BREAST CANCER

Sathish kumar Subbarayelu

Introduction

Random forests, also known as random decision forests, are a popular ensemble method that can be used to build predictive models for both classification and regression problems. Ensemble methods use multiple learning models to gain better predictive results - in the case of a random forest, the model creates an entire forest of random uncorrelated decision trees to arrive at the best possible answer. To demonstrate how this works in practice - specifically in a classification context - I'll be walking you through an example using a famous data set from the University of California, Irvine (UCI) Machine Learning Repository. The data set, called the Breast Cancer Wisconsin (Diagnostic) Data Set, deals with binary classification and includes features computed from digitized images of biopsies. The data set can be downloaded here. To follow this tutorial, you will need some familiarity with classification and regression tree (CART) modeling. I will provide a brief overview of different CART methodologies that are relevant to random forest, beginning with decision trees. If you'd like to brush up on your knowledge of CART modeling before beginning the tutorial, I highly recommend reading Chapter 8 of the book "An Introduction to Statistical Learning with Applications in R," which can be downloaded here (http://faculty.marshall.usc.edu/gareth-james/)

Decision Trees

Decision trees are simple but intuitive models that utilize a top-down approach in which the root node creates binary splits until a certain criteria is met. This binary splitting of nodes provides a predicted value based on the interior nodes leading to the terminal (final) nodes. In a classification context, a decision tree will output a predicted target class for each terminal node produced. Although intuitive, decision trees have limitations that prevent them from being useful in machine learning applications. You can learn more about implementing a decision tree here (https://scikit-learn.org/stable/modules/tree.html).

Limitations to Decision Trees

Decision trees tend to have high variance when they utilize different training and test sets of the same data, since they tend to overfit on training data. This leads to poor performance on unseen data. Unfortunately, this limits the usage of decision trees in predictive modeling. However, using ensemble methods, we can create models that utilize underlying decision trees as a foundation for producing powerful results.

Bootstrap Aggregating Trees

Through a process known as bootstrap aggregating (or bagging), it's possible to create an ensemble (forest) of trees where multiple training sets are generated with replacement, meaning data instances - or in the case of this tutorial, patients - can be repeated. Once the training sets are created, a CART model can be trained on each subsample. This approach helps reduce variance by averaging the ensemble's results, creating a majority-votes model. Another important feature of bagging trees is that the resulting model uses the entire feature space when considering node splits. Bagging trees allow the trees to grow

without pruning, reducing the tree-depth sizes and resulting in high variance but lower bias, which can help improve predictive power. However, a downside to this process is that the utilization of the entire feature space creates a risk of correlation between trees, increasing bias in the model.

Limitations to Bagging Trees

The main limitation of bagging trees is that it uses the entire feature space when creating splits in the trees. If some variables within the feature space are indicative of certain predictions, you run the risk of having a forest of correlated trees, thereby increasing bias and reducing variance. However, a simple tweak of the bagging trees methodology can prove advantageous to the model's predictive power.

Random Forest

Random forest aims to reduce the previously mentioned correlation issue by choosing only a subsample of the feature space at each split. Essentially, it aims to make the trees de-correlated and prune the trees by setting a stopping criteria for node splits, which I will cover in more detail later.

Load Packages

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages ---------------------------------------------------------------
tidyverse 1.3.0 --
```

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------------------ tidyv
erse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
```

```
## Warning: package 'caret' was built under R version 3.6.2
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
if(!require(ggcorrplot)) install.packages("ggcorrplot", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: ggcorrplot
```

```
## Warning: package 'ggcorrplot' was built under R version 3.6.2
```

```
if(!require(GGally)) install.packages("GGally", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: GGally
```

```
## Warning: package 'GGally' was built under R version 3.6.2
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
##
## Attaching package: 'GGally'
```

```
## The following object is masked from 'package:dplyr':
##
##     nasa
```

```
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.or
g")
```

```
## Loading required package: randomForest
```

```
## Warning: package 'randomForest' was built under R version 3.6.2
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
if(!require(e1071)) install.packages("e1071", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: e1071
```

```
## Warning: package 'e1071' was built under R version 3.6.2
```

```
if(!require(ROCR)) install.packages("ROCR", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: ROCR
```

```
## Warning: package 'ROCR' was built under R version 3.6.2
```

```
## Loading required package: gplots
```

```
## Warning: package 'gplots' was built under R version 3.6.2
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
if(!require(pROC)) install.packages("pROC", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: pROC
```

```
## Warning: package 'pROC' was built under R version 3.6.2
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

Load Data

For this section, let's load the data .

```
fileURL <- "https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wd
bc.data"

breast_cancer <- read.csv(fileURL, header = FALSE, sep = ",", quote = "\"'")
```

Downloaded data has a 32 attributes with 569 rows, but in the link below
https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29
(https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29)
Attributes details are given as
Attribute Information:
1) ID number
2) Diagnosis (M = malignant, B = benign)
3-32)

Ten real-valued features are computed for each cell nucleus:

** a) radius (mean of distances from center to points on the perimeter)
b) texture (standard deviation of gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2 / area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave points (number of concave portions of the contour)
i) symmetry
j) fractal dimension ("coastline approximation" - 1)**

I have confusion that only 12 attribute names are given let me download the wdbc names file and check the attributes names

```
attribute <- "https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/
wdbc.names"
mynames <- readLines(attribute)
```

```
## Warning in readLines(attribute): incomplete final line found on 'https://
## archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/
## wdbc.names'
```

