Politecnico Di Torino

Mathematics in Machine Learning
Dry Bean Dataset Analysis

Student:
Khudayar Farmanli
s276310

Professors:
Francesco Vaccarino
Mauro Gasparini

Table of Contents

1. Introduction

The dataset we are using for this analysis had been collected in Turkey. Seven different types of dry beans were used in particular research, taking into account the features such as form, shape, type, and structure by the market situation. A computer vision system was developed to distinguish seven different registered varieties of dry beans with similar features in order to obtain uniform seed classification. For the classification model, images of 13,611 grains of 7 different registered dry beans were taken with a high-resolution camera. Bean images obtained by computer vision system were subjected to segmentation and feature extraction stages, and a total of 16 features; 12 dimensions and 4 shape forms, were obtained from the grains.

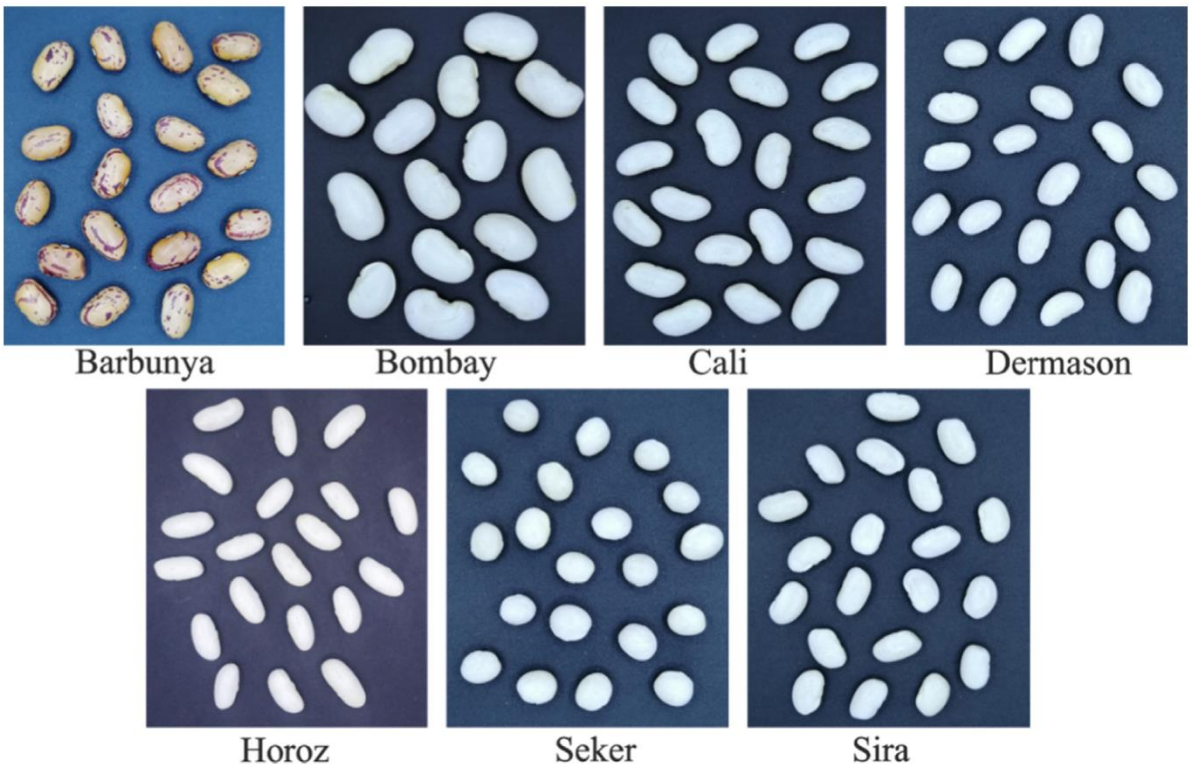

Figure 1. Sample of taken dry bean images.

2. Dataset Description

Our dataset contains 7 different classes which are defined by 16 different features as we mentioned in introduction. All these features are described below with their characteristics:

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)

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent
count	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000
mean	53048.284549	855.283459	320.141867	202.270714	1.583242	0.750895	53768.200206	253.064220	0.749733
std	29324.095717	214.289696	85.694186	44.970091	0.246678	0.092002	29774.915817	59.177120	0.049086
min	20420.000000	524.736000	183.601165	122.512653	1.024868	0.218951	20684.000000	161.243764	0.555315
25%	36328.000000	703.523500	253.303633	175.848170	1.432307	0.715928	36714.500000	215.068003	0.718634
50%	44652.000000	794.941000	296.883367	192.431733	1.551124	0.764441	45178.000000	238.438026	0.759859
75%	61332.000000	977.213000	376.495012	217.031741	1.707109	0.810466	62294.000000	279.446467	0.786851
max	254616.000000	1985.370000	738.860153	460.198497	2.430306	0.911423	263261.000000	569.374358	0.866195

Figure 2. Dataset description

3. Data Exploration

3.1 Null values

Data exploration step is starting point for understanding data for making further developments based on our aim. Firstly, we check the number of "Null" values in the dataset. A null value indicates a lack of a value, which is not the same thing as a value of zero. Null values can be handled with many ways, whether dropping them or replacing them with a custom value (with mean, median etc.) or with some default value. Depends on the dataset we are working on we can decide which way to use. Fortunately, our dataset does not contain any null value at all, and we will not need this step.

3.2 Balance of Classes

We have multiclass dataset and while feature extraction step data collectors used 1 kilogram from each bean variety which means it is highly probable that we will encounter with imbalanced dataset. Because each variety has its own particular size, and it affects the number of beans measured for data collection. From below Figure 3, we can see the frequencies of each Bean variety and we observe that Bombay is the least frequent one. It is related with that Bombay is biggest bean variety in terms of size with respect to others.

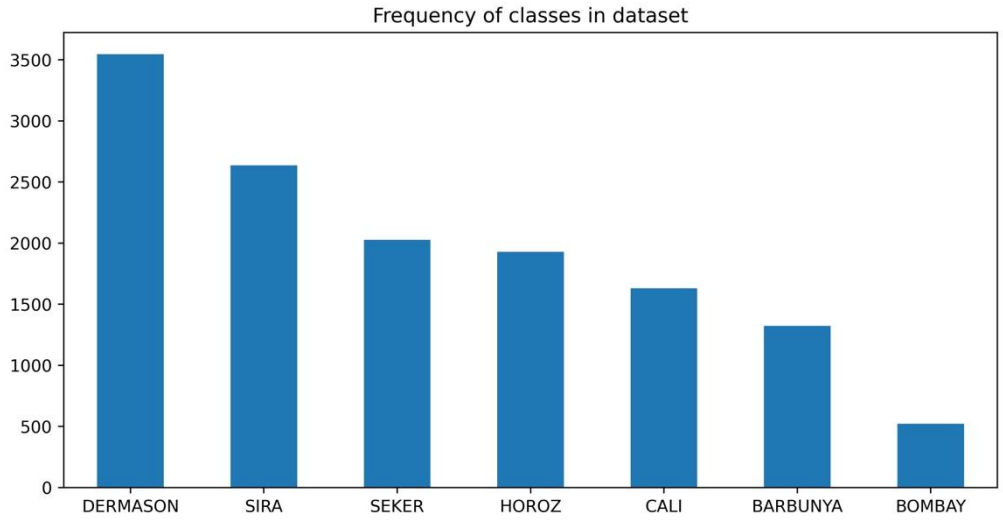

Figure 3. Frequency of Bean varieties in the dataset

In “Balanced” dataset we should have approximately same or close size of frequency for all classes, if the opposite is case then we have “Imbalanced” dataset like our case. We should address this issue, because some of the classification algorithms are sensitive to this case and they are prone to favor the classes that are more frequent. For instance, Random Forest classifier predict labels based on majority voting system and in this case, it is highly probable to fall to this trap. To cope with this issue, we can use one of Oversampling or Undersampling techniques. We will talk about them in further chapters.

