BBM469 - Assignment 3

Clustering and Classification with Pyspark

Group Number: 8

Student name: Ali Argun Sayılgan **Student no:** 21827775

Student name: Cihad Özcan **Student no:** 21827723

Task 1

Purpose

The main aim of this assignment is getting familiar with the basics of Apache Spark and machine learning methods using Spark Environment. For this task, we are expected to implement a simple *word count application.*
In order to accomplish that, we downloaded and used "The Complete Works in Philosophy, Politics and Morals of the late Dr. Benjamin" from the Project Gutenberg

Data Understanding & Preparation

There were 473.227 words in the dataset. After preprocessing, total number of words in the dataset decreased to 189.293.

Steps we followed for data preprocessing are provided below:

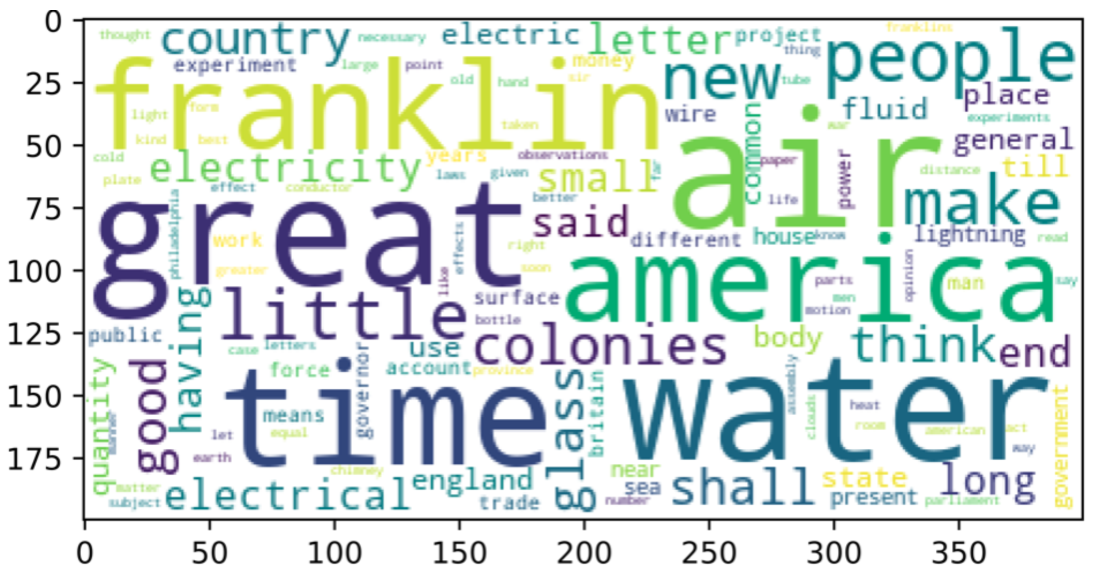
- Removing punctuation and converting all numbers to spaces
- Splitting all lines into words
- Removing words that are shorter than 3 characters
- Converting the words to lowercase and removing stop words utilising the *sklearn* library

Counting Words

We created a tuple for all words with the word and value 1. Then, counted the number of occurrences for each word by summing up those values considering words evaluated as keys. The 10 most common words were: “air” (1073), “great” (895), “water” (879), “time” (802), “franklin” (655), “america” (624), “people” (608), “little” (567), “new” (546) and “make” (546)

Results

A word cloud consists of the 120 most common words are shown at right



Task 2

Purpose

We are given a set of features extracted from the shape of the beans in images and it's expected to predict the type of each bean. There are 7 bean types in the dataset.

