CS-GY 6923 Machine Learning

Professor: Dr. Raman Kannan

Homework 1:
Exploratory Data Analysis

Hansheng Li

Contents

1. Overview of the Data set

The data set is called “Dry_Bean_Dataset.csv” and was taken from the UCI repository: https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset#

This dataset Seven different types of dry beans were used in this research, taking into account the features such as form, shape, type, and structure by the market situation. Here we are going to distinguish seven different registered varieties of dry beans with similar features in order to obtain uniform seed classification.

The data set consists of 16 features (independent variables), 1 dependent variable and 13611 observations. The dependent variable has seven categorical which is Seker, Barbunya, Bombay, Cali, Dermosan, Horoz and Sira. hence, this is a classification problem. And since the number of classes is larger than two, it is a multi-class classification. Before loading the data file into R, labels were assigned to the variables, and the file was saved as a csv file. The label” Class” was allocated to the class variable.

2. Loading the Data

The data set, containing all the variables and labels, was saved as a csv file named ‘ Dry_Bean_Dataset.csv’ and it was loaded into a variable called data.

```
> loadData = function(csvfile)
{ read.csv(csvfile,head=T,sep=',',stringsAsFactors=F)} # Function to load the data

> data = loadData('Dry_Bean_Dataset.csv') # Load the data > data = loadData('Dry_Bean_Dataset.csv') # Load the data
```

We then perform few steps to inspect the data to make sure that it was loaded properly. The dimensions of the data frame were found to be 13611 * 17, which corresponds to 13611 rows (number of observations) and 17 columns (number of variables). The number of columns includes the 16 features plus 1 column for the class variable. We also check that the labels are assigned correctly using the names(data) command. We can also see that the number 17 column is the class variable, and its name is " Class".

```
> head(data)[,1:6]
  Area Perimeter MajorAxisLength MinorAxisLength AspectRation
1 28395   610.291       208.1781       173.8887     1.197191
2 28734   638.018       200.5248       182.7344     1.097356
3 29380   624.110       212.8261       175.9311     1.209713
4 30008   645.884       210.5580       182.5165     1.153638
5 30140   620.134       201.8479       190.2793     1.060798
6 30279   634.927       212.5606       181.5102     1.171067
  Eccentricity
1    0.5498122
2    0.4117853
3    0.5627273
4    0.4986160
5    0.3336797
6    0.5204007
> dim(data) # Check the number of rows and columns of the data
[1] 13611    17
> names(data) # Check the names of each column (features and class variable)
 [1] "Area"           "Perimeter"      "MajorAxisLength"
 [4] "MinorAxisLength" "AspectRation"    "Eccentricity"
 [7] "ConvexArea"     "EquivDiameter"   "Extent"
[10] "Solidity"       "roundness"       "Compactness"
[13] "ShapeFactor1"    "ShapeFactor2"    "ShapeFactor3"
[16] "ShapeFactor4"    "Class"
> length(names(data)) # Double check that the length of "names" matches the
number of columns
```

```
[1] 17
> which(names(data)=='Class') # Position of the class variable column

[1] 17
```

3. Exploratory Data Analysis

Before we train the data, we need to understand the data and perform some analyses in order to ensure that it is consistent with the classifier to be used. Here we have 13611 observations and 17 features; hence, this is considered to be a large data set as it is, which requires heavy computations and not all classifiers might be able to handle depending on the resources available. Let's first find the spectral count of the class variable "Class", which will give us the number of classes, class names, and the number of observations for each class.

```
> # Find spectral count
> TBL=table(data$Class)
> names(TBL)
[1] "BARBUNYA" "BOMBAY"   "CALI"     "DERMASON" "HOROZ"    "SEKER"
[7] "SIRA"
> as.numeric(TBL)
[1] 1322  522 1630 3546 1928 2027 2636
> # Determine if binary or multi-class classification
> print(ifelse(length(TBL)==2, "Binary Classification", "MultiClass Classification"))
[1] "MultiClass Classification"
```

The output of the table(data$Class) command shows that we have 7 classes, these classes are Seker, Barbunya, Bombay, Cali, Dermosan, Horoz and Sira. And as mentioned earlier, since the number of classes are larger than 2, we are dealing with Multi-class classification.

