7BUIS008W Data Mining & Machine Learning - Coursework 2

"Show evidence of understanding of the clustering and modelling concepts, through the implementation of requested algorithms using real datasets. Implementation is performed in R environment, while students need to perform some critical evaluation of their results."

Andrew Keats

8th January 2017

Table of Contents

Task 1: Data Set Selection and Visualisation

Premise

> You need to select a data set of your own choice (i.e. you may use a dataset already used before in the lab, or from the literature review) for the purposes of building training and validating the above type of classifiers 1-3. With the aid of R package visualise and justify the properties of the selected data set.

Picking the data set

The first task was to find out where to look for datasets, most sources of advice referred to the UCI repository, other sources often reused this data and hosted it in a new format, like Kaggle. After looking online for suitable datasets, specifically for the problem of classification, 2 datasets were possible candidates for study; between the Taiwanese Credit Card dataset and the Abalone shellfish dataset, I opted for the Abalone dataset on the basis that if has fewer variables and fewer instances, so it should be easier to work with without so much preparation; the credit card data would require more considered cleansing of outliers and potentially pruning of variables considered as well as PCA for dimensionality reduction.

The Abalone dataset has a little over 4,000 instances compared to 30,000 for the Credit Card dataset, 8 attributes for Abalone compared to 24 for the Credit Card data. Hopefully the dimensionality of the Abalone dataset should lend itself to less data cleansing because the variables are concrete characteristics of the animals:

- Sex
- Length
- Diameter
- Height
- Whole weight
- Shucked weight
- Viscera weight
- Shell weight
- Rings

How to classify this data

> Before you start working with data for a machine learning project, it is vital to understand what the data is, and what we want to achieve. Without it, we have no basis from which to make our decisions about what data is relevant as we clean and prepare our data. - (Devlin, 2016)

The typical way to classify this data, is to determine approximate age based on the ring value, according to the UCI page for this dataset, the rings roughly suggest age such that $y = r + 2.5$, where y is number of years and r is number of rings. The simplest approach is to attempt to guess the correct number of rings for a test instance based on other attributes.

Getting the data

Having downloaded the data from the UCI repository (Hussain, no date, Brownlee (2016a)) we need convert the data into a table and check it's usable.

```
abalone_data = data.frame(read.table("../assets/data/abalone.data", sep =
","))
abalone_attr_names = c("sex", "length", "diameter", "height", "weight.whole",
"weight.shucked", "weight.viscera", "weight.shell", "rings")
colnames(abalone_data) = abalone_attr_names

#We need to check there are no missing values, as if any instances are
incomlete we will need to remove
missing_vals = sum(is.na(abalone_data))
print(c("Number of missing values:", missing_vals), quote = FALSE)
```

```
## [1] Number of missing values: 0
```

Here's a glance at the dataset.

```
kable(head(abalone_data))
```

sex	length	diameter	height	weight.whole	weight.shucked	weight.viscera	weight.shell	rings
M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7
I	0.425	0.300	0.095	0.3515	0.1410	0.0775	0.120	8

```
abalone_summary = summary(abalone_data)
kable(abalone_summary)
```

sex	length	diameter	height	weight.whole	weight.shucked	weight.viscera	weight.shell	rings
F:1307	Min. :0.075	Min. :0.0550	Min. :0.0000	Min. :0.0020	Min. :0.0010	Min. :0.0005	Min. :0.0015	Min. : 1.000
I:1342	1st Qu.:0.450	1st Qu.:0.3500	1st Qu.:0.1150	1st Qu.:0.4415	1st Qu.:0.1860	1st Qu.:0.0935	1st Qu.:0.1300	1st Qu.: 8.000
M:1528	Median :0.545	Median :0.4250	Median :0.1400	Median :0.7995	Median :0.3360	Median :0.1710	Median :0.2340	Median : 9.000
NA	Mean :0.524	Mean :0.4079	Mean :0.1395	Mean :0.8287	Mean :0.3594	Mean :0.1806	Mean :0.2388	Mean : 9.934

sex	length	diameter	height	weight.whole	weight.shucked	weight.viscera	weight.shell	rings
NA	3rd Qu.:0.615	3rd Qu.:0.4800	3rd Qu.:0.1650	3rd Qu.:1.1530	3rd Qu.:0.5020	3rd Qu.:0.2530	3rd Qu.:0.3290	3rd Qu.:11.000
NA	Max. :0.815	Max. :0.6500	Max. :1.1300	Max. :2.8255	Max. :1.4880	Max. :0.7600	Max. :1.0050	Max. :29.000

The attributes explained

The Abalone is a type of shellfish and each attribute describes a characteristic of the animal. Not being an expert on this animal, I've had to research a little bit about the Abalone in order to get an idea of what these traits are. Knowing a bit more about these properties, should hopefully lead to a better understanding of the data. What follows is a list of definitions for each column.

sex
M (male), F (female), or I (infant). Presumably, an infant Abalone lacks a sex or it's too hard to identify

length
Longest shell measurement, in millimetres

diameter
Perpendicular measurement to measured length, in millimetres

height
Height of abalone, including body, in millimetres

weight.whole
Weight of whole abalone, in grams

weight.shucked
Weight of meat alone, in grams

weight.viscera
Weight of guts and other non meat alone, in grams

weight.shell
weight of shell alone, after drying, in grams

rings
Number of rings in the shell, strongly correlated to age but with a delta

A little bit of basic data cleansing

The data was checked to see if there were any missing values but it's also worth checking if some values are technically impossible and therefore belonging to incorrect entries that need to be removed; to do this, all values are iterated over to check for assignments of 0, indicating bad data (Jolly, 2017).

```
abalone_numeric_attr <- abalone_attr_names[which(abalone_attr_names!="sex")]

for (abalone_attr in abalone_numeric_attr)
{
bad_vals = abalone_data[abalone_data[abalone_attr] == 0, ]
#col_data = unlist(abalone_data[abalone_attr])
if (nrow(bad_vals) > 0)
{
print(abalone_attr)
print(abalone_data[abalone_data[abalone_attr] == 0, ])
}
```

```
}
```

```
## [1] "height"
## sex length diameter height weight.whole weight.shucked weight.viscera
## 1258 I 0.430 0.34 0 0.428 0.2065 0.0860
## 3997 I 0.315 0.23 0 0.134 0.0575 0.0285
## weight.shell rings
## 1258 0.1150 8
## 3997 0.3505 6
```

```
rm(bad_vals)
```

It appears that there are two height values that are incorrect, so we should remove these instances from our dataset.

```
abalone_data$height[abalone_data$height==0] = NA
abalone_data = na.omit(abalone_data)
```

It's also worth examining the weight data, to ensure that the total data is not less than the combined values to the other weight values.

```
abalone_data_bad_weight = abalone_data[(abalone_data$weight.whole -
(abalone_data$weight.shucked + abalone_data$weight.viscera +
abalone_data$weight.shell)) < 0,]
```

```
kable(head(abalone_data_bad_weight))
```

	sex	length	diameter	height	weight.whole	weight.shucked	weight.viscera	weight.shell	rings
43	I	0.240	0.175	0.045	0.0700	0.0315	0.0235	0.020	5
44	I	0.205	0.150	0.055	0.0420	0.0255	0.0150	0.012	5
45	I	0.210	0.150	0.050	0.0420	0.0175	0.0125	0.015	4
46	I	0.390	0.295	0.095	0.2030	0.0875	0.0450	0.075	7
47	M	0.470	0.370	0.120	0.5795	0.2930	0.2270	0.140	9
61	M	0.450	0.345	0.105	0.4115	0.1800	0.1125	0.135	7

```
print(paste(c("Number of instances where total weight is less than constiuent
weights:", nrow(abalone_data_bad_weight)), sep = ""), quote = FALSE)
```

```
## [1] Number of instances where total weight is less than constiuent weights:
## [2] 154
```

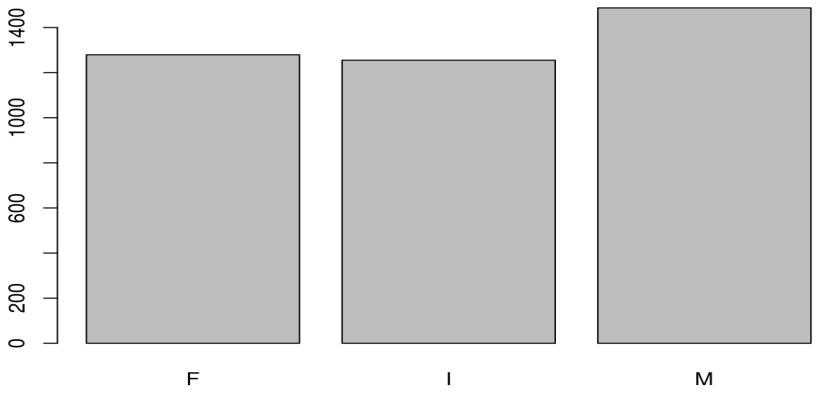
It would appear that the data isn't completely sanitized, with this sort of erroneous data entry needing to be handled as well. The simplest option is to rule out these instances as well.

```
abalone_data <- abalone_data[!(abalone_data$weight.whole -
(abalone_data$weight.shucked + abalone_data$weight.viscera +
abalone_data$weight.shell)) < 0,]
```

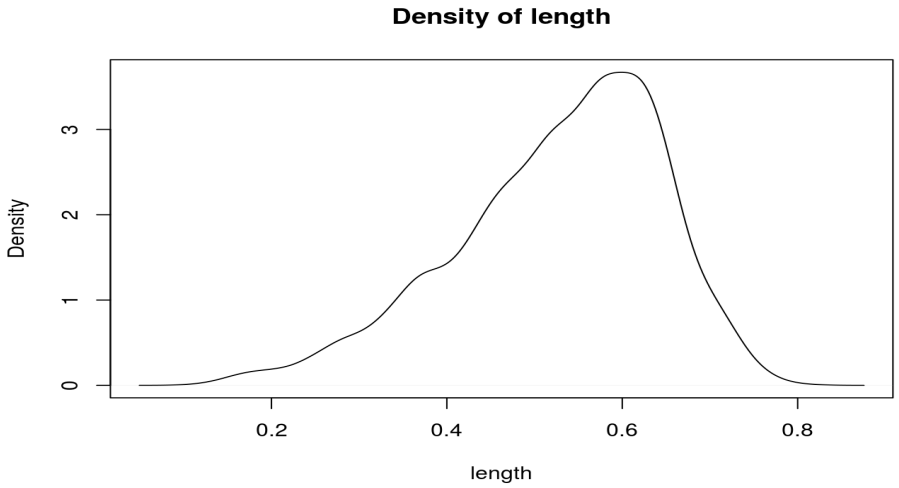
A graphical look at the attributes

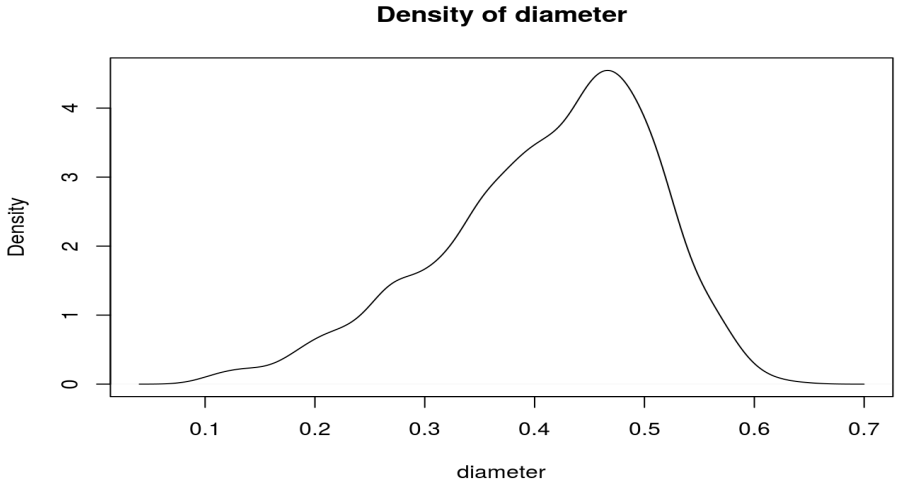
Below is an examination of each attribute to see if there are any obvious outliers that might need to be removed

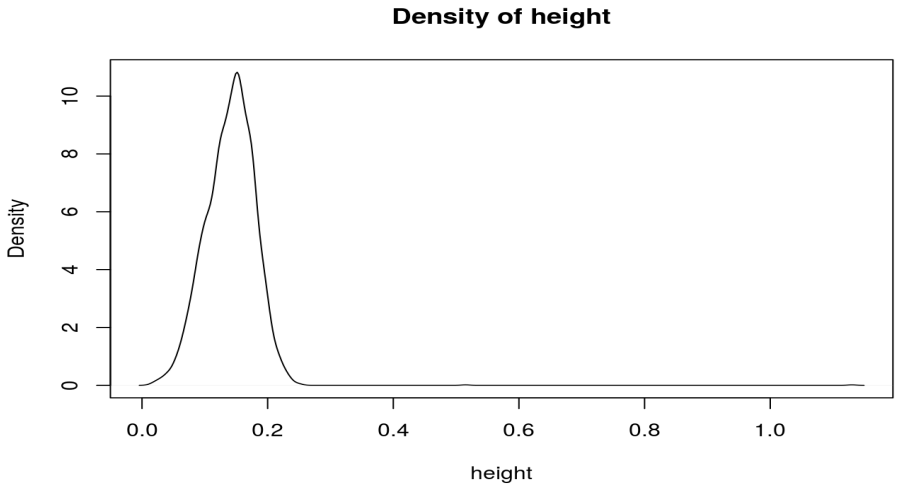
```
#boxplot(scale(abalone_data), main="Looking at the data graphically",
xlab="Abalone Attributes", ylab="Values")

plot(abalone_data$sex)
```

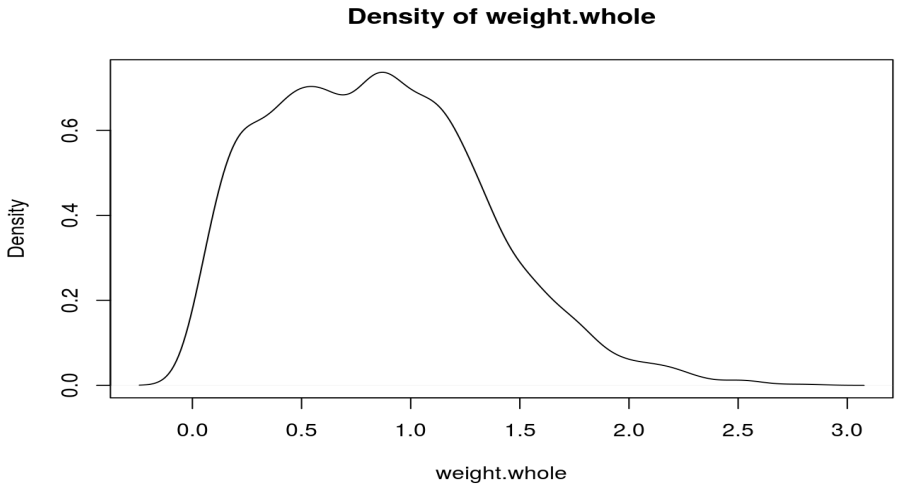


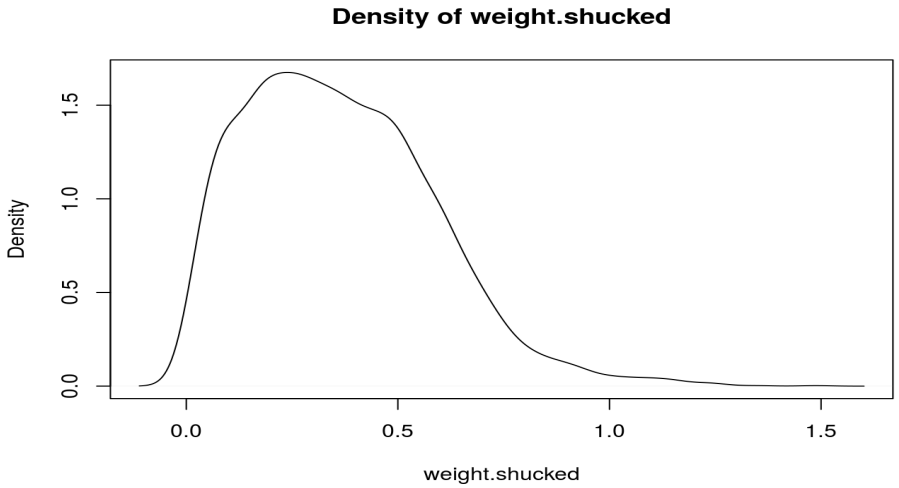

```
for (abalone_attr in abalone_numeric_attr)
{
#print(abalone_attr)
col_data = unlist(abalone_data[abalone_attr])
#print(col_data)
plot(density(col_data), xlab=abalone_attr, main=paste(c("Density of ",
abalone_attr), collapse = ''))
}
```

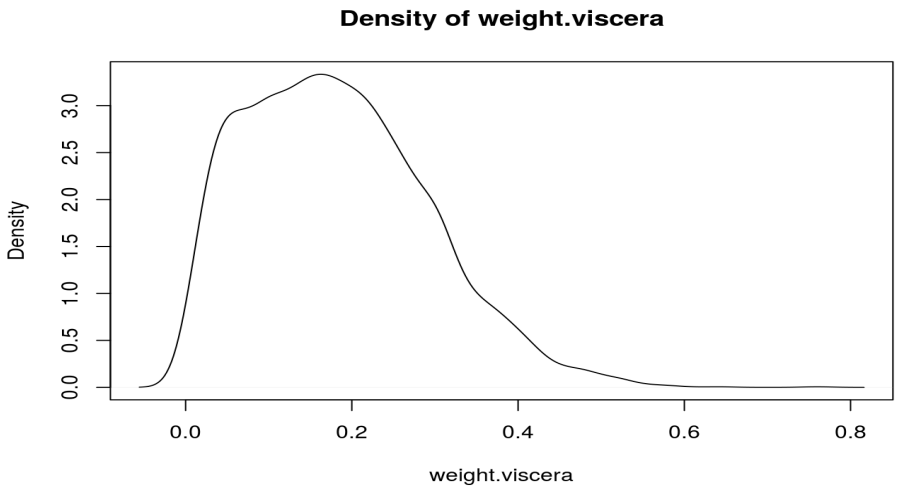

Density of length
Density
0
1
2
3
0.2
0.4
0.6
0.8
length

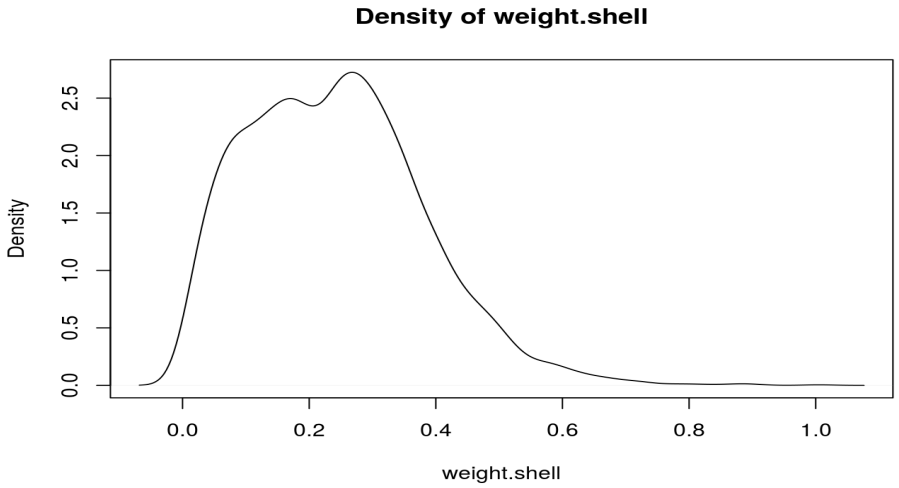

Density of diameter
Density
0
1
2
3
4
0.1
0.2
0.3
0.4
0.5
0.6
0.7
diameter

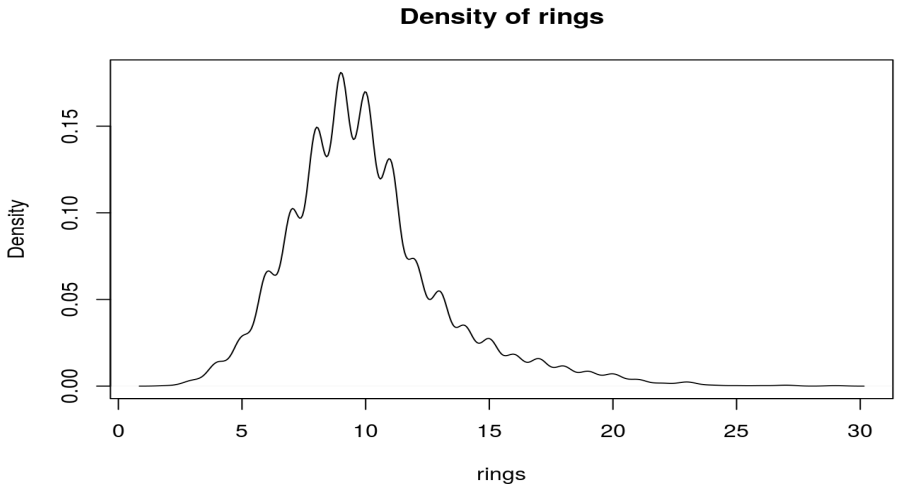

Density of height
Density
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0
height


Density of weight.whole
Density
0.0
0.2
0.4
0.6
0.0
0.5
1.0
1.5
2.0
2.5
3.0
weight.whole


Density of weight.shucked
Density
0.0
0.5
1.0
1.5
0.0
0.5
1.0
1.5
weight.shucked


Density of weight.viscera
Density
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.0
0.2
0.4
0.6
0.8
weight.viscera






```
rm(abalone_attr)
rm(col_data)
```

In order to work out which attributes should be considered to have valid outliers, a heuristic approach has been used to look at the distance between the uppermost outliers for each attribute and its nearest neighbour.

```
#Create a list to populate with our tail neighbour distances
tail_deltas <- c()

abalone_data_no_sex = abalone_data[, -1]

for (attrib in abalone_data_no_sex) {
#get the last two values
data_tails <- tail(sort(attrib),2)
#push the delta on to our list
tail_deltas <- c(tail_deltas, diff(data_tails))
}

#grab out attribute keys to include in our new table/frame
attributes <- names(abalone_data_no_sex)

#make a new dataframe from
dataframe <- data.frame(attributes = attributes, tail_neighbour_d=tail_deltas)

#get the order for the nearest neighbour starting with the greatest distance
and descending
```

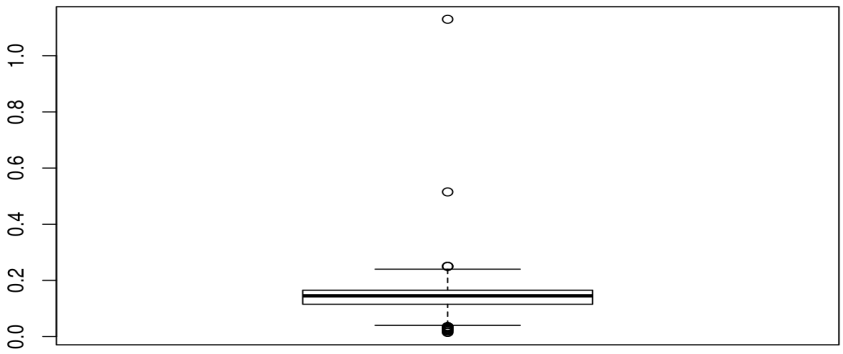
```
neighbour_order <- order(dataframe$tail_neighbour_d, decreasing=TRUE)

#now apply the order to the frame
sorted_attributes_by_neighbour_d <- dataframe[ neighbour_order, ]
sorted_attributes_by_neighbour_d
```

```
## attributes tail_neighbour_d
## 8 rings 2.0000
## 3 height 0.6150
## 5 weight.shucked 0.1395
## 6 weight.viscera 0.1185
## 7 weight.shell 0.1080
## 4 weight.whole 0.0460
## 2 diameter 0.0200
## 1 length 0.0150
```

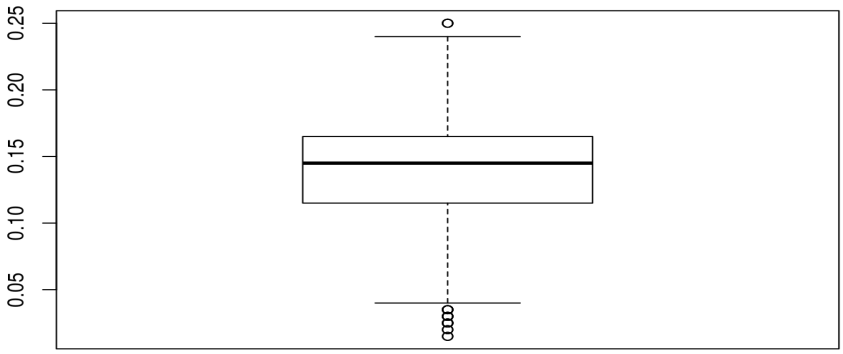
While rings is at the top of the leader-board regarding the delta, it's important to take into account that this data isn't scaled; it's very likely that this outlier is the results of a particularly lucky Abalone that managed to live longer than the rest of the long tail through some luck. Given that the other values are meant to be in grams and millimetres, it's reasonable to compare values like for like in this instance. As such the weight deltas are comparable and can be excluded from outlier cleansing, with the same applying to diameter and length; height seems to be anomalous though and will need further attention.

