2020

Wine Data Analysis - A supervised approach

Name - RASIKA BOSE

Roll No - 446

Supervisor's Name – PALLABI GHOSH

I have affirmed that I have identified all my sources and that no part of my dissertation paper uses unacknowledged materials.

Signed : Rasika Bose

TABLE OF CONTENTS

INTRODUCTION AND OBJECTIVE

The purpose of my project is to analyse the wine dataset which is drawn from an R package named rattle.data. This wine dataset contains the results of a chemical analysis of wines grown in a specific area of Italy . Three types of wine are represented in the 178 samples, with the results of 13 chemical analyses(Alcohol, Malic, Ash, Alcalinity, Magnesium, Phenols, Flavanoids, Non Flavanoids, Proanthocyanins, Color, Hue, Dilution, Proline) recorded for each sample. The Type variable is transformed into a categorical variable. The data contains no missing values and consists of only numeric data, with a three class target variable (Type) for classification.

The data provides us three types of wine data together with their thirteen chemical properties. Using these 13 chemical properties as predictors I will be building a three class classification model so that given the chemical properties one can accurately predict the type of the wine.

Objectives are:

- To go beyond linear classifiers and study some non parametric classifiers and certain tree and ensemble based learnings .
- To apply these methods on classification based problems where obviously linear classifiers would fail. Hence exploring the power of the above mentioned classifiers in these situations.

DATA OVERVIEW:

The first 6 set of observations for each of the 3 types of wine including the observations of 13 chemical analyses(Alcohol, Malic, Ash, Alcalinity, Magnesium, Phenols, Flavanoids, Non Flavanoids, Proanthocyanins, Color, Hue, Dilution, Proline) is given as follows :

	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids
1	1	14.23	1.71	2.43	15.6	127	2.80	3.06
2	1	13.20	1.78	2.14	11.2	100	2.65	2.76
3	1	13.16	2.36	2.67	18.6	101	2.80	3.24
4	1	14.37	1.95	2.50	16.8	113	3.85	3.49
5	1	13.24	2.59	2.87	21.0	118	2.80	2.69
6	1	14.20	1.76	2.45	15.2	112	3.27	3.39
60	2	12.37	0.94	1.36	10.6	88	1.98	0.57
61	2	12.33	1.10	2.28	16.0	101	2.05	1.09
62	2	12.64	1.36	2.02	16.8	100	2.02	1.41
63	2	13.67	1.25	1.92	18.0	94	2.10	1.79
64	2	12.37	1.13	2.16	19.0	87	3.50	3.10
65	2	12.17	1.45	2.53	19.0	104	1.89	1.75
131	3	12.86	1.35	2.32	18.0	122	1.51	1.25
132	3	12.88	2.99	2.40	20.0	104	1.30	1.22
133	3	12.81	2.31	2.40	24.0	98	1.15	1.09
134	3	12.70	3.55	2.36	21.5	106	1.70	1.20
135	3	12.51	1.24	2.25	17.5	85	2.00	0.58
136	3	12.60	2.46	2.20	18.5	94	1.62	0.66

	Nonflavanoids	Proanthocyanins	Color	Hue	Dilution	Proline
1	0.28	2.29	5.64	1.04	3.92	1065
2	0.26	1.28	4.38	1.05	3.40	1050
3	0.30	2.81	5.68	1.03	3.17	1185
4	0.24	2.18	7.80	0.86	3.45	1480
5	0.39	1.82	4.32	1.04	2.93	735
6	0.34	1.97	6.75	1.05	2.85	1450
60	0.28	0.42	1.95	1.05	1.82	520
61	0.63	0.41	3.27	1.25	1.67	680
62	0.53	0.62	5.75	0.98	1.59	450
63	0.32	0.73	3.80	1.23	2.46	630
64	0.19	1.87	4.45	1.22	2.87	420
65	0.45	1.03	2.95	1.45	2.23	355
131	0.21	0.94	4.10	0.76	1.29	630
132	0.24	0.83	5.40	0.74	1.42	530
133	0.27	0.83	5.70	0.66	1.36	560
134	0.17	0.84	5.00	0.78	1.29	600
135	0.60	1.25	5.45	0.75	1.51	650
136	0.63	0.94	7.10	0.73	1.58	695

```
> |
```

EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) is an approach/philosophy for data analysis that employs a variety of techniques (mostly graphical) to

- maximize insight into a data set
- uncover underlying structure
- extract important variables
- detect outliers and anomalies
- test underlying assumptions
- develop parsimonious models
- determine optimal factor setting

About: The EDA approach is precisely not a set of techniques, but an attitude/philosophy about how a data analysis should be carried out. EDA is not identical to statistical graphics although the two terms are used almost interchangeably. Statistical graphics is a collection of techniques--all graphically based and all focusing on one data characterization aspect. EDA encompasses a larger venue. EDA is an approach to data analysis that postpones the usual assumptions about what kind of model the data follow with the more direct approach of allowing the data itself to reveal its underlying structure and model. EDA is not a mere collection of techniques; EDA is a philosophy as to how we dissect a data set; what we look for; how we look; and how we interpret. It is true that EDA heavily uses the collection of techniques that we call "statistical graphics", but it is not identical to statistical graphics per se.

For Exploratory Data Analysis of the “wine” dataset we perform the following data visualization methods:

- Density plots
- Principal Component Analysis(PCA) plots
- Correlation Plots
- Testing for equality of the correlation matrices

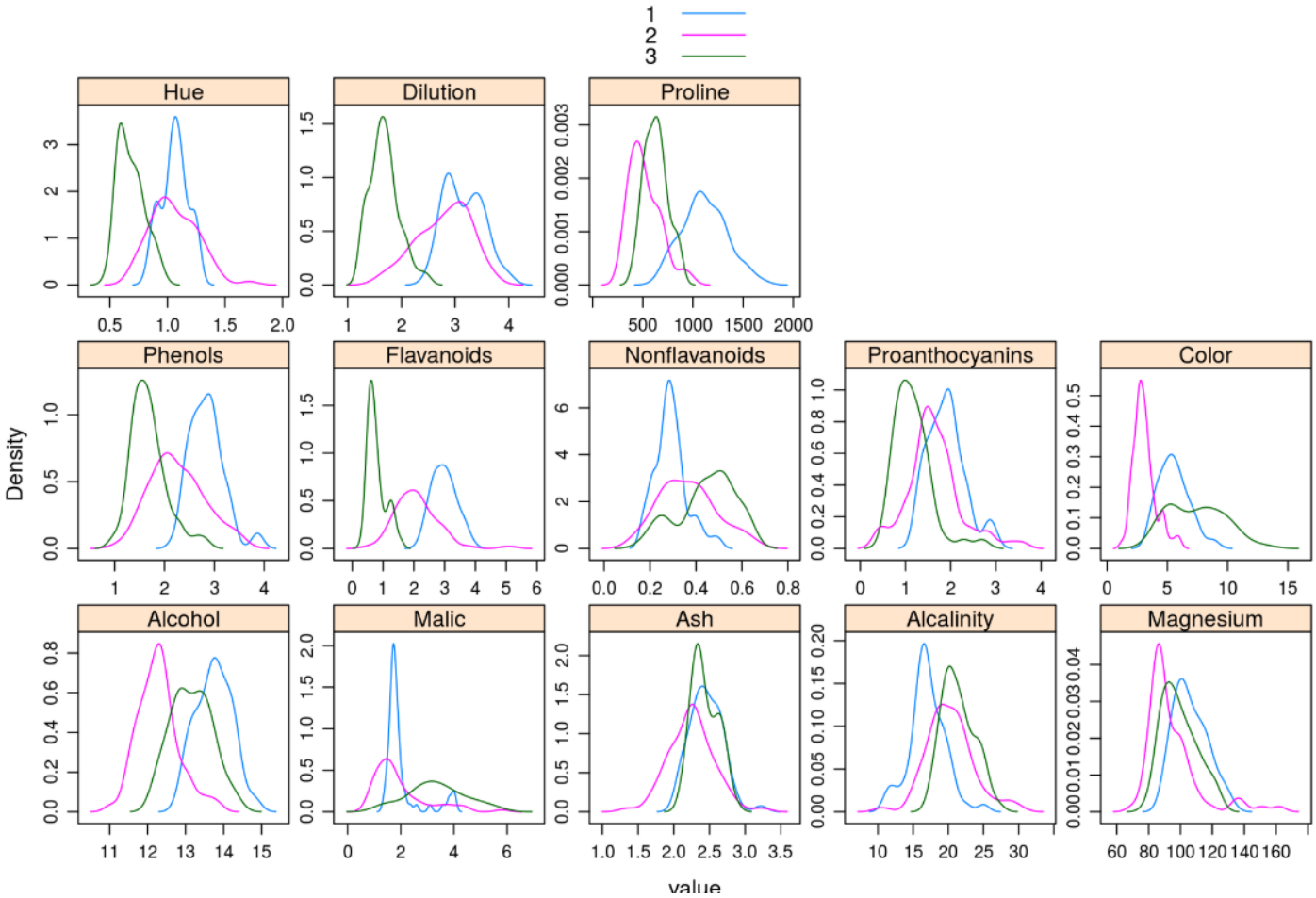
The above methods will be discussed in details along with the inference we can gain of the data from these methods.

