ABSTRACT

The purpose of this project is to conduct Data Analysis and check if we can classify different quality of Dry Beans into various classes and interpret the results of different classification methods and their algorithms on the data. The objective is to find the factors that have the most effect on classification of the Dry Bean data from various variables and analyse them. For this purpose, Jupyter notebook is used wherein the use of Python Programming language is made as it is a good tool for machine learning problems. Five analysis are performed on the data: Decision tree classification, XGBoost classification, Random Forest classification, Support Vector Machine Classification, and K- Nearest Neighbour Classification taking the variable ‘Class’ as the dependent variable. Principal Component analysis is used to get the top features that affect the dependent variable in the model and to summarise the data. The top features obtained are ‘MajorAxisLength’, ‘MinorAxisLength’, ‘Solidity’, ‘Extent’, ‘roundness’, ‘ShapeFactor1’, and ‘Eccentricity’. After standardizing the data, analysis is performed, where all the models give up to the mark results.

INTRODUCTION

Dry beans are put together in seed vessels and are members of the family of plants known as legumes. It is a plant that creates seeds in a fruit. The solid shape of these seeds helps differentiate them from each other. Most often, beans are oval, round, and kidney-shaped. The word “dry bean” refers to the beans which are dry- packed in air-tight bags or soaked in water and cooked.

Beans have a history of serving as easy to cook healthy food for a long time. They were manufactured over seven thousand years ago by the Indians in Mexico. There, they produced white beans, black beans, and a lot of other colour patterns. After some time, as Indians explored, traded and toured with other people and continents, dry beans and native farming habits unrolled progressively all over the North and South America. As a result, through early 1700s, they had turned into a very famous crop in the continents of Europe, Asia, and Africa. As of this day, they happen to be present in hundreds of classes in a large collection of taste, colour, and texture variations.

In this project, seven different kinds of dry beans were used with various features such as type, shape, form, and structure by the market situations. To segregate beans into seven categories with almost the same variables, a computer vision algorithm was designed and implemented. Pictures of 13,611 grains with 7 diverse listed dry beans were collected using a multispectral camera, which were then broken down and segmented to get 16 variables: 12 dimensions, and 4 shape forms.

OBJECTIVE

The purpose of this project is to find the variables that have the most effect to predict the Class of dry beans such that only those variables can be taken into consideration for classification of the model, to increase the speed and to prevent over- fitting, and then to learn about various classification algorithms and use them to classify the data of dry beans. Also, to see the pros and cons of each model, check how accurately the models have classified the data, the best method of classification on the dataset, and the ways that can be used to improve the accuracy.

METHODOLOGY

For this project, data is obtained from the venerable UCI Machine Learning repository. The dataset had the following:

- Numeric Columns:
    - Area(A) which represents the area of a bean.
    - Perimeter(P) which represents the circumference (length of border) of the bean.
    - Major Axis Length(L) which represents the length of the longest line of a bean.
    - Minor Axis Length(l) which represents the length of the longest line that is perpendicular to L.
    - Aspect Ratio(K) which represents the relation between L and l.
    - Eccentricity(Ec) which represents the eccentricity of the ellipse which have equal moments with the region.
    - Convex Area(C) which represents the number of pixels that exists in the smallest convex polygon which can contain A of a dry bean.
    - Equivalent diameter(Ed) which represents the diameter of a circle which have equal area of an dry bean.
    - Extent(Ex) which represents the ratio of the pixels in the bounding box to the dry-bean area.
    - Solidity(S) which represents the ratio of the pixels in the convex shell to those in dry-beans. Also called convexity.
    - Roundness(R) which represents roundness, found by the formula (4*pi*A)/(P^2).
    - Compactness(CO) which represents the roundness of dry-bean, found by the formula (E*d)/L
    - Shape Factor 1(SF1)
    - Shape Factor 2(SF2)
    - Shape Factor 3(SF3)

    - Shape Factor 4(SF4) following:
- String Columns: Class
    - ❑ Rows: 13,611 instances

Multivariate classification is carried out with the help of the software Python Programming Language as it is easy to use and it has lots of packages and libraries for Machine Learning applications. Different tools like pandas, numpy, matplotlib, sklearn, pydotplus are used to help in the process. Jupyter notebooks are very convenient as they work like a digital notebook where each line can be executed at a time.

STATISTICAL ANALYSIS

PRINCIPAL COMPONENT ANALYSIS:

Principal Component Analysis (PCA) is a method that is used for the computation of principal components and making their use for the performance of a basis change in the dataset, typically using solely the primary few principal components and nullifying the remainder of them.

It is employed in the exploratory data analysis (EDA) and for creating analytical and predicting models. It is usually used to reduce the dimensions of the data by projected every point from the data into solely the primary few principal components and get low- dimension data, all the while taking into account to preserve the maximum variation in the data. The 1st principal component can be defined as equivalent to a path which will maximize the data variation. The i^{th} principal component can be defined in a way that is orthogonal to the 1st i-1 principal components that have the power to maximize the variation of the predicted data.

For any of the objectives, it is easily proven that the major parts of PCA are the eigenvectors that are present as a part of co-variance matrix in the data. Therefore, these components are, most of the time, calculated either by using the breakdown of eigen- (decomposition) of the co-variance matrix of dataset or by single value- decomposition of a matrix of the dataset. The method of Principal Component analysis is the most- simple of the multi-variate analysis which has the basis of truest eigen-vectors, also which is in relation to factor analyses. Factor analyses usually includes assumptions which are specific to a particular field, around the most- basic structures and explains the eigen- vector of a matrix that is a little bit changed. Principal Component analysis is also in relation with CCA (Canonical Correlation analysis) which describes co-ordinate systems defined by the cross- covariance between various data- sets in an optimal sense. On the other hand, Principal Component analysis describes a separate co- ordinate structure which can be defined using variation in one data- set.

Principal Component analysis are often imagined as something that if we fit an ellipsoid having dimension p, to a data- set, then every axis in that ellipsoid is represented as a principal component. In the case of an axis in this ellipsoid being small, at that point the variation on that axis will also be small.

For the purpose of finding the axis of this ellipsoid, the average of every feature is subtracted from the data in order to make the origin the centre of the dataset. After that, covariance matrix is computed of the dataset, and eigen- values and eigen- vectors are calculated of the yielded matrix. For the next step, every orthogonal eigen- vectors are normalized into unit- vector. After that, all of the mutually orthogonal and having a unit eigen- vectors will easily be understood after their interpretation by the means of an axis of the ellipsoid which is close-fitted to given dataset. The selection of basis will in turn alter the co- variance matrix in a diagonalized method, where all diagonal components represents the variance of every axes. The amount of this variance which every eigen- vector will represent, will be computed after we divide the eigen- value of eigen- vector to that of the sum of all of the eigen-values.

Dimensionality Reduction

The step of reducing the dimensions is a very important step that is used for visualization and to process high- dimensional data, at the same time keeping most of the variation in the data. Like, if in a dataset, we select to keep only the primary two principal components, then we find that 2D plane of the high- dimensional data which affects the data in spreading out the maximum amount, therefore if this dataset comprises of clusters, then those also might be spreading out the most, which will thus be visible in a plot of 2D figure; on the other hand, if we randomly select two features from the original dataset, those clusters might be spreading out a lot less, and will actually be far more expected to significantly overlap each other, which will make those features vague.

In the same way, while doing regression analysis, we find that the more we have the number of independent variables, the more is the chance of over-fitting that model, which creates conclusions which miss the mark in generalising when applied on more data. One of the solutions to this problem is reducing the dimensions of the data to a small number of principal components, and then running a regression analysis on them, especially if there exists very high correlation among the independent features.

This is also an appropriate method to use when we have a lot of noise in the data. Noise, here refers to the irrelevant/ random information in the data. It refers to the part of the data that we don’t want, which makes it difficult for algorithms to detect the patterns in the dataset. Still, if we concentrate most of the total variation within the first one or two principal components, and compare it with equal noise variation, the proportional effects of this noise will be less, and those first one or two components will have a much greater signal- to- noise ratio. Principal Component Analysis, therefore, will impact out analysis in a good way by focusing a lot of the signal in the primary one or two principal components, that, the dimension reducing technique will apprehend in a useful way; whereas the other principal components might have a lot of noise, so we can dispose them while not facing a lot of loss.

Since PCA extracts the data from features, we can’t really pinpoint to a set of features and say that these are the most important ones from this dataset, instead it combines the useful information from all the features in a way and creates new columns, while retaining all the rows of data.

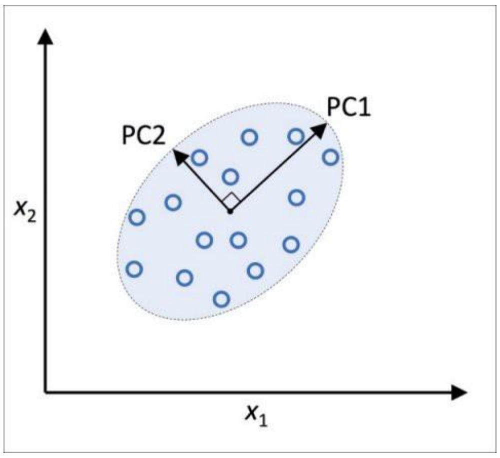


Figure 1.1

<u>Standardizing the data</u>

This is a very important step before applying Principal Component analysis on the data. The main objective of this step is to transform the independent variables in such a way that they all have the same contribution for the analysis. As the method of Principal Component analysis gives us a feature subspace which consists of the maximum variation on the axis, it seems normal and necessary to standardise given dataset. The data is transformed such that the mean of every explanatory variable becomes equal to 0, and the variance becomes equal to 1.

In precise words, the reasoning behind applying this method before we apply Principal Component analysis is because PCA is very sensitive when we talk about the variance of primary variables. In simpler terms, if we don’t apply Standard Scaler method to the data before applying PCA, then the variables having huge difference between the Maximum and minimum values will cause more effect as compared to the ones with a small difference between the Maximum and minimum values. This will, in turn, help in minimizing the bias towards those variables.

This method of Standardization is implemented by subtracting the average value for each variable, from the value and dividing the result by standard deviation for each variable, for each value present in a row for all the columns.

$$z = \frac{value - mean}{standard\ deviation}$$

Figure 1.2

Construction of Principal Components by PCA

By the use of Principal Component analysis, principal components are made such that the first variable of the resulting dataset will justify the maximum variation from the data, the second one will account for the second highest variation from the data, and so on. Putting it in mathematical terms, it represents a line which will maximise the variation, that is, the mean of the square distance when taken from expected values.

The next component will be calculated by using the same method, having one change in the background that this one should be un- correlated with and right- angled to the first one and it accounts for the second highest variation from the data.

This approach for calculating principal components continues till the time we have p number of principal components constructed, where p is equal to the original number of explanatory variables.

There are two methods for applying PCA, after Standard scalar method, first one being the usual: creating a new dataset having the number of columns equal to the original dataset, and another one is that you can indicate how much variance you want the resulting data to explain, and PCA will automatically select the number of variables that are needed for that accuracy such that the resulting dataset will contain less number of variables than the original dataset and variation explained will be greater than or equal to the specified accuracy. For my dataset, I mentioned an accuracy of 99% while applying PCA, and out the total 16 variables from the original dataset, the method retained 7 variables that explained 99% variation in my data.

After that, I specified the greatest variables which have the most contribution to these principal components, in decreasing order, by calculating the absolute values of all the variables on each component. A huge absolute value indicates that that feature has a huge impact. And the seven most important features that I got are

‘MajorAxisLength’, ‘MinorAxisLength’, ‘Solidity’, ‘Extent’, ‘roundness’, ‘ShapeFactor1’, and ‘Eccentricity’.

So, we’ve got the top seven features after applying PCA on this data, so as to retain 99% of accuracy for the classification: but the result of the top seven features that we’ve got, that is, MajorAxisLength, MinorAxisLength, Solidity, Extent, roundness, ShapeFactor1, and Eccentricity are not the same variables that we had- instead, these variables represents that after applying Principal Component Analysis on this dataset, while setting the accuracy level to 99%, when we get the top seven features, in that new data, these seven variables are the ones that contribute the most to the results. We can not get the correct and accurate interpretation of the top most independent variables after applying Principal Component Analysis to the data. But one very big advantage of this process is that it makes the resulting variables independent of each other- which is an assumption for most of the regression and classification algorithms.

Explained Variance

The explained variance is a method of telling you the amount of information or variation that should be credited to every variable of the resulting dataset. It is calculated by dividing the variation explained by a particular component by that of all the components. This concept is a key to understanding PCA as when we convert our data and reduce the number of variables, we know and specify the amount of variation that we want to be explained by the new data. For example if we convert a 4- dimension data to a 2- dimension data, we miss a bit of the variation while doing that.

In the dataset, the result of this is: array([0.55466439, 0.26430973, 0.08006564, 0.0511408, 0.02739293, 0.01149761, 0.00697651]). So, the first component explains 55.47% of the variation of the whole data, second component explains 26.43% of variation in the data and so on.

Visualize

Whenever we work with data, the biggest problem that we have is the huge volume of it, and the big number of features that it has. For solving such a challenge, we need to use to visualize the data to explore more into what it basically represents. Visualization solves a lot of our problems in today’s world as it helps us by showing various features and their distribution. Because of a lot of variables in the dataset, visualization sometimes becomes a problem, but because we have used PCA, it makes this process a little simpler as we can take the first two variables which explains most of the variation (55.47% + 26.43% = 81.90% in our case) and plot them in a 2- dimensional space. So, I’ve plotted the data using the first two principal components and have got promising results.

