# Report on Dry Bean Dataset

Ngai Chun Tsui

3/29/2021

## Contents

# 1 Overview

Our project is about identification of dry beans. Images of 13,611 grains of 7 different registered dry beans were taken with a high-resolution camera. We have a total of 16 features: 12 dimensions and 4 shape forms.

Identification of dry beans is useful because it may allow us to classify them into the correct classes for commercial trading. It may be helpful for botanists to identify them for study. The technique may also be applied in identification of other types of plant or food.

## 1.1 Project Overview

The project mainly consists of 4 parts.

1. Overview: An overview of the project
2. Analysis: We study and explore the data. We visualize the data to gain some insights. Then we try fitting different models to predict the result.
3. Results: We compare the accuracy of different models and see which one is the best.
4. Conclusion: A brief conclusion, limitation and future work to improve on this project.

As for the most important part - model fitting, we separate the data set into train and test datasets. For each model, the train data set will be used for training model and the test dataset is used for testing accuracy. Since this problem is to identity dry bean as one of the seven classes, this is a classification problem. We use accuracy as our measurement (percentage of correct prediction in test set).

## 1.2 Problem Overview

The seven classes of dry beans are: **Barbunya, Bombay, Cali, Dermosan, Horoz, Seker, Sira**

The features information:

1. Area (A): The area of a bean zone and the number of pixels within its boundaries.
2. Perimeter (P): Bean circumference is defined as the length of its border.
3. Major axis length (L): The distance between the ends of the longest line that can be drawn from a bean.
4. Minor axis length (l): The longest line that can be drawn from the bean while standing perpendicular to the main axis.
5. Aspect ratio (K): Defines the relationship between L and l.
6. Eccentricity (Ec): Eccentricity of the ellipse having the same moments as the region.
7. Convex area (C): Number of pixels in the smallest convex polygon that can contain the area of a bean seed.

8. Equivalent diameter (Ed): The diameter of a circle having the same area as a bean seed area.
9. Extent (Ex): The ratio of the pixels in the bounding box to the bean area.
10. Solidity (S): Also known as convexity. The ratio of the pixels in the convex shell to those found in beans.
11. Roundness (R): Calculated with the following formula: (4piA)/(Pˆ2)
12. Compactness (CO): Measures the roundness of an object: Ed/L
13. ShapeFactor1 (SF1)
14. ShapeFactor2 (SF2)
15. ShapeFactor3 (SF3)
16. ShapeFactor4 (SF4)

### 1.3 Project Data (Step 1 in R file)

The data can be found in UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/machine-learning-databases/00602/DryBeanDataset.zip

The background information of the dataset: https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset

We download the zip file from the link and then unzip it. We then use read_excel() to read the excel file. We save the read data frame into a rda file. This can save us time as we don't have to donwload again every time we start testing.

```
# Download Dry Bean Dataset
# If the script is already run, we can get from the rda file.
if (file.exists("drybeans.rda") == TRUE) {
  # load the rda file
  load("drybeans.rda")

} else {
  # Else we download from the link
  dl <- tempfile()
  download.file("https://archive.ics.uci.edu/ml/machine-learning-databases/00602/DryBeanDataset.zip",

  excel_file <- unzip(dl, "DryBeanDataset/Dry_Bean_Dataset.xlsx")
  drybeans <- read_excel(excel_file)

  # Save file for faster processing
  save(drybeans, file = "drybeans.rda")

  rm(dl)
}
```

---

## 2 Analysis

### 2.1 Data Wrangling (Step 2 in R file)

We do some data wrangling so the data are easier for analysis. We divide the dataset into 90% train set and 10% test set. We choose 90/10 split ratio because it is commonly used in data analysis.

```r
# Make Class as a factor
drybeans <- drybeans %>%
  mutate(Class = as.factor(Class))

# Divide the set into train and test set
# 90% train set, 10% test set
set.seed(1) #Random sampling
test_index <- createDataPartition(y = drybeans$Class, times = 1, p = 0.1, list = FALSE)
test_set <- drybeans[test_index,]
train_set <- drybeans[-test_index,]
```

### 2.2 Data Exploration (Step 3 in R file)

The data has 13611 rows and 16 features. The features are explained in Problem Overview section. We will see the summary of the data frame and the first 4 rows of the data. We can see that the first 16 columns are the features and the final column 'Class' is our target. The features are all numeric values and this makes our analysis easier.

```r
# number of rows
nrow(drybeans)
```

```
## [1] 13611
```

```r
# number of columns (One column 'Class' is the outcome)
ncol(drybeans)
```

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

```r
# summary
summary(drybeans)
```

```
##       Area          Perimeter      MajorAxisLength MinorAxisLength
##  Min.   : 20420   Min.   : 524.7   Min.   :183.6   Min.   :122.5  
##  1st Qu.: 36328   1st Qu.: 703.5   1st Qu.:253.3   1st Qu.:175.8  
##  Median : 44652   Median : 794.9   Median :296.9   Median :192.4  
##  Mean   : 53048   Mean   : 855.3   Mean   :320.1   Mean   :202.3  
##  3rd Qu.: 61332   3rd Qu.: 977.2   3rd Qu.:376.5   3rd Qu.:217.0  
##  Max.   :254616   Max.   :1985.4   Max.   :738.9   Max.   :460.2  
##                                                                   
##   AspectRation    Eccentricity      ConvexArea     EquivDiameter  
##  Min.   :1.025   Min.   :0.2190   Min.   : 20684   Min.   :161.2  
##  1st Qu.:1.432   1st Qu.:0.7159   1st Qu.: 36715   1st Qu.:215.1  
##  Median :1.551   Median :0.7644   Median : 45178   Median :238.4  
##  Mean   :1.583   Mean   :0.7509   Mean   : 53768   Mean   :253.1  
##  3rd Qu.:1.707   3rd Qu.:0.8105   3rd Qu.: 62294   3rd Qu.:279.4  
##  Max.   :2.430   Max.   :0.9114   Max.   :263261   Max.   :569.4  
##                                                                   
##      Extent          Solidity        roundness       Compactness
```

```
##  Min.   :0.5553   Min.   :0.9192   Min.   :0.4896   Min.   :0.6406
##  1st Qu.:0.7186   1st Qu.:0.9857   1st Qu.:0.8321   1st Qu.:0.7625
##  Median :0.7599   Median :0.9883   Median :0.8832   Median :0.8013
##  Mean   :0.7497   Mean   :0.9871   Mean   :0.8733   Mean   :0.7999
##  3rd Qu.:0.7869   3rd Qu.:0.9900   3rd Qu.:0.9169   3rd Qu.:0.8343
##  Max.   :0.8662   Max.   :0.9947   Max.   :0.9907   Max.   :0.9873
##
##   ShapeFactor1       ShapeFactor2        ShapeFactor3     ShapeFactor4
##  Min.   :0.002778   Min.   :0.0005642   Min.   :0.4103   Min.   :0.9477
##  1st Qu.:0.005900   1st Qu.:0.0011535   1st Qu.:0.5814   1st Qu.:0.9937
##  Median :0.006645   Median :0.0016935   Median :0.6420   Median :0.9964
##  Mean   :0.006564   Mean   :0.0017159   Mean   :0.6436   Mean   :0.9951
##  3rd Qu.:0.007271   3rd Qu.:0.0021703   3rd Qu.:0.6960   3rd Qu.:0.9979
##  Max.   :0.010451   Max.   :0.0036650   Max.   :0.9748   Max.   :0.9997
##
##       Class
##  BARBUNYA:1322
##  BOMBAY  : 522
##  CALI    :1630
##  DERMASON:3546
##  HOROZ   :1928
##  SEKER   :2027
##  SIRA    :2636
```

```
# The first 4 rows of the data
drybeans[1:4, 1:6] %>%
  knitr::kable()
```

| Area | Perimeter | MajorAxisLength | MinorAxisLength | AspectRation | Eccentricity |
|---|---|---|---|---|---|
| 28395 | 610.291 | 208.1781 | 173.8887 | 1.197191 | 0.5498122 |
| 28734 | 638.018 | 200.5248 | 182.7344 | 1.097357 | 0.4117853 |
| 29380 | 624.110 | 212.8261 | 175.9311 | 1.209713 | 0.5627273 |
| 30008 | 645.884 | 210.5580 | 182.5165 | 1.153638 | 0.4986160 |