3.3 Correlation

Correlation is very useful way to understand linear relationship between two features and in case of high correlation, eliminate one of them. This will simplify the next analytics steps and improve the performance of data-driven algorithms.

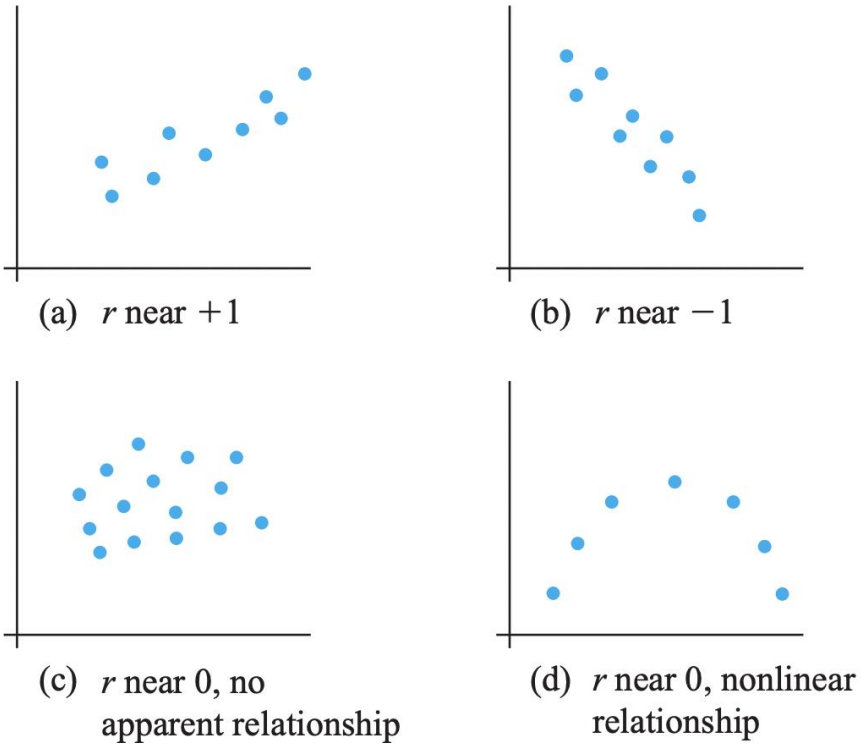

Figure 4. Data plots to see different correlations

For our case we use Pearson Correlation which is described below.

$$\rho = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \qquad \textbf{(1)}$$

- Cov(X, Y) is the covariance
- σ_X is the standard deviation of X
- σ_Y is the standard deviation of Y

In below Figure 5, we see part of Correlation Matrix of our Dry Bean dataset. This is useful way to understand what is going on in terms of linear relationship between 2 particular features. Because of symmetry, naturally, diagonal of matrix contains only values of 1. As we can easily observe that there are many highly correlated features in this dataset. Dark blue parts refer to them. We will use suitable technique to overcome this issue.

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness
Area	1.00	0.97	0.93	0.95	0.24	0.27	1.00	0.98	0.05	-0.20	-0.36
Perimeter	0.97	1.00	0.98	0.91	0.39	0.39	0.97	0.99	-0.02	-0.30	-0.55
MajorAxisLength	0.93	0.98	1.00	0.83	0.55	0.54	0.93	0.96	-0.08	-0.28	-0.60
MinorAxisLength	0.95	0.91	0.83	1.00	-0.01	0.02	0.95	0.95	0.15	-0.16	-0.21
AspectRation	0.24	0.39	0.55	-0.01	1.00	0.92	0.24	0.30	-0.37	-0.27	-0.77
Eccentricity	0.27	0.39	0.54	0.02	0.92	1.00	0.27	0.32	-0.32	-0.30	-0.72
ConvexArea	1.00	0.97	0.93	0.95	0.24	0.27	1.00	0.99	0.05	-0.21	-0.36
EquivDiameter	0.98	0.99	0.96	0.95	0.30	0.32	0.99	1.00	0.03	-0.23	-0.44
Extent	0.05	-0.02	-0.08	0.15	-0.37	-0.32	0.05	0.03	1.00	0.19	0.34
Solidity	-0.20	-0.30	-0.28	-0.16	-0.27	-0.30	-0.21	-0.23	0.19	1.00	0.61
roundness	-0.36	-0.55	-0.60	-0.21	-0.77	-0.72	-0.36	-0.44	0.34	0.61	1.00

Figure 5. Correlation between some features of Dry Bean Dataset

Another way of seeing the relationship between 2 variables visually is use of Pairs Plots from Seaborn library. Beside of this we also see how single variable distributed based on class labels. In Figure 6, we can see example of some features from our Dry Bean dataset.

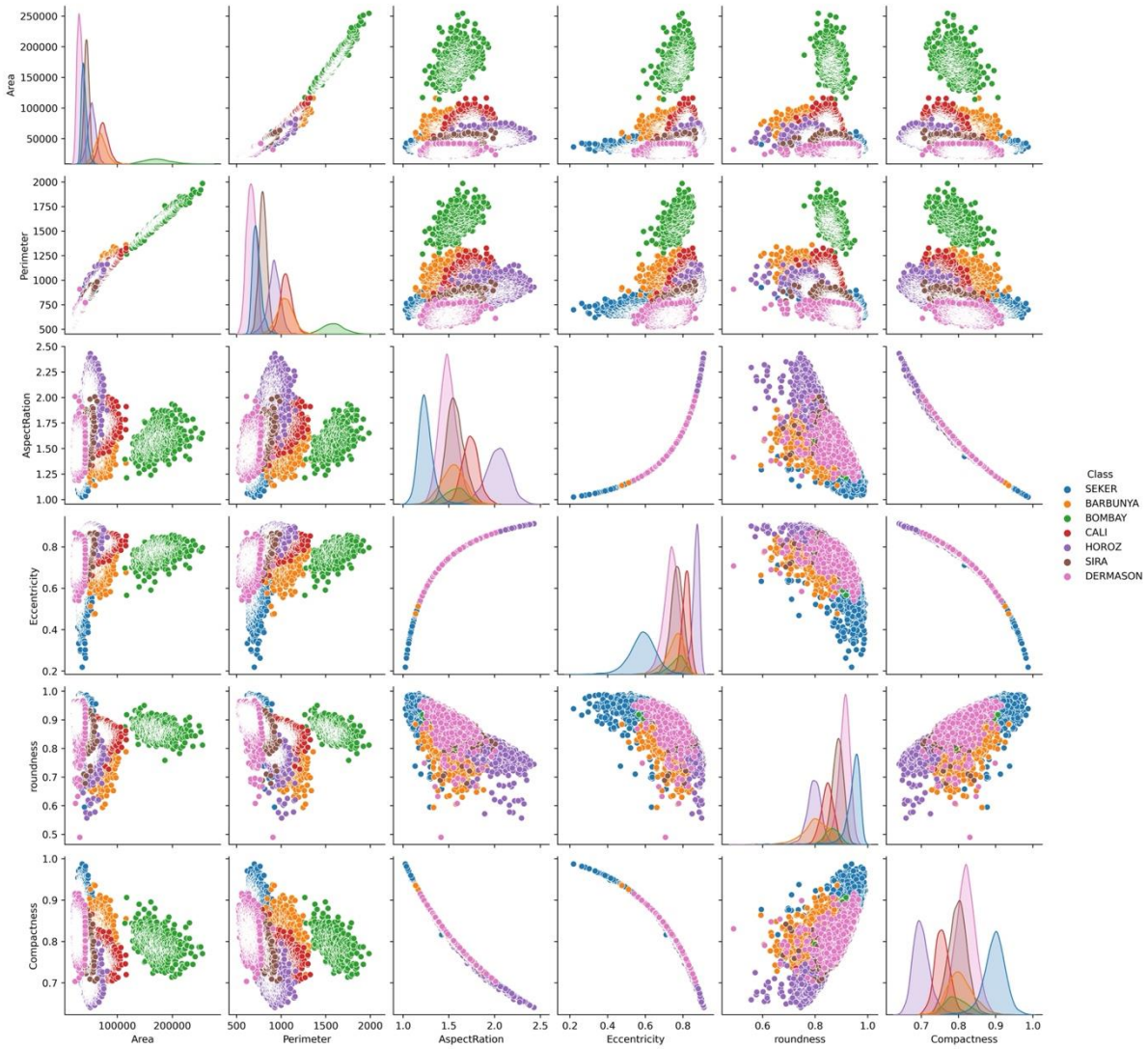

Figure 6. Pairs Plots of some features

4. Data Preparation

4.1 Standardization

Many machine learning algorithms perform better when numerical input variables are scaled to a standard range. Algorithms which are using weighted sum of the input, like linear regression or using distance measures, like k-nearest neighbors can be example to this situation. The two most popular technique for scaling numerical data prior to modeling are normalization and standardization.

