R Data Science Course 9: Choose Your Own Project

James Lo

2022-08-29

Contents

Introduction

Executive Summary

Machine learning has become increasingly relevant in agriculture today, as human population grows and researchers seek more efficient methods to meet demands for food on limited arable land. In particular, classifying large volumes of crops is important in modern agriculture in several contexts, including quality assessment after harvest and modeling the success of different crop varieties in various growing conditions. The use of machine learning algorithms for crop classification provides several advantages over manual approaches including higher throughput, superior accuracy and precision, and the development of more complex models capable of recognizing data trends that a human observer may have difficulty noticing. Focusing on the classification of different crop varieties, this project seeks to analyze the potential predictors in a previously generated dataset, before choosing a small collection of appropriate algorithms and evaluating the performances of the models generated.

This project is part of the EdX Harvard R Data Science program.

Data Loading

The dataset selected was sourced from the UCI Machine Learning Repository, under the name "Dry Bean Dataset". An inspection of the webpage description revealed that the data was originally generated by Koklu and Ozkan (2020), for developing models to distinguish between seven dried bean varieties commonly produced in Turkey.

First, the dataset was downloaded from the URL (part of the repository website) to the working directory as a .zip file and subsequently unzipped. The unzipped folder was named `DryBeanDataset`.

```
uci_url <- 'https://archive.ics.uci.edu/ml/machine-learning-databases/00602/DryBeanDataset.zip'

download.file(uci_url, 'DryBeanDataset.zip')
```

```
unzip('DryBeanDataset.zip')
```

Then, the first sheet of the Excel file (`Dry_Bean_Dataset.xlsx`) within the folder was extracted from its location within the working directory. The other sheet in the file contains a citation for the paper by Koklu and Ozkan (2020) which, while useful for a bibliography, is irrelevant for data analysis.

```
# specify path
path <- paste0(getwd(), '/DryBeanDataset/Dry_Bean_Dataset.xlsx')
# read the first sheet (where the data is)
beans <- read_xlsx(path, sheet = 1)
```

Exploratory Analysis

Before models could be trained and evaluated, the dataset itself was examined to reveal its format, number of potential predictors, and other information required for efficient data manipulation. Both the UCI Machine Learning Repository page and the Koklu and Ozkan (2020) paper were used as references for information not immediately obvious from the dataset alone.

Overview

```
# overview of the data's size and format
str(beans)
```

```
## tibble [13,611 x 17] (S3: tbl_df/tbl/data.frame)
##  $ Area           : num [1:13611] 28395 28734 29380 30008 30140 ...
##  $ Perimeter      : num [1:13611] 610 638 624 646 620 ...
##  $ MajorAxisLength: num [1:13611] 208 201 213 211 202 ...
##  $ MinorAxisLength: num [1:13611] 174 183 176 183 190 ...
##  $ AspectRation   : num [1:13611] 1.2 1.1 1.21 1.15 1.06 ...
##  $ Eccentricity   : num [1:13611] 0.55 0.412 0.563 0.499 0.334 ...
##  $ ConvexArea     : num [1:13611] 28715 29172 29690 30724 30417 ...
##  $ EquivDiameter  : num [1:13611] 190 191 193 195 196 ...
##  $ Extent         : num [1:13611] 0.764 0.784 0.778 0.783 0.773 ...
##  $ Solidity       : num [1:13611] 0.989 0.985 0.99 0.977 0.991 ...
##  $ roundness      : num [1:13611] 0.958 0.887 0.948 0.904 0.985 ...
##  $ Compactness    : num [1:13611] 0.913 0.954 0.909 0.928 0.971 ...
##  $ ShapeFactor1   : num [1:13611] 0.00733 0.00698 0.00724 0.00702 0.0067 ...
##  $ ShapeFactor2   : num [1:13611] 0.00315 0.00356 0.00305 0.00321 0.00366 ...
##  $ ShapeFactor3   : num [1:13611] 0.834 0.91 0.826 0.862 0.942 ...
##  $ ShapeFactor4   : num [1:13611] 0.999 0.998 0.999 0.994 0.999 ...
##  $ Class          : chr [1:13611] "SEKER" "SEKER" "SEKER" "SEKER" ...
```

The dataset contains 13,611 entries, each representing a scanned bean. There are 12 metrics describing the geometric features of each bean, plus four 'shape factors' which are derived from an earlier machine learning article on rice grains (Pazoki et al., 2014). All metrics except for `Class` (the outcome variable for this classification problem) are numeric, with `Class` being of class character.

A quick glance at the magnitudes of each numeric metric revealed large differences in scale; normalization will be needed for plotting and model fitting.

Checking for NA Values

```
which(is.na(beans))
```

```
## integer(0)
```

As described on the UCI repository webpage, the dataset indeed contains no NA entries.

Distribution of Outcome Classes

```r
# checking the distribution of classes (bean types) in the dataset
beans %>% group_by(Class) %>%
  summarize(count = n())
```

Class	count
BARBUNYA	1322
BOMBAY	522
CALI	1630
DERMASON	3546
HOROZ	1928
SEKER	2027
SIRA	2636

Table 1: Number of instances among the seven bean types (classes).

The dataset has a rather uneven distribution of the 7 bean varieties. The most frequently found type is Dermason at 3546 entries, while the least common is Bombay at 522 entries. This uneven distribution must be accounted for during model fitting and accuracy assessments.

Distribution of Potential Predictor Variables Among Classes

Before building models, the distribution of values among different bean varieties were visualized with boxplots for all 16 possible predictor variables. This was done to understand the levels of variability between bean varieties that were present in each potential predictor, and select the most informative predictors for models that perform best with lower-dimensional data.

As mentioned earlier, normalization was performed to produce all the plots featured within this report. While multiple formulae for normalization exist, the one used in this report is as follows:

$$X_{normalized} = \frac{(X - min)}{(max - min)}$$

Where X is the value of a single instance within a metric, max is the maximum value of the metric, and min is the minimum value of the metric. This formula normalizes the values of each metric from 0 (minimum) to 1 (maximum).

To conserve space, only the first instance in a series of very similar code chunks are displayed in the PDF file. A complete collection of code can be seen in the R Markdown and R files.

Area Koklu and Ozkan (2020) used a computer vision system (camera connected to a computer loaded with image-processing software) to scan each bean within the dataset. In their paper, the metric of Area was defined as the number of pixels within each scanned bean image, and denoted by A.

```r
# separately mark outliers
outliers <-
  beans %>%
  # do the following for each Class
  group_by(Class) %>%
  # filter for outliers; whiskers are 1.5 times IQR, while boxes go from Q1 to Q3
  filter(Area > (quantile(Area, 0.75)) + 1.5 * IQR(Area) |
           Area < (quantile(Area, 0.25)) - 1.5 * IQR(Area))
```

```
beans %>%
  # plot with jittering outliers
  ggplot(aes(x = reorder(Class, Area), y = Area)) +
  # outliers plotted separately in geom_point()
  geom_boxplot(outlier.shape = NA) +
  geom_jitter(data = outliers) +
  xlab('Class') +
  ggtitle('Distribution of Areas')
```

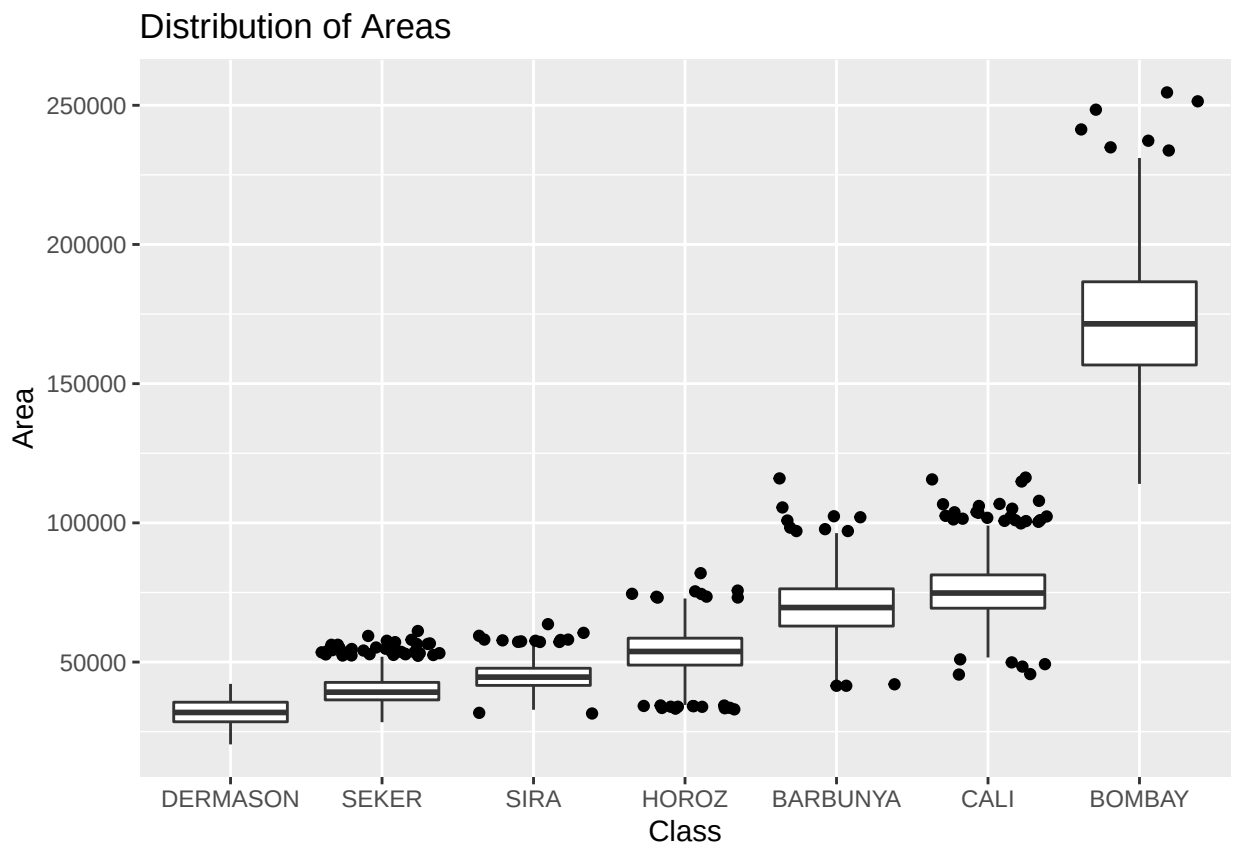

Figure 1: Normalized distribution of Areas among the seven bean varieties.

The area IQR for Bombay is distinctively higher than those of all other varieties. Heavily overlapping IQRs exist in Cali and Barbunya, as well as Sira and Seker. Dermason, Seker, and Sira all have much narrower IQRs compared to the other varieties, despite having very close distribution ranges. Most outliers are on the upper ends of their respective distributions.

Perimeter The metric of Perimeter (denoted by P) was defined as the length of each bean image's border (Koklu and Ozkan, 2020).

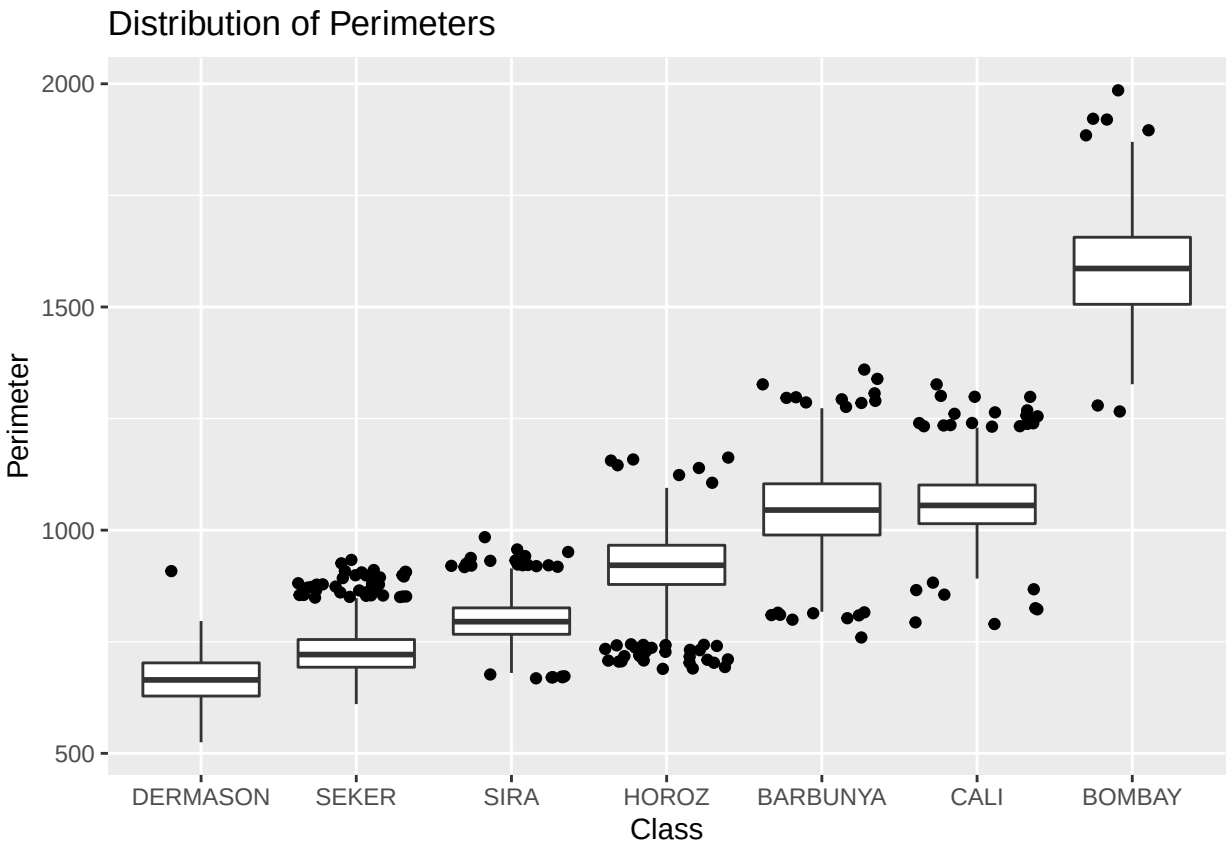

Figure 2: Normalized distribution of Perimeters among the seven bean varieties.

Similarly to the Area metric, Bombay has a distinctively higher range of Perimeter lengths than the other varieties. The IQR for Cali exists entirely within that of Barbunya, while the IQRs for Dermason and Seker overlap. The IQRs of perimeter lengths for smaller bean varieties are less tight than seen in the Area graph, but also somewhat more distinct relative to the scale of the y-axis.

Major Axis Length The Major Axis Length (L) was defined as the "distance between the ends of the longest line that can be drawn from a bean" (Koklu and Ozkan, 2020); in other words, the length of each bean.

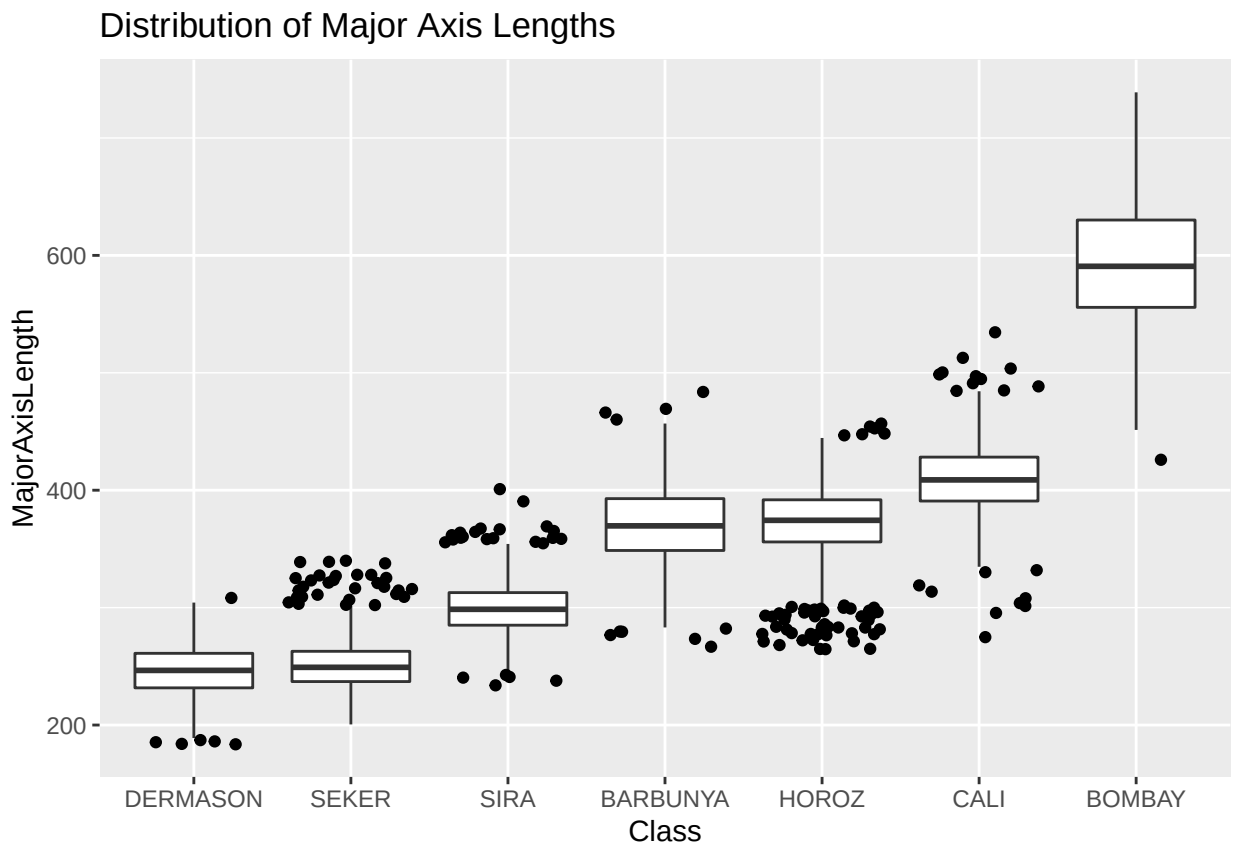

Figure 3: Normalized distribution of Major Axis Lengths among the seven bean varieties.

For Major Axis Length, Bombay once again has an IQR that was distinctively larger than all others. Barbunya/Horoz and Dermason/Seker have heavily overlapping IQRs, while Cali has a smaller IQR overlap with Horoz/Barbunya. Seker and Sira hav more outliers on the high end of their ranges, while Horoz has more on the low end.

Minor Axis Length The Minor Axis Length (l) was described as the "longest line that can be drawn from the bean while standing perpendicular to the main axis" (Koklu and Ozkan, 2020); it can be thought of as the width of each bean.

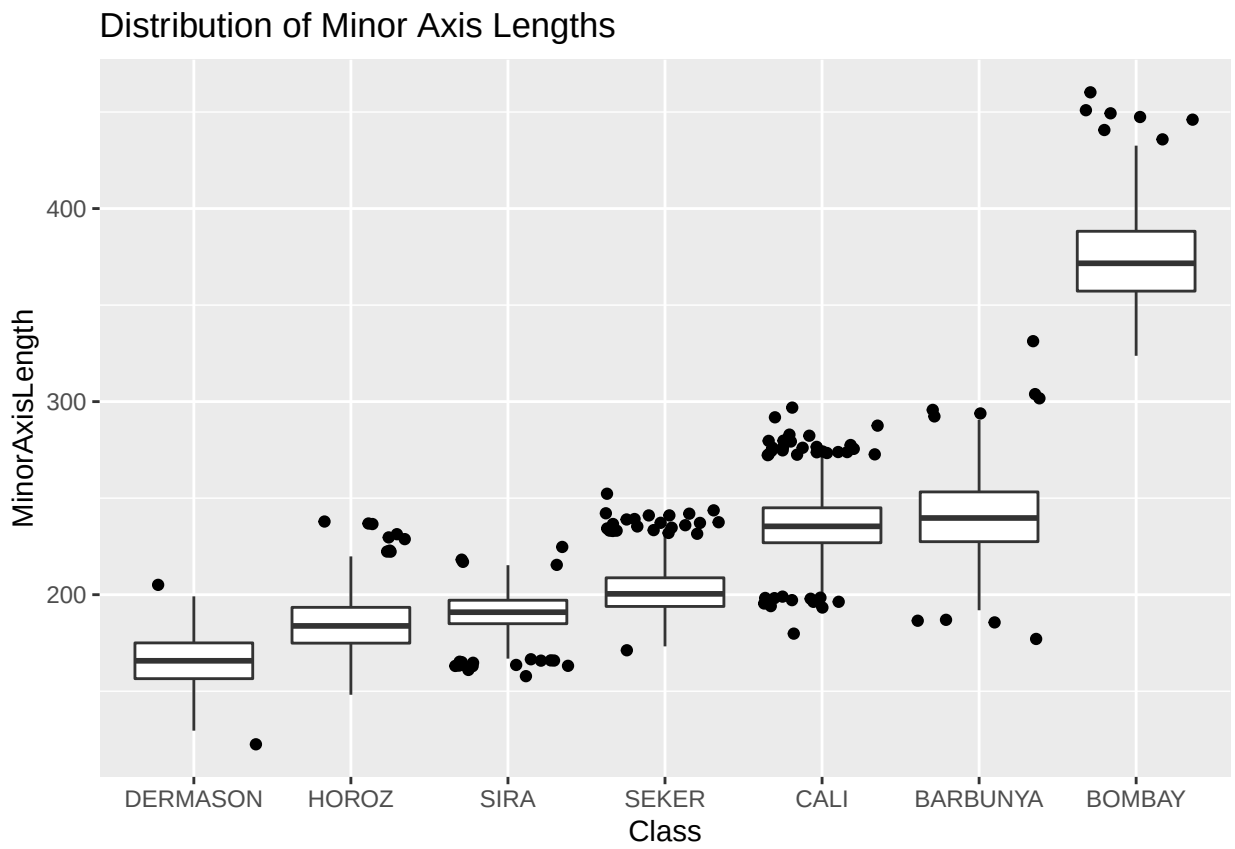

Figure 4: Normalized distribution of Minor Axis Lengths among the seven bean varieties.

The distribution of Minor Axis Lengths are similar to that of Major Axis lengths, but with a few differences: here the heaviest overlap is between Cali and Barbunya, followed by Horoz/Sira and Sira/Seker. Also, Dermason has a partial IQR overlap with Horoz rather than a near-complete one with Seker. Most outliers are on the high ends of their respective groups. Cali, Seker, and Horoz have more outliers on th high ends of their respective groups, while Sira has more on the low end.

Aspect Ratio The Aspect Ratio (K) was given in the original paper as the ratio between the Major and Minor Axis lengths (Koklu and Ozkan, 2020); in other words:

$$K = \frac{L}{l}$$

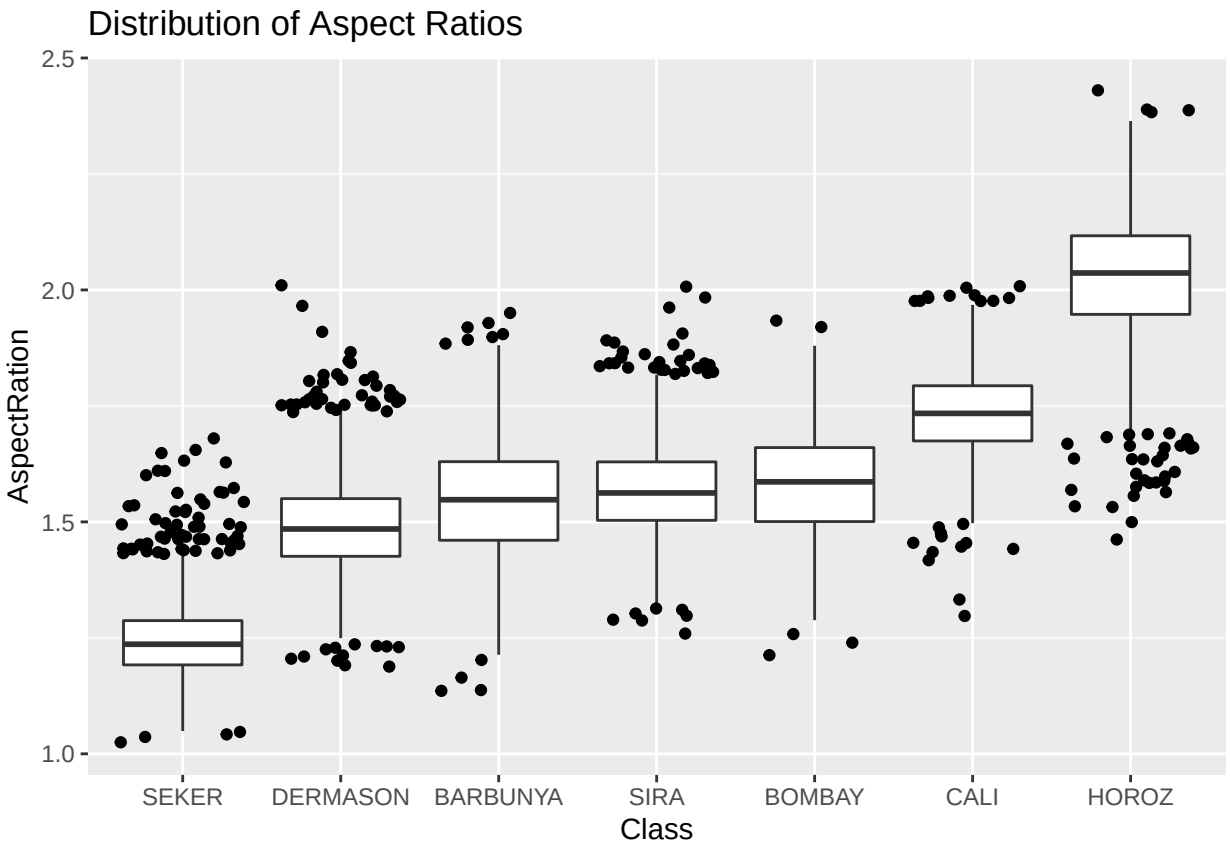

Figure 4: Normalized distribution of Aspect Ratios among the seven bean varieties.

