is that we produce 2 dummy variables for the 'sex' variable as opposed to 3.

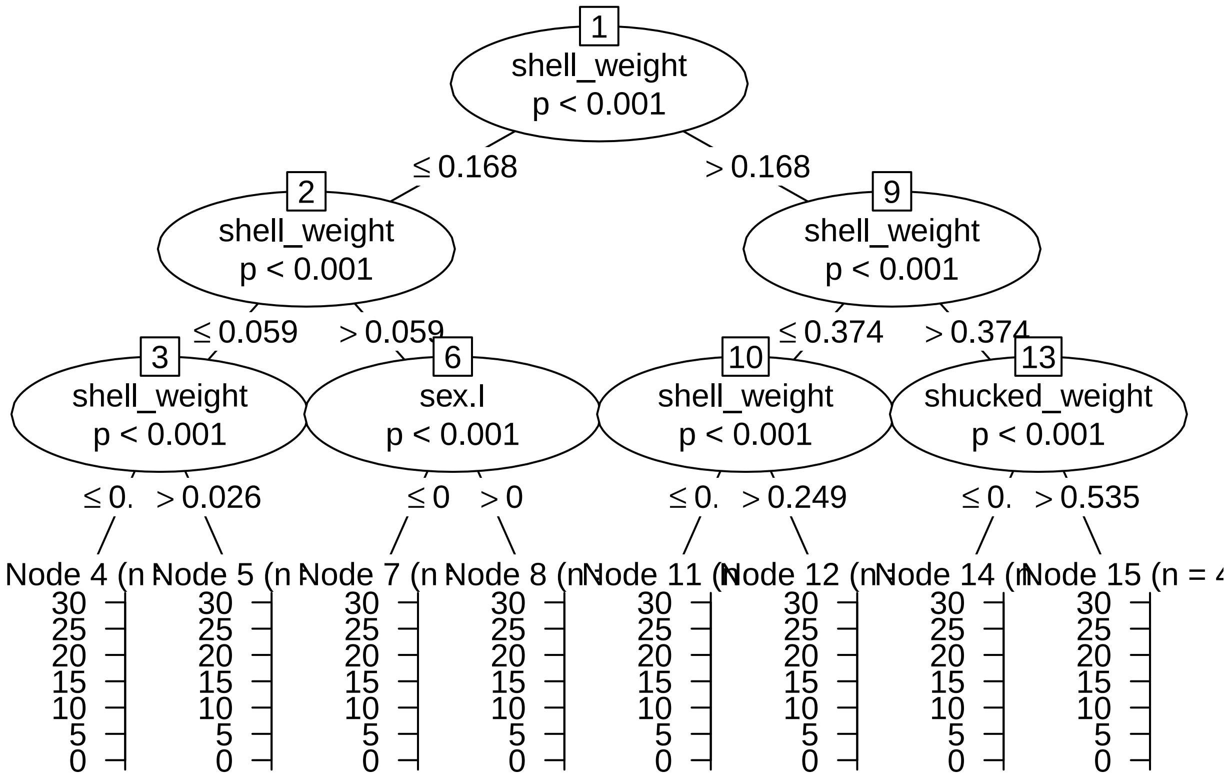
```
dmy <- dummyVars("~ .", data = abalone, fullRank = TRUE)
abalone <- data.frame(predict(dmy, newdata = abalone))
```

```
abba <- ctree(total_rings ~.,
              data = abalone,
              control = ctree_control(maxdepth = 3))
```

The plot The plot (below) shows that the DT does take the form of a flow chart. The plot says: For an abalone within a given level of shell_weight, sex, shucked_weight, etc, what value should we predict for total_rings? The graph shows the prediction procedure:

1. For our first cut in predicting the total rings, look to shell_weight. If it is less than or equal to 168 grams, go to the left branch of the flow chart, otherwise go right. WE have split Node 1 of the tree.
2. If you are on the right branch, continue to look at shell_weight. If the shell_weight is at most 374 grams, continue to look at shell_weight. And if the shell_weight is greater than 249 grams, select Node 12 where you will see that your total_rings prediction is approximately 11.
3. If you choose the left branch from Node 1 there will again be a decision based on shell_weight comparing it to 59 grams. If the shell_weight is greater than 59 grams we look to sex.I (infant) and wind up in either Node 7 or Node 8. If the shell_weight is less than 59 grams, we look again to shell_weight and compare it to 26 grams. If it is at most 26 grams we slect Node 4 else we select Node 5.

```
plot(abba)
```



Nodes 4, 5, 7, 8, 11, 12, 14, 15 are terminal nodes. They do not split. We can access these programmatically as seen below.

```
# Terminal Nodes
nodeids(abba, terminal = TRUE)
```

```
## [1]  4  5  7  8 11 12 14 15
```

In order to compute the predicted value for total_rings in say, Node 14, we would ordinarily use the **predict()** function. We can though examine the output object **abba**.