Data Understanding

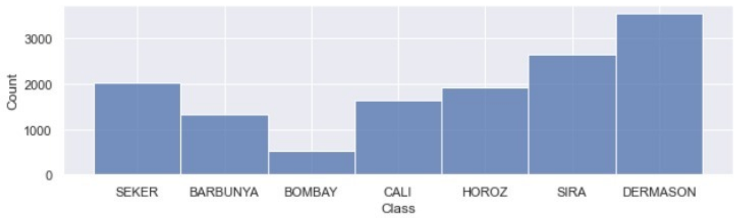
There are 13.611 observations in the dataset, but 68 of them were duplicates and we removed them.

Dataset involves 16 features and they are all numeric. Observations are labelled with one of seven classes which correspond to bean types. There is no missing data in the dataset.

Data is imbalanced and “Dermason” type is considerably concentrated while “Bombay” type is really deficient.

```
original_df_no_dup.printSchema()

root
 |-- Area: integer (nullable = true)
 |-- Perimeter: double (nullable = true)
 |-- MajorAxisLength: double (nullable = true)
 |-- MinorAxisLength: double (nullable = true)
 |-- AspectRation: double (nullable = true)
 |-- Eccentricity: double (nullable = true)
 |-- ConvexArea: integer (nullable = true)
 |-- EquivDiameter: double (nullable = true)
 |-- Extent: double (nullable = true)
 |-- Solidity: double (nullable = true)
 |-- roundness: double (nullable = true)
 |-- Compactness: double (nullable = true)
 |-- ShapeFactor1: double (nullable = true)
 |-- ShapeFactor2: double (nullable = true)
 |-- ShapeFactor3: double (nullable = true)
 |-- ShapeFactor4: double (nullable = true)
 |-- Class: string (nullable = true)
```



Statistics about features are given below:

Details for "Area" column:

summary	Area
count	13543
mean	53048.46038543897
stddev	29392.438324136998
min	20420
max	254616

Details for "Perimeter" column:

summary	Perimeter
count	13543
mean	854.9934058923425
stddev	214.7226835430482
min	524.736
max	1985.37

Details for "MajorAxisLength" column:

summary	MajorAxisLength
count	13543
mean	319.89560224022154
stddev	85.8092600339079
min	183.601165
max	738.8601535

Details for "MinorAxisLength" column:

summary	MinorAxisLength
count	13543
mean	202.36532072404856
stddev	45.05163174744838
min	122.5126535
max	460.1984968

Details for "AspectRation" column:

summary	AspectRation
count	13543
mean	1.5810750385186585
stddev	0.2452452699368285 7
min	1.024867596
max	2.430306447

Details for "ConvexArea" column:

summary	ConvexArea
count	13543
mean	53767.98670900096
stddev	29844.24852511157
min	20684
max	263261

Details for "Extent" column:

summary	Extent
count	13543
mean	0.7498294482518666
stddev	0.04893927702539592
min	0.555314717
max	0.866194641

Details for "roundness" column:

summary	roundness
count	13543
mean	0.8736714768990623
stddev	0.05939310923505456 6
min	0.489618256
max	0.9906854

Details for "ShapeFactor1" column:

summary	ShapeFactor1
count	13543
mean	0.006561215793472...
stddev	0.001129636104801...
min	0.002778013
max	0.010451169

Details for "ShapeFactor2" column:

summary	ShapeFactor2
count	13543
mean	0.001719229327180...
stddev	5.9547367210568E-4
min	5.64169E-4
max	0.003664972

Details for "Eccentricity" column:

summary	Eccentricity
count	13543
mean	0.7503150398336422
stddev	0.09185780580884984
min	0.218951263
max	0.911422968

Details for "EquivDiameter" column:

summary	EquivDiameter
count	13543
mean	253.03409437632712
stddev	59.30770906344923
min	161.2437642
max	569.3743583

Details for "Solidity" column:

summary	Solidity
count	13543
mean	0.9871519229184791
stddev	0.00465006552916899
min	0.919246157
max	0.9946775

Details for "Compactness" column:

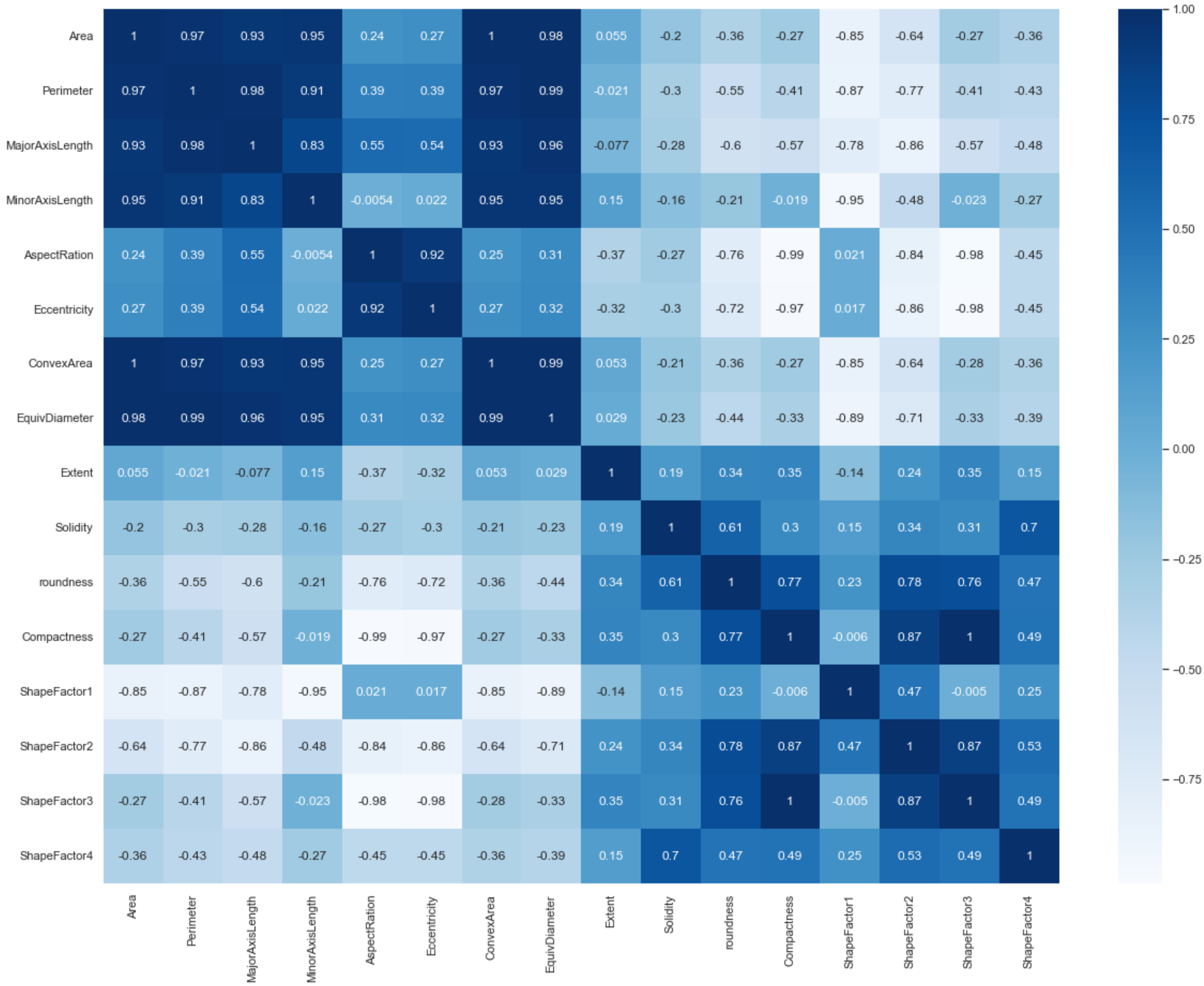
summary	Compactness
count	13543
mean	0.8003520468194636
stddev	0.06146423846431473
min	0.640576759
max	0.987302969

Details for "ShapeFactor3" column:

summary	ShapeFactor3
count	13543
mean	0.6443409725127377
stddev	0.09865302617868243
min	0.410338584
max	0.974767153

Details for "ShapeFactor4" column:

summary	ShapeFactor4
count	13543
mean	0.9950784824186639
stddev	0.004346768072418...
min	0.947687403
max	0.99973253



Since there are a lot of highly correlated features, PCA could be used to achieve better prediction accuracy results. Also with higher dimensioned data you generally need more data for not to underfit the data.