The Aspect Ratio distributions differ from the previous potential predictors in that Bombay no longer has the most distinct distribution of values. Instead, Horoz and Cali have IQRs distinctively higher than others, while Seker has a lower IQR. The remaining varieties (Dermason, Barbunya, Sira, Bombay) all have heavily overlapping IQRs. Horoz has more outliers on the bottom end of its distribution, while Sira, Dermason, and Seker have more on their high ends.

Eccentricity Eccentricity is a measure describing the shape of aconic section. The eccentricity of each bean (Ec) was defined by Koklu and Ozkan (2020) as being the "eccentricity of an ellipse having the same moments as the [bean image] region".

The eccentricity of an ellipse is in turn a ratio of two values: the distance to any point on the ellipse from the focus, and the distance to that same point on the ellipse from the directrix.

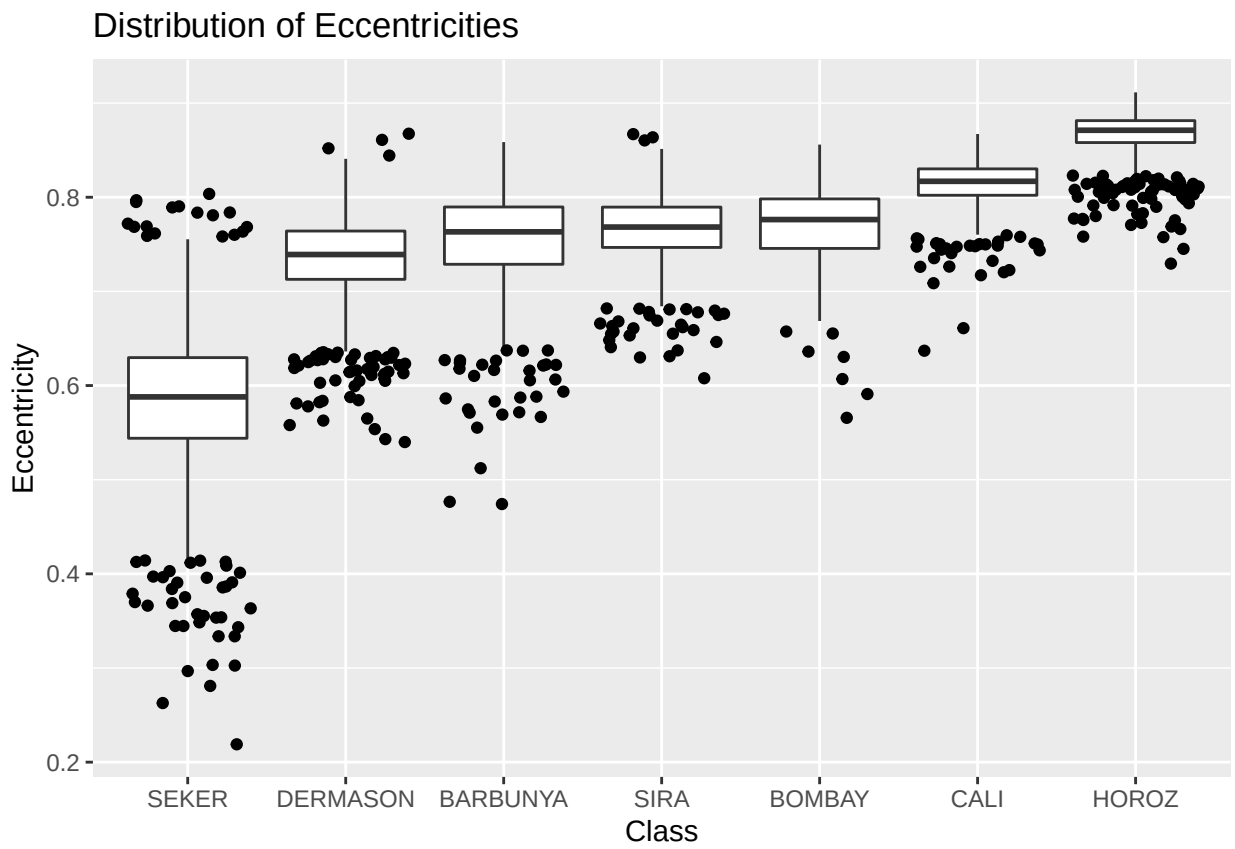

Figure 5: Normalized distribution of Eccentricities among the seven bean varieties.

The Eccentricity distributions are similar to that of Aspect Ratios, but with the two varieties having higher IQRs (Horoz, Cali) deviating from the others less while the one with lower IQR (Seker) deviating more. Most outliers are on the low ends of their respective ranges.

Convex Area Koklu and Ozkan (2020) defined the Convex Area (C) of a bean as the "number of pixels in the smallest convex polygon that can contain the area of a bean seed".

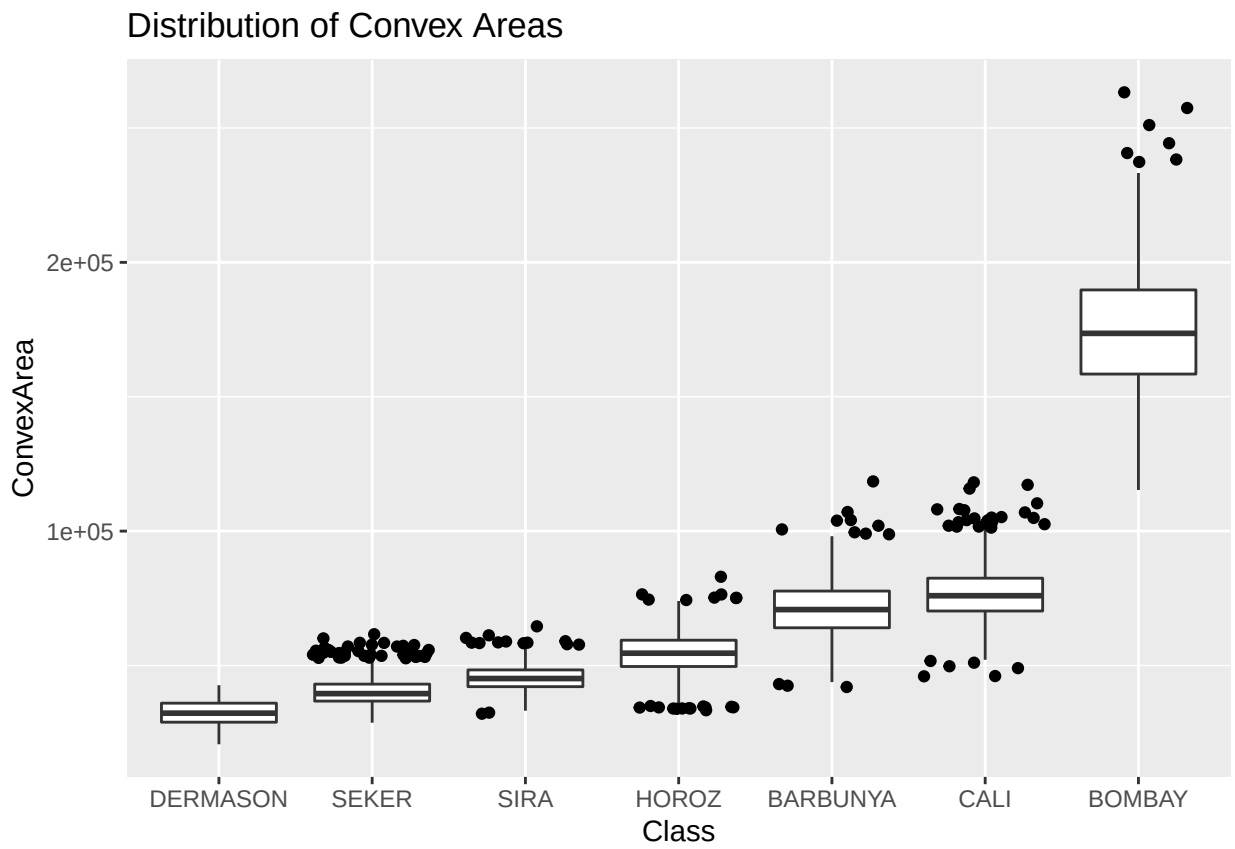

Figure 6: Normalized distribution of Convex Areas among the seven bean varieties.

With the Covex Area Sizes, a distribution somewhat similar to the Area, Perimeter, and Axis Length distributions could be seen again. Bombay has a distinctively high IQR of Convex Areas, while Dermason has the lowest range. Cali and Barbunya have heavily overlapping IQRs, with less overlap existing among the other varieties. The narrowest IQRs belong to Seker and Sira. Outliers are biased towards the top end for Bombay, Cali, Barbunya, Sira, and Seker.

Equivalent Diameter Koklu and Ozkan (2020) defined the Equivalent Diameter (Ed) as "the diameter of a circle having the same area as a bean seed area".

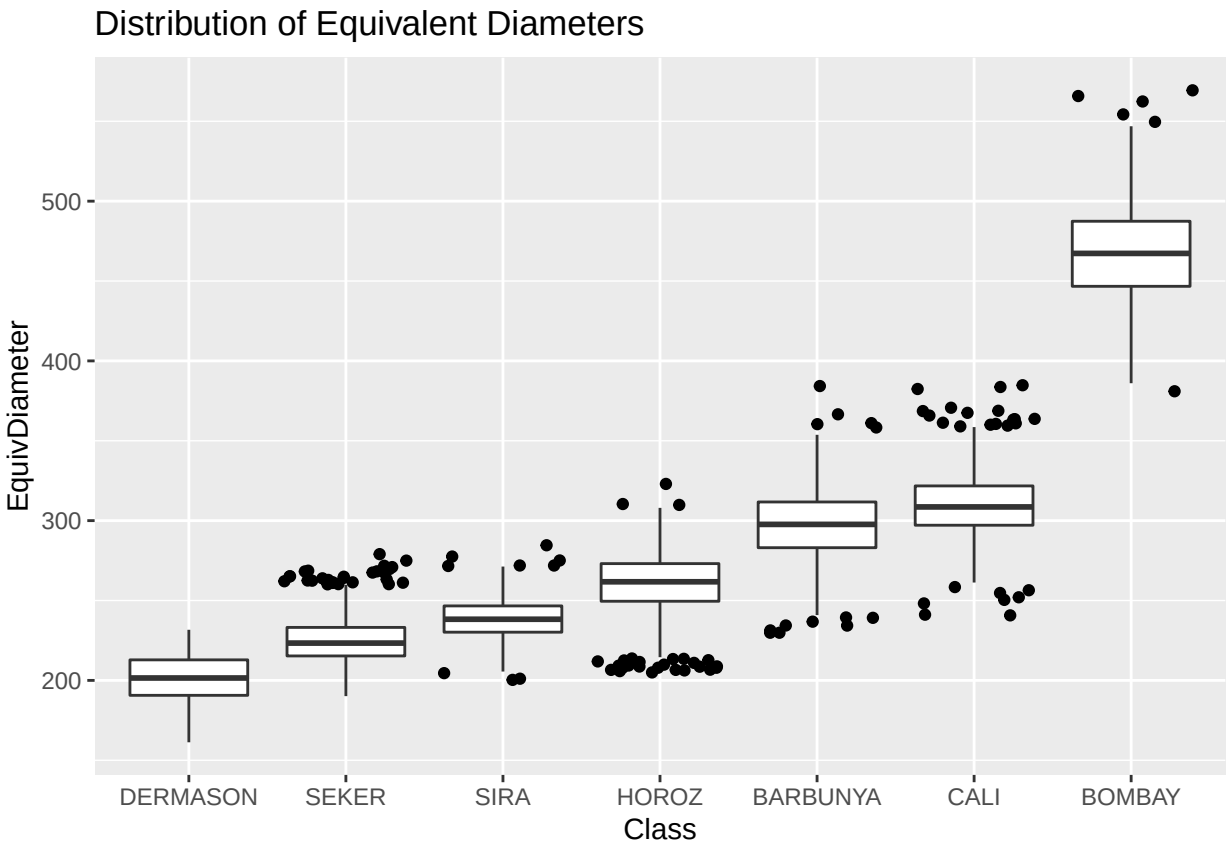

Figure 7: Normalized distribution of Equivalent Diameters among the seven bean varieties.

The distributions of Equivalent Diameters are fairly similar to that of Convex Areas. Once again, Cali and Barbunya have heavy overlaps, with Bombay having a distinctly higher IQR. Cali and Seker have more outliers on the top ends of their distributions, while Horoz has more on the bottom end.

Extent The Extent (Ex) of a bean is a metric for size, being defined as "ratio of the pixels in the bounding box to the bean area" (Koklu and Ozkan, 2020). What constituted a "bounding box" was not specified in the paper, but given its mathematical definition it was presumably a rectangular background containing all points within the bean image.

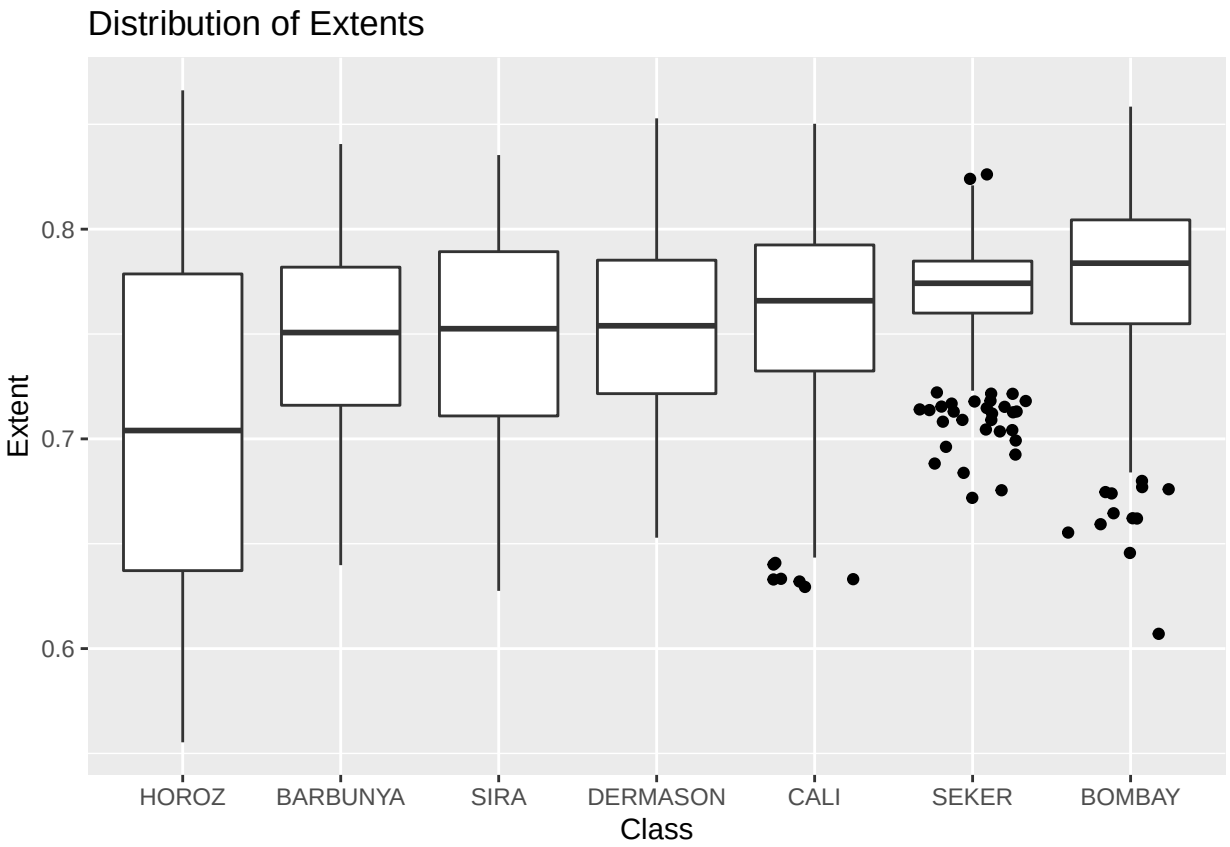

Figure 8: Normalized distribution of Extents among the seven bean varieties.

The distributions of Extent Ratios exhibit heavy IQR overlaps between all bean varieties. Among the three groups with outliers (Cali, Seker, Bombay), all have more on the bottom ends of their respective distributions.

Solidity Solidity (S) is another area-related metric, being the "ratio of the pixels in the convex shell to those found in beans" (Koklu and Ozkan, 2020). Note that "convex shell" does not appear to be a commonly used mathematical term; however, a similar term named "convex hull" exists and the authors presumably meant the latter.

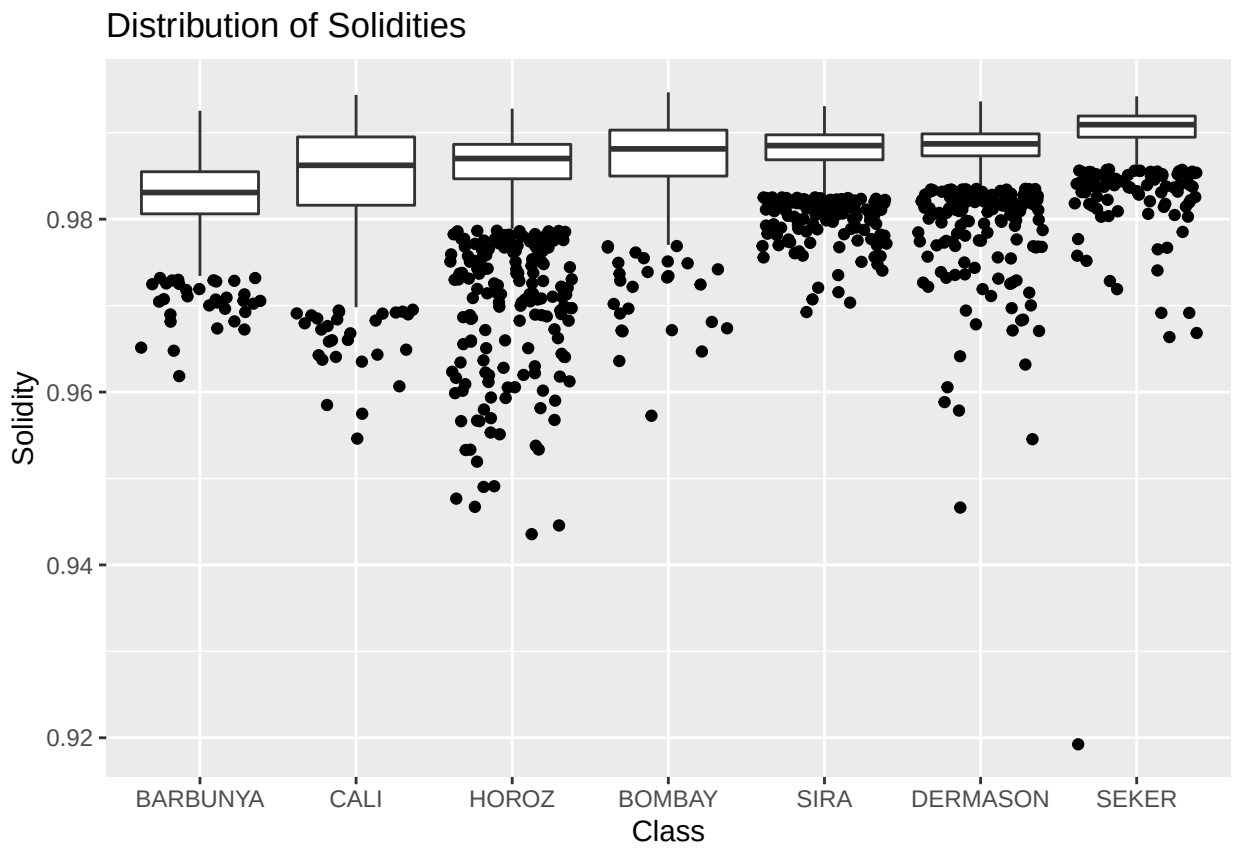

Figure 9: Normalized distribution of Solidities among the seven bean varieties.

The distributions of Solidity values exhibit heavy IQR overlaps between nearly all bean varieties. Also, every bean variety has many outliers (values outside the boxplot whiskers) on the lower end of their respective distributions.

Roundness Koklu and Ozkan (2020) gave the following formula for Roundness (R):

$$R = \frac{4\pi A}{P^2}$$

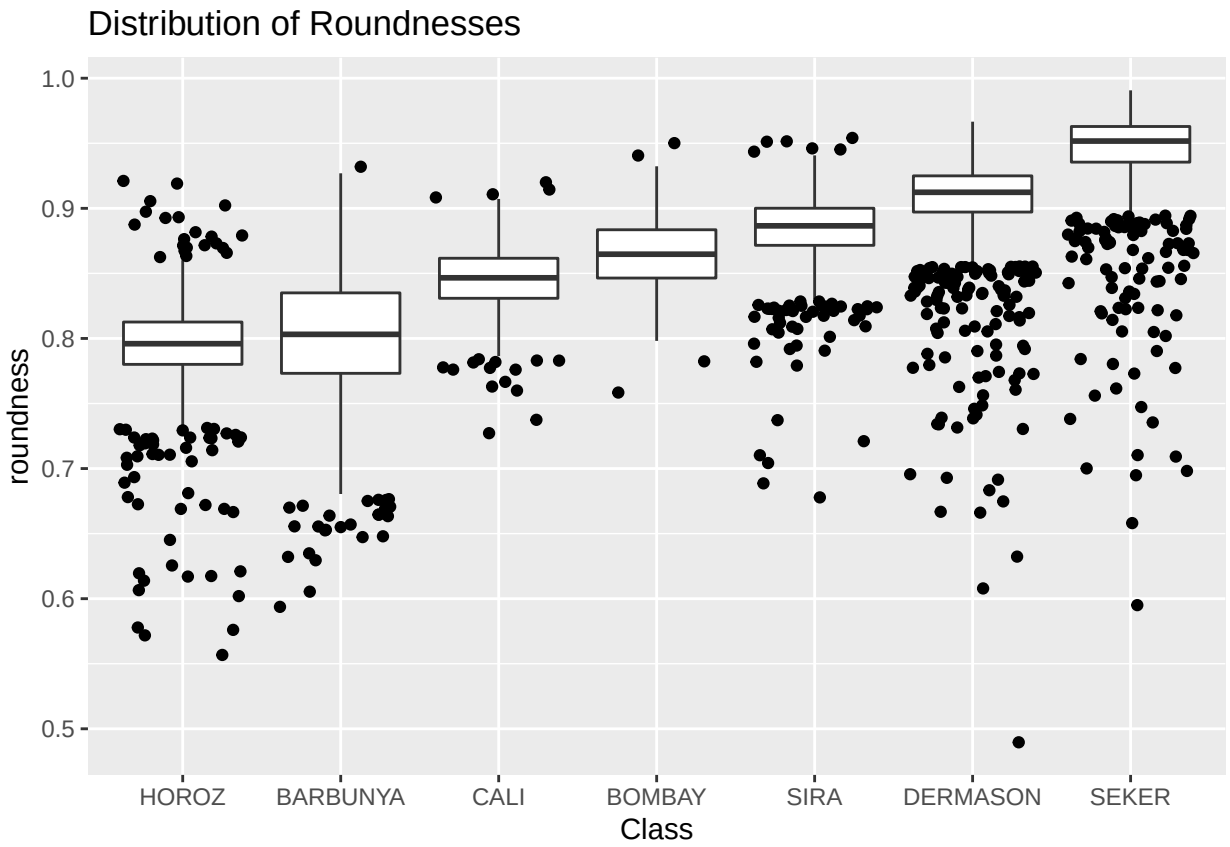

Figure 10: Normalized distribution of Roundness values among the seven bean varieties.