```
#mynames # gives the attributes information with extra data so i delete some unwanted lines in t
his.
mynames <- mynames[109:140]
mynames
```

```
##  [1] "7. Attribute information"
##  [2] ""
##  [3] "1) ID number"
##  [4] "2) Diagnosis (M = malignant, B = benign)"
##  [5] "3-32)"
##  [6] ""
##  [7] "Ten real-valued features are computed for each cell nucleus:"
##  [8] ""
##  [9] "\ta) radius (mean of distances from center to points on the perimeter)"
## [10] "\tb) texture (standard deviation of gray-scale values)"
## [11] "\tc) perimeter"
## [12] "\td) area"
## [13] "\te) smoothness (local variation in radius lengths)"
## [14] "\tf) compactness (perimeter^2 / area - 1.0)"
## [15] "\tg) concavity (severity of concave portions of the contour)"
## [16] "\th) concave points (number of concave portions of the contour)"
## [17] "\ti) symmetry "
## [18] "\tj) fractal dimension (\"coastline approximation\" - 1)"
## [19] ""
## [20] "Several of the papers listed above contain detailed descriptions of"
## [21] "how these features are computed. "
## [22] ""
## [23] "The mean, standard error, and \"worst\" or largest (mean of the three"
## [24] "largest values) of these features were computed for each image,"
## [25] "resulting in 30 features.  For instance, field 3 is Mean Radius, field"
## [26] "13 is Radius SE, field 23 is Worst Radius."
## [27] ""
## [28] "All feature values are recoded with four significant digits."
## [29] ""
## [30] "8. Missing attribute values: none"
## [31] ""
## [32] "9. Class distribution: 357 benign, 212 malignant"
```

From this output data,

[23] “The mean, standard error, and "worst" or largest (mean of the three”
[24] “largest values) of these features were computed for each image,”
[25] “resulting in 30 features. For instance, field 3 is Mean Radius, field”
[26] “13 is Radius SE, field 23 is Worst Radius.”
So we get the attributes names correctly as below from above output

```
names(breast_cancer) <- c('id_number', 'diagnosis', 'radius_mean',
           'texture_mean', 'perimeter_mean', 'area_mean',
           'smoothness_mean', 'compactness_mean',
           'concavity_mean','concave_points_mean',
           'symmetry_mean', 'fractal_dimension_mean',
           'radius_se', 'texture_se', 'perimeter_se',
           'area_se', 'smoothness_se', 'compactness_se',
           'concavity_se', 'concave_points_se',
           'symmetry_se', 'fractal_dimension_se',
           'radius_worst', 'texture_worst',
           'perimeter_worst', 'area_worst',
           'smoothness_worst', 'compactness_worst',
           'concavity_worst', 'concave_points_worst',
           'symmetry_worst', 'fractal_dimension_worst')


breast_cancer$id_number <- NULL
```

Let’s preview the data set utilizing the head() function which will give the first 6 values of our data frame.

```
head(breast_cancer)
```

```
##   diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean
## 1         M       17.99        10.38         122.80    1001.0         0.11840
## 2         M       20.57        17.77         132.90    1326.0         0.08474
## 3         M       19.69        21.25         130.00    1203.0         0.10960
## 4         M       11.42        20.38          77.58     386.1         0.14250
## 5         M       20.29        14.34         135.10    1297.0         0.10030
## 6         M       12.45        15.70          82.57     477.1         0.12780
##   compactness_mean concavity_mean concave_points_mean symmetry_mean
## 1          0.27760         0.3001              0.14710        0.2419
## 2          0.07864         0.0869              0.07017        0.1812
## 3          0.15990         0.1974              0.12790        0.2069
## 4          0.28390         0.2414              0.10520        0.2597
## 5          0.13280         0.1980              0.10430        0.1809
## 6          0.17000         0.1578              0.08089        0.2087
##   fractal_dimension_mean radius_se texture_se perimeter_se area_se
## 1                0.07871    1.0950     0.9053        8.589  153.40
## 2                0.05667    0.5435     0.7339        3.398   74.08
## 3                0.05999    0.7456     0.7869        4.585   94.03
## 4                0.09744    0.4956     1.1560        3.445   27.23
## 5                0.05883    0.7572     0.7813        5.438   94.44
## 6                0.07613    0.3345     0.8902        2.217   27.19
##   smoothness_se compactness_se concavity_se concave_points_se symmetry_se
## 1      0.006399        0.04904      0.05373           0.01587     0.03003
## 2      0.005225        0.01308      0.01860           0.01340     0.01389
## 3      0.006150        0.04006      0.03832           0.02058     0.02250
## 4      0.009110        0.07458      0.05661           0.01867     0.05963
## 5      0.011490        0.02461      0.05688           0.01885     0.01756
## 6      0.007510        0.03345      0.03672           0.01137     0.02165
##   fractal_dimension_se radius_worst texture_worst perimeter_worst area_worst
## 1             0.006193        25.38         17.33          184.60     2019.0
## 2             0.003532        24.99         23.41          158.80     1956.0
## 3             0.004571        23.57         25.53          152.50     1709.0
## 4             0.009208        14.91         26.50           98.87      567.7
## 5             0.005115        22.54         16.67          152.20     1575.0
## 6             0.005082        15.47         23.75          103.40      741.6
##   smoothness_worst compactness_worst concavity_worst concave_points_worst
## 1           0.1622            0.6656          0.7119               0.2654
## 2           0.1238            0.1866          0.2416               0.1860
## 3           0.1444            0.4245          0.4504               0.2430
## 4           0.2098            0.8663          0.6869               0.2575
## 5           0.1374            0.2050          0.4000               0.1625
## 6           0.1791            0.5249          0.5355               0.1741
##   symmetry_worst fractal_dimension_worst
## 1         0.4601                 0.11890
## 2         0.2750                 0.08902
## 3         0.3613                 0.08758
## 4         0.6638                 0.17300
## 5         0.2364                 0.07678
## 6         0.3985                 0.12440
```

Next, we'll give the dimensions of the data set; where the first value is the number of patients and the second value is the number of features. We print the data types of our data set this is important because this will often be an indicator of missing data, as well as giving us context to anymore data cleanage.