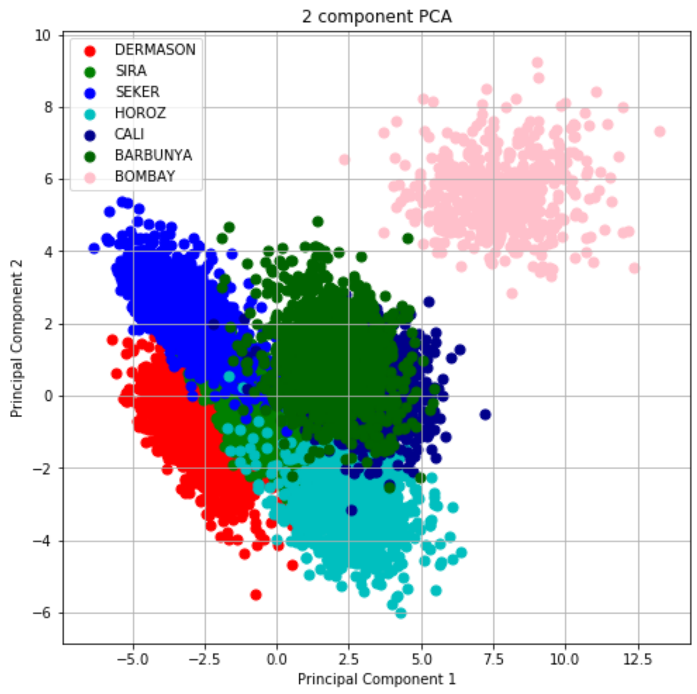


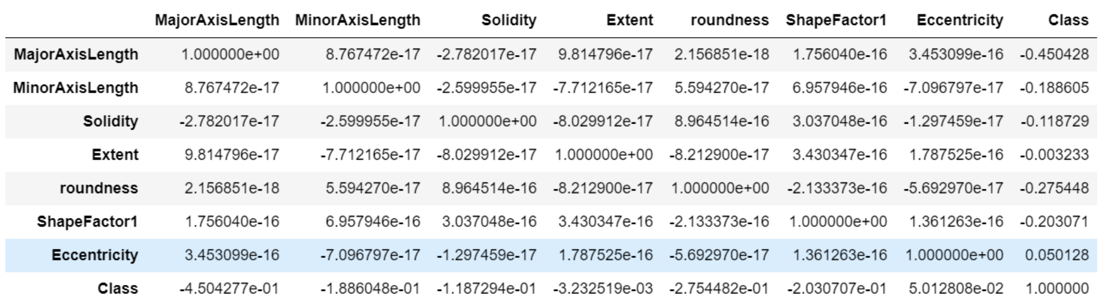
Figure 1.3

The different colours represent the seven different classes of the dependent variable.

<u>Covariance Matrix</u>

This matrix shows us the degree of dependence between the independent variables. The (ith, jth) positions of this matrix represents the covariance among the ith, and jth independent variables.

Note: This matrix is symmetric in the sense that even if we transpose it, we’ll get the same matrix. Also, all the values on the diagonal are equal to one as they represent the value of co- variance of the variable with itself. All the values should be close to 0.



	MajorAxisLength	MinorAxisLength	Solidity	Extent	roundness	ShapeFactor1	Eccentricity	Class
MajorAxisLength	1.000000e+00	8.767472e-17	-2.782017e-17	9.814796e-17	2.156851e-18	1.756040e-16	3.453099e-16	-0.450428
MinorAxisLength	8.767472e-17	1.000000e+00	-2.599955e-17	-7.712165e-17	5.594270e-17	6.957946e-16	-7.096797e-17	-0.188605
Solidity	-2.782017e-17	-2.599955e-17	1.000000e+00	-8.029912e-17	8.964514e-16	3.037048e-16	-1.297459e-17	-0.118729
Extent	9.814796e-17	-7.712165e-17	-8.029912e-17	1.000000e+00	-8.212900e-17	3.430347e-16	1.787525e-16	-0.003233
roundness	2.156851e-18	5.594270e-17	8.964514e-16	-8.212900e-17	1.000000e+00	-2.133373e-16	-5.692970e-17	-0.275448
ShapeFactor1	1.756040e-16	6.957946e-16	3.037048e-16	3.430347e-16	-2.133373e-16	1.000000e+00	1.361263e-16	-0.203071
Eccentricity	3.453099e-16	-7.096797e-17	-1.297459e-17	1.787525e-16	-5.692970e-17	1.361263e-16	1.000000e+00	0.050128
Class	-4.504277e-01	-1.886048e-01	-1.187294e-01	-3.232519e-03	-2.754482e-01	-2.030707e-01	5.012808e-02	1.000000

Figure 1.4

<u>Advantages of Principal Component Analysis:</u>

- It gets rid of and transforms correlated variables. As seen from the above figure, there does not exist any correlation between them.
- It helps in preventing over- fitting of the data by eliminating a lot of the features.
- It changes the dataset to the one with low dimension which makes it easier to visualize.

Dis- advantages of Principal Component Analysis:

- The method of PCA cannot be utilised when we have less data.
- The low dimension of data represented by this method do not reserve a lot of the variation from the original data.
- Outliers have to be removed before applying this method.
- Because it is a feature extraction method, we cannot interpret the new features which are formed after applying this method.

A. DECISION TREE CLASSIFIER

Decision Tree is a Tree Based algorithm, that is used for both regression as well as classification purposes. As the dependent variable in the dataset is categorical, so it is used it to classify the data here. These tree type algorithms are thought of as very good and accurate supervised- algorithms. These processes help in predicting the models pretty accurately, provide great stability, and are easy to interpret. In contrast to linear methods, these help in mapping non- linear relations very nicely. They adjust according to the problem that somebody is facing, that is, when the user have to classify or regress the data. Decision trees, along with Random Forest, and Gradient Boosting algorithms such as XGBoost are very famous when we need to classify data to find a solution in the field of Data Analysis. Therefore, it is very important for people who are trying to get into this field, to know the working of these algorithms, from scratch, along with knowing the correct way to apply them to data.

Classifying data is a two- step process- first one being the stage where learning happens, that is, developing the model created by using training data and let the machine learn the patterns from the dataset; and the second one is the stage where prediction happens, that is, using the model to find the expected values, without specifying the original values in the dependent variable column, and them comparing these predicted values with the original values. Decision Tree classifier is a very easy and famous algorithm used to classify the data and for understanding and interpretation purposes. Since Decision tree algorithm is used for both regression and classification, a gist of both these methods will be included.

Decision tree regressor is used when we want to predict a continuous dependent variable, using one or more than one continuous or categorical numerical independent variables. That is, when we don’t have to categorise our data into different segments, instead find a continuous value, for example, trying to predict the marks a student may get using the number of hours he or she studied.

On the other hand, Decision tree classifier is used when we want to predict a categorical dependent variable, using one or more than one continuous or categorical numerical independent variables. That is, when we want to distribute our data into different categories, class, or group (number of distinct categories can be equal to two: known as binary classification, and more than two: known as multinomial classification), for example, if in the previous example, we make 10 categories of this data that says that out of 100, if a student gets marks in 0-10 range, then that is one category, 11-20 is another category, and so on.

<u>Basis of Decision tree algorithm</u>

It is a flow- diagrammable construction made up of nodes, where each one of them denotes an attribute; the branches represent a decision rule, while every leaf node denotes the outcome from that branch. The highest top node is called a root node, like the basis of an actual tree, and it learns on partitioning the data on the basis of the feature value. This is used to partition the tree in a recursive way, and is therefore known as recursive- partitioning. The easy to understand structure of a decision tree aids us to make a decision. It is visualized in a flow- chart that is easy to comprehend.

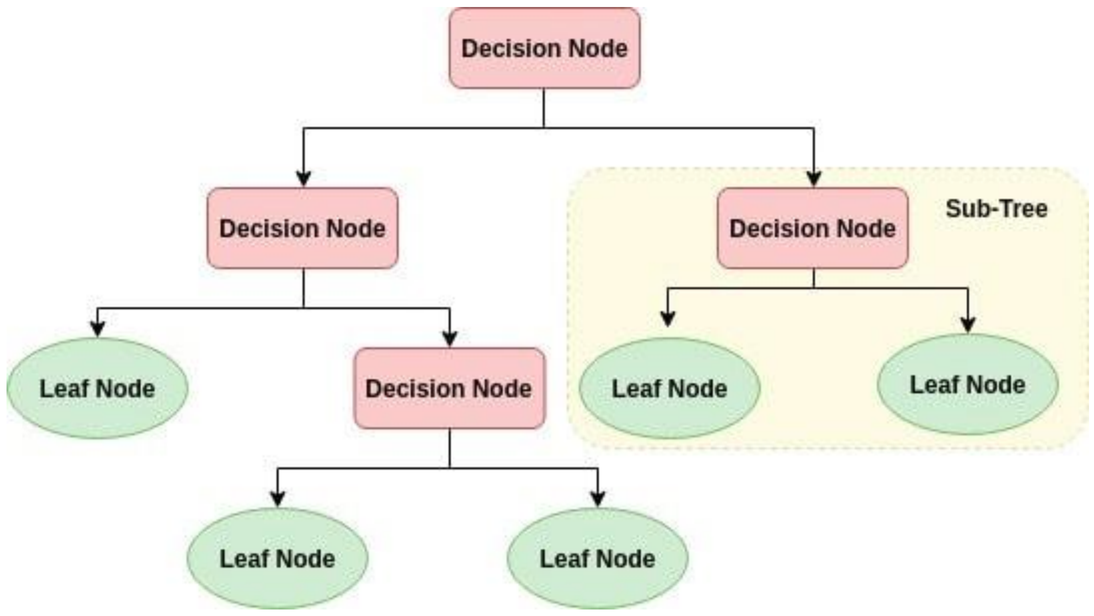


Figure 2.1

A decision tree classifies the data with the help of Top- Down approach, where it sorts down the examples in the tree, starting from the root node, to a leaf node, where this leaf node provides the output after classifying for each example. Every node from the tree performs like a trial situation used for some feature, and every hint has the chance of possibly being the solution for this situation. The same procedure is repeated for all the nodes with new roots as the sub- nodes.

Figure 2.2

Working of this algorithm

First, it selects the topmost feature (attribute) by the use of ASM (Attribute Selection Measures) and splits the dataset rows. Then, to divide the data in small subgroups, it makes the topmost feature act as a decision node. Then, start building the tree with repeated use of this procedure for every node till one out of these following conditions satisfies:

- Each and every tuple belongs in the matching feature.
- All of the attributes are exhausted.
- All of the instances are exhausted.

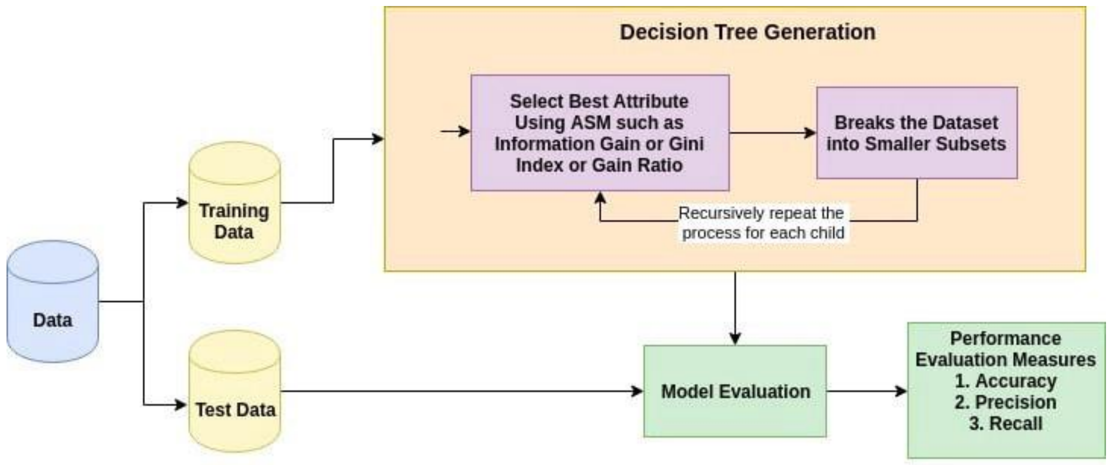

Decision Tree Generation
Select Best Attribute Using ASM such as Information Gain or Gini Index or Gain Ratio
Breaks the Dataset into Smaller Subsets
Recursively repeat the process for each child
Data
Training Data
Test Data
Model Evaluation
Performance Evaluation Measures
1. Accuracy
2. Precision
3. Recall

Figure 2.3

Understanding the Mathematics!

Measures that are used to select features: Attribute Selection Measure is an investigative method that is used to select the criteria to split which partitions the data in the greatest imaginable way. It's additionally called Splitting rules as it assists in determining break points intended for tuples for certain nodes. This method of Attribute Selection Measure assigns a position for every attribute with the help of explanation of current data. The feature having the largest score will then be designated to be the feature used for split. When the features are continuous, splitting arguments intended for divisions of every branch are also needed to be defined. The highest used actions for this are Information Gain, Gini Index, and Gain Ratio.

I. Information Gain

It is a numerical method which can specify and tell the measure of separation of our validation data by a specific feature column in accordance with the goal's (target) classifying algorithm.

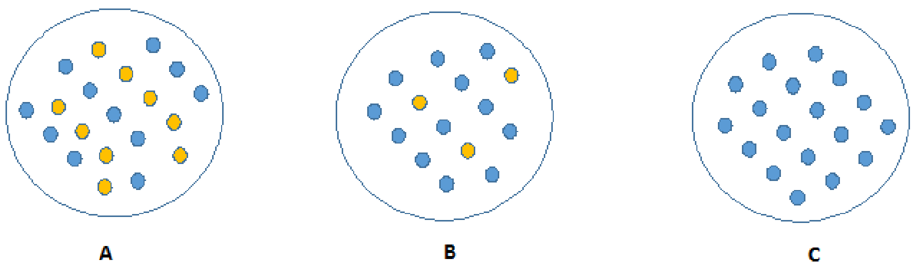


Figure 2.4

From the above figure, we can easily tell the nodes and rank them on the basis of difficulty that will be faced if we try to describe them. The easiest one is C as all the data is similar, then B node as there exists a few dissimilar data in this one, and then A node as it seems like a combination of blue and red dots distributed almost evenly in the data. That is, C is purest, while A is the opposite of C, that is, the most impure.