The ranking of median Roundness values are similar to that of that of Solidity, but with the lowest three (Cali, Barnunya, Horoz) switched. Less overlaps exist than in Solidity (except for heavy overlap Horoz and Barbunya), but many outliers outside the whiskers exist for all varieties, and are biased towards the low end of the range for all groups except Bombay.

Compactness Compactness (CO) is another metric for bean roundness, and was given by Koklu and Ozkan (2020) as:

$$CO = \frac{Ed}{L}$$

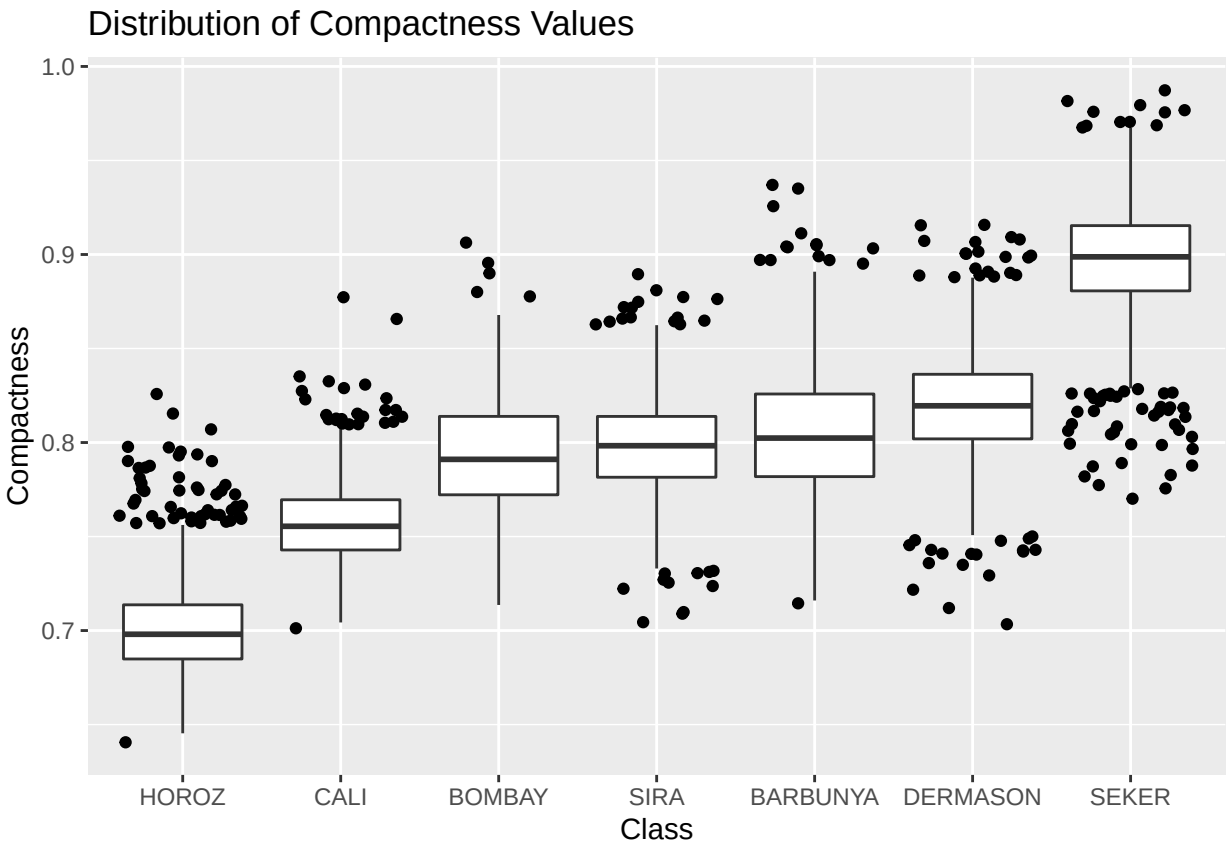

Figure 11: Normalized distribution of Compactness values among the seven bean varieties.

The distributions of Compactness values have heavy overlaps between Bombay, Sira, Barbunya, and Dermason. Seker has the highest range of Compactness, while Horoz has the lowest. All varieties have many outliers, mostly on the high ends of the distributions. Seker has most of its outliers on the low end of its range; the opposite was true for Bombay, Cali, and Horoz.

Shape Factor 1 In addition to the metrics described above, Koklu and Ozkan (2020) also made use of four "Shape Factors", from another machine learning article by Pazoki et al. (2014).

Shape Factor 1 ($SF1$) was defined as follows:

$$SF1 = \frac{L}{A}$$

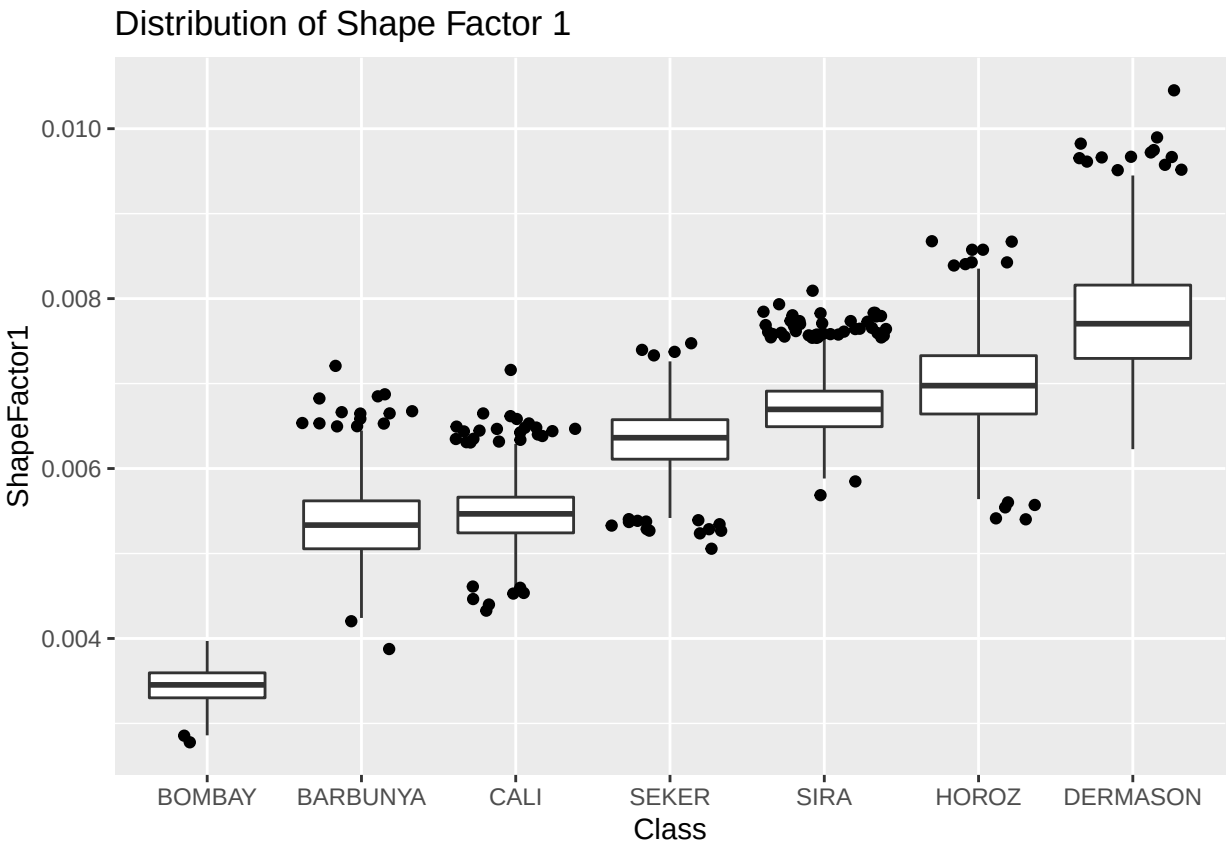

Figure 12: Normalized distribution of SF1 values among the seven bean varieties.

The distributions of Shape Factor 1 values feature heavy overlap between Barbunya/Cali, and relatively less overlap between Seker/Sira/Horoz. Dermason has the highest range of Shape Factor 1, and Bombay the lowest. All groups have more outliers on the top ends of their ranges except for Seker (more on the low end) and Bombay (two outliers on the low end only).

Shape Factor 2 Shape Factor 2 ($SF2$) was derived with the formula:

$$SF2 = \frac{l}{A}$$

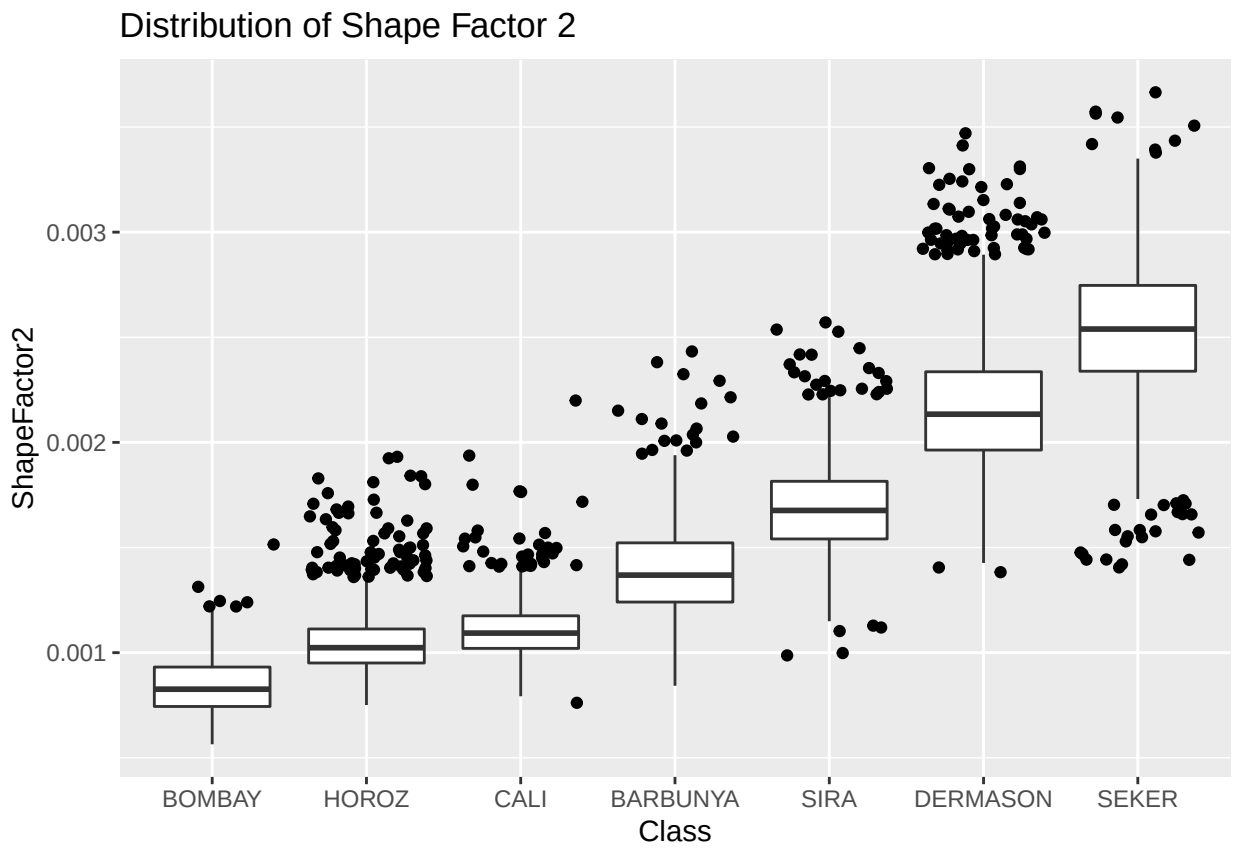

Figure 13: Normalized distribution of SF2 values among the seven bean varieties.

The distributions of Shape Factor 2 values have relatively less IQR overlap compared to most other variables examined, with the 4 highest Shape Factor 2 ranges (Seker, Dermason, Sira, Barbunya) all having non-overlapping IRRs. Horoz and Cali have a large IQR overlap, but neither has overlapping IQRs with Bombay. However, outliers on the high ends of the ranges are common. With the exception of Seker, all groups have more outliers on the top ends of their ranges.

Shape Factor 3 Shape Factor 3 ($SF3$) was derived from the formula:

$$SF3 = \frac{A}{(\frac{L}{2})^2\pi}$$

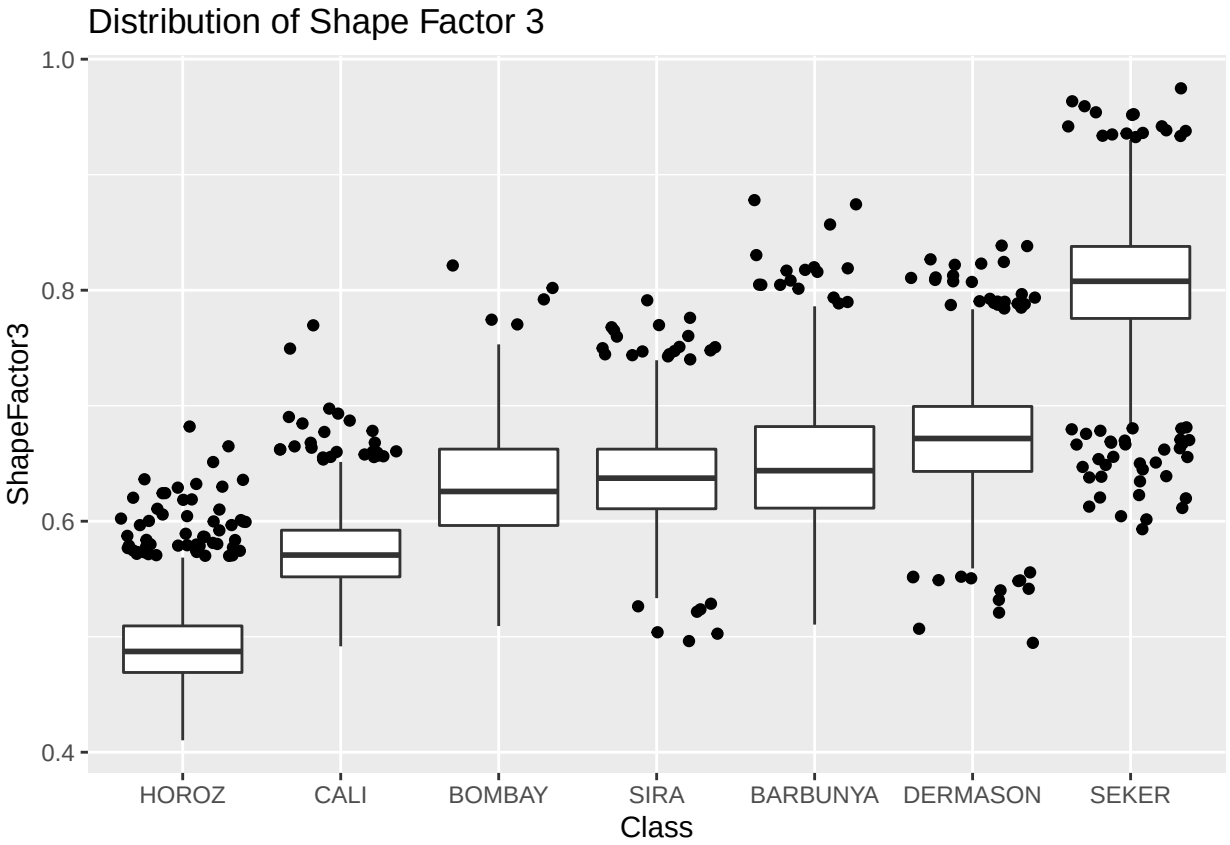

Figure 14: Normalized distribution of SF3 values among the seven bean varieties.

The distributions of Shape Factor 3 values have heavy overlaps in Bombay, Sira, Barbunya, and Dermason. Seker had a distinctively higher IQR than the other varieties but also many low outliers; the opposite was true for Horoz. Except for Seker, all groups have more outliers on the low ends of their ranges.

Shape Factor 4 Shape Factor 4 ($SF4$) was derived from the formula:

$$SF3 = \frac{A}{(\frac{L}{2})(\frac{l}{2})\pi}$$

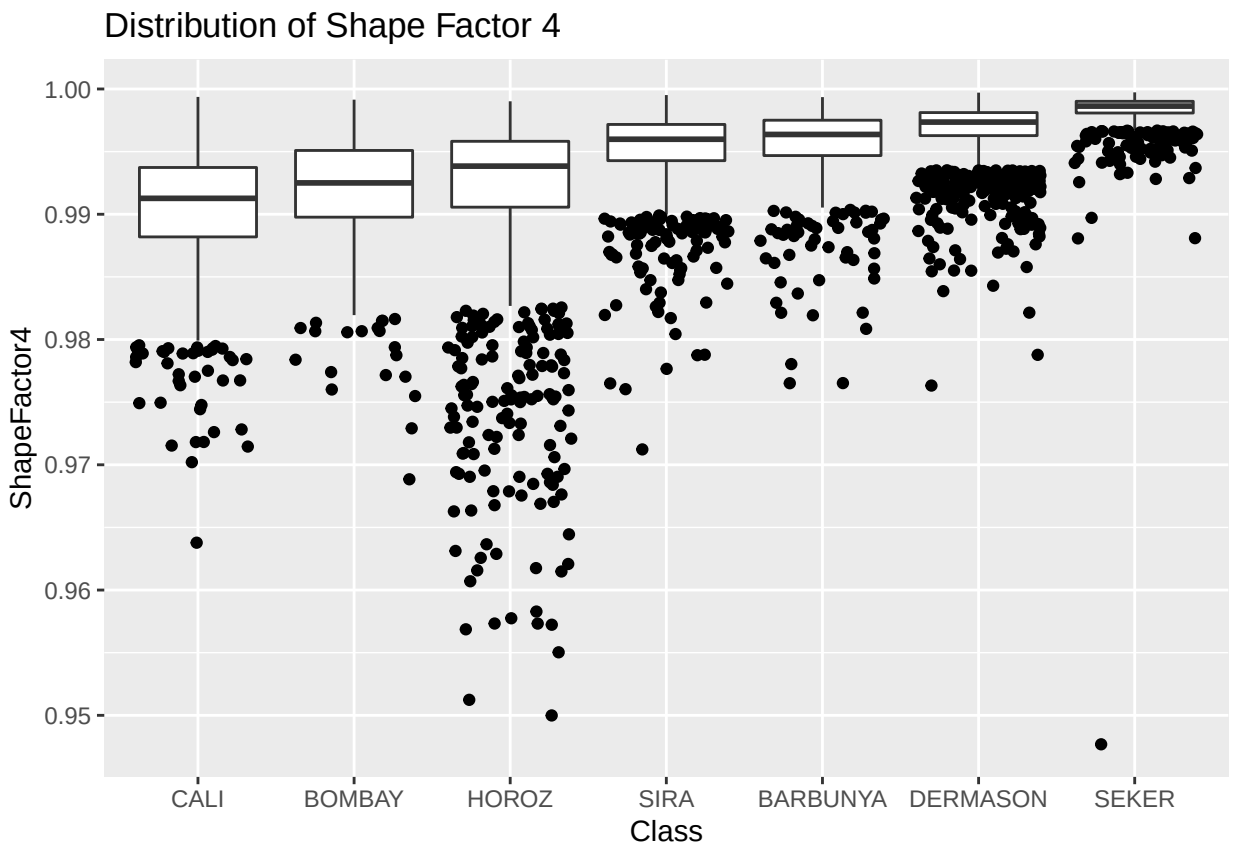

Figure 15: Normalized distribution of SF4 values among the seven bean varieties.

The distributions for Shape Factor 4 exhibit heavy overlaps between nearly all varieties, with the possible exception of Seker. In addition, every variety has many outliers on the low ends of their respective distributions. All groups have outliers on the low ends of their distributions; Horoz has an exceptionally long range of outlier values.

Combining Predictors for Scatterplots

Some common classification models, such as KNN, do not perform efficiently with high-dimensional data due to a phenomenon known as the Curse of Dimensionality, where an increase in the amount of predictors (dimensions) raises the possibility of overfitting the data in addition to making the data points sparser in space, requiring less flexible models to maintain the same data coverage. As such, it would be constructive to select a pair of "compatible" predictor variables for those models. The goal is to have each variable contain inter-group (between bean varieties) variances that the other does not have sufficient amounts of. Also, the variables should be independent, lacking in multicollinearity of measurements. For instance, if one variable in the pair is already derived from bean lengths, the other variable should not also contain bean lengths in its formula. In addition, each of the two variables should have as much distance (or failing that, the least amount of overlap) between its group IQRs as possible, with the least amount of outliers and skew. Some variables (like Area and Convex Area) exist over a far larger numerical scale than others; this could be solved through normalizing all variables, making relative distances along the y-axis (as seen on the boxplots) more relevant than raw numerical differences.

To identify promising combinations, the variables were roughly grouped into six categories, reflecting broad features in their boxplots:

- Group 0 ("don't bother"): Variables whose boxplots display heavily overlapping IQRs and outliers that are visibly more numerous (compared to the other variables in this dataset) and/or heavily skewed in distributions. Extent, Solidity, and Shape Factor 4 were placed under this category.

- Group 1 (“Bombay high, Dermason low”): Variables where Bombay have the highest IQR (usually by a relatively large margin), and Dermason the lowest. Other bean varieties tend to have less separate or even overlapping IQRs. Area, Perimeter, Major Axis, Minor Axis, Convex Area, and Equivalent Diameter all fell under this category.
- Group 2 (“Horoz high, Seker low”): Variables where Horoz has the highest IQR, and Seker the lowest. Again, other groups tend to have less separate or overlapping IQRs. Aspect Ratio and Eccentricity fell under this cetegory.
- Group 3 (“Seker high, Horoz low”): The opposite of group 2. Roundness and Compactness fell under this category. Note that as described in the original paper, Roundness and Compactness are both measures of how round a bean is, albeit calculated by different formulae. Both are also, by definition, the opposite metric of Aspect Ratio (which measures how elongated a bean is through the ratio between length/Major Axis Length and width/Minor Axis Width).
- Group 4 (“Dermason high, Bombay low”): The opposite of group 1, at least in terms of the defining feature (arbitrarily assigned for quick grouping). Shape Factor 1 was the sole variable assigned to this category.
- Group 5 (“Seker high, Bombay low”): Shape Factor 2 was the only variable assigned under this group.