```
dim(breast_cancer)
```

```
## [1] 569  31
```

```
str(breast_cancer)
```

```
## 'data.frame':    569 obs. of  31 variables:
##  $ diagnosis              : Factor w/ 2 levels "B","M": 2 2 2 2 2 2 2 2 2 2 ...
##  $ radius_mean            : num  18 20.6 19.7 11.4 20.3 ...
##  $ texture_mean           : num  10.4 17.8 21.2 20.4 14.3 ...
##  $ perimeter_mean         : num  122.8 132.9 130 77.6 135.1 ...
##  $ area_mean              : num  1001 1326 1203 386 1297 ...
##  $ smoothness_mean        : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
##  $ compactness_mean       : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
##  $ concavity_mean         : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
##  $ concave_points_mean    : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
##  $ symmetry_mean          : num  0.242 0.181 0.207 0.26 0.181 ...
##  $ fractal_dimension_mean : num  0.0787 0.0567 0.06 0.0974 0.0588 ...
##  $ radius_se              : num  1.095 0.543 0.746 0.496 0.757 ...
##  $ texture_se             : num  0.905 0.734 0.787 1.156 0.781 ...
##  $ perimeter_se           : num  8.59 3.4 4.58 3.44 5.44 ...
##  $ area_se                : num  153.4 74.1 94 27.2 94.4 ...
##  $ smoothness_se          : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
##  $ compactness_se         : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
##  $ concavity_se           : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
##  $ concave_points_se      : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
##  $ symmetry_se            : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
##  $ fractal_dimension_se   : num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
##  $ radius_worst           : num  25.4 25 23.6 14.9 22.5 ...
##  $ texture_worst          : num  17.3 23.4 25.5 26.5 16.7 ...
##  $ perimeter_worst        : num  184.6 158.8 152.5 98.9 152.2 ...
##  $ area_worst             : num  2019 1956 1709 568 1575 ...
##  $ smoothness_worst       : num  0.162 0.124 0.144 0.21 0.137 ...
##  $ compactness_worst      : num  0.666 0.187 0.424 0.866 0.205 ...
##  $ concavity_worst        : num  0.712 0.242 0.45 0.687 0.4 ...
##  $ concave_points_worst   : num  0.265 0.186 0.243 0.258 0.163 ...
##  $ symmetry_worst         : num  0.46 0.275 0.361 0.664 0.236 ...
##  $ fractal_dimension_worst: num  0.1189 0.089 0.0876 0.173 0.0768 ...
```

Class Imbalance

The distribution for diagnosis is important because it brings up the discussion of Class Imbalance within Machine learning and data mining applications. Class Imbalance refers to when a target class within a data set is outnumbered by the other target class (or classes). This can lead to misleading accuracy metrics, known as accuracy paradox (https://en.wikipedia.org/wiki/Accuracy_paradox), therefore we have to make sure our target classes aren't imblanaced. We do so by creating a function that will output the distribution of the target classes.

NOTE: If your data set suffers from class imbalance I suggest reading documentation on upsampling and downsampling.

```
breast_cancer %>%
  count(diagnosis) %>%
  group_by(diagnosis) %>%
  summarize(perc_dx = round((n / 569)* 100, 2))
```

```
## # A tibble: 2 x 2
##   diagnosis perc_dx
##   <fct>       <dbl>
## 1 B            62.7
## 2 M            37.3
```

Fortunately, this data set does not suffer from class imbalance. Next we will use a useful function that gives us standard descriptive statistics for each feature including mean, standard deviation, minimum value, maximum value, and range intervals.