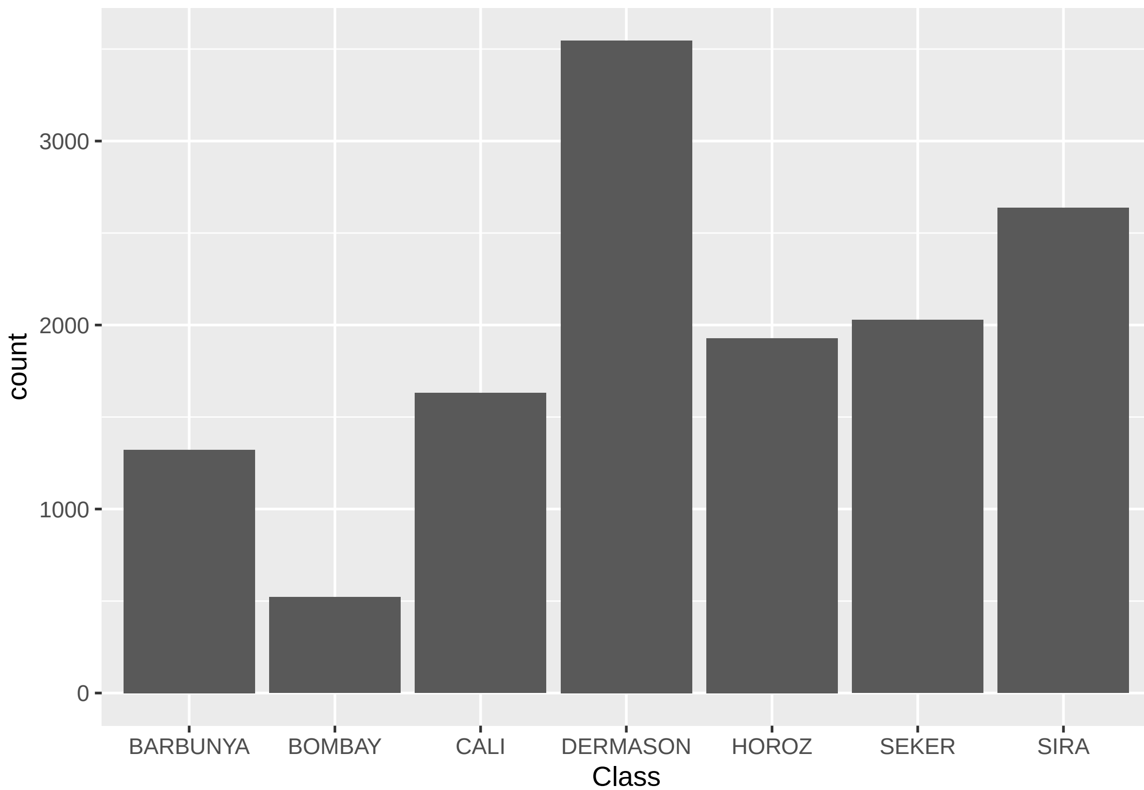
```
drybeans[1:4, 7:12] %>%
  knitr::kable()
```

| ConvexArea | EquivDiameter | Extent | Solidity | roundness | Compactness |
|---|---|---|---|---|---|
| 28715 | 190.1411 | 0.7639225 | 0.9888560 | 0.9580271 | 0.9133578 |
| 29172 | 191.2728 | 0.7839681 | 0.9849856 | 0.8870336 | 0.9538608 |
| 29690 | 193.4109 | 0.7781132 | 0.9895588 | 0.9478495 | 0.9087742 |
| 30724 | 195.4671 | 0.7826813 | 0.9766957 | 0.9039364 | 0.9283288 |

```
drybeans[1:4, 13:17] %>%
  knitr::kable()
```

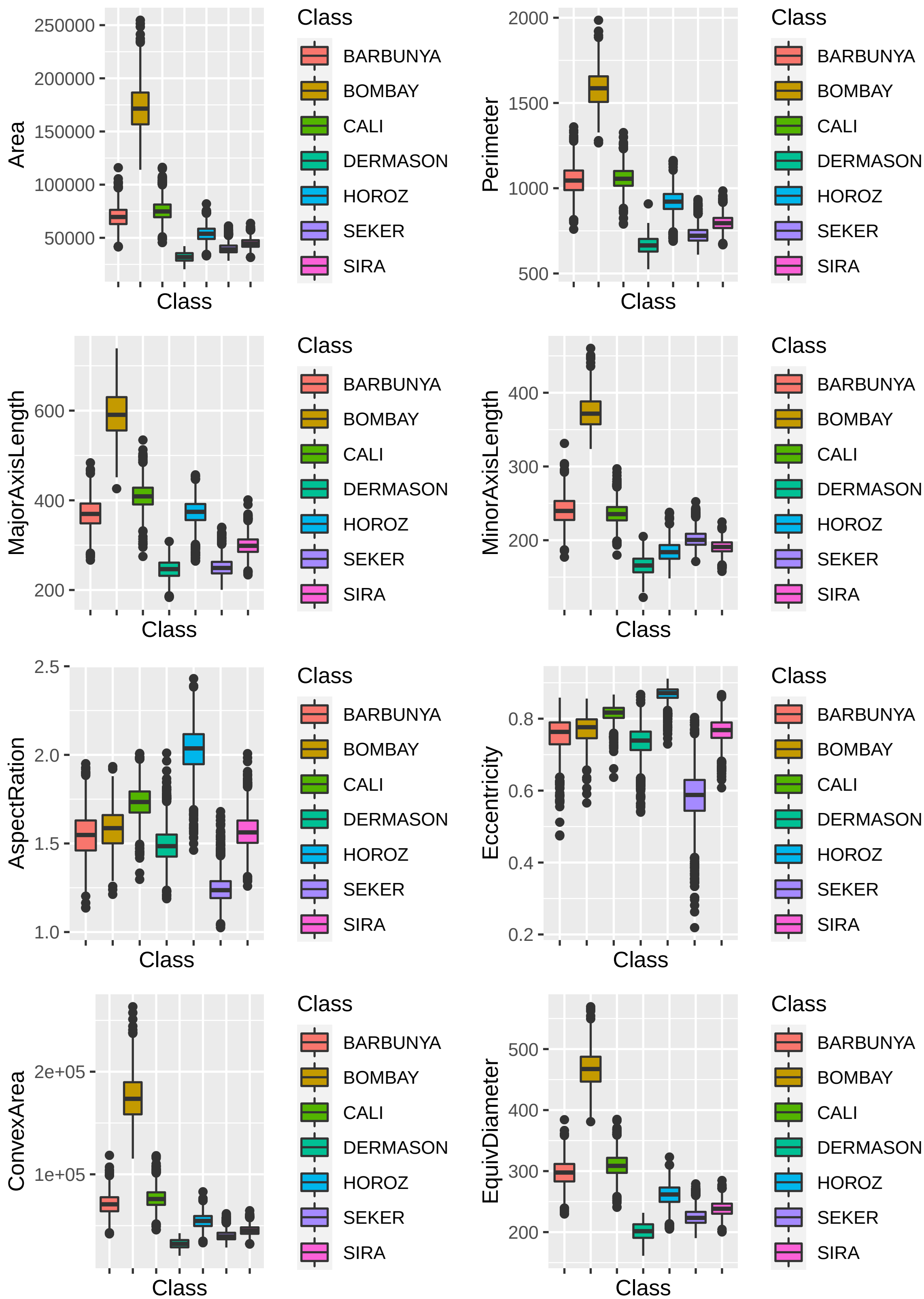
| ShapeFactor1 | ShapeFactor2 | ShapeFactor3 | ShapeFactor4 | Class |
|---|---|---|---|---|
| 0.0073315 | 0.0031473 | 0.8342224 | 0.9987239 | SEKER |
| 0.0069787 | 0.0035636 | 0.9098505 | 0.9984303 | SEKER |
| 0.0072439 | 0.0030477 | 0.8258706 | 0.9990661 | SEKER |
| 0.0070167 | 0.0032146 | 0.8617944 | 0.9941988 | SEKER |

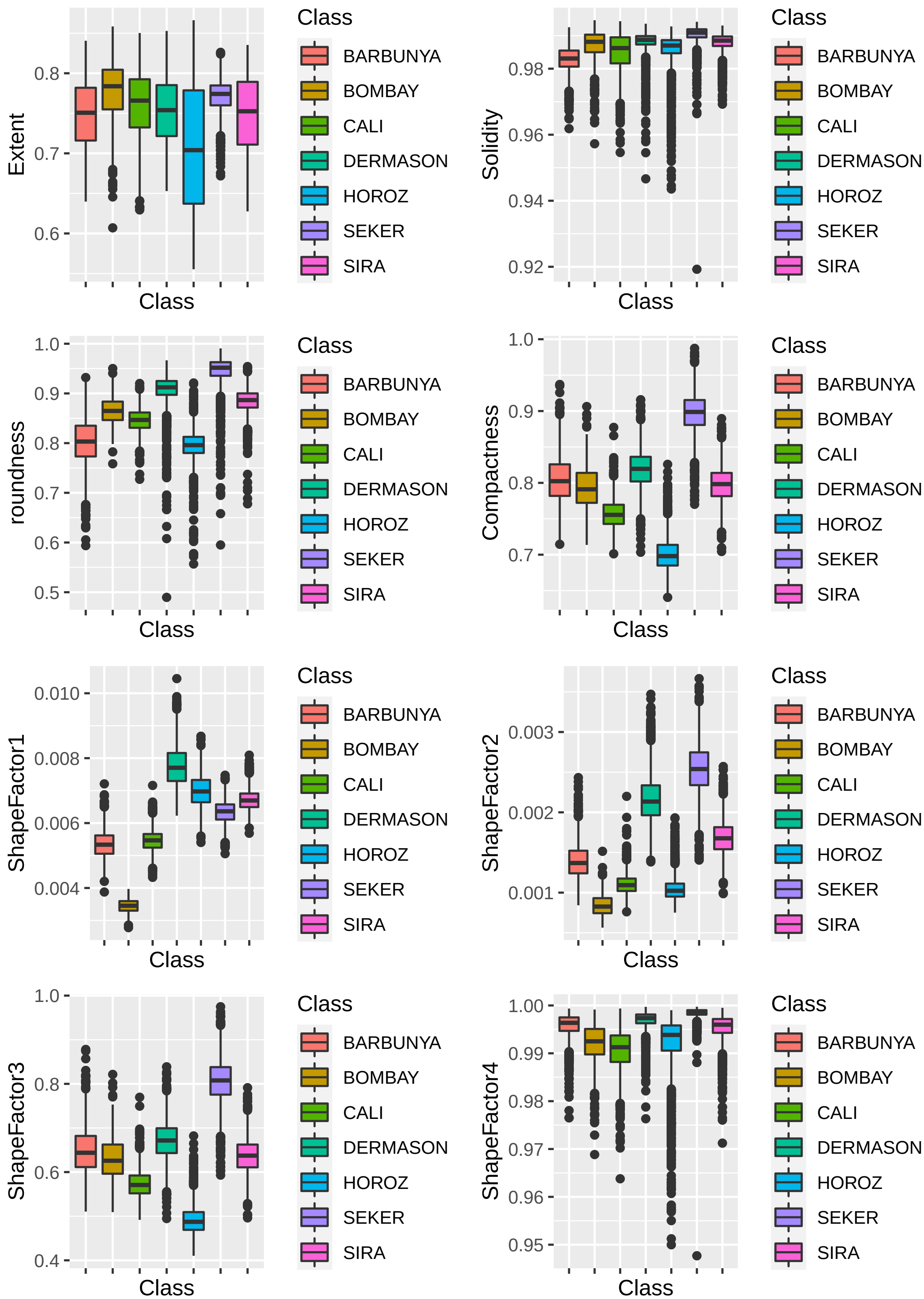
From the barchart, we can see that Dermason bean has the most number of samples and Bombay has the least number of samples.