It's easier to see on a box-plot that one value is way out far from any neighbours, with its nearest neighbour also having no nearby neighbour; we could probably benefit just from removing the two farthest outliers.

```
boxplot(abalone_data$height)
```



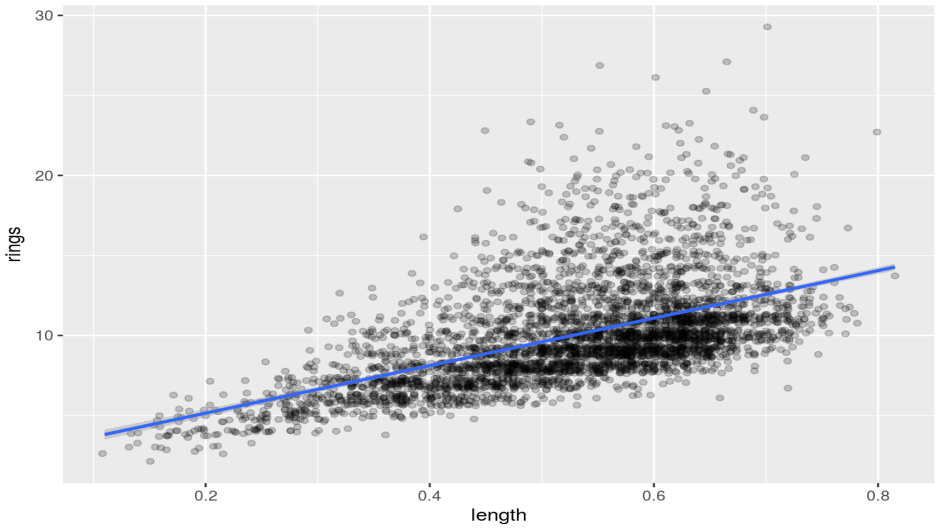

```
abalone_data_cleansed <- abalone_data[ abalone_data$height < .5, ]
boxplot(abalone_data_cleansed$height)
```



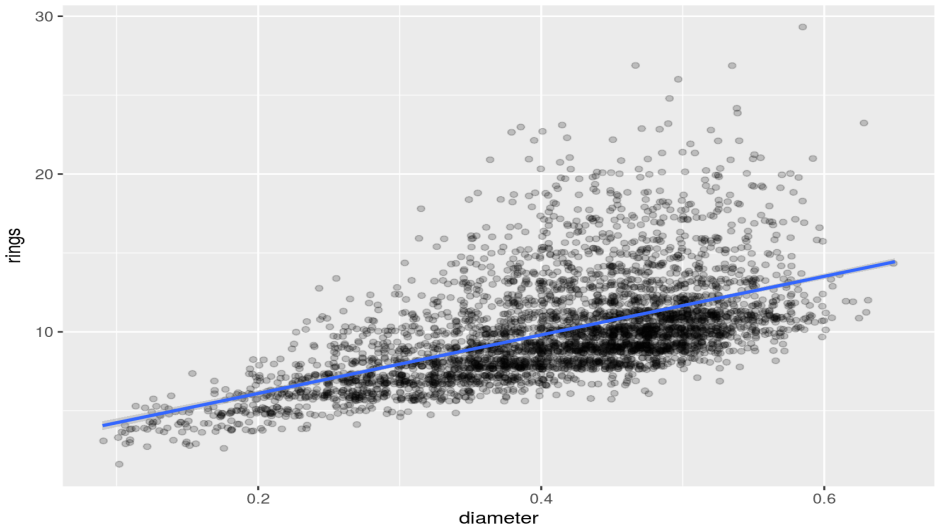
Correlation between the ‘rings’ attribute and attributes pertaining to length or mass.

Seeing as the intent is to see if classification can be used to determine approximate age from physical attributes (aside from rings), it’s worth looking for existing correlations between the attributes and the number of rings.

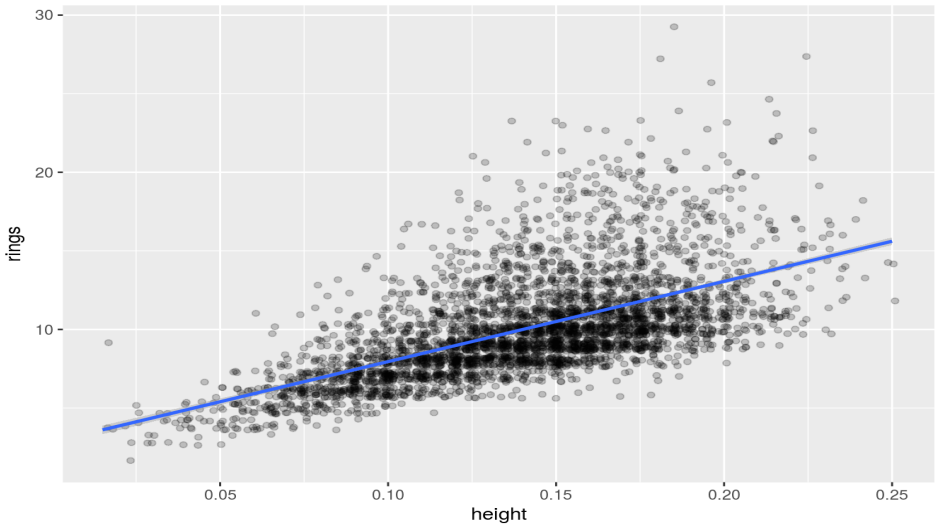
```
qplot(x = length,
y = rings,
data = abalone_data_cleansed,
alpha = I(0.2), # alpha to help convery density
geom = "jitter") + # jitter so points don't stack so much
geom_smooth(method = lm)
```



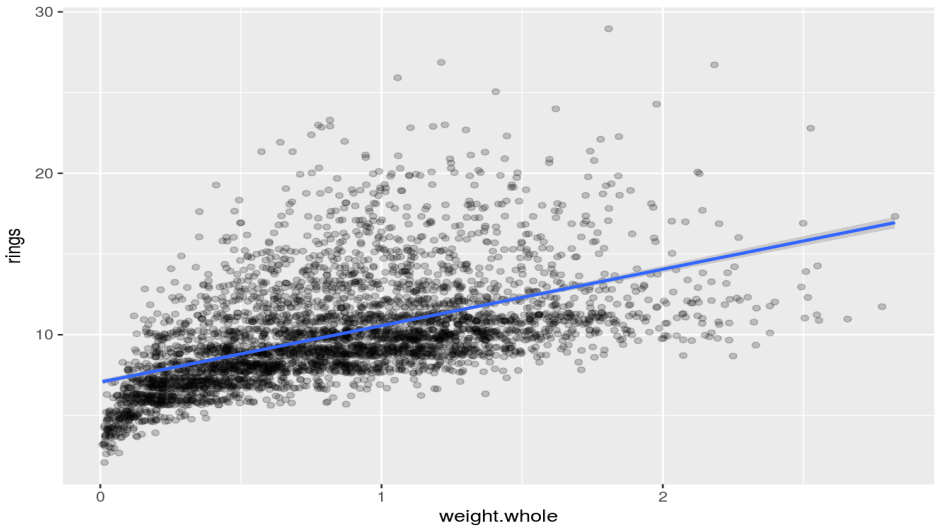

```
qplot(x = diameter,
y = rings,
data = abalone_data_cleansed,
alpha = I(0.2), # alpha to help convery density
geom = "jitter") + # jitter so points don't stack so much
geom_smooth(method = lm)
```



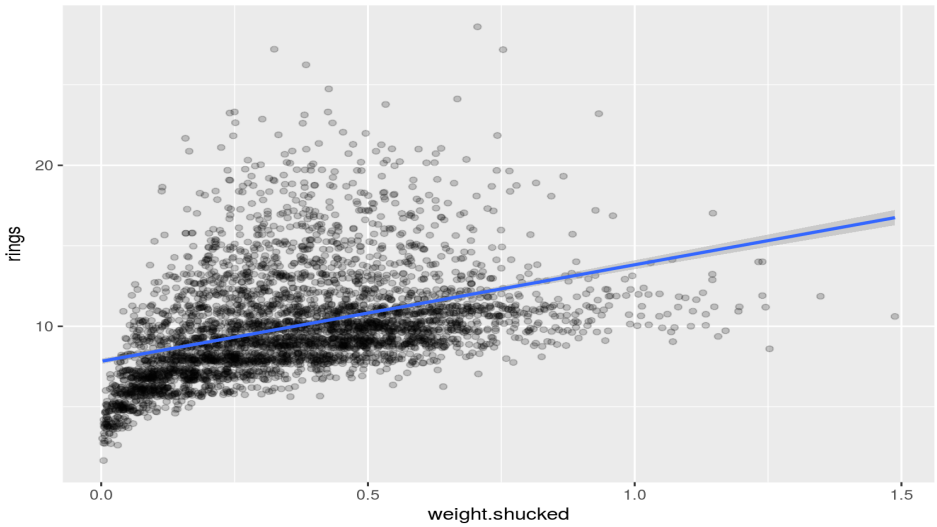

```
qplot(x = height,
y = rings,
data = abalone_data_cleansed,
alpha = I(0.2), # alpha to help convery density
geom = "jitter") + # jitter so points don't stack so much
geom_smooth(method = lm)
```



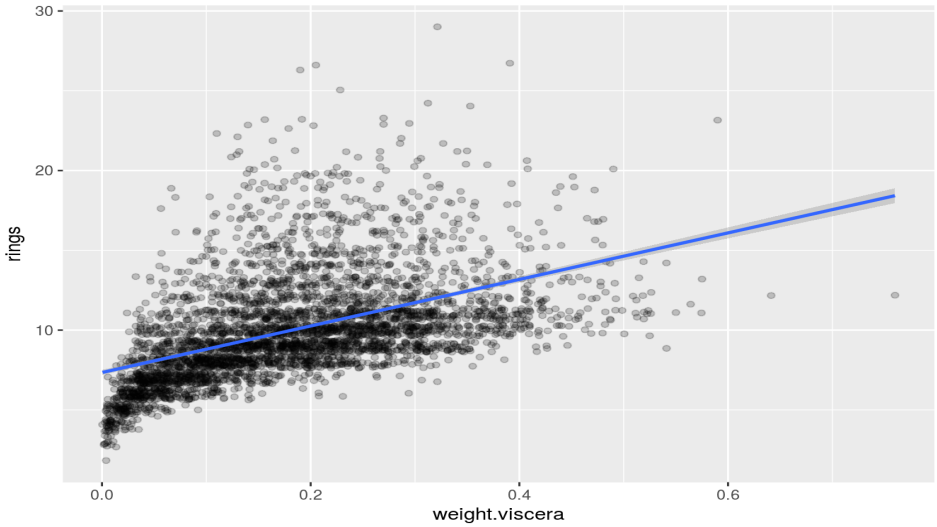

```
qplot(x = weight.whole,
y = rings,
data = abalone_data_cleansed,
alpha = I(0.2), # alpha to help convery density
geom = "jitter") + # jitter so points don't stack so much
geom_smooth(method = lm)
```



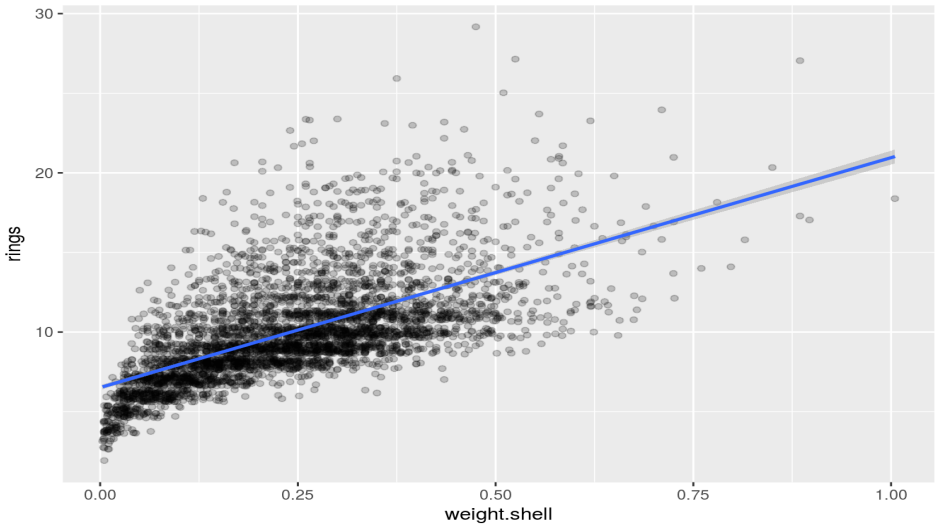

```
qplot(x = weight.shucked,
y = rings,
data = abalone_data_cleansed,
alpha = I(0.2), # alpha to help convery density
geom = "jitter") + # jitter so points don't stack so much
geom_smooth(method = lm)
```




```
qplot(x = weight.viscera,
y = rings,
data = abalone_data_cleansed,
alpha = I(0.2), # alpha to help convery density
geom = "jitter") + # jitter so points don't stack so much
geom_smooth(method = lm)
```




```
qplot(x = weight.shell,
y = rings,
data = abalone_data_cleansed,
alpha = I(0.2), # alpha to help convery density
geom = "jitter") + # jitter so points don't stack so much
geom_smooth(method = lm)
```



Picking data attributes to select

With our cleansed data we can see that there is an evident correlation between all of these attributes and the number of rings but in particular those relating to physical size show the strongest relationship as the points best match the line of best fit; the weight values are distributed in such a way that there's a little curve away from the line as you reach 0 on both axes and they also seem to diverge more from the line as the dimension values increase. Should any attributes be selected as ones to work with, discarding the others, it would be:

- length
- diameter
- height
- weight.whole

Given that the all attributes appear to display a fairly linear relationship to the ring count we can use r-squared, otherwise known as the Coefficient of Determination to verify how well the data matches the line of best fit (Frost, no date, Frost (no date)).

```
abalone.lm_length <- lm(data = abalone_data_cleansed, formula = rings ~
length)
abalone.lm_diameter <- lm(data = abalone_data_cleansed, formula = rings ~
diameter)
abalone.lm_height <- lm(data = abalone_data_cleansed, formula = rings ~
height)
abalone.lm_weight.whole <- lm(data = abalone_data_cleansed, formula = rings ~
weight.whole)
abalone.lm_weight.shucked <- lm(data = abalone_data_cleansed, formula = rings
~ weight.shucked)
abalone.lm_weight.viscera <- lm(data = abalone_data_cleansed, formula = rings
~ weight.viscera)
abalone.lm_weight.shell <- lm(data = abalone_data_cleansed, formula = rings ~
weight.shell)

abalone.r_squareds <- c(
summary(abalone.lm_length)$adj.r.squared,
summary(abalone.lm_diameter)$adj.r.squared,
summary(abalone.lm_height)$adj.r.squared,
summary(abalone.lm_weight.whole)$adj.r.squared,
summary(abalone.lm_weight.shucked)$adj.r.squared,
summary(abalone.lm_weight.viscera)$adj.r.squared,
summary(abalone.lm_weight.shell)$adj.r.squared
)

abalone_numeric_attr_no_rings <-
abalone_numeric_attr[which(abalone_numeric_attr!="rings")]

#make a new dataframe from
dataframe.rsquareds <- data.frame(attributes = abalone_numeric_attr_no_rings,
r_squared=abalone.r_squareds)

#get the order for the nearest neighbour starting with the greatest distance
and descending
rsquareds_order <- order(dataframe.rsquareds$r_squared, decreasing=TRUE)

#now apply the order to the frame
sorted_attributes_by_r_squared <- dataframe.rsquareds[ rsquareds_order, ]
sorted_attributes_by_r_squared



## attributes r_squared
## 7 weight.shell 0.3855553
## 3 height 0.3604153
## 2 diameter 0.3155304
## 1 length 0.2950611
## 4 weight.whole 0.2806291
## 6 weight.viscera 0.2439019
## 5 weight.shucked 0.1681050



#abalone.lm_length
```

After looking at these results it may be wiser to consider using the shell weight alone if necessary; the r-squared value isn't a perfect predictor of the fitness but perhaps in this case, it could make more sense to use a weight value that could fluctuate less in nature. Selection of the attributes relating to size appear to be validated.

Substantiating the selection

With the use of a correlation matrix, it's possible to further validate which attributes could be worth focusing on:

```
# calculate correlation matrix
correlationMatrix <- cor(abalone_data_cleansed[,2:9])
# summarize the correlation matrix
print(correlationMatrix)
```

```
## length diameter height weight.whole weight.shucked
## length 1.0000000 0.9861689 0.8974579 0.9259423 0.9003873
## diameter 0.9861689 1.0000000 0.9040452 0.9260010 0.8952472
## height 0.8974579 0.9040452 1.0000000 0.8873269 0.8361649
## weight.whole 0.9259423 0.9260010 0.8873269 1.0000000 0.9709880
## weight.shucked 0.9003873 0.8952472 0.8361649 0.9709880 1.0000000
## weight.viscera 0.9034261 0.8997239 0.8656499 0.9670606 0.9323660
## weight.shell 0.8977314 0.9055333 0.8897232 0.9561399 0.8830020
## rings 0.5433567 0.5618726 0.6004786 0.5299133 0.4102585
## weight.viscera weight.shell rings
## length 0.9034261 0.8977314 0.5433567
## diameter 0.8997239 0.9055333 0.5618726
## height 0.8656499 0.8897232 0.6004786
## weight.whole 0.9670606 0.9561399 0.5299133
## weight.shucked 0.9323660 0.8830020 0.4102585
## weight.viscera 1.0000000 0.9070347 0.4940547
## weight.shell 0.9070347 1.0000000 0.6210541
## rings 0.4940547 0.6210541 1.0000000
```

```
# find attributes that are highly corrected (ideally &gt;0.75)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.5)

abalone.correlation <- data.frame(correlation=correlationMatrix[,8])
#get rid of rings, that's obviously going to be max correlation to itself!
abalone.correlation <- data.frame(attributes=rownames(abalone.correlation)
[1:7], 'correlation to rings'=abalone.correlation[1:7, 0:1])

#get the order for correlations
correlation_order <- order(abalone.correlation$correlation, decreasing=TRUE)

#now apply the order to the frame
sorted_correlation_order <- abalone.correlation[ correlation_order, 1:2]
print(sorted_correlation_order)
```

```
## attributes correlation.to.rings
## 7 weight.shell 0.6210541
## 3 height 0.6004786
## 2 diameter 0.5618726
## 1 length 0.5433567
## 4 weight.whole 0.5299133
## 6 weight.viscera 0.4940547
## 5 weight.shucked 0.4102585
```

The correlation between ring values and the other attributes, when ordered, actually matches the order of the r-squared values; so much so, that weight.shell should be considered the primary correlate of them all, followed by height.

Creating a simpler classification value

Given that the number of rings is actually a range of integers from 1 to 29, to use each individual possible integer presents a problem; firstly as the ring value cannot be considered continuous it’s not really sensible to treat it as such, thus converting to a numeric floating range is not appropriate; in addition to this, the dataset might not have instances that cover every single possible ring value between 1 and 29, which will cause inaccuracy in some of the models.

An alternative dataset can be created that mitigates this issue by creating an approximate age factor; classification of the Abalone can be more loosely done against age ranges, like *young, middle, old* with the test instances still being compared by other attributes and mapped to one of these age group factor values. The split will be determined by the ring/age spread of the sampled population of Abalones, such that about a 1/3 of the sample size is in each age_group. This is because the number of Abalones with rings above 12 starts to drop off quite acutely and as a percentage even those above 15, are a small minority despite the highest values nearing 30 rings.

```
summary(abalone_data_cleansed$rings)
```

```
## Min. 1st Qu. Median Mean 3rd Qu. Max.
## 2 8 10 10 11 29
```

```
quantile(abalone_data_cleansed$rings, c(1/3, 2/3))
```

```
## 33.33333% 66.66667%
## 9 11
```

```
abalone_age_groups <- cut(abalone_data_cleansed$rings, breaks=c(-Inf, 9, 12, Inf), labels=c("young","middle","old"))

summary(abalone_age_groups)
```

```
## young middle old
## 1978 1357 684
```

```
abalone_data_cleansed_age_groups <- subset(abalone_data_cleansed, select=-rings)
abalone_data_cleansed_age_groups$age_group <- abalone_age_groups
rm(abalone_age_groups)
```

Task 2: Formation of Training and Test Sets

Premise

> Assuming we have collected one large dataset of already-classified instances, you need to look at three methods of forming training and test sets from this single dataset in R as described below. • The holdout method • Cross-validation • Leave-one-out cross-validation

Holdout datasets

The holdout method is the most basic separation of the dataset into training and testing data. This method just creates one division, albeit randomly selecting those values from across those dataset rather than in a linear fashion.

```
set.seed(4321)
kable(head(abalone_data_cleansed))
```

sex	length	diameter	height	weight.whole	weight.shucked	weight.viscera	weight.shell	rings
M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7
I	0.425	0.300	0.095	0.3515	0.1410	0.0775	0.120	8

```
holdout.train_indeces <- createDataPartition(y =
abalone_data_cleansed$weight.shell, p= 2/3, list = FALSE)
head(holdout.train_indeces, n=20)
```

```
## Resample1
## [1,] 1
## [2,] 2
## [3,] 3
## [4,] 4
## [5,] 6
## [6,] 7
## [7,] 8
## [8,] 9
## [9,] 10
## [10,] 13
## [11,] 15
## [12,] 16
## [13,] 20
## [14,] 21
## [15,] 22
## [16,] 23
```

```
## [17,] 24
## [18,] 25
## [19,] 26
## [20,] 28
```

```
holdout.training <- abalone_data_cleansed[holdout.train_indeces,]

save(holdout.training, file="holdout.training.rda")

holdout.testing <- abalone_data_cleansed[-holdout.train_indeces,]

save(holdout.testing, file="holdout.testing.rda")
```

Evaluating the holdout datasets

By looking at the dimensionality it's possible to confirm that the new datasets are of the correct size

```
dim(holdout.training)
```

```
## [1] 2681 9
```

```
dim(holdout.testing)
```

```
## [1] 1338 9
```

Holdout set for age group factor dataset

As an addition, the dataset modified to have an age_group factor attribute instead of rings attribute of integers, will also be partitioned for use with the holdout method.

```
kable(head(abalone_data_cleansed_age_groups))
```

sex	length	diameter	height	weight.whole	weight.shucked	weight.viscera	weight.shell	age_group
M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	old
M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	young
F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	young
M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	middle
I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	young
I	0.425	0.300	0.095	0.3515	0.1410	0.0775	0.120	young

```
holdout_age_groups.train_indeces <- createDataPartition(y =
abalone_data_cleansed_age_groups$weight.shell, p= 2/3, list = FALSE)
head(holdout_age_groups.train_indeces, n=20)
```

```
## Resample1
## [1,] 1
## [2,] 2
```

```
## [3,] 3
## [4,] 4
## [5,] 6
## [6,] 8
## [7,] 9
## [8,] 12
## [9,] 13
## [10,] 14
## [11,] 15
## [12,] 16
## [13,] 17
## [14,] 19
## [15,] 21
## [16,] 22
## [17,] 23
## [18,] 24
## [19,] 25
## [20,] 26
```

```
holdout_age_groups.training <-
abalone_data_cleansed_age_groups[holdout.train_indeces,]

holdout_age_groups.testing <- abalone_data_cleansed_age_groups[-
holdout.train_indeces,]
```

Cross-validation datasets

Cross-validation methods are variations on the holdout method (Brownlee, 2014a, Gupta (2017)). The k-folds cross-validation in particular is an extended holdout method whereby the dataset is chunked into smaller fragments (where the value of k is the fragment count), called ‘folds’ which are each in turn used as the test subset while the remaining folds make up the training subset; in this way, the training is carried out several times over the same dataset, rotating the role of the ‘folds’ such that every instance will be used several times as training data and once as test data.

