# PROJECT: CLASSIFICATION WITH DRY BEAN DATASET

*Murat Ulcay*

## 1. INTRODUCTION

This project focuses on a subset of machine learning: The subdomain called *supervised learning* focuses on training a machine to learn from prior examples. When the concept to be learned is a set of *categories*, the task is called **classification**.

From identifying different types of species, predicting the fraudulent transactions, or detecting whether an image contains a number between 1 to 10, classification tasks are diverse yet common.

### a. Data Exploration

The research problem is this project deals with the classification of the dry beans into seven different classes. The dataset for the models to be developed for this classification is taken from UCI Machine Learning Repository: "Dry Bean Dataset".

Images of 13,611 grains of 7 different registered dry beans were taken with a high-resolution camera. A total of 16 features; 12 dimensions and 4 shape forms, were obtained from the grains.

```r
# Loading `Dry_Bean_Dataset.csv`:
beans <- read.csv("https://github.com/ulcaymurat/data/raw/main/Dry_Bean_Dataset.csv")
```

```r
# Displaying the structure of the dataset:
str(beans)
```

```
## 'data.frame':    13611 obs. of  17 variables:
##  $ Area           : int  28395 28734 29380 30008 30140 30279 30477 30519 30685 30834 ...
##  $ Perimeter      : num  610 638 624 646 620 ...
##  $ MajorAxisLength: num  208 201 213 211 202 ...
##  $ MinorAxisLength: num  174 183 176 183 190 ...
##  $ AspectRation   : num  1.2 1.1 1.21 1.15 1.06 ...
##  $ Eccentricity   : num  0.55 0.412 0.563 0.499 0.334 ...
##  $ ConvexArea     : int  28715 29172 29690 30724 30417 30600 30970 30847 31044 31120 ...
##  $ EquivDiameter  : num  190 191 193 195 196 ...
##  $ Extent         : num  0.764 0.784 0.778 0.783 0.773 ...
##  $ Solidity       : num  0.989 0.985 0.99 0.977 0.991 ...
##  $ roundness      : num  0.958 0.887 0.948 0.904 0.985 ...
##  $ Compactness    : num  0.913 0.954 0.909 0.928 0.971 ...
##  $ ShapeFactor1   : num  0.00733 0.00698 0.00724 0.00702 0.0067 ...
##  $ ShapeFactor2   : num  0.00315 0.00356 0.00305 0.00321 0.00366 ...
##  $ ShapeFactor3   : num  0.834 0.91 0.826 0.862 0.942 ...
##  $ ShapeFactor4   : num  0.999 0.998 0.999 0.994 0.999 ...
##  $ Class          : Factor w/ 7 levels "BARBUNYA","BOMBAY",..: 6 6 6 6 6 6 6 6 6 6 ...
```

The dataset has one factor label variable: `Class`. Its values are the types of beans and shown in the output of the code below:

```r
# Printing the levels of the factor variable: Class
levels(beans$Class)
```

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

These seven classes (types) of beans have 16 features specified below:

1. Area (A): The area of a bean zone and the number of pixels within its boundaries.
2. Perimeter (P): Bean circumference is defined as the length of its border.
3. Major axis length (L): The distance between the ends of the longest line that can be drawn from a bean.
4. Minor axis length (l): The longest line that can be drawn from the bean while standing perpendicular to the main axis.
5. Aspect ratio (K): Defines the relationship between L and l.
6. Eccentricity (Ec): Eccentricity of the ellipse having the same moments as the region.
7. Convex area (C): Number of pixels in the smallest convex polygon that can contain the area of a bean seed.
8. Equivalent diameter (Ed): The diameter of a circle having the same area as a bean seed area.
9. Extent (Ex): The ratio of the pixels in the bounding box to the bean area.
10. Solidity (S): Also known as convexity. The ratio of the pixels in the convex shell to those found in beans.
11. Roundness (R): Calculated with the following formula: (4piA)/(P^2)
12. Compactness (CO): Measures the roundness of an object: Ed/L
13. ShapeFactor1 (SF1)
14. ShapeFactor2 (SF2)
15. ShapeFactor3 (SF3)
16. ShapeFactor4 (SF4)