```
summary(breast_cancer)
```

```
##  diagnosis  radius_mean      texture_mean   perimeter_mean     area_mean
##  B:357     Min.   : 6.981   Min.   : 9.71   Min.   : 43.79   Min.   : 143.5
##  M:212     1st Qu.:11.700   1st Qu.:16.17   1st Qu.: 75.17   1st Qu.: 420.3
##            Median :13.370   Median :18.84   Median : 86.24   Median : 551.1
##            Mean   :14.127   Mean   :19.29   Mean   : 91.97   Mean   : 654.9
##            3rd Qu.:15.780   3rd Qu.:21.80   3rd Qu.:104.10   3rd Qu.: 782.7
##            Max.   :28.110   Max.   :39.28   Max.   :188.50   Max.   :2501.0
##  smoothness_mean   compactness_mean  concavity_mean    concave_points_mean
##  Min.   :0.05263   Min.   :0.01938   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.08637   1st Qu.:0.06492   1st Qu.:0.02956   1st Qu.:0.02031
##  Median :0.09587   Median :0.09263   Median :0.06154   Median :0.03350
##  Mean   :0.09636   Mean   :0.10434   Mean   :0.08880   Mean   :0.04892
##  3rd Qu.:0.10530   3rd Qu.:0.13040   3rd Qu.:0.13070   3rd Qu.:0.07400
##  Max.   :0.16340   Max.   :0.34540   Max.   :0.42680   Max.   :0.20120
##  symmetry_mean    fractal_dimension_mean   radius_se        texture_se
##  Min.   :0.1060   Min.   :0.04996        Min.   :0.1115   Min.   :0.3602
##  1st Qu.:0.1619   1st Qu.:0.05770        1st Qu.:0.2324   1st Qu.:0.8339
##  Median :0.1792   Median :0.06154        Median :0.3242   Median :1.1080
##  Mean   :0.1812   Mean   :0.06280        Mean   :0.4052   Mean   :1.2169
##  3rd Qu.:0.1957   3rd Qu.:0.06612        3rd Qu.:0.4789   3rd Qu.:1.4740
##  Max.   :0.3040   Max.   :0.09744        Max.   :2.8730   Max.   :4.8850
##   perimeter_se       area_se        smoothness_se      compactness_se
##  Min.   : 0.757   Min.   :  6.802   Min.   :0.001713   Min.   :0.002252
##  1st Qu.: 1.606   1st Qu.: 17.850   1st Qu.:0.005169   1st Qu.:0.013080
##  Median : 2.287   Median : 24.530   Median :0.006380   Median :0.020450
##  Mean   : 2.866   Mean   : 40.337   Mean   :0.007041   Mean   :0.025478
##  3rd Qu.: 3.357   3rd Qu.: 45.190   3rd Qu.:0.008146   3rd Qu.:0.032450
##  Max.   :21.980   Max.   :542.200   Max.   :0.031130   Max.   :0.135400
##   concavity_se     concave_points_se   symmetry_se       fractal_dimension_se
##  Min.   :0.00000   Min.   :0.000000   Min.   :0.007882   Min.   :0.0008948
##  1st Qu.:0.01509   1st Qu.:0.007638   1st Qu.:0.015160   1st Qu.:0.0022480
##  Median :0.02589   Median :0.010930   Median :0.018730   Median :0.0031870
##  Mean   :0.03189   Mean   :0.011796   Mean   :0.020542   Mean   :0.0037949
##  3rd Qu.:0.04205   3rd Qu.:0.014710   3rd Qu.:0.023480   3rd Qu.:0.0045580
##  Max.   :0.39600   Max.   :0.052790   Max.   :0.078950   Max.   :0.0298400
##   radius_worst   texture_worst   perimeter_worst    area_worst
##  Min.   : 7.93   Min.   :12.02   Min.   : 50.41   Min.   : 185.2
##  1st Qu.:13.01   1st Qu.:21.08   1st Qu.: 84.11   1st Qu.: 515.3
##  Median :14.97   Median :25.41   Median : 97.66   Median : 686.5
##  Mean   :16.27   Mean   :25.68   Mean   :107.26   Mean   : 880.6
##  3rd Qu.:18.79   3rd Qu.:29.72   3rd Qu.:125.40   3rd Qu.:1084.0
##  Max.   :36.04   Max.   :49.54   Max.   :251.20   Max.   :4254.0
##  smoothness_worst  compactness_worst concavity_worst  concave_points_worst
##  Min.   :0.07117   Min.   :0.02729   Min.   :0.0000   Min.   :0.00000
##  1st Qu.:0.11660   1st Qu.:0.14720   1st Qu.:0.1145   1st Qu.:0.06493
##  Median :0.13130   Median :0.21190   Median :0.2267   Median :0.09993
##  Mean   :0.13237   Mean   :0.25427   Mean   :0.2722   Mean   :0.11461
##  3rd Qu.:0.14600   3rd Qu.:0.33910   3rd Qu.:0.3829   3rd Qu.:0.16140
##  Max.   :0.22260   Max.   :1.05800   Max.   :1.2520   Max.   :0.29100
##  symmetry_worst   fractal_dimension_worst
##  Min.   :0.1565   Min.   :0.05504
##  1st Qu.:0.2504   1st Qu.:0.07146
##  Median :0.2822   Median :0.08004
```

```
##  Mean   :0.2901   Mean   :0.08395
##  3rd Qu.:0.3179   3rd Qu.:0.09208
##  Max.   :0.6638   Max.   :0.20750
```

We can see through the maximum row that our data varies in distribution, this will be important when considering classification models. Standardization is an important requirement for many classification models that should be considered when implementing pre-processing. Some models (like neural networks) can perform poorly if pre-processing isn't considered, so the describe() function can be a good indicator for standardization. Fortunately Random Forest does not require any pre-processing (for use of categorical data see sklearn's Encoding Categorical Data section (https://scikit-learn.org/stable/modules/preprocessing.html#encoding-categorical-features)).

Creating Training and Test Sets

We split the data set into our training and test sets which will be (pseudo) randomly selected having a 80-20% splt. We will use the training set to train our model along with some optimization, and use our test set as the unseen data that will be a useful final metric to let us know how well our model does. When using this method for machine learning always be weary of utilizing your test set when creating models. The issue of data leakage is a grave and serious issue that is common in practice and can result in over-fitting. More on data leakage can be found in this Kaggle article (https://www.kaggle.com/wiki/Leakage)

```
set.seed(42, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(42)` instead
```