#### 2.2.1 Box Plots of Data of Each Feature Grouped by Class (Step 3a in R file)

We study the box plots of each feature grouped by class.


Area
250000
200000
150000
100000
50000
Class
Class
BARBUNYA
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA
Perimeter
2000
1500
1000
500
Class
Class
BARBUNYA
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA
MajorAxisLength
600
400
200
Class
Class
BARBUNYA
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA
MinorAxisLength
400
300
200
Class
Class
BARBUNYA
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA
AspectRation
2.5
2.0
1.5
1.0
Class
Class
BARBUNYA
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA
Eccentricity
0.8
0.6
0.4
0.2
Class
Class
BARBUNYA
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA
ConvexArea
2e+05
1e+05
Class
Class
BARBUNYA
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA
EquivDiameter
500
400
300
200
Class
Class
BARBUNYA
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA

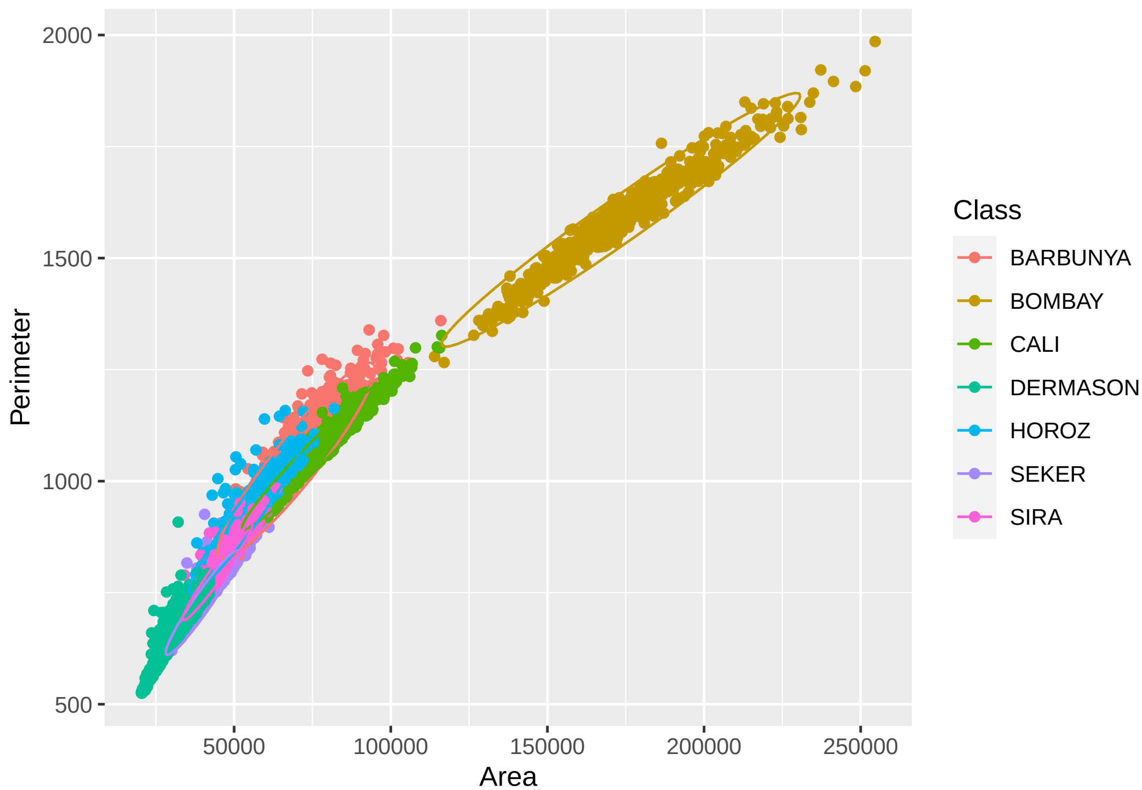

Extent
0.8
0.7
0.6
Class
Solidity
0.98
0.96
0.94
0.92
roundness
1.0
0.9
0.8
0.7
0.6
0.5
Compactness
1.0
0.9
0.8
0.7
ShapeFactor1
0.010
0.008
0.006
0.004
ShapeFactor2
0.003
0.002
0.001
ShapeFactor3
1.0
0.8
0.6
0.4
ShapeFactor4
1.00
0.99
0.98
0.97
0.96
0.95
Class
BARBUNYA
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA

From the plots, we can see that Area, MinorAxisLength and ShapeFactor1 may be useful feature to determine if a bean is of Bombay type, as the boxplots of those features almost do not overlap with those of other types. Solidity seems not quite useful in prediction as we see most of the boxplots overlap with each other. This gives us some insights when we build models to predict.

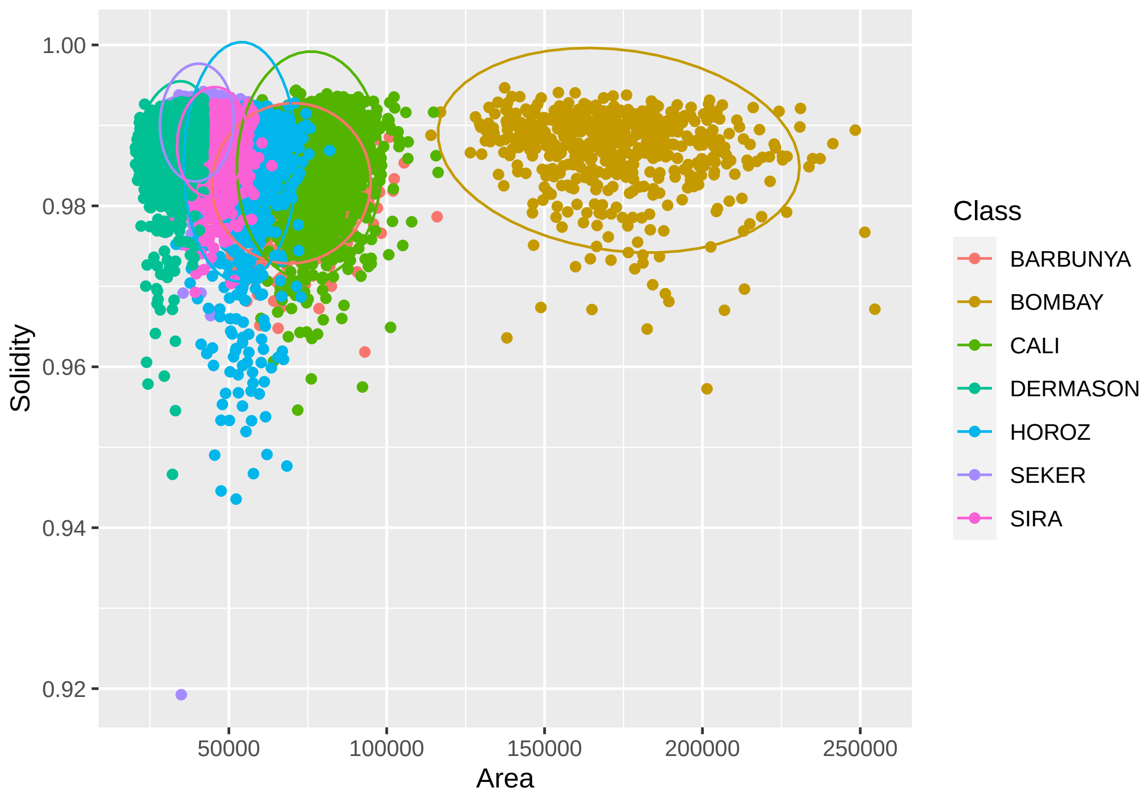
#### 2.2.2 Check Multivariate Normal Assumption (Step 3b in R file)

We use scatter plot to check if mulitvariate normal assumption holds for conditional probability of predictors given Class. However, there are 16 predictors and cross comparison will be a laborious task. We only pick a few of them to see.