This form of training makes better use of a small sample size and helps even out any biases that might occur from just taking one partition for training and another for testing. This averaging out of the training and testing, also happens to benefit larger datasets too, so it is generally considered superior to the basic holdout method.

Repeated k-fold Cross-validation

Repeated k-fold cross-validation takes the technique yet another step further, by splitting the dataset into k folds, repeatedly such that different fragmentation occurs each time; that is to say that even though the number of divisions are the same, each repetition creates a different set of subsets. By doing so, this sort of shuffling further economically reuses the dataset for training purposes.

For the sake of evaluating this method more thoroughly, the dataset will be used with k-folds 4 times, using Fibonacci sequence numbers 8 & 13 for *k* and 5 & 8 for the number of iterations of cross-validation. For comparison, a standard k-folds (not repeated) of 8 will also be used.

```
cv.train_control_8 <- trainControl(method="cv", number=8)
cv.train_control_8_5 <- trainControl(method="repeatedcv", number=8, repeats =
5)
cv.train_control_13_5 <- trainControl(method="repeatedcv", number=13, repeats
= 5)
cv.train_control_13_8 <- trainControl(method="repeatedcv", number=13, repeats
= 8)
```

These training controls will be used later on in this study to train the various models for the task of classification but for now they are merely abstract instructions on how to chunk the data.

Leave-one-out cross-validation

Leave-one-out cross-validation is a form of exhaustive cross-validation. Exhaustive cross-validation methods are said to *“learn and test on all possible ways to divide the original sample into a training and a validation set”* (Cross-validation (statistics), 2018). The leave-one-out method is a specific form of the leave-p-out technique, where instead of determine the test dataset as a fraction of the whole, as is the case with k-fold, *p* is the absolute count of instances to be used in the test subset. What makes this technique exhaustive is how the *p* subset is iterated such that every instance will be included in at least one test subset.

Leave-p-out can be computationally expensive because the larger *p* is, the greater coefficient is for testing and training with the subsets, bearing in mind that for a given size of *p*, as many possible permutations as can be created for the test subset of this size, from the original sample set, need to be created; this is why leave-one-out might be preferred since a set of one means that there need only be as many test and training sets as the sample size value.

```
# define training control
loocv.train_control <- trainControl(method="LOOCV")
```

As with the k-folds training controls, the leave-one-out control will be used later on but for now remains an abstract set of instructions on how to chunk the data.

Task 3: Build Train and Test a Decision Tree type Classifier

Premise

> You need to construct, train and test Decision Tree type classifier (C4.5, Random Forest) in R. Train and test your decision tree classifier using the training and test sets generated based on the methods tried as part of the 2nd Task.

Decision trees

Decision Trees are human-friendly models because they are able to expose their logic easily in a visual way (Hamel, 2015, Anh (2015)), instead of being considered ‘black boxes’; in addition to this, once trained they are quick to process new data as the rules of classification have already been definitively realised.

C4.5 Decision Tree

Finding the C4.5 method as an existing library within R, brought up more than one option, both appear to use an open-source equivalent of C4.5 rather than an official C4.5 implementation; other options included using C5.0 which apparently supersedes C4.5; seeing as the task was to investigate C4.5, J48 has been chosen as a more faithful example of the algorithm.

Below is not only an investigation into C4.5 but also, a comparison of two variations. Initially all available dimensions will be used for training.

The J48 method

```
load("holdout.training.rda")
load("holdout.testing.rda")
```

```
dt.c4_5_j48_h <- J48(as.factor(rings)~., holdout.training)
dt_sum.c4_5_j48_h <- summary(dt.c4_5_j48_h)
dt.c4_5_j48_h_party <- as.party(dt.c4_5_j48_h)
```

The J48 function is extremely quick with the dataset, training takes less than a second, which on its own is not necessarily worthy of note but certainly more interesting when compared to the speed of using the caret train method with a “J48” method argument value.

It is possible to look at the complexity of the tree by looking at the dimensionality, where the length is effectively the tree size, width is the number of leaves, or terminal nodes and the depth is effectively the number of conditional branch layers.

```
length(dt.c4_5_j48_h_party)
```

```
## [1] 1534
```

```
width(dt.c4_5_j48_h_party)
```

```
## [1] 784
```

```
depth(dt.c4_5_j48_h_party)
```

```
## [1] 22
```

The complexity of this tree is sufficiently complex to render a graphical representation useless. In fact the tree is so complex, it is time intensive as well as being of no value.

The caret train J48 argument

```
dt.c4_5_h2 <- train(as.factor(rings) ~., method="J48", holdout.training,
tuneLength = 8)
dt_sum.c4_5_h2 <- summary(dt.c4_5_h2)
dt.c4_5_h2_finalModel <- dt.c4_5_h2$finalModel
dt_sum.c4_5_h2_final <- summary(dt.c4_5_h2_finalModel)
```

Whilst in comparisons to other forms of training and data-mining algorithms, under 5 minutes for a 4,000 by 9 dataset might seem okay, in comparison to the J48 function, there seems to be something at odds. The proof will be in the comparison of the two models with regard to accuracy on the test dataset.

Again, the the complexity of the tree can be seen through the dimensionality, length being total size, width being leaf nodes and depth being branch layer count.

```
#dt.c4_5_h2
#dt.c4_5_h2_finalModel <- dt.c4_5_h2$finalModel
#dt.c4_5_h2_finalModel
dt.c4_5_h2_finalModel_party <- as.party(dt.c4_5_h2_finalModel)
#dt.c4_5_h2_finalModel_party <- as.party(dt.c4_5_h2$finalModel)
length(dt.c4_5_h2_finalModel_party)
```

```
## [1] 89
```

```
width(dt.c4_5_h2_finalModel_party)
```

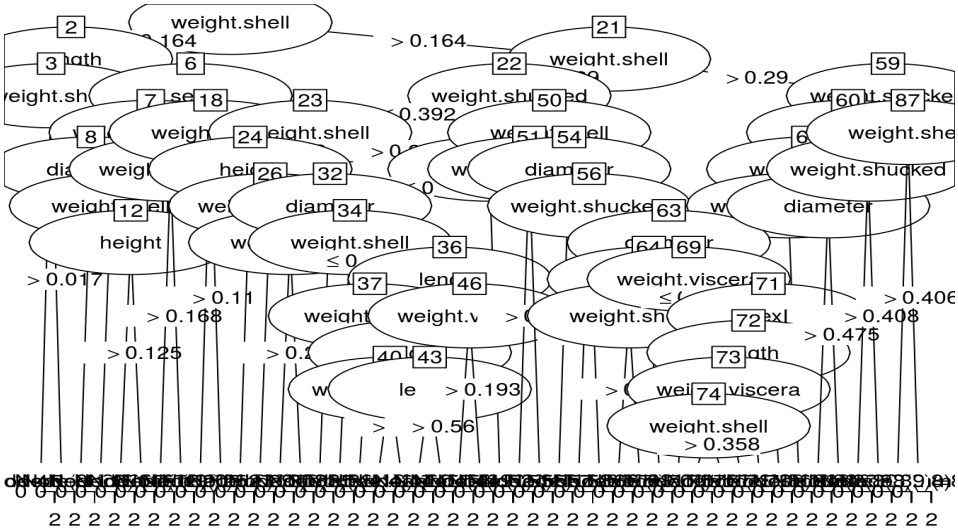
```
## [1] 45
```

```
depth(dt.c4_5_h2_finalModel_party)
```

```
## [1] 12
```

It would appear that the ‘final model’ derived from the train function has already been pruned, making for a simpler decision tree; this tree is actually able to be represented graphically within a reasonable amount of time (under 10 seconds).

```
#plot(dt.c4_5_h2_finalModel)
plot(dt.c4_5_h2_finalModel_party)
```



Even though the tree is able to be rendered it is not easy to get anything meaningful out of this. Perhaps the most pertinent point is that not only could the classification levels benefit from being simplified but using fewer dimensions for observation would also force a simpler set of conditional branching.

Comparing the two C4.5/J48 methods

Below follows the summaries from both methods, for examination:

```
dt_sum.c4_5_j48_h
```

```
##
## === Summary ===
##
## Correctly Classified Instances 2033 75.8299 %
## Incorrectly Classified Instances 648 24.1701 %
## Kappa statistic 0.7279
## Mean absolute error 0.0234
## Root mean squared error 0.1081
## Relative absolute error 35.265 %
## Root relative squared error 59.4069 %
## Total Number of Instances 2681
##
## === Confusion Matrix ===
##
## 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 aa <-- classified as
## 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | a = 2
## 0 5 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | b = 3
## 0 1 31 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | c = 4
## 0 1 4 57 7 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | d = 5
## 0 0 3 5 133 7 2 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | e = 6
## 0 0 1 3 12 203 12 5 5 2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | f = 7
## 0 0 2 1 8 22 294 25 12 6 0 1 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 | g = 8
## 0 0 0 3 4 6 21 363 13 21 2 2 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 | h = 9
## 0 0 0 0 2 4 16 31 342 11 5 5 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 | i = 10
## 0 0 0 0 3 7 14 13 18 260 5 2 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 | j = 11
## 0 0 0 0 0 1 9 11 15 9 114 3 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 | k = 12
## 0 0 0 0 1 1 4 8 11 12 3 85 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 | l = 13
## 0 0 0 0 0 4 5 6 11 8 6 4 42 1 2 2 0 0 0 0 0 0 0 0 0 0 0 | m = 14
## 0 0 0 0 0 0 4 4 7 4 5 4 2 29 0 0 0 0 0 1 0 0 0 0 0 0 0 | n = 15
## 0 0 0 0 0 0 1 6 3 3 1 1 3 1 26 0 1 0 0 0 0 0 0 0 0 0 0 | o = 16
## 0 0 0 0 0 0 2 4 1 0 2 1 3 1 0 23 0 0 0 0 0 0 0 0 0 0 0 | p = 17
```

```
## 0 0 0 0 0 1 2 1 5 4 1 1 2 2 0 0 10 0 0 0 0 0 0 0 0 0 0 | q = 18
## 0 0 0 0 1 0 1 1 2 3 1 3 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 | r = 19
## 0 0 0 0 0 1 0 0 1 1 2 1 0 0 0 1 0 0 7 0 0 0 0 0 0 0 0 | s = 20
## 0 0 0 0 0 0 0 0 0 3 1 0 1 1 1 0 0 0 0 6 0 0 0 0 0 0 0 | t = 21
## 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | u = 22
## 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 1 0 0 0 2 0 0 0 0 0 | v = 23
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 | w = 24
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 | x = 25
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 | y = 26
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 | z = 27
## 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | aa = 29


dt_sum.c4_5_h2


##
## === Summary ===
##
## Correctly Classified Instances 904 33.7188 %
## Incorrectly Classified Instances 1777 66.2812 %
## Kappa statistic 0.2414
## Mean absolute error 0.0578
## Root mean squared error 0.1699
## Relative absolute error 87.138 %
## Root relative squared error 93.3833 %
## Total Number of Instances 2681
##
## === Confusion Matrix ===
##
## 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 aa <-- classified as
## 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | a = 2
## 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | b = 3
## 0 0 20 11 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | c = 4
## 0 0 9 40 0 21 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | d = 5
## 0 0 0 34 0 85 20 10 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | e = 6
## 0 0 0 9 0 113 86 26 7 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 | f = 7
## 0 0 0 5 0 48 214 53 45 4 4 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 | g = 8
## 0 0 0 0 0 14 134 122 137 26 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 | h = 9
## 0 0 0 1 0 8 69 42 225 61 5 3 2 0 2 0 0 0 0 0 0 0 0 0 0 0 | i = 10
## 0 0 0 0 0 7 42 26 138 102 5 3 1 0 2 1 0 0 0 0 0 0 0 0 0 0 | j = 11
## 0 0 0 0 0 0 30 14 48 45 24 0 0 0 2 1 0 1 0 0 0 0 0 0 0 0 | k = 12
## 0 0 0 0 0 0 25 16 31 33 6 12 3 0 2 1 0 0 0 0 0 1 0 0 0 0 | l = 13
## 0 0 0 0 0 1 16 5 27 23 2 4 12 0 0 0 0 0 0 0 0 1 0 0 0 0 | m = 14
## 0 0 0 0 0 0 12 7 15 15 4 0 3 0 2 1 0 1 0 0 0 0 0 0 0 0 | n = 15
## 0 0 0 0 0 0 7 4 2 18 2 0 1 0 12 0 0 0 0 0 0 0 0 0 0 0 | o = 16
## 0 0 0 0 0 0 2 0 8 11 4 4 4 0 0 4 0 0 0 0 0 0 0 0 0 0 | p = 17
## 0 0 0 0 0 0 1 4 6 12 1 1 0 0 3 0 0 0 0 0 0 1 0 0 0 0 | q = 18
## 0 0 0 0 0 0 1 1 2 5 2 3 0 0 0 0 0 2 0 0 0 0 0 0 0 0 | r = 19
## 0 0 0 0 0 0 0 0 2 1 6 2 2 0 0 0 0 0 1 0 0 0 0 0 0 0 | s = 20
## 0 0 0 0 0 0 0 1 0 2 7 0 0 0 0 1 0 0 2 0 0 0 0 0 0 0 | t = 21
## 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 | u = 22
## 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 2 0 0 0 | v = 23
## 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | w = 24
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 | x = 25
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 | y = 26
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 | z = 27
## 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | aa = 29
```

Accuracy comparison of C4.5/J48 models

Despite the values being present in the summaries, to clarify understanding, they are repeated below:

```
dt_sum.c4_5_j48_h$details[1]
```

```
## pctCorrect
## 75.82991
```

```
dt_sum.c4_5_h2$details[1]
```

```
## pctCorrect
## 33.71876
```

The J48 function has a significantly higher accuracy compared to the train function J48 call but at this stage it is hard to be confident this is a good thing; given the difference in tree complexity, it could well be that the J48 function suffers from massive over-fitting, while the train call has done some excessive pruning which has not only accounted for the extra time for the function to complete but also the diminished accuracy. To understand things further it's really necessary to test the trees against the validation subset.

Comparing the two C4.5/J48 models after prediction

What follows is the output of testing the models against the test subset.

```
holdout.test_rings <- holdout.testing$rings
dt.c4_5_j48_h_test <- predict(dt.c4_5_j48_h, newdata = holdout.testing)
holdout.test_levels <- min(holdout.test_rings):max(holdout.test_rings)
dt.c4_5_j48_h_test_cm <- confusionMatrix(factor(dt.c4_5_j48_h_test,
levels=holdout.test_levels), factor(holdout.test_rings, levels =
holdout.test_levels))
kable(dt.c4_5_j48_h_test_cm$table)
```

	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
3	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	4	5	4	3	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	5	5	9	8	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	5	13	27	18	12	6	8	3	2	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	4	31	45	30	17	8	2	2	2	0	1	0	0	0	0	0	0	0	0	0	0
8	0	0	1	9	20	39	47	24	17	6	5	4	2	1	3	0	0	0	0	0	0	0	0
9	0	0	0	3	12	39	46	40	26	21	4	6	5	0	1	0	1	0	0	0	0	0	0
10	0	0	0	2	8	21	50	53	35	21	13	12	8	5	2	5	4	3	0	0	0	0	0
11	0	0	0	3	5	15	41	28	37	20	14	3	6	2	6	2	3	1	0	0	0	0	0
12	0	0	0	0	2	5	9	19	16	7	8	4	2	2	2	1	0	1	1	2	1	0	0
13	0	0	0	0	2	3	5	9	8	6	8	1	3	7	1	2	4	1	0	0	0	0	0
14	0	0	0	0	0	0	0	4	1	2	3	2	4	1	0	0	0	3	0	0	1	0	0
15	0	0	0	0	0	1	2	5	4	2	4	0	3	0	1	1	1	1	0	0	0	0	0
16	0	0	0	0	0	0	1	0	0	1	4	3	2	1	2	0	1	0	0	0	0	0	0
17	0	0	0	1	0	1	1	0	0	2	2	0	0	0	1	0	0	0	0	1	1	0	0
18	0	0	0	0	0	0	0	1	2	0	1	0	2	0	1	0	1	1	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	1	0	0	0	0	0

	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
21	0	0	0	0	0	0	0	0	0	3	1	0	0	1	0	1	0	0	0	1	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
dt.c4_5_j48_h_test_cm$overall[1]
```

```
## Accuracy
## 0.2092676
```

```
#dt.c4_5_h2
dt.c4_5_h2_test <- predict(dt.c4_5_h2, newdata = holdout.testing)
dt.c4_5_h2_test_cm <- confusionMatrix(factor(dt.c4_5_h2_test,
levels=holdout.test_levels), factor(holdout.test_rings, levels =
holdout.test_levels))
kable(dt.c4_5_h2_test_cm$table)
```

	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	5	8	6	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	8	15	10	11	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	8	61	53	21	5	5	1	3	1	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	10	45	78	91	44	32	7	10	8	8	3	2	1	0	0	1	0	0	0	0
9	0	0	0	3	12	33	35	30	15	16	5	3	4	3	0	1	1	1	0	0	0	0	0
10	0	0	0	0	3	25	72	82	63	33	21	10	11	3	4	6	5	2	0	1	0	0	0
11	0	0	0	1	0	6	16	31	36	26	21	5	10	7	12	3	6	5	0	3	1	0	0
12	0	0	0	0	0	2	4	6	2	6	3	2	0	1	1	0	0	1	0	0	0	0	0
13	0	0	0	1	0	0	0	2	0	1	0	1	5	0	0	0	0	1	0	0	0	0	0
14	0	0	0	0	0	0	0	0	1	1	1	3	2	3	1	1	0	0	0	0	2	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	1	1	1	0	5	1	0	0	1	1	2	0	0	0	0	0	0
17	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	2	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
23	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0	0	1	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
dt.c4_5_h2_test_cm$overall[1]
```

```
## Accuracy
## 0.2361734
```

It would appear that despite the J48 function derived model claiming a higher accuracy, the resulting confusion matrix data suggests otherwise. Neither 23.6% nor 20.9% is particularly encouraging, however for the improvement in speed, the loss of about 12% in accuracy (approximately 3/24), might be reasonable if the overall accuracy can be improved in other ways (like further tuning). To investigate this further, the next steps are to look at reducing the number of attributes observed based on the earlier analysis of data, and training the models to achieve simpler classification goals.

C4.5 with refined dataset

To attempt attainment of high accuracy, it's worth looking to run the very same decision tree functions against a streamline classification aim, with a more targeted formula. Using raw rings and the classification levels amounted to nearly 30 possible outcomes, so bringing that down to a tiered factor of 3 age groups has to make things not only more performant but it's easier to be more accurate when the intended classification type has a larger scope.

Refined formula

Previously, each of the attributes of a continuous numeric type were analysed for a correlation to the number of rings. To have a simpler decision tree it stands to reason that picking only the most relevant attributes for the problem of this particular classification are used. To this end, the attributes selected for the revised formula will be those top three correlates: *weight.shell, height, diameter*.

```
dt.formula <- as.formula(age_group ~ weight.shell + height + diameter)
```

The J48 method with updated formula and simpler classification

```
dt.c4_5_j48_h_a <- J48(dt.formula, holdout_age_groups.training)
dt_sum.c4_5_j48_h_a <- summary(dt.c4_5_j48_h_a)
dt.c4_5_j48_h_a_party <- as.party(dt.c4_5_j48_h_a)
#
dt_sum.c4_5_j48_h_a
```

```
##
## === Summary ===
##
## Correctly Classified Instances 1892 70.5707 %
## Incorrectly Classified Instances 789 29.4293 %
## Kappa statistic 0.5053
## Mean absolute error 0.2797
## Root mean squared error 0.374
```

```
## Relative absolute error 68.3388 %
## Root relative squared error 82.6712 %
## Total Number of Instances 2681
##
## === Confusion Matrix ===
##
## a b c <-- classified as
## 1084 223 14 | a = young
## 243 630 37 | b = middle
## 79 193 178 | c = old
```

```
#
length(dt.c4_5_j48_h_a_party)
```

```
## [1] 103
```

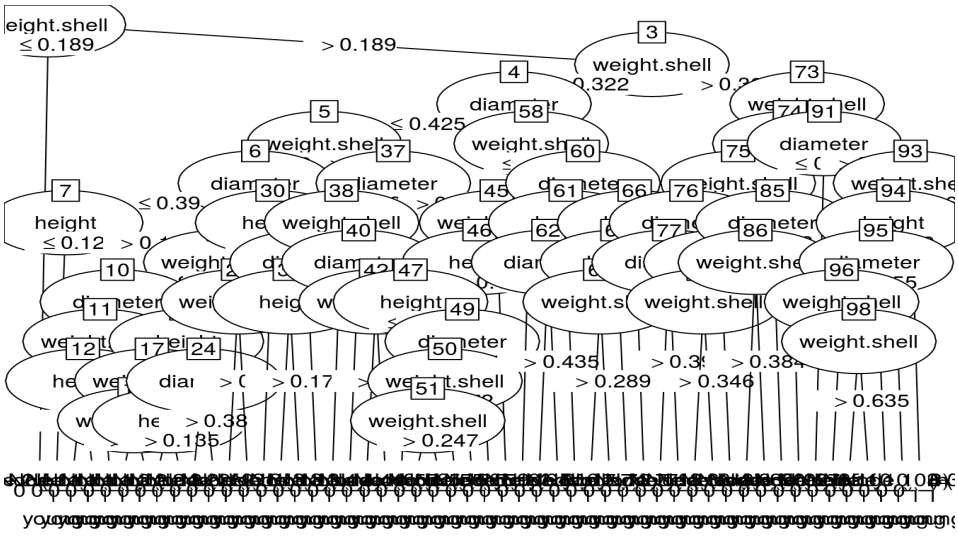
```
width(dt.c4_5_j48_h_a_party)
```

```
## [1] 52
```

```
depth(dt.c4_5_j48_h_a_party)
```

```
## [1] 11
```

```
#
plot(dt.c4_5_j48_h_a_party)
```



The caret train J48 argument with updated formula and simpler classification

```
dt.c4_5_h2_a <- train(dt.formula, method="J48", holdout_age_groups.training,
tuneLength = 8)
```

```
dt_sum.c4_5_h2_a <- summary(dt.c4_5_h2_a)
dt_sum.c4_5_h2_a_final <- summary(dt.c4_5_h2_a$finalModel)
#
dt_sum.c4_5_h2_a
```

```
##
## === Summary ===
##
## Correctly Classified Instances 1821 67.9224 %
## Incorrectly Classified Instances 860 32.0776 %
## Kappa statistic 0.4683
## Mean absolute error 0.2995
## Root mean squared error 0.387
## Relative absolute error 73.1807 %
## Root relative squared error 85.5498 %
## Total Number of Instances 2681
##
## === Confusion Matrix ===
##
## a b c <-- classified as
## 1048 229 44 | a = young
## 234 590 86 | b = middle
## 69 198 183 | c = old
```

```
#
dt.c4_5_h2_a_finalModel <- dt.c4_5_h2_a$finalModel
dt.c4_5_h2_a_finalModel_party <- as.party(dt.c4_5_h2_a_finalModel)
#
length(dt.c4_5_h2_a_finalModel_party)
```

```
## [1] 33
```

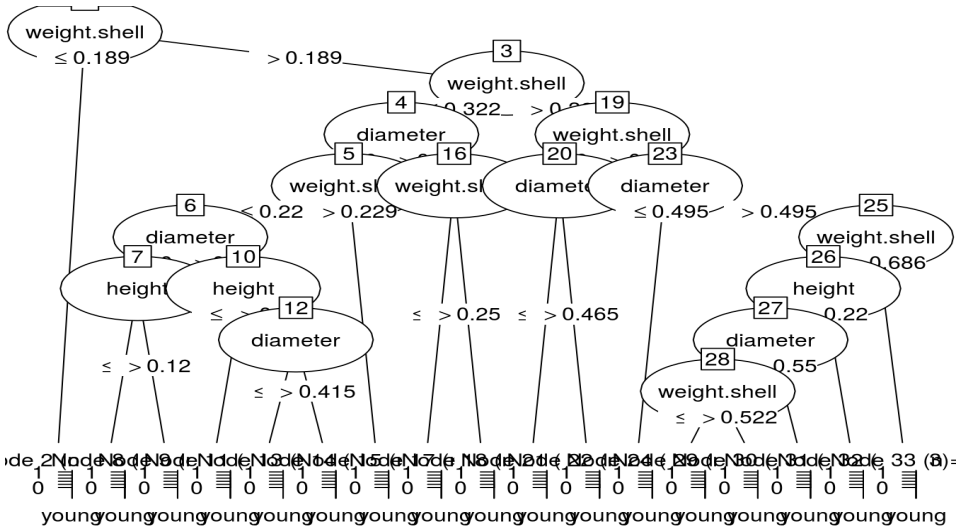
```
width(dt.c4_5_h2_a_finalModel_party)
```

```
## [1] 17
```

```
depth(dt.c4_5_h2_a_finalModel_party)
```

```
## [1] 8
```

```
#
plot(dt.c4_5_h2_a_finalModel_party)
```



Accuracy comparison of revised C4.5/J48 models

```
dt_sum.c4_5_j48_h_a$details[1]
```

```
## pctCorrect
## 70.57068
```

```
dt_sum.c4_5_h2_a$details[1]
```

```
## pctCorrect
## 67.92242
```

Interestingly the J48 function has produced an accuracy rating that is only marginally better than the previous version while the train J48 function call seems to have improved significantly so that it is nearly on a par with the J48 function result. Looking at the models after validation has happened will hopefully provide even more revealing findings.