```
## Warning in set.seed(42, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
trainIndex <- createDataPartition(breast_cancer$diagnosis,
                                  p = .8,
                                  list = FALSE,
                                  times = 1)
training_set <- breast_cancer[ trainIndex, ]
test_set <- breast_cancer[ -trainIndex, ]
```

NOTE: What I mean when I say pseudo-random is that we would want everyone who replicates this project to get the same results. So we use a random seed generator and set it equal to a number of our choosing, this will then make the results the same for anyone who uses this generator, awesome for reproducibility.

Fitting Random Forest

The R version is very different because the caret package hyperparameter optimization will be done in the same chapter as fitting model along with cross validation. If you want an in more depth look check the python version.

Hyperparameters Optimization

Here we'll create a custom model to allow us to do a grid search, I will see which parameters output the best model based on accuracy.

mtry: Features used in each split
ntree: Number of trees used in model
nodesize: Max number of node splits

Custom grid search From https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/ (https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/)

```
customRF <- list(type = "Classification", library = "randomForest", loop = NULL)
customRF$parameters <- data.frame(parameter = c("mtry", "ntree", "nodesize"),
                                  class = rep("numeric", 3),
                                  label = c("mtry", "ntree", "nodesize"))
customRF$grid <- function(x, y, len = NULL, search = "grid") {}
customRF$fit <- function(x, y, wts, param, lev, last, weights, classProbs, ...) {
  randomForest(x, y, mtry = param$mtry, ntree=param$ntree, nodesize=param$nodesize, ...)
}
customRF$predict <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
  predict(modelFit, newdata)
customRF$prob <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
  predict(modelFit, newdata, type = "prob")
customRF$sort <- function(x) x[order(x[,1]),]
customRF$levels <- function(x) x$classes
```

Now that we have the custom settings well use the train method which crossvalidates and does a grid search, giving us the best parameters.

```
fitControl <- trainControl(## 10-fold CV
  method = "repeatedcv",
  number = 3,
  ## repeated ten times
  repeats = 10)

grid <- expand.grid(.mtry=c(floor(sqrt(ncol(training_set))), (ncol(training_set) - 1), floor(log
(ncol(training_set)))),
                    .ntree = c(100, 300, 500, 1000),
                    .nodesize =c(1:4))
set.seed(42, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(42)` instead
```

```
## Warning in set.seed(42, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
fit_rf <- train(as.factor(diagnosis) ~ .,
                data = training_set,
                method = customRF,
                metric = "Accuracy",
                tuneGrid= grid,
                trControl = fitControl)
```

Let's print out the different models and best model given by model.

```
fit_rf$finalModel
```

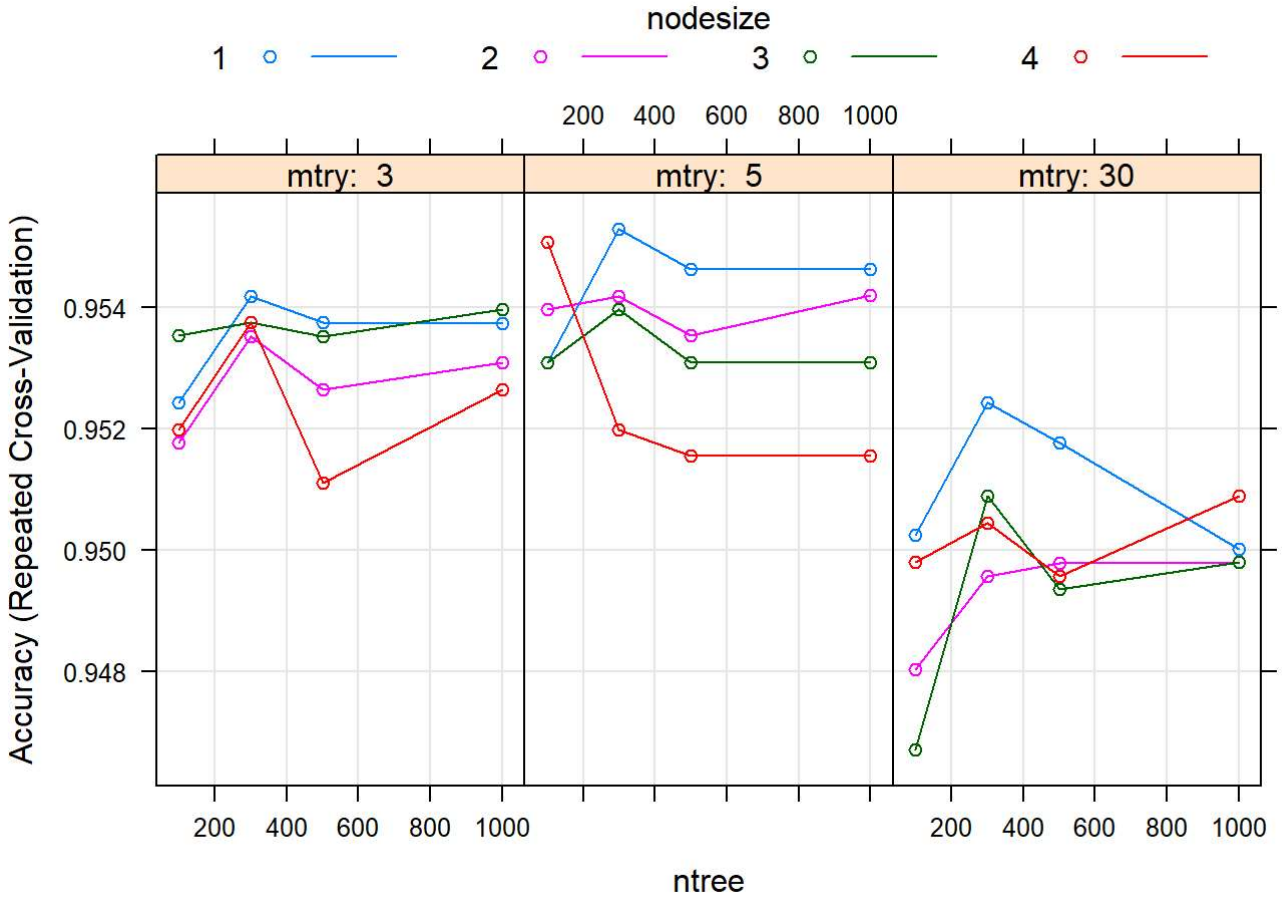
```
##
## Call:
##  randomForest(x = x, y = y, ntree = param$ntree, mtry = param$mtry,      nodesize = param$nod
esize)
##                Type of random forest: classification
##                      Number of trees: 300
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 4.61%
## Confusion matrix:
##     B   M class.error
## B 279   7  0.02447552
## M  14 156  0.08235294
```