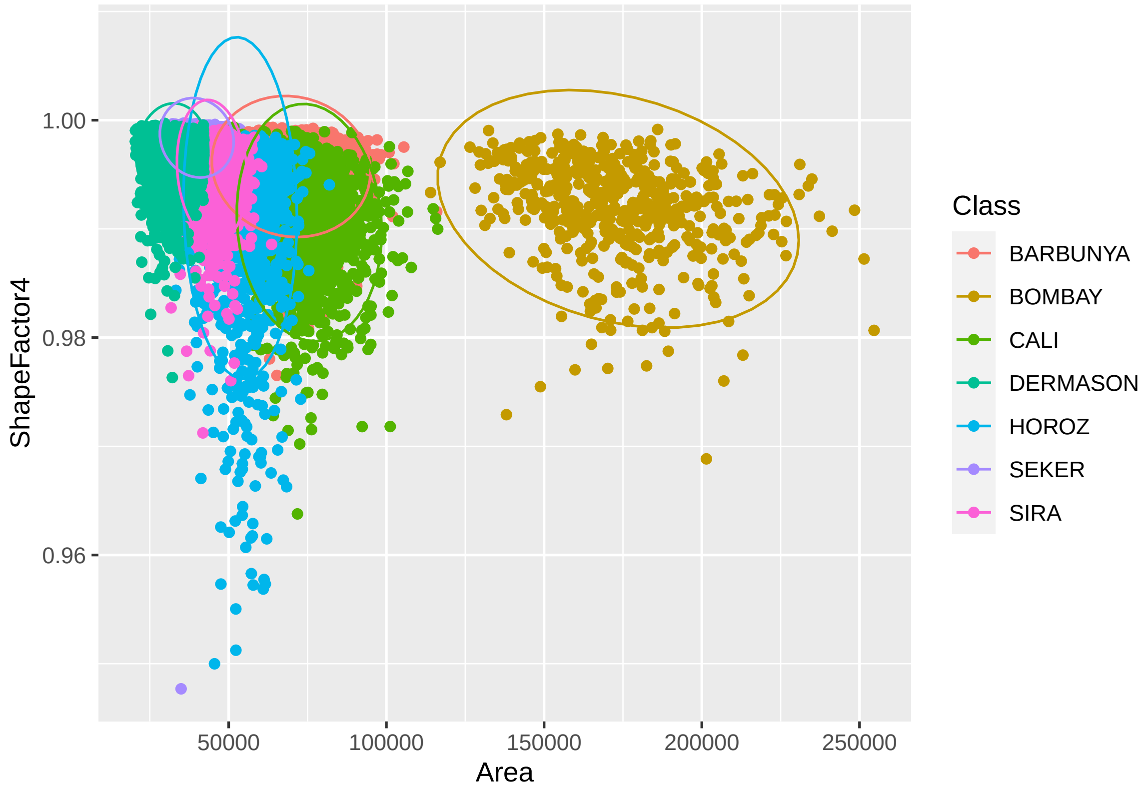
Area vs Perimeter




Area vs Solidity




Area vs ShapeFactor4




### 2.3 Model Fitting (Step 4 in R file)

The problem of interest is a classification problem. Hence, we use accuracy as the measure of effectiveness of our models. Accuracy is defined as the proportion of correct guessing in the test set. We want to pick a model with the highest accuracy.

#### 2.3.1 Model 1: Guessing with Proportion of Each Class in Train Set (Step 4a in R file)

In model 1, we start with a very simple model, pure guessing. As the proportion of each Class in the data is different, we use the proportion as the probability vector in sampling.

```
# Method 1: Get the proportion of each class in train set
# and we use it as the probability vector in sampling
prop <- train_set %>%
  group_by(Class) %>%
  summarize(n = n()) %>%
  .$n

# Probability vector
prop <- prop/nrow(train_set)
prop
```

```
## [1] 0.09708500 0.03829509 0.11978444 0.26055360 0.14166735 0.14893443 0.19368008
```

```r
# Sampling
y_hat_guess <- sample(class_vec, nrow(test_set), replace = TRUE, prob = prop) %>%
  factor()

# Use confustionMatrix() to get accuracy
conf_mat <- confusionMatrix(data = y_hat_guess, reference = test_set$Class)
conf_mat
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
##   BARBUNYA        7      6   18       35    24    14   27
##   BOMBAY          6      1    3       15    10    14    8
##   CALI           20      8   20       45    22    27   29
##   DERMASON       38     18   47       83    41    64   85
##   HOROZ          15      2   15       45    26    21   28
##   SEKER          21      5   27       52    23    32   40
##   SIRA           26     13   33       80    47    31   47
##
## Overall Statistics
##
##                Accuracy : 0.1584
##                  95% CI : (0.1394, 0.1788)
##     No Information Rate : 0.2603
##     P-Value [Acc > NIR] : 1.000
##
##                   Kappa : -0.0197
##
##  Mcnemar's Test P-Value : 0.195
##
## Statistics by Class:
##
##                      Class: BARBUNYA Class: BOMBAY Class: CALI Class: DERMASON
## Sensitivity                 0.052632     0.0188679     0.12270         0.23380
## Specificity                 0.899269     0.9572845     0.87427         0.70961
## Pos Pred Value              0.053435     0.0175439     0.11696         0.22074
## Neg Pred Value              0.897810     0.9602142     0.88013         0.72470
## Prevalence                  0.097507     0.0388563     0.11950         0.26026
## Detection Rate              0.005132     0.0007331     0.01466         0.06085
## Detection Prevalence        0.096041     0.0417889     0.12537         0.27566
## Balanced Accuracy           0.475950     0.4880762     0.49849         0.47171
##                      Class: HOROZ Class: SEKER Class: SIRA
## Sensitivity               0.13472      0.15764     0.17803
## Specificity               0.89240      0.85530     0.79091
## Pos Pred Value            0.17105      0.16000     0.16968
## Neg Pred Value            0.86221      0.85309     0.80037
## Prevalence                0.14150      0.14883     0.19355
## Detection Rate            0.01906      0.02346     0.03446
## Detection Prevalence      0.11144      0.14663     0.20308
## Balanced Accuracy         0.51356      0.50647     0.48447
```

```r
# Accuracy table to compare results of different models
acc_results <- tibble("Method" = "Guessing with Proportion",
                      Accuracy = conf_mat$overall["Accuracy"])
acc_results %>% knitr::kable()
```

| Method | Accuracy |
|---|---|
| Guessing with Proportion | 0.1583578 |

#### 2.3.2 Model 2: Multinomial Logistic Regression (Step 4b in R file)

Multinomial logistic regression is used to model nominal outcome variables, in which the log odds of the outcomes are modeled as a linear combination of the predictor variables.

This project is an example of nominal outcome variables. The Class we want to predict includes 7 types of beans (Barbunya, Bombay, Cali, Dermosan, Horoz, Seker, Sira). We use multinom function inside nnet package to do the regression.

In the model, we choose a level to be our baseline. In the R file, we do this by specifying in the relevel() function. We choose Barbunya as our baseline. The model assumes that the log of the odds of the outcomes is a linear combination of predictor variables. For example:

$$ln(P(Class = Bombay)/P(Class = Barbunya)) = b_{1,0}+b_{1,1}Area+b_{1,2}Perimeter+...+b_{1,16}ShapeFactor4$$

$$ln(P(Class = Cali)/P(Class = Barbunya)) = b_{2,0}+b_{2,1}Area+b_{2,2}Perimeter+...+b_{2,16}ShapeFactor4$$

As there are 7 classes, there will be 6 linear formulas. The parameters can be interpreted as:

E.g. $b_{1,1}$ is the increase/decrease in the log odds of being in Bombay v.s. Barbunya for each unit increase in Area.

The results of the models:

```r
# Method 2: Multinomial Logistic Regression
# Create a duplicate of train and test set. We will relevel and set the base line level to do regress
train_set2 <- train_set
train_set2$Class <- relevel(train_set2$Class, ref = "BARBUNYA")

test_set2 <- test_set
test_set2$Class <- relevel(test_set2$Class, ref = "BARBUNYA")

# Perform multinomial logistic regression
fit_multinom <- multinom(Class ~ ., data = train_set2)
```

```
## # weights:  126 (102 variable)
## initial  value 23831.561595
## iter  10 value 18578.711538
## iter  20 value 14388.805314
## iter  30 value 11292.611939
## iter  40 value 5959.809955
```

```
## iter  50 value 2647.142047
## iter  60 value 2497.703942
## iter  70 value 2459.470964
## iter  80 value 2442.896500
## iter  90 value 2434.512412
## iter 100 value 2427.941229
## final  value 2427.941229
## stopped after 100 iterations
```

```
summary(fit_multinom)
```

```
## Call:
## multinom(formula = Class ~ ., data = train_set2)
##
## Coefficients:
##          (Intercept)        Area   Perimeter MajorAxisLength MinorAxisLength
## BOMBAY      8.066406 0.002202472 -0.07556368       0.6652611        1.220175
## CALI       31.286931 0.002739978 -0.17057034       2.0130820        2.539452
## DERMASON   23.098411 0.004024030  0.19862034       0.7423646        1.513844
## HOROZ      17.599040 0.007707237  0.10165199       2.3543638        4.177007
## SEKER     -30.603397 0.008181353  0.17816608      -1.1728511       -2.380028
## SIRA       73.305190 0.004156003 -0.35762572       2.0820825        2.809615
##          AspectRation Eccentricity    ConvexArea EquivDiameter     Extent
## BOMBAY      30.146942     3.429004 -0.0006672731     -2.270577  -4.090906
## CALI       -70.492202    99.718065 -0.0028965807     -3.879202   4.189140
## DERMASON     6.796794    65.722531 -0.0019373719     -3.992561 -15.865878
## HOROZ       -2.984438    91.735544 -0.0062769769     -7.397253  -5.337211
## SEKER        6.563053   -63.819707 -0.0071882038      2.295306  -9.809124
## SIRA       -50.059825   134.545204 -0.0034492107     -4.148625  -6.555307
##            Solidity  roundness Compactness ShapeFactor1 ShapeFactor2
## BOMBAY     8.539328   13.78807    7.807329    0.8503265   0.23040930
## CALI      36.002941  -36.54630    2.468611    0.6408676   0.04843753
## DERMASON   1.289623  142.10538    2.888811    0.2662306  -0.10536432
## HOROZ     34.408821   69.82023  -19.285132    1.5120431  -0.08670580
## SEKER    -10.682862  101.19992   20.560757   -1.2726950   0.10771880
## SIRA      56.905149 -131.97822   41.344342   -0.8083527  -0.24956051
##          ShapeFactor3 ShapeFactor4
## BOMBAY       8.822289    11.398904
## CALI       -30.052649    -5.684851
## DERMASON   -22.479099     6.390750
## HOROZ      -57.781718    -1.091476
## SEKER       72.194244    10.911620
## SIRA        -3.692053    28.599374
##
## Std. Errors:
##           (Intercept)         Area   Perimeter MajorAxisLength MinorAxisLength
## BOMBAY   7.578451e-06 0.0025477665 0.003962103    0.0015592809    0.0008903055
## CALI     2.449872e-06 0.0003323737 0.001100854    0.0005766613    0.0004245156
## DERMASON 6.910087e-06 0.0006519402 0.002175456    0.0015669235    0.0017313668
## HOROZ    3.381746e-06 0.0004245823 0.001513024    0.0004880527    0.0004916816
## SEKER    4.190128e-06 0.0007867384 0.001848575    0.0006109810    0.0005340301
```

```
## SIRA     5.246452e-06 0.0004652936 0.001727038    0.0020455519    0.0020829423
##          AspectRation Eccentricity   ConvexArea EquivDiameter       Extent
## BOMBAY   1.256605e-05 5.911706e-06 0.0025074764  0.0011776762 5.561182e-06
## CALI     4.994087e-06 1.945305e-06 0.0003294044  0.0003448880 1.989136e-06
## DERMASON 1.743317e-05 6.623232e-06 0.0006601127  0.0007620996 6.799306e-06
## HOROZ    4.554518e-06 2.403623e-06 0.0004197703  0.0004814120 2.538118e-06
## SEKER    5.346864e-06 2.831804e-06 0.0007848982  0.0005709730 3.232698e-06
## SIRA     2.176939e-05 7.114558e-06 0.0004661434  0.0006671851 5.891994e-06
##              Solidity    roundness  Compactness ShapeFactor1 ShapeFactor2
## BOMBAY   7.482197e-06 6.644369e-06 5.918891e-06 6.036101e-08 1.454778e-08
## CALI     2.417166e-06 2.397254e-06 2.347184e-06 2.078645e-08 7.614702e-09
## DERMASON 6.828661e-06 8.156642e-06 1.028405e-05 5.853134e-08 5.599666e-08
## HOROZ    3.323478e-06 3.079512e-06 3.000908e-06 2.758798e-08 8.710391e-09
## SEKER    4.144503e-06 3.852697e-06 3.726713e-06 3.442196e-08 1.305173e-08
## SIRA     5.162061e-06 7.827888e-06 1.076398e-05 3.283377e-08 5.787071e-08
##          ShapeFactor3 ShapeFactor4
## BOMBAY   4.644848e-06 7.569048e-06
## CALI     2.339447e-06 2.440367e-06
## DERMASON 1.280169e-05 6.916848e-06
## HOROZ    2.647216e-06 3.361361e-06
## SEKER    3.331003e-06 4.181629e-06
## SIRA     1.422009e-05 5.316424e-06
##
## Residual Deviance: 4855.882
## AIC: 5059.882
```

```r
# Predict probability with multinorm model
p_hat_multinom <- predict(fit_multinom, newdata = test_set2, type = "probs")

# Pick the highest probability one as our prediction
y_hat_multinom <- class_vec[apply(p_hat_multinom, 1, which.max)] %>%
  factor()
conf_mat <- confusionMatrix(data = y_hat_multinom, reference = test_set$Class)

# Append results to accurcay table
acc_results <- bind_rows(acc_results, tibble("Method" = "Multinomial Logistic Regression",
                      Accuracy = conf_mat$overall["Accuracy"]))
acc_results %>% knitr::kable()
```