- Standardization (or z-score normalization) – is the one we apply to Dry Bean dataset. During this procedure the feature values are rescaled so that they have the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 1$, where μ is the mean and σ is the standard deviation from the mean. Standard scores of features are calculated as follows:

$$\hat{x}^{(j)} = \frac{x^{(j)} - \mu^{(j)}}{\sigma^{(j)}} \quad \textbf{(2)}$$

- Normalization – is the process of converting an actual range of values which a numerical feature can take, into a standard range of values, typically in the interval [-1, 1] or [0, 1].

$$\bar{x}^{(j)} = \frac{x^{(j)} - \min^{(j)}}{\max^{(j)} - \min^{(j)}} \quad \textbf{(3)}$$

Where $\min^{(j)}$ and $\max^{(j)}$ are, respectively, the minimum and the maximum value of the feature j in the dataset.

Also, this step is important for applying PCA, because it calculates a new projection of the dataset, and the new axis are based on the standard deviation of the variables. So, a variable with a high standard deviation will have a higher weight for calculation of axis than a variable with a low standard deviation. If we normalize the data, all variables have the same standard deviation, thus all variables have the same weight and PCA calculates relevant axis. The two most popular technique for scaling numerical data prior to modeling are normalization and standardization. In the Figure 7 below, we can observe how the distribution of data became after the application of Standard Scaler. Violin Plots have additional benefits with respect to Box Plots in which we only observe mean/median and interquartile ranges, the Violin Plots show the entire distribution of the data with its different peaks and their relative amplitudes.

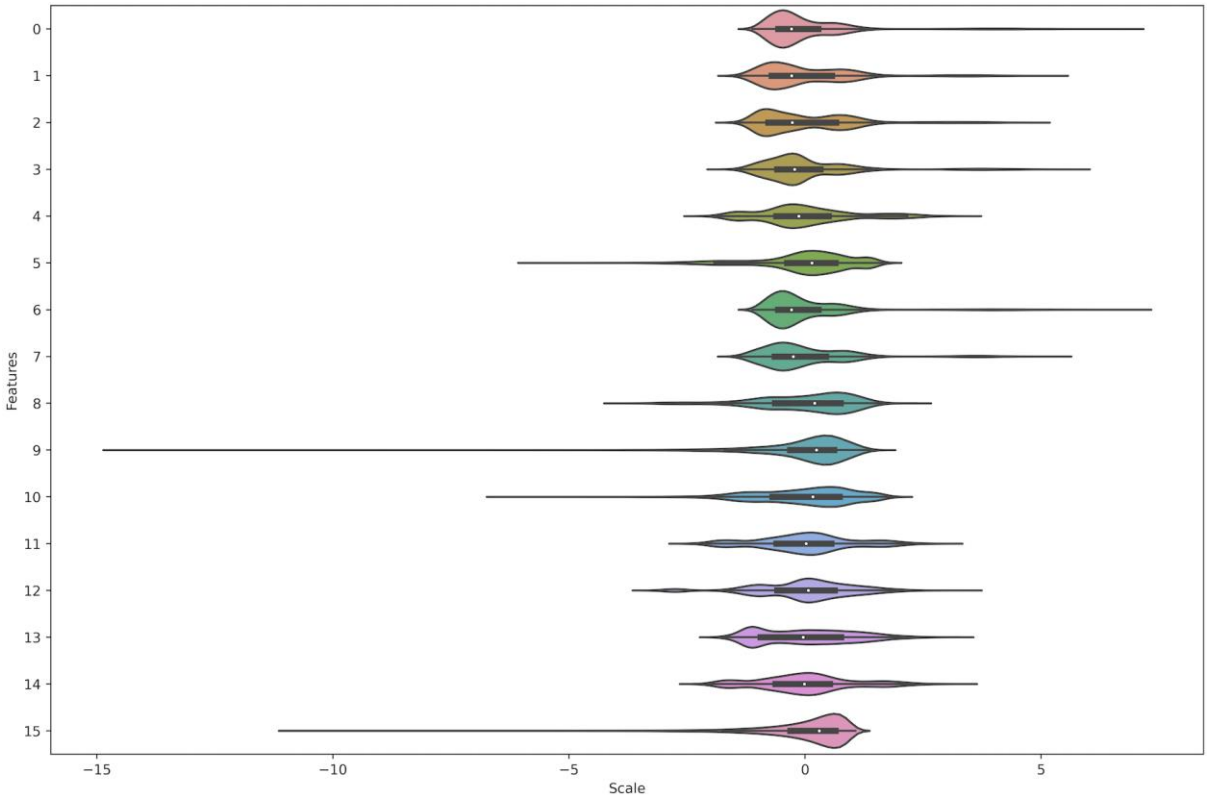

Figure 7. Violin plots after applying Standard Scaler

4.2 Principal Component Analysis (PCA)

In some cases, we need to implement dimensionality reduction. Dimensionality reduction removes redundant or highly correlated features, also reduces the noise in the data. All of those contribute to the interpretability of the model. One of the main and most used examples of dimensionality reduction techniques is Principal Component Analysis, which is generally referred shortly, as PCA. Principal components are vectors that define a new coordinate system in which the first axis goes in the direction of the highest variance in the data. The second axis is orthogonal to the first one and goes in the direction of the second highest variance. If our data is three-dimensional, the third axis will be orthogonal to both the first and the second axes and go in the direction of the third highest variance, and so on. In Figure 8, we see example of two-dimensional dataset.

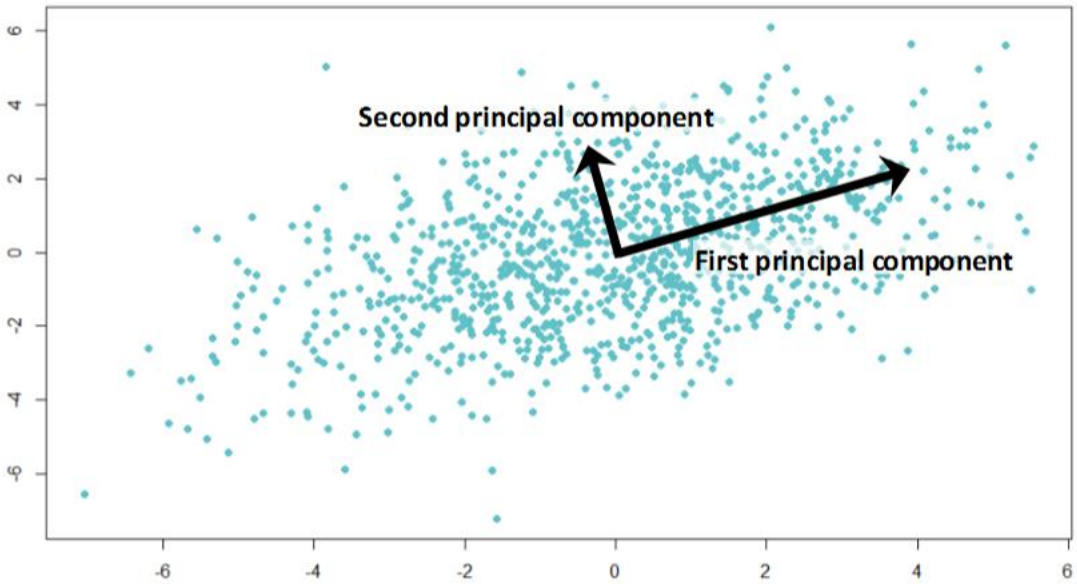

Figure 8. Two principal components displayed as vectors

For our case, after implementation of PCA, through cumulative sum graph we observed that 6 principal components explain 99% of our dataset. It means we can reduce number of features from 16 to 6 without considerable information loss. In Figure 9, this can be observed.