### b. Data Preparation

#### i. Missing Values:

`beans` dataset has no missing values:

```
# Checking if data has any missing values:
sum(is.na(beans))
```

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

For the remainder of the paper, features and label variable are separated for analysis into dataset `X` and dataset `y` respectively.

```
# Dividing `beans` dataset into features (X) and labels (y) datasets:
X <- beans[1:16]
y <- as.data.frame(beans[17])
```

#### ii. Normalization

Data preprocessing comes after cleaning up the data and after doing some exploratory analysis to understand dataset. After understanding the dataset, some idea about how to model the data is generally formed. Machine learning models in R may require transformation of the variables in the dataset.

Data normalization, also called "feature scaling", is an important step in data preprocessing pipeline. Although it is not always needed, most of the times it is beneficial to any machine learning model. Decision trees, for example, can deal quite well with features having dissimilar and disproportionate scales, but this is not the case for the majority of the machine learning models used in this paper, such as Support Vector Machines, K-nearest neighbors, Logistic Regression, Neural Networks. It is therefore a good practice to consider normalizing data before passing it on to other components in a machine learning pipeline.

We use Z-score normalization which is more robust to outliers but produces normalized values in different scales.

```
# Printing the `summary` of the features dataset:
summary(X)
```

```
##       Area          Perimeter      MajorAxisLength MinorAxisLength
##  Min.   : 20420   Min.   : 524.7   Min.   :183.6   Min.   :122.5
##  1st Qu.: 36328   1st Qu.: 703.5   1st Qu.:253.3   1st Qu.:175.8
##  Median : 44652   Median : 794.9   Median :296.9   Median :192.4
##  Mean   : 53048   Mean   : 855.3   Mean   :320.1   Mean   :202.3
##  3rd Qu.: 61332   3rd Qu.: 977.2   3rd Qu.:376.5   3rd Qu.:217.0
##  Max.   :254616   Max.   :1985.4   Max.   :738.9   Max.   :460.2
##   AspectRation    Eccentricity      ConvexArea     EquivDiameter
##  Min.   :1.025   Min.   :0.2190   Min.   : 20684   Min.   :161.2
##  1st Qu.:1.432   1st Qu.:0.7159   1st Qu.: 36714   1st Qu.:215.1
##  Median :1.551   Median :0.7644   Median : 45178   Median :238.4
##  Mean   :1.583   Mean   :0.7509   Mean   : 53768   Mean   :253.1
##  3rd Qu.:1.707   3rd Qu.:0.8105   3rd Qu.: 62294   3rd Qu.:279.4
##  Max.   :2.430   Max.   :0.9114   Max.   :263261   Max.   :569.4
##      Extent          Solidity        roundness       Compactness
##  Min.   :0.5553   Min.   :0.9192   Min.   :0.4896   Min.   :0.6406
##  1st Qu.:0.7186   1st Qu.:0.9857   1st Qu.:0.8321   1st Qu.:0.7625
##  Median :0.7599   Median :0.9883   Median :0.8832   Median :0.8013
##  Mean   :0.7497   Mean   :0.9871   Mean   :0.8733   Mean   :0.7999
##  3rd Qu.:0.7869   3rd Qu.:0.9900   3rd Qu.:0.9169   3rd Qu.:0.8343
##  Max.   :0.8662   Max.   :0.9947   Max.   :0.9907   Max.   :0.9873
##   ShapeFactor1       ShapeFactor2        ShapeFactor3     ShapeFactor4
##  Min.   :0.002778   Min.   :0.0005642   Min.   :0.4103   Min.   :0.9477
##  1st Qu.:0.005900   1st Qu.:0.0011535   1st Qu.:0.5814   1st Qu.:0.9937
##  Median :0.006645   Median :0.0016935   Median :0.6420   Median :0.9964
##  Mean   :0.006564   Mean   :0.0017159   Mean   :0.6436   Mean   :0.9951
##  3rd Qu.:0.007271   3rd Qu.:0.0021703   3rd Qu.:0.6960   3rd Qu.:0.9979
##  Max.   :0.010451   Max.   :0.0036650   Max.   :0.9748   Max.   :0.9997
```

```r
# Calculating and printing the variances of the features dataset variables:
diag(var(X))
```

```
##            Area       Perimeter MajorAxisLength MinorAxisLength
##    8.599026e+08    4.592007e+04    7.343494e+03    2.022309e+03
##    AspectRation    Eccentricity      ConvexArea   EquivDiameter
##    6.085026e-02    8.464324e-03    8.865456e+08    3.501932e+03
##          Extent        Solidity       roundness     Compactness
##    2.409471e-03    2.171913e-05    3.542617e-03    3.808552e-03
##    ShapeFactor1    ShapeFactor2    ShapeFactor3    ShapeFactor4
##    1.272380e-06    3.550668e-07    9.800238e-03    1.906595e-05
```