(Principal Component Analysis is a statistical procedure that uses an orthogonal transformation that converts a set of correlated variables to a set of uncorrelated variables. PCA assumes that the principal component with high variance must be paid attention and the PCs with lower variance are disregarded as noise. Pearson correlation coefficient framework led to the origin of PCA, and there it was assumed first that the axes with high variance would only be turned into principal components.)

Data Preparation

In order to run clustering and classification methods successfully, we must convert non-numeric values to numeric values and use one-hot encoding. Luckily the dataset consists of 16 numeric features and 1 categorical label class of string type.

We also needed to normalize the dataset using min-max standardization to create the normalized dataset (ND). Note that it is required to vectorize numeric values before scaling them, since pyspark library can operate such operations on vectors only. So that we scaled columns after vectorizing them, and utilized a pipeline to combine those tasks.

Lastly we converted vectors to double values back, and removed vectorized columns, in order to make this operation reusable.

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import MinMaxScaler
from pyspark.sql import functions as f
from pyspark.ml.feature import VectorAssembler
from pyspark.sql.functions import udf
from pyspark.sql.types import DoubleType

OD = original_df_no_dup.cache()
# Scales numeric columns
columns_to_scale = OD.columns
columns_to_scale.remove("Class")
assemblers = [VectorAssembler(inputCols=[col], outputCol=col + "_vec") for col in columns_to_scale]
scalers = [MinMaxScaler(inputCol=col + "_vec", outputCol=col + "_scaled") for col in columns_to_scale]
pipeline = Pipeline(stages=assemblers + scalers)
scalerModel = pipeline.fit(OD)
scaledData = scalerModel.transform(OD)

# Converts column type from vector to double type
convert_to_double = udf(lambda x: float(list(x)[0]), DoubleType())
for column in columns_to_scale:
    scaled_column = column + "_scaled"
    scaledData = scaledData.withColumn(column, convert_to_double(scaled_column))

# Drops intermediary columns
all_original_columns = columns_to_scale + ["Class"]
ND = scaledData.select(all_original_columns).cache()
```

Modeling for Clustering

We have tested Bi-secting KMeans and KMeans clustering methods with different numbers of components from Principal Component Analysis(PCA) in order to achieve better clustering scores.

The reason behind experimenting with these methods was because KMeans is relatively simple to implement, scales well to large data sets, works well with low dimensioned datasets, and guarantees convergence and we have the required cluster size beforehand, as it is set to the class size of the dataset(bean types) which is 7.

Bisecting K-Means Algorithm is a modification of the K-Means algorithm. It beats K-Means in entropy measurement and it can recognize clusters of any shape and size. Thus we wanted to give it a try.

Area	ConvexArea	0.999940
Compactness	ShapeFactor3	0.998684
Perimeter	EquivDiameter	0.991453
AspectRation	Compactness	0.987644
ConvexArea	EquivDiameter	0.985255
Area	EquivDiameter	0.984998
Eccentricity	ShapeFactor3	0.981058
AspectRation	ShapeFactor3	0.978528
Perimeter	MajorAxisLength	0.977561
Eccentricity	Compactness	0.970308
Perimeter	ConvexArea	0.967871
Area	Perimeter	0.966908
MajorAxisLength	EquivDiameter	0.962271
Area	MinorAxisLength	0.952041
MinorAxisLength	ConvexArea	0.951780
	EquivDiameter	0.949214
	ShapeFactor1	0.947194
MajorAxisLength	ConvexArea	0.933392
Area	MajorAxisLength	0.932623
AspectRation	Eccentricity	0.924185
Perimeter	MinorAxisLength	0.914336
EquivDiameter	ShapeFactor1	0.893403
ShapeFactor2	ShapeFactor3	0.872318
Compactness	ShapeFactor2	0.868347
Perimeter	ShapeFactor1	0.865756
MajorAxisLength	ShapeFactor2	0.859401
Eccentricity	ShapeFactor2	0.859246
Area	ShapeFactor1	0.848390
ConvexArea	ShapeFactor1	0.848382
AspectRation	ShapeFactor2	0.837338
MajorAxisLength	MinorAxisLength	0.828360
roundness	ShapeFactor2	0.781468
MajorAxisLength	ShapeFactor1	0.775840
Perimeter	ShapeFactor2	0.768590
roundness	Compactness	0.765995
AspectRation	roundness	0.764975
roundness	ShapeFactor3	0.761012