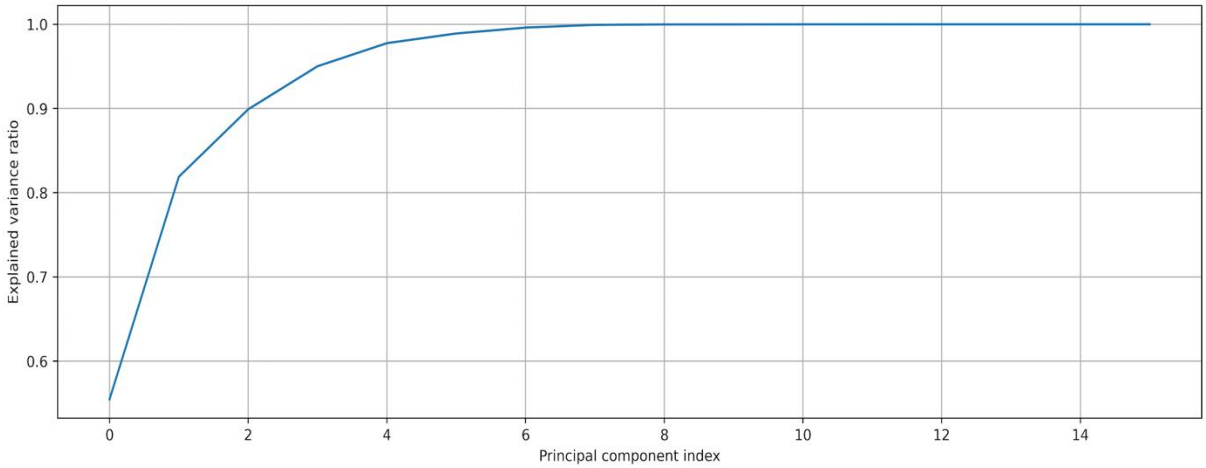

Figure 9. Cumulative sum of Principal Components

5. Settings & Metrics

5.1 Cross Validation

Cross Validation is a method developed to increase the security of classification in which we randomly split our training set into several subsets (k folds) of the same size. In our case it is 10 equal subsets, k = 1, . . . , 10. Each subset contains 10% of our training data. Then we train 10 models as follows. We take the first fold as validation set and the rest folds as training set. Then we continue this process with second fold as validation and the rest as training set etc. till the last fold as validation and the rest as training set. At the end, we average the ten values of metric to get the final value. Unlike Hold-out method which only splits dataset into training and test set, cross validation is more computationally time demanding task but at the same time less dependent on test set like its counterpart. So, this makes it more robust to overfitting problem with respect to Hold-out method.

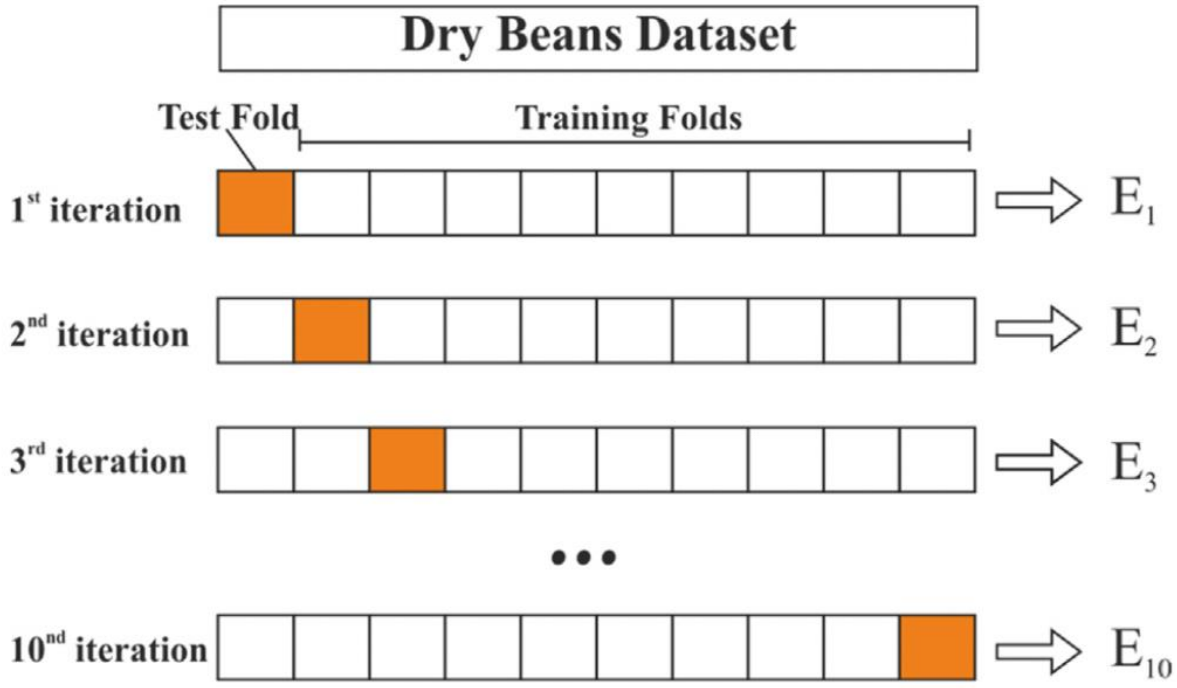

Figure 10. An example of 10-fold cross validation.

5.2 Smote Oversampling

While working with imbalanced data, we should take into consideration that most of the classification algorithms are sensitive to make predictions for the benefit of majority class and biased towards minority class. For addressing this issue, we can use one of the Oversampling and Undersampling techniques. The main point we consider while picking one of them is the size of our dataset. Undersampling mainly, is used for big datasets, otherwise it can cause loss of information. In our particular case, we will use Oversampling which is more suitable.

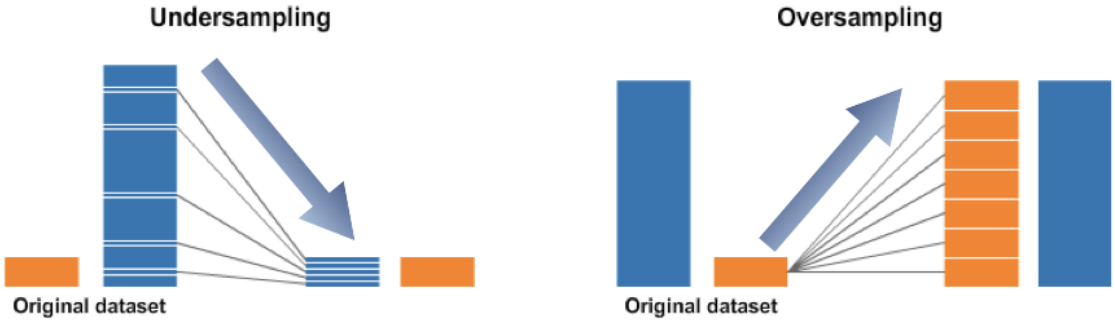

Figure 11. Visual explanation of Undersampling and Oversampling

After deciding Oversampling is more suitable for our dataset, we also should decide how to generate new data points to reach the size of majority class. We can do it by just duplicating examples in the minority dataset or we can generate synthetic data points with SMOTE (Synthetic Minority Oversampling Technique). SMOTE is the most widely used one and is also our choice to implement. SMOTE first selects a random example from the minority class. Then k of the nearest neighbors for that example are found (typically k = 5). A randomly selected neighbor is chosen, and a synthetic example is created at a randomly selected point between the two examples in feature space. In Figure 12, we can visually observe how this process happens.