```
fit_rf
```

```
## 456 samples
##  30 predictor
##   2 classes: 'B', 'M'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 10 times)
## Summary of sample sizes: 304, 304, 304, 304, 304, 304, ...
## Resampling results across tuning parameters:
##
##   mtry  ntree  nodesize  Accuracy   Kappa
##    3     100   1         0.9524203  0.8970735
##    3     100   2         0.9517710  0.8958407
##    3     100   3         0.9535225  0.8994780
##    3     100   4         0.9519816  0.8962304
##    3     300   1         0.9541790  0.9009817
##    3     300   2         0.9535196  0.8995097
##    3     300   3         0.9537419  0.9000437
##    3     300   4         0.9537404  0.8999347
##    3     500   1         0.9537404  0.9000032
##    3     500   2         0.9526424  0.8975990
##    3     500   3         0.9535211  0.8995336
##    3     500   4         0.9511088  0.8943280
##    3    1000   1         0.9537317  0.9000145
##    3    1000   2         0.9530825  0.8985643
##    3    1000   3         0.9539626  0.9004467
##    3    1000   4         0.9526352  0.8975458
##    5     100   1         0.9530912  0.8986861
##    5     100   2         0.9539582  0.9006339
##    5     100   3         0.9530781  0.8985848
##    5     100   4         0.9550605  0.9028285
##    5     300   1         0.9552741  0.9034418
##    5     300   2         0.9541718  0.9011492
##    5     300   3         0.9539655  0.9006053
##    5     300   4         0.9519846  0.8962870
##    5     500   1         0.9546119  0.9021030
##    5     500   2         0.9535225  0.8996643
##    5     500   3         0.9530825  0.8986747
##    5     500   4         0.9515561  0.8953758
##    5    1000   1         0.9546220  0.9020328
##    5    1000   2         0.9541819  0.9010498
##    5    1000   3         0.9530854  0.8986223
##    5    1000   4         0.9515489  0.8953814
##   30     100   1         0.9502431  0.8931445
##   30     100   2         0.9480443  0.8886176
##   30     100   3         0.9467242  0.8857903
##   30     100   4         0.9497987  0.8922443
##   30     300   1         0.9524275  0.8978794
##   30     300   2         0.9495766  0.8918272
##   30     300   3         0.9508953  0.8946752
##   30     300   4         0.9504524  0.8936931
##   30     500   1         0.9517725  0.8965182
##   30     500   2         0.9497959  0.8922731
##   30     500   3         0.9493573  0.8912974
```

```
##   30     500   4         0.9495766  0.8917992
##   30    1000   1         0.9500166  0.8928626
##   30    1000   2         0.9497988  0.8922197
##   30    1000   3         0.9497973  0.8922699
##   30    1000   4         0.9508953  0.8945905
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 5, ntree = 300 and nodesize = 1.
```