DENSITY PLOTS

A density plot is a representation of the distribution of a numeric variable. It uses a kernel density estimate to show the probability density function of the variable or variables. Density plots are used to study the distribution of one or a few variables. Checking the distribution of your variables one by one is probably the first task you should do when you get a new dataset. It delivers a good quantity of information.

We create density plots of the 3 types of wine(classes) for each of the 13 chemical analyses (Alcohol, Malic, Ash, Alcalinity, Magnesium, Phenols, Flavanoids, Non Flavanoids, Proanthocyanins, Color, Hue, Dilution, Proline) to compare the effect of each of the chemicals in the classification of the type of wine.

In the diagram providing the density plot given in the next page the blue, pink and green curves represent the Type 1, Type 2 and Type 3 of wine respectively.



Analysing the density plots of each of the 13 chemicals we make several inferences which are given below

- As we can see by Hue we can easily distinguish between Type 1 and Type 3 of wine but it is expected to predict Type 2 of wine poorly.
- As for the dilution we can see it can easily predict Type 3 of wine but no such inference can be made about Type 2 and Type 1 of wine.
- As we can see by the chemical proline we can easily predict the Type 1 of wine but it is hard to distinguish or classify between Type 2 and Type 3 of wine.

- From the chemical phenol we can easily classify or predict Type 1 and Type 3 of wine but no such inference can be made about Type 2 of wine.
- As we can see by the chemical Flavanoids we can easily predict Type 3 and Type 1 of wine but it is expected to predict Type 2 of wine poorly.
- As we can see by the chemical Non Flavanoids we can easily predict Type 1 of wine but it is expected to predict Type 2 and Type 3 of wine poorly.
- As we can see by the chemicals Proanthocyanin, Ash and Magnesium the classes or the 3 Types of wine are highly overlapped and are expected to predict any response poorly.
- As for the color we can see it can easily predict Type 2 of wine but no such inference can be made about Type 1 and Type 3 of wine.
- As for the chemical alcohol we can easily distinguish or classify between Type 1 and Type 2 of wine but no such inference can be made of Type 3 of wine.
- As for the chemical Malic we can easily predict Type 1 of wine but it is expected to predict Type 2 and Type 3 poorly
- Lastly for the chemical Alcalinity we see that we can classify in between Type 1 and Type 3 of wine but no such inference can be made of Type 2 of wine.