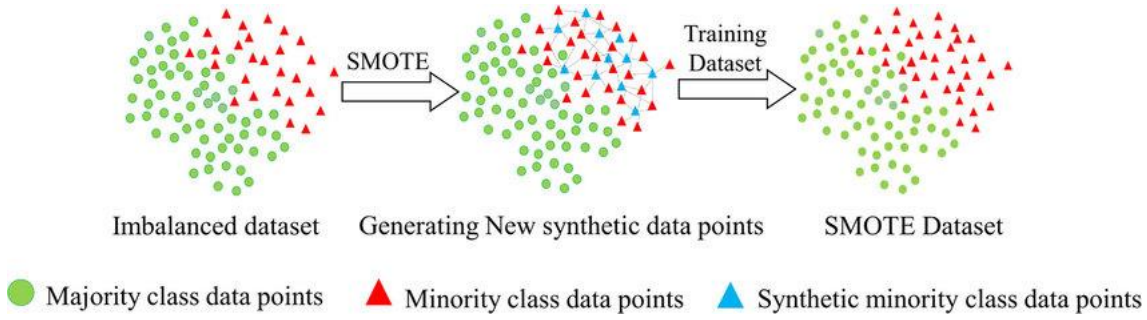

Figure 12. Visual way of SMOTE technique

After all, we also should consider when and how to oversample the dataset. Because oversampling before cross validation can lead to overfitting problem. For instance, if we are doing oversample by duplicating the data points, and if we are doing this step before cross validation, because of repetition in the dataset our classifier will encounter same data points in training and validation datasets at the same time and it will lead to overfitting. Instead, we should split dataset into training and validation folds. Then, on each fold we should

1. Oversample the minority class
2. Train the classifier on training folds
3. Validate the classifier on the remaining fold.

We use pipeline from "imblearn" library which let us to do all these steps easily at one step.

5.3 Model Performance Assessment

Once we have a model which our learning algorithm has built using the training set, we also need some metrics to evaluate how good our model performs. We use the test set to assess the model. For this project we use some of the most widely used metrics and tools such as Confusion matrix, Accuracy, Recall, F1 score etc. to understand what is going on generally.
In order to understand them better we should first introduce Confusion matrix. In Figure 13, we can visually see it. The Confusion matrix is a table that summarizes how successful the classification model is at predicting examples belonging to various classes. One axis of the confusion matrix is the label that the model predicted, the other axis is the actual label. All the other metrics we use are based on this matrix.

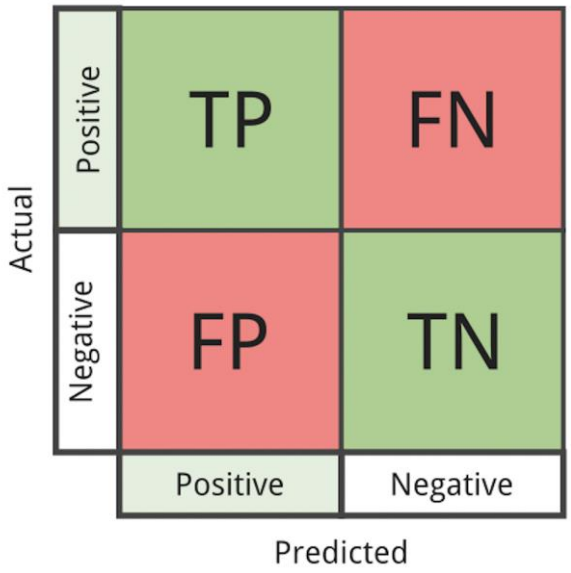

Figure 13. Confusion Matrix

Explanation of TP, FN, FP, TN:

1. TP – True Positive. Both predicted and actual values are positive.
2. FN – False Negative. Predicted value is negative, actual value is positive.
3. FP – False Positive. Predicted value is positive, actual value is negative.
4. TN – True Negative. Both predicted and actual values are negative.

Other metrics we are using:

- Precision – the ratio of correct positive predictions to the overall number of positive predictions. In terms of Confusion matrix, it is given by:

$$\text{Precision (p)} = \frac{TP}{TP+FP} \quad \textbf{(4)}$$

- Recall – the ratio of correct positive predictions to the overall number of positive examples in the dataset. It is given by:

$$\text{Recall (r)} = \frac{TP}{TP+FN} \quad \textbf{(5)}$$

- Accuracy – given by the number of correctly classified examples divided by the total number of classified examples. It is given by:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad \textbf{(6)}$$

- F1 score – harmonic mean of Precision and Recall. It is given by:

$$F1 = \frac{2rp}{r+p} \quad \textbf{(7)}$$

All these metrics have their own pros and cons. For instance, Accuracy is not appropriate measure when dataset is imbalanced, on the other hand, the others perform better in that case. Also, for instance when FP is important for our analysis because of our final target, Precision will be better measure for our case or FN is important and it is better to refer Recall.

6. Model Selection

6.1 Decision Tree

A Decision Tree is an acyclic graph that can be used to make decisions. In each branching node of the graph, a specific feature j of the feature vector is examined. If the value of the feature is below a specific threshold, then the left branch is followed; otherwise, the right branch is followed. As the leaf node is reached, the decision is made about the class which the example belongs. Nodes of Decision tree are splitted based on Entropy or Gini index which is chosen by us. In Figure 14, graphically this process is depicted. Decision trees can handle both categorical and numerical data. Advantages of Decision trees are computationally inexpensive and easy to interpret for humans but at the same time prone to overfitting (it refers to the process when model is trained on training data too well that any noise in testing data can bring negative impacts to performance of model) which can be taken as disadvantage.

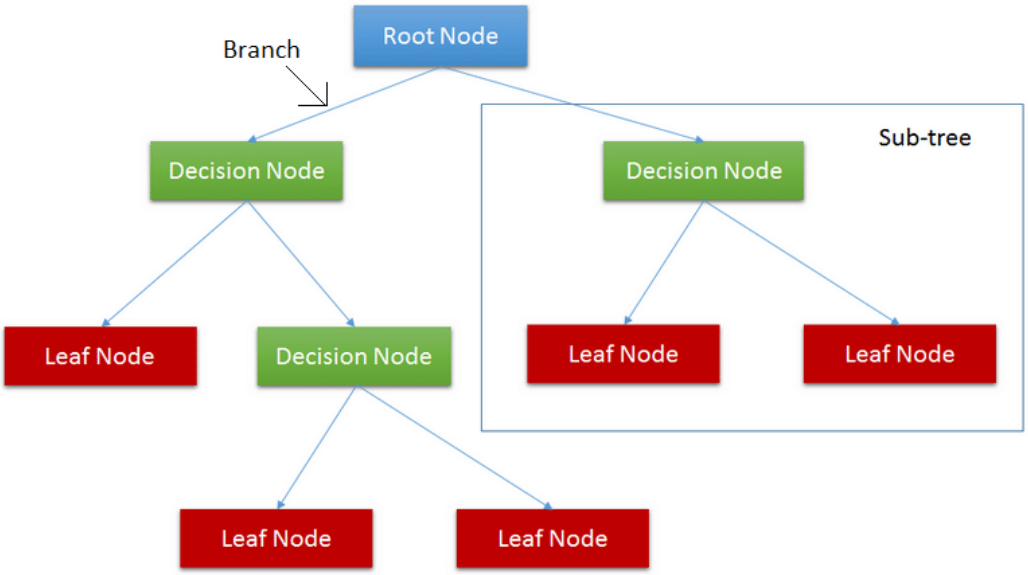

Figure 14. Decision Tree Structure

The parameters evaluated through hyperparameter tuning are:

- “max_features” : ['auto', 'sqrt', 'log2'],
- “ccp_alpha” : [0.1, .01, .001],
- “max_depth” : [5, 6, 7, 8, 9],
- “criterion” : ['gini', 'entropy']

Best parameters obtained:

- 'ccp_alpha' : 0.001
- 'criterion' : 'entropy'
- 'max_depth' : 9
- 'max_features' : ' log2'

We defined k-fold as k=10. In Figure 15, we see confusion matrix which is summary of prediction results of Decision tree classifier. Through confusion matrix we can observe the number of correct and incorrect predictions which are summarized with count values and broken down by each class.