| Method | Accuracy |
|---|---|
| Guessing with Proportion | 0.1583578 |
| Multinomial Logistic Regression | 0.9193548 |

For the coefficients part in the summary of the fitted model, there are six rows. It is because we have six linear formula corresponding to 6 other classes against the baseline Barbunya.

### 2.3.3 Model 3: Multinomial Logistic Regression with Cross Validation (Step 4c in R file)

Model 2 only uses one data set to train model. To improve accuracy, we can use multiple data sets to train the model. This can be done by cross validation. We will use train() function to do this. In later models, we also use train() function to do cross validation. We set trainControl so that we train with 5 validatiom samples, each comprises of 10% of observations.

```
# Set control with train: 5 validation samples comprise of 10% of observations each
control <- trainControl(method = "cv", number = 5, p = 0.9)
```

```
# Model 3: Multinom with train

# train with multinom model
fit_multinom_tr <- train(Class ~ .,
                         data = train_set,
                         method = "multinom",
                         trControl = control,
                         trace = FALSE)
# Predicted probability
p_hat_multinom_tr <- predict(fit_multinom_tr, newdata = test_set, type = "prob")
# Predicted outcome
y_hat_multinom_tr <- predict(fit_multinom_tr, newdata = test_set, type = "raw")

conf_mat <- confusionMatrix(data = y_hat_multinom_tr, reference = test_set$Class)

# Append results to accurcay table
acc_results <- bind_rows(acc_results, tibble("Method" = "Multinomial Logistic Regression with CV",
                                             Accuracy = conf_mat$overall["Accuracy"]))
acc_results %>% knitr::kable()
```

| Method | Accuracy |
|---|---|
| Guessing with Proportion | 0.1583578 |
| Multinomial Logistic Regression | 0.9193548 |
| Multinomial Logistic Regression with CV | 0.9193548 |

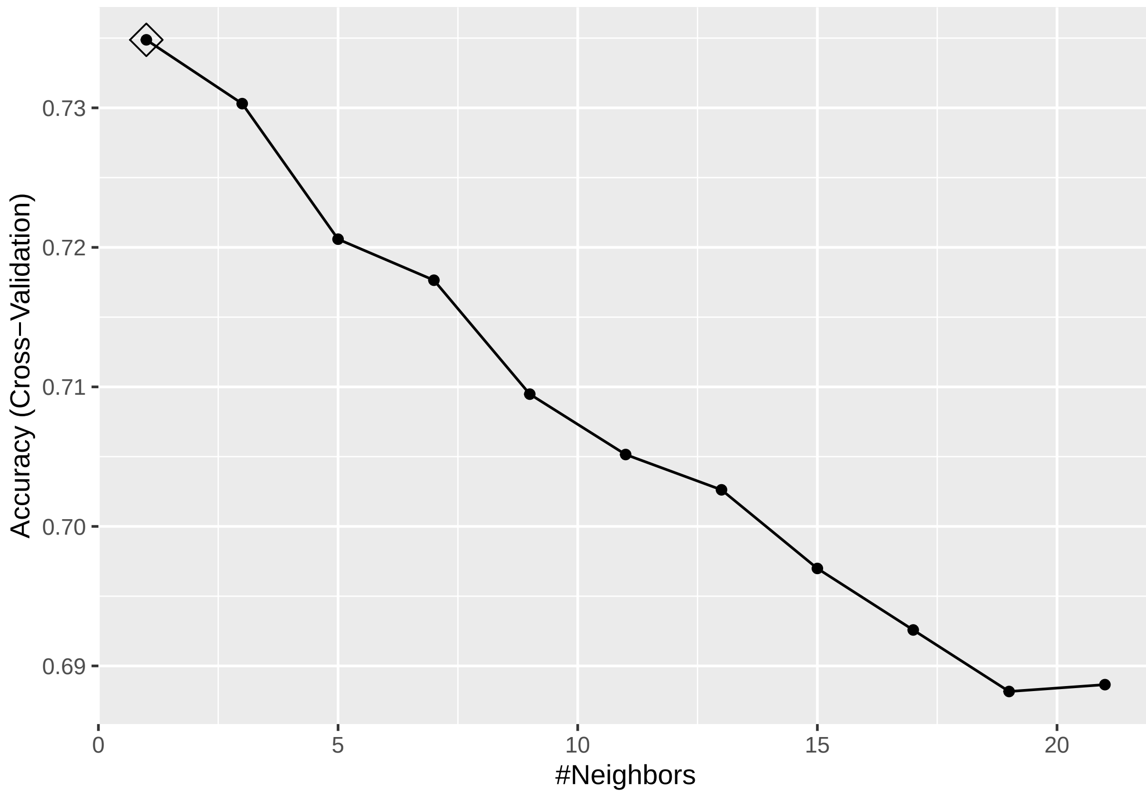
### 2.3.4 Model 4: Knn Model with Cross Validation (Step 4d in R file)

Knn model searches for the k-th nearest neighbours to a target. It uses a majority rule to make a prediction.

Results:

```
# Method 4: Knn with train()
fit_knn_tr <- train(Class ~ .,
                    data = train_set,
                    method = "knn",
                    trControl = control,
```

```r
                        tuneGrid = data.frame(k = seq(1, 21, by = 2)))
ggplot(fit_knn_tr, highlight = TRUE)
```




```r
# Predicted probability
p_hat_knn_tr <- predict(fit_knn_tr, newdata = test_set, type = "prob")
# Predicted outcome
y_hat_knn_tr <- predict(fit_knn_tr, newdata = test_set, type = "raw")

conf_mat <- confusionMatrix(data = y_hat_knn_tr, reference = test_set$Class)

# Append results to accurcay table
acc_results <- bind_rows(acc_results, tibble("Method" = "Knn with CV",
                                             Accuracy = conf_mat$overall["Accuracy"]))
acc_results %>% knitr::kable()
```

| Method | Accuracy |
|---|---|
| Guessing with Proportion | 0.1583578 |
| Multinomial Logistic Regression | 0.9193548 |
| Multinomial Logistic Regression with CV | 0.9193548 |
| Knn with CV | 0.7353372 |

The Knn model seems not to be a good model. We see the best fit occurs when k = 1. We only take

1 nearest observation to make prediction. This may overtrain the model. In fact, if we allow k = 0, the best fit occurs when k = 0. This means the best prediction is the observation itself. This does not provide much useful information.

#### 2.3.5 Model 5: QDA Model with Cross Validation (Step 4e in R file)

QDA model is a generative model in which we assume the conditional probabilities of the predictors are multivariate normal.