```
plot(fit_rf)
```



Variable Importance

Once we have trained the model, we are able to assess this concept of variable importance. A downside to creating ensemble methods with Decision Trees is we lose the interpretability that a single tree gives. A single tree can outline for us important node splits along with variables that were important at each split.

Forunately ensemble methods utilzing CART models use a metric to evaluate homogeneity of splits. Thus when creating ensembles these metrics can be utilized to give insight to important variables used in the training of the model. Two metrics that are used are gini impurity and entropy.

The two metrics vary and from reading documentation online, many people favor gini impurity due to the computational cost of entropy since it requires calculating the logarithmic function. For more discussion I recommend reading this article (https://github.com/rasbt/python-machine-learning-book/blob/master/faq/decision-tree-binary.md).

Here we define each metric:

$$\text{Gini Impurity} = 1 - \sum_i p_i$$

$$\text{Entropy} = \sum_i -p_i * \log_2 p_i$$

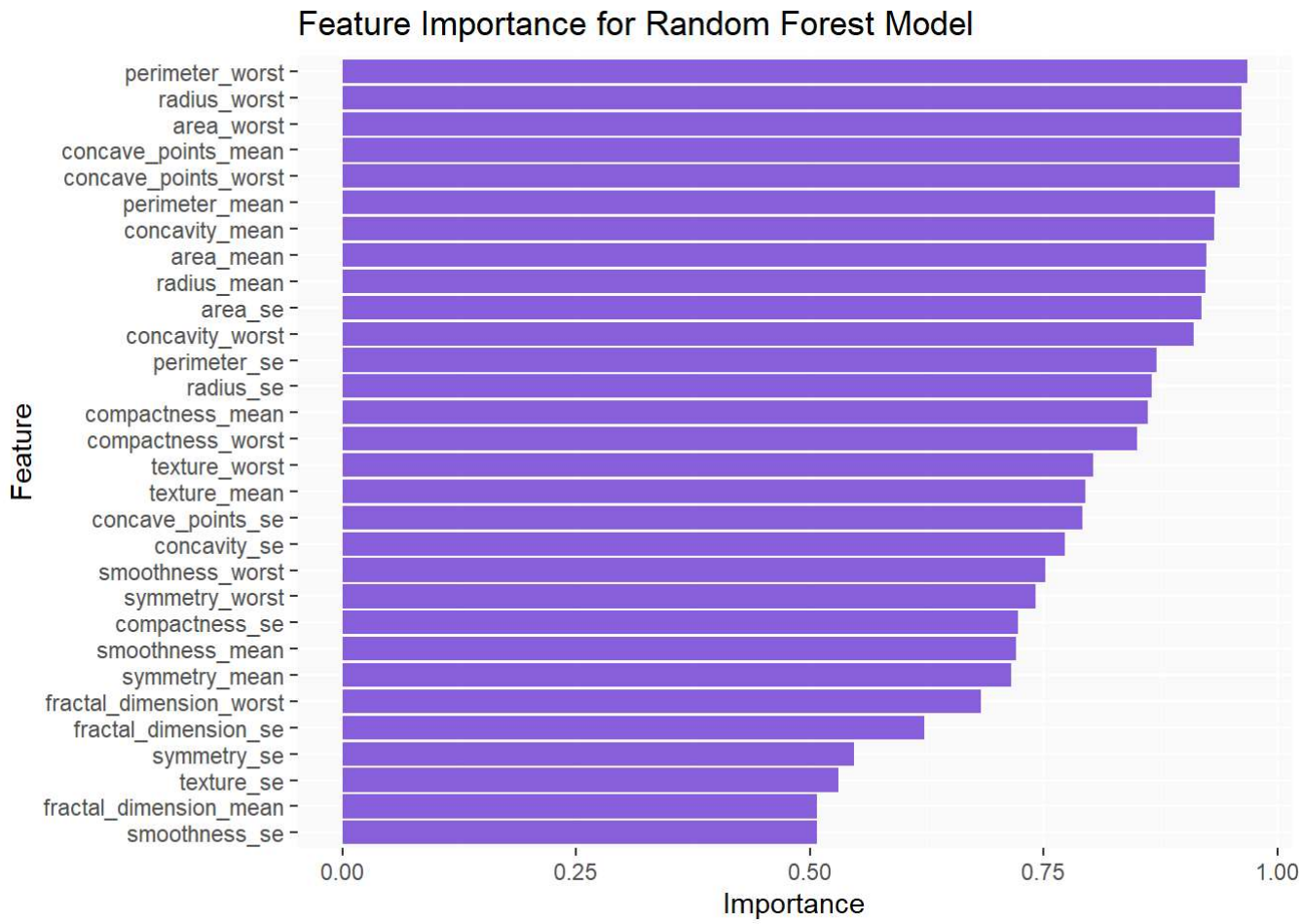
where pi is defined as the proportion of subsamples that belong to a certain target class. For the package randomForest, I believe the gini index is used without giving the choice to the information gain.

```
varImportance <- varImp(fit_rf, scale = FALSE)

varImportanceScores <- data.frame(varImportance$importance)

varImportanceScores <- data.frame(names = row.names(varImportanceScores), var_imp_scores = varIm
portanceScores$B)

varImportanceScores
```

```
##                       names var_imp_scores
## 1               radius_mean      0.9229638
## 2              texture_mean      0.7948272
## 3            perimeter_mean      0.9336487
## 4                 area_mean      0.9239305
## 5           smoothness_mean      0.7208865
## 6          compactness_mean      0.8613534
## 7            concavity_mean      0.9328877
## 8       concave_points_mean      0.9599548
## 9             symmetry_mean      0.7148807
## 10  fractal_dimension_mean      0.5073118
## 11                radius_se      0.8651584
## 12               texture_se      0.5300288
## 13             perimeter_se      0.8709070
## 14                  area_se      0.9187269
## 15            smoothness_se      0.5070444
## 16           compactness_se      0.7222851
## 17             concavity_se      0.7726347
## 18        concave_points_se      0.7910839
## 19              symmetry_se      0.5474496
## 20    fractal_dimension_se      0.6225422
## 21             radius_worst      0.9622378
## 22            texture_worst      0.8026018
## 23          perimeter_worst      0.9675339
## 24               area_worst      0.9615693
## 25         smoothness_worst      0.7520053
## 26        compactness_worst      0.8500103
## 27          concavity_worst      0.9108700
## 28     concave_points_worst      0.9595640
## 29           symmetry_worst      0.7410325
## 30 fractal_dimension_worst      0.6824558
```

Visual Representation