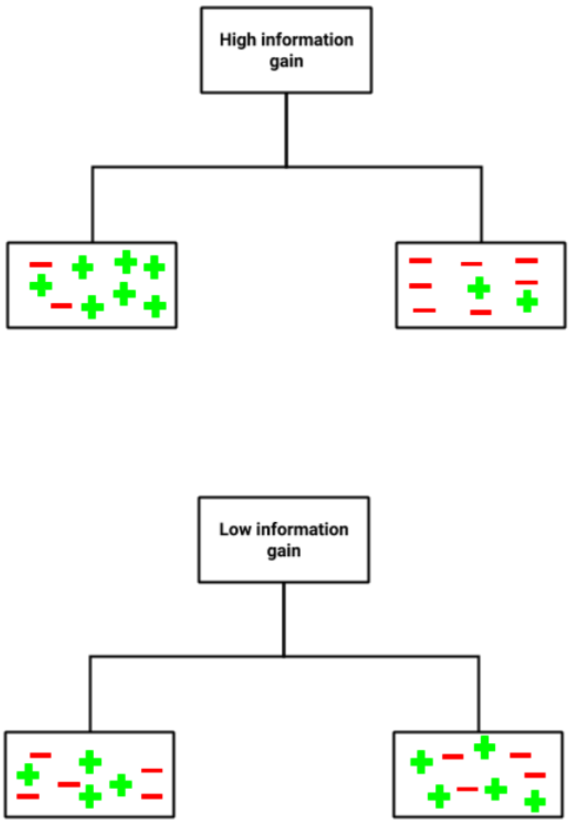


Figure 2.5

Let us consider another example to better understand the concept of Information Gain, and look at the above figure, it can easily be seen that if the data is almost equally distributed with two or more than two different classes, it is an indication of low information gain, and it becomes relatively difficult for us to segment it. On the other hand, if the data is not equally distributed, that is, if it classifies in an irregular sense, giving more weightage to one than the others, then that is an indication of high information gain, and that makes it easier to segment it.

To better understand this concept of information gain, ‘entropy’ will be defined first.

Entropy

The idea of Entropy was coined by Claude Shannon, that is used to measure how impure the given system is. This term was first taught in Science classes to refer to the haphazardness, surprise or uncertainty of the dataset. When entropy decreases, we see good information gain, so a decrease in entropy, that is, randomness is good for classification problems. According to Shannon, entropy shows a total limit of mathematics as to measure how finely we can compress sourced dataset without loss or with a little loss into the perfect noise-less channel. He reinforced the resultant significantly for the channel with a lot of noise when he formulated the coding theorem of noisy channel.

For example, let a person has a wheel of lottery having a total of 100 green balls. This is 100% pure as there is no other coloured ball present here, so the entropy (or uncertainty) is equal to zero. Now let, he replaces 25 from this set by 25 red colours, and 25 from this set by blue coloured balls.

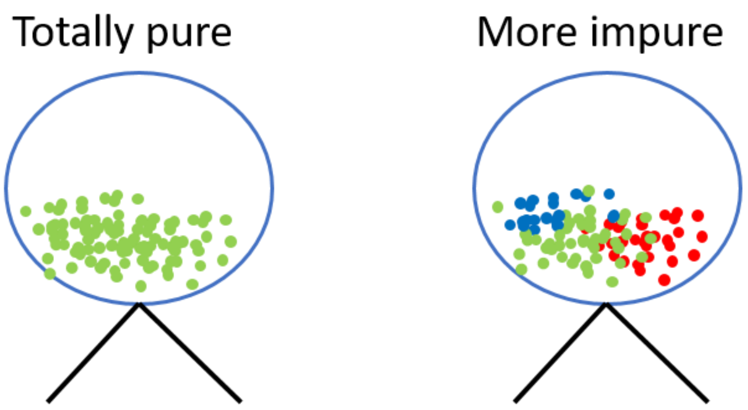


Figure 2.6

So, the new probability of getting the green- coloured balls has come down to 0.5 from 1.0. We can see that with the increase in impure set, there is a decrease in the pure set. Therefore, it can be said that a dataset of high impurity makes it have a large entropy and a dataset of low impurity makes it have a small entropy. Entropy equals zero for a totally pure dataset. The value of entropy varies in the range of 0 and 1.

Process to calculate entropy is

$$\text{Entropy} = -p \log_2 p - q \log_2 q$$

After calculating the entropy of the main (parent) nodes, calculating the entropy for every single (individual) node of split in calculating weighted- average of every sub-nodes from a split will give its entropy.

Where p represents positive and q represents negative for the node. It is preferred to have less entropy for good gain in information.

The distinction in the entropy before the split and average entropy after the split in data on the basis of given features. It is calculated as

$$\textit{Information Gain} = \textit{Entropy(parent node)} — [\textit{Avg Entropy(children)}]$$

II. <u>Gini</u>

This is a quantity that represents the degree of inequality in some dataset. For example, in one column of a dataset, all the rows have the same value, then Gini for that row is equal to 1, whereas if all of the rows have different values, then Gini is equal to 0. So, the value of Gini lies between 0 and 1. When the values of Gini are closer to 1, the data is close to homogenous. When we are classifying the data and (or) applying regression on the data, then Gini is used to split it into two (binary).

Process to find Gini is

First, Gini is calculated for sub- nodes with the help of sum of squares of positive and negative outcome chances, that is, $p^2 + q^2$. Then, Gini is calculated for splits with the help of biased (weighted) Gini index for every node in the split.

Gini impurity is calculated as the difference of Gini index from 1, that is,

G. I. = 1 – G, where G.I. is Gini Impurity and G is Gini index.

This is a widely used feature while applying tree- based classifiers.

So, now after knowing all of this about Decision Tree classifier, and applying this classifier on the dataset of Dry Beans, that is, after fitting this algorithm on the training set, and predicting values of the dependent variable on the test set, using values of independent variable on the test set, an accuracy of 89.45% is established.

Accuracy of a classifier can be calculated if we compare the theoretical values of our dataset with the values that we got after the prediction of the algorithm. This accuracy shows us that the data have been classified pretty accurately and can thus be used to predict future values for this dataset.

<u>Visualization!</u>

For the visualization of decision tree classifier, the function ‘export_graphviz’ from library ‘Scikit-learn’ is used to display the tree in the Jupyter notebook, as well as to export it to be saved in the device., ‘write_png’ is used for which ‘graphviz’ and ‘pydotplus’ are used. Figure 2.7 is the Decision Tree that is created from these commands. As we can see, we can’t understand much from it, so Figure 2.8 is a much closer look of one of the nodes from this figure.

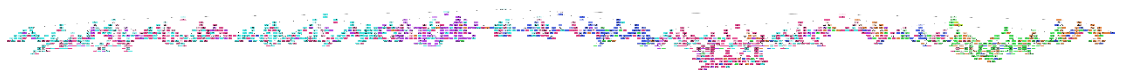
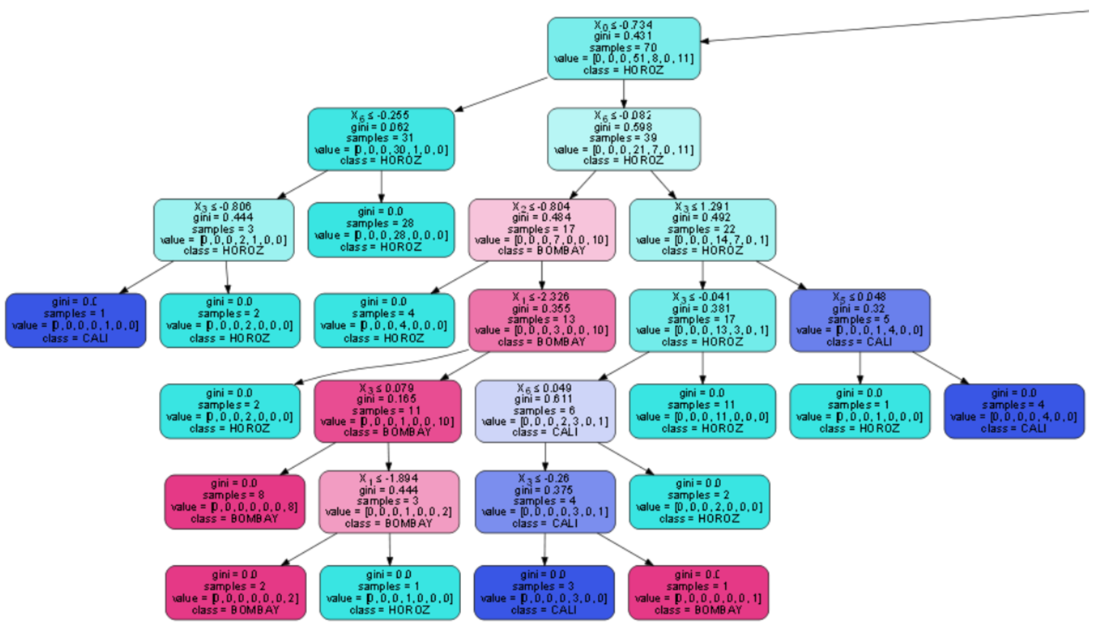


Figure 2.7



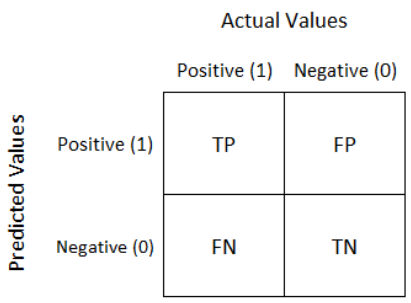
$X_0 \leq -0.734$ gini = 0.431 samples = 70 value = [0, 0, 0, 51, 8, 0, 11] class = HOROZ
$X_6 \leq -0.255$ gini = 0.062 samples = 31 value = [0, 0, 0, 30, 1, 0, 0] class = HOROZ
$X_6 \leq -0.082$ gini = 0.598 samples = 39 value = [0, 0, 0, 21, 7, 0, 11] class = HOROZ
$X_3 \leq -0.806$ gini = 0.444 samples = 3 value = [0, 0, 0, 2, 1, 0, 0] class = HOROZ
gini = 0.0 samples = 28 value = [0, 0, 0, 28, 0, 0, 0] class = HOROZ
$X_2 \leq -0.804$ gini = 0.484 samples = 17 value = [0, 0, 0, 7, 0, 0, 10] class = BOMBAY
$X_3 \leq 1.291$ gini = 0.492 samples = 22 value = [0, 0, 0, 14, 7, 0, 1] class = HOROZ
gini = 0.0 samples = 1 value = [0, 0, 0, 0, 1, 0, 0] class = CALI
gini = 0.0 samples = 2 value = [0, 0, 0, 2, 0, 0, 0] class = HOROZ
gini = 0.0 samples = 4 value = [0, 0, 0, 4, 0, 0, 0] class = HOROZ
$X_1 \leq -2.326$ gini = 0.355 samples = 13 value = [0, 0, 0, 3, 0, 0, 10] class = BOMBAY
$X_3 \leq -0.041$ gini = 0.381 samples = 17 value = [0, 0, 0, 13, 3, 0, 1] class = HOROZ
$X_5 \leq 0.048$ gini = 0.32 samples = 5 value = [0, 0, 0, 1, 4, 0, 0] class = CALI
gini = 0.0 samples = 2 value = [0, 0, 0, 2, 0, 0, 0] class = HOROZ
$X_3 \leq 0.079$ gini = 0.165 samples = 11 value = [0, 0, 0, 1, 0, 0, 10] class = BOMBAY
$X_6 \leq 0.049$ gini = 0.611 samples = 6 value = [0, 0, 0, 2, 3, 0, 1] class = CALI
gini = 0.0 samples = 11 value = [0, 0, 0, 11, 0, 0, 0] class = HOROZ
gini = 0.0 samples = 1 value = [0, 0, 0, 1, 0, 0, 0] class = HOROZ
gini = 0.0 samples = 4 value = [0, 0, 0, 0, 4, 0, 0] class = CALI
gini = 0.0 samples = 8 value = [0, 0, 0, 0, 0, 0, 8] class = BOMBAY
$X_1 \leq -1.894$ gini = 0.444 samples = 3 value = [0, 0, 0, 1, 0, 0, 2] class = BOMBAY
$X_3 \leq -0.26$ gini = 0.375 samples = 4 value = [0, 0, 0, 0, 3, 0, 1] class = CALI
gini = 0.0 samples = 2 value = [0, 0, 0, 2, 0, 0, 0] class = HOROZ
gini = 0.0 samples = 2 value = [0, 0, 0, 0, 0, 0, 2] class = BOMBAY
gini = 0.0 samples = 1 value = [0, 0, 0, 1, 0, 0, 0] class = HOROZ
gini = 0.0 samples = 3 value = [0, 0, 0, 0, 3, 0, 0] class = CALI
gini = 0.0 samples = 1 value = [0, 0, 0, 0, 0, 0, 1] class = BOMBAY

Figure 2.8

<u>Confusion Matrix</u>

This is a measure to check how well our algorithm has performed for a binary or multinomial classification. This is a tabulated form represented as a square matrix which comprises of various combinations of the theoretical and the casted values. It’s generally used to measure Recall, Precision, and accuracy in a model.

Now, let us understand some terms that are extremely important to understand Recall and Precision.



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 2.9

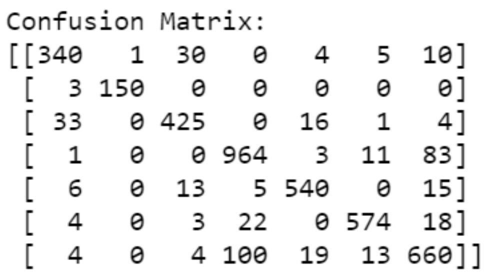
1. True Positive (TP) is when your predicted value of Positive is equal to the theoretical value of the term being positive.