Results:

```
# Model 5: QDA
# Normal assumption
fit_qda_tr <- train(Class ~ ., method = "qda", data = train_set,
                    trControl = control)

# Predicted probability
p_hat_qda_tr <- predict(fit_qda_tr, newdata = test_set, type = "prob")
# Predicted outcome
y_hat_qda_tr <- predict(fit_qda_tr, newdata = test_set, type = "raw")

conf_mat <- confusionMatrix(data = y_hat_qda_tr, reference = test_set$Class)

# Append results to accurcay table
acc_results <- bind_rows(acc_results, tibble("Method" = "QDA with CV",
                                             Accuracy = conf_mat$overall["Accuracy"]))
acc_results %>% knitr::kable()
```

| Method | Accuracy |
|---|---|
| Guessing with Proportion | 0.1583578 |
| Multinomial Logistic Regression | 0.9193548 |
| Multinomial Logistic Regression with CV | 0.9193548 |
| Knn with CV | 0.7353372 |
| QDA with CV | 0.9098240 |

#### 2.3.6 Model 6: LDA Model with Cross Validation (Step 4f in R file)

LDA model is a variate of generative model based on different assumptions.

Results:

```
# Model 6: LDA
fit_lda_tr <- train(Class ~ ., method = "lda", data = train_set,
                    trControl = control)

# Predicted probability
p_hat_lda_tr <- predict(fit_lda_tr, newdata = test_set, type = "prob")
# Predicted outcome
y_hat_lda_tr <- predict(fit_lda_tr, newdata = test_set, type = "raw")
```

```r
conf_mat <- confusionMatrix(data = y_hat_lda_tr, reference = test_set$Class)

# Append results to accurcay table
acc_results <- bind_rows(acc_results, tibble("Method" = "LDA with CV",
                                             Accuracy = conf_mat$overall["Accuracy"]))
acc_results %>% knitr::kable()
```

| Method | Accuracy |
|---|---|
| Guessing with Proportion | 0.1583578 |
| Multinomial Logistic Regression | 0.9193548 |
| Multinomial Logistic Regression with CV | 0.9193548 |
| Knn with CV | 0.7353372 |
| QDA with CV | 0.9098240 |
| LDA with CV | 0.9024927 |

#### 2.3.7 Model 7: Classification Tree Model with Cross Validation (Step 4g in R file)

We use a classification tree to predict Class. At each tree node, there is a decision rule. Following the decision rules will bring us to the predicted value.

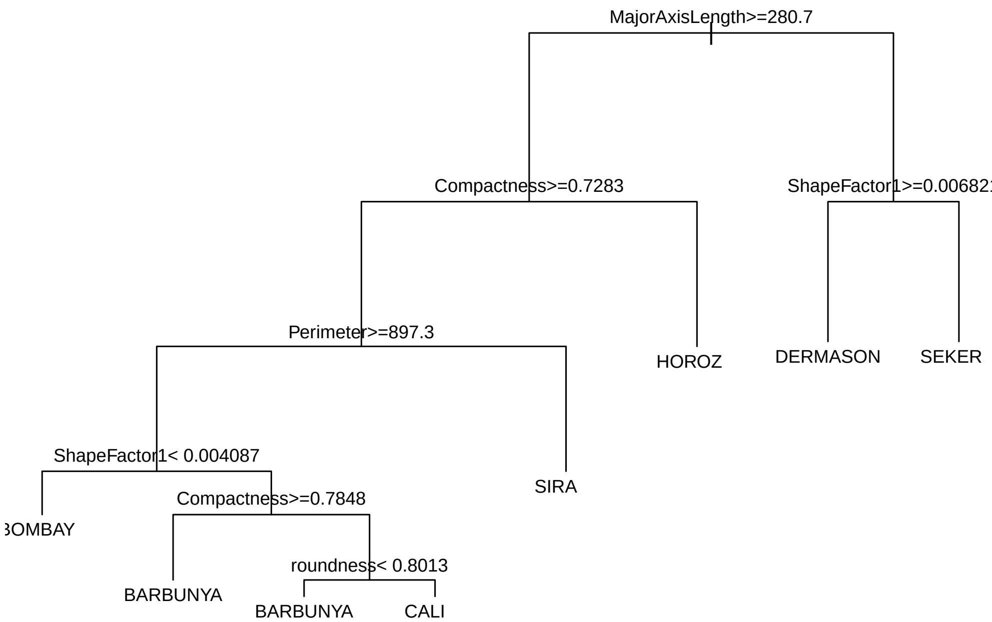
Results:

```r
# Model 7: Classification Tree
fit_rpart_tr <- train(Class ~ ., method = "rpart", data = train_set,
                      trControl = control,
                      tuneGrid = data.frame(cp = seq(0.01, 0.05, leng = 25)))

# Predicted probability
p_hat_rpart_tr <- predict(fit_rpart_tr, newdata = test_set, type = "prob")
# Predicted outcome
y_hat_rpart_tr <- predict(fit_rpart_tr, newdata = test_set, type = "raw")

conf_mat <- confusionMatrix(data = y_hat_rpart_tr, reference = test_set$Class)

plot(fit_rpart_tr$finalModel)
text(fit_rpart_tr$finalModel, cex = 0.75)
```




```r
# Append results to accurcay table
acc_results <- bind_rows(acc_results, tibble("Method" = "Classification Tree with CV",
                                             Accuracy = conf_mat$overall["Accuracy"]))
acc_results %>% knitr::kable()
```

| Method | Accuracy |
|---|---|
| Guessing with Proportion | 0.1583578 |
| Multinomial Logistic Regression | 0.9193548 |
| Multinomial Logistic Regression with CV | 0.9193548 |
| Knn with CV | 0.7353372 |
| QDA with CV | 0.9098240 |
| LDA with CV | 0.9024927 |
| Classification Tree with CV | 0.8731672 |

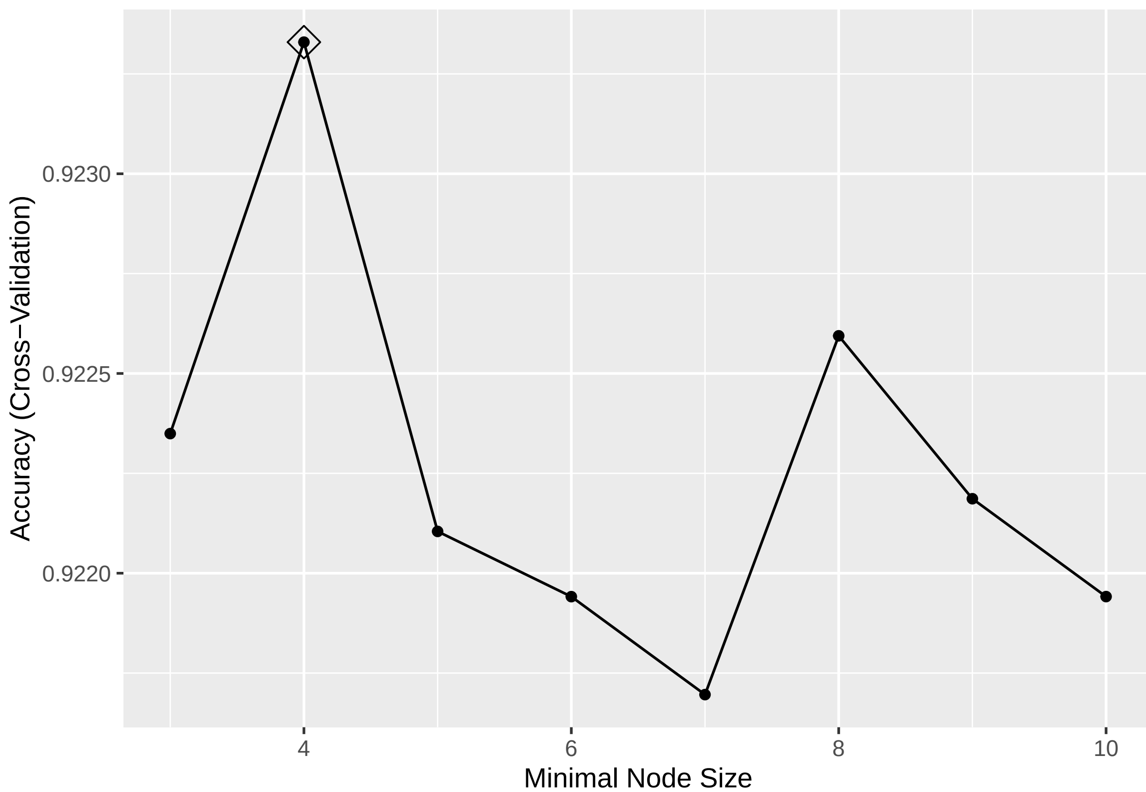
#### 2.3.8 Model 8: Random Forest Model with Cross Validation (Step 4h in R file)

Random forest model is like a collection of decision trees. We average out the values and make predictions. Note that this model may take 15-30 minutes to run.

Results:

```r
# Model 8: Random Forest
# This may take 30 mins to run
# May set minNode to seq(3, 50), but take longer time
fit_rf_tr <- train(Class ~ ., method = "Rborist", data = train_set,
                   trControl = control,
                   tuneGrid = data.frame(predFixed = 2, minNode = seq(3, 10)))

ggplot(fit_rf_tr, highlight = TRUE)
```




```r
# Predicted probability
p_hat_rf_tr <- predict(fit_rf_tr, newdata = test_set, type = "prob")
# Predicted outcome
y_hat_rf_tr <- predict(fit_rf_tr, newdata = test_set, type = "raw")

conf_mat <- confusionMatrix(data = y_hat_rf_tr, reference = test_set$Class)

# Append results to accurcay table
acc_results <- bind_rows(acc_results, tibble("Method" = "Random Forest with CV",
                                             Accuracy = conf_mat$overall["Accuracy"]))
acc_results %>% knitr::kable()
```

| Method | Accuracy |
|---|---|
| Guessing with Proportion | 0.1583578 |
| Multinomial Logistic Regression | 0.9193548 |
| Multinomial Logistic Regression with CV | 0.9193548 |
| Knn with CV | 0.7353372 |
| QDA with CV | 0.9098240 |
| LDA with CV | 0.9024927 |
| Classification Tree with CV | 0.8731672 |
| Random Forest with CV | 0.9186217 |

#### 2.3.9 Model 9: Ensemble (Step 4i in R file)

We take average of the previous models and predict based on the maximum probability.