```
for i in range(10,17):
    pca = PCA(
        k = i,
        inputCol = 'scaled_feat',
        outputCol = 'pcaFeatures'
    ).fit(scaled_df)
    output = pca.transform(scaled_df)

    bkm_fit = bkm_algo.fit(output)
    km_fit = kmeans_algo.fit(output)

    outputBKM = bkm_fit.transform(output)
    outputKM = km_fit.transform(output)

    scoreBKM = evalBKM.evaluate(outputBKM)
    scoreKM = evalKM.evaluate(outputKM)
    print("PCA("+str(i)+")")
    print("BisectingKMeans silhouette score:\t",scoreBKM)
    print("KMeans silhouette score:\t\t",scoreKM)
    print("----------------------")
```

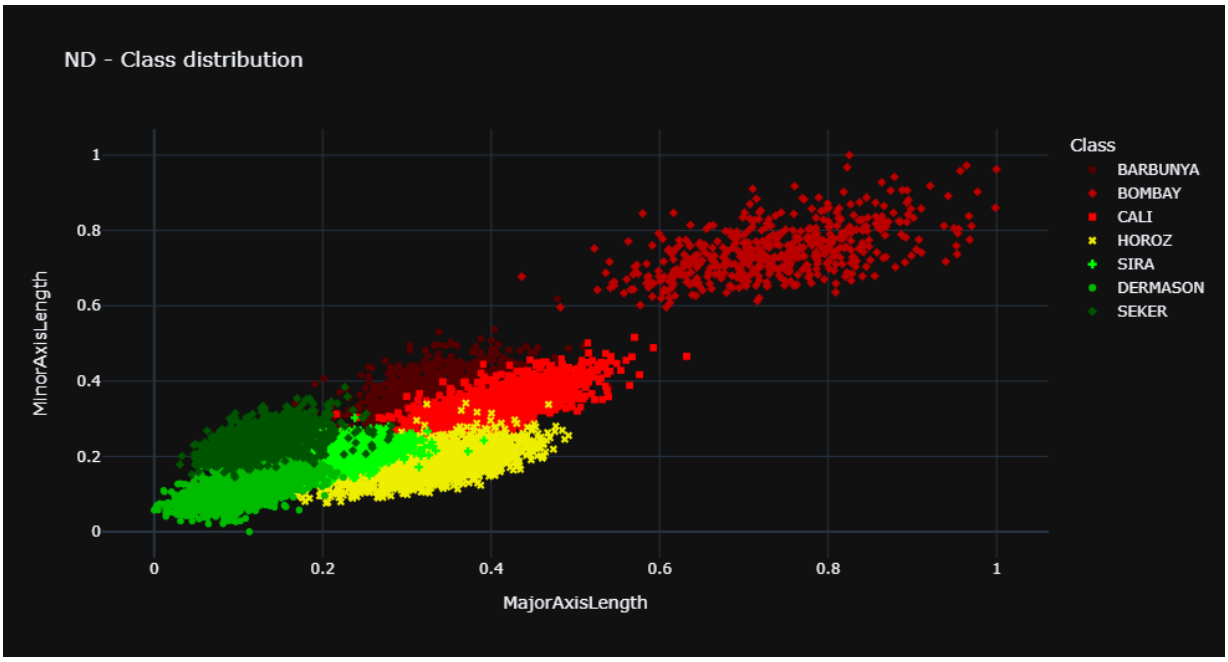
Clustering Results

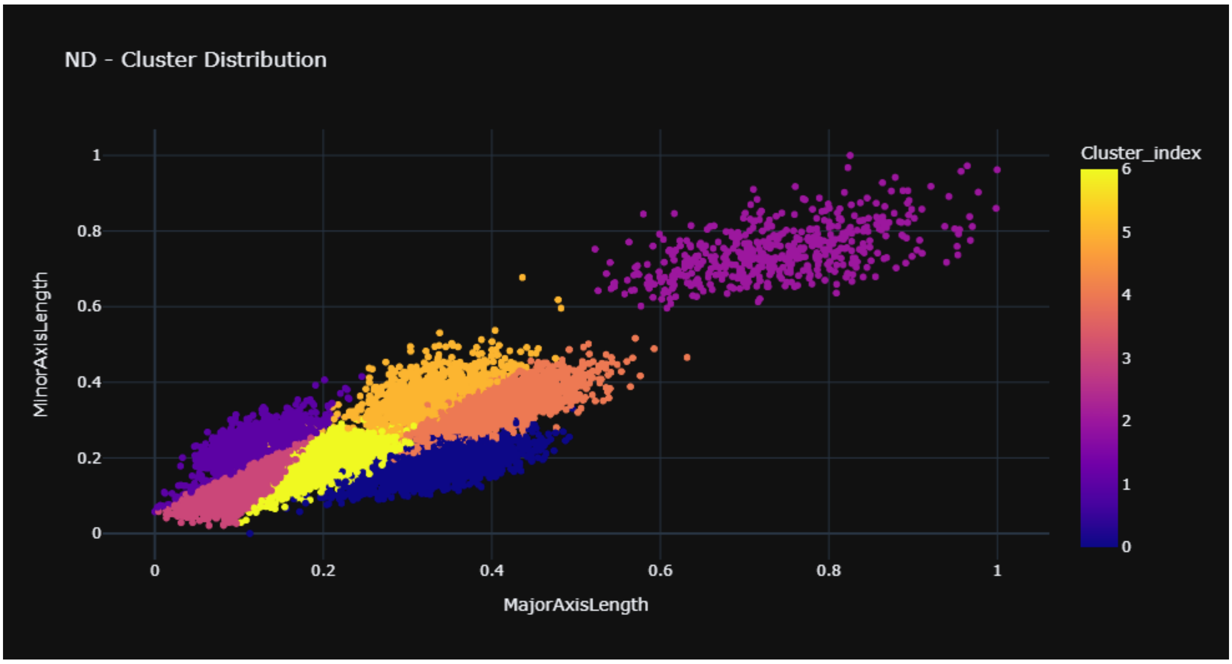
```
PCA(13)
BisectingKMeans silhouette score:       ND | OD   0.4242384161917358      | 0.692467990313307
KMeans silhouette score:                ND | OD   0.4888912773227822      | 0.6931805393382635
----------------------
PCA(14)
BisectingKMeans silhouette score:       ND | OD   0.4242384161917358      | 0.692467990313307
KMeans silhouette score:                ND | OD   0.4888912773227822      | 0.6931805393382635
----------------------
PCA(15)
BisectingKMeans silhouette score:       ND | OD   0.4242384161917358      | 0.692467990313307