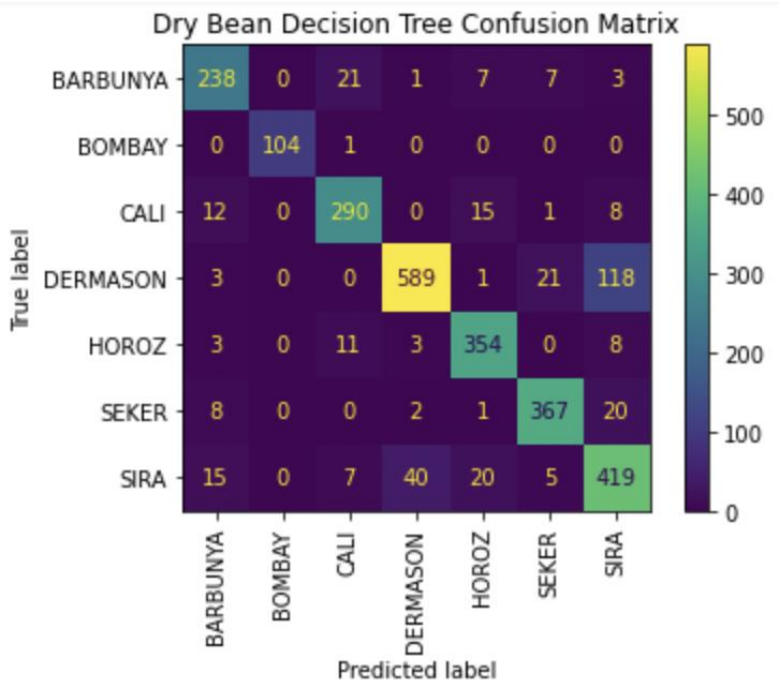

Figure 15. Decision Tree Confusion Matrix

In Figure 16, we see overall accuracy score for classification and precision, recall and f1-score for each class label.

	precision	recall	f1-score	support
BARBUNYA	0.853047	0.859206	0.856115	277.000000
BOMBAY	1.000000	0.990476	0.995215	105.000000
CALI	0.878788	0.889571	0.884146	326.000000
DERMASON	0.927559	0.804645	0.861741	732.000000
HOROZ	0.889447	0.934037	0.911197	379.000000
SEKER	0.915212	0.922111	0.918648	398.000000
SIRA	0.727431	0.828063	0.774492	506.000000
accuracy	0.867058	0.867058	0.867058	0.867058
macro avg	0.884498	0.889730	0.885936	2723.000000
weighted avg	0.872636	0.867058	0.867986	2723.000000

Figure 16. Decision Tree Classification Report

6.2 Random Forest

Random Forest is ensemble learning algorithm, and it is simply set of decision trees. Some number of decision trees are built at training time and the class assigned by majority voting. Random Forest is based on the idea of Bagging which consists of creating many "copies" of training data (each copy is slightly different from another) and then apply the weak learner (decision tree in our case) to each copy to obtain multiple weak models and then combine them. Random Forest is one of the widely used algorithms, it is because of the reason that by using multiple samples of original dataset, we reduce the variance of the model. Low variance means low overfitting which is very desirable target in classification. Accuracy of Random Forest is also high especially from Decision trees which is base model.

The parameters evaluated through hyperparameter tuning are:

- 'n_estimators' : [50, 100, 200],
- 'max_depth' : [4, 6, 10, 12],
- 'random_state' : [13]

Best parameters obtained:

- 'max_depth' : 12
- 'n_estimators' : 200
- 'random_state' : 13

We defined k-fold as k = 10. In Figure 17, we can see Confusion matrix to observe performance of Random Forest while classifying labels for each class.

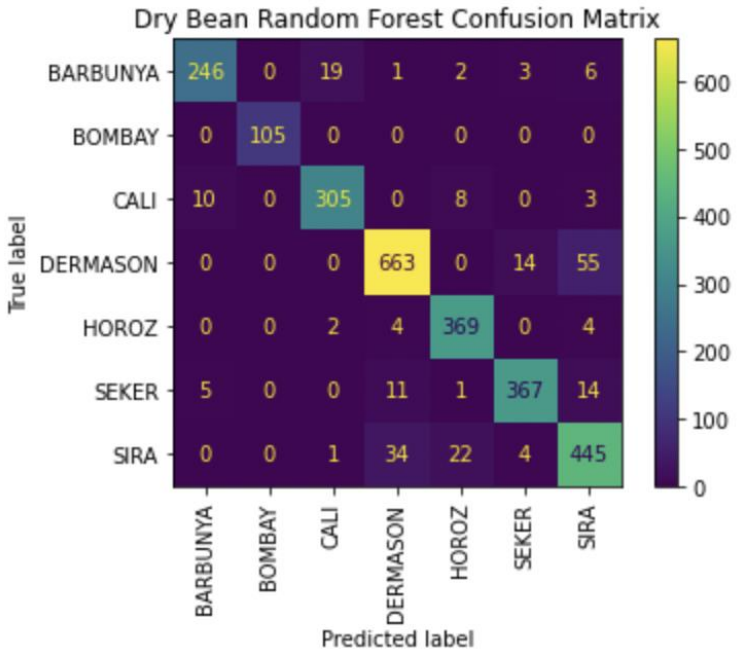

Figure 17. Random Forest Confusion Matrix

In Figure 18, we see overall accuracy score for classification and precision, recall and f1-score for each class label.

	precision	recall	f1-score	support
BARBUNYA	0.942529	0.888087	0.914498	277.000000
BOMBAY	1.000000	1.000000	1.000000	105.000000
CALI	0.932722	0.935583	0.934150	326.000000
DERMASON	0.929874	0.905738	0.917647	732.000000
HOROZ	0.917910	0.973615	0.944942	379.000000
SEKER	0.945876	0.922111	0.933842	398.000000
SIRA	0.844402	0.879447	0.861568	506.000000
accuracy	0.918105	0.918105	0.918105	0.918105
macro avg	0.930473	0.929226	0.929521	2723.000000
weighted avg	0.918997	0.918105	0.918223	2723.000000

Figure 18. Random Forest Classification Report

6.3 K-Nearest Neighbor (KNN) Classifier

KNN is a non-parametric learning algorithm. Contrary to other learning algorithms that allow discarding the training data after the model is built, KNN keeps all training examples in memory. Once a new, previously unseen example x comes in, the KNN algorithm finds its k training examples closest to x and return the majority label, in case of classification. The closeness of two examples is given by a distance function. As distance function generally, Euclidean distance is preferred. Also, we can pick one of the Mahalanobis, Chebychev, Cosine similarity etc. distances too. For instance, Cosine similarity is defined as,

$$s(x_i, x_k) = \cos\big(\angle(x_i, x_k)\big) = \frac{\sum_{j=1}^{D} {x_i}^{(j)}, {x_k}^{(j)}}{\sqrt{\sum_{j=1}^{D} ({x_i}^{(j)})^2}\sqrt{\sum_{j=1}^{D} ({x_k}^{(j)})^2}} \quad \textbf{(8)}$$

is a measure of similarity of the directions of two vectors. IF the angle between two vector is 0 degrees, then two vectors point to the same direction, and cosine

similarity is equal to 1. If the vectors are orthogonal, the cosine similarity is 0. For vectors pointing in opposite directions, the cosine similarity is -1. If we want to use cosine similarity as a distance metric, we need to multiply it by -1.