Now let's check the class are balanced or not. If a class have 20-40% of the data set, it will be a Mild imbalance, if a class have 1-20% of the data set, it

will be a Moderate imbalance, If a class have <1% of the data set, it will be an Extreme imbalance. Since we have seven class, which means if the number of data = number of observation / 7, then it is 100%

```
for(i in 1:nrow(TBL)){
 temp = TBL[i]/(nrow(data)/7)
 if(temp < 0.4){
  if(temp > 0.2){
   cat(names(TBL)[i], temp*100, "% Mild Imbalanced \n")
  } else if (temp > 0.01) {
   cat(names(TBL)[i], temp*100, "% Moderate Imbalanced \n")
  } else {
   cat(names(TBL)[i],temp*100, "% Extreme Imbalanced \n")
  }
 } else {
  cat(names(TBL)[i],temp*100, "% No Imbalanced \n")
 }
}
```

```
BARBUNYA 67.98913 % No Imbalanced
BOMBAY 26.84593 % Mild Imbalanced
CALI 83.82926 % No Imbalanced
DERMASON 182.3672 % No Imbalanced
HOROZ 99.1551 % No Imbalanced
SEKER 104.2466 % No Imbalanced
SIRA 135.5668 % No Imbalanced
```

As the result we can see the data is have a fairly balanced, only BOMBAY 26.84593 % is Mild Imbalanced which should not be a big problem.

Now let see how many empty or null value in each column, print out the column name and number of missing + null value.

```
> # Check missing value
> cont = 0
> for (i in 1:ncol(data)){
```

```
+   len = length(which(data[i] == ""))+sum(is.na(data[i]))
+   if (len > 0){
+   cont = cont + 1
+   }
+   cat(i, names(data[i]), len, '\n')
+ }
1 Area 0
2 Perimeter 0
3 MajorAxisLength 0
4 MinorAxisLength 0
5 AspectRation 0
6 Eccentricity 0
7 ConvexArea 0
8 EquivDiameter 0
9 Extent 0
10 Solidity 0
11 roundness 0
12 Compactness 0
13 ShapeFactor1 0
14 ShapeFactor2 0
15 ShapeFactor3 0
16 ShapeFactor4 0
17 Class 0
> cat("Total:", cont)
Total: 0
```

Here we can see our data is prefect clean with zero missing or Null value.

Then we need the understand attribute Information:

1.) Area (A): The area of a bean zone and the number of pixels within its boundaries.
2.) Perimeter (P): Bean circumference is defined as the length of its border.
3.) Major axis length (L): The distance between the ends of the longest line that can be drawn from a bean.
4.) Minor axis length (l): The longest line that can be drawn from the bean while standing perpendicular to the main axis.

5.) Aspect ratio (K): Defines the relationship between L and l.
6.) Eccentricity (Ec): Eccentricity of the ellipse having the same moments as the region.
7.) Convex area (C): Number of pixels in the smallest convex polygon that can contain the area of a bean seed.
8.) Equivalent diameter (Ed): The diameter of a circle having the same area as a bean seed area.
9.) Extent (Ex): The ratio of the pixels in the bounding box to the bean area.
10.)Solidity (S): Also known as convexity. The ratio of the pixels in the convex shell to those found in beans.
11.)Roundness (R): Calculated with the following formula: (4piA)/(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)
17.)Class (Seker, Barbunya, Bombay, Cali, Dermosan, Horoz and Sira)

Next, we need to check if standardization or normalization of the data is required. The standard deviation and mean of each feature column were calculated, and the results are plotted in Fig. 1. The figure shows the number one and number seven feature have extreme high value in both charts, Which is because of area of the bean. Since it affects our observation of the data, we temporarily remove them to see the effect, plotted in Fig. 2. Now we can see other than length feature, the standard deviation and mean are almost zero which is particularly good. Then we try to observe the range of each feature, plotted in Fig. 3. The result shows other than length, feature range is between 0 to 1.

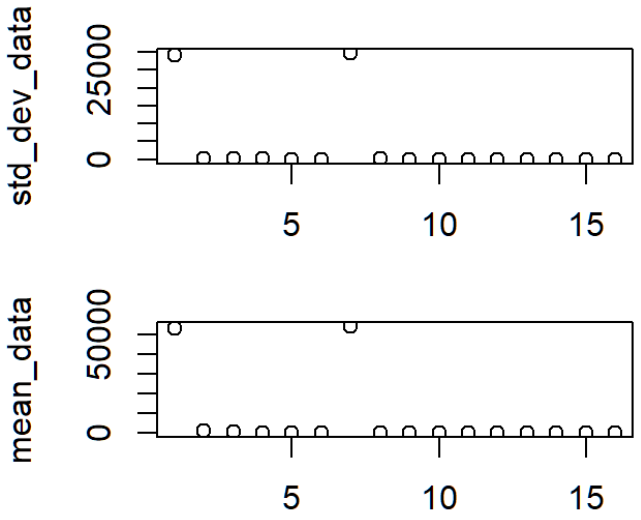

Figure 1: Calculated standard deviations (top figure) and means (bottom figures) of each feature column.