Comparing the revised C4.5/J48 model after prediction

What follows is the output of testing the models against the test subset.

```
#holdout.test_rings <- holdout.testing$rings
dt.c4_5_j48_h_a_test <- predict(dt.c4_5_j48_h_a, newdata =
holdout_age_groups.testing)
#holdout.test_levels <- min(holdout.test_rings):max(holdout.test_rings)
#dt.c4_5_j48_h_a_test_cm <- confusionMatrix(factor(dt.c4_5_j48_h_test,
levels=holdout.test_levels), factor(holdout.test_rings, levels =
holdout.test_levels))
dt.c4_5_j48_h_a_test_cm <- confusionMatrix(dt.c4_5_j48_h_a_test,
holdout_age_groups.testing$age_group)
kable(dt.c4_5_j48_h_a_test_cm$table)
```

	young	middle	old
young	517	147	34
middle	119	267	138
old	21	33	62

```
dt.c4_5_j48_h_a_test_cm$overall[1]
```

```
## Accuracy
## 0.632287
```

```
#dt.c4_5_h2
dt.c4_5_h2_a_test <- predict(dt.c4_5_h2_a, newdata =
holdout_age_groups.testing)
dt.c4_5_h2_a_test_cm <- confusionMatrix(dt.c4_5_h2_a_test,
holdout_age_groups.testing$age_group)
kable(dt.c4_5_h2_a_test_cm$table)
```

	young	middle	old
young	510	134	27
middle	120	263	123
old	27	50	84

```
dt.c4_5_h2_a_test_cm$overall[1]
```

```
## Accuracy
## 0.6405082
```

The accuracy has improved markedly so it's safe to say that the combination of streamlining the formula and creating a simpler classification requirement has improved things; this is the new baseline, now it's worth looking at any improvement that can be made through using the k-folds and leave-one-out cross-validation techniques.

C4.5 with cross-validation techniques

Given how the accuracy between the two forms of C4.5 model generation narrowed to an absolute percentage delta of less than 1 percent, coupled with the speed at which the J48 function returns, the next phase of experimentation will occur only with this function and the relevant training control options.

J48 with k-folds training

The J48 function does not accept the caret training control objects as valid control parameters; calling the Weka function `WOW` with `"J48"` as the sole argument presents the list of arguments that can be passed in to a `Weka_control` function call to configure training.

```
WOW("J48")
```

```
## -U Use unpruned tree.
## -O Do not collapse tree.
## -C <pruning confidence>
## Set confidence threshold for pruning. (default 0.25)
## Number of arguments: 1.
## -M <minimum number of instances>
## Set minimum number of instances per leaf. (default 2)
## Number of arguments: 1.
## -R Use reduced error pruning.
## -N <number of folds>
## Set number of folds for reduced error pruning. One fold is
## used as pruning set. (default 3)
```

```
## Number of arguments: 1.
## -B Use binary splits only.
## -S Do not perform subtree raising.
## -L Do not clean up after the tree has been built.
## -A Laplace smoothing for predicted probabilities.
## -J Do not use MDL correction for info gain on numeric
## attributes.
## -Q <seed>
## Seed for random data shuffling (default 1).
## Number of arguments: 1.
## -doNotMakeSplitPointActualValue
## Do not make split point actual value.
## -output-debug-info
## If set, classifier is run in debug mode and may output
## additional info to the console
## -do-not-check-capabilities
## If set, classifier capabilities are not checked before
## classifier is built (use with caution).
## -num-decimal-places
## The number of decimal places for the output of numbers in
## the model (default 2).
## Number of arguments: 1.
## -batch-size
## The desired batch size for batch prediction (default 100).
## Number of arguments: 1.
```

Unfortunately, for the performant J48 method, none of these options seems to allow for custom methods of training. At this point, J48 has to be disregarded due to inflexibility despite such good run-time training speeds.

Train with J48 method and k-folds training

Three types of k-folds cross-validation configurations were created; at this point it's opportune to examine which, if any, are able to improve the accuracy of training and validation. Each of the three types of k-folds configurations will be used to train the decision tree.

The important implementation detail to emphasise here is that while the holdout method required a one-off explicit call to partition the data, with the user assigning which part as training or testing subset, with cross-validation the entire dataset is passed into the training call; this is because the training will occur several times using many subsampled testing subsets from the original sample. For the sake of completeness, the holdout testing subset can still be used to further interrogate the model, in order to better compare against other classifiers.

K-folds, with 8 folds

```
dt.c4_5_j48_kf_a <- train(dt.formula, method="J48",
abalone_data_cleansed_age_groups, tuneLength = 8, trControl =
cv.train_control_8)
```

K-folds, with 8 folds, 5 repetitions

```
dt.c4_5_j48_r5kf8_a <- train(dt.formula, method="J48",
abalone_data_cleansed_age_groups, tuneLength = 8, trControl =
cv.train_control_8_5)
```

K-folds, with 13 folds, 5 repetitions

```
dt.c4_5_j48_r5kf13_a <- train(dt.formula, method="J48",
abalone_data_cleansed_age_groups, tuneLength = 8, trControl =
```

```
cv.train_control_13_5)
```

K-folds, with 13 folds, 8 repetitions

```
dt.c4_5_j48_r8kf13_a <- train(dt.formula, method="J48",
abalone_data_cleansed_age_groups, tuneLength = 8, trControl =
cv.train_control_13_8)
```

Training results J48 method and k-folds training

Below is a comparison of the different training results for k-folds; ultimately, the best one will be picked as the preferred use of k-folds going forward; obviously should this preferred configuration turn out to be too time intensive with other types of classifier, falling back to another configuration will be considered.

K-folds, with 8 folds

```
dt_sum.c4_5_j48_kf_a <- summary(dt.c4_5_j48_kf_a)
dt.c4_5_j48_kf_a_final <- summary(dt.c4_5_j48_kf_a$finalModel)
#
dt_sum.c4_5_j48_kf_a
```

```
##
## === Summary ===
##
## Correctly Classified Instances 2771 68.9475 %
## Incorrectly Classified Instances 1248 31.0525 %
## Kappa statistic 0.4824
## Mean absolute error 0.2887
## Root mean squared error 0.3799
## Relative absolute error 70.4376 %
## Root relative squared error 83.9297 %
## Total Number of Instances 4019
##
## === Confusion Matrix ===
##
## a b c <-- classified as
## 1647 268 63 | a = young
## 402 831 124 | b = middle
## 95 296 293 | c = old
```

```
#
dt.c4_5_j48_kf_a_finalModel <- dt.c4_5_h2_a$finalModel
dt.c4_5_j48_kf_a_finalModel_party <- as.party(dt.c4_5_h2_a$finalModel)
#
length(dt.c4_5_j48_kf_a_finalModel_party)
```

```
## [1] 33
```

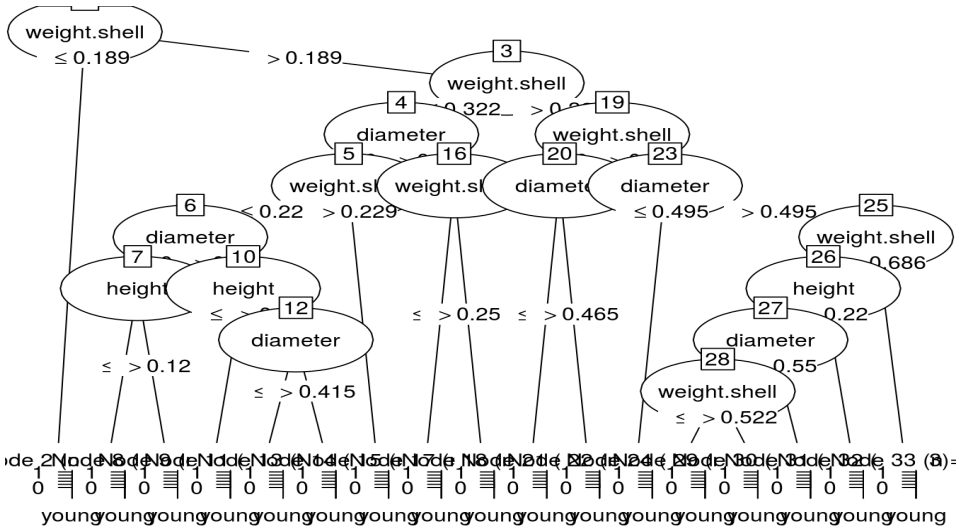
```
width(dt.c4_5_j48_kf_a_finalModel_party)
```

```
## [1] 17
```

```
depth(dt.c4_5_j48_kf_a_finalModel_party)
```

```
## [1] 8
```

```
#
plot(dt.c4_5_j48_kf_a_finalModel_party)
```



K-folds, with 8 folds, 5 repetitions

```
dt_sum.c4_5_r5kf8_a <- summary(dt.c4_5_j48_r5kf8_a)
dt.c4_5_j48_r5kf8_a_final <- summary(dt.c4_5_j48_r5kf8_a$finalModel)
#
dt_sum.c4_5_r5kf8_a
```

```
##
## === Summary ===
##
## Correctly Classified Instances 2694 67.0316 %
## Incorrectly Classified Instances 1325 32.9684 %
## Kappa statistic 0.4385
## Mean absolute error 0.3154
## Root mean squared error 0.3971
## Relative absolute error 76.9366 %
## Root relative squared error 87.7163 %
## Total Number of Instances 4019
##
## === Confusion Matrix ===
##
## a b c <-- classified as
## 1696 239 43 | a = young
## 516 746 95 | b = middle
## 168 264 252 | c = old
```

```
#
dt.c4_5_j48_r5kf8_a_finalModel <- dt.c4_5_j48_r5kf8_a$finalModel
dt.c4_5_j48_r5kf8_a_finalModel_party <-
as.party(dt.c4_5_j48_r5kf8_a_finalModel)
#
length(dt.c4_5_j48_r5kf8_a_finalModel_party)
```

```
## [1] 39
```

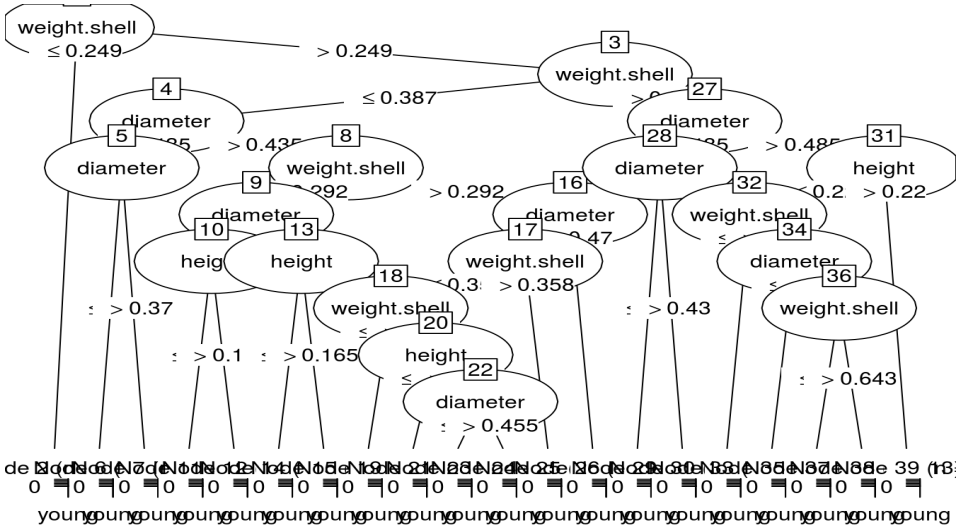
```
width(dt.c4_5_j48_r5kf8_a_finalModel_party)
```

```
## [1] 20
```

```
depth(dt.c4_5_j48_r5kf8_a_finalModel_party)
```

```
## [1] 9
```

```
#
plot(dt.c4_5_j48_r5kf8_a_finalModel_party)
```



K-folds, with 13 folds, 5 repetitions

```
dt_sum.c4_5_r5kf13_a <- summary(dt.c4_5_j48_r5kf13_a)
dt.c4_5_j48_r5kf13_a_final <- summary(dt.c4_5_j48_r5kf13_a$finalModel)
#
dt_sum.c4_5_r5kf13_a
```

```
##
## === Summary ===
##
## Correctly Classified Instances 2731 67.9522 %
## Incorrectly Classified Instances 1288 32.0478 %
## Kappa statistic 0.4628
## Mean absolute error 0.2957
## Root mean squared error 0.3845
## Relative absolute error 72.1388 %
## Root relative squared error 84.9372 %
## Total Number of Instances 4019
##
## === Confusion Matrix ===
##
## a b c <-- classified as
## 1655 264 59 | a = young
```

```
## 436 801 120 | b = middle
## 113 296 275 | c = old
```

```
#
dt.c4_5_j48_r5kf13_a_finalModel <- dt.c4_5_j48_r5kf13_a$finalModel
dt.c4_5_j48_r5kf13_a_finalModel_party <-
as.party(dt.c4_5_j48_r5kf13_a_finalModel)
#
length(dt.c4_5_j48_r5kf13_a_finalModel_party)
```

```
## [1] 59
```

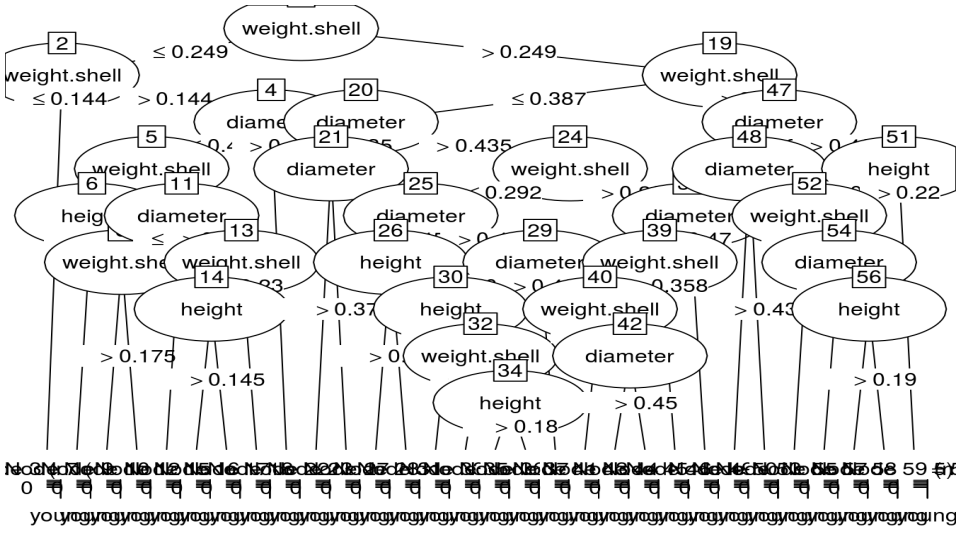
```
width(dt.c4_5_j48_r5kf13_a_finalModel_party)
```

```
## [1] 30
```

```
depth(dt.c4_5_j48_r5kf13_a_finalModel_party)
```

```
## [1] 9
```

```
#
plot(dt.c4_5_j48_r5kf13_a_finalModel_party)
```



K-folds, with 13 folds, 8 repetitions

```
dt_sum.c4_5_r8kf13_a <- summary(dt.c4_5_j48_r8kf13_a)
dt.c4_5_j48_r8kf13_a_final <- summary(dt.c4_5_j48_r8kf13_a$finalModel)
#
dt_sum.c4_5_r8kf13_a
```

```
##
## === Summary ===
##
## Correctly Classified Instances 2731 67.9522 %
## Incorrectly Classified Instances 1288 32.0478 %
```

```
## Kappa statistic 0.4628
## Mean absolute error 0.2957
## Root mean squared error 0.3845
## Relative absolute error 72.1388 %
## Root relative squared error 84.9372 %
## Total Number of Instances 4019
##
## === Confusion Matrix ===
##
## a b c <-- classified as
## 1655 264 59 | a = young
## 436 801 120 | b = middle
## 113 296 275 | c = old
```

```
#
dt.c4_5_j48_r8kf13_a_finalModel <- dt.c4_5_j48_r8kf13_a$finalModel
dt.c4_5_j48_r8kf13_a_finalModel_party <-
as.party(dt.c4_5_j48_r8kf13_a_finalModel)
#
length(dt.c4_5_j48_r8kf13_a_finalModel_party)
```

```
## [1] 59
```

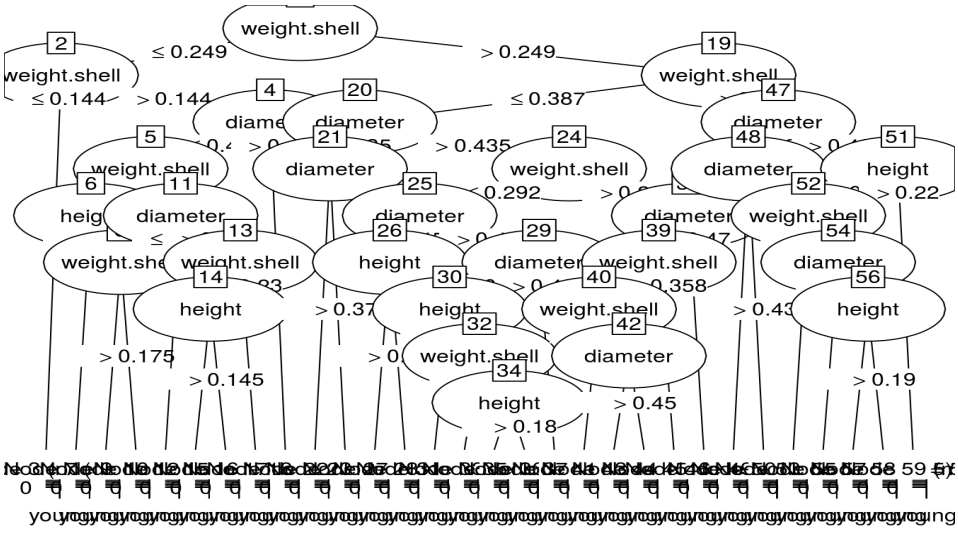
```
width(dt.c4_5_j48_r8kf13_a_finalModel_party)
```

```
## [1] 30
```

```
depth(dt.c4_5_j48_r8kf13_a_finalModel_party)
```

```
## [1] 9
```

```
#
plot(dt.c4_5_j48_r8kf13_a_finalModel_party)
```




```
dt_sum.c4_5_j48_kf_a$details[1]
```

```
## pctCorrect
## 68.9475
```

```
dt_sum.c4_5_r5kf8_a$details[1]
```

```
## pctCorrect
## 67.0316
```

```
dt_sum.c4_5_r5kf13_a$details[1]
```

```
## pctCorrect
## 67.95223
```

After having run the training, the preliminary results suggest that while repeating the whole k-folds process a number of times can add some more accuracy, increasing the *k* count doesn't add much more; depending on the size of the real-world dataset and the economic benefit of even the slightest improvement in accuracy, it's arguable that a trade-off between accuracy and performance can be made and sometimes a quicker less accurate option is the right tool for the job. That being said, prediction is significantly faster than training for decision trees; once the model has been built, new data is just processed into a class.

The fact that for the k-folds with 5 repetitions, the *k* = 13 version took about twice as long to compute as the *k* = 8 version, the 8 version will be considered the preferred choice, assuming the validation doesn't suggest otherwise.

Testing results J48 method and k-folds training

K-folds, with 8 folds

```
dt.c4_5_j48_kf_a_test <- predict(dt.c4_5_j48_kf_a, newdata =
holdout_age_groups.testing)
dt.c4_5_j48_kf_a_test_cm <- confusionMatrix(dt.c4_5_j48_kf_a_test,
holdout_age_groups.testing$age_group)
kable(dt.c4_5_j48_kf_a_test_cm$table)
```

	young	middle	old
young	543	146	28
middle	90	257	116
old	24	44	90

```
dt.c4_5_j48_kf_a_test_cm$overall[1]
```

```
## Accuracy
## 0.6651719
```

K-folds, with 8 folds, 5 repetitions

```
dt.c4_5_j48_r5kf8_a_test <- predict(dt.c4_5_j48_r5kf8_a, newdata =
holdout_age_groups.testing)
dt.c4_5_j48_r5kf8_a_test_prob <- predict(dt.c4_5_j48_r5kf8_a, newdata =
```

```
holdout_age_groups.testing, type = "prob")
dt.c4_5_j48_r5kf8_a_test_cm <- confusionMatrix(dt.c4_5_j48_r5kf8_a_test,
holdout_age_groups.testing$age_group)
kable(dt.c4_5_j48_r5kf8_a_test_cm$table)
```

	young	middle	old
young	565	186	47
middle	78	227	108
old	14	34	79

```
dt.c4_5_j48_r5kf8_a_test_cm$overall[1]
```

```
## Accuracy
## 0.6509716
```

K-folds, with 13 folds, 5 repetitions

```
dt.c4_5_j48_r5kf13_a_test <- predict(dt.c4_5_j48_r5kf13_a, newdata =
holdout_age_groups.testing)
dt.c4_5_j48_r5kf13_a_test_cm <- confusionMatrix(dt.c4_5_j48_r5kf13_a_test,
holdout_age_groups.testing$age_group)
kable(dt.c4_5_j48_r5kf13_a_test_cm$table)
```

	young	middle	old
young	547	159	32
middle	88	244	114
old	22	44	88

```
dt.c4_5_j48_r5kf13_a_test_cm$overall[1]
```

```
## Accuracy
## 0.6569507
```

K-folds, with 13 folds, 8 repetitions

```
dt.c4_5_j48_r8kf13_a_test <- predict(dt.c4_5_j48_r8kf13_a, newdata =
holdout_age_groups.testing)
dt.c4_5_j48_r8kf13_a_test_prob <- predict(dt.c4_5_j48_r8kf13_a, newdata =
holdout_age_groups.testing, type = "prob")
dt.c4_5_j48_r8kf13_a_test_cm <- confusionMatrix(dt.c4_5_j48_r8kf13_a_test,
holdout_age_groups.testing$age_group)
kable(dt.c4_5_j48_r8kf13_a_test_cm$table)
```

	young	middle	old
young	547	159	32
middle	88	244	114
old	22	44	88

```
dt.c4_5_j48_r8kf13_a_test_cm$overall[1]
```

```
## Accuracy
## 0.6569507
```

J48 with leave-one-out training

The last C4.5 decision tree to explore is a model using the leave-one-out training method, however research (Steinberg, 2013) and experimentation suggests this is too computationally expensive for decision trees; when attempting to train a tree with this training control, over an hour passed without any output; it would appear that for trees, not only *p* but *n* (the total number of observations) should also be very low, certainly smaller than the Abalone dataset.