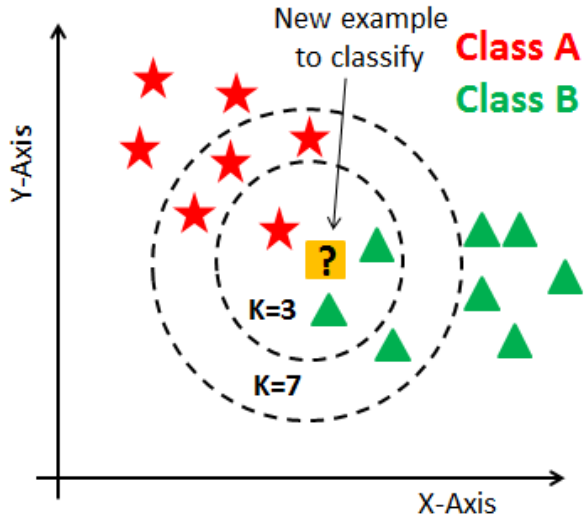

Figure 19. Graphical explanation of KNN

The parameters evaluated through hyperparameter tuning are:

- "n_neighbors" : [3,5,7,9,11,13,15]
- "weights" : ["uniform", "distance"]

Best parameters obtained:

- 'n_neighbors' : 11
- 'weights' : 'uniform'

K-fold defined as k = 10. Figure 20 shows how KNN classifier performed while classifying labels.

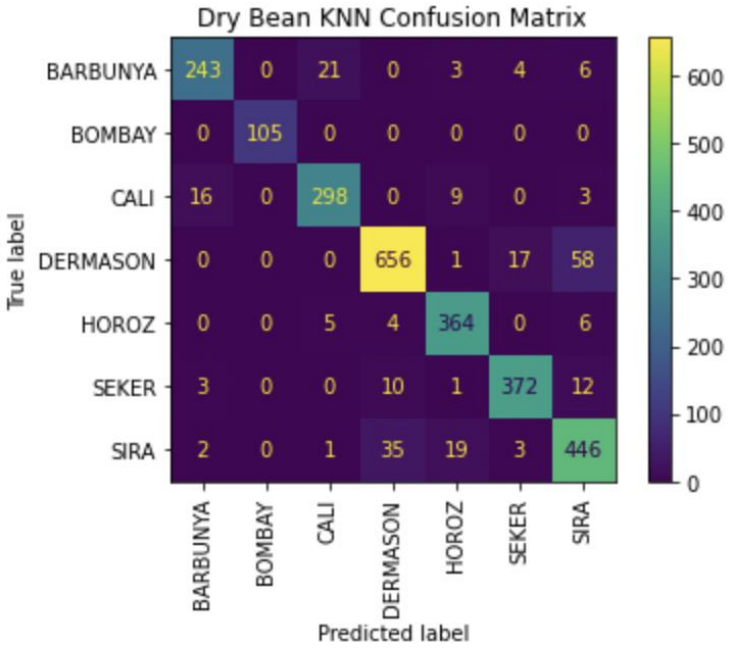

Figure 20. KNN classifier Confusion Matrix

In Figure 21, we see overall accuracy score for classification and precision, recall and f1-score for each class label.

	precision	recall	f1-score	support
BARBUNYA	0.920455	0.877256	0.898336	277.000000
BOMBAY	1.000000	1.000000	1.000000	105.000000
CALI	0.916923	0.914110	0.915515	326.000000
DERMASON	0.930496	0.896175	0.913013	732.000000
HOROZ	0.916877	0.960422	0.938144	379.000000
SEKER	0.939394	0.934673	0.937028	398.000000
SIRA	0.839925	0.881423	0.860174	506.000000
accuracy	0.912229	0.912229	0.912229	0.912229
macro avg	0.923438	0.923437	0.923173	2723.000000
weighted avg	0.913104	0.912229	0.912363	2723.000000

Figure 21. KNN Classification Report

6.4 Support Vector Machine (SVM)

The objective of the Support Vector Machine algorithm is to find a hyperplane in an N-dimensional space (where N is the number of features) that distinctly

classifies the data points. To separate two classes of data point, there are many possible hyperplanes that can be chosen as it can be seen in the left-side graph of Figure 22. Our objective is to find a plane that has the maximum margin like in the right-side graph of the same Figure. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

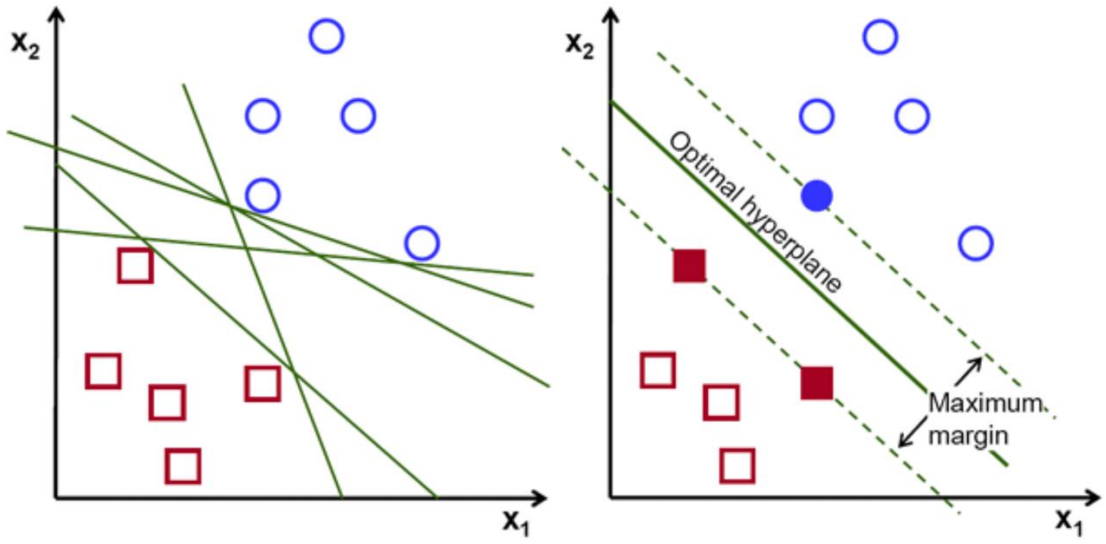

Figure 22. Possible hyperplanes

Hyperplanes are decision boundaries that help to classify the data points. Data points on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. In case of two input features, hyperplane is just a line, in case of three, it becomes two-dimensional plane like depicted in Figure 23.

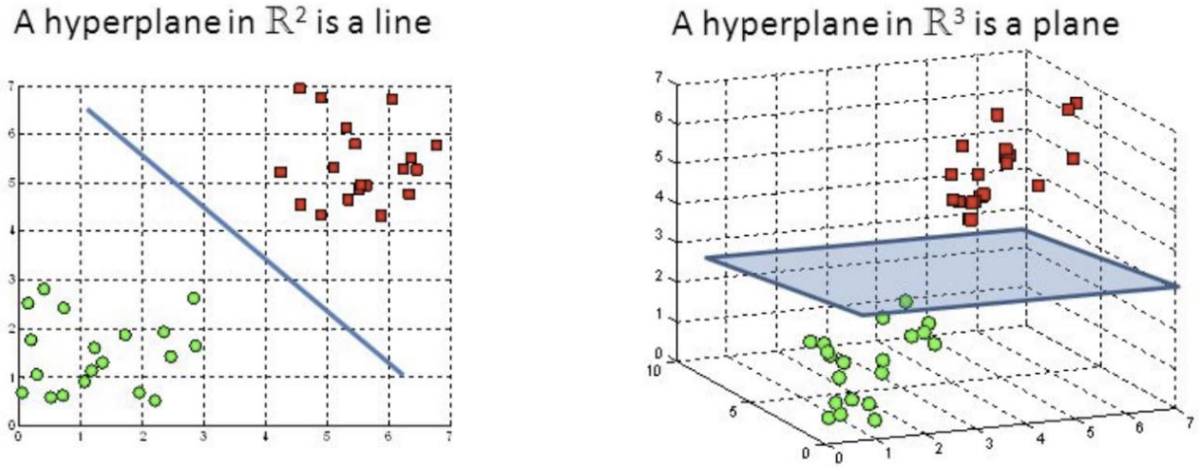

Figure 23. Hyperplanes in different feature spaces

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM. Support vectors given by (for linear kernel):

$$wx_i - b \geq +1 \quad if\ y_i = +1 \quad \textbf{(9)}$$
$$wx_i - b \leq +1 \quad if\ y_i = -1 \quad \textbf{(10)}$$

Hyperplane (decision boundary) given by:

$$wx - b = 0. \quad \textbf{(11)}$$

Where x_i is the feature vector of example i and y_i is its label that takes value either -1 or +1, w is any vector that is located perpendicular to the decision boundary and b indicates the margin of SVM.