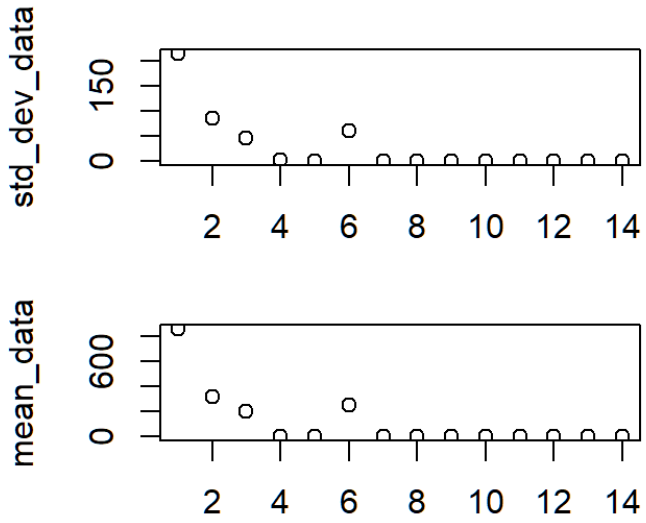

Figure 2: Calculated standard deviations (top figure) and means (bottom figures) of each feature column (removed Area feature).

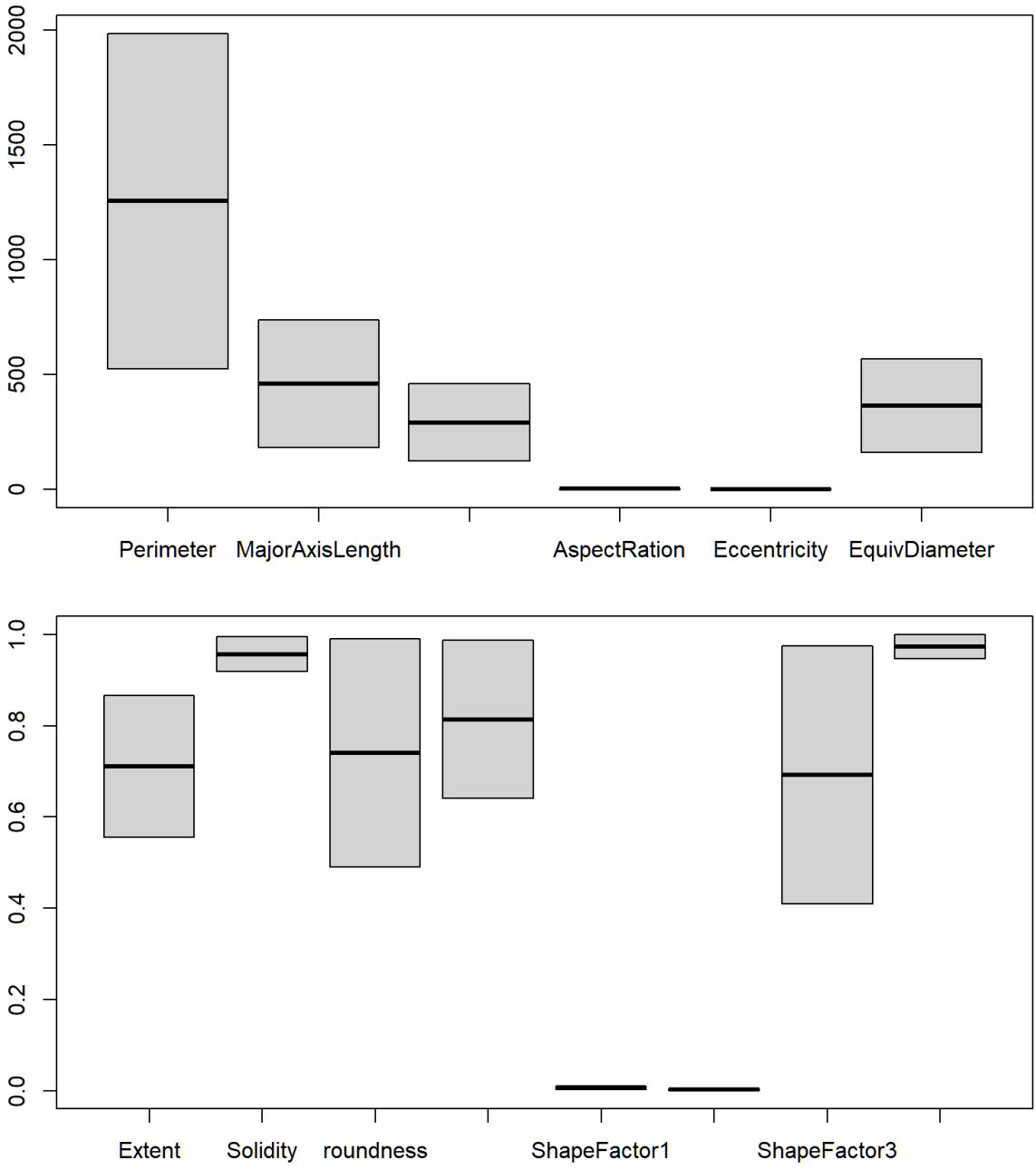

Figure 3: Boxplots showing the ranges of each feature column. Above is data[,-c(1,7,17,9:16)], bottom is data[,9:16].