2. True Negative (TN) is when your predicted value of Negative is equal to the theoretical value of the term being negative.

3. False positive (FP) is when your predicted value of Positive is not equal to the theoretical value of the term actually being negative. This is known as the Type- 1 error.

4. False Negative is when your predicted value of Negative is not equal to the theoretical value of the term actually being Positive. This is known as the Type- 2 error.

The following Figure 2.10 is the Confusion matrix of the dataset of dry bean after fitting the algorithm of Decision Tree Classifier and the results seem very promising.



```
Confusion Matrix:
[[340   1  30   0   4   5  10]
 [  3 150   0   0   0   0   0]
 [ 33   0 425   0  16   1   4]
 [  1   0   0 964   3  11  83]
 [  6   0  13   5 540   0  15]
 [  4   0   3  22   0 574  18]
 [  4   0   4 100  19  13 660]]
```

Figure 2.10

Classification Report

This shows us the values of Precision, Recall, F1- Score, and Support in the fitted model.

1. Precision is that quality in a classifying algorithm that tells no to predict something as positive when it is negative. It is calculated by the ratio of True Positives by the sum of True Positives and False Positives.

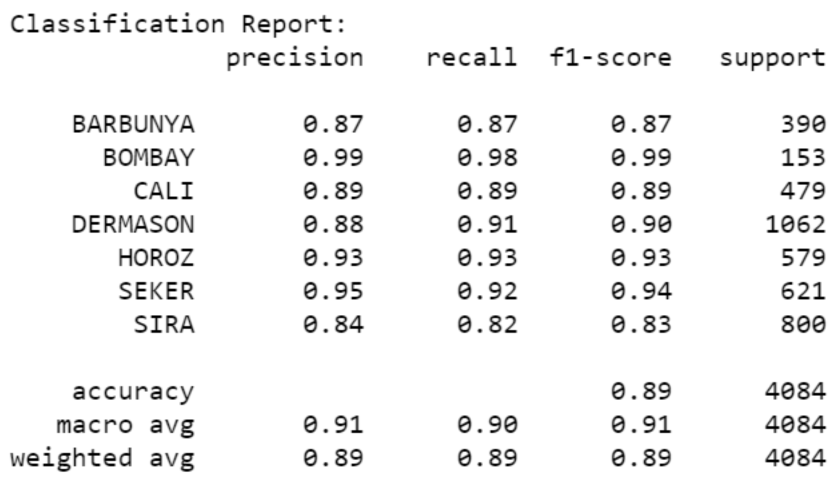
2. Recall is that quality in a classifying algorithm that tells us the quantity of Positives we got right. It is calculated by the ratio of True Positives by the sum of True Positives and False Negatives.

3. F1- Score is that quality in a classifying algorithm that tells us the quantity of Positives that were correct.

$$\text{F1 Score} = 2*(\text{Recall} * \text{Precision}) / (\text{Recall} + \text{Precision})$$

4. Support is that number in a classifying algorithm that shows us the real existences of the given class for the given data. Unequal support for trained data indicates a weakness in the structure of reported score in the classifying algorithm which indicates that data should be rebalanced.

The following Figure 2.11 is the Classification Report of the dataset of dry bean after fitting the algorithm of Decision Tree Classifier and the results seem very promising.



```
Classification Report:
              precision    recall  f1-score   support

    BARBUNYA       0.87      0.87      0.87       390
      BOMBAY       0.99      0.98      0.99       153
        CALI       0.89      0.89      0.89       479
    DERMASON       0.88      0.91      0.90      1062
       HOROZ       0.93      0.93      0.93       579
       SEKER       0.95      0.92      0.94       621
        SIRA       0.84      0.82      0.83       800

    accuracy                           0.89      4084
   macro avg       0.91      0.90      0.91      4084
weighted avg       0.89      0.89      0.89      4084
```

Figure 2.11

<u>Advantages of Decision Tree Algorithm:</u>

- While working with raw data, and doing pre- processing of the data, this algorithm needs a little less struggle.
- This algorithm does not have the requirement of normalizing of the Data- set.

- We do not need to scale our data- set before applying this algorithm.
- Even if we have absent values in our data- set, it does not cause it much harm.
- This algorithm is easily explainable to people at all levels.

Dis- advantages of Decision Tree Algorithm:

- A little modification of the Data- set causes big changes while building the algorithm, while causes it to unbalance.
- A lot of complex calculations involved in this algorithm if we compare with similar Classifiers.
- This algorithm takes a lot of time while training the data- set.
- This algorithm is not good for regressive problems where we need to predict real values for the dependent variable of the data- set.

B. XGBoost CLASSIFIER

This classifier is based on Decision Tree classifier and is an algorithm which works by making the use of the optimization algorithm Gradient Descent algorithm on data. When working with dataset having little or no structure, we tend to apply Neural Networks, like in problems concerning working with image data or texts data, etc. Likewise, when working with small structure, or medium structure data, the algorithms which have the basis as Decision Trees are used as they give very promising results for these data types. XGBoost Classifying algorithm is an optimisation of Gradient Boosting which parallel processes the data, uses tree- pruning, handles missing data, and uses regularisation to evade over- fitting and biasness. XGBoost algorithm is very versatile. It serves a lot of purposes, like it solves regressive problems, classification problems, rank- based problems, and also any problem for predictive output that is defined by the user. It is a very portable algorithm as it can run nicely on Windows, OS X, and Linux. It is a supporter of most big and widely used coding languages such as Python, Java, Julia, Scala, C++, and R. The instinct behind this algorithm is that a Decision- Tree, in the most basic way, are very easy to envision, and can be interpreted very easily. But creating a more detailed and better version of Decision Tree is a little complicated. XGBoost is also called Extreme Gradient Boosting algorithm as it can be thought of as 'Gradient Boosting' of steroids. Now, let us dive a little deeper to understand what 'Boosting' and 'Gradient Boosting' is.

Boosting:

In a basic algorithm like Decision Tree Classification, one model is trained using the Data and the dependent variable is predicted using that. Different measures can be used before and after implementing the algorithm, like getting the best features, tuning the features, standardization, PCA, measures for under- fitting or over- fitting, etc., but overall, only one model is fitted, predicted, and tested. If an ensemble is built, even, we make it in a way so as to train and apply various models in a separate manner. But if we consider Boosting, it is a little different from these. It is a computational process this works in a loop repeatedly. It is, in technical terms, a collaborative method in which a lot of models are collected to combine and performed as one, nonetheless, this is a little crafty and wisely made. What Boosting does is, it takes all of the models, and trains them separately, away from each other, boosting as it trains them, that is, in simple terms, after one model is trained, it identifies the errors made, and corrects them and apply what's learnt from this in the next model that it trains, and it keeps doing this, training better models till it can't enhance the model any more.

The best thing about this technique or reiterative method is it focuses on the betterment or correction of the faults that were made by the previous model. In the basic ensemble techniques in which every model is trained in seclusion with one another, every model ends up creating the exactly identical errors. Now, let us see how this is different from 'Gradient Boosting'.

Gradient Boosting:

'Gradient Boosting', as the name suggests, is made up of two terms: 'Gradient' and 'Boosting'. Now, we know all about what Boosting does, let us dive deeper and get to know more about the role of Gradient in this method.

'Gradient Boosting' explains the 'Boosting' in a problem in which mathematical optimized algorithm is required such that the focus goes on minimizing a function of loss in such a way as to add the weakly learning elements in the model by the use of a method called 'Gradient Descent. Gradient Descent is a method having order one, which is used for optimizing problems to find a local- minima in a function that can be differentiated. Since the method of Gradient Boosting has a basis to minimization of a loss function so that is why there can be various functions of loss, and an flexible method, which could help in giving us a result for various problems for regressive, or classifying algorithms for data. In simple terms, Gradient Boosting is an algorithm in which learning new elements are added while this procedure is running, that is, there are cases where a new tree is added to the prevailing set of already present trees, without the removal of any tree. This new addition to the existing model is due to the use of the algorithm of Gradient Descent which points and tells us about the weakly learning element. The total error of the strongly learning element is minimised to calculate how much every tree contributes to the model.

The method of ‘Gradient Boosting’ is not used to change the sampling distribution while the weakly learning elements are training using the data- set model left- over faults in a strongly learning element. Because this algorithm trains on the residuals in the method, it takes the mis- classified values to be of much greater importance. That is, the weakly learning elements get added to those focus areas in which the already present learning elements perform badly. The Gradient Descent algorithm that optimises the process and minimises the total error of the strongly learning element is the basis for influence and in getting the weakly learning elements for the overall prediction of the model.

<u>Intuition for XGBoost:</u>

The algorithm of XGBoost is the best mixture that has software as well as hardware betterment methods so as to give very good outcomes with the use of little computational power in very little time. This algorithm is very similar to ‘Gradient Boosting’ in the sense that these two methods use ensemble trees and boost weakly learning elements with the use of the algorithm of ‘Gradient Descent’. Still, XGBoost method performs a little better than Gradient Boosting Algorithm as it optimises the system and enhances the algorithm.

<u>System Optimization:</u>

I. Parallel Processing: XGBoost algorithm implements and processes its loop where it builds trees in a parallel manner. It is because the algorithm uses loops which can be interchanged while it is building the base learning elements, with the outer and the second inner loops which computes tree leaf nodes and the features, respectively. The loop nestling somehow confines parallelism, as while innermost loop is incomplete, it can not start outer loop. So, for the betterment of the time, it takes for running, it does some global scanning for every instance and sorts by the use of parallelized thread, to change internally the order of the loops. Because of this adjustment, the performance of the algorithm is improved, as this counteracts the parallelized computational overhead.

II. Tree Pruning: The criteria to stop the tree splits in the Gradient Boosting Method is acquisitive naturally, which is dependent of the minus loss criteria where splitting happens. For XGBoost Algorithm, to improve the calculative power in a significant manner, the algorithm uses the "max_depth" variable as the "depth-first" tactic in the place of first indicator, to prune the tree backwards.

III. Hardware Optimization: The XGBoost algorithm, designed in a way, uses hardware resources efficiently. It accomplishes this with caches aware with the help of allocation of various inner shields in every thread with the objective of storing "Gradient Statistics". Additional improvement like "out-of-core" computation optimizes the accessible (available) memory as it handles large Data Frame which were not accessible to process by desktop.

Algorithm Enhances:

I. Regularization: XGBoost does a lot of fine- tuning on complicated models by using "LASSO" , that is, L1, and "RIDGE", that is, L2, to solve the problem of over- fitting of the data.

II. Alertness of Sparsity: This algorithm let the sparse variables enter in a natural sense for different observations as it learns the top blank values which depends on the train examples of loss by itself. It is also very efficient in handling various kinds of sparse patterns in the dataset.

III. Biased Quantile Sketch: This algorithm also uses the "Distributed Weighted Quantile sketch" algorithmic rule so as to discover the optimum splitting spaces in the weighted data in an effective way.

IV. Cross-Validation: The algorithmic rule is made such that it employs and cross- validates the technique by itself in every step of the loop, which helps us as then it removes the necessity to run the step for this examination, and also, the need to tell the precise amount of repetition the algorithm requires for optimal result in one go.

So, now after knowing all of this about XGBoost classifier, and applying this classifier on the dataset of Dry Beans, that is, after fitting this algorithm on the training set, and predicting values of the dependent variable on the test set, using values of independent variable on the test set, an accuracy of 92.90% is established, which is much more that of Decision Tree classifier on this Dataset of Dry Beans.

Accuracy of a classifier can be calculated if we compare the theoretical values of our dataset with the values that we got after the prediction of the algorithm. This accuracy shows us that the data have been classified pretty accurately and can thus be used to predict future values for this dataset.

Confusion Matrix

This is a measure to check how well our algorithm has performed for a binary or multinomial classification. This is a tabulated form represented as a square matrix which comprises of various combinations of the theoretical and the casted values. It’s generally used to measure Recall, Precision, and accuracy in a model.

Now, let us understand some terms that are extremely important to understand Recall and Precision.

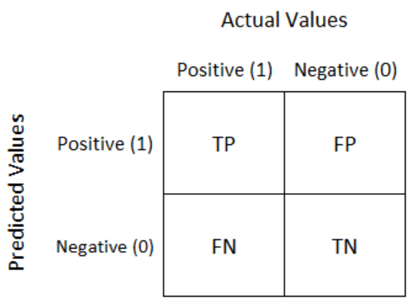


Figure 3.1

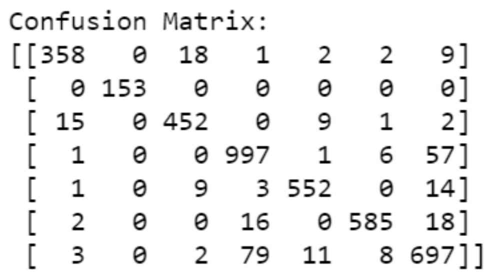
1. True Positive (TP) is when your predicted value of Positive is equal to the theoretical value of the term being positive.

2. True Negative (TN) is when your predicted value of Negative is equal to the theoretical value of the term being negative.

3. False positive (FP) is when your predicted value of Positive is not equal to the theoretical value of the term actually being negative. This is known as the Type- 1 error.

4. False Negative is when your predicted value of Negative is not equal to the theoretical value of the term actually being Positive. This is known as the Type- 2 error.