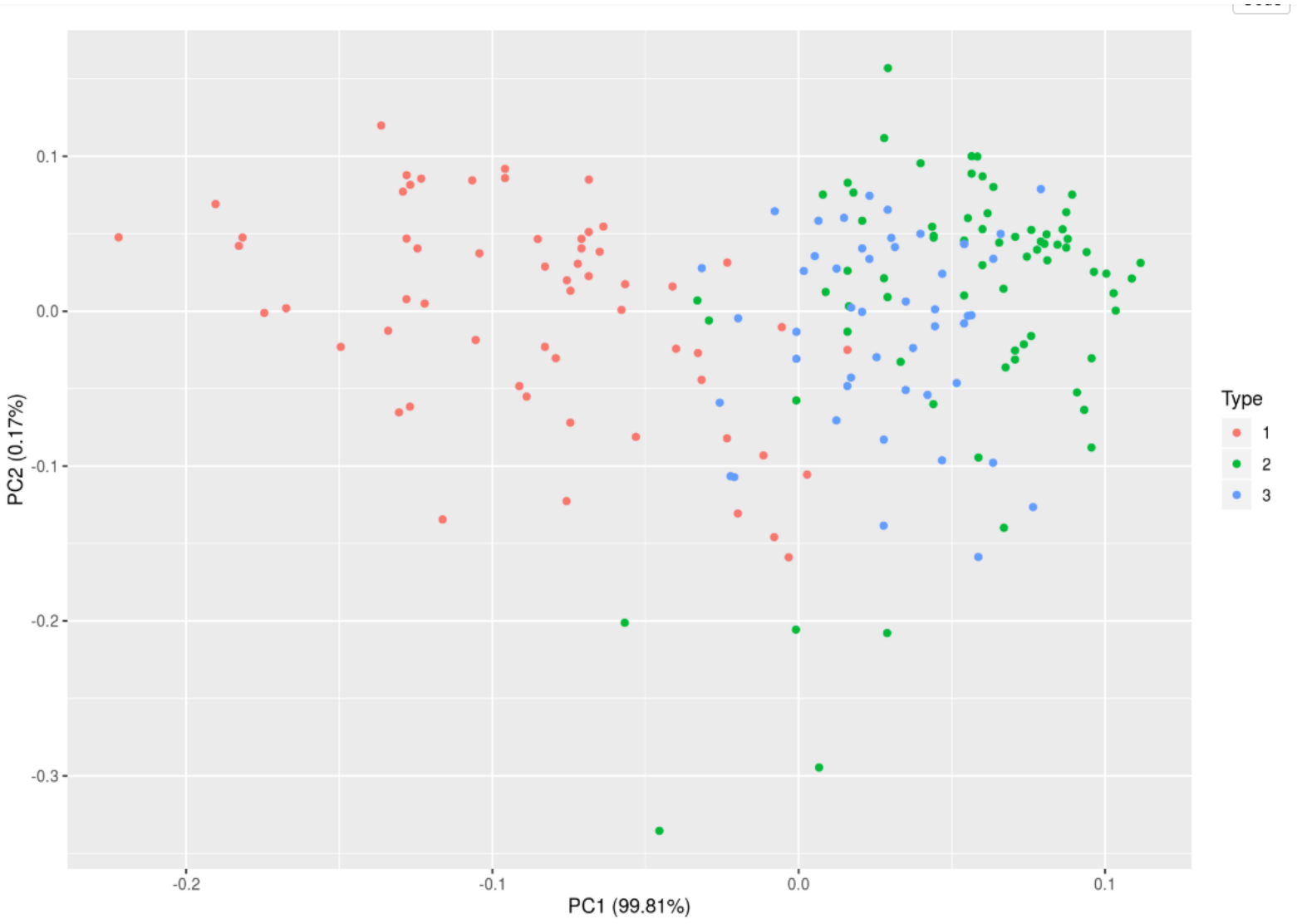
PRINCIPAL COMPONENT ANALYSIS:

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors (each being a linear combination of the variables and containing *n* observations) are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables. PCA is mostly used as a tool in exploratory data analysis and for making predictive models. It is often used to visualize genetic distance and relatedness between populations. PCA can be done by eigenvalue decomposition of a data covariance (or correlation) matrix or singular value decomposition of a data matrix, usually after a normalization step of the initial data. The normalization of each attribute consists of *mean centering* – subtracting each data value from its variable's measured mean so that its empirical mean (average) is zero. Some fields, in addition to normalizing the mean, do so for each variable's variance (to make it equal to 1);. The results of a PCA are usually discussed in terms of *component scores*, sometimes called *factor scores* (the transformed variable values corresponding to a particular data point), and *loadings* (the weight by which each standardized original variable should be multiplied to get the component score). If component scores are standardized to unit variance, loadings must contain the data variance in them (and that is the magnitude of eigenvalues). If component scores are not standardized (therefore they contain the data variance) then loadings must be unit-scaled, ("normalized") and

these weights are called eigenvectors; they are the cosines of orthogonal rotation of variables into principal components or back.

Now on performing Principal Component Analysis on the given "wine" dataset using prcomp function in R and then plotting PC1 against PC2 using the autoplot function of the package ggfortify in R we get the PCA plot given below.In the plot the red,green and blue dots represent Type 1,Type 2 and Type 3 of wine respectively.