Then, we inspect the data for constant and correlated predictors (features). If two predictors are highly correlated, then collinearity exists we can keep one of them and discard the other since only one of those predictors provides sufficient information to have a proper model fit.

Here I run plot(data[,1:8]) and plot(data[,9:16]) in Figure 4 and Figure 5 which is a scatter plots between each pair of the 1 to 8 features and 9 to 16 features. By visual inspection, we can tell that there is a strong correlation between features, some are almost linear, some are clustered very uniformly.

Then I tried to run corrplot (cor(data[,-17]),method="circle") to find correlation matrix, A correlation coefficient of -1 indicates that the predictors are uncorrelated, and a value of 1 indicates that they are perfectly correlated. As you can see the higher the correlation, the bluer the circle, the lower the correlation, the redder the circle will be. It is interesting to find out for feature 1 to 8 have a good correlation, for feature 9 to 16 have a good correlation, but for 1 to 8 features compare to 9 to 6 features have very low correlation.

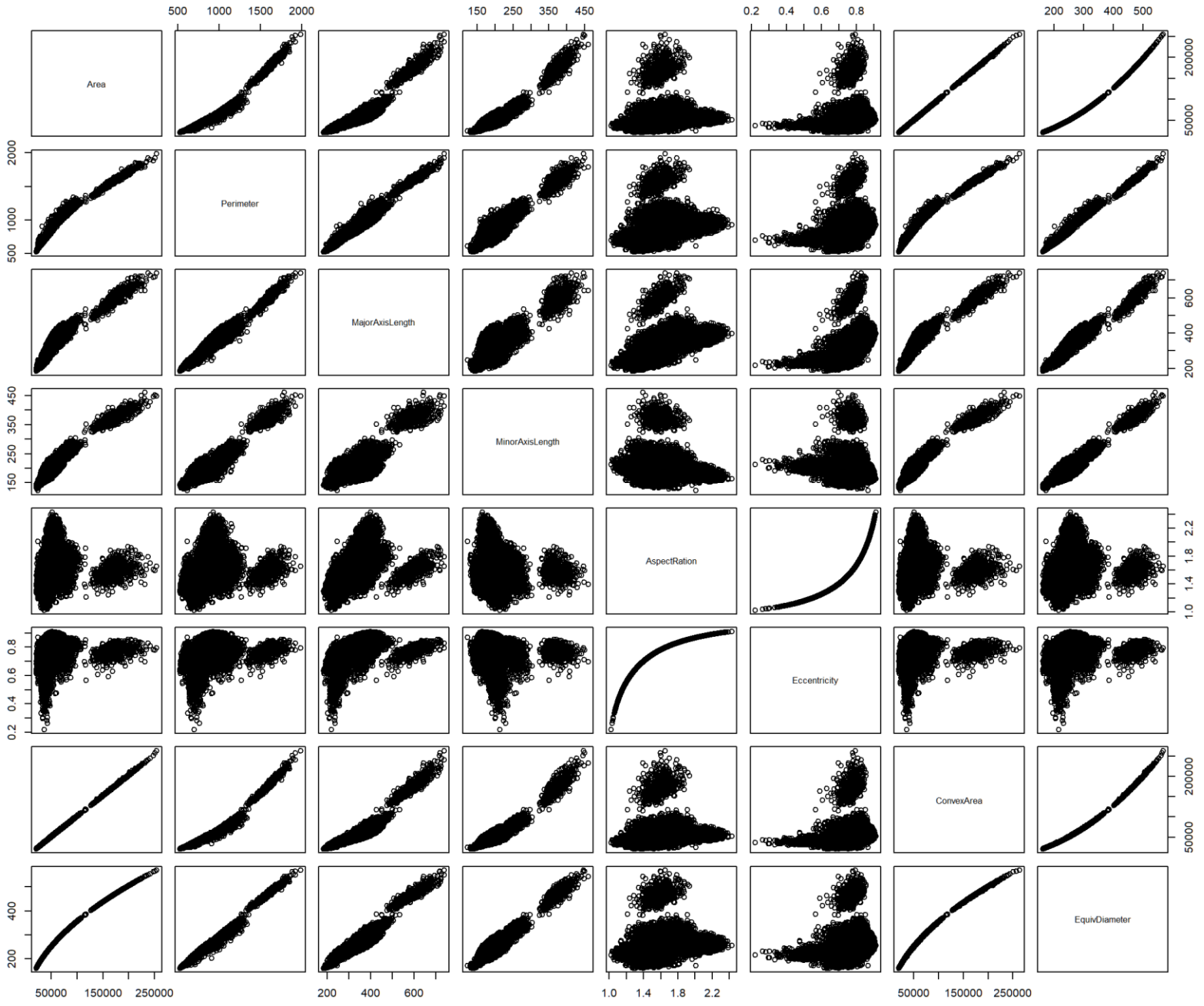

Figure 4: Scatter plots generated by the plot(data[,1:8]) command.