The following Figure 3.2 is the Confusion matrix of the dataset of dry bean after fitting the algorithm of XGBoost Classifier and the results seem very promising.



```
Confusion Matrix:
[[358   0  18   1   2   2   9]
 [  0 153   0   0   0   0   0]
 [ 15   0 452   0   9   1   2]
 [  1   0   0 997   1   6  57]
 [  1   0   9   3 552   0  14]
 [  2   0   0  16   0 585  18]
 [  3   0   2  79  11   8 697]]
```

Figure 3.2

<u>Classification Report</u>

This shows us the values of Precision, Recall, F1- Score, and Support in the fitted model.

1. Precision is that quality in a classifying algorithm that says not to predict something as positive when it is negative. It is calculated by the ratio of True Positives by the sum of True Positives and False Positives.

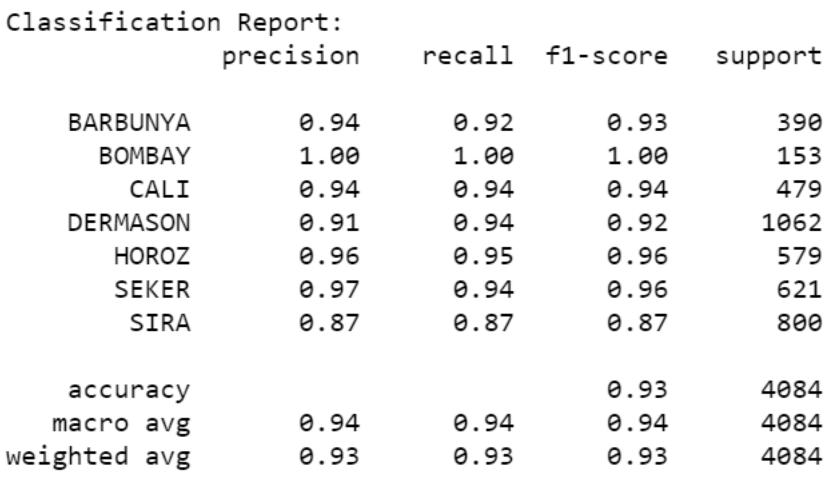
2. Recall is that quality in a classifying algorithm that tells us the quantity of Positives we got right. It is calculated by the ratio of True Positives by the sum of True Positives and False Negatives.

3. F1- Score is that quality in a classifying algorithm that tells us the quantity of Positives that were correct.

$$\text{F1 Score} = 2*(\text{Recall} * \text{Precision}) / (\text{Recall} + \text{Precision})$$

4. Support is that number in a classifying algorithm that shows us the real existences of the given class for the given data. Unequal support for trained data indicates a weakness in the structure of reported score in the classifying algorithm which indicates that data should be rebalanced.

The following Figure 3.3 is the Classification Report of the dataset of dry bean after fitting the algorithm of XGBoost Classifier and the results seem very promising.



```
Classification Report:
               precision    recall  f1-score   support

    BARBUNYA       0.94      0.92      0.93       390
      BOMBAY       1.00      1.00      1.00       153
        CALI       0.94      0.94      0.94       479
    DERMASON       0.91      0.94      0.92      1062
       HOROZ       0.96      0.95      0.96       579
       SEKER       0.97      0.94      0.96       621
        SIRA       0.87      0.87      0.87       800

    accuracy                           0.93      4084
   macro avg       0.94      0.94      0.94      4084
weighted avg       0.93      0.93      0.93      4084
```

Figure 3.3

<u>Advantages of XGBoost Algorithm:</u>

- It requires very little Features Engineering such as we don't need to normalize the data, and also, it handles blank values very well.
- We can find how important every variable is which will be very helpful for selection of features.
- There is no or very little impact on the algorithm of XGBoost of outliers present in the data.
- It can handle very large data very well.
- It is fast as compared to other tree- based algorithms.
- It gives good accuracy in the results.
- It is safe from over- fitting of the Data- set.

<u>Dis- Advantages of XGBoost Algorithm:</u>

- It is difficult to interpret quickly and easily.
- It is very hard to find the cause of mis- classification and tuning because of the presence of a lot of hyper- parameters.
- It is not able to extrapolate the focus (or target) variable, that is, it is likely to fail in the case of repetitive (monotonous) targets.

C. RANDOM FOREST CLASSIFIER

The algorithm of Random Forest, as implied by the term, is a combination of a lot of Decision Tree models, and works as a group of those trees. Every single one from those

Decision Trees gives an output prediction of the classified in the algorithm of Random Forest Classification model and the overall output that is given by the model for every single observation is the one which has got the maximum number of same predictions by the different trees. Refer the given image.

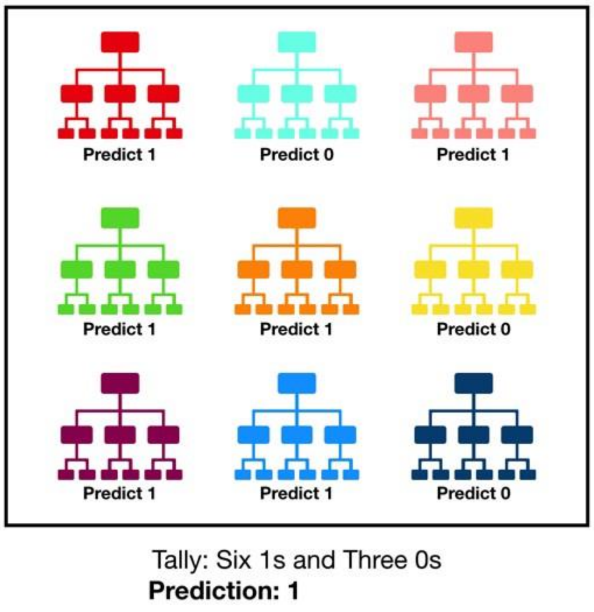


Figure 4.1

The main cause for the good and accurate predictions from this algorithm lies in how basic it is and is like voting where the one having the maximum number of votes, wins, that is, listening to the crowd. Technically speaking, the biggest cause of this algorithm giving good results is that when a combination of models with comparatively little or no correlation runs together, it is projected to beat the single basic model.

The main reason is the presence of uncorrelated model projections. This algorithm of Random Forest Classifier having no presence of correlation in each model produces collaborative prediction for every observation which is much more precise and in line with the theoretical observation than most other classification algorithms. This happens because of the protection of every single tree from each other's distinct error going in one way. Even if a few from the trees are incorrect, all of them won't be, and so we won't get the same incorrect result and it is projected that most of them will give the right result. Therefore, the basic requirements before building the Random Forest Classifier to work in a good manner are that:

1. We need a particular sign in the selected variables to ensure the building of every model with the use of that variable gives good results than the one which is built using accidental guesswork.
2. The predicted values as well as the error values of every single tree should have the presence of little or no correlation in them.

<u>Diversification in models:</u>

The way that Random Forest makes sure that there is a presence of little or no correlation in different trees is by the use of Bagging, and Feature Randomness.

I. Bagging: Random Forests make use of Bagging, also known as Bootstrap aggregation, in which every single tree has a sample of the same size in a random order from data, with repeating data, that is if there was a sample size of ‘n’, then while applying Random Forest also, the sample size will be equal to ‘n’. This is because now we are taking a sample randomly from the data of the same size. Not like decreasing the size of the data into small sample datasets to work with it. This is thought of because of the property of Decision Trees where it changes the whole structure of its tree just because of a small change while the data is trained. Let, for example, that we have a dataset of [10, 11, 12, 13, 14, 15], then Random Forest algorithm might change it to datasets [11, 13, 13, 14, 15, 15], [10, 10, 10, 12, 13, 13], [10, 11, 11, 11, 14, 15], etc. Note that all of these datasets are of length 6, and because repetitions is allowed, we’ve got various samples from one sample of data.

II. Feature Randomness: While splitting of nodes is required in a Decision Tree Classifier, each variable is considered for the possible variation; and the one containing the maximum variation is picked which is possibly producing to separate the rows to every node in the best possible manner. Whereas, if we apply the Random Forest Classifier to the data, instead of separating the observations on the basis of every single variable, it separates them only on the basis of a few features which are selected randomly because of which every tree is different and has large variance in between which helps greatly in diversifying the results for each tree as well as to reduce the covariance in every tree.

<u>Comparison: Decision Tree Classifier and Random Forest Classifier:</u>

Consider the following figure 4.2:

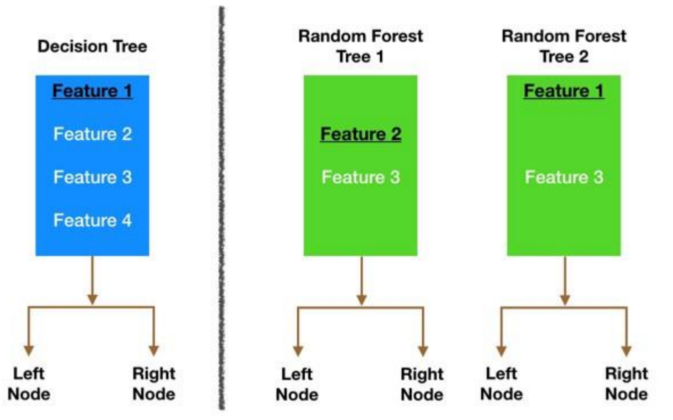


Figure 4.2

We can see that the Decision Tree model considers all of the four features from the data and selects feature 1 as the utmost important while segregating and dividing the observations into nodes, where the Random Forest model has created two different Decision Trees here and in one of the Decision Trees, Feature 1 is absent, so the algorithm selects Feature 2 as the most important factor for segregating the data into nodes, and in the other Decision Tree, Feature 2 is absent, so the algorithm selects Feature 1 as the most important factor for segregating the data into nodes. This gives us the conclusion that in the case of Random Forest Classifier, we do not have to worry about correlation between the trees, which are a result of buffering and gives protection by unwanted error terms to every tree because every tree has distinct datasets because of Bootstrap Aggregation and has distinctly selected features.

<u>Visualization!</u>

For the visualization in the case of a Random Forest classifier, since we cannot draw the whole forest, it is shown with the help of visualizing just a single Decision Tree. The function 'export_graphviz' from library 'Scikit-learn' is used to display the tree in the Jupyter notebook, as well as to export it to be saved in the device, 'write_png' is used for which 'graphviz' and 'pydotplus' are used. Figure 4.3 is the Decision Tree that is created from these commands. As we can see, we can't understand much from it, so Figure 4.4 is a tree with its depth limited to 3, which gives a much closer look of splitting of the nodes in this figure.

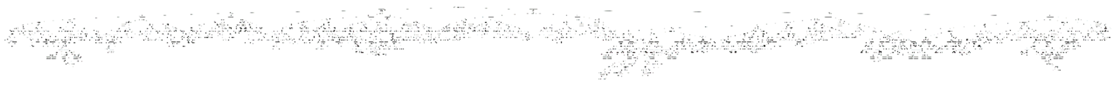


Figure 4.3

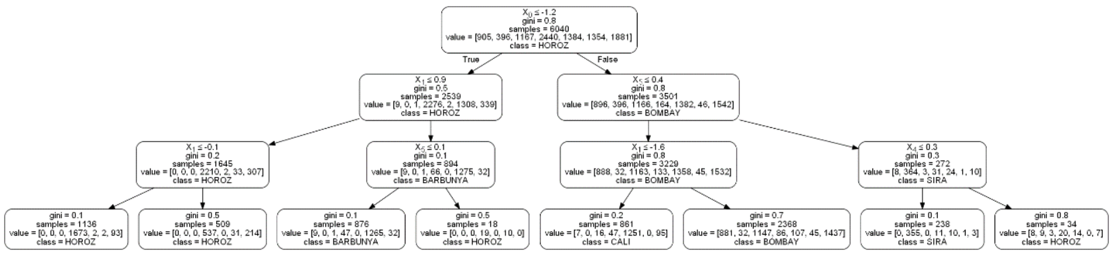


Figure 4.4

<u>**Confusion Matrix**</u>

This is a measure to check how well our algorithm has performed for a binary or multinomial classification. This is a tabulated form represented as a square matrix which comprises of various combinations of the theoretical and the casted values. It's generally used to measure Recall, Precision, and accuracy in a model.

Now, let us understand some terms that are extremely important to understand Recall and Precision.

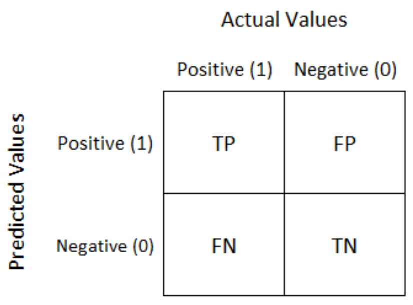


Figure 4.5

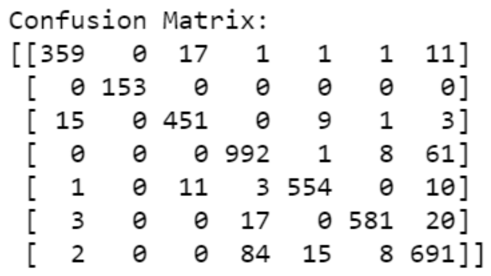
1. True Positive (TP) is when your predicted value of Positive is equal to the theoretical value of the term being positive.

2. True Negative (TN) is when your predicted value of Negative is equal to the theoretical value of the term being negative.

3. False positive (FP) is when your predicted value of Positive is not equal to the theoretical value of the term actually being negative. This is known as the Type- 1 error.

4. False Negative is when your predicted value of Negative is not equal to the theoretical value of the term actually being Positive. This is known as the Type- 2 error.

The following Figure 4.6 is the Confusion matrix of the dataset of dry bean after fitting the algorithm of Random Forest Classifier and the results seem very promising.



```
Confusion Matrix:
[[359   0  17   1   1   1  11]
 [  0 153   0   0   0   0   0]
 [ 15   0 451   0   9   1   3]
 [  0   0   0 992   1   8  61]
 [  1   0  11   3 554   0  10]
 [  3   0   0  17   0 581  20]
 [  2   0   0  84  15   8 691]]
```

Figure 4.6

<u>Classification Report</u>

This shows us the values of Precision, Recall, F1- Score, and Support in the fitted model.