Results:

```
# Model 9: Ensemble
# Take average of the previous models and predict based on the maximum probability.

# Predicted probability
p_hat_ensemble <- (p_hat_multinom_tr + p_hat_knn_tr +
                     p_hat_qda_tr + p_hat_lda_tr +
                     p_hat_rpart_tr + p_hat_rf_tr) / 6

# The maximum probability
p_max_ensemble <- apply(p_hat_ensemble, 1, max)

# Predicted outcome
y_hat_ensemble <- class_vec[apply(p_hat_ensemble, 1, which.max)] %>%
  factor()

conf_mat <- confusionMatrix(data = y_hat_ensemble, reference = test_set$Class)

# Append results to accurcay table
acc_results <- bind_rows(acc_results, tibble("Method" = "Ensemble with CV",
                                             Accuracy = conf_mat$overall["Accuracy"]))
acc_results %>% knitr::kable()
```

| Method | Accuracy |
|---|---|
| Guessing with Proportion | 0.1583578 |
| Multinomial Logistic Regression | 0.9193548 |
| Multinomial Logistic Regression with CV | 0.9193548 |
| Knn with CV | 0.7353372 |
| QDA with CV | 0.9098240 |
| LDA with CV | 0.9024927 |
| Classification Tree with CV | 0.8731672 |
| Random Forest with CV | 0.9186217 |
| Ensemble with CV | 0.9178886 |

# 3 Results

## 3.1 Results (Step 5 in R file)

```
acc_results %>% knitr::kable()
```

| Method | Accuracy |
|---|---|
| Guessing with Proportion | 0.1583578 |
| Multinomial Logistic Regression | 0.9193548 |
| Multinomial Logistic Regression with CV | 0.9193548 |
| Knn with CV | 0.7353372 |
| QDA with CV | 0.9098240 |
| LDA with CV | 0.9024927 |
| Classification Tree with CV | 0.8731672 |
| Random Forest with CV | 0.9186217 |
| Ensemble with CV | 0.9178886 |

## 3.2 Further Study Why We Make Errors (Step 5a in R file)

From the results, multinomial logistic regression, random forest and ensemble model seem to top the list in performance. Our best performing accuracy is 0.9193548. Is there room to improve? We are going to study the samples which we are so sure about but make mistake in predicting them.

```
# Check the cases we are so sure about but we make a wrong prediction
ind <- which(y_hat_ensemble != test_set$Class)
ind <- ind[order(p_max_ensemble[ind], decreasing = TRUE)]

# Table showing the probability in ensemble and our prediction and the actual value.
data.frame(p_hat_ensemble[ind[1:10],]) %>%
  mutate(predict = y_hat_multinom_tr[ind[1:10]],
         actual = test_set$Class[ind[1:10]],
         row = ind[1:10])
```

```
##          BARBUNYA       BOMBAY         CALI     DERMASON        HOROZ
## 396  7.027723e-04 6.001142e-04 9.996771e-03 9.987378e-05 9.872412e-01
## 394  3.857609e-03 6.676514e-04 1.169129e-02 1.024479e-04 9.763877e-01
## 399  2.933073e-03 6.104767e-04 2.791678e-02 9.993583e-05 9.669074e-01
## 749  4.610415e-08 2.492077e-07 1.229066e-09 9.658699e-01 6.119706e-04
## 1354 4.575474e-04 3.497925e-06 9.097264e-05 1.968483e-02 9.877682e-05
## 748  5.061238e-09 6.051470e-08 4.275366e-10 9.566842e-01 6.521543e-04
## 205  1.549029e-02 2.642431e-05 1.400639e-03 1.158401e-02 6.894478e-03
## 570  4.794100e-03 7.788532e-06 6.410925e-04 2.795481e-02 9.032745e-03
## 752  1.518799e-08 1.995208e-07 1.508782e-09 9.475125e-01 3.356910e-03
## 747  1.298832e-08 6.502981e-08 1.711512e-10 9.407751e-01 2.300602e-03
##             SEKER        SIRA  predict   actual  row
## 396  4.698022e-12 0.001359317    HOROZ     CALI  396
## 394  2.257796e-07 0.007293062    HOROZ     CALI  394
```

```
## 399  5.341032e-10 0.001532316    HOROZ     CALI  399
## 749  6.319050e-03 0.027198832 DERMASON     SIRA  749
## 1354 9.656678e-01 0.013996577    SEKER DERMASON 1354
## 748  4.270395e-03 0.038393197 DERMASON     SIRA  748
## 205  9.787368e-03 0.954816794     SIRA BARBUNYA  205
## 570  9.774370e-03 0.947795089     SIRA    HOROZ  570
## 752  4.271900e-03 0.044858522 DERMASON     SIRA  752
## 747  4.266358e-03 0.052657818 DERMASON     SIRA  747
```

```
# Prediction under different models
data.frame(multinom = y_hat_multinom_tr[ind[1:10]],
           knn = y_hat_knn_tr[ind[1:10]],
           qda = y_hat_qda_tr[ind[1:10]],
           lda = y_hat_lda_tr[ind[1:10]],
           rpart = y_hat_rpart_tr[ind[1:10]],
           rf = y_hat_rf_tr[ind[1:10]],
           actual = test_set$Class[ind[1:10]]) %>%
  knitr::kable()
```

| multinom | knn | qda | lda | rpart | rf | actual |
|---|---|---|---|---|---|---|
| HOROZ | HOROZ | HOROZ | HOROZ | HOROZ | HOROZ | CALI |
| HOROZ | HOROZ | HOROZ | HOROZ | HOROZ | HOROZ | CALI |
| HOROZ | HOROZ | HOROZ | HOROZ | HOROZ | HOROZ | CALI |
| DERMASON | DERMASON | DERMASON | DERMASON | DERMASON | DERMASON | SIRA |
| SEKER | SEKER | SEKER | SEKER | SEKER | SEKER | DERMASON |
| DERMASON | DERMASON | DERMASON | DERMASON | DERMASON | DERMASON | SIRA |
| SIRA | SIRA | SIRA | SIRA | SIRA | SIRA | BARBUNYA |
| SIRA | SIRA | SIRA | SIRA | SIRA | SIRA | HOROZ |
| DERMASON | DERMASON | DERMASON | DERMASON | DERMASON | DERMASON | SIRA |
| DERMASON | DERMASON | DERMASON | DERMASON | DERMASON | DERMASON | SIRA |

We can see that of those records which we make mistake, almost all of our models make errors in predicting the correct class. For example, for the first row in the table which shows prediction under different models, all models predict 'Horoz' but the actual class is 'Cali'.

We choose the Area and AspectRation of the two classes to study. We see that Area and AspectRation of the observation is within the main chunk in Horoz boxplot but outliers in Cali boxplot.

```
# Study of the the fisrt item which we make a wrong prediction
myrow <- ind[1]

mytable <- test_set %>%
  slice(myrow)
mytable
```

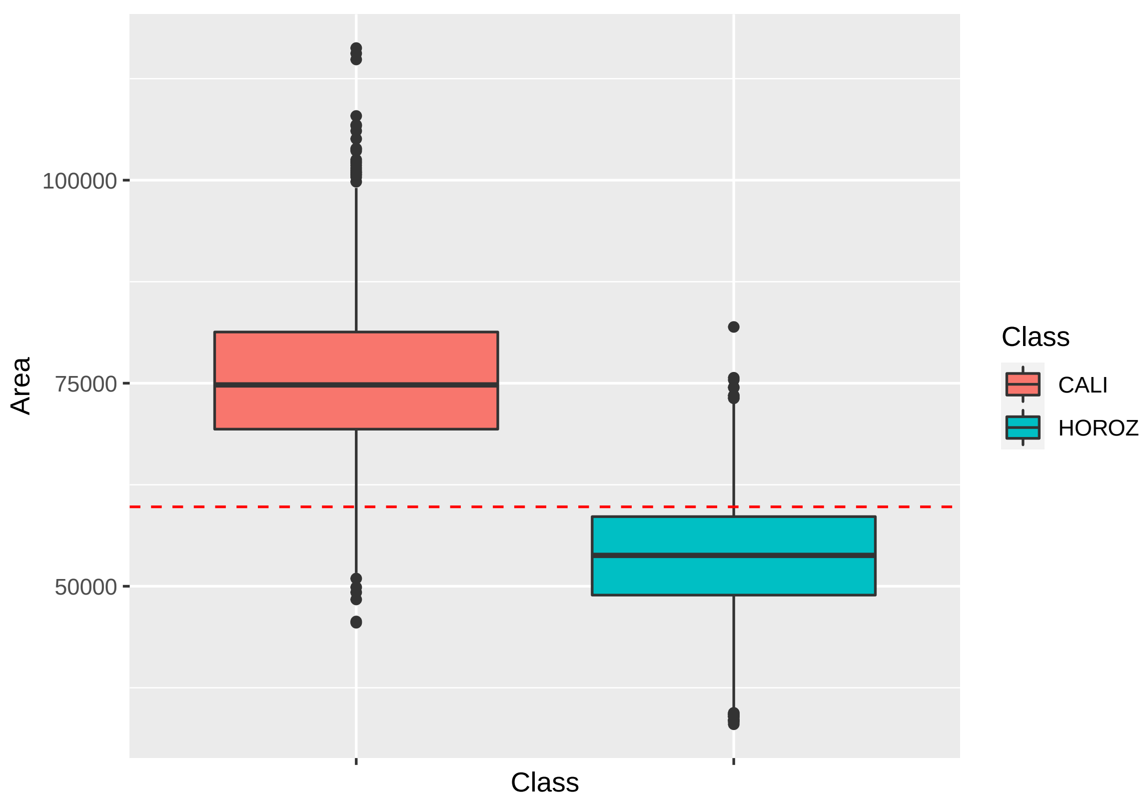
```
## # A tibble: 1 x 17
##    Area Perimeter MajorAxisLength MinorAxisLength AspectRation Eccentricity
##   <dbl>     <dbl>           <dbl>           <dbl>        <dbl>        <dbl>
## 1 59780      977.            390.            196.         1.99        0.864
```

```
## # ... with 11 more variables: ConvexArea <dbl>, EquivDiameter <dbl>,
## #   Extent <dbl>, Solidity <dbl>, roundness <dbl>, Compactness <dbl>,
## #   ShapeFactor1 <dbl>, ShapeFactor2 <dbl>, ShapeFactor3 <dbl>,
## #   ShapeFactor4 <dbl>, Class <fct>
```

```
# Compare of CALI & HOROZ Area
mytable$Area
```

```
## [1] 59780
```