KMeans silhouette score:                ND | OD   0.4888912773227822      | 0.6931805393382635
----------------------
PCA(16)
BisectingKMeans silhouette score:       ND | OD   0.4242384161917358      | 0.692467990313307
KMeans silhouette score:                ND | OD   0.4888912773227822      | 0.6931805393382635
----------------------
```

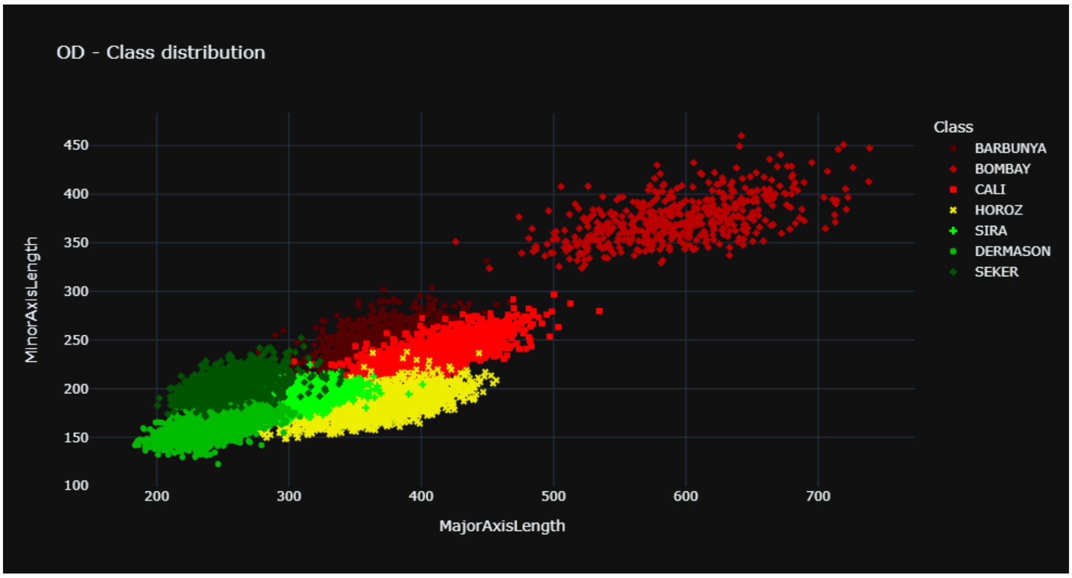
As can be seen, PCA didn't affect the silhouette score much.

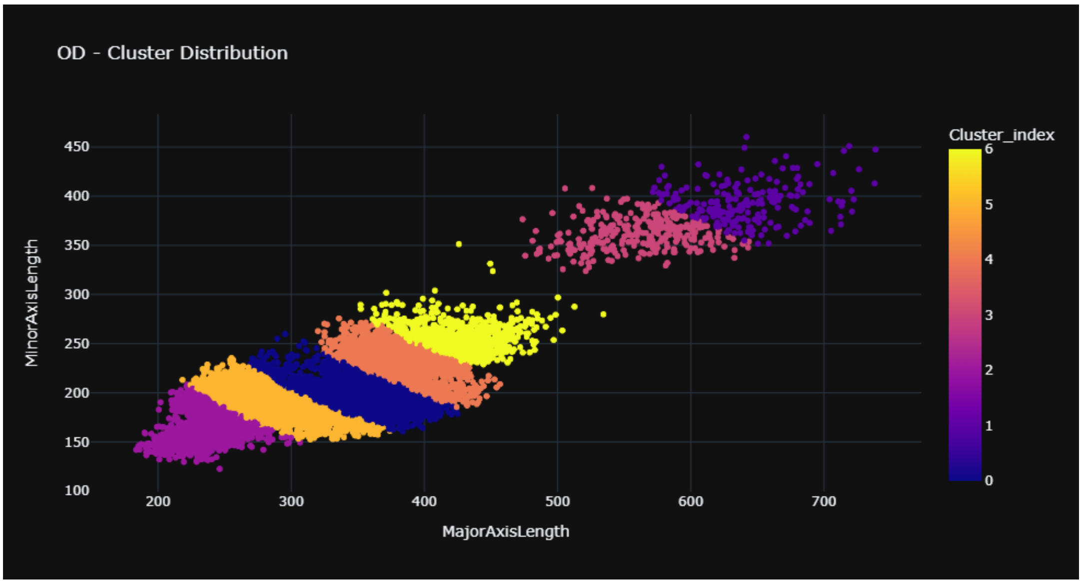
Surprisingly, OD(non-normalized dataset) got significantly higher silhouette scores but this doesn't mean much. When we consider the cluster distribution visually and compare OD-ND, real clustering performance can be viewed better.

The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighbouring clusters.


ND - Class distribution
Class
BARBUNYA
BOMBAY
CALI
HOROZ
SIRA
DERMASON
SEKER
MinorAxisLength
MajorAxisLength
0
0.2
0.4
0.6
0.8
1


ND - Cluster Distribution
Cluster_index
0
1
2
3
4
5
6
MinorAxisLength
MajorAxisLength
0
0.2
0.4
0.6
0.8
1





Commenting on the scatterplots:

X and Y axes are set to highly correlated features to show clustering better.

It should not be forgotten that while interpreting clustering results evaluation scores of the clusters do not always give meaningful answers.