1. Precision is that quality in a classifying algorithm that says no to predict something as positive when it is negative. It is calculated by the ratio of True Positives by the sum of True Positives and False Positives.

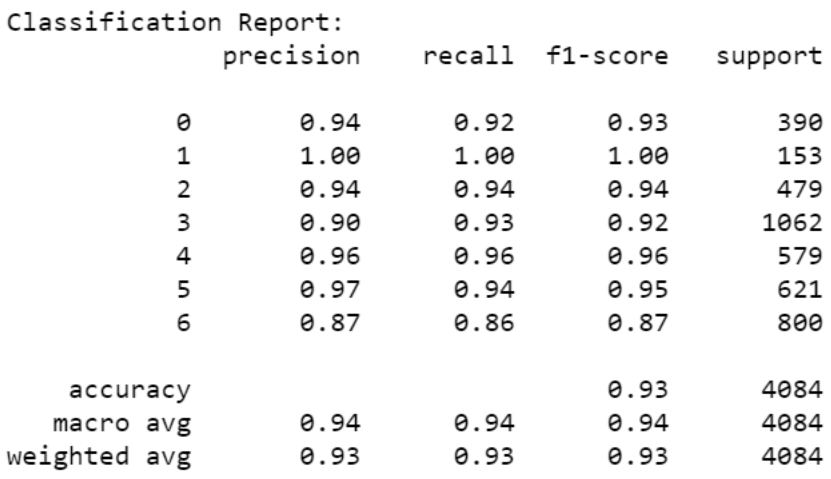
2. Recall is that quality in a classifying algorithm that tells us the quantity of Positives we got right. It is calculated by the ratio of True Positives by the sum of True Positives and False Negatives.

3. F1- Score is that quality in a classifying algorithm that tells us the quantity of Positives that were correct.

$$\text{F1 Score} = 2*(\text{Recall} * \text{Precision}) / (\text{Recall} + \text{Precision})$$

4. Support is that number in a classifying algorithm that shows us the real existences of the given class for the given data. Unequal support for trained data indicates a weakness in the structure of reported score in the classifying algorithm which indicates that data should be rebalanced.

The following Figure 4.7 is the Classification Report of the dataset of dry bean after fitting the algorithm of Random Forest Classifier and the results seem very promising.


```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.92      0.93       390
           1       1.00      1.00      1.00       153
           2       0.94      0.94      0.94       479
           3       0.90      0.93      0.92      1062
           4       0.96      0.96      0.96       579
           5       0.97      0.94      0.95       621
           6       0.87      0.86      0.87       800

    accuracy                           0.93      4084
   macro avg       0.94      0.94      0.94      4084
weighted avg       0.93      0.93      0.93      4084
```

Figure 4.7

<u>Advantages of Random Forest Classifying Algorithm:</u>

- It helps in decreasing the correlation coefficient to a lot of extent. Because of its algorithm having the methods of Bagging and Feature Randomness, it proves to be better than Decision Tree Classifier.
- It reduces the error and gives much more accurate results as it consists of a Decision Trees, and the one Class that has the greatest number of votes wins.
- It gives great performance even when the data is not balanced.
- It can handle large data having very high dimensions in each variable.
- It can even handle the data containing missing values well.
- Outliers in the data doesn’t cause much harm to the prediction of this Classifier.
- It generalises more and overfits the data less as for each tree only a few features are considered.

<u>Dis- advantages of Random Forest Classifying Algorithm:</u>

- Every feature should have the ability to predict the final class of the dependent variable.
- Predicted values from each individual tree should not be correlated.
- It is difficult to understand what is happening inside of the Classifier and thus it is difficult to understand the cause of misclassification in the case of low accuracy.

D. SUPPORT VECTOR MACHINES

Support Vector Machines (SVM) is a widely used algorithm for both regression and classification purposes since it is used to produce accurate results with good statistical significance and it takes only a little computational power. For our Dataset of Dry Beans, we'll use this great algorithm for classification purposes.

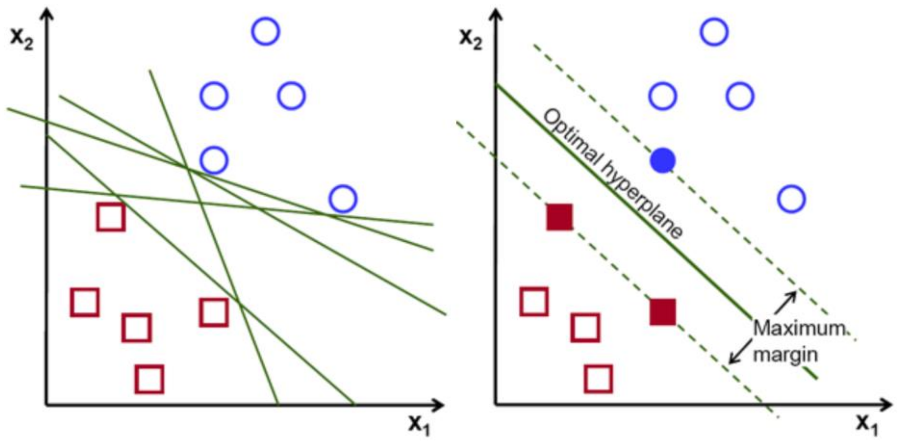


Figure 5.1

The primary task of Support Vector Machines (SVMs) is to detect an optimal hyperplane such that the points in the plane of dimension equal to N, where N is the number of distinct classes, are separated with the maximum partition between them according to the training dataset using the observations. For the separation of each point in the dataset, a lot of such hyperplanes could be selected, but the main task of this algorithm is to detect a hyperplane that separates the points with the maximum distance (margin) between them for every different class. If this distance is maximised, that will prove to give more confidence to the algorithm to run and predict the future results of test cases with good accuracy. In Figure 5.1, on the left, if we select any of the hyperplane, it can be seen that it won't be an optimal solution of this problem, but on the right side, the hyperplane that is selected and highlighted with green dotted lines can be seen to separate the two classes of the example problem really well and is the chosen hyperplane of the Support Vector Machine Classifier.

Hyperplanes:

A boundary used to make decisions for classification of the dataset in the Support Vector Machines algorithm is known as a Hyperplane. Hyperplanes separates the classified data into different classes as the data present on every side of this plane belongs to a distinct class. The dimensions in a hyperplane are totally dependent on the feature count. In the case of feature count equal to two, the dimension of the hyperplane is equal to one, that is, it's a line; in the case of feature count equal to three, the dimension of the hyperplane is equal to two, that is, it's a 2- D plane; and as we increase the number of features, the dimension of hyperplane also keeps on increasing, therefore, it is very difficult to be able to visualize it for out data set of Dry Beans. The hyperplane in the case of 2- D and 3- D features, that is hyperplane of a line and a 2- D plane respectively is shown in Figure 5.2.

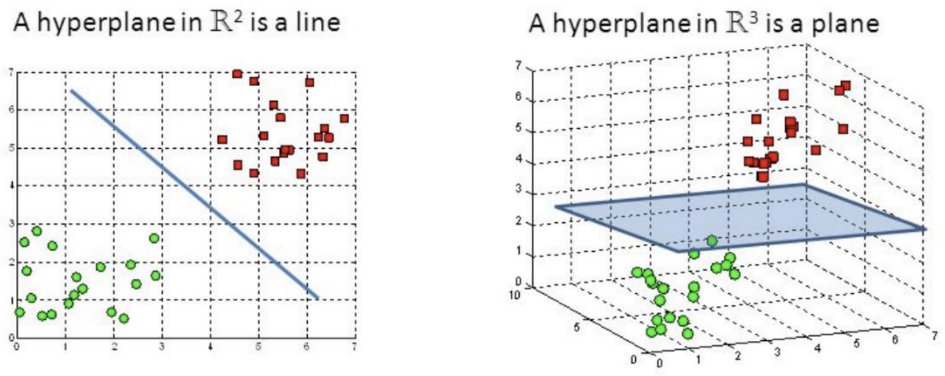


Figure 5.2

<u>Support Vector:</u>

These are the points from the data set which belong to different classes but are closest to each other and thus matter the most when the algorithm tries to create a hyperplane to separate the classes, so in turn, they are closest to the hyperplane that separates the classes. These Support vectors are crucial for the building of the hyperplane and maximization of the distance between distinct classes. If we try to delete even one of the support vectors, the hyperplane that separates the classes may change drastically.

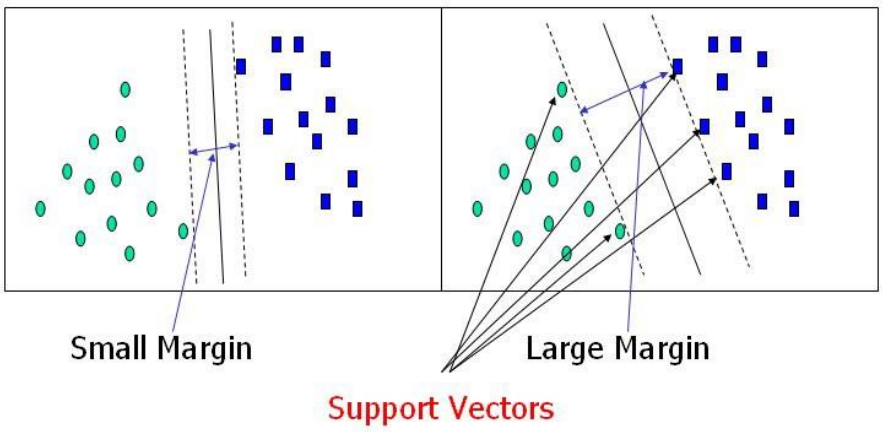


Figure 5.3

Margin Insight:

The way Logistic Regression works is that the result of the given function is taken and then processed in sigmoid such that every one of its value becomes something (a rational number) that is greater than or equal to 0 and less than or equal to 1. And then we set a threshold (the default value of this threshold is 0.5) and then process the results such that if the resultant value is less than this threshold value, it is assigned the value 0 and if the resultant value is more than this threshold value, it is assigned the value 1. But here, in the case of Support Vector Machines, the result of the given function is taken and if that value is more than or equal to 1, it is assigned a class, and if that value is less than or equal to -1, it is assigned another of the classes; and these are taken as margins.

Function of cost and update in the Gradient:

For successful application of the Support Vector Machines Classification algorithm, we want to find the hyperplane such that there is maximum distance between the hyperplanes and the different points in the data set. Hinge Loss function (Figure 5.4) is used as the function of the loss which is used to find a hyperplane such that there is maximum distance between the hyperplanes and the different points in the data set.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

Figure 5.4

When the theoretical value matches the classified value, this cost function gives a value equal to 0; and when the theoretical value does not match the classified value, this value of cost function changes and takes a value according to the regularised value such that it balances the overall loss and marginal maximum values. Figure 5.5 shows the function of total loss after partial differentiation w. r. t. the weight and gradient are found using that. With the help of these gradient functions, every one of the weights is updated.

$$\frac{\delta}{\delta w_k}\lambda \parallel w \parallel^2 \quad = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k}\big(1 - y_i\langle x_i, w\rangle\big)_+ \quad = \begin{cases} 0, & \text{if } y_i\langle x_i, w\rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

Figure 5.5

If every single one of the observations is correctly predicted by the algorithm, only the gradient is updated using the regulation factor (Figure 5.6).

$$w = w - \alpha \cdot (2\lambda w)$$

Figure 5.6

If for every observation, some of the observations are not predicted correctly by the algorithm, the loss function is included and the regulation factor is also included for the performance of the updating of gradient (Figure 5.7).

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

Figure 5.7

<u>Non- linear data:</u>

If the data that we need classifying is not linearly separable, then we need to introduce something else that can separate our non- linear data. For example, consider Figure 5.8, we can see in this figure that we can’t separate the red triangles and blue circular data points with the help of a single straight line.

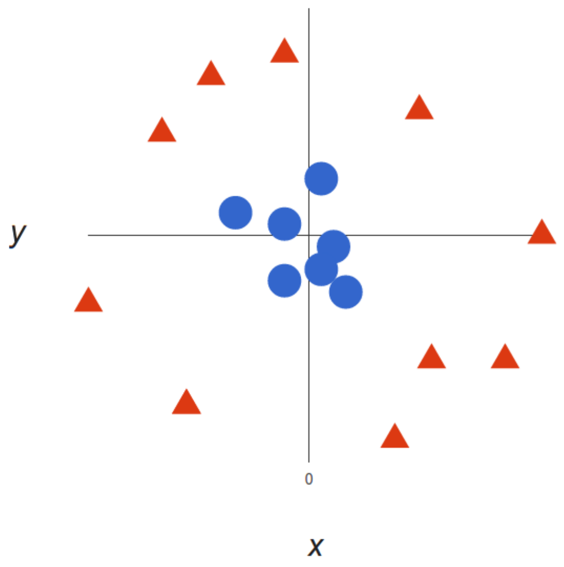


Figure 5.8

Still, we can see in this example that the data points are making clusters and it should not be very difficult to separate them as if we somehow create a circle and close the blue circular data points inside it, then they can be separated from the red triangular data points. So, a new dimension, the Z- dimension will be created that will be equal to the formula of the circle, that is, $Z = X^2 + Y^2$. And the Z- dimension looks somewhat like Figure 5.9.

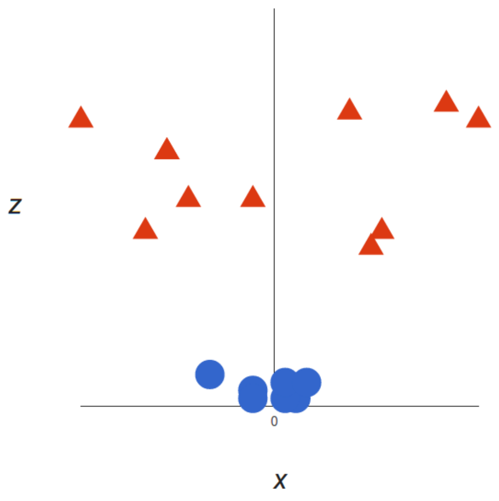


Figure 5.9

No, it is seen that we can separate these data points using a line which will be a hyperplane that is parallel to the X- axis and cuts the Z- axis at some point (maybe at Z

= 3?), that we defined to separate the points for points which were not linearly separable. If we try to map it back to a 2- Dimensional plane, it would look like the following Figure 5.10.