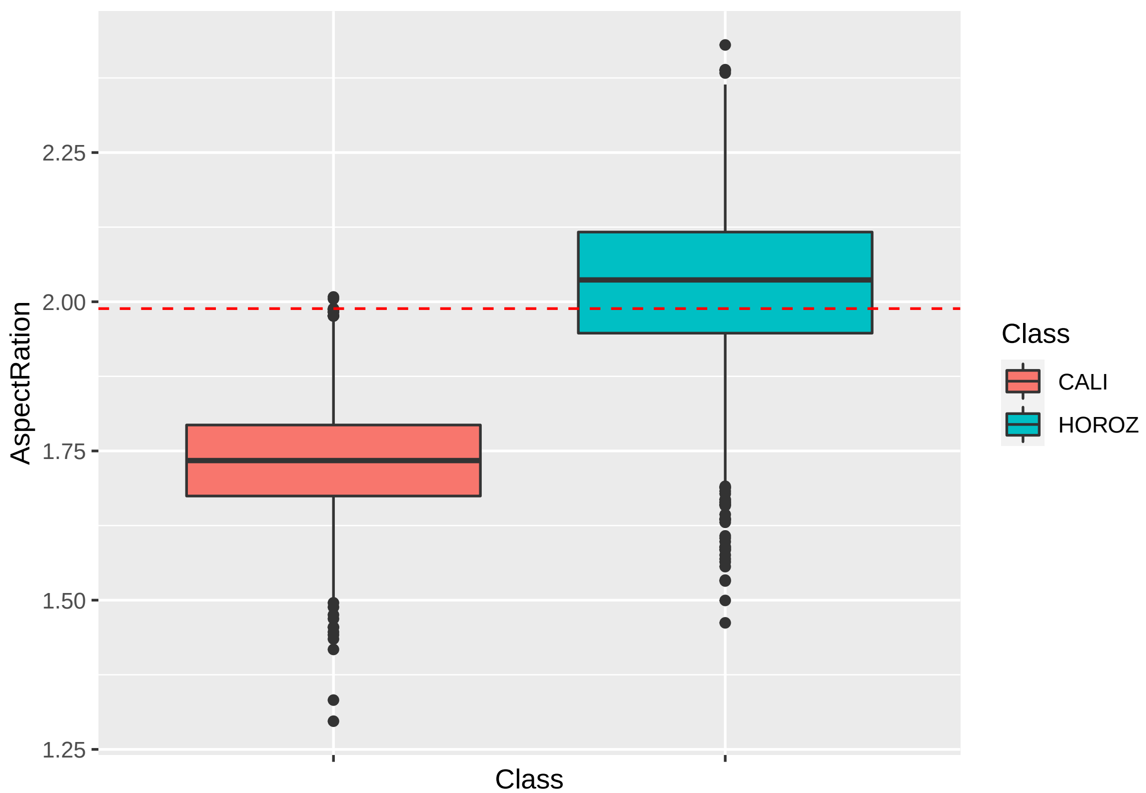
```
# The value of the observation is marked as the red line
drybeans %>%
  filter(Class %in% c("CALI", "HOROZ")) %>%
  ggplot(aes(Class, Area, fill = Class)) +
  geom_boxplot() +
  theme(axis.text.x = element_blank()) +
  geom_hline(yintercept = mytable$Area, linetype = "dashed", color = "red")
```




```
# Compare of CALI & HOROZ AspectRation
mytable$AspectRation
```

```
## [1] 1.988539
```

```r
# The value of the observation is marked as the red line
drybeans %>%
  filter(Class %in% c("CALI", "HOROZ")) %>%
  ggplot(aes(Class, AspectRation, fill = Class)) +
  geom_boxplot() +
  theme(axis.text.x = element_blank()) +
  geom_hline(yintercept = mytable$AspectRation, linetype = "dashed", color = "red")
```




The red line in the plots above is the value of the observation of interest. It lies close to the middle 50% region for class 'Horoz' and on the ends of distribution for class 'Cali'. This is why the models tend to predict 'Horoz' instead of 'Cali'.

We can see the number of class SDs the observation predictor values lie away from the class averages.

```r
# How many SDs does the statistics of the sample deviate from Class average
cali_mean <- drybeans %>%
  filter(Class %in% c("CALI")) %>%
  select(-Class) %>%
  summarize_all(mean)

cali_sd <- drybeans %>%
  filter(Class %in% c("CALI")) %>%
  select(-Class) %>%
  summarize_all(sd)
```

```r
horoz_mean <- drybeans %>%
  filter(Class %in% c("HOROZ")) %>%
  select(-Class) %>%
  summarize_all(mean)

horoz_sd <- drybeans %>%
  filter(Class %in% c("HOROZ")) %>%
  select(-Class) %>%
  summarize_all(sd)

# How many SDs does the statistics of the sample deviate from Cali average
v1 <- mytable %>%
  select(-Class)
v1 <- (v1 - cali_mean)/cali_sd
v1
```

```
##        Area Perimeter MajorAxisLength MinorAxisLength AspectRation Eccentricity
## 1 -1.680001 -1.187268      -0.6472625       -2.729723     2.785501     2.169354
##   ConvexArea EquivDiameter    Extent  Solidity roundness Compactness
## 1  -1.696555     -1.757871 0.1842126 0.4390987 -2.534037   -2.432497
##   ShapeFactor1 ShapeFactor2 ShapeFactor3 ShapeFactor4
## 1     3.180869   -0.7932642    -2.348756    0.5586412
```

```r
# Number of features that are within 2 SD from the class average
sum(abs(v1) <= 2)
```

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

```r
# How many SDs does the statistics of the sample deviate from Horoz average
v2 <- mytable %>%
  select(-Class)
v2 <- (v2 - horoz_mean)/horoz_sd
v2
```

```
##       Area Perimeter MajorAxisLength MinorAxisLength AspectRation Eccentricity
## 1 0.835194 0.8208518       0.5899884       0.9015387   -0.2776292   -0.1465733
##   ConvexArea EquivDiameter    Extent Solidity  roundness Compactness
## 1  0.8175518      0.838121 0.8008965 0.342498 -0.2511262   0.2455818
##   ShapeFactor1 ShapeFactor2 ShapeFactor3 ShapeFactor4
## 1   -0.9399277   -0.2866797    0.2263297    0.1786512
```

```r
# Number of features that are within 2 SD from the class average
sum(abs(v2) <= 2)
```

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

For the statistics of Horoz, the observation of interest has all its values lying within 2 SD from the class average. However, for the statistics of Cali, the observation of interest only has 9 out of 16 values lying within 2 SD from the class average. This means we have a Cali bean with feature that resembles a Horoz bean. This is a difficult problem to tackle. Maybe there is some feature that has not been captured by the image.

# 4 Conclusion

In this project, we try to build a model to identify 7 types of bean based on features obtained from high-resolution images taken from a camera. We try a total of 9 models. The maximum accuracy is 0.9193548 and is quite satisfactory. It passes 90%. We found that multinomial logistic regression model, random forest model and ensemble model perform the best. The QDA and LDA models also provide satisfying results with slightly lower accuracy. When we studied the results which we made mistakes, we found that some beans have features that look like the other bean types. One possible reason is that the photos cannot completely reflect features of bean types. There are some features we have ignored. This may be improved in future work to increase accuracy of the models.

---

# Resources

The resources that I have referenced are listed here.


1. KOKLU, M. and OZKAN, I.A. (2020, September 14). Dry Bean Dataset Data Set. UCI Machines Learning Repository. https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset (The data source)

2. KOKLU, M. and OZKAN, I.A. (2020). Multiclass Classification of Dry Beans Using Computer Vision and Machine Learning Techniques. Computers and Electronics in Agriculture, 174, 105507. Science Direct. https://www.sciencedirect.com/science/article/abs/pii/S0168169919311573?via%3Dihub (Citation request from the data source)

3. (n.d.). MULTINOMIAL LOGISTIC REGRESSION | R DATA ANALYSIS EXAMPLES. UCLA: Statistical Consulting Group. https://stats.idre.ucla.edu/r/dae/multinomial-logistic-regression/ (How to use multinomial logistics regression)

4. David D. (2020, October 28). R for Statistical Learning - Chapter 21 The caret Package. Retrieved March 30, 2021, from https://daviddalpiaz.github.io/r4sl/the-caret-package.html (How to train with multinom regression)