We can see from the code outputs above that the scale of means and variances of the features are so different calling for a normalization in the data.

```r
# Normalizing (scaling) the features dataset (X):
X_norm <- as.data.frame(scale(X))

# Calculating and printing the summary statistics of the normalized features dataset
# (X_norm):
summary(X_norm)
```

```
##       Area            Perimeter        MajorAxisLength   MinorAxisLength
##  Min.   :-1.1127   Min.   :-1.5425   Min.   :-1.5933   Min.   :-1.7736
##  1st Qu.:-0.5702   1st Qu.:-0.7082   1st Qu.:-0.7800   1st Qu.:-0.5876
##  Median :-0.2863   Median :-0.2816   Median :-0.2714   Median :-0.2188
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
```

```
##  3rd Qu.: 0.2825   3rd Qu.: 0.5690   3rd Qu.: 0.6576   3rd Qu.: 0.3282
##  Max.   : 6.8738   Max.   : 5.2736   Max.   : 4.8862   Max.   : 5.7355
##   AspectRation      Eccentricity       ConvexArea      EquivDiameter
##  Min.   :-2.2636   Min.   :-5.7819   Min.   :-1.1111   Min.   :-1.5516
##  1st Qu.:-0.6119   1st Qu.:-0.3801   1st Qu.:-0.5728   1st Qu.:-0.6421
##  Median :-0.1302   Median : 0.1472   Median :-0.2885   Median :-0.2472
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 0.5021   3rd Qu.: 0.6475   3rd Qu.: 0.2863   3rd Qu.: 0.4458
##  Max.   : 3.4339   Max.   : 1.7448   Max.   : 7.0359   Max.   : 5.3451
##      Extent           Solidity          roundness         Compactness
##  Min.   :-3.9607   Min.   :-14.5689   Min.   :-6.4460   Min.   :-2.5811
##  1st Qu.:-0.6336   1st Qu.: -0.3160   1st Qu.:-0.6920   1st Qu.:-0.6059
##  Median : 0.2063   Median :  0.2446   Median : 0.1659   Median : 0.0229
##  Mean   : 0.0000   Mean   :  0.0000   Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 0.7562   3rd Qu.:  0.6159   3rd Qu.: 0.7323   3rd Qu.: 0.5575
##  Max.   : 2.3726   Max.   :  1.6167   Max.   : 1.9725   Max.   : 3.0373
##   ShapeFactor1       ShapeFactor2       ShapeFactor3
##  Min.   :-3.35603   Min.   :-1.93292   Min.   :-2.35617
##  1st Qu.:-0.58838   1st Qu.:-0.94387   1st Qu.:-0.62863
##  Median : 0.07231   Median :-0.03762   Median :-0.01562
##  Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.00000
##  3rd Qu.: 0.62749   3rd Qu.: 0.76244   3rd Qu.: 0.52948
##  Max.   : 3.44642   Max.   : 3.27086   Max.   : 3.34535
##   ShapeFactor4
##  Min.   :-10.8500
##  1st Qu.: -0.3116
##  Median :  0.3029
##  Mean   :  0.0000
##  3rd Qu.:  0.6457
##  Max.   :  1.0693
```

```r
# Calculating and printing the variances of the normalized features dataset (X_norm):
diag(var(X_norm))
```

```
##            Area       Perimeter MajorAxisLength MinorAxisLength
##               1               1               1               1
##    AspectRation    Eccentricity      ConvexArea   EquivDiameter
##               1               1               1               1
##          Extent        Solidity       roundness     Compactness
##               1               1               1               1
##    ShapeFactor1    ShapeFactor2    ShapeFactor3    ShapeFactor4
##               1               1               1               1
```

As can be seen from the code outputs above, after the normalization operation, every feature in the `X` dataset is standardized with a mean of "zero" and a variance (and also standard deviation) of "one".