C4.5 Conclusion

The results were lower than expected, with barely any improvement between the holdout method and the k-folds cross-validation technique; while further experimentation into increasing the number of steps might return higher accuracy rates, the cost of training goes up significantly. It will have to be seen later how well the results compare to all the rest.

With regard to the Abalone dataset in particular, it may not be the most suitable model for classification, even if cross-validation if able to improve training slightly.

Random Forest Decision Tree

The Random Forest Decision Tree is an algorithm that creates multiple internalised decision trees (Polamuri, 2017), which are all used when trying to classify new data. Each tree will vote, with a weight as to what the classification will be, with the final classification going to the greatest vote value. Random Forests are fast, less prone to over-fitting and work well with incomplete data. The major disadvantage of Random Forests is that they can't be used for regression, failing to extrapolate a result for data that is out-of-bounds of the original training dataset; given the scope of this particular Abalone age group classification, this shouldn't be too much of an issue.

With the previous Decision Tree (C4.5) training, much experimenting was done as to how the dataset should be prepared for the purpose of classification and the optimal training controls for Decision Trees; it seems reasonable enough to streamline the analysis of the Random Forest by carrying over into this sub-section the two main lessons learned:

- The classification problem should just aim to categorise the Abalone instances into 3 age groups
- The k-folds technique benefits from repetition to more thoroughly train with the data but the number of folds needn't be too high.

For the Random Forest training, only one configuration of repeated k-folds will be used, namely '8 folds, 5 times' as it provided adequate improvement whereupon '13 folds, 5 times' barely improved.

Random Forest Holdout

The same holdout values derived in *Task 2* that were used for C4.5, will be applied for testing and training the Random Forest.

Random Forest Holdout training

```
dt.rf_h_a <- train(dt.formula, data = holdout_age_groups.training, method =
"rf", prox= TRUE)
```

```
## note: only 2 unique complexity parameters in default grid. Truncating the
grid to 2 .
```

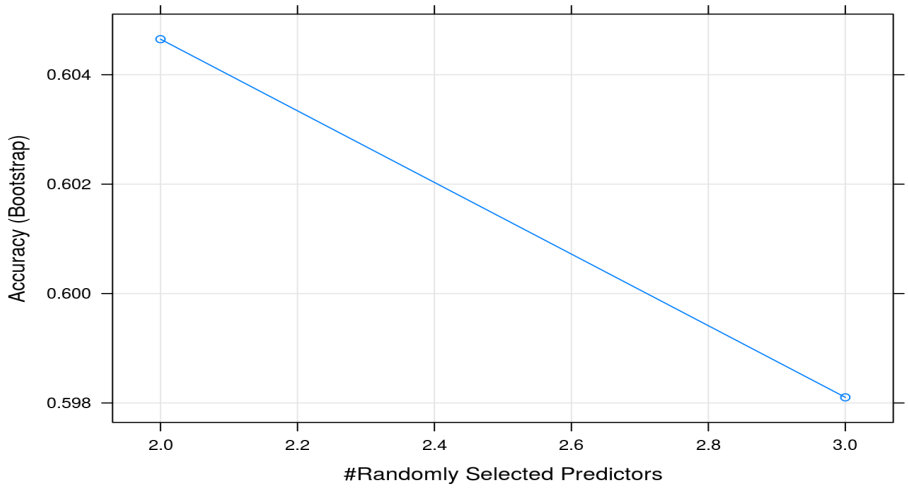
```
dt.rf_h_a_finalModel <- dt.rf_h_a$finalModel
dt_sum.rf_h_a <- summary(dt.rf_h_a)
#
dt.rf_h_a_finalModel$forest[11]
```

```
## $nrnodes
## [1] 1561
```

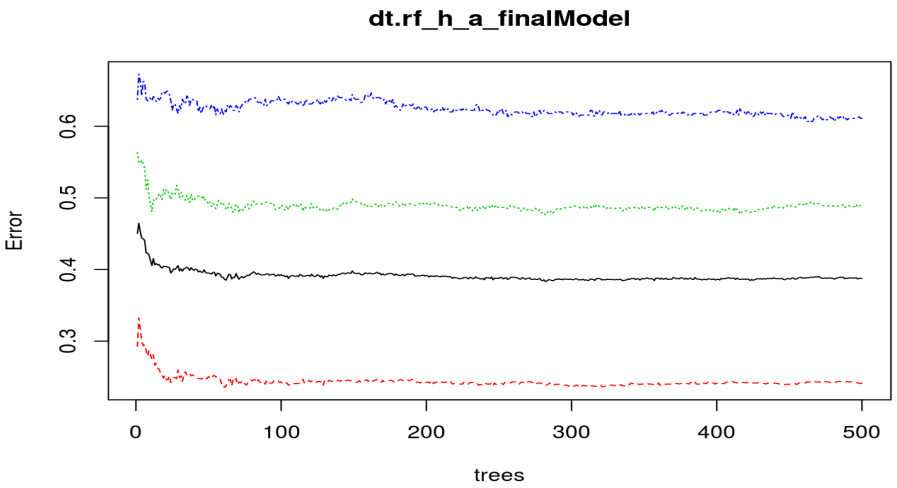
```
dt.rf_h_a_finalModel$forest[12]
```

```
## $ntree
## [1] 500
```

```
#
plot(dt.rf_h_a)
```




```
plot(dt.rf_h_a_finalModel)
```



After training, it's possible to observe there are 500 trees in the forest. Another observation is that even though no control was provided, the default caret control for random forest uses the Bootstrap method, this in effect reuses and recycles the holdout training data to train the forest against more data. As such, another

library will be used to attempt to examine the results with a more strict holdout training of a Random Forest.

Random Forest Holdout testing

```
dt.rf_h_a_test <- predict(dt.rf_h_a, newdata = holdout_age_groups.testing)

dt.rf_h_a_test_cm <- confusionMatrix(dt.rf_h_a_test,
holdout_age_groups.testing$age_group)
kable(dt.rf_h_a_test_cm$table)
```

	young	middle	old
young	500	161	36
middle	116	207	114
old	41	79	84

```
dt.rf_h_a_test_cm$overall[1]
```

```
## Accuracy
## 0.5911809
```

Another Random Forest Holdout

The `ranger` package has a method designed specifically for holdout Random Forests, which is used next to explore if there could be any better performance out of the traditional holdout technique.

```
dt.hrf_h_a <- holdoutRF(dt.formula, holdout_age_groups.training)
dt.hrf_h_a_prob <- holdoutRF(dt.formula, holdout_age_groups.training,
probability = TRUE)
```

For some reason, not quite clear from the documentation, this method produces 2 Random Forests.

```
#dt.hrf_h_a_finalModel <- dt.hrf_h_a$finalModel
dt_sum.hrf_h_a <- summary(dt.hrf_h_a)
#
dt.hrf_h_a$rf1
```

```
## Ranger result
##
## Call:
## ranger(..., importance = "permutation", case.weights = weights, replace =
FALSE, holdout = TRUE)
##
## Type: Classification
## Number of trees: 500
## Sample size: 2681
## Number of independent variables: 3
## Mtry: 1
## Target node size: 1
## Variable importance mode: permutation
## OOB prediction error: 37.69 %
```

```
dt.hrf_h_a$rf2
```

```
## Ranger result
##
## Call:
## ranger(..., importance = "permutation", case.weights = 1 - weights, replace
= FALSE, holdout = TRUE)
##
## Type: Classification
## Number of trees: 500
## Sample size: 2681
## Number of independent variables: 3
## Mtry: 1
## Target node size: 1
## Variable importance mode: permutation
## OOB prediction error: 34.17 %
```

```
dt.hrf1_h_a_test <- predict(dt.hrf_h_a$rf1, holdout_age_groups.testing)
dt.hrf1_h_a_test_prob <- predict(dt.hrf_h_a_prob$rf1,
holdout_age_groups.testing)

dt.hrf1_h_a_test_cm <- confusionMatrix(dt.hrf1_h_a_test$predictions,
holdout_age_groups.testing$age_group)
kable(dt.hrf1_h_a_test_cm$table)
```

	young	middle	old
young	520	163	41
middle	113	225	124
old	24	59	69

```
dt.hrf1_h_a_test_cm$overall[1]
```

```
## Accuracy
## 0.6083707
```

```
#
dt.hrf2_h_a_test <- predict(dt.hrf_h_a$rf2, holdout_age_groups.testing)
dt.hrf2_h_a_test_prob <- predict(dt.hrf_h_a_prob$rf2,
holdout_age_groups.testing)

dt.hrf2_h_a_test_cm <- confusionMatrix(dt.hrf2_h_a_test$predictions,
holdout_age_groups.testing$age_group)
kable(dt.hrf2_h_a_test_cm$table)
```

	young	middle	old
young	519	156	33
middle	122	233	132
old	16	58	69

```
dt.hrf2_h_a_test_cm$overall[1]
```

```
## Accuracy
## 0.6136024
```

These results are comparable to the caret bootstrap results, to the extent that while the caret package Random Forest call has a lower accuracy value, the fact it correctly classifies more old Abalone instances needs to be considered in its favour when compared to the first *(rf1)* Random Forest returns a slightly better accuracy rate.

Random Forest Cross-validation

Even though opinion in the Data Science community is that a Random Forest uses techniques that effectively provide internal solutions for cross-validation while training, setting an explcit control will ensure a complete rotation of data for testing and training.

Random Forest Cross-validation training

```
dt.rf_cv_r5kf8_a <- train(dt.formula, data = abalone_data_cleansed_age_groups,
method = "rf", prox= TRUE, trControl = cv.train_control_8_5)
```

```
## note: only 2 unique complexity parameters in default grid. Truncating the
grid to 2 .
```

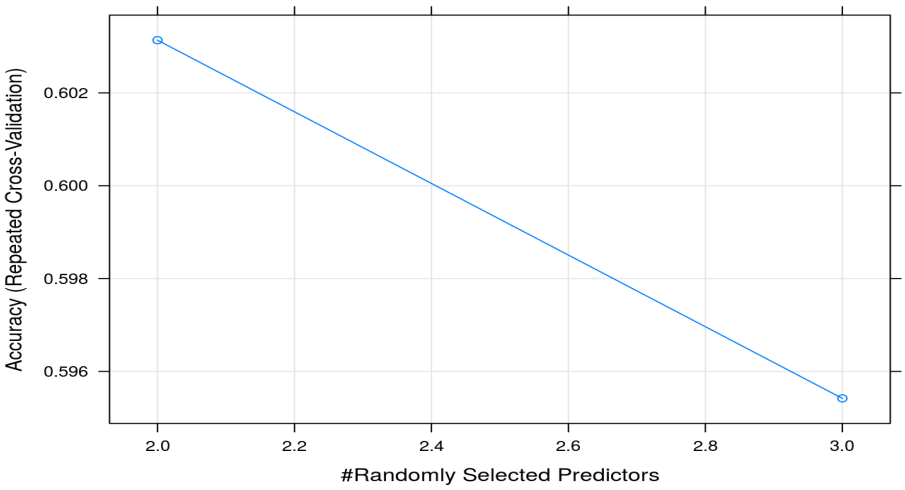
```
dt.rf_cv_r5kf8_a_finalModel <- dt.rf_cv_r5kf8_a$finalModel
dt_sum.rf_cv_r5kf8_a <- summary(dt.rf_cv_r5kf8_a)
#
dt.rf_cv_r5kf8_a_finalModel$forest[11]
```

```
## $nrnodes
## [1] 2371
```

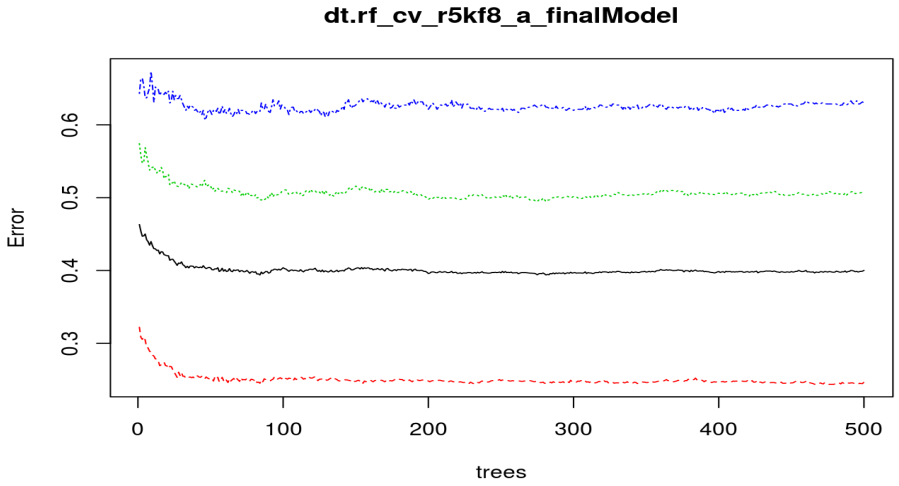
```
dt.rf_cv_r5kf8_a_finalModel$forest[12]
```

```
## $ntree
## [1] 500
```

```
#
plot(dt.rf_cv_r5kf8_a)
```



```
plot(dt.rf_cv_r5kf8_a_finalModel)
```



Random Forest Cross-validation testing

```
dt.rf_cv_r5kf8_a_test <- predict(dt.rf_cv_r5kf8_a, newdata =
holdout_age_groups.testing)
dt.rf_cv_r5kf8_a_test_prob <- predict(dt.rf_cv_r5kf8_a, newdata =
holdout_age_groups.testing, type = "prob")

dt.rf_cv_r5kf8_a_test_cm <- confusionMatrix(dt.rf_cv_r5kf8_a_test,
holdout_age_groups.testing$age_group)
kable(dt.rf_cv_r5kf8_a_test_cm$table)
```

	young	middle	old
young	650	6	3
middle	5	439	4
old	2	2	227

```
dt.rf_cv_r5kf8_a_test_cm$overall[1]


## Accuracy
## 0.9835575
```

Out of bag predictions

```
dt.rf_cv_r5kf8_a_test2 <- predict(dt.rf_cv_r5kf8_a)
dt.rf_cv_r5kf8_a_test2_prob <- predict(dt.rf_cv_r5kf8_a, newdata =
holdout_age_groups.testing, type = "prob")

dt.rf_cv_r5kf8_a_test2_cm <- confusionMatrix(dt.rf_cv_r5kf8_a_test2,
abalone_data_cleansed_age_groups$age_group)
kable(dt.rf_cv_r5kf8_a_test2_cm$table)
```

	young	middle	old
young	1955	28	13
middle	18	1324	13
old	5	5	658

```
        young   middle    old
dt.rf_cv_r5kf8_a_test2_cm$overall[1]
```

```
## Accuracy
## 0.9795969
```

While there appears to be little difference between the graphs depicting the nature of the two caret Random Forests, this second example using cross-validation has done a near perfect classification of the Abalone dataset by age grouping. While this seems like a useful result, it could well be that this is a result of incorrectly using the entire sample set with the Random Forest and assuming cross-validation would not cause overfitting. Consultation with a reliable source will be required to confirm this one way or another. With that in mind, yet another Random Forest will be trained using cross-validation but only with the holdout training data.

Random Forest Cross-validation with Holdout

```
dt.rf_cv2_r5kf8_a <- train(dt.formula, data = holdout_age_groups.training,
method = "rf", prox= TRUE, trControl = cv.train_control_8_5)
```

```
## note: only 2 unique complexity parameters in default grid. Truncating the
grid to 2 .
```

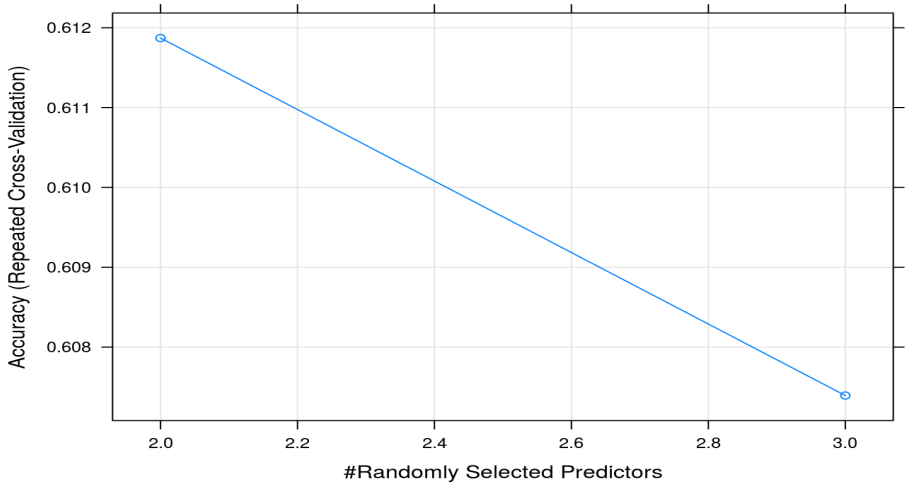
```
dt.rf_cv2_r5kf8_a_finalModel <- dt.rf_cv2_r5kf8_a$finalModel
dt_sum.rf_cv2_r5kf8_a <- summary(dt.rf_cv2_r5kf8_a)
#
dt.rf_cv2_r5kf8_a_finalModel$forest[11]
```

```
## $nrnodes
## [1] 1557
```

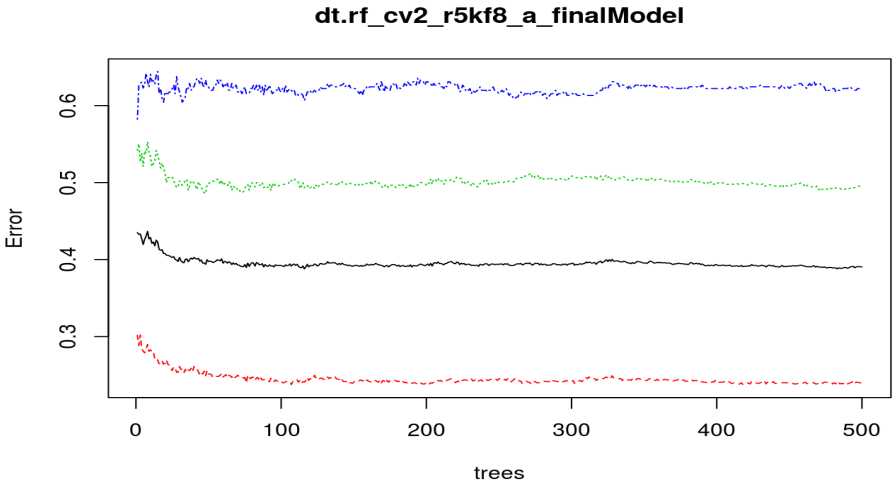
```
dt.rf_cv2_r5kf8_a_finalModel$forest[12]
```

```
## $ntree
## [1] 500
```

```
#
plot(dt.rf_cv2_r5kf8_a)
```




```
plot(dt.rf_cv2_r5kf8_a_finalModel)
```



Random Forest Cross-validation with Holdout testing

```
dt.rf_cv2_r5kf8_a_test <- predict(dt.rf_cv2_r5kf8_a, newdata =
holdout_age_groups.testing)
dt.rf_cv2_r5kf8_a_test_prob <- predict(dt.rf_cv2_r5kf8_a, newdata =
holdout_age_groups.testing, type = "prob")

dt.rf_cv2_r5kf8_a_test_cm <- confusionMatrix(dt.rf_cv2_r5kf8_a_test,
holdout_age_groups.testing$age_group)
kable(dt.rf_cv2_r5kf8_a_test_cm$table)
```

	young	middle	old
young	503	163	37
middle	116	208	116
old	38	76	81

```
dt.rf_cv2_r5kf8_a_test_cm$overall[1]
```

```
## Accuracy
## 0.5919283
```

The results of this latest Random Forest model appear near identical to that of the first, which applied bootstrapping to the holdout data. Given the similarities between the various Random Forests apart from the one that appears too perfect, it’s worth analysing one of these models later when comparing all models.

Random Forest Leave-one-out

As with the C4.5 decision tree, Leave-one-out Cross-validation is not suitable for this type of model.

Random Forest Decision Tree Conclusion

Random forests are good for avoiding over-fitting, so long as you know what you’re doing, particularly with regard to cross-validation.

Decision Tree Conclusion

Given all the data processing and feature selection work, the results from the decision tree have not been a fruitful as one might have hoped. Perhaps further investigation into more fine tuning could help yield better results.

Task 4: Build Train and Test a Naïve Bayes type Classifier

Premise

> You need to construct, train and test Naïve Bayes type classifier in R. Train and test your Naïve Bayes classifier using the training and test sets generated based on the methods tried as part of the 2nd Task.

Naïve Bayes is a relatively quick and simple probabilistic classifier that is often used as a benchmark (Brownlee, 2014b) for other forms of classification; if another technique is in some way superior, be that in terms of speed or accuracy, then it's worth using or sharing. If a proposed algorithm cannot improve on Naïve Bayes, then it needs further work, or consigned to only be of use to a very niche problem or set aside completely.

Naïve Bayes training

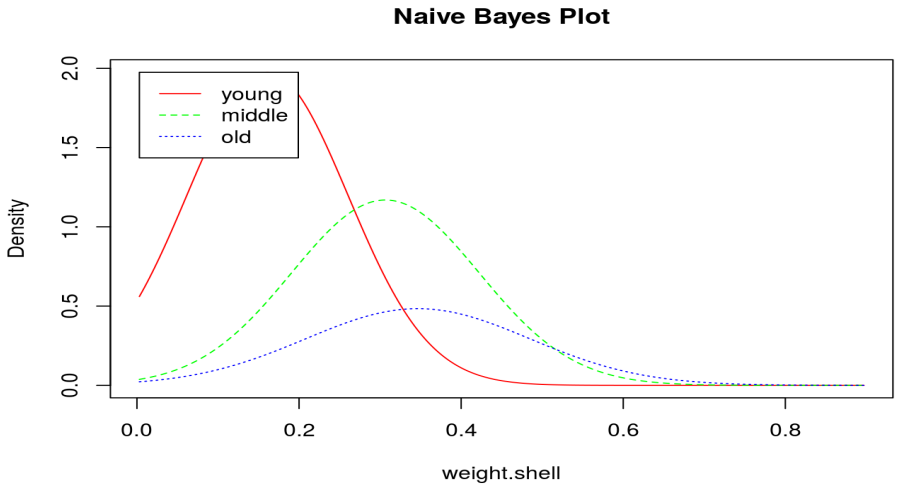
Naïve Bayes formula and simpler classification

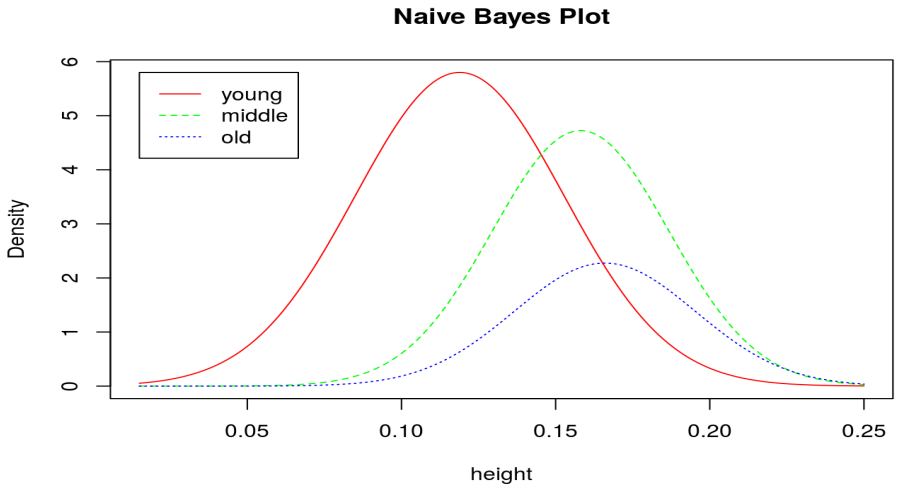
Precedent was set in the previous task to use a specific formula to target key predictors *(weight.shell, height, diameter)* and classify data instances based on a factor representing simplified age grouping; that precedent applies to the modelling in this chapter and following sections.