To conserve memory, Group 0 variables were removed via the following code:

```
# trim off group 0 ("don't bother") variables, which will not be considered further
beans <-
  subset(beans, select = -c(Extent, Solidity, ShapeFactor4))
head(beans, 3)
```

Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Roundness	Compactness	ShapeFactor1	ShapeFactor2	ShapeFactor3	Class
28395	610.291	208.1781	173.8887	1.197191	0.549812	28715	190.141	0.958027	0.913357	0.007331	0.003147	0.834222	SEKER
28734	638.018	200.5248	182.7344	1.097357	0.411785	29172	191.272	0.887034	0.953861	0.006978	0.003563	0.909850	SEKER
29380	624.110	212.8261	175.9311	1.209713	0.562727	29690	193.410	0.947849	0.908774	0.007243	0.003047	0.825870	SEKER

Among group 1 variables, Perimeter and Equivalent Diameter appeared to be the most suitable choices, as they both have relatively less overlaps between the “middle groups” representing varieties other than Bombay and Dermason. Perimeter has large overlaps in Cali/Barbunya and Seker/Dermason, while Equivalent Diameter has large overlaps in Cali/Barbunya and Sira/Seker. Observing the boxplots, Aspect Ratio, Compactness, Shape Factor 2, and Shape Factor 3 were selected as the best complements due to having higher variances lacking in the group 1 choices, in addition to relatively less numerous and skewed outliers. Equivalent Diameter is incompatible with Compactness (Equivalent Diameter divided by Major Axis length), Shape Factor 2 (Minimum Axis length divided by Area), and Shape Factor 3 (which is derived from Area and Major Axis length). Meanwhile, Perimeter is compatible with Aspect Ratio, Compactness, Shape Factor 2, and Shape Factor 3. Included below are five scatterplots to further visualize the variables’ ability to model bean varieties as distinct clusters of data points.

```
beans %>%
  # normalize variables: get the difference of each value from the minimum, then divide by the range
  # this process will also be used to train and test the algorithms
  mutate(perim_tf = (Perimeter - min(Perimeter)) / (max(Perimeter) - min(Perimeter)),
         ar_tf = (AspectRation - min(AspectRation)) / (max(AspectRation) - min(AspectRation))) %>%
  # plot with coloring by bean variety (Class)
  ggplot(aes(x = perim_tf, y = ar_tf, color = Class)) +
  geom_point(alpha = 0.6) +
  # using the viridis color scale for color-based accessibility
  scale_color_viridis(discrete = T) +
  xlab('Perimeter (Normalized)') +
  ylab('Aspect Ratio (Normalized)') +
  ggtitle('Normalized Perimeter and Aspect Ratio Combinations on a Cartesian Plane')
```

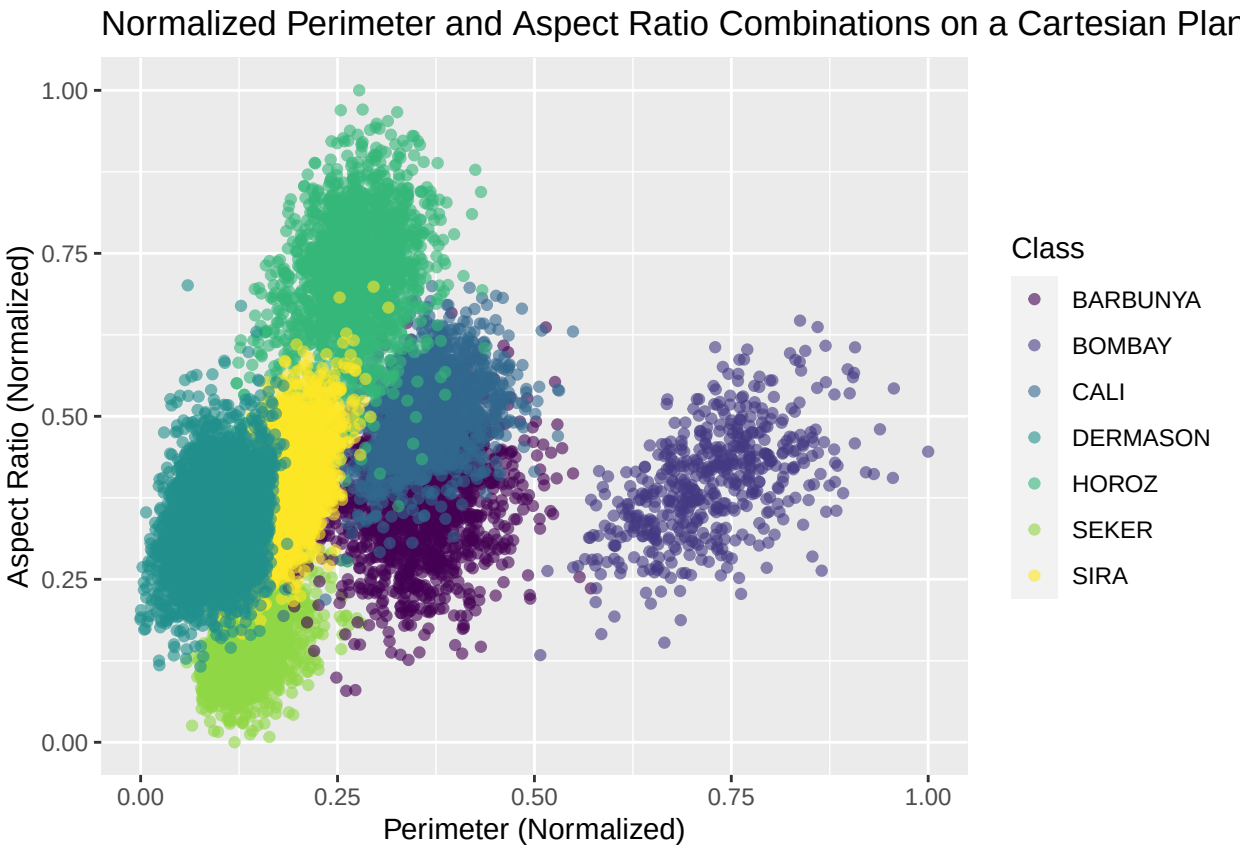

Figure 16: Scatterplot of Normalized Perimeter and Aspect Ratio values, color-coded by bean varieties.

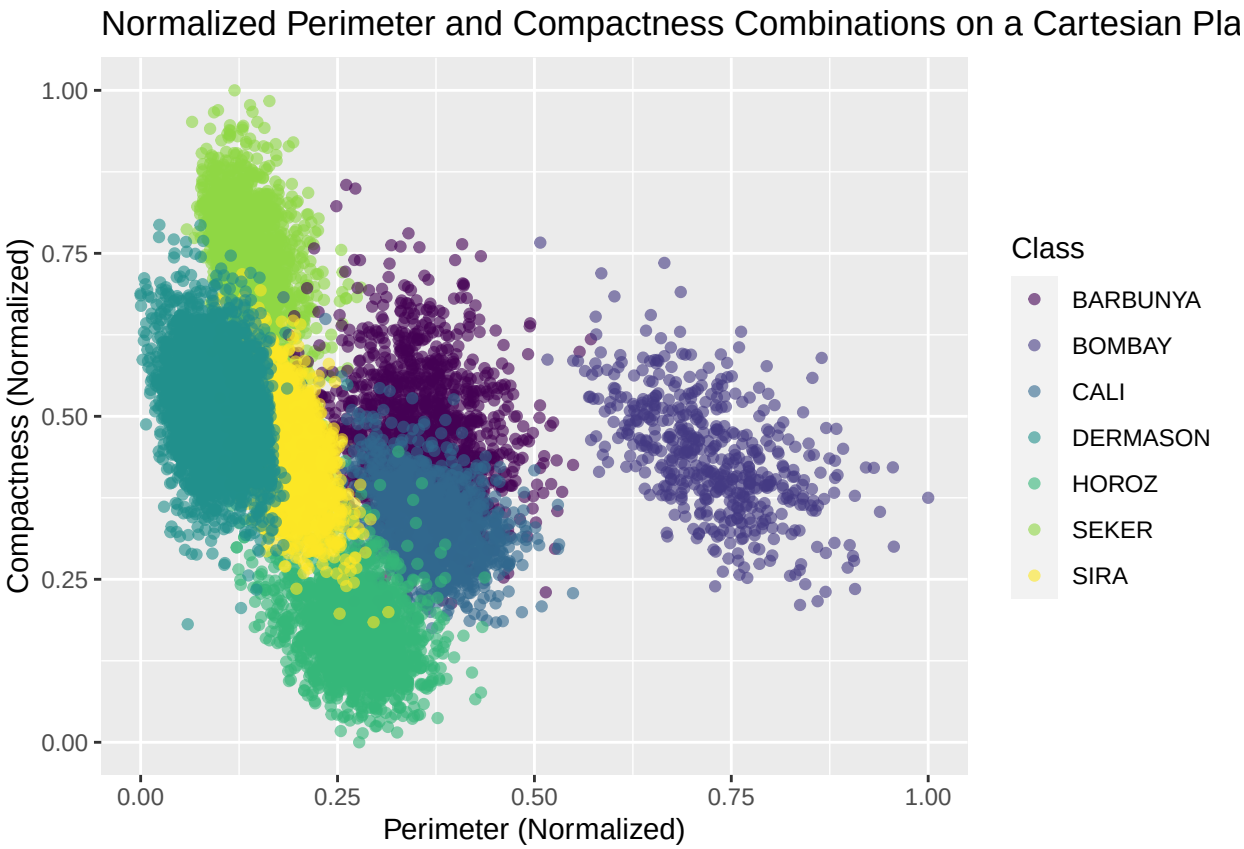

Figure 17: Scatterplot of Normalized Perimeter and Compactness values, color-coded by bean varieties.

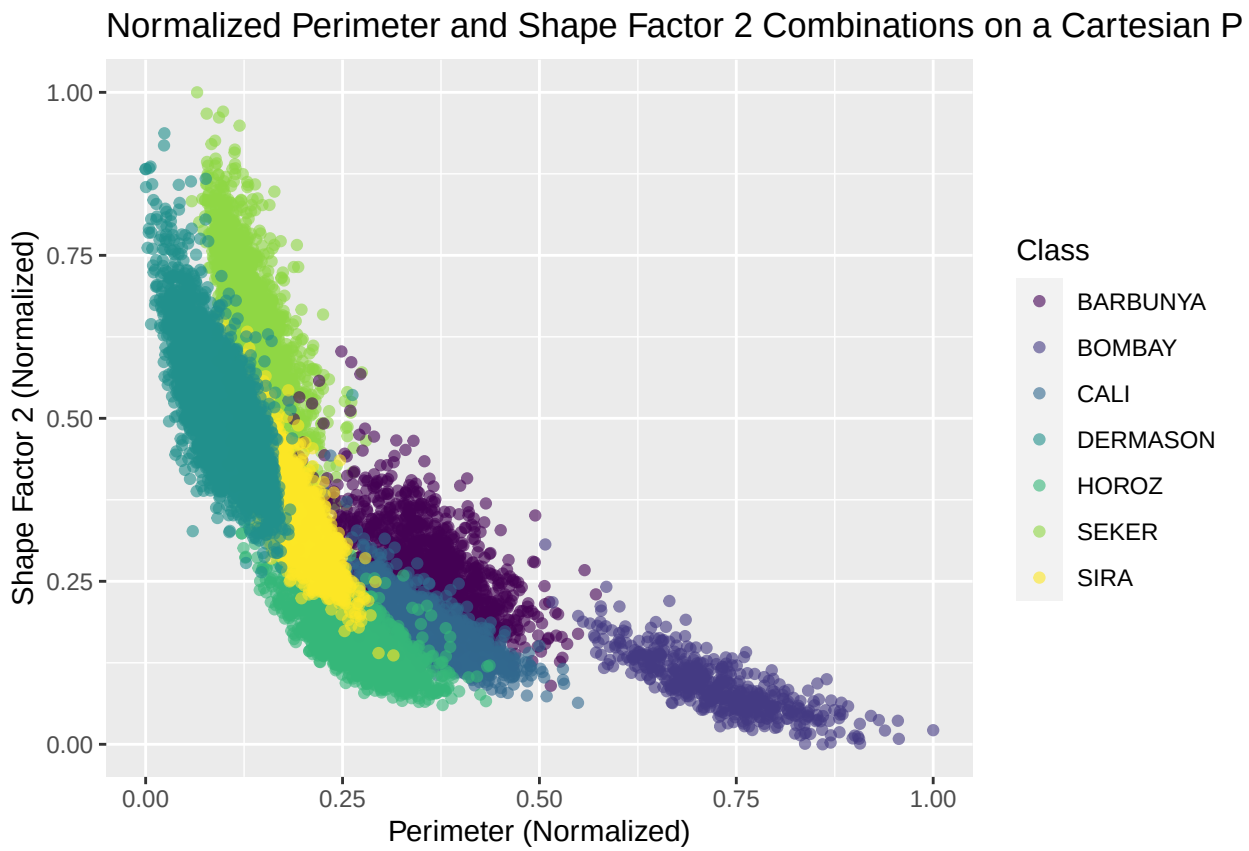

Figure 18: Scatterplot of Normalized Perimeter and Shape Factor 2 values, color-coded by bean varieties.

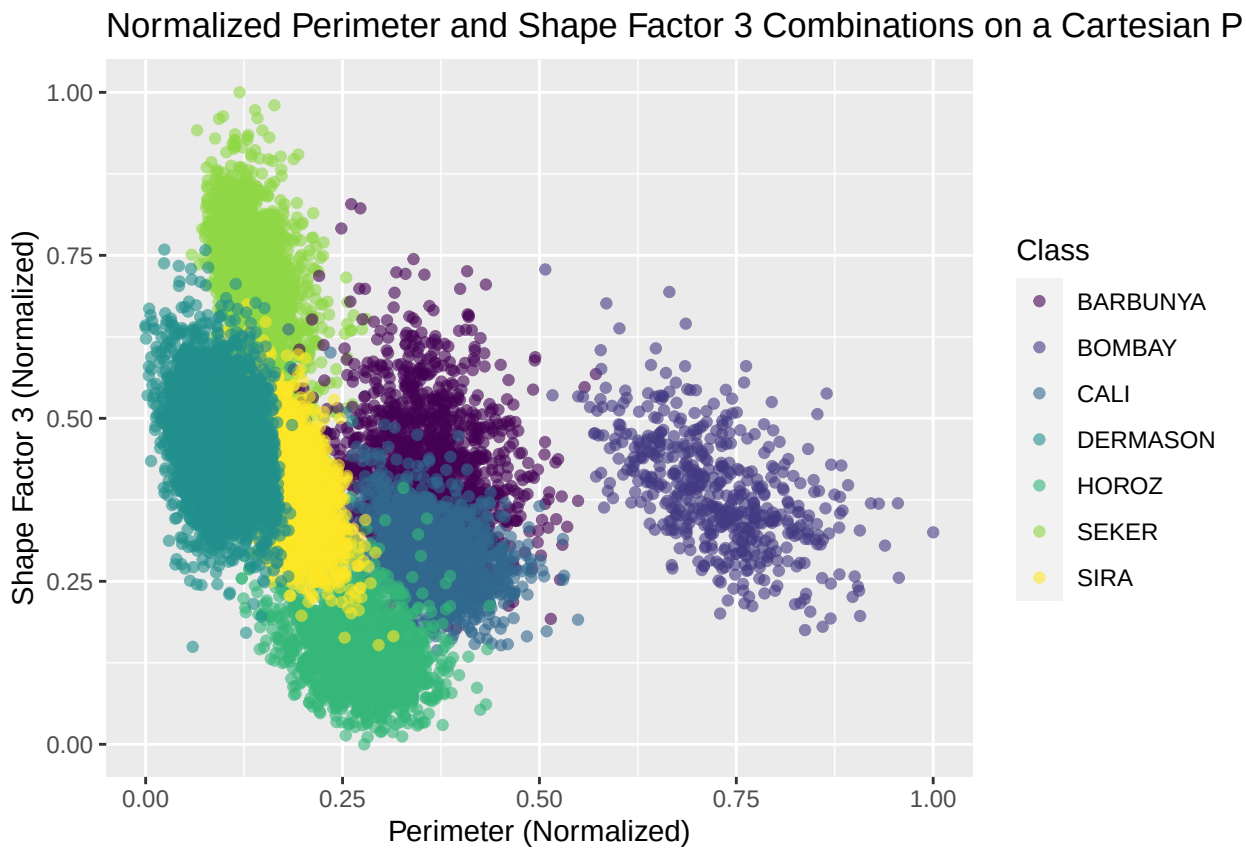

Figure 19: Scatterplot of Normalized Perimeter and Shape Factor 3 values, color-coded by bean varieties.

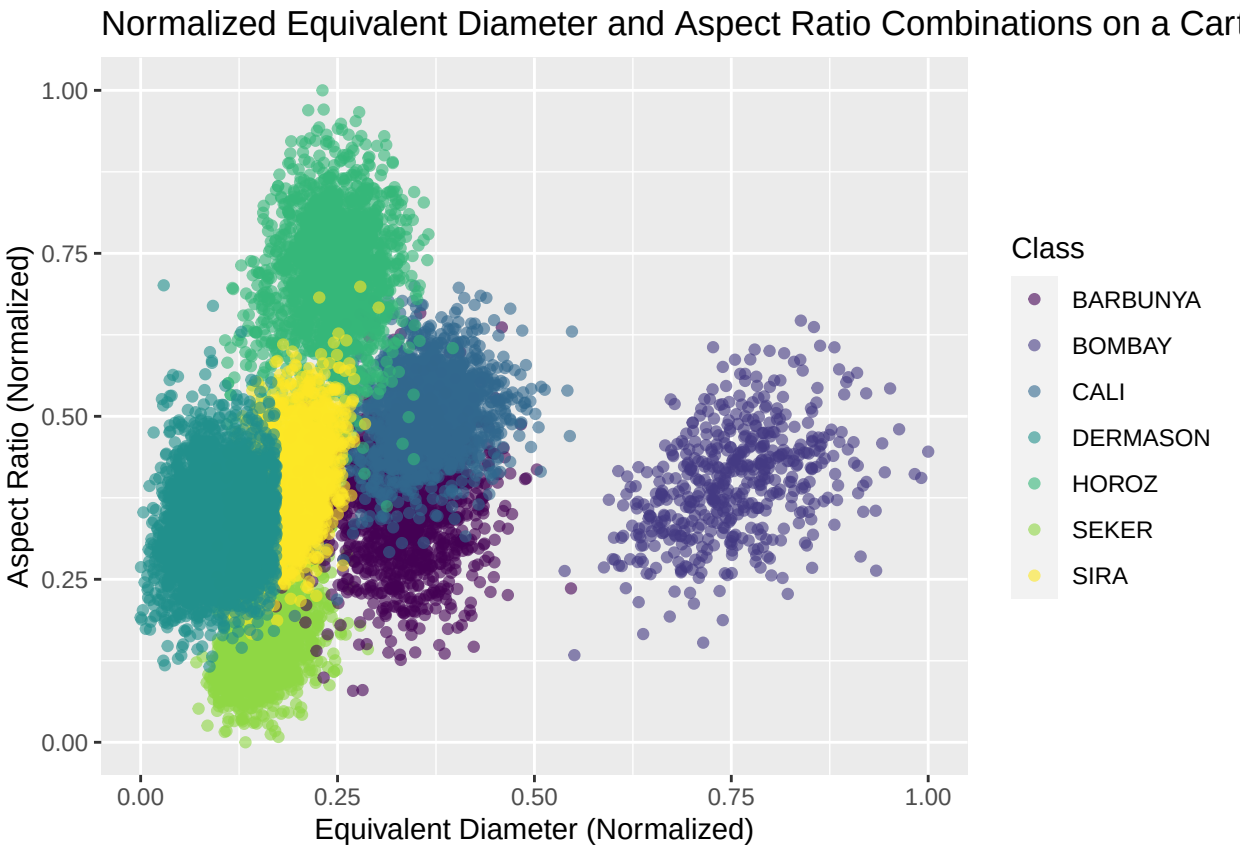

Figure 20: Scatterplot of Normalized Equivalent Diameter and Aspect Ratio values, color-coded by bean varieties.

In all 5 scatterplots, adding another variable improved separation of the bean Class clusters that were poorly resolved in the group 1 variables (Barbunya/Cali, Sira/Seker, Seker/Dermason). However, large overlaps could still observed, especially in Dermason/Sira and Barbunya/Cali. Although the separation between Barbunya and Cali had improved from the near complete overlap in Perimeter and Equivalent Diameter boxplots, the scatterplots still revealed that a sizeable portion of the Barbunya cluster had been covered by the Cali cluster. Cluster visualization would be useful for further determination of which variable combination had the least amounts of overlap, but such a procedure would involve visualizing already tuned clustering algorithms and defeat the purpose of exploratory analysis. As a result, all five variable combinations were chosen for model building.

Methods

With the insights gained from exploratory analyses and the original paper, the following steps were taken to prepare then analyze the data:

Data Formatting

Before training any algorithms, the data was first split into two parts: a set containing all the potential predictors (the other 15 variables besides `Class`) named `beans_x`, and a set containing the bean varieties (`Class`) only named `beans_y`. This was done to comply with the argument format of `caret::train`, the function used to train all models in this project.

```
beans_x <- subset(beans, select = -c(Class))
beans_y <- subset(beans, select = c(Class))
head(beans_y, 3)
```

Class
SEKER
SEKER
SEKER

Table 2: A very small sample of the beans_y dataset, as proof that the correct subset was selected.

While `caret::train` is capable of processing predictor data in the dataframe format using the formula `y ~ x1 + x2 + ...`, attempts to train algorithms using this input method resulted in repeated errors. As such, an alternative input method was chosen with the predictor entered in matrix format; `beans_x` was converted into matrix form to enable this decision.

```
beans_x <- as.matrix(beans_x)
head(beans_x, 3)
```

```
##       Area Perimeter MajorAxisLength MinorAxisLength AspectRation Eccentricity
## [1,] 28395   610.291        208.1781        173.8887     1.197191    0.5498122
## [2,] 28734   638.018        200.5248        182.7344     1.097356    0.4117853
## [3,] 29380   624.110        212.8261        175.9311     1.209713    0.5627273
##      ConvexArea EquivDiameter roundness Compactness ShapeFactor1 ShapeFactor2
## [1,]      28715      190.1411 0.9580271   0.9133578  0.007331506  0.003147289
## [2,]      29172      191.2728 0.8870336   0.9538608  0.006978659  0.003563624
## [3,]      29690      193.4109 0.9478495   0.9087742  0.007243912  0.003047733
##      ShapeFactor3
## [1,]    0.8342224
## [2,]    0.9098505
## [3,]    0.8258706
```

As previously done in the scatterplots (see Introduction section), the predictor variables must be normalized for algorithms such as KNN to work effectively. Since the variables vary greatly in scale, using them unaltered would lead to biased models that favor variables that exist over larger scales, regardless of their actual importance in relation to the outcome variable.

```
# for every column, apply the range formula onto each cell value:
# new value = (value - column min) / (column max - column min)
max_predictors <- apply(beans_x, 2, max) # column maxima
min_predictors <- apply(beans_x, 2, min) # column minima
colranges <- max_predictors - min_predictors # column ranges
beans_x <- sweep(beans_x, 2, STAT = min_predictors, FUN = '-') # subtract minima
beans_x <- sweep(beans_x, 2, STAT = colranges, FUN = '/') # divide by col ranges
# x is each matrix cell value, y is each entry in colranges
head(beans_x, 3)
```

```
##           Area  Perimeter MajorAxisLength MinorAxisLength AspectRation
## [1,] 0.03405267 0.05857388      0.04426214       0.1521417   0.12261211
## [2,] 0.03550018 0.07755673      0.03047881       0.1783367   0.05157739
```

```
## [3,] 0.03825855 0.06803484      0.05263303       0.1581899   0.13152124
##      Eccentricity ConvexArea EquivDiameter roundness Compactness ShapeFactor1
## [1,]    0.4777970 0.03310701    0.07080413 0.9348226   0.7867331    0.5934316
## [2,]    0.2784720 0.03499095    0.07357691 0.7931380   0.9035489    0.5474470
## [3,]    0.4964478 0.03712636    0.07881580 0.9145106   0.7735137    0.5820159
##      ShapeFactor2 ShapeFactor3
## [1,]    0.8330488    0.7509964
## [2,]    0.9673155    0.8849870
## [3,]    0.8009423    0.7361995
```

Training and Validation

Simply using the entire dataset to train models incurs the risk of overtraining, where the model is well-fitted to the particular dataset used for training but generalizes poorly to new data of the same type. As such, the Dry Beans Dataset (both the predictor and outcome subsets) was split into two portions: a training set to train the models on, and a validation set simulating new data to evaluate the "real" performance of each model after training is complete.

Many different training:validation ratios have been proposed by machine learning resources online, with 90:10, 80:20, and 70:30 being especially common. Note that a trade-off exists between the sizes of training and validation sets due to the original dataset's finite size. While having a large pool of data for training is certainly beneficial as it reduces variance in the predicted outcomes, the validation dataset cannot get so small as to cause imprecise performance evaluation due to high variance in the performance statistics.

With 13611 observations/beans, the Dry Bean Dataset is fairly large and allows for a somewhat more generous allocation of data to the validation set; Baeldung CS suggests a 70:30 split for small datasets containing less than 10000 observations, with a 80:20 split being applicable for most cases. As such, a training:validation split of 80:20 was selected for this project.

Note that in most online sources outside EdX, the comparison dataset used to gauge final model performance is called the "training set", and the comparison dataset used for tuning is the "vaidation set"; in short, the two terms are switched between EdX and outside sources. For the sake of consistency, this project uses the EdX naming conventions.

```
# arbitrary seed for reproducibility
set.seed(1, sample.kind = 'Rounding')
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
# creating the validation indeces
val_index <-
  createDataPartition(y = beans$Class,
                      times = 1,
                      p = 0.2,
                      list = F)
# split the data
# as the x-y split had been made earlier, the train-val split must be made for both x and y
beans_x_train <- beans_x[-val_index,]
beans_y_train <- beans_y[-val_index,]
beans_x_val <- beans_x[val_index,]
beans_y_val <- beans_y[val_index,]
```

Unlike with the previous project (MovieLens Rating Prediction), the models used in this project should not rely on the same titles being present in the training and validation (or testing) sets. As such, the `semi_join` function will not be

used to ensure that everything in the validation set being present in the training set. However, it must still be confirmed that both sets have all 7 bean classes.

```
dim(beans_x_val)
```

```
## [1] 2726   13
```

```
as.data.frame(beans_y_val) %>%
  group_by(Class) %>%
  summarize(counts = n())
```

Class	counts
BARBUNYA	265
BOMBAY	105
CALI	326
DERMASON	710
HOROZ	386
SEKER	406
SIRA	528

Table 3: Distribution of bean varieties within the validation subset, split from the original Dry Beans Dataset.

```
dim(beans_x_train)
```

```
## [1] 10885    13
```

```
as.data.frame(beans_y_train) %>%
  group_by(Class) %>%
  summarize(counts = n())
```

Class	counts
BARBUNYA	1057
BOMBAY	417
CALI	1304
DERMASON	2836
HOROZ	1542
SEKER	1621
SIRA	2108

Table 4: Distribution of bean varieties within the training subset, split from the original Dry Beans Dataset.