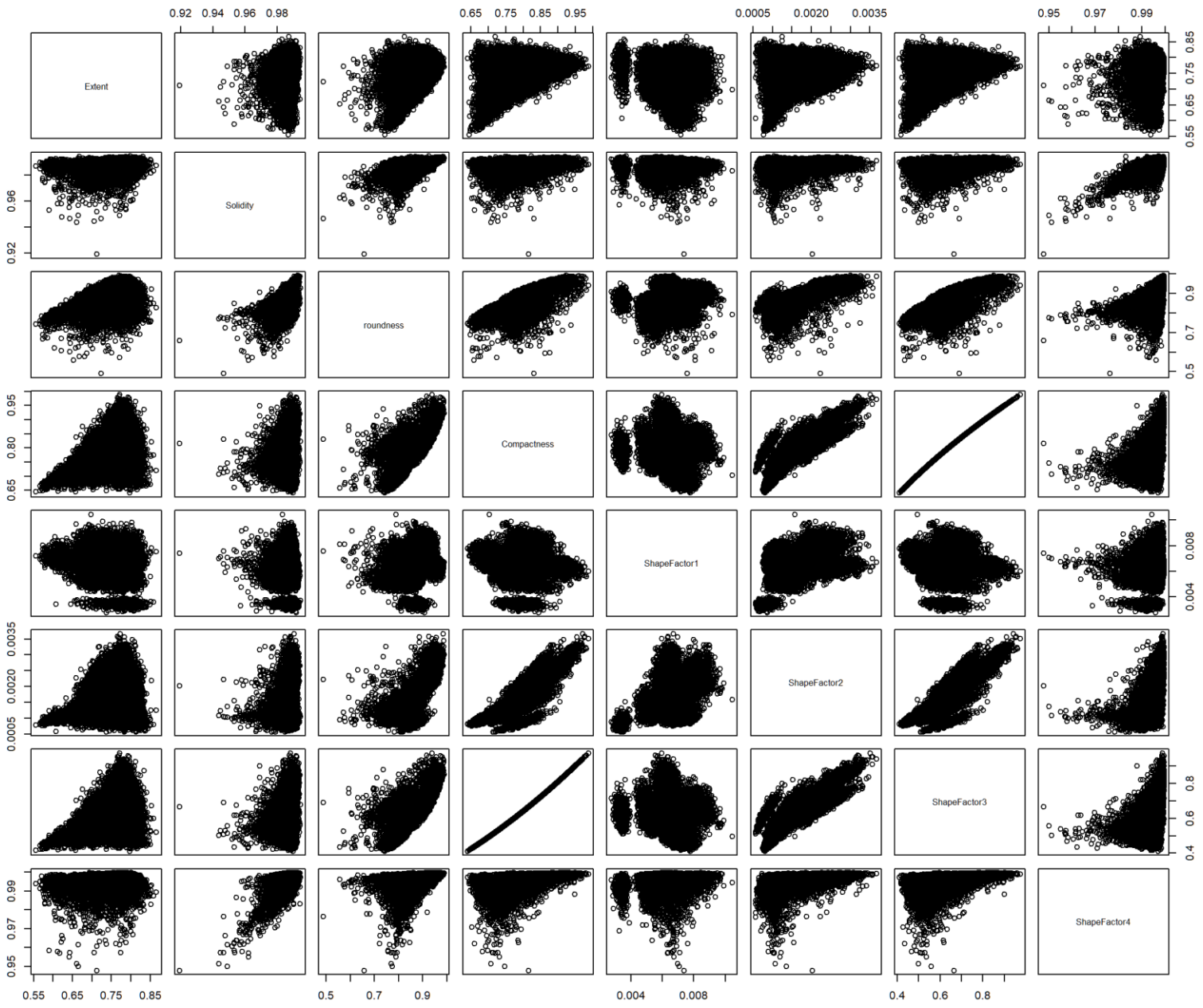

Figure 5: Scatter plots generated by the plot(data[,9:16]) command.