### iii. Anomaly Detection:

Dealing with anomalous data points (also known as outliers) is an important step in the data preparation phase. If not properly handled, outliers could skew the analysis and produce misleading conclusions.

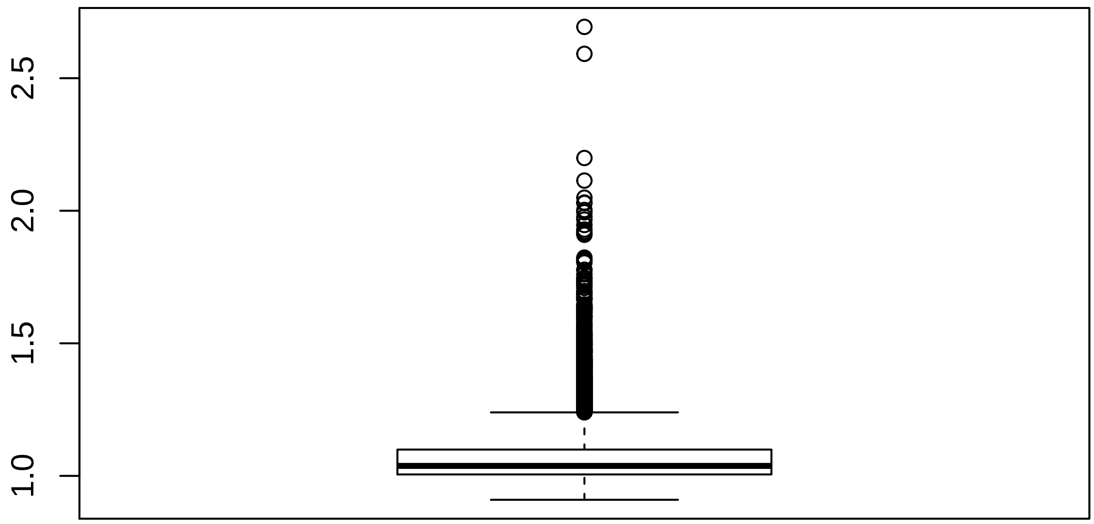
The Local Outlier Factor (LOF) which is an algorithm that measures the local deviation of a data point relative to its neighbors is used in this paper. Outliers are defined as data points with substantially lower density than their neighbors. Each observation receives an LOF score that indicates whether it is deemed to be a regular data point, an inlier or an outlier.

```r
# Creating a copy of `X_norm` dataset:
X_norm1 <- X_norm

# Computing the LOF anomaly score for each data point in `X_norm1` dataset
# using the 7 nearest neighbors and adding it to `X_norm1`:
X_norm1$lof_score <- lof(X_norm, k = 7)
```

In order to detect the outlier LOF scores, a boxplot can help visually:

```r
# Plotting the box_plot of `lof_score` column of `X_norm1` to detect outlier lof scores visually:
boxplot(X_norm1$lof_score)
```




Although the process of detecting outliers requires subjective evaluation at this point, it seems that we can label four datapoints from the dataset as outliers and we can eliminate them from the dataset:

```r
# Creating a mask for the locations of the outliers (i.e. the datapoints
# having the four highest LOF scores):
outlier_mask <- order(X_norm1$lof_score, decreasing = TRUE)[1:4]
```

```r
# Excluding the outliers from both features dataset (X_norm) and label dataset (y):
X_norm <- X_norm[-outlier_mask,]
y <- y[-outlier_mask,]
```