In OD clustering results some anomalies have arisen. The reason behind this result can be KMeans Algorithm's tendency to create uniform-sized clusters.

In non-normalized dataset variance of the features differs a lot and some features influence the clustering result more. Thus clustering score of OD, gives poor results compared to ND.

Modeling for Classification

We used the Random Forests as our model.

We haven't chosen Naive Bayes Classifier (NBC) treats features as if they are independent but our features most likely involve certain interactions between each other. Computational complexity of SVC is much higher than for Random Forests (RF), hence, RF works well with big datasets. Moreover, Random Forests are good at handling missing values and can use both categorical and numerical features together, even if they are not well-scaled. Additionally, Random Forests generally require a large number of instances because of the randomization concept to generalize well. So, we have preferred RF to SVC considering our dataset.

To determine hyperparameters, we used a grid search algorithm. The best model parameters were:

- maxDepth: 7
- maxBins): 20
- numTrees: 30

We split 20% of the dataset as test data, while leftovers were used for training. We didn't use K-Fold cross-validation to determine train/test split size this time since there were enough observations in the dataset. We tried 30% and 15% splits and didn't see a meaningful change in the results, though. Data distribution in test split is proportional to whole dataset distribution.

Classification Results

<u>Metrics for Original Dataset:</u>

Accuracy: 90.3%

Precision: 89.2%

Recall: 92.6%

F1-score: 90.3%

```
Confusion Matrix:
[[655  43   9   0   0   0   0]
 [ 63 446   5   4   3   0   0]
 [ 13  21 390   0   0   2   0]
 [  4  17   0 339   7   3   0]
 [  0   1   1   5 306  16   3]
 [  0   9   2   0  28 225   4]
 [  0   0   0   0   0   0  86]]
```

Metrics for Normalized Dataset:

Accuracy: 91.1%

Precision: 91.3%

Recall: 92.8%

F1-score: 91.1%

```
Confusion Matrix:
[[654  40  11   0   0   0   0]
 [ 53 460   5   6   0   1   0]
 [  7  19 379   0   0   3   0]
 [  2  13   0 350  16   1   0]
 [  0   2   0   0 312  20   4]
 [  0  11   1   0  26 208   1]
 [  0   0   0   0   0   0 105]]
```

Both results can be regarded as rather successful since all results are higher than 90%. But we can say that normalization has a slight (about 1%) boost in performance.