With 2726 observations in the validation set and 10885 in the training set, the two portions add up to 13611 and no observations were lost during the split. Also, both sets have all 7 bean classes.

Accuracy Metric

For a classification model, some commonly used performance metrics include accuracy, confusion matrix, and F1 score. Due to the class imbalance within the Dry Bean Dataset (see Introduction section), training models on accuracy could lead to misleading results where high accuracy scores are produced through predictions that favor more common classes within the dataset at the expense of rarer ones; such a situation is known as the accuracy paradox, and is caused by the model making predictions based on the imbalanced data distribution alone rather than any connections to the predictors.

Common ways to combat the accuracy paradox include taking larger samples, using a different performance metric, and resampling the dataset. For the choice of accuracy metric, the F1 score was chosen for this project for two reasons: it does not involve alterations of the data through resampling techniques, and it is robust against class imbalances.

The F1 score is the harmonic mean of two metrics: precision (percentage of true positive predictions among all positive predictions) and recall (percentage of true positive predictions among actual positives).

The formula for precision is written as:

$$Precision = \frac{TP}{TP + FP}$$

Where TP stands for True Positive predictions and FP stands for False Positive predictions (actually negative, predicted as positive).

And the formula for recall is written as:

$$Recall = \frac{TP}{TP + FN}$$

Where FN stands for False Negatives, data instances that are actually positive but incorrectly predicted as negative.

The F1 score formula then computes the harmonic mean as written below:

$$F1 = 2\frac{(Precision)(Recall)}{Precision + Recall}$$

For a binary classification problem, one outcome class could be arbitrarily defined as the positive and the other, the negative. However, since the Dry Beans Dataset contains more than two outcome classes, the simple positive/negative distinction must be expanded upon to derive multiclass F1 scores. Multiclass F1 is computed by averaging the F1 score for each outcome category; when computing the F1 score of each class, the class in question is considered a "positive" and all other classes "negatives". Both macro F1 (unweighted average) and micro F1 (average F1 weighted by the size of each class) are used as accuracy metrics; as the goal of choosing F1 scores in this project is to avoid biases favoring larger classes, macro F1 was chosen for turning the models. For the Dry Beans Dataset, the macro F1 formula is thus:

$$mF1 = \frac{\Sigma_{class=1}^{7} F1_{class}}{7}$$

Where there are 7 classes, each denoted as $class$ in the formula.

Two functions, `macro_f1` and `f1`, were written; the former was designed to be a custom summary function under `caret::trainControl` and the later, a helper function for calculating individual F1 scores that together make up the macro F1.

```
# this func should take same args as a function called defaultSummary()
# data = a dataframe with columns 'obs' and 'pred' (observed/actual and predicted)
macro_f1 <- function(data, lev = NULL, model = NULL) {
  mF1 <- mean(f1(data$pred, data$obs)) # arithmetic mean for macro F1
```

```
  names(mF1) <- 'mF1'# names the function output
  return(mF1)
}

f1 <- function(predicted, actual) {
  mat <- as.matrix(table(predicted, actual)) # should make a table where rows are pred, cols are actual,
  precision <- diag(mat) / rowSums(mat) # true pos / all predicted pos
  recall <- diag(mat) / colSums(mat) # true pos / all actual pos
  f1 <- ifelse(precision + recall == 0,
                      0,
                      2 * (precision * recall) / (precision + recall))
  return(f1)
}
```

As seen above, `macro_f1` (and all custom summary functions written for `caret::trainControl`) accepts a dataframe `data` with two columns: `obs` for observed outcomes, and `pred` for predicted outcomes. `f1` converts the dataframe into a confusion matrix tallying correct and incorrect predictions, where rows represent predicted outcomes and columns represent actual or observed outcomes. In such a matrix, a row sum is equal to the total predicted instances of a class, while a column sum represents the total actual instances of a class; the diagonal vector is then a series of correct predictions or True Positives.

Cross-Validation

Regardless of the algorithm used, machine learning contains some randomness on several levels. Sampling and splitting the dataset are random processes, as are the ordering of data instances fed into an algorithm and the iterative building of a predictive model using the algorithm. Setting random seeds before running each model imparts reproducibility, but the reality that each performance metric from a trained model is effectively a sampling statistic with variance attached remains.

One common way to reduce variance in machine learning is k-folds cross-validation, a resampling technique that divides the training set into k folds. For each fold, the remaining data (outside the fold) is used to build a model, with the model accuracy tested using the data within the fold. The performance metrics of all k folds are then averaged to give a metric with lower variance; this is done for each hyperparameter (or combination of hyperparameters) tested during model tuning and training. Common values of k include 3, 5, and 10; while higher values of k are possible, 10 was chosen for this project as a compromise between reducing variance and avoiding overly long computational times. The parameters for a 10-fold cross validation, with each fold being 10 % of the training set, was specified using `caret::trainControl` as seen in the code chunk below.

Note that with 10-fold cross-validation, each fold would be 8 % (10 % of 80 %) of the full beans dataset.

```
# 10-fold cross-validation
tenfold_xval <-
  trainControl(method = 'cv',
               number = 10,
               p = 0.9,
               summaryFunction = macro_f1) # custom macro f1 func in progress
```

Model Choices

Many algorithms are available in the `caret` package for multiclass classification problems such as the one dealt with in this project. To demonstrate familiarity with the EdX program content, the following model types were chosen due to their detailed coverage in the machine learning course of the program:

Logistic Regression

Logistic regression is a type of regression model used to find predictor-outcome correlations (and make predictions from them) in cases where the outcome/dependent variable is discrete rather than continuous. As its name suggests, logistic regression differs from linear regression in its use of the logistic function, the basic form of which is written below:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

This basic logistic function creates a sigmoid curve whose $f(x)$ value goes from approaching 0 to approaching L, with an inflection point at $x = x_0$. When applied to regression, the formula becomes this:

$$y = Pr(y = +|X = x) = \frac{e^{b_0+b_1x}}{1 + e^{b_0+b_1x}}$$

With b_0 being the intercept and b_1 being the coefficient for the variable x, like in linear regression. Unlike in linear regression, y in logistic regression is not the outcome variable itself but the *probability* of y turning out to be a given class (arbitrarily denoted as $+$ in the formula above) given a certain value of predictor X (denoted as lower case x).

Then using this function:

$$g(y) = ln\frac{y}{1-y}$$

The logistic regression function is made linear again, as shown below:

$$g(y) = g(Pr(y = +|X = x)) = b_0 + b_1$$

Where $g(y)$ is the natural log of the odds ratio, for the probability of y being class $+$ given $X = x$.

For this project, five pairs of predictors (the ones visualized on the scatterplots from figures 16-20, in the Introduction section) will be used build models using logistic regression to circumvent the curse of dimensionality, with the highest-performing one being the representative for the algorithm. As mentioned in the introduction section, the five pairs were chosen due to their ability to complement each others' lack of variance among certain classes, and lack of any redundancy in their derivation which would result in multicollinearity.

Since the `glm` method in `caret::train` (the usual method for training logistic regression models) is suited for binary classification only, the method `multinom` was used instead. `multinom` allows for the tuning of decay, a regularization term that penalizes variables with less contributions to the model. As only pairs of variables were selected and fed into the logistic regression algorithm, decay was set to 0.

K-Nearest Neighbors (KNN)

The KNN algorithm is a classification technique based on processes that are intuitive to understand. For each unclassified data point, the algorithm examines a set amount (the namesake k; not to be confused with the one from k-folds cross-validation) of nearest classified data points, before classifying the data point into the most represented class among the k nearest neighbors; for instance, if $k = 10$ and Seker makes up 7 of the 10 neighbors, the new data point would be classified as Seker. Euclidean distance is the default metric for neighbor "nearness" (and the one used in this project), although other distance metrics exist.

For this project, the method `knn` under `caret::train` was used to train all KNN-based models. Like in Logistic regression, the five pairs of predictors from figures 16-20 were used to avoid the curse of dimensionality. k itself was the only hyperparameter to be tuned, and an arbitrary range of 1 to 100 was used in an attempt to capture the optimal k over a large span of possible values.

Decision Tree

The Decision Tree algorithm works by randomly selecting a series of thresholds (for instance, "is normalized perimeter greater than 0.5?") to partition data by, selecting the split (called a node) that leads to maximum homogeneity (uniformity of outcome class) within each branch, and recursively split the data with more sub-nodes and branches until no more sub-nodes could be made (either because some constraint is met or the sub-node has only one data point left).

For this project, the method `rpart` under `caret::train` was used. The only allowed tuning parameter under the `caret` version of `rpart` is Complexity Parameter (CP), a measure of how much improvement (in terms of reducing incorrect classifications) must be achieved for a node to be made. A CP range of 0.005 to 0.050 (in intervals of 0.001) was used to explore smaller CP values (the default CP value is 0.01) for the standalone `rpart` function) after initial test runs found that results were suboptimal at higher CP values; CP values below 0.005 were avoided to avoid overfitting (and overplotted trees too dense for R to visualize) caused by unrestrained node splitting. Unlike with logistic regression and KNN, additional compatible variables were added on top of the best-performing pair from the previous two algorithms to examine the performance of higher-dimensional models.

Random Forest

Since decision trees often fit the training data rather closely, they are naturally prone to overfitting. A random forest mitigates this issue by building multiple independent decision trees, each using a randomly selected subset of the predictor features. The trees are then averaged to provide predictions.

For this project, the method `ranger` under `caret::train` was used for a faster creation of random forests. The tuning hyperparameters consist of `mtry` (how many features to consider at each split; this project tried a range from 1 to all the predictors available), `splitrule` (the homogeneity metric to evaluate goodness of split; Gini impurity was chosen as the classification standard for `ranger`), and `min.node.size` (the minimal number of data points each node must have; an arbitrary range of 1-40 was chosen for coverage of the performance peak region after test runs).

Despite being designed to run random forests faster than most other algorithms, `ranger` still consumed large amounts of time due to the 10-fold cross-validation and large amount of trees making the computational volume extremely large for the laptop used in this project. Due to practical reasons, a threshold of 2 hours (considered the maximum acceptable wait time, given the situation of this project's author at the time of its creation) was set for the Random Forest. Only one combination (the best-performing one from the Decision Tree algorithm) of predictors were used, and the number of independent trees set to 250 (more trees were attempted, but resulted in run times exceeding 2 hours unless if the ranges of tuning hyperparameters were narrowed at the risk of possibly omitting optimal combinations).

Results

Model 1: Logistic Regression

```
# arbitrary seed for reproducibility
set.seed(1, sample.kind = 'Rounding')
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
 # trying various k values (# nearest neighbors)
nearest_neighbors <- seq(1, 100, 1)

# using the y ~ x formula seems to keep throwing errors
model_1a <-
```

```
  train(y = beans_y_train$Class,
        x =  beans_x_train[,c(2,5)], # col 2 for perimeter, 5 for aspect ratio
        method = 'multinom', # multinomial log reg
        trControl = tenfold_xval, # use your 10-fold X-val
        tuneGrid = data.frame(decay = 0), # no need for regularized variable weights
        metric = 'mF1') # my custom metric
```

```
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 7071.845724
## iter  20 value 3119.412293
## iter  30 value 3016.592917
## iter  40 value 3008.702813
## iter  50 value 2989.150993
## iter  60 value 2987.567146
## iter  70 value 2987.484485
## final  value 2987.484247
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 7090.090077
## iter  20 value 3181.720866
## iter  30 value 3038.710115
## iter  40 value 3031.493432
## iter  50 value 3011.499916
## iter  60 value 3009.983721
## iter  70 value 3009.881441
## final  value 3009.880964
## converged
## # weights:  28 (18 variable)
## initial  value 19066.027640
## iter  10 value 7205.676494
## iter  20 value 3228.589311
## iter  30 value 3065.835565
## iter  40 value 3055.088216
## iter  50 value 3037.390151
## iter  60 value 3034.458239
## iter  70 value 3034.155310
## final  value 3034.153059
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 7122.503185
## iter  20 value 3198.089824
## iter  30 value 3050.719007
## iter  40 value 3042.699519
## iter  50 value 3022.951649
## iter  60 value 3020.592795
## iter  70 value 3020.453592
## final  value 3020.452573
## converged
## # weights:  28 (18 variable)
## initial  value 19064.081730
```

```
## iter  10 value 7135.550393
## iter  20 value 3266.925707
## iter  30 value 3061.049650
## iter  40 value 3051.653813
## iter  50 value 3030.777242
## iter  60 value 3029.081719
## iter  70 value 3028.982959
## final  value 3028.982630
## converged
## # weights:  28 (18 variable)
## initial  value 19066.027640
## iter  10 value 7247.363193
## iter  20 value 3184.107893
## iter  30 value 3010.559816
## iter  40 value 3000.418914
## iter  50 value 2980.118408
## iter  60 value 2977.317004
## iter  70 value 2977.066040
## final  value 2977.061178
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 7090.875136
## iter  20 value 3140.829254
## iter  30 value 2991.751084
## iter  40 value 2984.475692
## iter  50 value 2965.200180
## iter  60 value 2963.652042
## iter  70 value 2963.551254
## final  value 2963.550035
## converged
## # weights:  28 (18 variable)
## initial  value 19064.081730
## iter  10 value 7151.281686
## iter  20 value 3295.756049
## iter  30 value 3099.469275
## iter  40 value 3090.330199
## iter  50 value 3070.895642
## iter  60 value 3069.263949
## iter  70 value 3068.929921
## final  value 3068.927990
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 7312.182324
## iter  20 value 3203.418142
## iter  30 value 3033.660464
## iter  40 value 3024.193545
## iter  50 value 3004.465273
## iter  60 value 3002.576434
## iter  70 value 3002.479278
## final  value 3002.478916
## converged
## # weights:  28 (18 variable)
```

```
## initial  value 19060.189910
## iter  10 value 7111.514905
## iter  20 value 3213.947219
## iter  30 value 3007.864971
## iter  40 value 3000.647098
## iter  50 value 2980.775609
## iter  60 value 2979.679071
## iter  70 value 2979.597334
## final  value 2979.597008
## converged
## # weights:  28 (18 variable)
## initial  value 21181.231972
## iter  10 value 12525.427139
## iter  20 value 4113.797896
## iter  30 value 3390.938227
## iter  40 value 3363.748943
## iter  50 value 3343.922186
## iter  60 value 3342.464724
## iter  70 value 3342.452224
## iter  70 value 3342.452209
## iter  70 value 3342.452209
## final  value 3342.452209
## converged
```

```
model_1a$results
```

decay	mF1	mF1SD
0	0.8857739	0.0152556

Table 4: Macro F1 score of Model 1a (Logistic Regression from Perimeter and Aspect Ratio).

Logistic Regression using Perimeter and Aspect Ratio returned a macro F1 score of `0.8857739`, with a standard deviation of `0.01525563`.

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 9664.447677
## iter  20 value 3551.396436
## iter  30 value 2945.509475
## iter  40 value 2922.421639
## iter  50 value 2901.560747
## iter  60 value 2897.294360
## iter  70 value 2897.137447
## final  value 2897.132267
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 9670.967504
## iter  20 value 3426.705131
```

```
## iter  30 value 2973.720683
## iter  40 value 2947.710456
## iter  50 value 2930.724841
## iter  60 value 2926.973392
## iter  70 value 2926.853002
## final  value 2926.850596
## converged
## # weights:  28 (18 variable)
## initial  value 19066.027640
## iter  10 value 9684.852388
## iter  20 value 3468.972422
## iter  30 value 2997.365486
## iter  40 value 2970.318950
## iter  50 value 2952.473112
## iter  60 value 2948.522562
## iter  70 value 2948.340147
## final  value 2948.332944
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 9665.877866
## iter  20 value 3509.704488
## iter  30 value 2984.011679
## iter  40 value 2957.184504
## iter  50 value 2939.764592
## iter  60 value 2933.595378
## iter  70 value 2933.419543
## final  value 2933.402268
## converged
## # weights:  28 (18 variable)
## initial  value 19064.081730
## iter  10 value 9669.164164
## iter  20 value 3416.343413
## iter  30 value 2985.791144
## iter  40 value 2961.592384
## iter  50 value 2943.067896
## iter  60 value 2938.619680
## iter  70 value 2938.541263
## final  value 2938.537295
## converged
## # weights:  28 (18 variable)
## initial  value 19066.027640
## iter  10 value 9641.976619
## iter  20 value 3380.731814
## iter  30 value 2945.587803
## iter  40 value 2918.740996
## iter  50 value 2900.090330
## iter  60 value 2895.047755
## iter  70 value 2894.800024
## final  value 2894.782189
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 9660.690038
```

```
## iter  20 value 3373.099919
## iter  30 value 2922.010907
## iter  40 value 2898.529883
## iter  50 value 2881.513412
## iter  60 value 2877.579220
## iter  70 value 2877.453680
## final  value 2877.448557
## converged
## # weights:  28 (18 variable)
## initial  value 19064.081730
## iter  10 value 9681.021445
## iter  20 value 3457.466740
## iter  30 value 3031.207016
## iter  40 value 3005.166744
## iter  50 value 2985.544840
## iter  60 value 2982.009358
## iter  70 value 2981.937813
## final  value 2981.937748
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 9649.204965
## iter  20 value 3418.192763
## iter  30 value 2959.465253
## iter  40 value 2935.834220
## iter  50 value 2918.931215
## iter  60 value 2916.235708
## iter  70 value 2916.173257
## final  value 2916.171976
## converged
## # weights:  28 (18 variable)
## initial  value 19060.189910
## iter  10 value 9657.082686
## iter  20 value 3332.662107
## iter  30 value 2939.509954
## iter  40 value 2913.845560
## iter  50 value 2896.920162
## iter  60 value 2893.481972
## iter  70 value 2893.405759
## final  value 2893.399828
## converged
## # weights:  28 (18 variable)
## initial  value 21181.231972
## iter  10 value 11210.825427
## iter  20 value 4036.823325
## iter  30 value 3320.202514
## iter  40 value 3274.870143
## iter  50 value 3251.621528
## iter  60 value 3246.531423
## iter  70 value 3246.399419
## final  value 3246.383104
## converged
```

decay	mF1	mF1SD
0	0.8903862	0.0153813

Table 5: Macro F1 score of Model 1b (Logistic Regression from Perimeter and Compactness).

Logistic Regression using Perimeter and Compactness returned a macro F1 score of `0.8903862`, with a standard deviation of `0.01538134`.

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used

## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 6960.452425
## iter  20 value 3074.616657
## iter  30 value 2999.780349
## iter  40 value 2988.511564
## iter  50 value 2973.976475
## iter  60 value 2972.314707
## iter  70 value 2971.916007
## final  value 2971.866027
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 6975.373057
## iter  20 value 3099.981953
## iter  30 value 3023.723837
## iter  40 value 3011.888757
## iter  50 value 2996.892967
## iter  60 value 2995.385104
## iter  70 value 2994.985418
## final  value 2994.929729
## converged
## # weights:  28 (18 variable)
## initial  value 19066.027640
## iter  10 value 6999.272095
## iter  20 value 3119.495949
## iter  30 value 3048.367474
## iter  40 value 3038.724931
## iter  50 value 3024.331476
## iter  60 value 3022.439109
## iter  70 value 3021.694787
## iter  80 value 3021.626287
## iter  90 value 3021.608875
## final  value 3021.606855
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 6976.664132
## iter  20 value 3116.500477
## iter  30 value 3042.459571
## iter  40 value 3030.317480
## iter  50 value 3015.611581
```

```
## iter  60 value 3013.280994
## iter  70 value 3012.517591
## iter  80 value 3012.405677
## iter  80 value 3012.405668
## iter  90 value 3012.393001
## iter 100 value 3012.388999
## final  value 3012.388999
## stopped after 100 iterations
## # weights:  28 (18 variable)
## initial  value 19064.081730
## iter  10 value 6996.038334
## iter  20 value 3121.927349
## iter  30 value 3044.790556
## iter  40 value 3032.447926
## iter  50 value 3015.909542
## iter  60 value 3013.877193
## iter  70 value 3013.458554
## final  value 3013.417096
## converged
## # weights:  28 (18 variable)
## initial  value 19066.027640
## iter  10 value 6930.689000
## iter  20 value 3084.290359
## iter  30 value 3004.326094
## iter  40 value 2992.197680
## iter  50 value 2977.338189
## iter  60 value 2974.810768
## iter  70 value 2974.024938
## final  value 2973.906836
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 6973.745946
## iter  20 value 3035.838193
## iter  30 value 2956.014390
## iter  40 value 2942.775184
## iter  50 value 2927.151534
## iter  60 value 2925.622540
## iter  70 value 2925.105305
## final  value 2925.068617
## converged
## # weights:  28 (18 variable)
## initial  value 19064.081730
## iter  10 value 7008.119977
## iter  20 value 3174.850730
## iter  30 value 3076.590551
## iter  40 value 3070.900461
## iter  50 value 3055.567719
## iter  60 value 3053.779739
## iter  70 value 3053.308219
## iter  80 value 3053.216033
## iter  90 value 3053.190055
## iter 100 value 3053.186862
## final  value 3053.186862
```

```
## stopped after 100 iterations
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 6947.902759
## iter  20 value 3100.953753
## iter  30 value 3025.157527
## iter  40 value 3014.326861
## iter  50 value 2999.865946
## iter  60 value 2998.316288
## iter  70 value 2997.987331
## final  value 2997.950405
## converged
## # weights:  28 (18 variable)
## initial  value 19060.189910
## iter  10 value 6960.570727
## iter  20 value 3076.704031
## iter  30 value 3003.715138
## iter  40 value 2991.304772
## iter  50 value 2976.005194
## iter  60 value 2974.536658
## iter  70 value 2974.096645
## final  value 2974.068071
## converged
## # weights:  28 (18 variable)
## initial  value 21181.231972
## iter  10 value 11495.475093
## iter  20 value 4066.569089
## iter  30 value 3445.859569
## iter  40 value 3369.218188
## iter  50 value 3334.368257
## iter  60 value 3328.973639
## iter  70 value 3328.128544
## iter  80 value 3327.758134
## iter  90 value 3327.720712
## iter 100 value 3327.712665
## final  value 3327.712665
## stopped after 100 iterations
```

decay	mF1	mF1SD
0	0.8974943	0.0095792

Table 6: Macro F1 score of Model 1c (Logistic Regression from Perimeter and Shape Factor 2).

Logistic Regression using Perimeter and Shape Factor 2 returned a macro F1 score of `0.8974943`, with a standard deviation of `0.009579192`.