**iv. Training and Test Datasets:**

```r
# Setting the seed:
set.seed(123)

# Creating a mask from `X_norm` for the `training` dataset:
mask = sort(sample(nrow(X_norm), nrow(X_norm)*.7))

# Extracting `training` dataset from `X_norm`:
X_train <- X_norm[mask,]

# Extracting `training` dataset from `y`:
y_train <- y[mask]

# Extracting `test` dataset from `X_norm`:
X_test <- X_norm[-mask,]
```

```r
# Extracting `test` dataset from `y`:
y_test <- y[-mask]

# Creating a dataframe by combining X_train and y_train:
training <- cbind(X_train,y_train)
```

## 2. METHODS

Classification models are a method of high importance used in various fields. In class determination, classification models are used to determine which class the data belongs to. The classification model is a model that works by making predictions. The purpose of the classification is to make use of the common characteristics of the data to parse the data in question.

### a. Algorithms

In this paper, in order to classify beans according to their characteristics, models are developed using the algorithms below:

> k- NN (K Nearest Neighbors),
> NB (Naive Bayes),
> SVM (Support Vector Machine), DT (Decision Tree),
> RF (Random Forest),

The model development is made on the `training` set and model evaluations is made on the `X_test` datasets.

### b. Model Evaluation

The **confusion matrix** for each model is prepared and **accuracy** measure is used to evaluate the models created

### i. Confusion Matrix:

Confusion matrices for each class of beans are below:

```r
download.file(url = "http://veribilim.online/harvardx/confusion.png",
          destfile = "confusion.png",
          mode = 'wb')
knitr::include_graphics(path = "confusion.png")
```

| Confusion Matrix For "Barbunya" Class | | Predicted Bean | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | Barbunya | Bombay | Cali | Dermason | Horoz | Seker | Sira |
| Actual Bean | Barbunya | tp | fn | fn | fn | fn | fn | fn |
| | Bombay | fp | tn | tn | tn | tn | tn | tn |
| | Cali | fp | tn | tn | tn | tn | tn | tn |
| | Dermason | fp | tn | tn | tn | tn | tn | tn |
| | Horoz | fp | tn | tn | tn | tn | tn | tn |
| | Seker | fp | tn | tn | tn | tn | tn | tn |
| | Sira | fp | tn | tn | tn | tn | tn | tn |

| Confusion Matrix For "Bombay" Class | | Predicted Bean | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | Barbunya | Bombay | Cali | Dermason | Horoz | Seker | Sira |
| Actual Bean | Barbunya | tn | fp | tn | tn | tn | tn | tn |
| | Bombay | fn | tp | fn | fn | fn | fn | fn |
| | Cali | tn | fp | tn | tn | tn | tn | tn |
| | Dermason | tn | fp | tn | tn | tn | tn | tn |
| | Horoz | tn | fp | tn | tn | tn | tn | tn |
| | Seker | tn | fp | tn | tn | tn | tn | tn |
| | Sira | tn | fp | tn | tn | tn | tn | tn |

| Confusion Matrix For "Cali" Class | | Predicted Bean | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | Barbunya | Bombay | Cali | Dermason | Horoz | Seker | Sira |
| Actual Bean | Barbunya | tn | tn | fp | tn | tn | tn | tn |
| | Bombay | tn | tn | fp | tn | tn | tn | tn |
| | Cali | fn | fn | tp | fn | fn | fn | fn |
| | Dermason | tn | tn | fp | tn | tn | tn | tn |
| | Horoz | tn | tn | fp | tn | tn | tn | tn |
| | Seker | tn | tn | fp | tn | tn | tn | tn |
| | Sira | tn | tn | fp | tn | tn | tn | tn |

| Confusion Matrix For "Dermason" Class | | Predicted Bean | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | Barbunya | Bombay | Cali | Dermason | Horoz | Seker | Sira |
| Actual Bean | Barbunya | tn | tn | tn | fp | tn | tn | tn |
| | Bombay | tn | tn | tn | fp | tn | tn | tn |
| | Cali | tn | tn | tn | fp | tn | tn | tn |
| | Dermason | fn | fn | fn | tp | fn | fn | fn |
| | Horoz | tn | tn | tn | fp | tn | tn | tn |
| | Seker | tn | tn | tn | fp | tn | tn | tn |
| | Sira | tn | tn | tn | fp | tn | tn | tn |