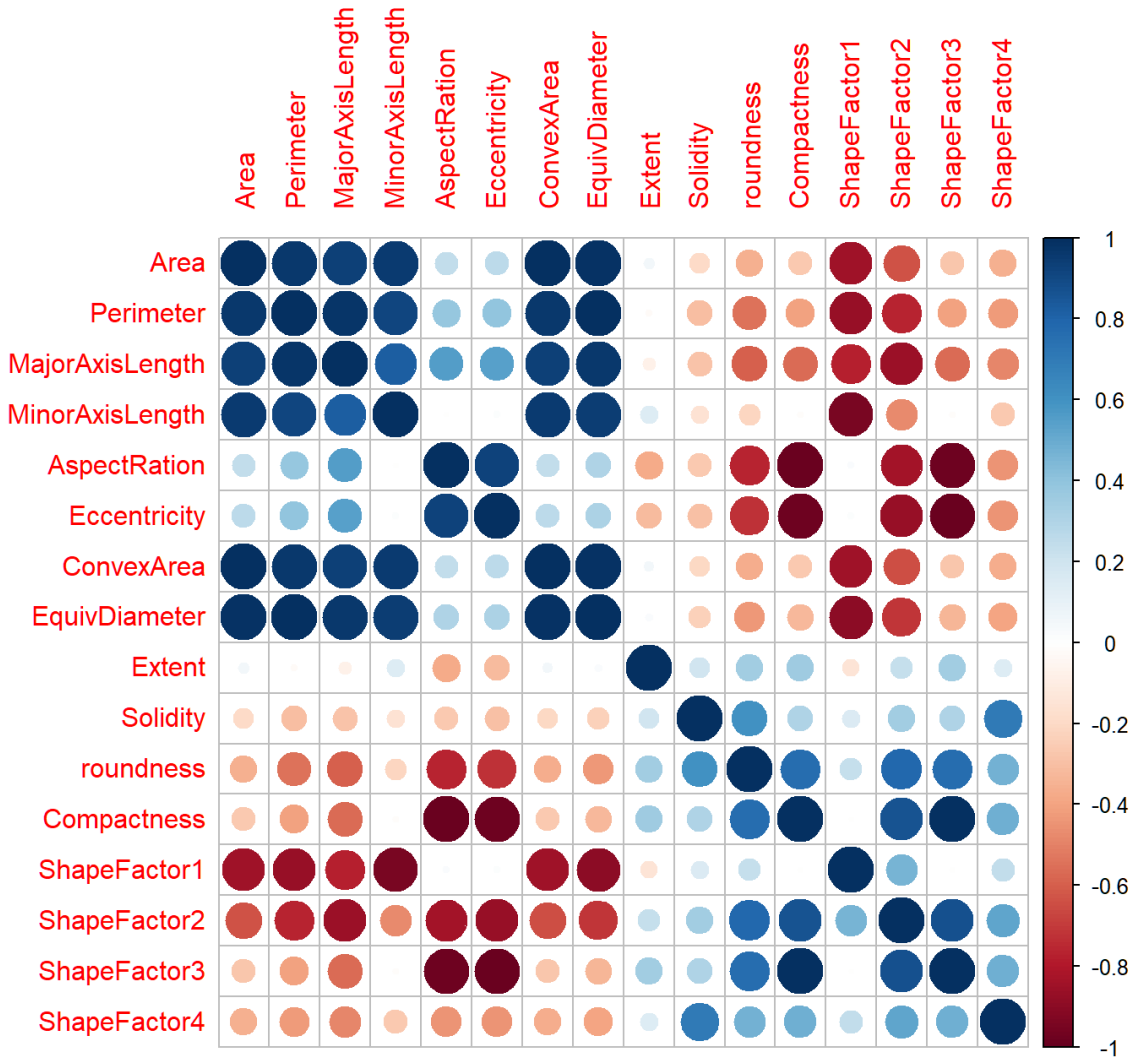

Figure 6: Scatter plots generated by corrplot(cor(data[,-17]),method="circle").

Now we are going to identify the outliers in the data set. An outlier is a value or an observation that is distant from other observations, here we are using the 3-sigma approach, which is to select data point that is between mean-3* standard deviation and mean + 3* standard deviation. Here are histograms of all the features are plotted and shown in Fig. 7. As you can see the result looks like gaussian distribution, but there are many outliers that is outside of 3-sigma.

After calculating, we get 1124 outliers which is not too bad. But we are not yet very sure whether these outliers are really useless. The specific situation needs to be judged by referring to the results shown by the model later, which can better reflect the importance of the features.

```
> count = c()
> par(mfrow=c(6,3))
> for(i in 1:16){
+   mean=mean(data[,i])
+   sd=sd(data[,i])
+   count=c(count, which(data[,i]<mean-3*sd | data[,i]>mean+3*sd) )
+   hist(data[,i],
+        main = names(data[i]),
+        breaks = sqrt(nrow(data[,1:16]))
+   ) # set number of bins
+ }
> count=unique(count)  #  Remove  duplicated  row  indeces
> cat("Number  of  Outilers  =",length(count))
Number  of  Outilers  = 1124
```