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 7691.151234
```

```
## iter  20 value 2972.027938
## iter  30 value 2921.873510
## iter  40 value 2915.443320
## iter  50 value 2898.707858
## iter  60 value 2896.705303
## iter  70 value 2896.665599
## final  value 2896.664836
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 7718.342796
## iter  20 value 3005.251877
## iter  30 value 2952.458620
## iter  40 value 2946.065251
## iter  50 value 2930.272941
## iter  60 value 2927.942972
## iter  70 value 2927.891847
## final  value 2927.890945
## converged
## # weights:  28 (18 variable)
## initial  value 19066.027640
## iter  10 value 7733.258206
## iter  20 value 3023.504382
## iter  30 value 2975.202223
## iter  40 value 2968.494470
## iter  50 value 2952.311328
## iter  60 value 2949.488547
## iter  70 value 2949.404728
## final  value 2949.403358
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 7710.135957
## iter  20 value 3014.702587
## iter  30 value 2960.514590
## iter  40 value 2953.368932
## iter  50 value 2937.142673
## iter  60 value 2934.230553
## iter  70 value 2934.158557
## final  value 2934.157346
## converged
## # weights:  28 (18 variable)
## initial  value 19064.081730
## iter  10 value 7709.546259
## iter  20 value 3019.033401
## iter  30 value 2965.781594
## iter  40 value 2958.309882
## iter  50 value 2941.325580
## iter  60 value 2938.867479
## iter  70 value 2938.832774
## final  value 2938.831942
## converged
## # weights:  28 (18 variable)
## initial  value 19066.027640
```

```
## iter  10 value 7669.383755
## iter  20 value 2978.856881
## iter  30 value 2923.377362
## iter  40 value 2916.347514
## iter  50 value 2899.319205
## iter  60 value 2896.266315
## iter  70 value 2896.129517
## final  value 2896.126978
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 7705.826398
## iter  20 value 2952.830753
## iter  30 value 2902.564282
## iter  40 value 2895.817753
## iter  50 value 2880.037441
## iter  60 value 2877.753714
## iter  70 value 2877.718663
## final  value 2877.717864
## converged
## # weights:  28 (18 variable)
## initial  value 19064.081730
## iter  10 value 7735.170432
## iter  20 value 3056.352816
## iter  30 value 3007.665711
## iter  40 value 3001.485483
## iter  50 value 2985.384803
## iter  60 value 2982.713353
## iter  70 value 2982.660491
## final  value 2982.660428
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 7695.177873
## iter  20 value 2992.556422
## iter  30 value 2941.593828
## iter  40 value 2935.370384
## iter  50 value 2919.531452
## iter  60 value 2917.663155
## iter  70 value 2917.636410
## final  value 2917.636106
## converged
## # weights:  28 (18 variable)
## initial  value 19060.189910
## iter  10 value 7704.658889
## iter  20 value 2970.771775
## iter  30 value 2919.221447
## iter  40 value 2912.553884
## iter  50 value 2896.877033
## iter  60 value 2894.849759
## iter  70 value 2894.818540
## final  value 2894.817276
## converged
## # weights:  28 (18 variable)
```

```
## initial  value 21181.231972
## iter  10 value 11441.999957
## iter  20 value 3735.521095
## iter  30 value 3315.515646
## iter  40 value 3271.716892
## iter  50 value 3251.955010
## iter  60 value 3247.372217
## iter  70 value 3247.265704
## final  value 3247.265096
## converged
```

decay	mF1	mF1SD
0	0.890064	0.0160369

Table 7: Macro F1 score of Model 1d (Logistic Regression from Perimeter and Shape Factor 3).

Logistic Regression using Perimeter and Shape Factor 3 returned a macro F1 score of `0.890064`, with a standard deviation of `0.01603693`.

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 8148.570087
## iter  20 value 3197.064312
## iter  30 value 3107.333385
## iter  40 value 3101.772725
## iter  50 value 3089.173845
## iter  60 value 3087.201173
## iter  70 value 3087.009243
## final  value 3086.987280
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 7988.751770
## iter  20 value 3219.185598
## iter  30 value 3133.823319
## iter  40 value 3128.345504
## iter  50 value 3115.590698
## iter  60 value 3113.786242
## iter  70 value 3113.563772
## final  value 3113.529712
## converged
## # weights:  28 (18 variable)
## initial  value 19066.027640
## iter  10 value 7823.373275
## iter  20 value 3250.064012
## iter  30 value 3158.729910
## iter  40 value 3152.868599
## iter  50 value 3140.319110
## iter  60 value 3137.828869
```

```
## iter  70 value 3136.959103
## iter  80 value 3136.492517
## iter  90 value 3136.466114
## iter 100 value 3136.463375
## final  value 3136.463375
## stopped after 100 iterations
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 7915.393857
## iter  20 value 3221.001095
## iter  30 value 3137.979334
## iter  40 value 3132.242769
## iter  50 value 3119.603677
## iter  60 value 3117.966208
## iter  70 value 3117.700715
## final  value 3117.692715
## converged
## # weights:  28 (18 variable)
## initial  value 19064.081730
## iter  10 value 7878.305646
## iter  20 value 3233.921338
## iter  30 value 3151.377926
## iter  40 value 3145.275324
## iter  50 value 3132.262930
## iter  60 value 3130.398427
## iter  70 value 3130.140561
## final  value 3130.095425
## converged
## # weights:  28 (18 variable)
## initial  value 19066.027640
## iter  10 value 7511.118916
## iter  20 value 3188.601439
## iter  30 value 3100.531882
## iter  40 value 3094.206878
## iter  50 value 3082.228386
## iter  60 value 3079.413661
## iter  70 value 3079.097295
## final  value 3079.068705
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 7973.370234
## iter  20 value 3189.745384
## iter  30 value 3094.819199
## iter  40 value 3089.146196
## iter  50 value 3075.851843
## iter  60 value 3074.019746
## iter  70 value 3073.789073
## final  value 3073.750340
## converged
## # weights:  28 (18 variable)
## initial  value 19064.081730
## iter  10 value 7914.831633
## iter  20 value 3276.057383
```

```
## iter  30 value 3195.560632
## iter  40 value 3189.897422
## iter  50 value 3177.188109
## iter  60 value 3175.248740
## iter  70 value 3175.034705
## final  value 3175.004054
## converged
## # weights:  28 (18 variable)
## initial  value 19062.135820
## iter  10 value 7981.975728
## iter  20 value 3196.615571
## iter  30 value 3125.120770
## iter  40 value 3119.977070
## iter  50 value 3107.911337
## iter  60 value 3105.879888
## iter  70 value 3105.696294
## final  value 3105.680720
## converged
## # weights:  28 (18 variable)
## initial  value 19060.189910
## iter  10 value 7945.268012
## iter  20 value 3178.171208
## iter  30 value 3092.160146
## iter  40 value 3087.089989
## iter  50 value 3074.872284
## iter  60 value 3073.239251
## iter  70 value 3073.008826
## final  value 3072.957445
## converged
## # weights:  28 (18 variable)
## initial  value 21181.231972
## iter  10 value 12618.893056
## iter  20 value 4001.088725
## iter  30 value 3485.362793
## iter  40 value 3471.432364
## iter  50 value 3457.676309
## iter  60 value 3455.950989
## iter  70 value 3455.724295
## final  value 3455.718606
## converged
```

decay	mF1	mF1SD
0	0.8820995	0.0141042

Table 8: Macro F1 score of Model 1e (Logistic Regression from Equivalent Diameter and Aspect Ratio).

Logistic Regression using Equivalent Diameter and Aspect Ratio returned a macro F1 score of `0.8820995`, with a standard deviation of `0.01410419`.

Model 2: K-Nearest Neighbors (KNN)

```
# arbitrary seed for reproducibility
set.seed(1, sample.kind = 'Rounding')
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
 # trying various k values (# nearest neighbors)
nearest_neighbors <- seq(1, 100, 1)

model_2a <-
  train(y = beans_y_train$Class,
        x =  beans_x_train[,c(2,5)], # col 2 for perimeter, 5 for aspect ratio
        method = 'knn',
        tuneGrid = data.frame(k = nearest_neighbors),
        trControl = tenfold_xval, # use your 10-fold X-val
        metric = 'mF1') # my custom metric

ggplot(model_2a, highlight = T)
```

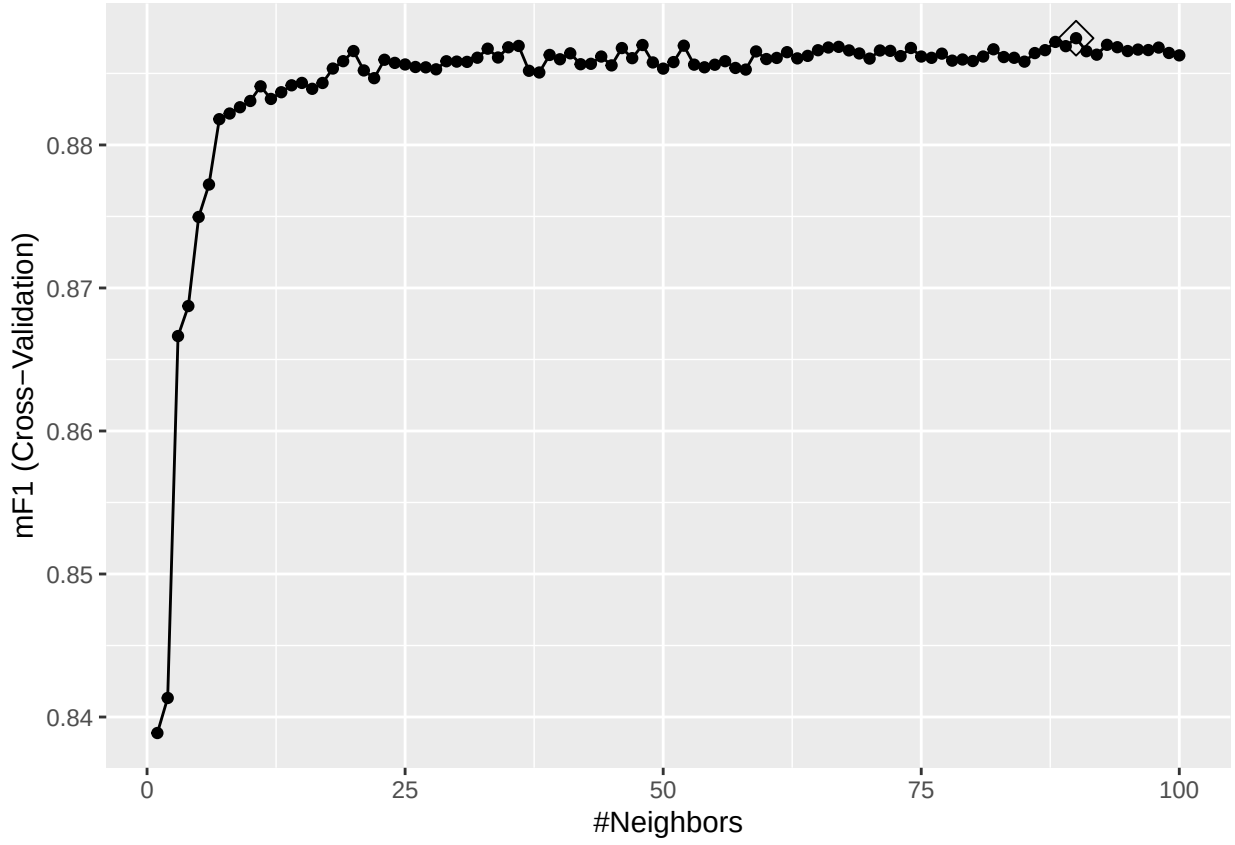

Figure 21: Training macro F1 scores for Model 2a (Perimeter and Aspect Ratio) with 10-fold cross validation, over a range of k from 0 to 100.

`k = 90` (within the diamond label) resulted in the highest macro F1 score, although all results from `k = 25` onwards are similarly high in macro F1.

```
model_2a$bestTune # k = 90 is best for this one
```

	k
90	90

```
model_2a$results[90,]
```

	k	mF1	mF1SD
90	90	0.8874641	0.0120319

Table 9: Macro F1 score of Model 2a (KNN from Perimeter and Aspect Ratio).

KNN using Perimeter and Shape Factor 2 returned a macro F1 score of `0.8820995`, with a standard deviation of `0.01410419`.

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

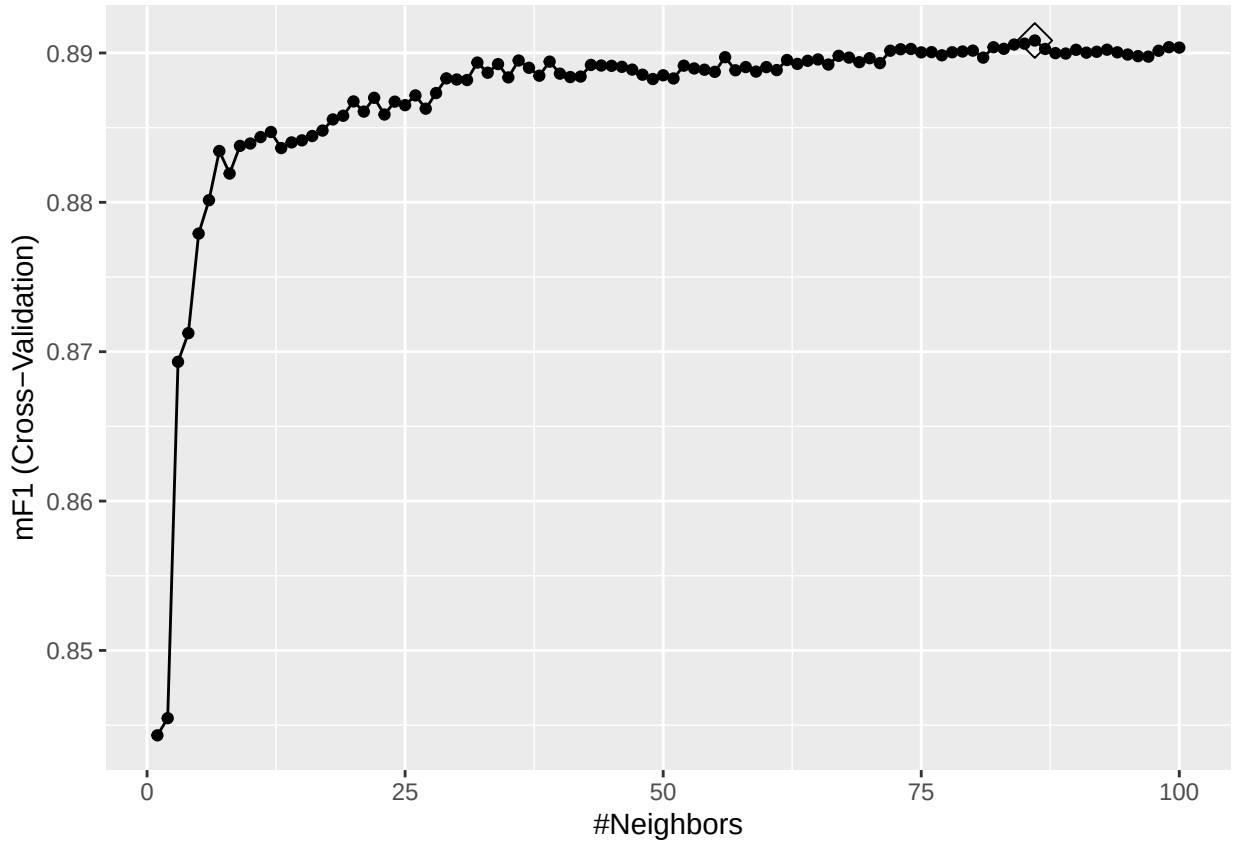

Figure 22: Training macro F1 scores for Model 2b (Perimeter and Compactness) with 10-fold cross validation, over a range of k from 0 to 100.

`k = 86` (within the diamond label) resulted in the highest macro F1 score, although all results from `k = 25` onwards are similarly high in macro F1.

	k
86	86

	k	mF1	mF1SD
86	86	0.8908325	0.011703

Table 10: Macro F1 score of Model 2b (KNN from Perimeter and Compactness).

KNN using Perimeter and Compactness returned a macro F1 score of `0.8908325`, with a standard deviation of `0.01170298`.

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

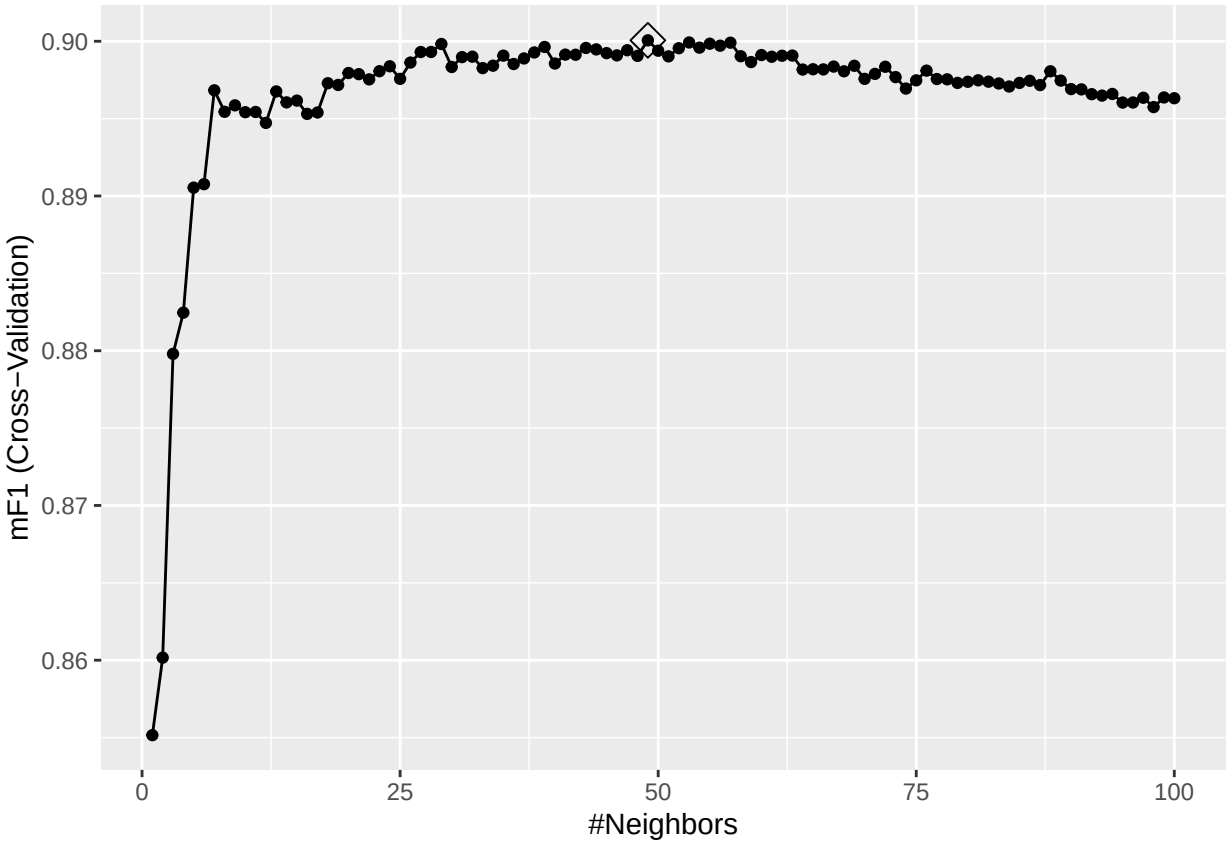

Figure 23: Training macro F1 scores for Model 2c (Perimeter and Shape Factor 2) with 10-fold cross validation, over a range of k from 0 to 100.

`k = 49` (within the diamond label) resulted in the highest macro F1 score, although all results from `k = 25` onwards are similarly high in macro F1. Unlike with the previous two KNN models, mF1 scores exhibited a slight and gradula decrease after `k = 49`.

	k
49	49

	k	mF1	mF1SD
49	49	0.9000603	0.0093475

Table 11: Macro F1 score of Model 2c (KNN from Perimeter and Shape Factor 2).

KNN using Perimeter and Shape Factor 2 returned a macro F1 score of `0.9000603`, with a standard deviation of `0.009347538`.

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

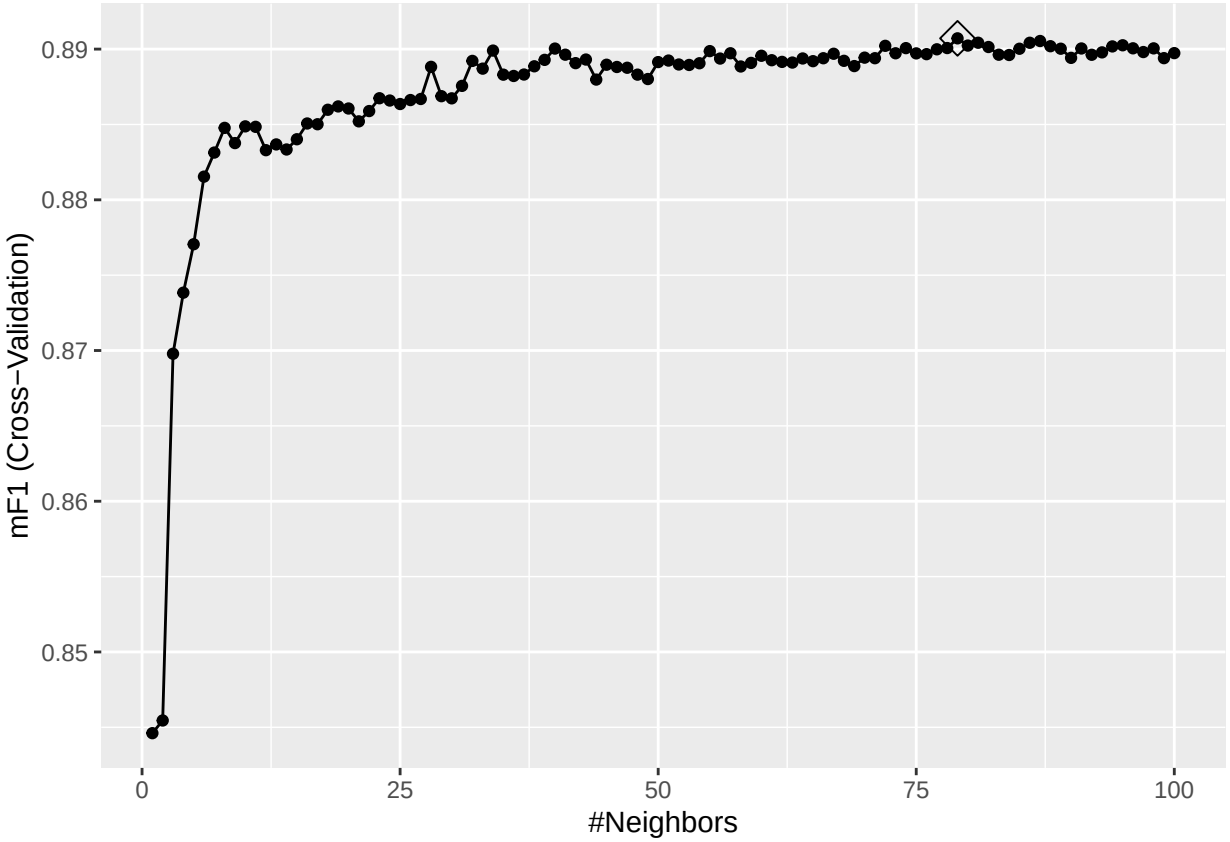

Figure 24: Training macro F1 scores for Model 2d (Perimeter and Shape Factor 3) with 10-fold cross validation, over a range of k from 0 to 100.

`k = 79` (within the diamond label) resulted in the highest macro F1 score. Unlike with the previous KNN models, the plateauing of mF1 values occurred slightly later at a `k` value of 30-40.

	k
79	79

	k	mF1	mF1SD
79	79	0.8907138	0.0120543

Table 12: Macro F1 score of Model 2d (KNN from Perimeter and Shape Factor 3).

KNN using Perimeter and Shape Factor 2 returned a macro F1 score of `0.8907138`, with a standard deviation of `0.01205434`.

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

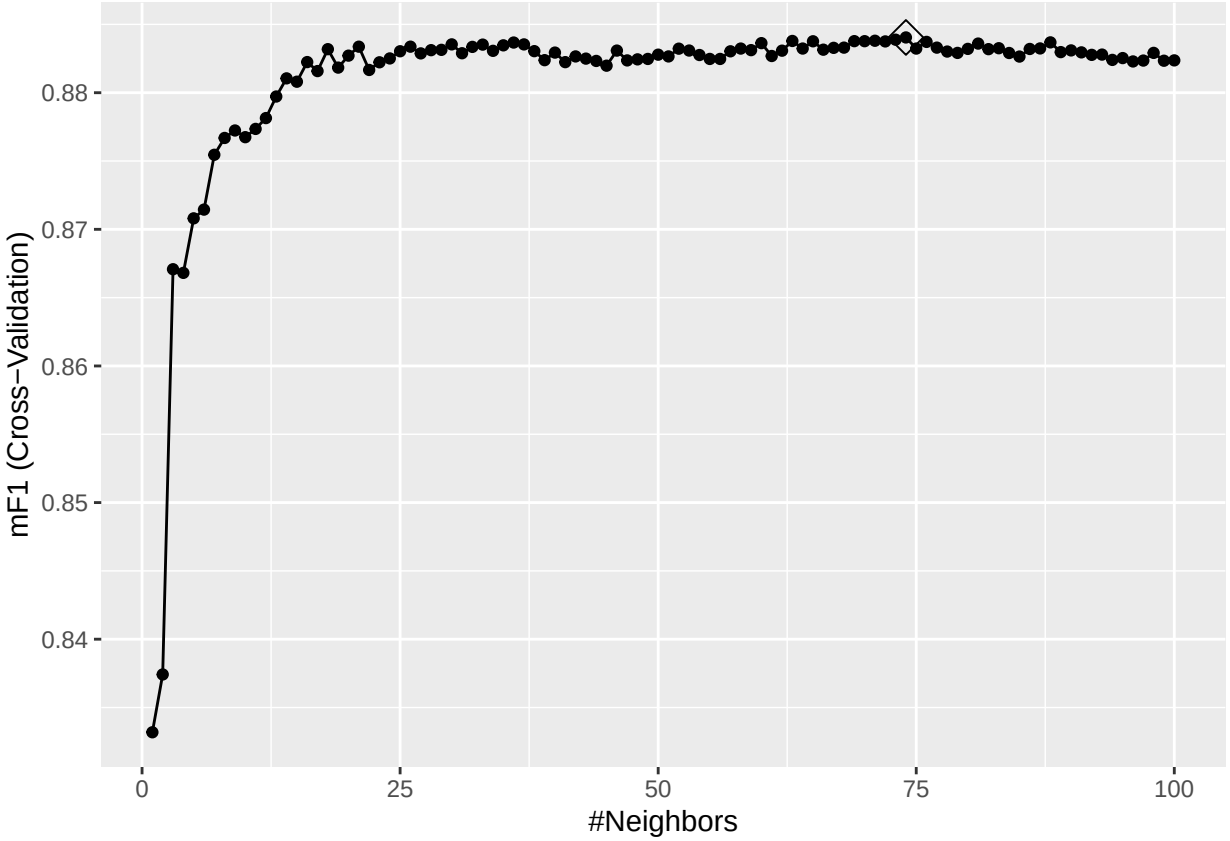

Figure 25: Training macro F1 scores for Model 2e (Equivalent Diameter and Aspect Ratio) with 10-fold cross validation, over a range of k from 0 to 100.