| Confusion Matrix For "Horoz" Class | | Predicted Bean | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | Barbunya | Bombay | Cali | Dermason | Horoz | Seker | Sira |
| Actual Bean | Barbunya | tn | tn | tn | tn | fp | tn | tn |
| | Bombay | tn | tn | tn | tn | fp | tn | tn |
| | Cali | tn | tn | tn | tn | fp | tn | tn |
| | Dermason | tn | tn | tn | tn | fp | tn | tn |
| | Horoz | fn | fn | fn | fn | tp | fn | fn |
| | Seker | tn | tn | tn | tn | fp | tn | tn |
| | Sira | tn | tn | tn | tn | fp | tn | tn |

| Confusion Matrix For "Seker" Class | | Predicted Bean | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | Barbunya | Bombay | Cali | Dermason | Horoz | Seker | Sira |
| Actual Bean | Barbunya | tn | tn | tn | tn | tn | fp | tn |
| | Bombay | tn | tn | tn | tn | tn | fp | tn |
| | Cali | tn | tn | tn | tn | tn | fp | tn |
| | Dermason | tn | tn | tn | tn | tn | fp | tn |
| | Horoz | tn | tn | tn | tn | tn | fp | tn |
| | Seker | fn | fn | fn | fn | fn | tp | fn |
| | Sira | tn | tn | tn | tn | tn | fp | tn |

| Confusion Matrix For "Sira" Class | | Predicted Bean | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | Barbunya | Bombay | Cali | Dermason | Horoz | Seker | Sira |
| Actual Bean | Barbunya | tn | tn | tn | tn | tn | tn | fp |
| | Bombay | tn | tn | tn | tn | tn | tn | fp |
| | Cali | tn | tn | tn | tn | tn | tn | fp |
| | Dermason | tn | tn | tn | tn | tn | tn | fp |
| | Horoz | tn | tn | tn | tn | tn | tn | fp |
| | Seker | tn | tn | tn | tn | tn | tn | fp |
| | Sira | fn | fn | fn | fn | fn | fn | tp |

In these matrices:

**tp:** True Positive
**tn:** True Negative
**fp:** False Positive
**fn:** False Negative

### ii. Performance Measure - Accuracy:

Each models' predictive ability of each class is evaluated with their **accuracy** metrics:

Accuracy = (tp + tn) / (tp + fp + tn + fn)

## 3. ANALYSIS

### a. k-NN (K Nearest Neighbors)

```
# Applying the k-NN algorithm for classification:
beans_pred_knn <- knn(
                    train = X_train, #training dataset is specified
                    test = X_test, #test dataset is specified
                    cl = y_train #labels for the training data
)

# Calculating and printing out the `confusion_matrix` for knn:
confusion_knn <- table(y_test,beans_pred_knn)
confusion_knn
```

```
##             beans_pred_knn
## y_test      BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
##   BARBUNYA       363      0   32        0     4     6    9
##   BOMBAY           0    149    0        0     0     0    0
##   CALI            20      0  456        0     9     0    4
##   DERMASON         0      0    0      955     7    15   86
##   HOROZ            0      0   17        6   559     0   16
##   SEKER            4      0    1       16     0   554   19
##   SIRA             3      0    2       86    19    18  648
```

### b. NB (Naive Bayes)

```
# Changing the column name of `y_train` in `training_NB`dataframe:
names(training)[names(training) == "y_train"] <- "Class"

# Applying NB algorithm for classification:
model_NB <-
  naive_bayes(
      Class ~ . ,#formula: Class depending on the other features on the
                 #`training_NB` dataframe
      data = training, #dataframe for NB algorithm
      laplace = 1 #using the laplace correction to prevent some potential
                  #outcomes from being predicted to be impossible.
)

# predicting the `y_test` labels using `model_NB`:
beans_pred_nb <- predict(model_NB, X_test)

# Calculating and printing out the `confusion_matrix` for NB:
confusion_nb <- table(y_test,beans_pred_nb)
confusion_nb
```

```
##             beans_pred_nb
## y_test      BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
##   BARBUNYA       328      0   65        0     3     2   16
##   BOMBAY           0    149    0        0     0     0    0
```

```
##    CALI            37      0  445        0     5     0     2
##    DERMASON         0      0    0      929     1    27   106
##    HOROZ            0      0   11        7   569     0    11
##    SEKER            3      0    0        9     0   561    21
##    SIRA             2      0    2       59    20    15   678
```