SVMs can also classify nonlinear data by moving the data to a higher size by a method called kernel trick. Usually, data that cannot be separated linearly in the original input space can be separated in high dimensional feature space. Figure 24 shows it visually.

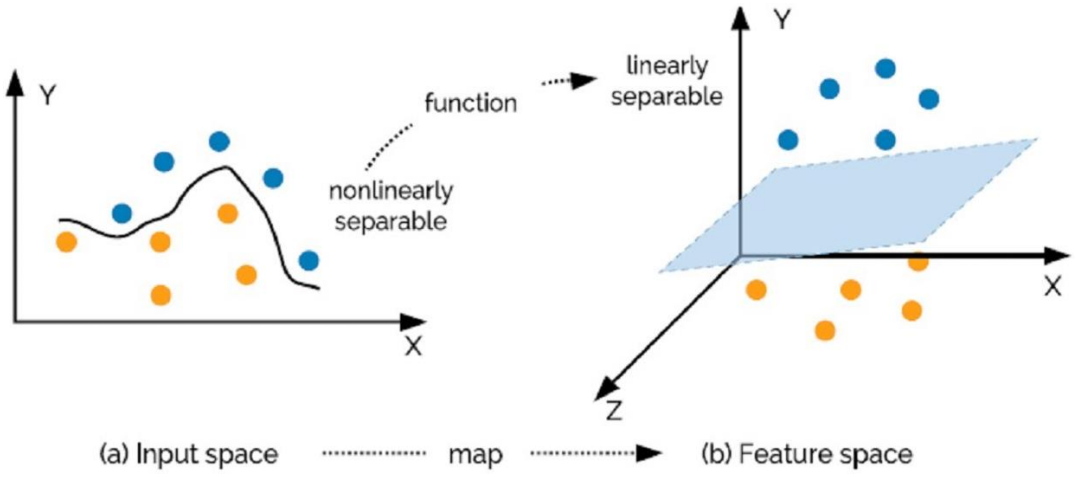

Figure 24. Kernel Trick

The parameters evaluated through hyperparameter tuning are:

- 'C' : [1, 10]
- 'gamma' : [0.001, 0.01, 1]

Best parameters obtained:

- 'svc__C' : 10
- 'svc__gamma' : 0.01

K-fold defined as k = 10. Figure 25 shows how SVM classifier performed while classifying labels.

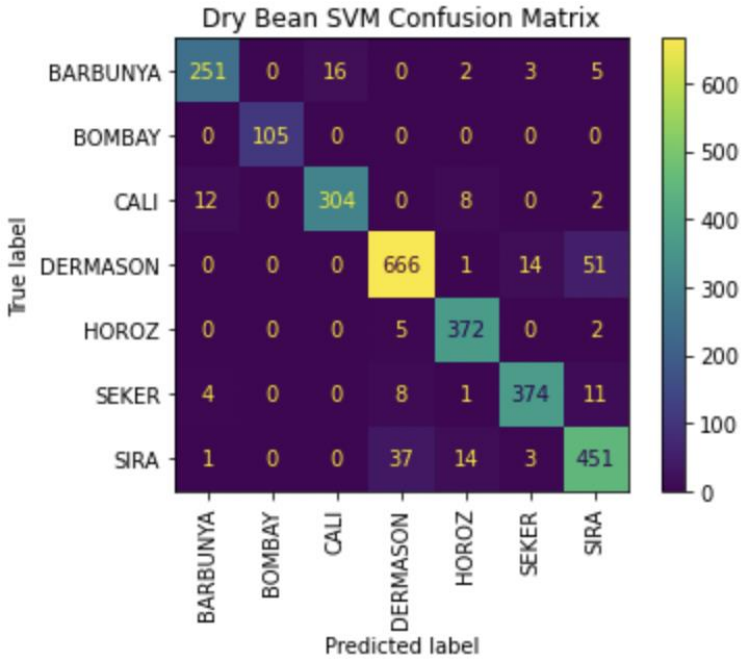

Figure 25. SVM Confusion Matrix

In Figure 26, overall accuracy score for classification and precision, recall and f1-score for each class label have been described.

	precision	recall	f1-score	support
BARBUNYA	0.936567	0.906137	0.921101	277.000000
BOMBAY	1.000000	1.000000	1.000000	105.000000
CALI	0.950000	0.932515	0.941176	326.000000
DERMASON	0.930168	0.909836	0.919890	732.000000
HOROZ	0.934673	0.981530	0.957529	379.000000
SEKER	0.949239	0.939698	0.944444	398.000000
SIRA	0.863985	0.891304	0.877432	506.000000
accuracy	0.926552	0.926552	0.926552	0.926552
macro avg	0.937804	0.937289	0.937367	2723.000000
weighted avg	0.927002	0.926552	0.926589	2723.000000

Figure 26. SVM Classification Report

7. Conclusion

In conclusion, we can say that all the models performed fairly good in terms of accuracy. From Figure 27, we observe it visually. Also, from classification report tables, we observed how each classification model performed on each bean variety. Roughly, all the results in terms of other metrics overlapped and this shows the data distribution is regular. Of course, there were some problems too. For instance, it is seen that all classification models have the lowest sorting performance of the Sira variety due to low distinguishing of the Sira bean variety with the Dermason variety. The fact that the flatness and the roundness of the Dermason and Sira varieties are similar is effective in this result. Bombay variety was classified 100% right for all classification models other than Decision tree which performed 99 % right. It was because of distinctive features of this variety which can be easily seen in Figure 1. For the other bean varieties there was not distinctive difference like those we mentioned. Overall, best performed classification model was SVM and the least successful one was Decision tree.

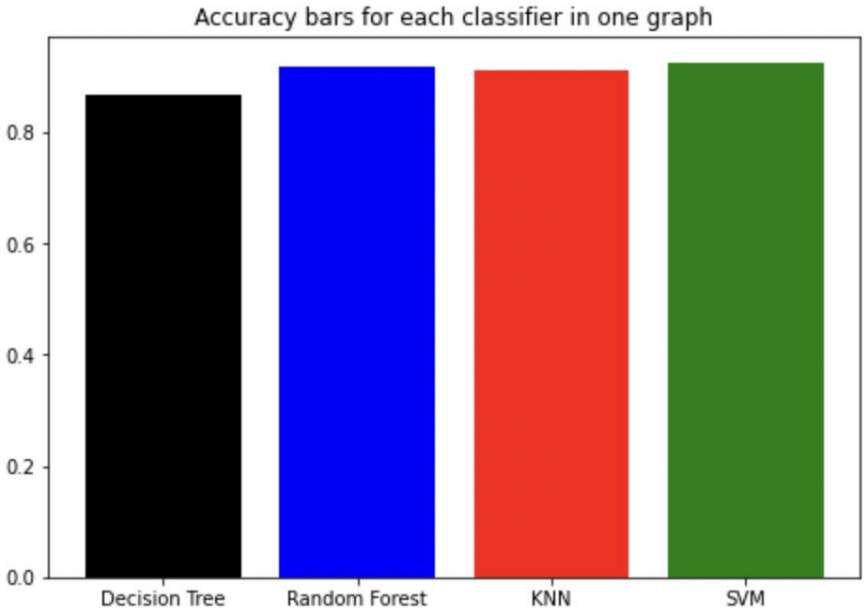

Figure 27. Accuracy results

8. References

- Dataset available at: https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset
- Related paper available at: https://www.sciencedirect.com/science/article/pii/S0168169919311573?via%3Dihub
- Shai Ben-David and Shai Shalev-Shwartz, Understanding Machine learning. 1994
- Andriy Burkov, The Hundred-Page Machine Learning Book. 2019
- Jay L. Devore, Probability and Statistics for Engineering and the Sciences, 1982