`k = 74` (within the diamond label) resulted in the highest macro F1 score, although most mF1 values after `k = 25` are similarly high.

	k
74	74

	k	mF1	mF1SD
74	74	0.884044	0.0104357

Table 13: Macro F1 score of Model 2e (KNN from Equivalent Diameter and Aspect Ratio).

KNN using Perimeter and Shape Factor 2 returned a macro F1 score of `0.884044`, with a standard deviation of `0.01043572`.

Comparing Two-Variable Model Results

```
models_1_2 <-
  data.table(variables = c('Perimeter, Aspect Ratio',
                           'Perimeter, Compactness',
                           'Perimeter, Shape Factor 2',
                           'Perimeter, Shape Factor 3',
                           'Equivalent Diameter, Aspect Ratio'),
             logreg_mF1_unregularized = c(model_1a$results[,2],
                                          model_1b$results[,2],
                                          model_1c$results[,2],
                                          model_1d$results[,2],
                                          model_1e$results[,2]),
             knn_mF1 = c(model_2a$results[90,2],
                         model_2b$results[86,2],
                         model_2c$results[49,2],
                         model_2d$results[79,2],
                         model_2e$results[74,2]),
             k = c(90, 86, 49, 79, 74))
models_1_2
```

variables	logreg_mF1_unregularized	knn_mF1	k
Perimeter, Aspect Ratio	0.8857739	0.8874641	90
Perimeter, Compactness	0.8903862	0.8908325	86
Perimeter, Shape Factor 2	0.8974943	0.9000603	49
Perimeter, Shape Factor 3	0.8900640	0.8907138	79
Equivalent Diameter, Aspect Ratio	0.8820995	0.8840440	74

Table 14: A summary of macro F1 values from all five of the predictor pairs tested for both Logistic Regression and KNN, plus the k value used to obtain the KNN results.

KNN performed better than logistic regression in all 5 factor combinations, but not by much. In both models, Perimeter and Shape Factor 2 produced the best macro F1 scores (`0.9000603` for KNN, `0.8974943` for Logistic Regression). For the Decision Tree models, more variables will be included on top of those two to see if the macro F1 score can be improved further.

Model 3: Decision Trees

To start off the Decision Tree models, the predictor combination of Perimeter and Shape Factor 2 (which performed optimal results with Logistic Regression and KNN) was used to tune and build a Decision Tree model.

```
# arbitrary seed for reproducibility
set.seed(1, sample.kind = 'Rounding')
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
# Trying various Complexity Parameters
cps <- seq(0.005, 0.050, 0.001)

model_3a <-
  train(y = beans_y_train$Class,
        x =  beans_x_train[,c(2,12)], # col 2 for perimeter, 12 for SF2
        method = 'rpart',
        tuneGrid = data.frame(cp = cps),
        trControl = tenfold_xval, # use your 10-fold X-val
        metric = 'mF1') # my custom metric

model_3a$results
```

cp	mF1	mF1SD
0.005	0.8559795	0.0136723
0.006	0.8515813	0.0142218
0.007	0.8490813	0.0156530
0.008	0.8478499	0.0147111
0.009	0.8478499	0.0147111
0.010	0.8380341	0.0160599
0.011	0.8356017	0.0157822
0.012	0.8333821	0.0124090
0.013	0.8317190	0.0121242
0.014	0.8317190	0.0121242
0.015	0.8317190	0.0121242
0.016	0.8317190	0.0121242
0.017	0.8317190	0.0121242
0.018	0.8317190	0.0121242
0.019	0.8317190	0.0121242
0.020	0.8317190	0.0121242
0.021	0.8317190	0.0121242
0.022	0.8317190	0.0121242
0.023	0.8299456	0.0141594
0.024	0.8299456	0.0141594
0.025	0.8299456	0.0141594
0.026	0.8299456	0.0141594
0.027	0.8299456	0.0141594
0.028	0.8299456	0.0141594
0.029	0.8299456	0.0141594
0.030	0.8229236	0.0098669
0.031	0.8165257	0.0119080
0.032	0.8165257	0.0119080
0.033	0.8165257	0.0119080
0.034	0.8165257	0.0119080
0.035	0.8165257	0.0119080
0.036	0.8141533	0.0119057
0.037	0.8141533	0.0119057
0.038	0.8141533	0.0119057
0.039	0.8141533	0.0119057

cp	mF1	mF1SD
0.040	0.8097317	0.0198460
0.041	0.8045715	0.0155545
0.042	0.7992440	0.0144488
0.043	0.7901095	0.0141454
0.044	0.7809163	0.0130159
0.045	0.7752270	0.0124109
0.046	0.7696884	0.0157268
0.047	0.7656766	0.0156432
0.048	0.7629369	0.0143169
0.049	0.7629369	0.0143169
0.050	0.7629369	0.0143169

Table 15: Tuning results for Model 3a (Decision Tree from Perimeter and Shape Factor 2), with Complexity Parameters, macro F1 values, and standard deviations.

For Model 3a (Decision Tree using Perimeter and Shape Factor 2), a CP of `0.005` produced the highest macro F1 (`0.8559795`) with a standard deviation of `0.013672321`.

The subsequent Decision Trees were run with additional predictors included on top of Perimeter and Shape Factor 2. As mentioned in the Introduction section, perdictors whose formulae involve P (Perimeter), l (Minor Axis Length), and A (Area) will cause multicollinearity when used with Perimeter and Shape Factor 2 (SF2 being derived from dividing l by A). Referring back to the original paper by Koklu and Ozkan (2020), only Major Axis Length (L) and Eccentricity were compatible as additional predictors. Compactness is derived from Equivalent Diameter which contains Area, while Convex Area is defined in relation to Area despite having no formula listed; the remaining perdictors all contained either one or multiple of P, l, and A.

Below are three more Decision trees; two of them have one additional predictor (Major Axis Length or Eccentricity), while the third one has four (both Major Axis Length and Eccentricity on top of Perimeter and SF2).

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

cp	mF1	mF1SD
0.005	0.8658147	0.0159060
0.006	0.8553960	0.0177346
0.007	0.8440579	0.0139585
0.008	0.8440579	0.0139585
0.009	0.8415938	0.0146252
0.010	0.8338241	0.0140643
0.011	0.8251118	0.0146928
0.012	0.8158309	0.0130953
0.013	0.8118943	0.0133893
0.014	0.8096328	0.0125842
0.015	0.7948377	0.0224539
0.016	0.7844982	0.0171420
0.017	0.7832566	0.0157481
0.018	0.7832566	0.0157481
0.019	0.7832566	0.0157481
0.020	0.7832566	0.0157481
0.021	0.7832566	0.0157481
0.022	0.7832566	0.0157481

cp	mF1	mF1SD
0.023	0.7832566	0.0157481
0.024	0.7832566	0.0157481
0.025	0.7832566	0.0157481
0.026	0.7832566	0.0157481
0.027	0.7832566	0.0157481
0.028	0.7832566	0.0157481
0.029	0.7832566	0.0157481
0.030	0.7832566	0.0157481
0.031	0.7832566	0.0157481
0.032	0.7832566	0.0157481
0.033	0.7832566	0.0157481
0.034	0.7832566	0.0157481
0.035	0.7832566	0.0157481
0.036	0.7832566	0.0157481
0.037	0.7832566	0.0157481
0.038	0.7832566	0.0157481
0.039	0.7832566	0.0157481
0.040	0.7832566	0.0157481
0.041	0.7832566	0.0157481
0.042	0.7832566	0.0157481
0.043	0.7832566	0.0157481
0.044	0.7832566	0.0157481
0.045	0.7832566	0.0157481
0.046	0.7832566	0.0157481
0.047	0.7832566	0.0157481
0.048	0.7832566	0.0157481
0.049	0.7832566	0.0157481
0.050	0.7832566	0.0157481

Table 16: Tuning results for Model 3b (Decision Tree from Perimeter, Shape Factor 2, and Major Axis Length), with Complexity Parameters, macro F1 values, and standard deviations.

For Model 3b, a CP of `0.005` produced the highest macro F1 (`0.8658147`) with a standard deviation of `0.01590596`.

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

cp	mF1	mF1SD
0.005	0.8674779	0.0086666
0.006	0.8652256	0.0092363
0.007	0.8652256	0.0092363
0.008	0.8652256	0.0092363
0.009	0.8652256	0.0092363
0.010	0.8652256	0.0092363
0.011	0.8652256	0.0092363
0.012	0.8652256	0.0092363
0.013	0.8652256	0.0092363
0.014	0.8652256	0.0092363
0.015	0.8652256	0.0092363
0.016	0.8652256	0.0092363

cp	mF1	mF1SD
0.017	0.8652256	0.0092363
0.018	0.8652256	0.0092363
0.019	0.8652256	0.0092363
0.020	0.8652256	0.0092363
0.021	0.8652256	0.0092363
0.022	0.8652256	0.0092363
0.023	0.8652256	0.0092363
0.024	0.8652256	0.0092363
0.025	0.8652256	0.0092363
0.026	0.8652256	0.0092363
0.027	0.8652256	0.0092363
0.028	0.8652256	0.0092363
0.029	0.8652256	0.0092363
0.030	0.8652256	0.0092363
0.031	0.8652256	0.0092363
0.032	0.8652256	0.0092363
0.033	0.8652256	0.0092363
0.034	0.8652256	0.0092363
0.035	0.8652256	0.0092363
0.036	0.8652256	0.0092363
0.037	0.8652256	0.0092363
0.038	0.8652256	0.0092363
0.039	0.8652256	0.0092363
0.040	0.8652256	0.0092363
0.041	0.8652256	0.0092363
0.042	0.8621499	0.0148399
0.043	0.8568988	0.0171117
0.044	0.8568988	0.0171117
0.045	0.8568988	0.0171117
0.046	0.8568988	0.0171117
0.047	0.8568988	0.0171117
0.048	0.8534504	0.0171548
0.049	0.8373392	0.0159731
0.050	0.8373392	0.0159731

Table 17: Tuning results for Model 3c (Decision Tree from Perimeter, Shape Factor 2, and Eccentricity), with Complexity Parameters, macro F1 values, and standard deviations.

For Model 3c, a CP of `0.005` produced the highest macro F1 (`0.8674779`) with a standard deviation of `0.008666564`. CPs from `0.006` to `0.041` all returned a mF1 of `0.8652256`.

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

cp	mF1	mF1SD
0.005	0.8673091	0.0099159
0.006	0.8600591	0.0104971
0.007	0.8578345	0.0095030
0.008	0.8549304	0.0106634
0.009	0.8549304	0.0106634

cp	mF1	mF1SD
0.010	0.8549304	0.0106634
0.011	0.8549304	0.0106634
0.012	0.8549304	0.0106634
0.013	0.8549304	0.0106634
0.014	0.8549304	0.0106634
0.015	0.8549304	0.0106634
0.016	0.8549304	0.0106634
0.017	0.8549304	0.0106634
0.018	0.8549304	0.0106634
0.019	0.8549304	0.0106634
0.020	0.8549304	0.0106634
0.021	0.8549304	0.0106634
0.022	0.8549304	0.0106634
0.023	0.8549304	0.0106634
0.024	0.8549304	0.0106634
0.025	0.8549304	0.0106634
0.026	0.8549304	0.0106634
0.027	0.8549304	0.0106634
0.028	0.8549304	0.0106634
0.029	0.8549304	0.0106634
0.030	0.8549304	0.0106634
0.031	0.8549304	0.0106634
0.032	0.8549304	0.0106634
0.033	0.8549304	0.0106634
0.034	0.8549304	0.0106634
0.035	0.8549304	0.0106634
0.036	0.8549304	0.0106634
0.037	0.8549304	0.0106634
0.038	0.8549304	0.0106634
0.039	0.8549304	0.0106634
0.040	0.8549304	0.0106634
0.041	0.8549304	0.0106634
0.042	0.8549304	0.0106634
0.043	0.8549304	0.0106634
0.044	0.8549304	0.0106634
0.045	0.8549304	0.0106634
0.046	0.8549304	0.0106634
0.047	0.8549304	0.0106634
0.048	0.8549304	0.0106634
0.049	0.8549304	0.0106634
0.050	0.8549304	0.0106634

Table 18: Tuning results for Model 3d (Decision Tree from Perimeter, Shape Factor 2, Major Axis Length, and Eccentricity), with Complexity Parameters, macro F1 values, and standard deviations.

For Model 3d, a CP of `0.005` produced the highest macro F1 (`0.8673091`) with a standard deviation of `0.009915875`. CPs from `0.008` to `0.050` all returned a mF1 of `0.8549304`.

The trained Decision Tree with the best performance was thus Model 3c with a CP of `0.005`. Below is a visualization of the tree:

```
rpart.plot(model_3c$finalModel)
```

```
## Warning: All boxes will be white (the box.palette argument will be ignored) because
## the number of classes in the response 7 is greater than length(box.palette) 6.
## To silence this warning use box.palette=0 or trace=-1.
```

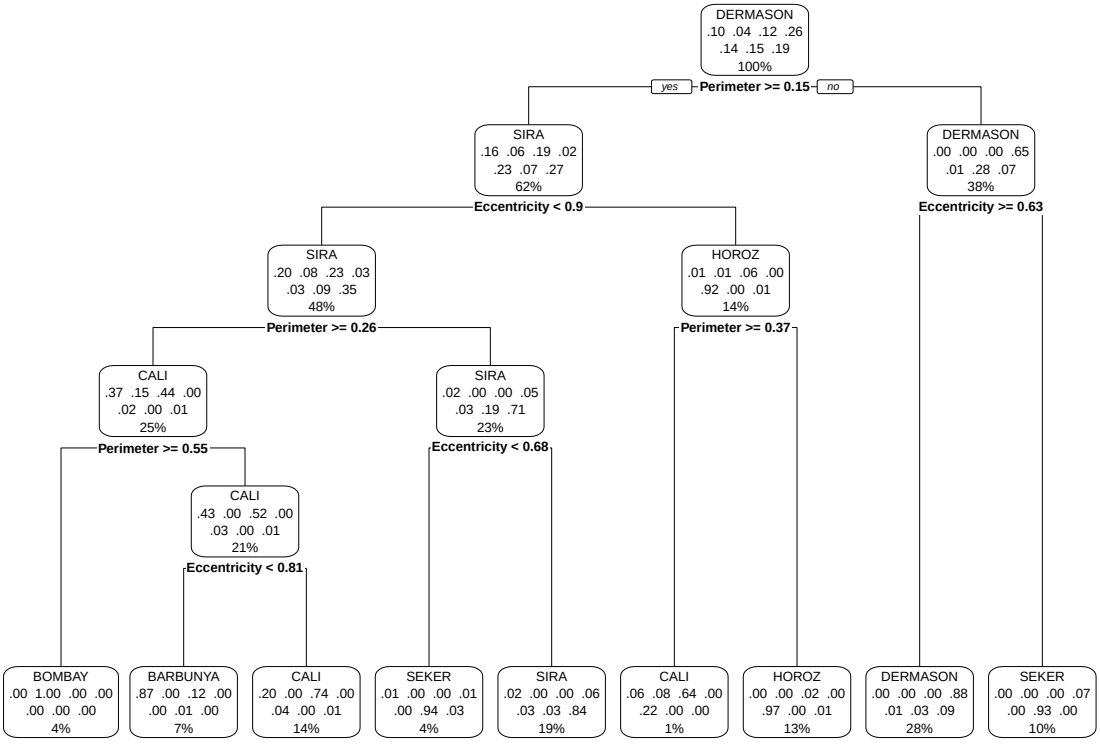

Figure 21: The trained Decision Tree (Model 3c) using variables Perimeter, Shape Factor 2, and Eccentricity with CP = 0.05.

Using `rpart.plot` to generating a diagram of the Model 3c tree, the criteria for splitting (bold inequality statements under the non-terminal nodes), the percentage of total training data accounted for (the percentage at the bottom of each node), and the probabilities of different bean classes under each node (the series of decimals on the middle, between the bean class names and data percentage). The order of classes for the decimal probabilities are: Barbunya, Bombay, Cali, Dermason, Horoz, Seker, and Sira. The left branch to each non-terminal node represents data points that return `TRUE` for the splitting criteria, and the right branch represents data returning `FALSE` to the same statement. For example, the left-most terminal (Perimeter >= 0.66 AND Eccentricity < 0.9) predicted bean variety Bombay (which accounts for 4 % of the training data) with 100 % acuracy (probability of Bombay = 1.00). Bombay, Horoz, and Seker had the highest accuracies, while Cali had the lowest.

Model 4: Random Forests

With the optimal combination of Decision Tree input predictors found, the following Random Forest was tuned:

```r
# arbitrary seed for reproducibility
set.seed(1, sample.kind = 'Rounding')
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
# RF tuning grid
forest_grid <-
  expand.grid(
    mtry = 1:3, # how many variables to use for splitting at each tree?
    splitrule = 'gini', # for classification. Not changing this
    min.node.size = 1:40
  )

model_4 <-
  train(y = beans_y_train$Class,
        x =  beans_x_train[,c(2,12,6)],
        method = 'ranger',
        tuneGrid = forest_grid,
        num.trees = 250, # rough runtime of 2 hrs
        trControl = tenfold_xval, # use your 10-fold X-val
        metric = 'mF1') # my custom metric

model_4$results
```

mtry	splitrule	min.node.size	mF1	mF1SD
1	gini	1	0.9067708	0.0110281
1	gini	2	0.9076547	0.0105223
1	gini	3	0.9085230	0.0100075
1	gini	4	0.9069243	0.0083958
1	gini	5	0.9095317	0.0102279
1	gini	6	0.9088801	0.0088810
1	gini	7	0.9086677	0.0101455
1	gini	8	0.9091978	0.0097288
1	gini	9	0.9092881	0.0100745
1	gini	10	0.9103829	0.0101511
1	gini	11	0.9101893	0.0098243
1	gini	12	0.9100917	0.0106396
1	gini	13	0.9102974	0.0108383
1	gini	14	0.9106326	0.0106461
1	gini	15	0.9088118	0.0105709
1	gini	16	0.9096342	0.0100507
1	gini	17	0.9107815	0.0104377
1	gini	18	0.9086020	0.0094118
1	gini	19	0.9096116	0.0111050
1	gini	20	0.9086069	0.0121275
1	gini	21	0.9089261	0.0109909
1	gini	22	0.9101619	0.0114776
1	gini	23	0.9096473	0.0119767
1	gini	24	0.9101888	0.0107457
1	gini	25	0.9098249	0.0111246

mtry	splitrule	min.node.size	mF1	mF1SD
1	gini	26	0.9099331	0.0112675
1	gini	27	0.9097291	0.0116202
1	gini	28	0.9077802	0.0110463
1	gini	29	0.9091939	0.0110135
1	gini	30	0.9075202	0.0126540
1	gini	31	0.9093819	0.0120522
1	gini	32	0.9079256	0.0110793
1	gini	33	0.9088011	0.0116562
1	gini	34	0.9072837	0.0124067
1	gini	35	0.9080999	0.0124333
1	gini	36	0.9066916	0.0117469
1	gini	37	0.9078818	0.0122035
1	gini	38	0.9058290	0.0120276
1	gini	39	0.9069641	0.0124573
1	gini	40	0.9077013	0.0122032
2	gini	1	0.9070176	0.0099658
2	gini	2	0.9071122	0.0103828
2	gini	3	0.9078065	0.0099157
2	gini	4	0.9094818	0.0092408
2	gini	5	0.9081818	0.0099406
2	gini	6	0.9084090	0.0103214
2	gini	7	0.9080919	0.0106656
2	gini	8	0.9092792	0.0112327
2	gini	9	0.9097003	0.0101497
2	gini	10	0.9096315	0.0102679
2	gini	11	0.9104148	0.0101773
2	gini	12	0.9100787	0.0099533
2	gini	13	0.9098406	0.0100366
2	gini	14	0.9095593	0.0101763
2	gini	15	0.9102904	0.0102193
2	gini	16	0.9107858	0.0094701
2	gini	17	0.9092326	0.0102515
2	gini	18	0.9089504	0.0102798
2	gini	19	0.9089684	0.0098881
2	gini	20	0.9097102	0.0102269
2	gini	21	0.9103838	0.0099616
2	gini	22	0.9105283	0.0097284
2	gini	23	0.9099828	0.0099251
2	gini	24	0.9105445	0.0091299
2	gini	25	0.9094632	0.0091174
2	gini	26	0.9106493	0.0090847
2	gini	27	0.9093792	0.0098166
2	gini	28	0.9091686	0.0097127
2	gini	29	0.9096835	0.0098717
2	gini	30	0.9094397	0.0086056
2	gini	31	0.9104230	0.0108034
2	gini	32	0.9092608	0.0102822
2	gini	33	0.9082718	0.0099892
2	gini	34	0.9099437	0.0098575
2	gini	35	0.9093819	0.0103490
2	gini	36	0.9092616	0.0108241
2	gini	37	0.9092541	0.0112230

mtry	splitrule	min.node.size	mF1	mF1SD
2	gini	38	0.9093234	0.0108006
2	gini	39	0.9091872	0.0104489
2	gini	40	0.9092970	0.0100660
3	gini	1	0.9069613	0.0095516
3	gini	2	0.9071340	0.0098477
3	gini	3	0.9086934	0.0097480
3	gini	4	0.9075331	0.0091581
3	gini	5	0.9088370	0.0103467
3	gini	6	0.9083604	0.0110528
3	gini	7	0.9097639	0.0093959
3	gini	8	0.9099142	0.0104437
3	gini	9	0.9085175	0.0099562
3	gini	10	0.9110494	0.0097912
3	gini	11	0.9094465	0.0106152
3	gini	12	0.9096376	0.0098222
3	gini	13	0.9101784	0.0104767
3	gini	14	0.9094404	0.0105358
3	gini	15	0.9095904	0.0103489
3	gini	16	0.9101347	0.0094494
3	gini	17	0.9088356	0.0099209
3	gini	18	0.9103826	0.0106315
3	gini	19	0.9097175	0.0090182
3	gini	20	0.9096474	0.0086959
3	gini	21	0.9081786	0.0085765
3	gini	22	0.9096510	0.0089952
3	gini	23	0.9102110	0.0085978
3	gini	24	0.9108200	0.0083702
3	gini	25	0.9083543	0.0090854
3	gini	26	0.9091016	0.0105096
3	gini	27	0.9084657	0.0100451
3	gini	28	0.9094219	0.0093445
3	gini	29	0.9088570	0.0101295
3	gini	30	0.9088273	0.0094022
3	gini	31	0.9083694	0.0104971
3	gini	32	0.9096591	0.0092009
3	gini	33	0.9089711	0.0100475
3	gini	34	0.9084149	0.0096280
3	gini	35	0.9091714	0.0094674
3	gini	36	0.9089017	0.0105027
3	gini	37	0.9085937	0.0103163
3	gini	38	0.9083118	0.0096023
3	gini	39	0.9074987	0.0104812
3	gini	40	0.9089170	0.0103071

Table 19: Random Forest (Model 4) tuning results using Ranger, with predictors Perimeter, Shape Factor 2, and Eccentricity.