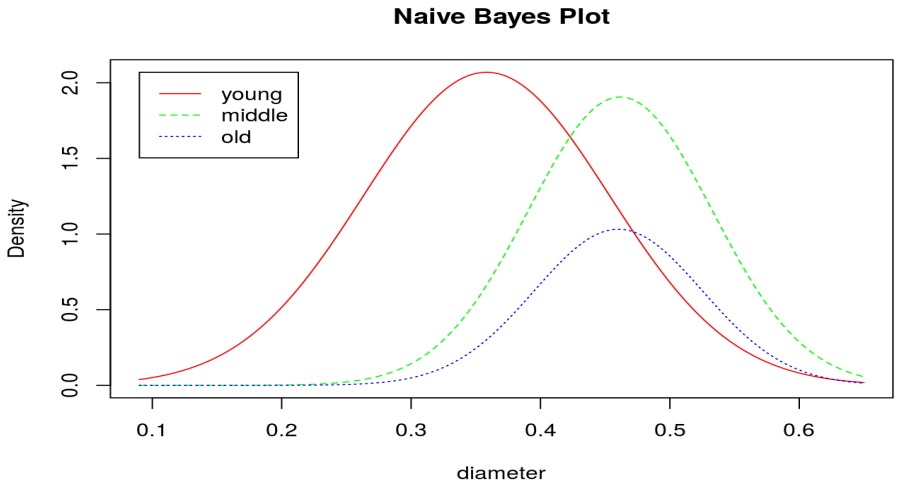
Naïve Bayes with holdout

```
nb.h_a <- NaiveBayes(dt.formula, data=holdout_age_groups.training)
```

```
plot(nb.h_a)
```







What is interesting about these Naïve Bayes plots is that they suggest the merit of using this technique to select features that could be used when creating a formula for use with another model. Another aspect worth noting is how the *height* and *diameter* probability graphs are extremely similar, which could suggest redundancy, where another feature could have been used as a classifier variable; in addition to this, the location of the curves for *middle* and *old* on these graphs share a similar range on the x axis, highlighting how difficult it is to differentiate between the *middle* and *old* age groups based on these features. Searching for another feature that has bell curves located more discretely (not occupying the same space) on the graphs, would probably suggest a feature that could provide more successful classification.

Naïve Bayes with holdout, training results

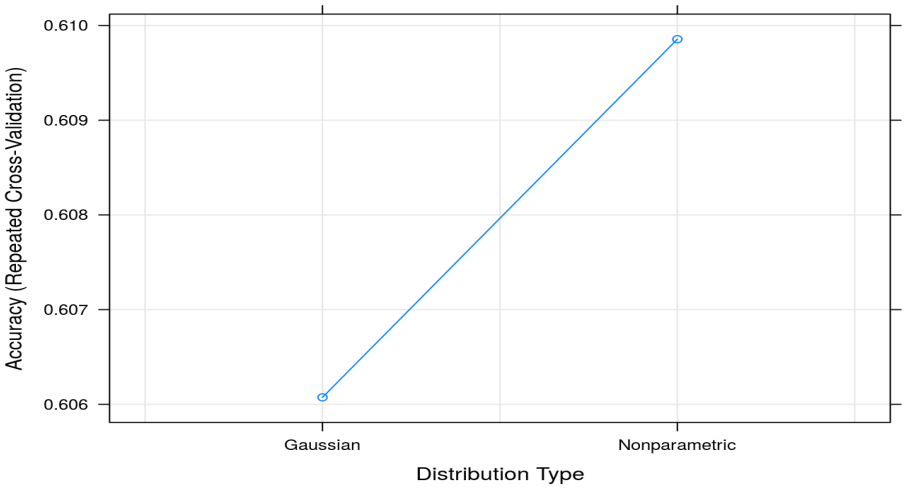
```
nb.h_a_test <- predict(nb.h_a, holdout_age_groups.testing)
nb.h_a_test_prob <- predict(nb.h_a, holdout_age_groups.testing, type = "prob")
# summarize results
nb.h_a_test_cm <- confusionMatrix(nb.h_a_test$class,
holdout_age_groups.testing$age_group)
nb.h_a_test_cm
```

```
## Confusion Matrix and Statistics
##
## Reference
## Prediction young middle old
## young 468 121 45
## middle 186 293 149
## old 3 33 40
##
## Overall Statistics
##
## Accuracy : 0.5987
## 95% CI : (0.5718, 0.6251)
## No Information Rate : 0.491
## P-Value [Acc > NIR] : 1.811e-15
##
## Kappa : 0.3318
## Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
## Class: young Class: middle Class: old
## Sensitivity 0.7123 0.6555 0.1709
## Specificity 0.7562 0.6240 0.9674
## Pos Pred Value 0.7382 0.4666 0.5263
## Neg Pred Value 0.7315 0.7831 0.8463
## Prevalence 0.4910 0.3341 0.1749
## Detection Rate 0.3498 0.2190 0.0299
## Detection Prevalence 0.4738 0.4694 0.0568
## Balanced Accuracy 0.7343 0.6397 0.5692
```

Naïve Bayes with Repeated K-folds Cross-validation

```
nb.r5kf8_a <- train(dt.formula, data=abalone_data_cleansed_age_groups,
trControl = cv.train_control_8_5, method="nb")
```

```
plot(nb.r5kf8_a)
```



Naïve Bayes with Repeated K-folds, training results

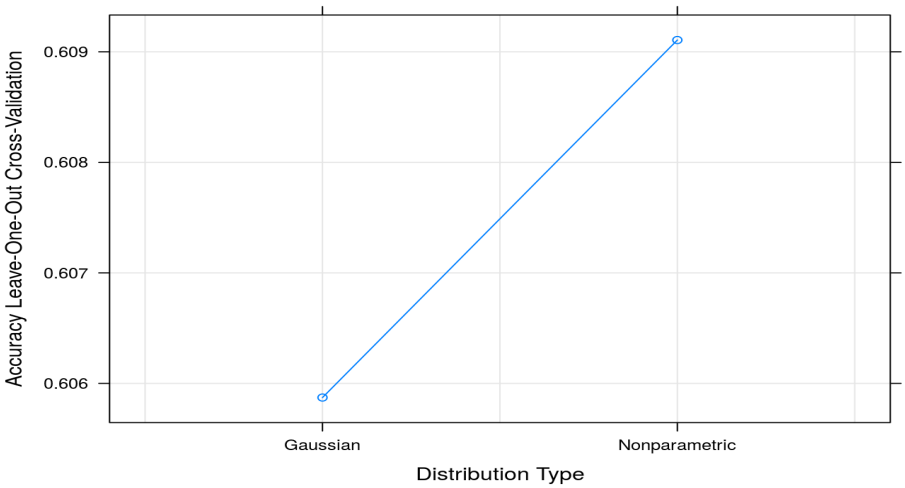
```
nb.r5kf8_a_test <- predict(nb.r5kf8_a, holdout_age_groups.testing)
nb.r5kf8_a_test_prob <- predict(nb.r5kf8_a, holdout_age_groups.testing, type =
"prob")
# summarize results
nb.r5kf8_a_test_cm <- confusionMatrix(nb.r5kf8_a_test,
holdout_age_groups.testing$age_group)
nb.r5kf8_a_test_cm
```

```
## Confusion Matrix and Statistics
##
## Reference
## Prediction young middle old
## young 481 128 53
## middle 174 288 145
## old 2 31 36
##
## Overall Statistics
##
## Accuracy : 0.6016
## 95% CI : (0.5748, 0.628)
## No Information Rate : 0.491
## P-Value [Acc > NIR] : 3.029e-16
##
## Kappa : 0.3321
## Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
## Class: young Class: middle Class: old
## Sensitivity 0.7321 0.6443 0.15385
## Specificity 0.7342 0.6420 0.97011
## Pos Pred Value 0.7266 0.4745 0.52174
## Neg Pred Value 0.7396 0.7825 0.84397
## Prevalence 0.4910 0.3341 0.17489
## Detection Rate 0.3595 0.2152 0.02691
## Detection Prevalence 0.4948 0.4537 0.05157
## Balanced Accuracy 0.7332 0.6431 0.56198
```

Naïve Bayes with Leave-one-out Cross-validation

```
nb.loo_a <- train(dt.formula, data=abalone_data_cleansed_age_groups, trControl
= loocv.train_control, method="nb")
```

```
plot(nb.loo_a)
```



Naïve Bayes with Leave-one-out, training results

```
nb.loo_a_test <- predict(nb.loo_a, holdout_age_groups.testing)
# summarize results
nb.loo_a_test_cm <- confusionMatrix(nb.loo_a_test,
holdout_age_groups.testing$age_group)
nb.loo_a_test_cm
```

```
## Confusion Matrix and Statistics
##
## Reference
## Prediction young middle old
## young 481 128 53
## middle 174 288 145
## old 2 31 36
##
## Overall Statistics
##
## Accuracy : 0.6016
## 95% CI : (0.5748, 0.628)
## No Information Rate : 0.491
## P-Value [Acc > NIR] : 3.029e-16
##
## Kappa : 0.3321
## Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
## Class: young Class: middle Class: old
## Sensitivity 0.7321 0.6443 0.15385
## Specificity 0.7342 0.6420 0.97011
## Pos Pred Value 0.7266 0.4745 0.52174
## Neg Pred Value 0.7396 0.7825 0.84397
## Prevalence 0.4910 0.3341 0.17489
## Detection Rate 0.3595 0.2152 0.02691
## Detection Prevalence 0.4948 0.4537 0.05157
## Balanced Accuracy 0.7332 0.6431 0.56198
```

Accuracy comparison of NB models

```
nb.h_a_test_cm$overall[1]
```

```
## Accuracy
## 0.5986547
```

```
nb.r5kf8_a_test_cm$overall[1]
```

```
## Accuracy
## 0.6016442
```

```
nb.loo_a_test_cm$overall[1]
```

```
## Accuracy
## 0.6016442
```

Naïve Bayes Conclusion

While this technique doesn't produce the strongest results, it's a useful tool alongside other models which can be used for preparatory work in order to inform further work on modelling a classification problem. It's a fast way to do preliminary analysis, which can benefit somewhat from cross-validation but for the Abalone dataset at least, using the leave-one-out method bore no extra benefit over k-folds.

When compared to the decision trees, the Naïve Bayes models perform similarly, only marginally worse in some cases; the one classification out of the three age groups where they underperform by a notable amount is the *'old'* group; it's not entirely clear why this is but it is likely that while the chosen features have a direct correlation to age, the coefficients are not pronounced enough to make classification any better than it is.

Task 5: Build Train and Test a K-NN type Classifier

Premise

> You need to construct, train and test K-NN type classifier in R. Train and test your K-NN classifier using the training and test sets generated based on the methods tried as part of the 2nd Task.

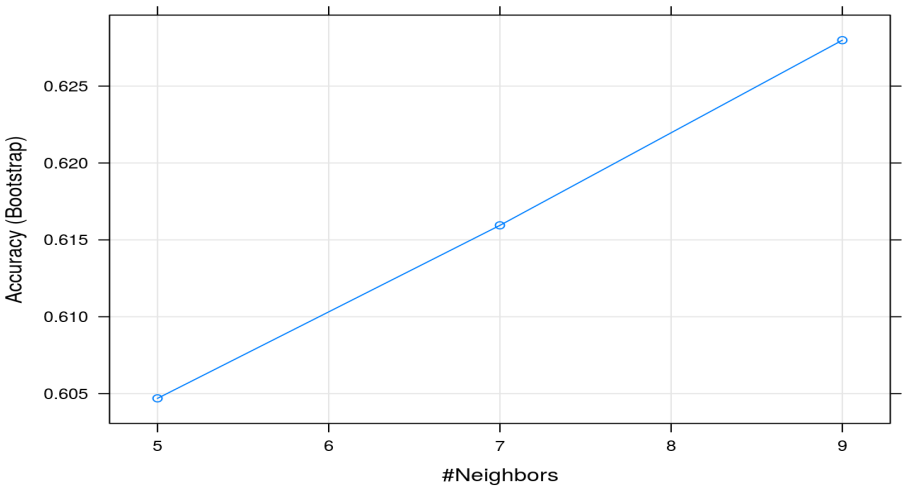
KNN modelling

K-Nearest-Neighbours looks to classify new data based on its dimensional proximity to an already classified node or group of nodes (Srivastava, 2014a, Raut (2017)). This use of dimensional distance allows it to be used for regression but every new classification needs to be computed against the relative distance to the instances within the training dataset; this makes it fast to train because a lot of the computation is deferred but then using this method to classify is more expensive because every new classification is calculated at run-time instead of relying on an established rule-set.

KNN with holdout

```
knn.h_a <- train(dt.formula, data=holdout_age_groups.training, method = "knn")
```

```
plot(knn.h_a)
```



KNN with holdout, testing

```
knn.h_a_test <- predict(knn.h_a, holdout_age_groups.testing)
knn.h_a_test_prob <- predict(knn.h_a, holdout_age_groups.testing, type =
"prob")
# summarize results
knn.h_a_test_cm <- confusionMatrix(knn.h_a_test,
```

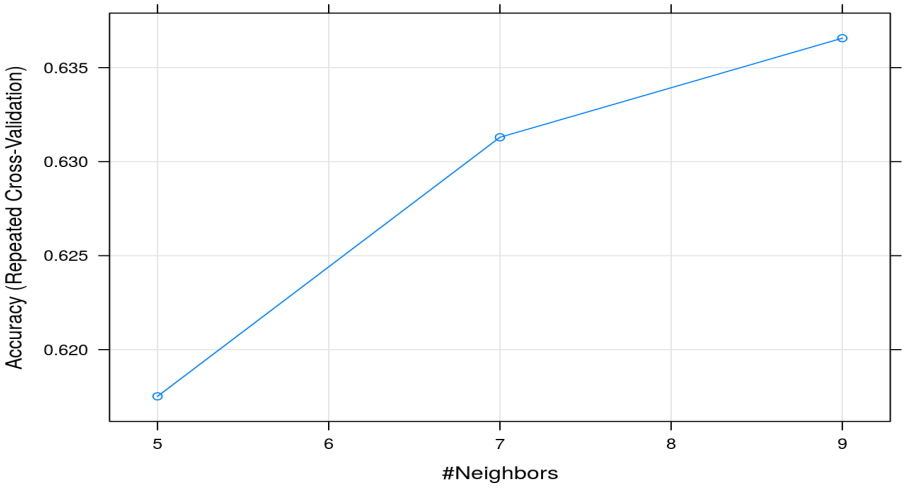
```
holdout_age_groups.testing$age_group)
knn.h_a_test_cm$table
```

```
## Reference
## Prediction young middle old
## young 531 160 35
## middle 106 233 131
## old 20 54 68
```

KNN with Repeated K-folds Cross-validation

```
knn.r5kf8_a <- train(dt.formula, data=abalone_data_cleansed_age_groups,
trControl = cv.train_control_8_5, method = "knn")
```

```
plot(knn.r5kf8_a)
```



KNN with Repeated K-folds Cross-validation, testing

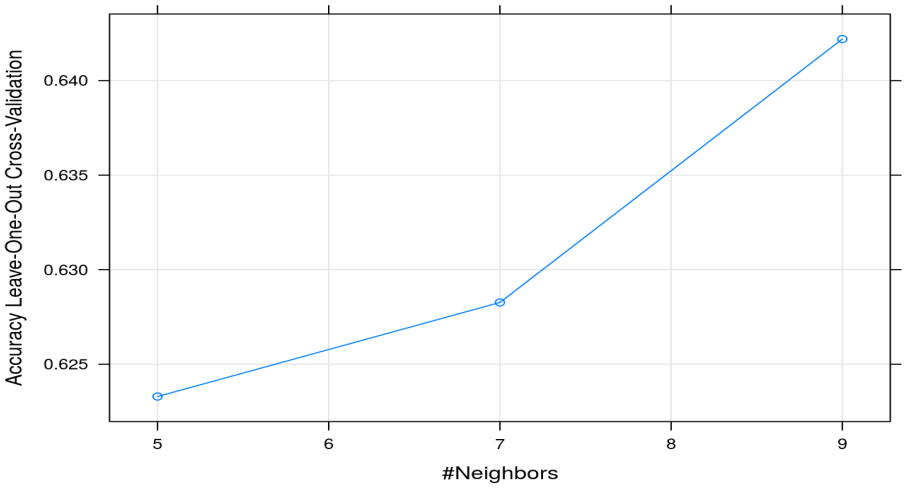
```
knn.r5kf8_a_test <- predict(knn.r5kf8_a, holdout_age_groups.testing)
knn.r5kf8_a_test_prob <- predict(knn.r5kf8_a, holdout_age_groups.testing, type
= "prob")
# summarize results
knn.r5kf8_a_test_cm <- confusionMatrix(knn.r5kf8_a_test,
holdout_age_groups.testing$age_group)
knn.r5kf8_a_test_cm$table
```

```
## Reference
## Prediction young middle old
## young 541 127 36
## middle 99 279 97
## old 17 41 101
```

KNN with Leave-one-out Cross-validation

```
knn.loo_a <- train(dt.formula, data=abalone_data_cleansed_age_groups,
trControl = loocv.train_control, method="knn")
```

```
plot(knn.loo_a)
```



KNN with Leave-one-out, training results

```
knn.loo_a_test <- predict(knn.loo_a, holdout_age_groups.testing)
# summarize results
knn.loo_a_test_cm <- confusionMatrix(knn.loo_a_test,
holdout_age_groups.testing$age_group)
knn.loo_a_test_cm$table
```

```
## Reference
## Prediction young middle old
## young 543 131 34
## middle 94 276 98
## old 20 40 102
```

Accuracy comparison of KNN models

```
knn.h_a_test_cm$overall[1]
```

```
## Accuracy
## 0.6218236
```

```
knn.r5kf8_a_test_cm$overall[1]
```

```
## Accuracy
## 0.6883408
```

```
knn.loo_a_test_cm$overall[1]
```

```
## Accuracy
## 0.6883408
```

KNN model conclusion

Once again, sufficient cross-validation appears to be about as good as using the leave-one-out technique for improving the accuracy of the model but cheaper to train than leave-one-out. In this case, there’s quite a marked jump in improvement between the holdout version and the other two forms of training. The holdout KNN is only marginally more accurate than the Naïve Bayes while the k-folds and leave-one-out methods are more than 10% more accurate. The results suggest that KNN could be the most suitable model for age group classification of Abalone.

Task 6: Measure Performance

Premise

For each type of classifier calculate and display the following performance related metrics in R. Use the library ROCR 1 Confusion matrix 2 Precision vs. Recall 3 Accuracy estimation 4 ROC (receiver operating characteristic curve) 5 RAUC (receiver under the curve area)

Confusion matrix comparison

Every model has had a confusion matrix displayed within its respective section; they are not presented here together, as the following graphs do a better job of indicating to the human eye which models are the most successful. The confusion matrix was instrumental in selecting which models were best in their field but the other metrics provide a better way to convey performance at a glance, visualising with 2 dimensions the nature of the models.

Precision & Recall comparison

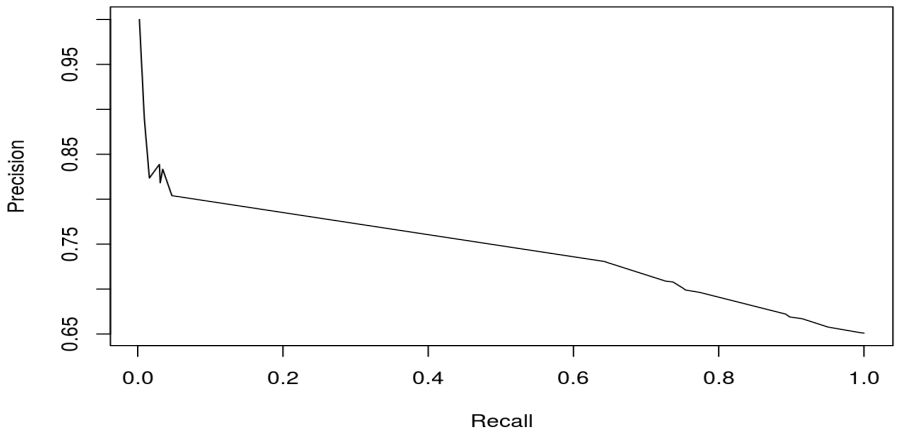
The precision-recall plot is a model-wide evaluation measure that is based on two basic evaluation measures – recall and precision. Recall is a performance measure of the whole positive part of a dataset, whereas precision is a performance measure of positive predictions. (Saito and Rehmsmeier, 2015)

Precision is the measure of result confidence, while recall is the measure true positive results returned. The precision-recall curve shows the tradeoff between precision and recall at different thresholds.

C4.5 Decision Tree candidate, 8 folds & 5 repetitions

```
pred_dt.c4_5_highest_probs <- apply(dt.c4_5_j48_r5kf8_a_test_prob, 1, "max")
pred_dt.c4_5_correct <- dt.c4_5_j48_r5kf8_a_test ==
holdout_age_groups.testing$age_group

pred_dt.c4_5 <- prediction(pred_dt.c4_5_highest_probs, pred_dt.c4_5_correct)
```

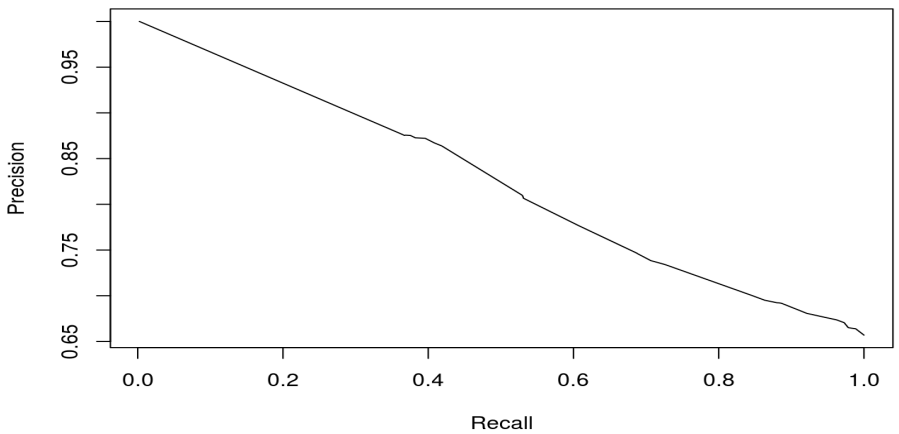
```
## precision/recall curve (x-axis: recall, y-axis: precision)
pr_dt.c4_5 <- performance(pred_dt.c4_5, "prec", "rec")
plot(pr_dt.c4_5)
```



This is a strange line but appears to suggest that the precision of this model drops off dramatically as there's a lot of overlap between false and true positives.