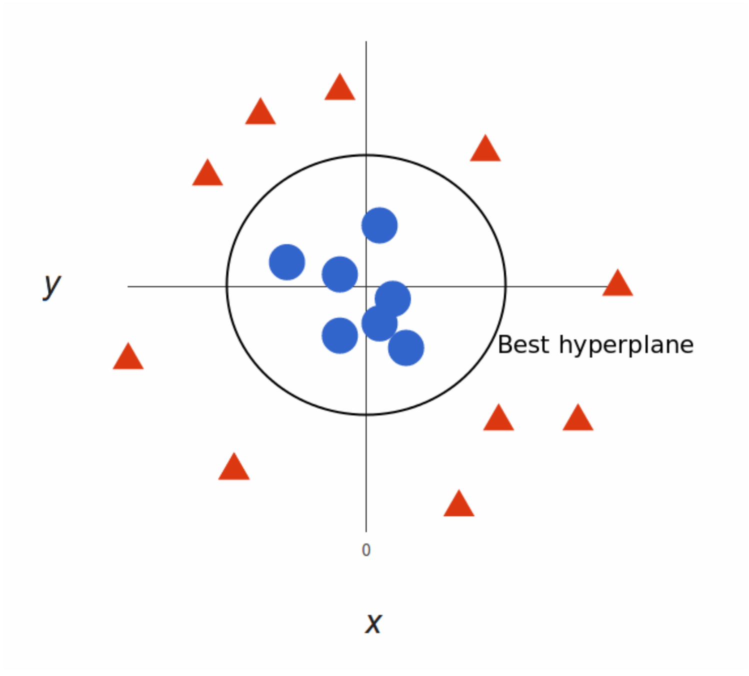


Figure 5.10

Isn't this absolutely amazing? Well, I think it is! This shown circle in the above Figure 5.10 is the best hyperplane that is able to separate the blue circles and red triangles very well.

Major Applications of Support Vector Classifier:

The algorithm of Support Vector Classifier solves real- world problematic situations such as:

- We can classify different images with the help of Support Vector Machine Classifier as they can help us differentiate various images with a lot of accuracy.
- We can use Support Vector Machines to classify data available from satellites such as SAR.
- We can use this to understand and differentiate hand- written text data by giving supervised training set.
- The algorithm of Support Vector Machines is also applied hugely in the field of biology and Science. This can be used in the classification of proteins and other

minerals, etc. It has proved to classify this type of data with good accuracy and results in the past.

Confusion Matrix

This is a measure to check how well our algorithm has performed for a binary or multinomial classification. This is a tabulated form represented as a square matrix which comprises of various combinations of the theoretical and the casted values. It's generally used to measure Recall, Precision, and accuracy in a model.

Now, let us understand some terms that are extremely important to understand Recall and Precision.

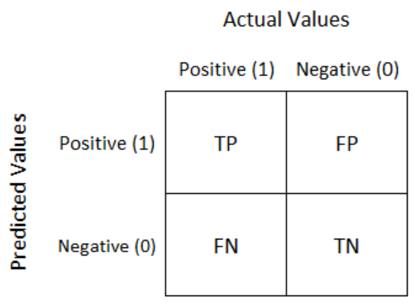


Figure 5.11

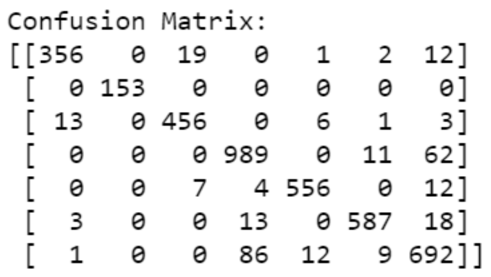
1. True Positive (TP) is when your predicted value of Positive is equal to the theoretical value of the term being positive.

2. True Negative (TN) is when your predicted value of Negative is equal to the theoretical value of the term being negative.

3. False positive (FP) is when your predicted value of Positive is not equal to the theoretical value of the term actually being negative. This is known as the Type- 1 error.

4. False Negative is when your predicted value of Negative is not equal to the theoretical value of the term actually being Positive. This is known as the Type- 2 error.

The following Figure 5.12 is the Confusion matrix of the dataset of dry bean after fitting the algorithm of Support Vector Machine Classifier and the results seem very promising.



```
Confusion Matrix:
[[356   0  19   0   1   2  12]
 [  0 153   0   0   0   0   0]
 [ 13   0 456   0   6   1   3]
 [  0   0   0 989   0  11  62]
 [  0   0   7   4 556   0  12]
 [  3   0   0  13   0 587  18]
 [  1   0   0  86  12   9 692]]
```

Figure 5.12

Classification Report

This shows us the values of Precision, Recall, F1- Score, and Support in the fitted model.

1. Precision is that quality in a classifying algorithm that tells not to predict something as positive when it is negative. It is calculated by the ratio of True Positives by the sum of True Positives and False Positives.

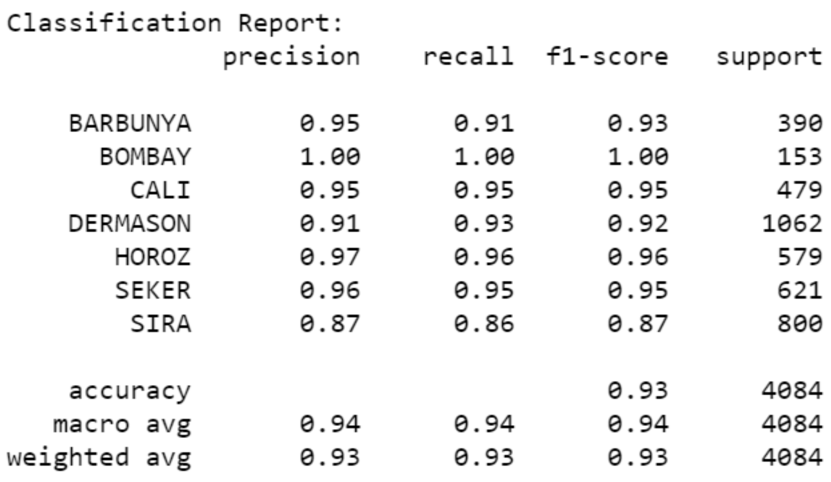
2. Recall is that quality in a classifying algorithm that tells us the quantity of Positives we got right. It is calculated by the ratio of True Positives by the sum of True Positives and False Negatives.

3. F1- Score is that quality in a classifying algorithm that tells us the quantity of Positives that were correct.

$$\text{F1 Score} = 2*(\text{Recall} * \text{Precision}) / (\text{Recall} + \text{Precision})$$

4. Support is that number in a classifying algorithm that shows us the real existences of the given class for the given data. Unequal support for trained data indicates a weakness in the structure of reported score in the classifying algorithm which indicates that data should be rebalanced.

The following Figure 5.13 is the Classification Report of the dataset of dry bean after fitting the algorithm of SVM Classifier and the results seem very promising.



```
Classification Report:
               precision    recall  f1-score   support

    BARBUNYA       0.95      0.91      0.93       390
      BOMBAY       1.00      1.00      1.00       153
        CALI       0.95      0.95      0.95       479
    DERMASON       0.91      0.93      0.92      1062
       HOROZ       0.97      0.96      0.96       579
       SEKER       0.96      0.95      0.95       621
        SIRA       0.87      0.86      0.87       800

    accuracy                           0.93      4084
   macro avg       0.94      0.94      0.94      4084
weighted avg       0.93      0.93      0.93      4084
```

Figure 5.13

Advantages of Support Vector Machine Classifying Algorithm:

- It helps in classifying data and gives good and accurate results even for data of higher dimensions.
- It works well even in the case when dataset has more dimensions than that of the observations, that is when observations are less.
- Since it works by using the subset of training examples, known as a Support Vector, hence it saves the memory and uses it efficiently.
- This algorithm is very versatile as we can give it various Kernels as a function for its decision function. Customised kernels can be specified.
- This works well when the different class can be separated easily.
- Does not perform very bad even in the presence of outliers.

Dis- advantages of Support Vector Machine Classifying Algorithm:

- For the case when there are a lot of features and the number of observations is less, we need to specifically choose the function of Kernel and the variable that regularises the data in order to prevent from data over- fitting.
- This is non- probabilistic algorithm, so we do not get the estimations for probabilities.
- It is a little slower when the data- set is large as compared to some other algorithms.
- It works badly sometimes when different class overlaps each other.

E. k- NEAREST NEIGHBORS

The algorithm of k- Nearest Neighbors, or k-NN, can be used both for classification and regression, and works on the principle of grouping of observations from the train set of the data- set, which is in turn used to assign the observations from the test data-set to the groups made from train data- set. When we try to classify the data, the resultant variable is categorical, that is, it assigns each and every observation to a category. In the algorithm of k- nearest neighbors, first the data is trained using training data, and while testing the data on the test set, the observations are given classes according to the most commonly given classes when compared with the k (a number, previously specified) nearest neighboring observations to them. For example, when k = 1, then the observations are given classes according to the class of one sole class of neighboring observations.

The k- Nearest Neighbors algorithm works on the principle of approximation and its function approximates local values which in turn computes further values which can't be evaluated until the function has completed with its evaluating to be done. And as distance is a very important measure for this algorithm to work properly and give the assigned values and accordingly approximating them, if there are variables present in the data- set which are very different from each other or represent vastly different quantities, like length, breadth of some quantity along with the volume, then standardised scaling the data becomes very important which will bring all the variables to have a mean of 0 and variance of 1, or normalizing the data will also work in that case. It is important not to just let small and large quantities together for different observations. This will also help in increasing the accuracy and results of the classification by a huge margin.

A method that has proved to help in increasing the accuracy of the k- nearest neighbors classification and regression algorithm is to give more weightage to the contribution of every neighbour which are more near to the test observation than the ones who are farther away from them, to the overall average.

Choosing the value of K:

Firstly, we should know how, why and how much the value of 'k' affects the results when we apply this algorithm. Consider the following example in Figure 6.1:

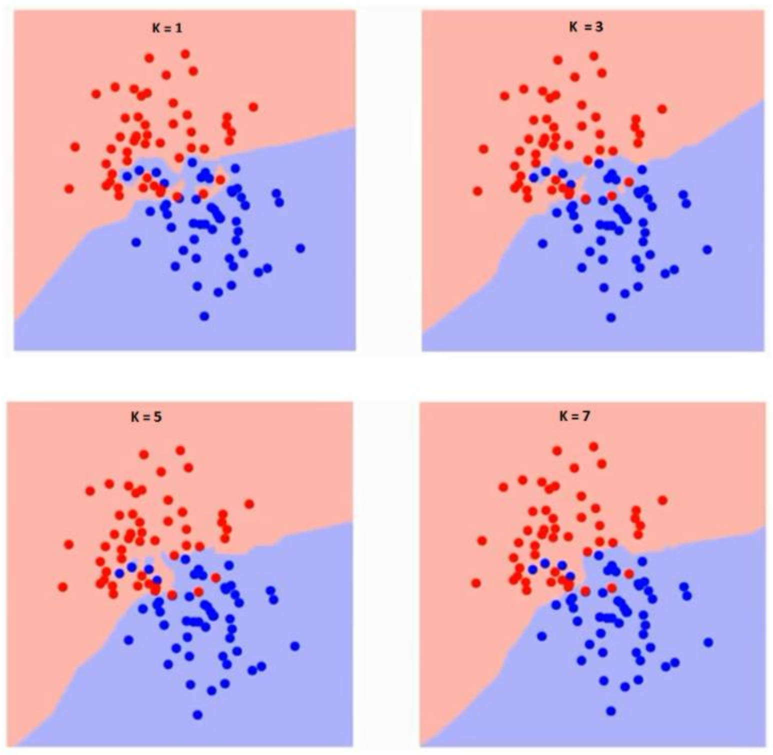


Figure 6.1

If we look closely, it can be easily seen that as we increase K, the border line between the red dots and the blue dots, which represents different classes here, becomes softer and smoother. If we put K equal to infinite value, the whole plane will either become red or it will become blue which depends on what the major class here is. There are 2 factors: the training error rate, and the validation error rate, whose values changes with this change in the value of 'k', and which are very important for the determination of the optimum value of 'k' for every training data- set.

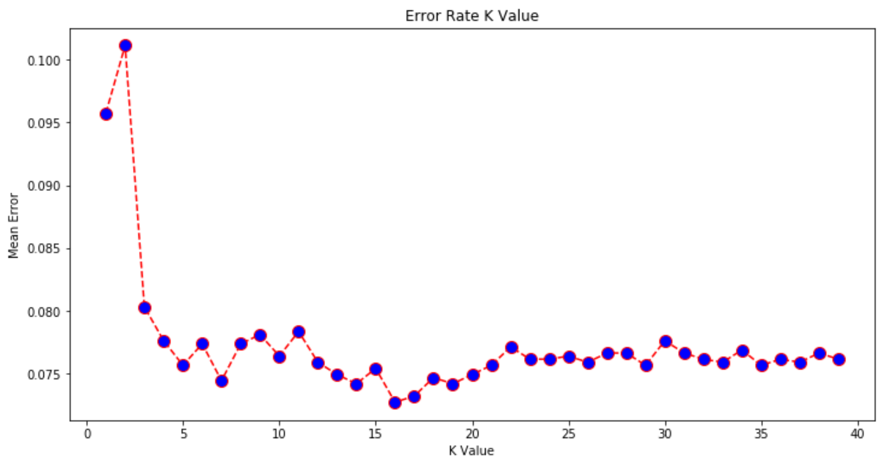


Figure 6.2

If we fit the training data- set and then predict the class for the test data- set, then the mean error rate between the theoretical values and the predicted values for different values of ‘k’ until 40 is plotted in Figure 6.2. As we can see, it increases and decreases for the initial values, and with the help of this figure, I’ve chosen k = 7 to carry out the analysis as this value seems perfect for this data- set and the next value which gives mean error rate less than this value is 16.