```
ggplot(varImportanceScores,
       aes(reorder(names, var_imp_scores), var_imp_scores)) +
  geom_bar(stat='identity',
           fill = '#875FDB') +
  theme(panel.background = element_rect(fill = '#fafafa')) +
  coord_flip() +
  labs(x = 'Feature', y = 'Importance') +
  ggtitle('Feature Importance for Random Forest Model')
```



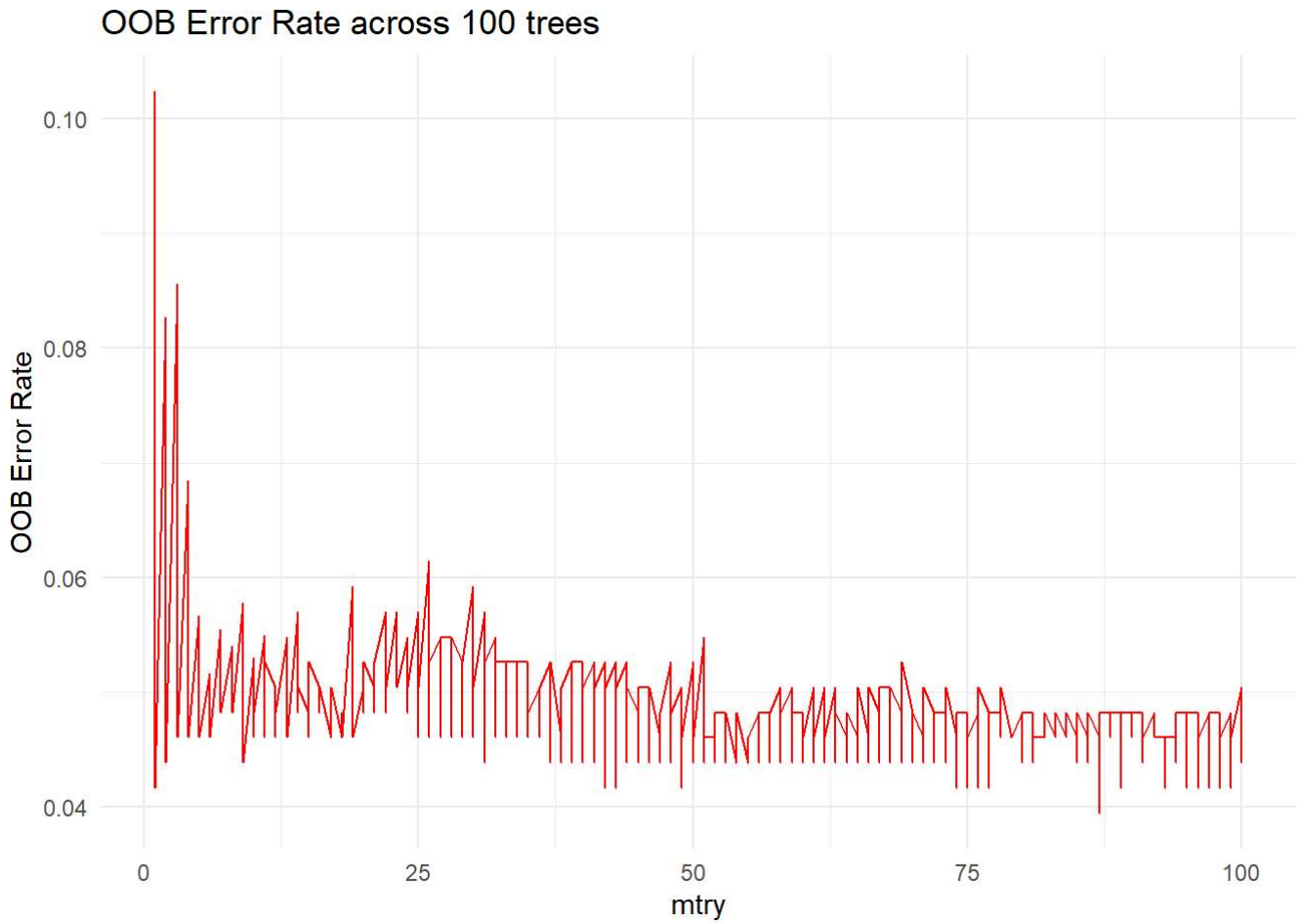
Out of Bag Error Rate

Another useful feature of Random Forest is the concept of Out of Bag Error Rate or OOB error rate. When creating the forest, typically only 2/3 of the data is used to train the trees, this gives us 1/3 of unseen data that we can then utilize.

```
oob_error <- data.frame(mtry = seq(1:100), oob = fit_rf$finalModel$err.rate[, 'OOB'])

paste0('Out of Bag Error Rate for model is: ', round(oob_error[100, 2], 4))
```

```
## [1] "Out of Bag Error Rate for model is: 0.0504"
```

```
ggplot(oob_error, aes(mtry, oob)) +
  geom_line(colour = 'red') +
  theme_minimal() +
  ggtitle('OOB Error Rate across 100 trees') +
  labs(y = 'OOB Error Rate')
```



Test Set Metrics

Now we will be utilizing the test set that was created earlier to receive another metric for evaluation of our model. Recall the importance of data leakage and that we didn't touch the test set until now, after we had done hyperparamter optimization.

```
predict_values <- predict(fit_rf, newdata = test_set)

ftable(predict_values, test_set$diagnosis)
```

```
##                  B  M
## predict_values
## B               70  1
## M                1 41
```

```
paste0('Test error rate is: ', round(((2/113)), 4))
```

```
## [1] "Test error rate is: 0.0177"
```

Conclusions

For this tutorial we went through a number of metrics to assess the capabilites of our Random Forest, but this can be taken further when using background information of the data set. Feature engineering would be a powerful tool to extract and move forward into research regarding the important features. As well defining key metrics to utilize when optimizing model paramters.

There have been advancements with image classification in the past decade that utilize the images intead of extracted features from images, but this data set is a great resource to become with machine learning processes. Especially for those who are just beginning to learn machine learning concepts. If you have any suggestions, recommendations, or corrections please reach out to me.