C4.5 Decision Tree candidate, 13 folds & 8 repetitions

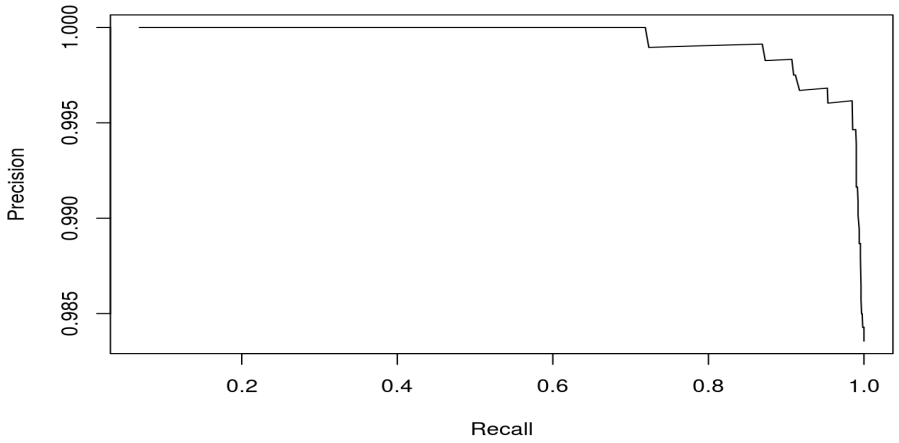
```
pred_dt.c4_5_r8kf13_highest_probs <- apply(dt.c4_5_j48_r8kf13_a_test_prob, 1,
"max")
pred_dt.c4_5_r8kf13_correct <- dt.c4_5_j48_r8kf13_a_test ==
holdout_age_groups.testing$age_group

pred_dt.c4_5_r8kf13 <- prediction(pred_dt.c4_5_r8kf13_highest_probs,
pred_dt.c4_5_r8kf13_correct)

## precision/recall curve (x-axis: recall, y-axis: precision)
pr_dt.c4_5_r8kf13 <- performance(pred_dt.c4_5_r8kf13, "prec", "rec")
plot(pr_dt.c4_5_r8kf13)
```



This line suggests a pretty linear relationship between precision & recall, which is at least better than the previous graph but a better line would remain higher on the *y* only dropping off towards *1.0* on the *x*.

Random Forest Decision Tree repeated k-folds candidate (possibly overfit)

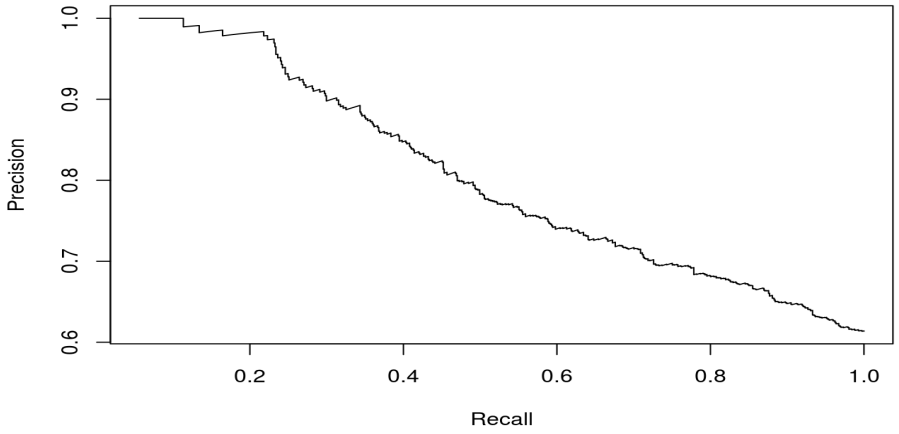
```
pred_dt.rf_highest_probs <- apply(dt.rf_cv_r5kf8_a_test_prob, 1, "max")
pred_dt.rf_correct <- dt.rf_cv_r5kf8_a_test ==
holdout_age_groups.testing$age_group

pred_dt.rf <- prediction(pred_dt.rf_highest_probs, pred_dt.rf_correct)

## precision/recall curve (x-axis: recall, y-axis: precision)
pr_dt.rf <- performance(pred_dt.rf, "prec", "rec")
plot(pr_dt.rf)
```



This is the (possibly) over-fit example which shows an example of what the line should be like if precision of classification stayed more or less constant as recall scales.

Random Forest Decision Tree holdout candidate

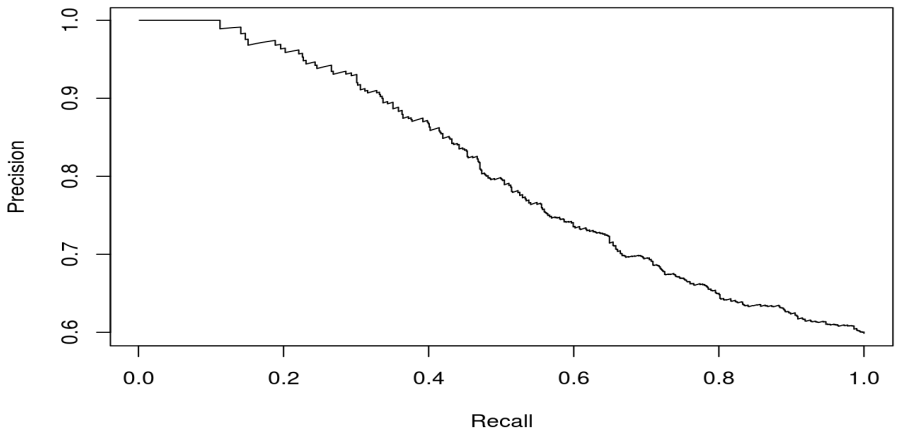
```
#dt.hrf2_h_a_test_prob
pred_dt.hrf2_h_highest_probs <- apply(dt.hrf2_h_a_test_prob$predictions, 1,
"max")
pred_dt.hrf2_h_correct <- dt.hrf2_h_a_test$predictions ==
holdout_age_groups.testing$age_group

pred_dt.hrf2_h <- prediction(pred_dt.hrf2_h_highest_probs,
pred_dt.hrf2_h_correct)

## precision/recall curve (x-axis: recall, y-axis: precision)
pr_dt.hrf2_h <- performance(pred_dt.hrf2_h, "prec", "rec")
plot(pr_dt.hrf2_h)
```



This line shows a saw-tooth pattern that is apparently common in this sort of graph; the slope is approximately the same as the better C4.5 line though the area under the line might well be a little greater, indicating slightly better performance.

Naïve Bayes holdout candidate

```
pred_nb.h_highest_probs <- apply(nb.h_a_test_prob$posterior, 1, "max")
pred_nb.h_correct <- nb.h_a_test_prob$class ==
holdout_age_groups.testing$age_group

pred_nb.h <- prediction(pred_nb.h_highest_probs, pred_nb.h_correct)

## precision/recall curve (x-axis: recall, y-axis: precision)
pr_nb.h <- performance(pred_nb.h, "prec", "rec")
plot(pr_nb.h)
```

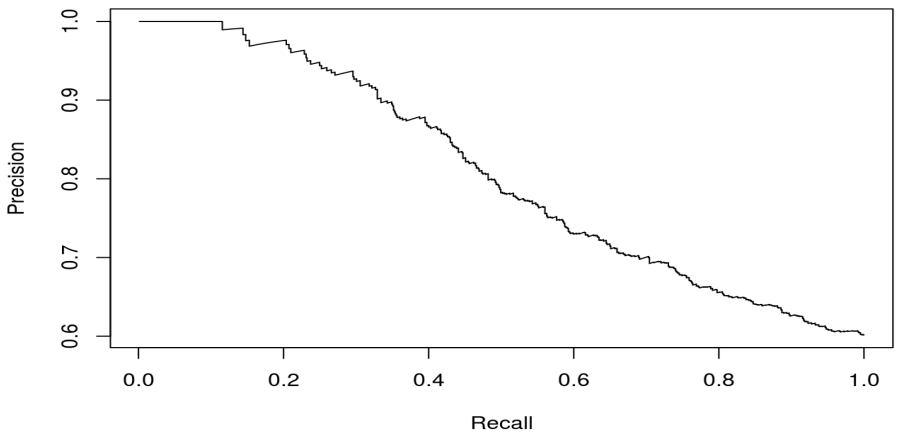


This graph is interesting as it suggests an initially good rate of success which literally tails off.

Naïve Bayes repeat k-folds cross-validation candidate

```
pred_nb.r5kf8_highest_probs <- apply(nb.r5kf8_a_test_prob, 1, "max")
pred_nb.r5kf8_correct <- nb.r5kf8_a_test ==
holdout_age_groups.testing$age_group

pred_nb.r5kf8 <- prediction(pred_nb.r5kf8_highest_probs,
pred_nb.r5kf8_correct)
```

```
## precision/recall curve (x-axis: recall, y-axis: precision)
pr_nb.r5kf8 <- performance(pred_nb.r5kf8, "prec", "rec")
plot(pr_nb.r5kf8)
```

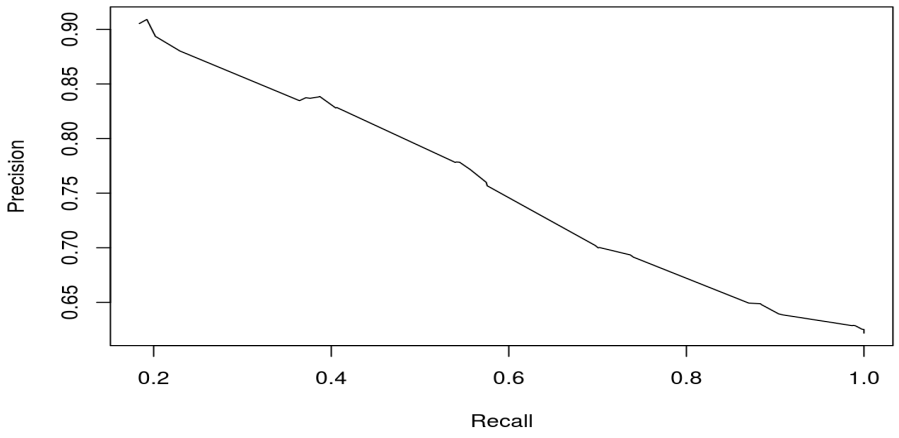


When comparing this line to the line of the previous graph, it's possible to see how cross validation manages to improve the line, increasing the area below the line ever so slightly.

KNN candidates

KNN holdout candidate

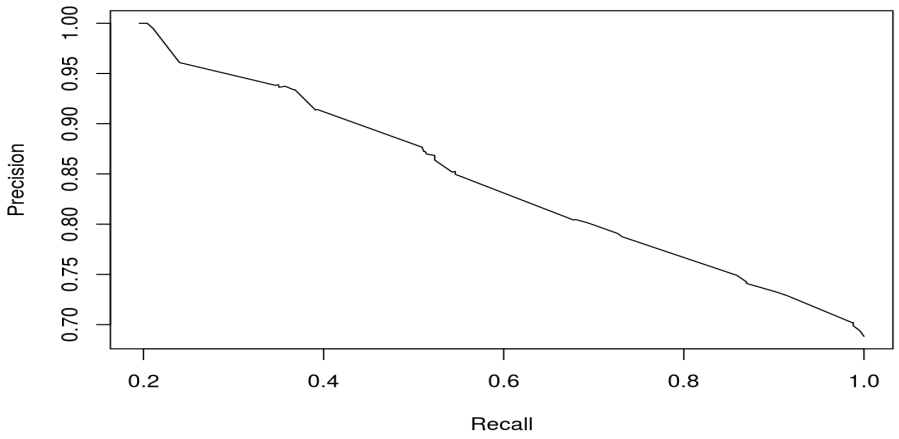
```
pred_knn.h_highest_probs <- apply(knn.h_a_test_prob, 1, "max")
pred_knn.h_correct <- knn.h_a_test == holdout_age_groups.testing$age_group

pred_knn.h <- prediction(pred_knn.h_highest_probs, pred_knn.h_correct)

## precision/recall curve (x-axis: recall, y-axis: precision)
pr_knn.h <- performance(pred_knn.h, "prec", "rec")
plot(pr_knn.h)
```



This line would appear to suggest slightly worse than a linear, negative coefficient; the highest precision value is only about *0.75*, so this doesn't suggest good results where precision is concerned.

KNN repeated k-folds cross-validation candidate

```
pred_knn.r5kf8_highest_probs <- apply(knn.r5kf8_a_test_prob, 1, "max")
pred_knn.r5kf8_correct <- knn.r5kf8_a_test ==
holdout_age_groups.testing$age_group

pred_knn.r5kf8 <- prediction(pred_knn.r5kf8_highest_probs,
pred_knn.r5kf8_correct)

## precision/recall curve (x-axis: recall, y-axis: precision)
pr_knn.r5kf8 <- performance(pred_knn.r5kf8, "prec", "rec")
plot(pr_knn.r5kf8)
```



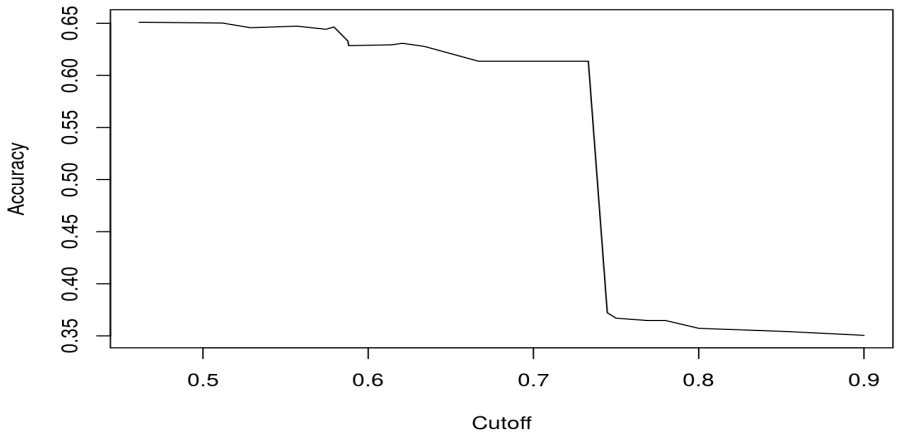
When compared to the previous KNN PRC, this supports the recommendation of cross-validation to improve precision; this also manifests in an increase of the area under the line.

Accuracy estimation comparison

The following graphs represent the overall accuracy of the model; like with all these graphs, the area under the line is important with the larger volume being preferred.

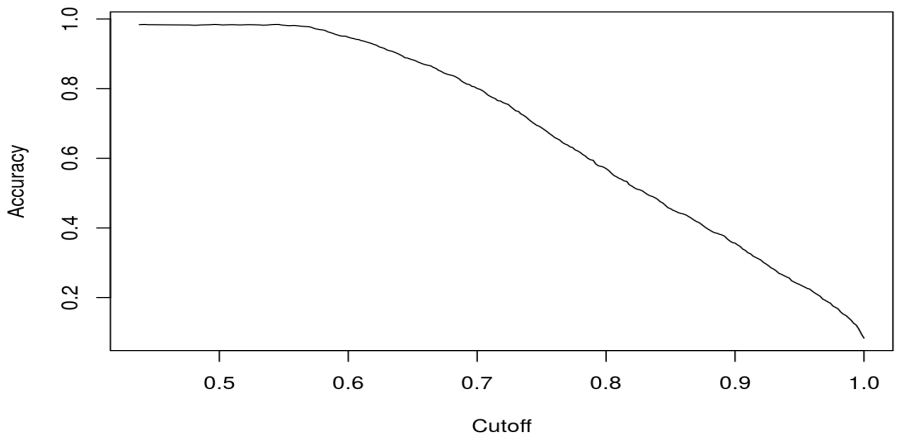
C4.5 Decision Tree candidate

```
acc_dt.c4_5 <- performance(pred_dt.c4_5, "acc")
plot(acc_dt.c4_5)
```



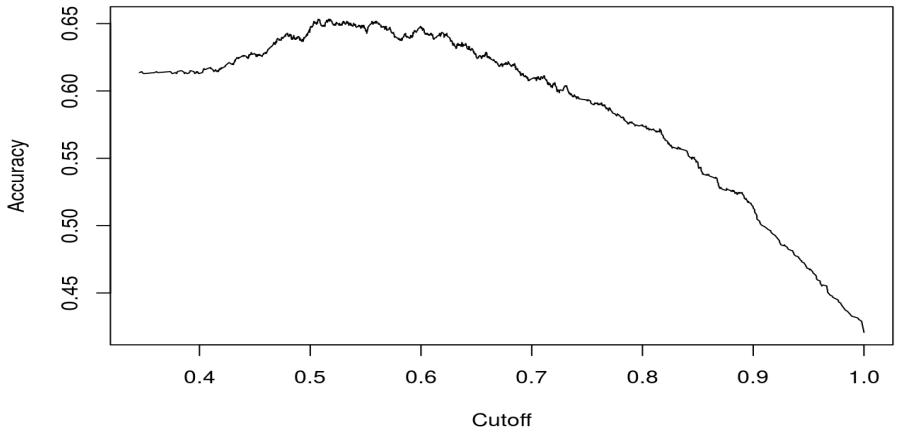
Random Forest Decision Tree k-folds candidate (possibly overfit)

```
acc_dt.rf <- performance(pred_dt.rf, "acc")
plot(acc_dt.rf)
```



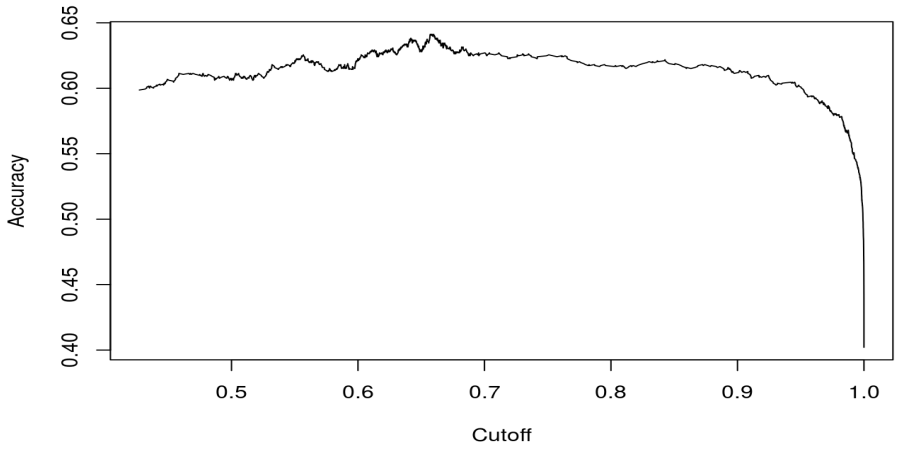
Random Forest Decision Tree holdout candidate

```
acc_dt.hrf2_h <- performance(pred_dt.hrf2_h, "acc")
plot(acc_dt.hrf2_h)
```



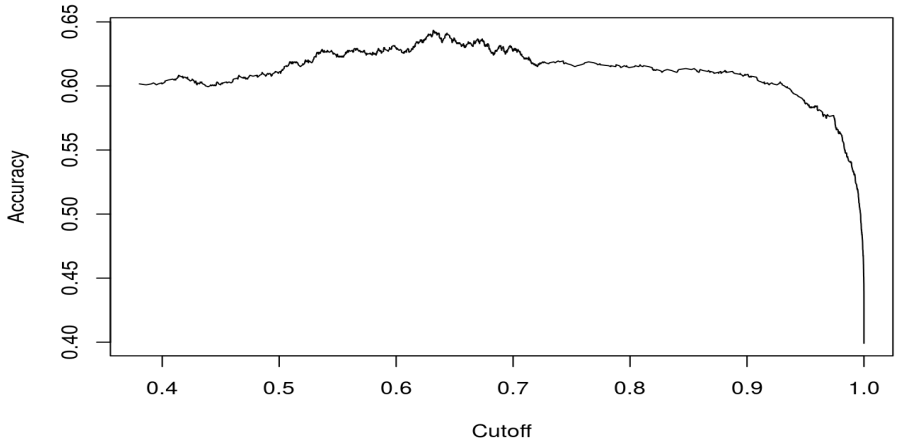
Naïve Bayes holdout candidate

```
acc_nb.h <- performance(pred_nb.h, "acc")
plot(acc_nb.h)
```



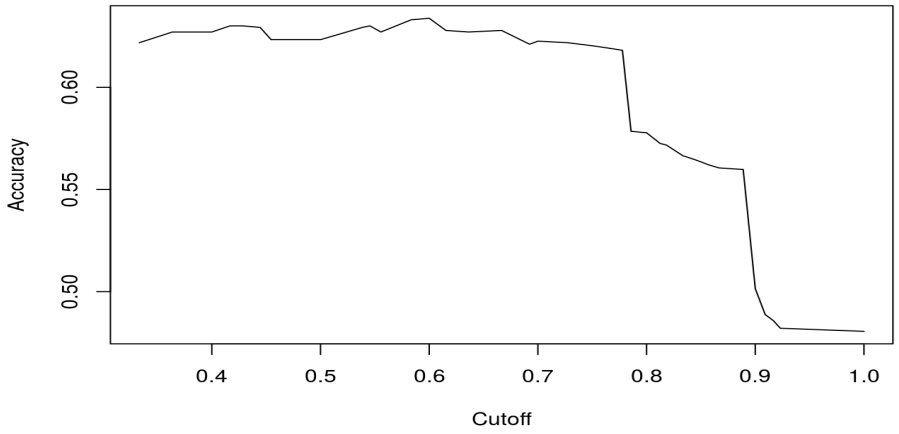
Naïve Bayes repeated k-folds cross-validation candidate

```
acc_nb.r5kf8 <- performance(pred_nb.r5kf8, "acc")
plot(acc_nb.r5kf8)
```



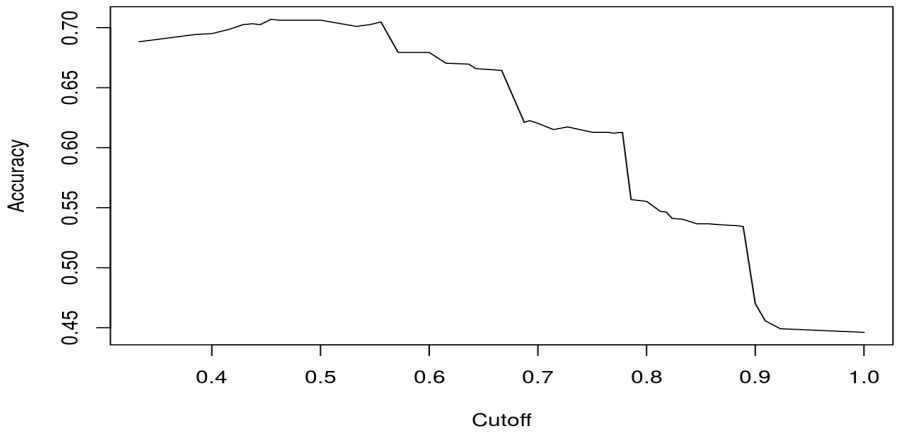
KNN holdout candidate

```
acc_knn.h <- performance(pred_knn.h, "acc")
plot(acc_knn.h)
```



KNN repeated k-folds cross-validation candidate

```
acc_knn.r5kf8 <- performance(pred_knn.r5kf8, "acc")
plot(acc_knn.r5kf8)
```



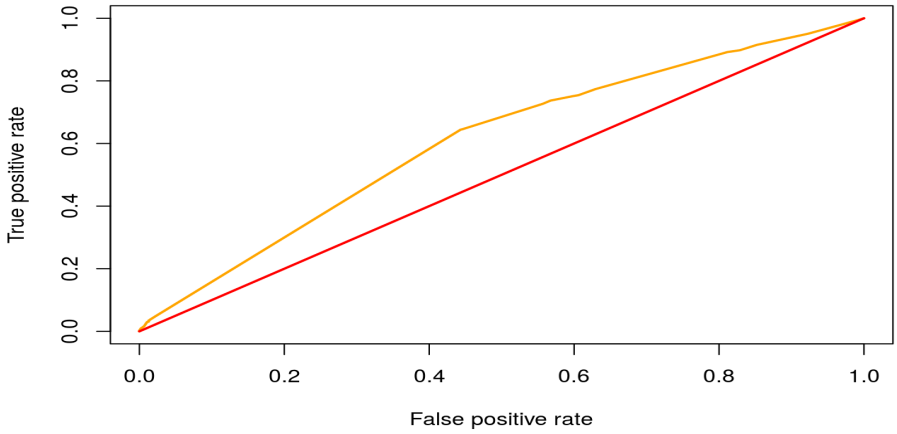
ROC comparison

The Receiver-operating-characteristic compares the rate of true positives with false positives; in graph form true positive rate is on the *y* axis, while *x* maps the false positive rate. The better the model is at performing classification the larger the area under the line and the more ‘rectangular’ in shape, with the best performing results being where the line is closest to the top left corner (Schoonjans, no date). The better the model performs, the higher the true positive relative to the false positive.