#### c. SVM (Support Vector Machines)

```
# Applying SVM algorithm for classification:
model_SVM <-
  svm(
    Class ~ . ,#formula: Class depending on the other features on the `training` dataframe
    data = training, #dataframe for algorithm
    type = "C-classification", #for LR
    kernel = "linear", # to build a linear SVM classifier
    scale = FALSE #our data is already scaled (normalized)
)

# predicting the `y_test` labels using `model_SVM`:
beans_pred_svm <- predict(model_SVM, X_test)

# Calculating and printing out the `confusion_matrix` for SVM:
confusion_svm <- table(y_test,beans_pred_svm)
confusion_svm
```

```
##           beans_pred_svm
## y_test     BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
##   BARBUNYA      371      0   28        0     3     4    8
##   BOMBAY          0    148    1        0     0     0    0
##   CALI           13      0  466        0     6     0    4
##   DERMASON        0      0    0      980     3    13   67
##   HOROZ           2      0    8        6   570     0   12
##   SEKER           4      0    0        8     0   568   14
##   SIRA            3      0    0       69    13    13  678
```

#### d. RF (Random Forest)

```
# Applying RF algorithm for classification:
model_RF <-
  ranger(
      Class ~ . ,#formula: Class depending on the other features on the `training` dataframe
      data = training, #dataframe for algorithm
      num.trees = 1000, #number of trees in the random forest model
      respect.unordered.factors = "order",
      seed = 1 #for reproducible results
)

# predicting the `y_test` labels using `model_RF`:
beans_pred_rf <- predict(model_RF, X_test)

# Calculating and printing out the `confusion_matrix` for RF:
confusion_rf <- table(y_test,beans_pred_rf$predictions)
confusion_rf
```

```
##
## y_test     BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
##   BARBUNYA      366      0   30        0     3     8    7
##   BOMBAY          0    149    0        0     0     0    0
##   CALI           13      0  466        0     5     0    5
##   DERMASON        0      0    0      982     6    13   62
##   HOROZ           0      0   16        4   565     0   13
##   SEKER           1      0    0       14     0   566   13
##   SIRA            4      0    1       76    10     9  676
```