The display shows the number of original data points ending up in each terminal node, the mean squared prediction error for those points as well as the mean prediction value (Y) in each of the nodes. So, by example, Node 4has a prediction value of 4.458 total_rings and a mean squared error value of 123.3 for the 118 points in Node 4.

```
# Printed version of the output.
abba
```

```
##
## Model formula:
```

```
## total_rings ~ sex.I + sex.M + length + diameter + height + whole_weight +
##     shucked_weight + viscera_weight + shell_weight
##
## Fitted party:
## [1] root
## |   [2] shell_weight <= 0.1675
## |   |   [3] shell_weight <= 0.0585
## |   |   |   [4] shell_weight <= 0.026: 4.458 (n = 118, err = 123.3)
## |   |   |   [5] shell_weight > 0.026: 6.284 (n = 243, err = 455.4)
## |   |   [6] shell_weight > 0.0585
## |   |   |   [7] sex.I <= 0: 9.051 (n = 412, err = 1665.9)
## |   |   |   [8] sex.I > 0: 7.647 (n = 654, err = 1827.4)
## |   [9] shell_weight > 0.1675
## |   |   [10] shell_weight <= 0.3745
## |   |   |   [11] shell_weight <= 0.249: 9.955 (n = 840, err = 4760.3)
## |   |   |   [12] shell_weight > 0.249: 11.112 (n = 1250, err = 9112.3)
## |   |   [13] shell_weight > 0.3745
## |   |   |   [14] shucked_weight <= 0.535: 14.882 (n = 161, err = 2498.8)
## |   |   |   [15] shucked_weight > 0.535: 12.148 (n = 499, err = 4325.0)
##
## Number of inner nodes:    7
## Number of terminal nodes: 8
```

Results The results for the prediction model are as seen below. See the results for the mean prediction value and median prediction value for each node.

```
# First see the node terminals.
nodeIDs <- nodeids(abba, terminal = TRUE)

# Then the median Y for each node.
mdn <- function(yvals, weights) median(yvals)
predict_party(abba, id=nodeIDs, FUN = mdn)
```

```
##  4  5  7  8 11 12 14 15
##  4  6  9  7  9 10 14 11
```

```
# And finally the mean Y for each node
predict_party(abba, id=nodeIDs, FUN = function(yvals, weights) mean(yvals))
```

```
##         4         5         7         8        11        12        14        15
##  4.457627  6.283951  9.050971  7.646789  9.954762 11.112000 14.881988 12.148297
```

If you wish to see the predicted values for all the data points for an individual node then the code below applies. Select the individual terminal node (id) to observe the predicted results for the data points:

```
# The Y values in a given node. Select the node number and insert in the id argument.
f1 <- function(yvals, weights) c(yvals)
predict_party(abba, id=4, FUN = f1)
```

```
## $`4`
##   [1] 5 5 4 4 5 4 5 4 1 3 3 5 5 4 4 3 4 5 6 5 6 5 5 3 5 4 4 3 7 4 7 5 5 4 4 6 4
##  [38] 2 3 5 6 5 6 5 3 4 6 4 3 4 4 4 4 5 7 4 5 5 3 5 5 4 4 4 5 5 4 3 4 6 5 6 6 6
##  [75] 3 5 5 6 5 5 4 4 4 5 4 4 3 4 4 5 4 4 4 4 5 6 5 5 4 5 4 5 4 3 4 3 4 4 3 4 4
## [112] 4 4 6 5 6 4 5
```

Or if you wish to see the median of a selected node use the following code:

```r
# The median of the Y values of a specific node.
mdn <- function(yvals, weights) median(yvals)
predict_party(abba, id=15, FUN = mdn)
```

```
## 15
## 11
```

Below you see the code to predict the total_rings of a new item.

```r
# Here's how to predict the total_rings for a recently retrieved abalone.
newx <- data.frame(sex.I=0,sex.M=1,length=0.495,diameter=0.503,height=0.137,
                   whole_weight=0.5347, shucked_weight=0.372, viscera_weight=0.301,
                   shell_weight=0.267)
predict(abba,newx,FUN=mdn)
```

```
##  1
## 10
```

Conclusion This report presents the results of the prediction of the age, indicated by rings, of abalone by using the **kNN()** function of the **regtools** package, and the **ctree()** function of the **partykit** package. A loss function is presented in the **kNN()** function section and the mean squared prediction error is calculated for the points in each node.

As to impact, I am unsure, but I humbly assume that the approach could be useful to the marine and fisheries regulatory authorities in South Africa and possibly to the marine research communities.

There are obvious limitations. I think more effective Machine Learning algorithms should produce even better results. I am curious about the Random Forests function of the **partykit** package. Hopefully when I become better skilled in ML and ML mathematics I can apply alternative ML algorithms. I found the loss calculations not as intuitive as the RMSE loss calculation for the **kNN()** and **ctree()** functions.