This method of classification is a little different from some other algorithms that we’ve learnt about till now in the sense that this doesn’t learn the model per se, but this takes all of the training data every time we try to make a classification and use it to predict the class for very observation. So, this will make the training part of the whole method very fast as it is not learning anything from it, just taking it and dumping it to save to disk, just that it takes up a lot of space and memory, and the time it takes to predict the class for the test data- set is more since every time it tries to make a prediction, the algorithm has to run through the whole data- set. Overall, this is a good method for regression and classification for small data- sets and for learning purposes.

<u>Over- fitting and Under- fitting:</u>

Over- fitting in a data- set arises in the case of train data- set learning the particular detail and the noises from every training example of the data- set, which gives less training error but increases the test data- set error a lot. Consider Figure 6.3 to better understand how a model can over- fir a training data and learn from its noise:

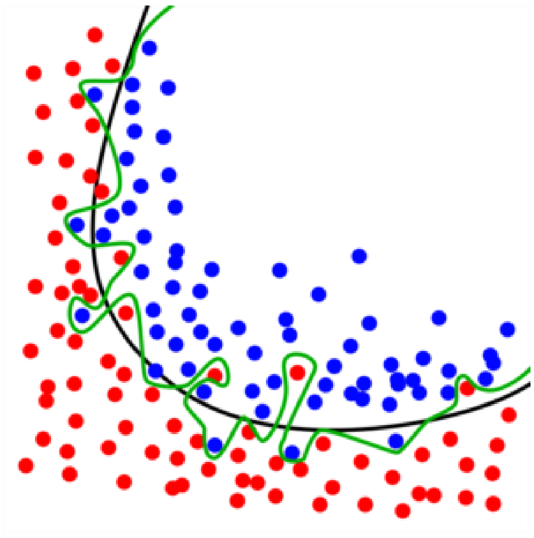


Figure 6.3

In the Figure 6.3, we can see that if we follow the green line, it is dividing the two classes of red dots and blue dots very nicely, but if we try to use this setup for new data, it will fail miserably as it is learning all the noise from the training data. Now, if we try to follow the black line that divides the red dots and the blue dots, it is not dividing them like the green line does, but when we use it to classify for new test data- set, it will work just as good, if not better, on the test data- set. This is a regular version of the model. Under- fitting, on the other hand, gives high error rate of prediction for both training and testing data. For example, if we try to fit a multi- dimensional model on a line, that would be an under- fit and perform very poorly. For the best results, we have to find a model that fits the data in a regular way, neither over- fitting it, nor under- fitting it and that works well on both the train and test data- sets.

<u>Distance metrics:</u>

A lot of different ways can be used to find the nearest class by finding the distance. There are various methods, such as Euclidean distance, Manhattan distance, Minkowski distance, Hamming distance, etc to find the distance between points and selecting the best one depends on the data- set as one may work of one data- set and another for another data- set, since no one size fits all.

I. Euclidean distance: This is calculated by the squared root of the addition of the squared distance between two data points. It is shown in Figure 6.4, which will make its understanding a little clear.

$$\boxed{\text{Euclidean Distance between } (x_1, y_1) \text{ and} (x_2, y_2) = \quad \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}$$

Figure 6.4

II. Manhattan distance: This is calculated by summing the absolute values of the difference between two data points. The formula is shown in Figure 6.5, which makes it a little simpler to understand.

$$\boxed{\text{Manhattan Distance between } (x_1, y_1) \text{ and} (x_2, y_2) = \quad |x_1 - x_2| + |y_1 - y_2|}$$

Figure 6.5

III. Minkowski distance: This is calculated by using the formula given in Figure 6.6, and it can be seen that this depends on the value of p, as when p = 1, this is equal to Manhattan distance and when p = 2, this is equal to Euclidean distance.

Minkowski Distance between (x_1, y_1) and(x_2, y_2) = $(|x_1 - x_2|^p + |y_1 - y_2|^p)^{1/p}$

p is any real value. Usually set to a value between 1 and 2

Figure 6.6

IV. Hamming distance: This is used to check the likeness in variables of category. If the two categories are same, the value is equal to 0 and if they are different, then the value is equal to 1. An example is given in Figure 6.7 to better understand this distance measure.

Hemming distance between categorical variables Large and Medium = 1

Hemming distance between categorical variables Large and Large = 0

Figure 6.7

<u>Major Applications and examples of k- nearest neighbors Classifier:</u>

The algorithm of k- nearest neighbors Classifier solves real- world problematic situations such as:

- We can give credits rating to people whose information database is available such as their finance options and compare it with others with almost the same options to give credits rating.
- This can be used by a Bank to determine if they want to give a loan to a person or not, and make a list which are good prospects to give loans to, who have had good history and transaction records in the past. Banks can check if two people are similar to each other who have or have not paid on their loan in the past.
- In politics, parties can check the history of voting candidates on the basis of several factors such as if they've ever voted in the past, are they more likely to cast a vote this time or not, etc.
- Other applications include hand- writing recognition, image and video analysis, etc.

<u>Confusion Matrix</u>

This is a measure to check how well our algorithm has performed for a binary or multinomial classification. This is a tabulated form represented as a square matrix which comprises of various combinations of the theoretical and the casted values. It's generally used to measure Recall, Precision, and accuracy in a model.

Now, let us understand some terms that are extremely important to understand Recall and Precision.

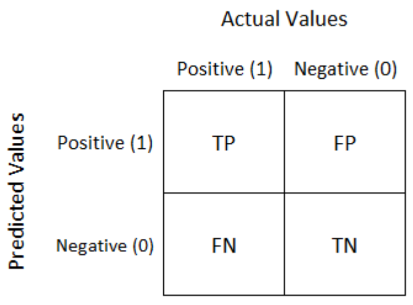


Figure 6.8

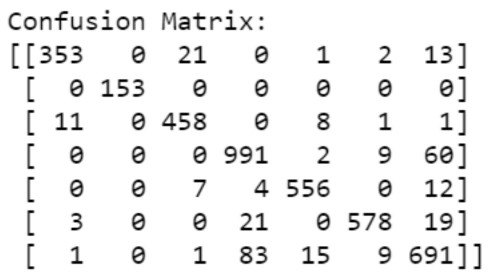
1. True Positive (TP) is when your predicted value of Positive is equal to the theoretical value of the term being positive.

2. True Negative (TN) is when your predicted value of Negative is equal to the theoretical value of the term being negative.

3. False positive (FP) is when your predicted value of Positive is not equal to the theoretical value of the term actually being negative. This is known as the Type- 1 error.

4. False Negative is when your predicted value of Negative is not equal to the theoretical value of the term actually being Positive. This is known as the Type- 2 error.

The following Figure 6.9 is the Confusion matrix of the dataset of dry bean after fitting the algorithm of k- nearest neighbors Classifier and the results seem very promising.



```
Confusion Matrix:
[[353   0  21   0   1   2  13]
 [  0 153   0   0   0   0   0]
 [ 11   0 458   0   8   1   1]
 [  0   0   0 991   2   9  60]
 [  0   0   7   4 556   0  12]
 [  3   0   0  21   0 578  19]
 [  1   0   1  83  15   9 691]]
```

Figure 6.9

<u>Classification Report</u>

This shows us the values of Precision, Recall, F1- Score, and Support in the fitted model.

1. Precision is that quality in a classifying algorithm that tells not to predict something as positive when it is negative. It is calculated by the ratio of True Positives by the sum of True Positives and False Positives.

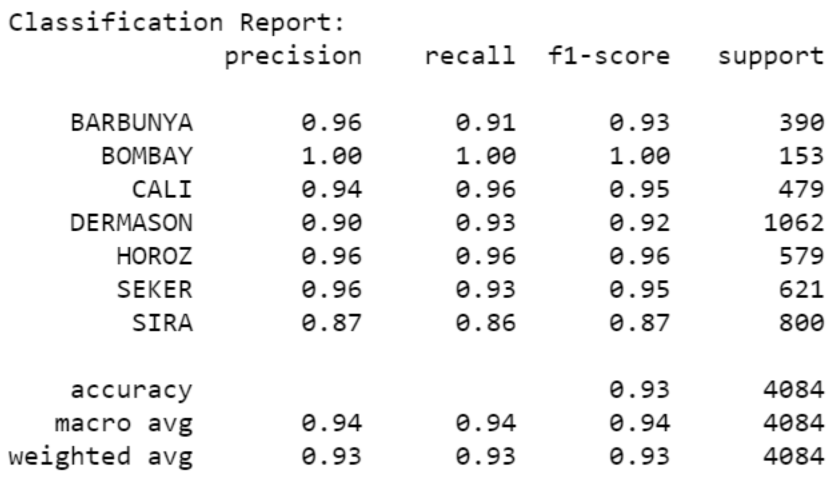
2. Recall is that quality in a classifying algorithm that tells us the quantity of Positives we got right. It is calculated by the ratio of True Positives by the sum of True Positives and False Negatives.

3. F1- Score is that quality in a classifying algorithm that tells us the quantity of Positives that were correct.

$$\text{F1 Score} = 2*(\text{Recall} * \text{Precision}) / (\text{Recall} + \text{Precision})$$

4. Support is that number in a classifying algorithm that shows us the real existences of the given class for the given data. Unequal support for trained data indicates a weakness in the structure of reported score in the classifying algorithm which indicates that data should be rebalanced.

The following Figure 6.10 is the Classification Report of the dataset of dry bean after fitting the algorithm of k- nearest neighbors Classifier and the results seem very promising.



```
Classification Report:
              precision    recall  f1-score   support

    BARBUNYA       0.96      0.91      0.93       390
      BOMBAY       1.00      1.00      1.00       153
        CALI       0.94      0.96      0.95       479
    DERMASON       0.90      0.93      0.92      1062
       HOROZ       0.96      0.96      0.96       579
       SEKER       0.96      0.93      0.95       621
        SIRA       0.87      0.86      0.87       800

    accuracy                           0.93      4084
   macro avg       0.94      0.94      0.94      4084
weighted avg       0.93      0.93      0.93      4084
```

Figure 6.10

Advantages of k- nearest neighbors Classifying Algorithm:

- This is a very simple algorithm which is easily understandable.
- We do not need to make a lot of assumptions here ourselves.

- It is a multipurpose algorithm as it can be regress, classify and search for data.
- It evolves itself according to the needs and when new data is added.
- It can be used for multi- class classification.
- We only need to select the value of ‘k’ once for a data- set and then almost everything is done.
- It takes almost no time to train the data- set.

Dis- advantages of k- nearest neighbors Classifying Algorithm:

- It is slow for very big data- sets as it takes all of the data- set and then computes the class for every test observation.
- It works somewhat badly for data- sets having huge number of variables as compared to other algorithms.
- We should always scale the data before applying this algorithm.
- It works badly when there are a lot of outliers present in the data- set.

  It does not work well when the data- set has a lot of missing points.

DISCUSSION AND CONCLUSIONS

A lot of different classification algorithms were applied on the Dry Bean data set to project the dependent variable (class) using Python Programming Language. For this purpose, after applying Principal Component Analysis on all of the 16 variables, and setting the output accuracy as 99%, we get seven features, which are a combination of all the variables, as is the case with the algorithm of Principal Component Analysis, it takes all the variables and gives an output such that it contains the important contributing factors from all those variables, but if we try and find the most important features that have contributed the most to those seven features are MajorAxisLength, MinorAxisLength, Solidity, Extent, roundness, ShapeFactor1, and Eccentricity.

After the application of Decision Tree Classifying algorithm on the Pre-Processed data- set, the accuracy was found to be 89.45%, the algorithm of XGBoost Classifier gave an accuracy of 92.899%, Random Forest Classifier gave an accuracy of 92.58%, Support Vector Machines Classifier gave an accuracy of 92.78%, and finally the accuracy in the case of k- Nearest Neighbors was found to be 92.56%. So, we conclude that all of these algorithms are good and can be used to classify categorical data and all of them gave almost the same accuracy, it just depends on data to know which one will perform the best on that. Here, on this data- set of Dry Beans, XGBoost Classifier gave the highest accuracy of 92.899%.

REFERENCES

- Data downloaded from UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset
- Michael Galarnyk, “PCA using Python (scikit-learn)”, Towards Data Science (2017)
- Devesh Poojari, “Machine Learning Basics: Decision Tree From Scratch”, Towards Data Science (2019)
- Tony Yiu, “Understanding Random Forest: How the Algorithm Works and Why it Is So Effective”, Towards Data Science (2019)
- Will Koehrsen, “An Implementation and Explanation of the Random Forest in Python”, Towards Data Science (2018)
- George Seif, “A Beginner’s guide to XGBoost”, Towards Data Science (2019)
- Jason Brownlee, “How to Develop Your First XGBoost Model in Python”, Machine Learning Mastery (2016)
- Sunil Ray, “Understanding Support Vector Machine(SVM) algorithm from examples (along with code)”, Analytics Vidhya (2017)
- Rohith Gandhi, “Support Vector Machine — Introduction to Machine Learning Algorithms”, Towards Data Science (2018)
- Czako Zoltan, “KNN in Python”, Towards Data Science (2018)
- Kevin Zakka, “A Complete Guide to K-Nearest-Neighbors with Applications in Python and R”, Kevin Zakka's Blog (2016)