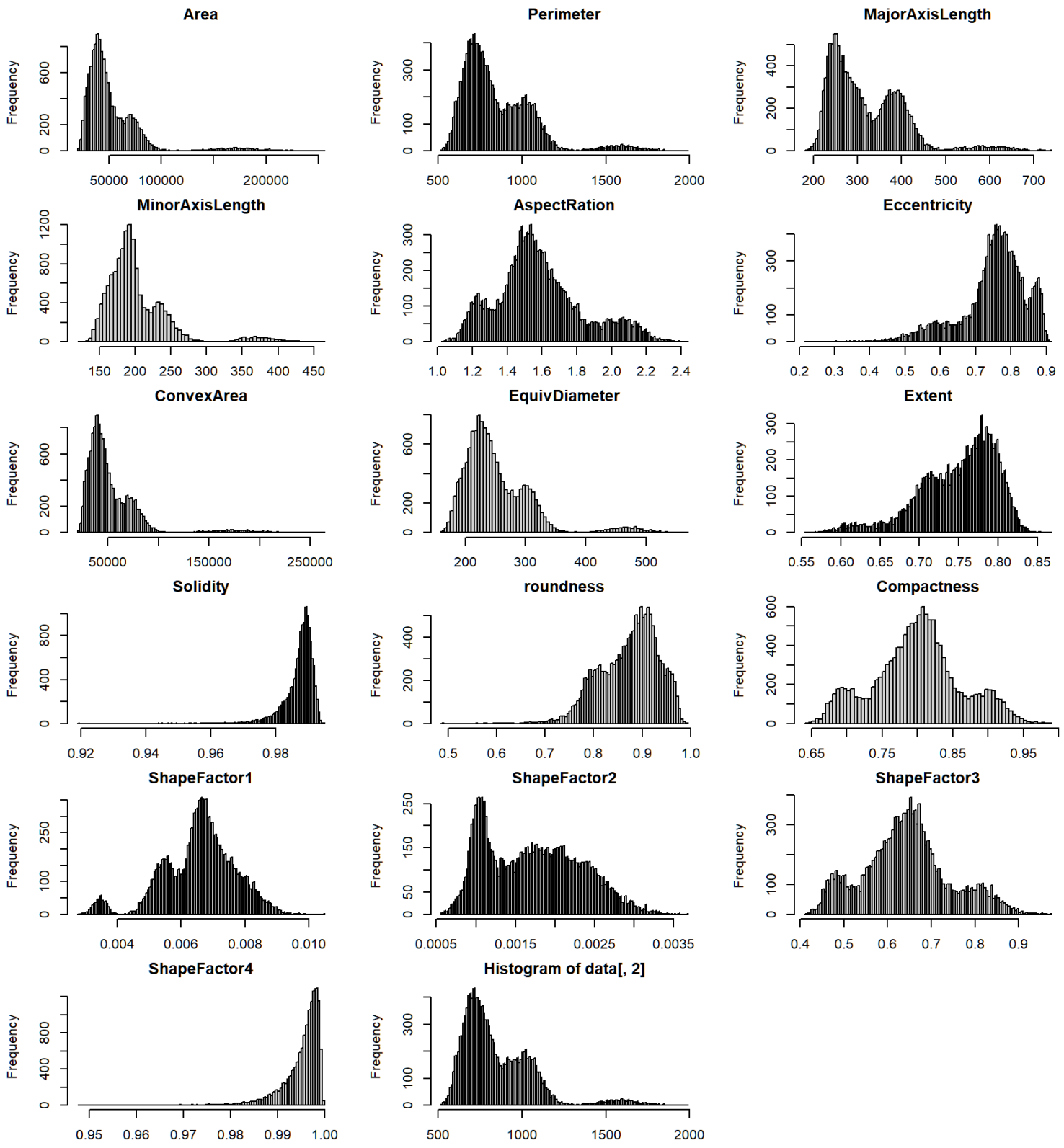

Figure 7: Histograms of the data.

Here we are trying to computing their Variance Inflation Factor (VIF). This can be performed using the vif() function from the car package in R. However, this function requires a logistic regressing model generated by glm(). As stated previously, using glm() directly is not suit- able for our data since it is not binomial; therefore, we employ a one-vs-all approach for each class. Accordingly, we will have 7 glm models, and 7 sets of VIFs, each corresponding to a specific class, as can be seen in the output below.

As we can see below, all the VIF number are pretty small, almost every number is small than one, which implies that all the features do not introduces collinearity for those specific classes. But to aggregating VIFs for multiple classes is very appropriate, so we are not able to determine which features exhibit multicollinearity.

```
> for(i in unique(data$Class)){
+   class_idx=which(data$Class==i)
+   vif_df = data
+   vif_df$Class[class_idx]=1
+   vif_df$Class[-class_idx]=0
+   bi_model=glm(as.numeric(Class)~., vif_df, family="binomial")
+   vifs=vif(bi_model) # Compute VIFs for class i
+   cat("\nVIFs for class", i, '\n')
+   print(vifs)
+ }

VIFs for class SEKER
        Area      Perimeter MajorAxisLength MinorAxisLength    AspectRation
  2.389425e+05   1.088074e+03   4.451618e+05   3.360916e+05   1.004237e+05
  Eccentricity     ConvexArea   EquivDiameter         Extent       Solidity
  1.069926e+03   1.270030e+05   1.627602e+06   1.080498e+00   6.557122e+01
    roundness    Compactness   ShapeFactor1   ShapeFactor2   ShapeFactor3
  9.423058e+01   1.188838e+06   2.349123e+04   2.757536e+04   1.011265e+06
  ShapeFactor4
```