The optimal Model 4 tuning hyperparameters were `mtry = 3` and `min.node.size = 10`. The resulting macro F1 was `0.9110494` with a standard deviation of `0.009791189`.

```
model_4$bestTune
```

	mtry	splitrule	min.node.size
90	3	gini	10

Table 20: Optimized Random Forest tuning parameters.

```
plot(model_4)
```

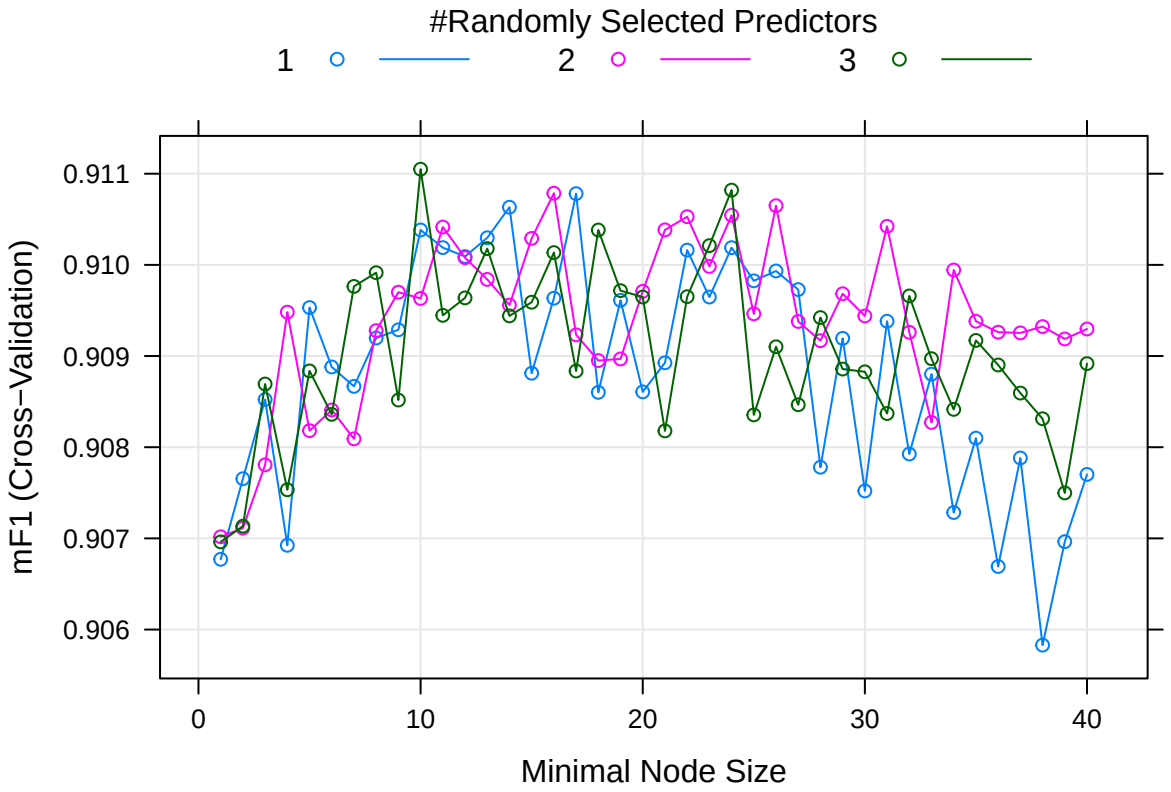

Figure 22: mF1 values over different minimal node sizes for mtry = 1 (blue), mtry = 2 (magenta), and mtry = 3 (green).

The trend lines for Random Forest mf1 values over minimal node sizes were rather jagged (indicating high variance); however, growing more trees would result in an unacceptably long run time.

Validation Run

While model 4 (the sole Random Forests model) returned the highest training mF1 score, it would be worthwhile to perform validation runs on the representatives of all algorithm used not just to check for overfitting, but build confusion matrices to compare class-specific prediction weaknesses (if any are present) in each model. For Decision Trees, the basic 2-variable model was also validated alongside the optimized 3-variable model to compare their performances.

```
# convert beans_y_val$Class to class: Factor
beans_y_val$Class <- as.factor(beans_y_val$Class) # convert to factor class
levels(beans_y_val$Class) # checking that format is right
```

```
## [1] "BARBUNYA" "BOMBAY"   "CALI"     "DERMASON" "HOROZ"    "SEKER"    "SIRA"
```

```
validation_preds_LR <- predict(model_1c$finalModel, beans_x_val)

confusionMatrix(validation_preds_LR, beans_y_val$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
##   BARBUNYA      216      0   18        0     1     0    2
##   BOMBAY          0    105    0        0     0     0    0
##   CALI           41      0  300        0    20     0    1
##   DERMASON        0      0    0      648     7    12   43
##   HOROZ           0      0    5        1   352     0   17
##   SEKER           3      0    0       12     0   377   11
##   SIRA            5      0    3       49     6    17  454
##
## Overall Statistics
##
##                Accuracy : 0.8995
##                  95% CI : (0.8876, 0.9105)
##     No Information Rate : 0.2605
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8784
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BARBUNYA Class: BOMBAY Class: CALI Class: DERMASON
## Sensitivity                  0.81509       1.00000      0.9202          0.9127
## Specificity                  0.99147       1.00000      0.9742          0.9692
## Pos Pred Value               0.91139       1.00000      0.8287          0.9127
## Neg Pred Value               0.98031       1.00000      0.9890          0.9692
## Prevalence                   0.09721       0.03852      0.1196          0.2605
## Detection Rate               0.07924       0.03852      0.1101          0.2377
## Detection Prevalence         0.08694       0.03852      0.1328          0.2605
## Balanced Accuracy            0.90328       1.00000      0.9472          0.9410
##                      Class: HOROZ Class: SEKER Class: SIRA
## Sensitivity                0.9119       0.9286      0.8598
## Specificity                0.9902       0.9888      0.9636
## Pos Pred Value             0.9387       0.9355      0.8502
## Neg Pred Value             0.9855       0.9875      0.9662
## Prevalence                 0.1416       0.1489      0.1937
## Detection Rate             0.1291       0.1383      0.1665
## Detection Prevalence       0.1376       0.1478      0.1959
## Balanced Accuracy          0.9510       0.9587      0.9117
```

Table 21: Validation run results for the representative Logistic Regression model (1c; Perimeter and Shape Factor 2).

```
# KNN in R is not like the others, in that it fits and evaluates in one function
# Thus, a second training is done with the pre-tuned k value
```

```r
# arbitrary seed for reproducibility
set.seed(1, sample.kind = 'Rounding')
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
validation_preds_KNN <- knn(train = beans_x_train[,c(2,12)],
                            cl = beans_y_train$Class,
                            test = beans_x_val[,c(2,12)],
                            k = 49) # KNN run on optimized k value tuned from caret::train

confusionMatrix(validation_preds_KNN, beans_y_val$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
##   BARBUNYA      219      0   22        0     2     0    0
##   BOMBAY          0    105    0        0     0     0    0
##   CALI           39      0  299        0    16     0    1
##   DERMASON        0      0    0      648     7     7   55
##   HOROZ           0      0    3        2   350     0    7
##   SEKER           3      0    0       19     0   381    8
##   SIRA            4      0    2       41    11    18  457
##
## Overall Statistics
##
##                Accuracy : 0.9021
##                  95% CI : (0.8903, 0.913)
##     No Information Rate : 0.2605
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8815
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BARBUNYA Class: BOMBAY Class: CALI Class: DERMASON
## Sensitivity                  0.82642       1.00000      0.9172          0.9127
## Specificity                  0.99025       1.00000      0.9767          0.9658
## Pos Pred Value               0.90123       1.00000      0.8423          0.9038
## Neg Pred Value               0.98147       1.00000      0.9886          0.9691
## Prevalence                   0.09721       0.03852      0.1196          0.2605
## Detection Rate               0.08034       0.03852      0.1097          0.2377
## Detection Prevalence         0.08914       0.03852      0.1302          0.2630
## Balanced Accuracy            0.90833       1.00000      0.9469          0.9392
##                      Class: HOROZ Class: SEKER Class: SIRA
## Sensitivity                0.9067       0.9384      0.8655
## Specificity                0.9949       0.9871      0.9654
## Pos Pred Value             0.9669       0.9270      0.8574
## Neg Pred Value             0.9848       0.9892      0.9676
```

```
## Prevalence                0.1416       0.1489      0.1937
## Detection Rate            0.1284       0.1398      0.1676
## Detection Prevalence      0.1328       0.1508      0.1955
## Balanced Accuracy         0.9508       0.9627      0.9155
```

Table 22: Validation run results for the representative KNN model (2c; Perimeter and Shape Factor 2).

```
# DT preds take dataframes...
validation_preds_2DT <- predict(model_3a$finalModel, data.frame(beans_x_val), type = 'class')

confusionMatrix(validation_preds_2DT, beans_y_val$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
##   BARBUNYA      202      0   46        0     3     1    2
##   BOMBAY          0    105    0        0     0     0    0
##   CALI           48      0  250        0    16     0    0
##   DERMASON        0      0    0      661     7    23   58
##   HOROZ           6      0   27        0   347     0    9
##   SEKER           1      0    0       17     0   363    7
##   SIRA            8      0    3       32    13    19  452
##
## Overall Statistics
##
##                Accuracy : 0.8731
##                  95% CI : (0.86, 0.8853)
##     No Information Rate : 0.2605
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8462
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BARBUNYA Class: BOMBAY Class: CALI Class: DERMASON
## Sensitivity                  0.76226       1.00000     0.76687          0.9310
## Specificity                  0.97887       1.00000     0.97333          0.9563
## Pos Pred Value               0.79528       1.00000     0.79618          0.8825
## Neg Pred Value               0.97451       1.00000     0.96849          0.9752
## Prevalence                   0.09721       0.03852     0.11959          0.2605
## Detection Rate               0.07410       0.03852     0.09171          0.2425
## Detection Prevalence         0.09318       0.03852     0.11519          0.2748
## Balanced Accuracy            0.87057       1.00000     0.87010          0.9437
##                      Class: HOROZ Class: SEKER Class: SIRA
## Sensitivity                0.8990       0.8941      0.8561
## Specificity                0.9821       0.9892      0.9659
## Pos Pred Value             0.8920       0.9356      0.8577
## Neg Pred Value             0.9833       0.9816      0.9654
## Prevalence                 0.1416       0.1489      0.1937
## Detection Rate             0.1273       0.1332      0.1658
```

```
## Detection Prevalence       0.1427       0.1423      0.1933
## Balanced Accuracy          0.9405       0.9417      0.9110
```

Table 23: Validation run results for the 2-variable Decision Tree model (3a; Perimeter and Shape Factor 2).

```
validation_preds_3DT <- predict(model_3c$finalModel, data.frame(beans_x_val), type = 'class')

confusionMatrix(validation_preds_3DT, beans_y_val$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
##   BARBUNYA      184      0   23        0     1     1    0
##   BOMBAY          0    105    0        0     0     0    0
##   CALI           76      0  294        0    34     0    4
##   DERMASON        0      0    0      645     7    19   61
##   HOROZ           0      0    8        0   329     0    1
##   SEKER           3      0    0       33     0   367    5
##   SIRA            2      0    1       32    15    19  457
##
## Overall Statistics
##
##                Accuracy : 0.8734
##                  95% CI : (0.8604, 0.8857)
##     No Information Rate : 0.2605
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8468
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BARBUNYA Class: BOMBAY Class: CALI Class: DERMASON
## Sensitivity                  0.69434       1.00000      0.9018          0.9085
## Specificity                  0.98984       1.00000      0.9525          0.9568
## Pos Pred Value               0.88038       1.00000      0.7206          0.8811
## Neg Pred Value               0.96782       1.00000      0.9862          0.9674
## Prevalence                   0.09721       0.03852      0.1196          0.2605
## Detection Rate               0.06750       0.03852      0.1079          0.2366
## Detection Prevalence         0.07667       0.03852      0.1497          0.2685
## Balanced Accuracy            0.84209       1.00000      0.9272          0.9326
##                      Class: HOROZ Class: SEKER Class: SIRA
## Sensitivity                0.8523       0.9039      0.8655
## Specificity                0.9962       0.9823      0.9686
## Pos Pred Value             0.9734       0.8995      0.8688
## Neg Pred Value             0.9761       0.9832      0.9677
## Prevalence                 0.1416       0.1489      0.1937
## Detection Rate             0.1207       0.1346      0.1676
## Detection Prevalence       0.1240       0.1497      0.1930
## Balanced Accuracy          0.9242       0.9431      0.9171
```

Table 24: Validation run results for the 3-variable Decision Tree model (3c; Perimeter, Shape Factor 2, and Eccentricity).

```
validation_preds_RF <- predict(model_4$finalModel, beans_x_val)

confusionMatrix(validation_preds_RF$predictions, beans_y_val$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
##   BARBUNYA      236      1   15        0     2     0    2
##   BOMBAY          0    104    0        0     0     0    0
##   CALI           23      0  301        0    11     0    0
##   DERMASON        0      0    0      656     6     9   55
##   HOROZ           0      0    7        3   355     0    7
##   SEKER           3      0    0       16     0   383   11
##   SIRA            3      0    3       35    12    14  453
##
## Overall Statistics
##
##                Accuracy : 0.9127
##                  95% CI : (0.9015, 0.923)
##     No Information Rate : 0.2605
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8944
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BARBUNYA Class: BOMBAY Class: CALI Class: DERMASON
## Sensitivity                  0.89057       0.99048      0.9233          0.9239
## Specificity                  0.99187       1.00000      0.9858          0.9653
## Pos Pred Value               0.92187       1.00000      0.8985          0.9036
## Neg Pred Value               0.98826       0.99962      0.9895          0.9730
## Prevalence                   0.09721       0.03852      0.1196          0.2605
## Detection Rate               0.08657       0.03815      0.1104          0.2406
## Detection Prevalence         0.09391       0.03815      0.1229          0.2663
## Balanced Accuracy            0.94122       0.99524      0.9546          0.9446
##                      Class: HOROZ Class: SEKER Class: SIRA
## Sensitivity                0.9197       0.9433      0.8580
## Specificity                0.9927       0.9871      0.9695
## Pos Pred Value             0.9543       0.9274      0.8712
## Neg Pred Value             0.9868       0.9901      0.9660
## Prevalence                 0.1416       0.1489      0.1937
## Detection Rate             0.1302       0.1405      0.1662
## Detection Prevalence       0.1365       0.1515      0.1908
## Balanced Accuracy          0.9562       0.9652      0.9137
```

Table 25: Validation run results for the Random Forest model (3c; Perimeter, Shape Factor 2, and Eccentricity).

Final Tabulation of F1 Scores

model	Barbunya	Bombay	Cali	Dermason	Horoz	Seker	Sira	Macro
Logistic Regression	0.8605578	1.0000000	0.8720930	0.9126761	0.9250986	0.9320148	0.8549906	0.9082044
KNN	0.8622047	1.0000000	0.8781204	0.9550479	0.9358289	0.9326805	0.8614515	0.9179048
Decision Tree (2 Variables)	0.7784200	1.0000000	0.7812500	0.9061001	0.8954839	0.9143577	0.8568720	0.8760691
Decision Tree (3 variables)	0.7763713	1.0000000	0.8010899	0.8945908	0.8635171	0.9017199	0.8671727	0.8720660
Random Forest (2 variables)	0.9059501	0.9952153	0.9107413	0.9136490	0.9366755	0.9352869	0.8645038	0.9231460

Table 26: Tabulated validation F1 scores (by bean class and macro) of Logistic Regression, KNN, 2-variable Decision Tree (Perimeter and Shape Factor 2), 3-variable Decision Tree (Perimeter, SF2, Eccentricity), and Random Forest (ditto).

Validating the highest-performing models (plus the two-variable Decision Tree for comparison), Random Forest with Perimeter, Shape Factor 2, and Eccentricity produced the best predictive performance in terms of macro F1 score. Interestingly, the 3-variable Random Forest was the only model that did not have perfect prediction for Bombay despite being the highest-performing model overall (in terms of macro F1) and in all bean varieties except Bombay and Sira. KNN provided the second best overall performance, followed by Logistic Regression, 2-variable Decision Tree, then 3-variable Decision Tree.

For Logistic Regression, KNN, and Random Forest, the weakest points were in Barbunya and Sira. Meanehile, the two weakest predictions in the two Decision Trees are in Barbunya and Cali. This is in line with discoveries from the exploratory analysis, where scatterplots generated from the starting variable pairs had large overlaps in Barbunya/Cali and Sira/Seker. As expected from the scatterplots (where Bombay consistently formed a cluster distant from all other classes), Bombay had the best predictions in all models tabulated.

Discussion

Summary of Results

The training and validation runs both revealed that Random Forest (with predictors Perimeter, Shape Factor 2, and Eccentricity) produced the highest F1 scores, followed by KNN, Logistic Regression, then Decision Tree. The addition of Eccentricity in the Decision Tree as a third predictor resulted in a higher training macro F1 score than just two predictors, adding Major Axis Length, or adding both Eccentricity and Major Axis Length; however, in validation the Decision Tree with two variables returned a higher macro F1 score than the 3-variable tree, implying that the 3-variable tree's superior performance during training might have been caused by overfitting.

The choice of the initial predictor pairs (a "group 1" variable paired with another variable outside the project-assigned group) during exploratory analysis influenced the performance of all machine learning models built in this project. Although the two "group 1" predictors chosen (Perimeter and Equivalent Diameter) had separate IQRs for most classes with less outliers than most other predictors (see boxplots from Introduction section), both of them had poor resolution between Barbunya/Cali (through the IQRs) and Sira/Seker (through the outliers). Combining those "group 1" predictors with other predictors that have better resolution between those classes (such as Shape Factor 2) mitigated the weakness somewhat, but still resulted in the class-specific lowered performances seen in Table 26.

Limitations and Potential Future Solutions

Perimeter Combinations

The choice of the initial predictor pair not only influenced model performance directly, but also determined what additional predictors could be added for higher-dimensional algorithms since many of the predictors in the dataset

have shared terms in their derivation and therefore exhibit multicollinearity. With each predictor chosen, the list of allowed additional predictors for model-building narrowed; this was especially true for Shape Factor 2 (Minor Axis Length divided by Area), as many other dataset predictors were derived in relation to the area of each bean (see Introduction section and original peper by Koklu and Ozkan (2020) for formulae). If time permits, future attempts at similar problems could experiment with more variable combinations; however, depending on the dataset the amount of suitable predictor combinations (no multicollinearity and capable of resolving all outcome classes) may be similarly limited as the one used in this project.

Statistical Significance of the Performance Evaluations

During algorithm training, it was found that many of the models (Random Forest/KNN, KNN/Logistic Regression, and the 2/3 variable Decision Trees) had standard deviations large enough for the training mF1s to overlap by adding and subtracting by their standard deviations (see Results section; not tabulated, but individual mF1s and SDs available). With such standard deviation values, it goes without calculating 95 % confidence intervals (which is +/- $1.96/\sqrt{10}$ times the standard deviation, with $n = 10$ due to there being 10 folds and therefore 10 results to average) that statistically significant performance differences could not be determined from the training phase alone.

To increase model quality and decrease prediction variance during training, the most obvious solution would be a more generous training/validation dataset split in favor of the training set, using a ratio of 90:10 or perhaps even higher. However, as mentioned in the Methods section, overly high training/validation ratios would increase the variance of validation performance metrics instead, also hindering a proper assessment of model performances. To ensure a balance between training and validation precisions by examining them both, future machine learning projects could use `caret::train` or an equivalent function to perform another k-fold cross-validation for each validation, but with the tuning hyperparameters fixed at the combinations optimized from the training phase. Increasing the number of folds for cross-validation is another way to reduce variance while also improving model quality, although some experimentation on the optimal fold number would have to be done as increasing fold numbers has diminishing returns. For the Random Forests algorithm in particular, more trees could be grown during training, but this is a computationally costly decision that could only be feasible if subsequent projects make use of computers with more memory and higher processing speeds.

Finally, the above options for decreasing variance and improving performance could be combined with other algorithms. The Support Vector Machine (SVM), used in the original paper by Koklu and Ozkan (2020), is a classification algorithm that searches for optimal hyperplanes that can divide the data into groups, corresponding to different outcome classes. Meanwhile, Singular Vector Decomposition (SVD) is a matrix factorization technique that uses linear algebra to break down a data matrix into three simpler matrices to reveal latent features that are then combined to make predictions. Principle Component Analysis (PCA), another matrix factorization technique, analyzes predictor covariances and reduces high-dimensional data down to a smaller selection of principle components that explain the maximum possible variance in the original data.

Practical Implications of Results and Possible Future Directions

Refining means to classify large volumes of crops has many applications both on the marketing and agricultural side of the food economy. Koklu and Ozkan (2020) stated in their original paper that their models would be useful as a method of high-throughput quality assessment for planting and marketing alike. Beyond those applications, further improvements on projects such as this one could also be adapted with additional types of predictors and/or other crops for identifying new traits or metrics that are more efficient for crop yield prediction, more effective phenotyping of various crop cultivars for selective breeding purposes, and envirotyping crops by modeling interactions between genotype, phenotype, and growing conditions both biotic and abiotic over time. As world population density grows and climate change continues to cause degradation of existing arable land, all of the above applications will prove important for farmers of the present and future.

Citations

All referenced sources have been enclosed in links within the report document, and listed below:

Machine Learning: How a Game of Checkers is Changing Agriculture from Bayer Global.

The Professional Certificate in Data Science Program by Harvard on EdX, taught by Dr. Rafael Irizarry.

The UCI Machine Learning Repository - Main Page.

The UCI Repository Page for the Dry Beans Dataset

Multiclass Classification of Dry Beans Using Computer Vision and Machine Learning Techniques, scientific article by Murat Koklu and Ilker Ali Ozkan (2020).

Classification of Rice Grain Varieties Using Two Artificial Neural Networks (MLP and neuro-fuzzy), scientific article by Alireza Pazoki et al. (2014).

Definition of Eccentricity from Cuemath.

Definition of Conic Sections from Cuemath.

Moment of Inertia of an Ellipse from BYJU's.

Eccentricity of Ellipse from Cuemath.

Foci of Ellipse from Cuemath.

Directrix of Ellipse from Cuemath.

Circumscribed Rectangle, or Bounding Box from Math Open Reference.

Convex Hull from Wolfram Mathworld

The Curse of Dimensionality in Classification by Vincent Spruyt, for Computer Vision for Dummies.

32.10.1 - The Curse of Dimensionality by Dr. Irizarry.

Linear Regression Analysis – 3 Common Causes of Multicollinearity and What Do to About Them by Karen Grace-Martin from The Analysis Factor.

The caret::train documentation from RDocumentation.org.

How, When, and Why Should You Normalize / Standardize / Rescale Your Data?, blog post by the Editorial Team at Towards AI

Train, Validation, Test Split for Machine Learning by Jacob Solawetz for Roboflow.

Train Test Validation Split: How To & Best Practices 2022 by Pragati Baheti for V7.

Splitting a Dataset into Train and Test Sets by A. Aylin Tokuç for Baeldung CS.

Metrics to Evaluate Your Classification Model to Take the Right Decisions by Sumeet Kumar Argawal for Analytics Vidhya.

8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset by Jason Brownlee for Machine Learning Mastery.

Harmonic Mean by Adam Hayes, reviewed by Khadija Khartit, and fact checked by Amanda Jackson for Investopedia.

F-1 Score for Multiclass Classification from Baeldung CS.

What is a Confusion Matrix in Machine Learning by Jason Brownlee for Machine Learning Mastery.

Embrace Randomness in Machine Learning by Jason Brownlee for Machine Learning Mastery.

A Gentle Introduction to k-fold Cross-Validation by Jason Brownlee for Machine Learning Mastery.

How to Configure k-Fold Cross-Validation by Jason Brownlee for Machine Learning Mastery.

Logistic Regression for Machine Learning by Jason Brownlee for Machine Learning Mastery.

Odds Ratio on Psych Scene Hub.

Penalized Logistic Regression Essentials in R: Ridge, Lasso and Elastic Net by kassambara on STHDA.

What Is K-Nearest Neighbor? An ML Algorithm to Classify Data by Amal Joby for Learn Hub.

Euclidean Distance Formula from Cuemath.

Decision Tree Algorithm, Explained by Nagesh Singh Chauhan for KDnuggets.

List of Available caret Models by Max Kuhn.

Decision Tree Tutorial from Learn by Marketing.

Documentation for rpart.control from RDocumentation.org.

Random Forest explanation from IBM Cloud Education.

PDF of the ranger Documentation by Marvin N. Wright, Stefan Wager, and Philipp Probst.

Multiple-k: Picking the Number of Folds for Cross-Validation by Ludvig Renbo Olsen on cran.r-project.org.

Support Vector Machines: A Simple Explanation by Noel Bambrick for KDnuggets.

The Netflix Prize and Singular Value Decomposition, course material for the New Jersey Institute of Technology.

A Step-by-Step Explanation of Principal Component Analysis (PCA) by Zakaria Jaadi for Built In.

Application of Machine Learning Algorithms in Plant Breeding: Predicting Yield From Hyperspectral Reflectance in Soybean, scientific article by Mohsen Yoosefzadeh-Najafabadi et al. (2021).

Phenotyping: Using Machine Learning for Improved Pairwise Genotype Classification Based on Root Traits by Jiangsan Zhao et al. (2016).

Envirotyping for Deciphering Environmental Impacts on Crop Plants, scientific article by Yunbi Xu (2016).