## 4. RESULTS

The evaluations and comparisions of the models are performed below with the computations of the accuracy metrics:

```r
# Defining a function to form a named vector of accuracy metric
# of each model for the prediction of the classes:
model_accuracy <- function(x) {

    # Calculating "Accuracy" for the prediction of "Barbunya" class:
    ACC_Barbunya =
      (x[1:1] + sum(x[2:7,2:7])) / length(y_test)

    # Calculating "Accuracy" for the prediction of "Bombay" class:
    ACC_Bombay =
      (x[2:2] + sum(x[3:7,3:7]) + sum(x[3:7,1])+ x[1:1]) / length(y_test)

    # Calculating "Accuracy" for the prediction of "Cali" class:
    ACC_Cali =
      (x[3:3] + sum(x[4:7,4:7]) + sum(x[4:7,1:2])+ sum(x[1:2,1:2])) / length(y_test)

    # Calculating "Accuracy" for the prediction of "Dermason" class:
    ACC_Dermason =
      (x[4:4] + sum(x[5:7,5:7]) + sum(x[5:7,1:3])+ sum(x[1:3,1:3])) / length(y_test)

    # Calculating "Accuracy" for the prediction of "Horoz" class:
    ACC_Horoz =
      (x[5:5] + sum(x[6:7,6:7]) + sum(x[6:7,1:4])+ sum(x[1:4,1:4])) / length(y_test)

    # Calculating "Accuracy" for the prediction of "Seker" class:
    ACC_Seker =
      (x[6:6] + x[7:7] + sum(x[7:7,1:5])+ sum(x[1:5,1:5])) / length(y_test)

    # Calculating "Accuracy" for the prediction of "Sira" class:
    ACC_Sira =
      (x[7:7] + sum(x[1:6,1:6])) / length(y_test)

    # Storing the accuracy results to a vector:
    accuracy <- c(
                  ACC_Barbunya,
                  ACC_Bombay,
                  ACC_Cali,
                  ACC_Dermason,
                  ACC_Horoz,
                  ACC_Seker,
```

```r
                  ACC_Sira
                  )

    # Naming the vector elements accordingly:
    names(accuracy) <- c("Barbunya","Bombay","Cali",
                         "Dermason","Horoz","Seker","Sira")

    return(accuracy)
}

# Calculating the accuracy scores of
# k-NN model for each class and storing them in a named vector:
accuracy_knn <- model_accuracy(confusion_knn)

# Calculating the accuracy scores of
# Naive Bayes model for each class and storing them in a named vector:
accuracy_nb <- model_accuracy(confusion_nb)

# Calculating the accuracy scores of
# Support Vector Machines model for each class and storing them in a named vector:
accuracy_svm <- model_accuracy(confusion_svm)

# Calculating the accuracy scores of
# Random Forest model for each class and storing them in a named vector:
accuracy_rf <- model_accuracy(confusion_rf)
```

```r
# Creating a dataframe from the accuracy scores:
accuracy_model_df <- as.data.frame(
  rbind(accuracy_knn,accuracy_nb,accuracy_svm,accuracy_rf),
  row.names=c("KNN","Naive Bayes","Support Vector Machines","Random Forest"))

# Adding a column showing the average accuracy scores over the classes for each model:
accuracy_model_df["Average_Model"] <- apply(accuracy_model_df, 1,mean)

# Printing the `accuracy_model_df` dataframe:
accuracy_model_df
```

```
##                          Barbunya    Bombay      Cali  Dermason     Horoz
## KNN                     0.9808964 0.9510164 0.8677443 0.7053637 0.8145971
## Naive Bayes             0.9686505 0.9424443 0.8650502 0.7144257 0.8089640
## Support Vector Machines 0.9840803 0.9529758 0.8706833 0.7132011 0.8243938
## Random Forest           0.9838354 0.9517512 0.8674994 0.7097722 0.8243938
##                             Seker      Sira Average_Model
## KNN                     0.6598090 0.7778594     0.8224695
## Naive Bayes             0.6458486 0.7722263     0.8168014
## Support Vector Machines 0.6605437 0.7849620     0.8272629
## Random Forest           0.6615234 0.7864315     0.8264581
```

## CONCLUSION, LIMITATIONS AND SUGGESTIONS FOR FURTHER ANALYSIS

From the summary of the accuracy scores shown above, It can be seen that, the accuracy scores of different models used are very close to each other around 83%. it's somehow inconclusive as to which model has a superior performance over others.

In this paper, the model evaluation is made solely based on the accuracy score. However, depending on the aim of the model and the expectations of the end user, the other performance metrics can also be included for further analysis.

Full set of performance metrics are presented below for reference:

```
download.file(url = "http://veribilim.online/harvardx/performance.png",
          destfile = "performance.png",
          mode = 'wb')
knitr::include_graphics(path = "performance.png")
```

| **Performance Measure** | **Formula** |
|---|---|
| Accuracy | $ACC = \frac{tp + tn}{tp + fp + tn + fn} x100$ |
| Sensivitiy | $TPR = \frac{tp}{tp + fn} x100$ |
| Specificity | $SPC = \frac{tn}{tn + fp} x100$ |
| Precision | $PPV = \frac{tp}{tp + fp} x100$ |
| F1-Score | $F1 = \frac{2tp}{2tp + fp + fn} x100$ |
| Negative Predictive Value | $NPV = \frac{tn}{tn + fn} x100$ |
| False Positive Rate | $FPR = \frac{fp}{tn + fp} x100$ |
| False Discovery Rate | $FDR = \frac{fp}{tp + fp} x100$ |
| False Negative Rate | $FNR = \frac{fn}{tp + fn} x100$ |

Moreover, hyperparameter tuning techniques are not extensively used in this study. By explicitly including these techniques, the performance of the models may be increased and other insights may be extracted from the analyses.