3.154833e+02

VIFs for class BARBUNYA

Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation
1.109025e+05	1.721875e+03	1.521327e+05	1.237183e+05	1.664947e+04
Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity
8.383777e+02	9.507180e+04	4.683773e+05	1.116333e+00	6.816093e+01
roundness	Compactness	ShapeFactor1	ShapeFactor2	ShapeFactor3
1.201184e+02	2.956136e+05	4.067807e+03	6.096094e+03	1.979595e+05
ShapeFactor4				
2.834910e+02				

VIFs for class BOMBAY

Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation
3.809277e+05	8.080495e+03	1.875396e+05	2.462028e+05	3.981406e+04
Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity
9.383050e+03	3.176645e+05	9.453264e+05	2.208065e+00	6.015301e+01
roundness	Compactness	ShapeFactor1	ShapeFactor2	ShapeFactor3
8.346441e+01	1.271180e+06	7.715429e+03	2.904548e+03	1.002195e+06
ShapeFactor4				
9.915823e+01				

VIFs for class CALI

Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation
9.574774e+04	5.789379e+03	1.702200e+05	1.491366e+05	3.696721e+04
Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity
5.375726e+02	9.199200e+04	4.429380e+05	1.058177e+00	1.177933e+02
roundness	Compactness	ShapeFactor1	ShapeFactor2	ShapeFactor3
2.996596e+02	3.326812e+05	4.009850e+03	2.353917e+03	1.986316e+05
ShapeFactor4				
1.303762e+03				

VIFs for class HOROZ

Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation
7.835522e+04	1.925135e+03	1.447837e+05	6.729945e+04	4.115314e+04

```
  Eccentricity     ConvexArea   EquivDiameter        Extent       Solidity
 5.138553e+03    7.201016e+04    3.195703e+05    1.054706e+00    1.102580e+02
    roundness    Compactness    ShapeFactor1    ShapeFactor2    ShapeFactor3
 1.128569e+02    3.183418e+05    3.769052e+03    3.451823e+03    1.638849e+05
 ShapeFactor4
 1.479758e+03
```

VIFs for class SIRA

```
       Area      Perimeter MajorAxisLength MinorAxisLength    AspectRation
 2.398549e+05    5.245216e+03    1.944736e+05    1.372745e+05    3.904532e+04
 Eccentricity     ConvexArea   EquivDiameter        Extent       Solidity
 7.238604e+02    1.327404e+05    6.199344e+05    1.094075e+00    1.083582e+02
    roundness    Compactness    ShapeFactor1    ShapeFactor2    ShapeFactor3
 3.744239e+02    5.774707e+05    1.982746e+04    1.300313e+04    3.380308e+05
 ShapeFactor4
 6.089283e+02
```

VIFs for class DERMASON

```
       Area      Perimeter MajorAxisLength MinorAxisLength    AspectRation
 5.956064e+05    1.474925e+03    6.528697e+05    3.334064e+05    1.785204e+05
 Eccentricity     ConvexArea   EquivDiameter        Extent       Solidity
 3.843601e+03    1.789256e+05    8.004731e+05    1.140507e+00    3.039685e+02
    roundness    Compactness    ShapeFactor1    ShapeFactor2    ShapeFactor3
 2.649454e+02    1.498948e+06    1.746686e+05    5.853589e+04    1.049449e+06
 ShapeFactor4
 1.433302e+03
```

The last step we are going to do is to split the data set into train and test data sets. First we need a seed to shuffle the data set to avoid overfitting and misleading trend. Then we split 70% of data for training, 30% of data for testing. After splitting, we need to ensure that the class distributions between the training set, testing set, and original full data set are similar, so they are not overly impacted the sampling process.

```
> # Split the data set into train and test data sets
> set.seed(18)
> random_data = data[sample(1:nrow(data),nrow(data)),]  # Shuffle
> train_index = sample(1:nrow(data),0.7*nrow(data),replace=F)
> trdf = random_data[train_index,]  # Define training data set
> tstdf = random_data[-train_index,] # Define testing data set
> table(data$Class)/nrow(data)  # Check if class distribution is similar
  BARBUNYA    BOMBAY      CALI  DERMASON     HOROZ     SEKER
0.09712732 0.03835133 0.11975608 0.26052458 0.14165014 0.14892366
     SIRA
0.19366689
> table(trdf$Class)/nrow(trdf)
  BARBUNYA    BOMBAY      CALI  DERMASON     HOROZ     SEKER
0.09383856 0.03789231 0.11745565 0.25968301 0.14191246 0.15209405
     SIRA
0.19712396
> table(tstdf$Class)/nrow(tstdf)
  BARBUNYA    BOMBAY      CALI  DERMASON     HOROZ     SEKER
0.10479922 0.03942214 0.12512243 0.26248776 0.14103820 0.14152791
     SIRA
0.18560235
```