In the following ROC graphs, a red line indicates what model performance would be like were a model to have a 50% change of correctly classifying a data instance. The yellow line is the relationship between true-positive-rate (TPR) and false-positive-rate (FPR).

C4.5 Decision Tree candidate

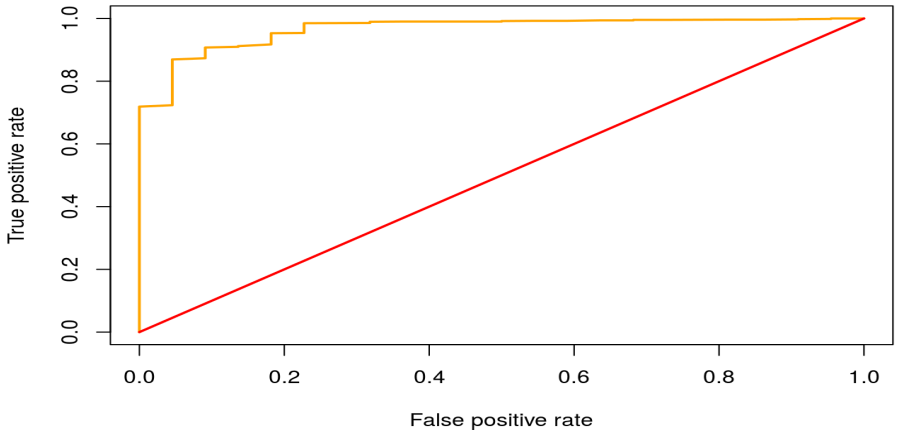
```
roc_dt.c4_5 = performance(pred_dt.c4_5, measure="tpr", x.measure="fpr")
plot(roc_dt.c4_5, col="orange", lwd=2)
lines(x=c(0, 1), y=c(0, 1), col="red", lwd=2)
```



This doesn't seem to be a particularly good result, though it is somewhat better than a 50:50 chance of correct classification, with peak performance easily identifiable; the model has the best rate of performance around the middle of the line where there is the best balance between the TPR and the coefficient with FPR, with TPR at over 0.6 and about 50% better than the FPR. After the peak point, the graph suggests accuracy goes down.

Random Forest Decision Tree k-folds candidate (possibly overfit)

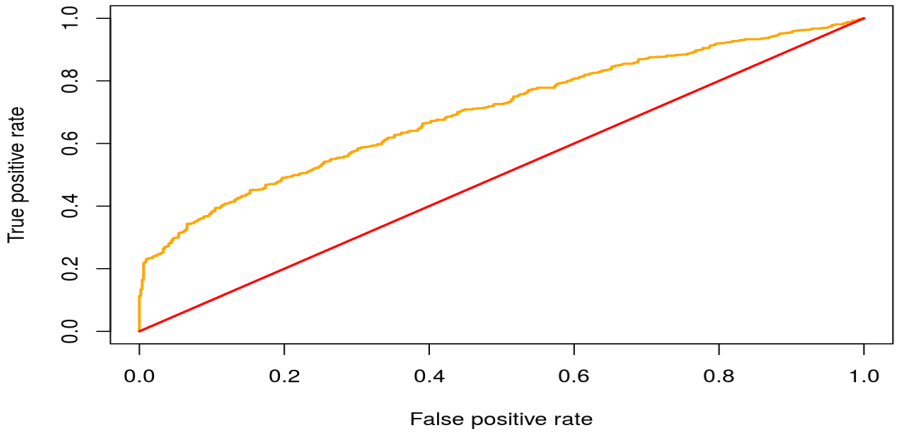
```
roc_dt.rf = performance(pred_dt.rf, measure="tpr", x.measure="fpr")
plot(roc_dt.rf, col="orange", lwd=2)
lines(x=c(0, 1), y=c(0, 1), col="red", lwd=2)
```



This graph is a near perfect example, which has been included even though it may demonstrate overfitting; this was because the test data was used as part of the training data. The yellow line draws a blocky, rectangular shape indicating a high rate of successful classification; if this was not the result of overfitting it would be an excellent result; Further testing with new data would help determine this.

Random Forest Decision Tree holdout candidate

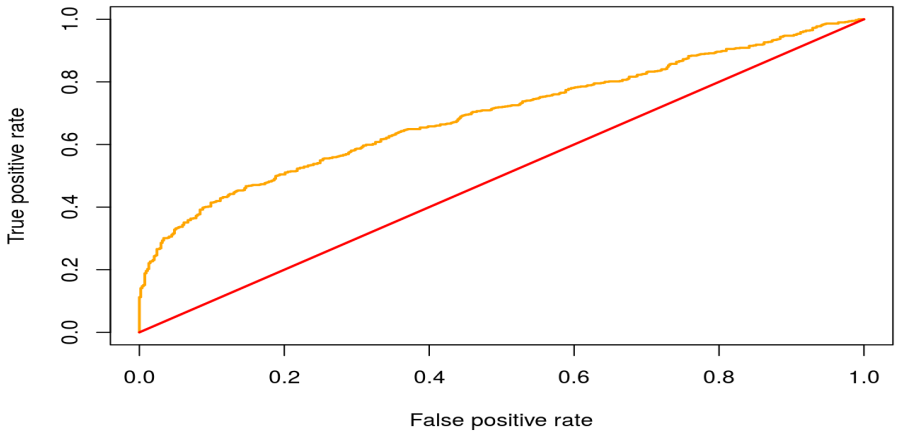
```
roc_dt.hrf2_h = performance(pred_dt.hrf2_h, measure="tpr", x.measure="fpr")
plot(roc_dt.hrf2_h, col="orange", lwd=2)
lines(x=c(0, 1), y=c(0, 1), col="red", lwd=2)
```



This is a far more conservative result but it is at least, markedly better than the C4.5 decision tree. What's worth doing is comparing this result to the Naïve Bayes graphs below, which happen to have similar looking lines.

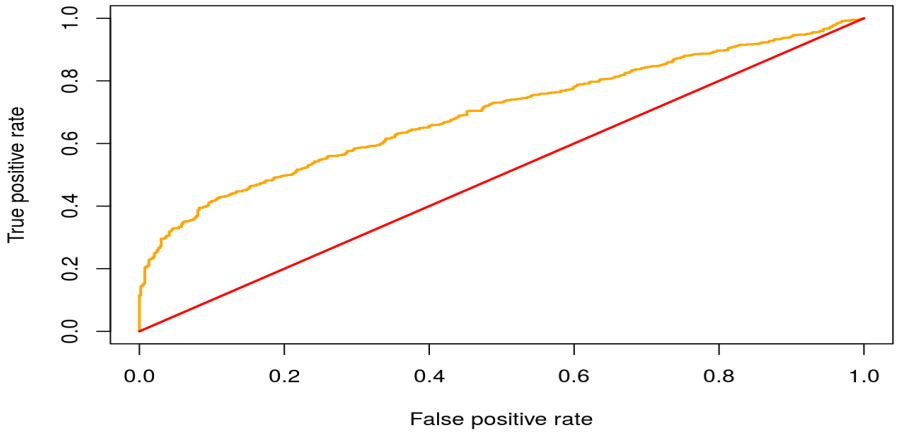
Naïve Bayes holdout candidate

```
roc_nb.h = performance(pred_nb.h, measure="tpr", x.measure="fpr")
plot(roc_nb.h, col="orange", lwd=2)
lines(x=c(0, 1), y=c(0, 1), col="red", lwd=2)
```



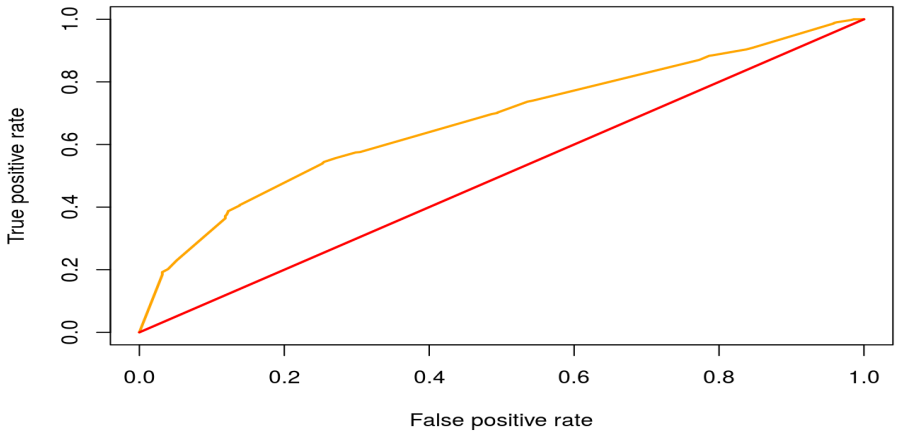
Naïve Bayes repeated k-folds cross-validation candidate

```
roc_nb.r5kf8 = performance(pred_nb.r5kf8, measure="tpr", x.measure="fpr")
plot(roc_nb.r5kf8, col="orange", lwd=2)
lines(x=c(0, 1), y=c(0, 1), col="red", lwd=2)
```



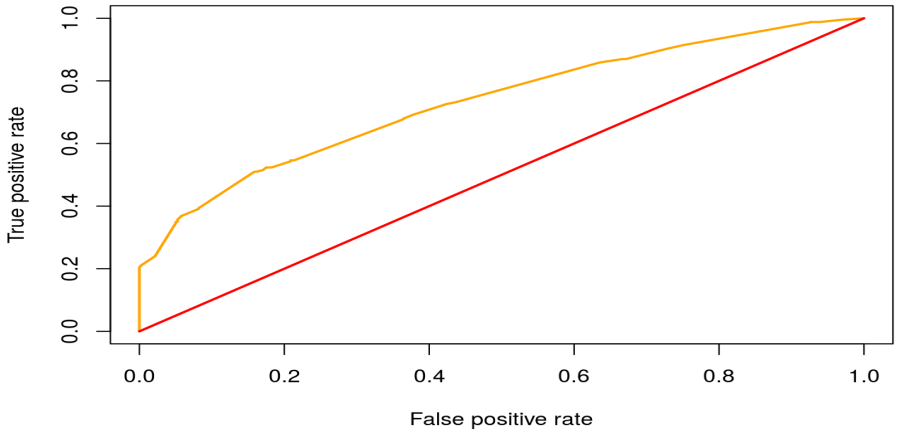
KNN holdout candidate

```
roc_knn.h = performance(pred_knn.h, measure="tpr", x.measure="fpr")
plot(roc_knn.h, col="orange", lwd=2)
lines(x=c(0, 1), y=c(0, 1), col="red", lwd=2)
```



KNN repeated k-folds cross-validation candidate

```
roc_knn.r5kf8 = performance(pred_knn.r5kf8, measure="tpr", x.measure="fpr")
plot(roc_knn.r5kf8, col="orange", lwd=2)
lines(x=c(0, 1), y=c(0, 1), col="red", lwd=2)
```



RAUC comparison

The following values essentially represent the volume under the lines for the corresponding ROC curves, thus reducing the performance into one number between *0* and *1*, where *1* would indicate perfect classification, *0* would represent incorrect classification 100% of the time and the red line on the ROC graphs above would essentially resolve to a value of *0.5*. It can be considered another form of accuracy indicator in this way (ROC curves and Area Under the Curve explained (video), 2014).

C4.5 Decision Tree candidate

```
auc_dt.c4_5 = performance(pred_dt.c4_5, "auc")
print(paste(c(auc_dt.c4_5@y.name, auc_dt.c4_5@y.values)), quote=FALSE)
```

```
## [1] Area under the ROC curve 0.608080500151196
```

Random Forest Decision Tree k-folds candidate (possibly overfit)

```
auc_dt.rf = performance(pred_dt.rf, "auc")
print(paste(c(auc_dt.rf@y.name, auc_dt.rf@y.values)), quote=FALSE)
```

```
## [1] Area under the ROC curve 0.965840011052777
```

Random Forest Decision Tree holdout candidate

```
auc_dt.hrf2_h = performance(pred_dt.hrf2_h, "auc")
print(paste(c(auc_dt.hrf2_h@y.name, auc_dt.hrf2_h@y.values)), quote=FALSE)
```

```
## [1] Area under the ROC curve 0.698969506922962
```

Naïve Bayes holdout candidate

```
auc_nb.h = performance(pred_nb.h, "auc")
print(paste(c(auc_nb.h@y.name, auc_nb.h@y.values)), quote=FALSE)
```

```
## [1] Area under the ROC curve 0.694184643497304
```

Naïve Bayes repeated k-folds cross-validation candidate

```
auc_nb.r5kf8 = performance(pred_nb.r5kf8, "auc")
print(paste(c(auc_nb.r5kf8@y.name, auc_nb.r5kf8@y.values)), quote=FALSE)
```

```
## [1] Area under the ROC curve 0.694908696817499
```

KNN holdout candidate

```
auc_knn.h = performance(pred_knn.h, "auc")
print(paste(c(auc_knn.h@y.name, auc_knn.h@y.values)), quote=FALSE)
```

```
## [1] Area under the ROC curve 0.672607793022195
```

KNN repeated k-folds cross-validation candidate

```
auc_knn.r5kf8 = performance(pred_knn.r5kf8, "auc")
print(paste(c(auc_knn.r5kf8@y.name, auc_knn.r5kf8@y.values)), quote=FALSE)
```

```
## [1] Area under the ROC curve 0.732314474153576
```

Conclusion

If the over-fit Random Forest wasn't considered over-fit, then it would be the recommended model; further experimentation would probably demonstrate how it has been too tightly shaped to the test dataset to deliver similar results for fresh Abalone data.

If one metric were to be used to generally determine which were the better performing models then it would be the RAUC, in which case the cross-validated KNN model comes out on top *(ROAC: 0.722)* while the Random Forest with holdout training comes in second *(ROAC: 0.701)*; both of these values outperform the best Naïve Bayes model *(ROAC: 0.695)* though not by much, so further investigation into tuning would probably be a good idea. The good thing about these models, Random Forest and KNN is that in their own way they are fast, so while the classification performance could be improved, these two type of models would be recommended over the C4.5 decision tree.

Whichever model is used, repeated cross-validation has benefit, and the leave-one-out method doesn't seem worth the effort.

Further work

With more time, extra analysis would be carried out to determine the accuracy of classification for each particular age group. At the moment the *old* age group is generally, very poorly classified, so attempts to improve that particular classification would be beneficial to overall model performance.

Experimentation with Random Forests against incomplete data would be interesting with regard to exploring the usefulness of this type of model over other types. More study of Random Forests and overfitting is required to get a better understanding of whether or not the best performing model as actually overfit or not;

a quick update, to look at out-of-bag prediction suggests that the results are good; this will have to be continued…

References

Anh, V. (2015). In:.Available from https://datayo.wordpress.com/2015/05/13/using-c4-5/ (https://datayo.wordpress.com/2015/05/13/using-c4-5/).

Benyamin, D. (2012). *A Gentle Introduction to Random Forests, Ensembles, and Performance Metrics in a Commercial System* [].Available from http://blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics (http://blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics).

Brownlee, J. (2014a). In:.Available from https://machinelearningmastery.com/how-to-estimate-model-accuracy-in-r-using-the-caret-package/ (https://machinelearningmastery.com/how-to-estimate-model-accuracy-in-r-using-the-caret-package/).

Brownlee, J. (2014b). In:.Available from https://machinelearningmastery.com/how-to-get-baseline-results-and-why-they-matter/ (https://machinelearningmastery.com/how-to-get-baseline-results-and-why-they-matter/).

Brownlee, J. (2014c). In:.Available from https://machinelearningmastery.com/an-introduction-to-feature-selection/ (https://machinelearningmastery.com/an-introduction-to-feature-selection/).

Brownlee, J. (2014d). In:.Available from https://machinelearningmastery.com/feature-selection-with-the-caret-r-package/ (https://machinelearningmastery.com/feature-selection-with-the-caret-r-package/).

Brownlee, J. (2014e). In:.Available from https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/ (https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/).

Brownlee, J. (2014f). In:.Available from https://machinelearningmastery.com/tuning-machine-learning-models-using-the-caret-r-package/ (https://machinelearningmastery.com/tuning-machine-learning-models-using-the-caret-r-package/).

Brownlee, J. (2016a). *Machine Learning Datasets in R (10 datasets you can use right now)* [].Available from https://machinelearningmastery.com/machine-learning-datasets-in-r/ (https://machinelearningmastery.com/machine-learning-datasets-in-r/).

Brownlee, J. (2016b). In:.Available from https://machinelearningmastery.com/confusion-matrix-machine-learning/ (https://machinelearningmastery.com/confusion-matrix-machine-learning/).

Brownlee, J. (2016c). In:.Available from https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/ (https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/).

Brownlee, J. (2016d). In:.Available from https://machinelearningmastery.com/use-classification-machine-learning-algorithms-weka/ (https://machinelearningmastery.com/use-classification-machine-learning-algorithms-weka/).

Cross-validation (statistics) (2018).Available from https://en.wikipedia.org/w/index.php?title=Cross-validation_(statistics)&oldid=818145919 (https://en.wikipedia.org/w/index.php?title=Cross-validation_(statistics)&oldid=818145919).

Datasets for Data Science and Machine Learning (2017).Available from https://elitedatascience.com/datasets (https://elitedatascience.com/datasets).

Devlin, J. (2016). *The first step in a machine learning project* [].Available from http://www.dataquest.io/blog/machine-learning-preparing-data/ (http://www.dataquest.io/blog/machine-learning-preparing-data/).

Esuli, A. and Sebastiani, F. (2009). Training Data Cleaning for Text Classification. In: *Advances in Information Retrieval Theory*. Springer, Berlin, Heidelberg, 29–41.Available from https://link.springer.com/chapter/10.1007/978-3-642-04417-5_4 (https://link.springer.com/chapter/10.1007/978-3-642-04417-5_4).

Frost, J. (no date). *How to Interpret a Regression Model with Low R-squared and Low P values* [].Available from http://blog.minitab.com/blog/adventures-in-statistics-2/how-to-interpret-a-regression-model-with-low-r-squared-and-low-p-values (http://blog.minitab.com/blog/adventures-in-statistics-2/how-to-interpret-a-regression-model-with-low-r-squared-and-low-p-values).

Frost, J. (no date). *Regression Analysis: How Do I Interpret R-squared and Assess the Goodness-of-Fit?* [].Available from http://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit (http://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit).

Gu, S. and Wu, Q. (2017). In:.Available from https://medium.com/@Synced/how-random-forest-algorithm-works-in-machine-learning-3c0fe15b6674 (https://medium.com/@Synced/how-random-forest-algorithm-works-in-machine-learning-3c0fe15b6674).

Gupta, P. (2017). *Cross-Validation in Machine Learning* [].Available from https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f (https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f).

Hamel, G. (2015). *Introduction to R Part 29: Decision Trees* [].Available from http://hamelg.blogspot.com/2015/09/introduction-to-r-part-29-decision-trees.html (http://hamelg.blogspot.com/2015/09/introduction-to-r-part-29-decision-trees.html).

Hiemstra, P. (2010). *[R] Extract R-squared from summary of lm* [].Available from https://stat.ethz.ch/pipermail/r-help/2010-January/225345.html (https://stat.ethz.ch/pipermail/r-help/2010-January/225345.html).

Hussain, S.F. (no date). *Datasets - Syed Fawad Hussain* [].Available from https://sites.google.com/site/fawadsyed/datasets (https://sites.google.com/site/fawadsyed/datasets).

J48 decision tree - Mining at UOC (no date).Available from http://data-mining.business-intelligence.uoc.edu/home/j48-decision-tree (http://data-mining.business-intelligence.uoc.edu/home/j48-decision-tree).

Jolly, K. (2017). In:.Available from http://lineardata.net/the-ultimate-guide-to-cleaning-data-in-r/ (http://lineardata.net/the-ultimate-guide-to-cleaning-data-in-r/).

Joshi, R. (2016). *Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures* [].Available from http://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/ (http://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/).

Kuhn, M. (2014). *Comparing the Bootstrap and Cross-Validation* [].Available from http://appliedpredictivemodeling.com/blog/2014/11/27/08ks7leh0zof45zpf5vqe56d1sahb0 (http://appliedpredictivemodeling.com/blog/2014/11/27/08ks7leh0zof45zpf5vqe56d1sahb0).

Linear Regression in R: Abalone Dataset | Statistical Consulting Group (no date).Available from http://scg.sdsu.edu/linear-regression-in-r-abalone-dataset/ (http://scg.sdsu.edu/linear-regression-in-r-abalone-dataset/).

Mourya, S. (no date). *Accuracy Vs Precision – NoSimpler* [].Available from http://www.nosimpler.me/accuracy-precision/ (http://www.nosimpler.me/accuracy-precision/).

Polamuri, S. (2017). In:.Available from http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/ (http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/).

Predictive Analytics made easy | ClearPredictions.com (no date).Available from https://clearpredictions.com/Home/DecisionTree (https://clearpredictions.com/Home/DecisionTree).

Raut, A. (2017). In:.Available from https://medium.com/data-science-group-iitr/k-nearest-neighbors-knn-500f0d17c8f1 (https://medium.com/data-science-group-iitr/k-nearest-neighbors-knn-500f0d17c8f1).

ROC curves and Area Under the Curve explained (video) (2014).Available from http://www.dataschool.io/roc-curves-and-auc-explained/ (http://www.dataschool.io/roc-curves-and-auc-explained/).

Saito, T. and Rehmsmeier, M. (2015). In:.Available from https://classeval.wordpress.com/introduction/introduction-to-the-precision-recall-plot/ (https://classeval.wordpress.com/introduction/introduction-to-the-precision-recall-plot/).

Schoonjans, F. (no date). *ROC curve analysis with MedCalc* [].Available from https://www.medcalc.org/manual/roc-curves.php (https://www.medcalc.org/manual/roc-curves.php).

Srivastava, T. (2014a). In:.Available from https://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/ (https://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/).

Srivastava, T. (2014b). In:.Available from https://www.analyticsvidhya.com/blog/2014/06/introduction-random-forest-simplified/ (https://www.analyticsvidhya.com/blog/2014/06/introduction-random-forest-simplified/).

Steinberg, D. (2013). *Why Leave-One-Out (LOO) Cross-Validation Does Not Work For Trees* []. Salford Systems.Available from https://info.salford-systems.com/blog/bid/312328/Why-Leave-One-Out-LOO-Cross-Validation-Does-Not-Work-For-Trees (https://info.salford-systems.com/blog/bid/312328/Why-Leave-One-Out-LOO-Cross-Validation-Does-Not-Work-For-Trees).

Strong, E. (2016). *Predicting Abalone Rings, Part 1- Multiple Regression – Eric Strong* [].Available from http://ericstrong.org/predicting-abalone-rings-part-1/ (http://ericstrong.org/predicting-abalone-rings-part-1/).

Thirumuruganathan, S. (2010). In:.Available from https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/ (https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/).

Venturini, S. (2016). In:.Available from http://www.milanor.net/blog/cross-validation-for-predictive-analytics-using-r/ (http://www.milanor.net/blog/cross-validation-for-predictive-analytics-using-r/).

(no date). In:.Available from http://bigdatalearn.tumblr.com/post/55854866580/about-j48-decision-tree-pruning-in-weka (http://bigdatalearn.tumblr.com/post/55854866580/about-j48-decision-tree-pruning-in-weka).