From the PCA plot it is clear that the response categories are overlapped to a great extent and the categories(i.e 3 Types of wines) are not seperated by hyperplanes.



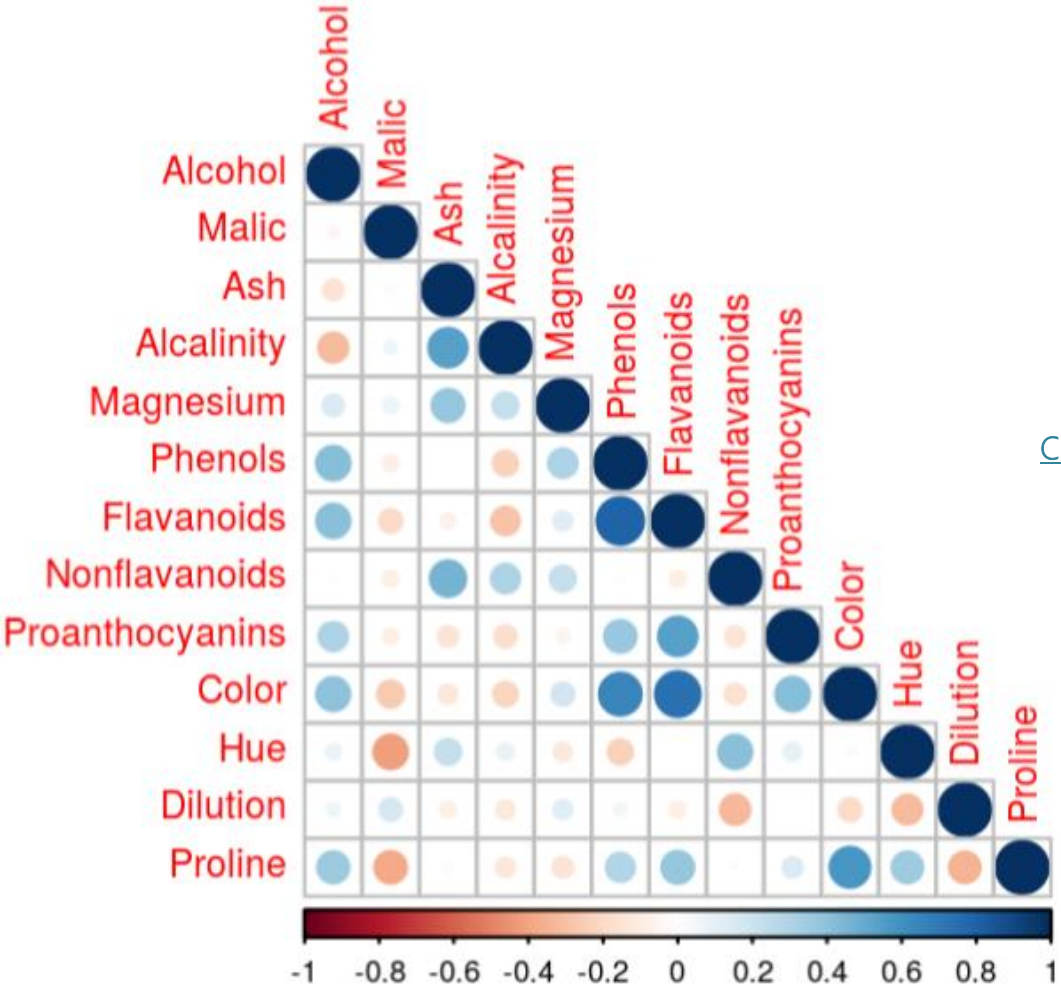
CORRELATION PLOT

Correlation plot or Correlogram is a graphical display of correlation matrix, where a correlation matrix is a table showing correlation coefficients between sets of variables. Correlation plots can be used to quickly find insights. It is used to investigate the dependence between multiple variables at the same time and to highlight the most correlated variables in a data table. Usually correlation coefficients are colored according to the value. Correlation matrix can be also reordered according to the degree of association between variables or clustered using hierarchical clustering algorithm. The usage of this visual is very simple and intuitive.

We use the "cor" function in R to compute the correlation matrices for each type of wine with respect to the 13 predictors(chemicals) and then used "corrplot" function from the package corrplot to create the lower triangular correlogram for each of the 3 Types of wine for the 13 chemicals.

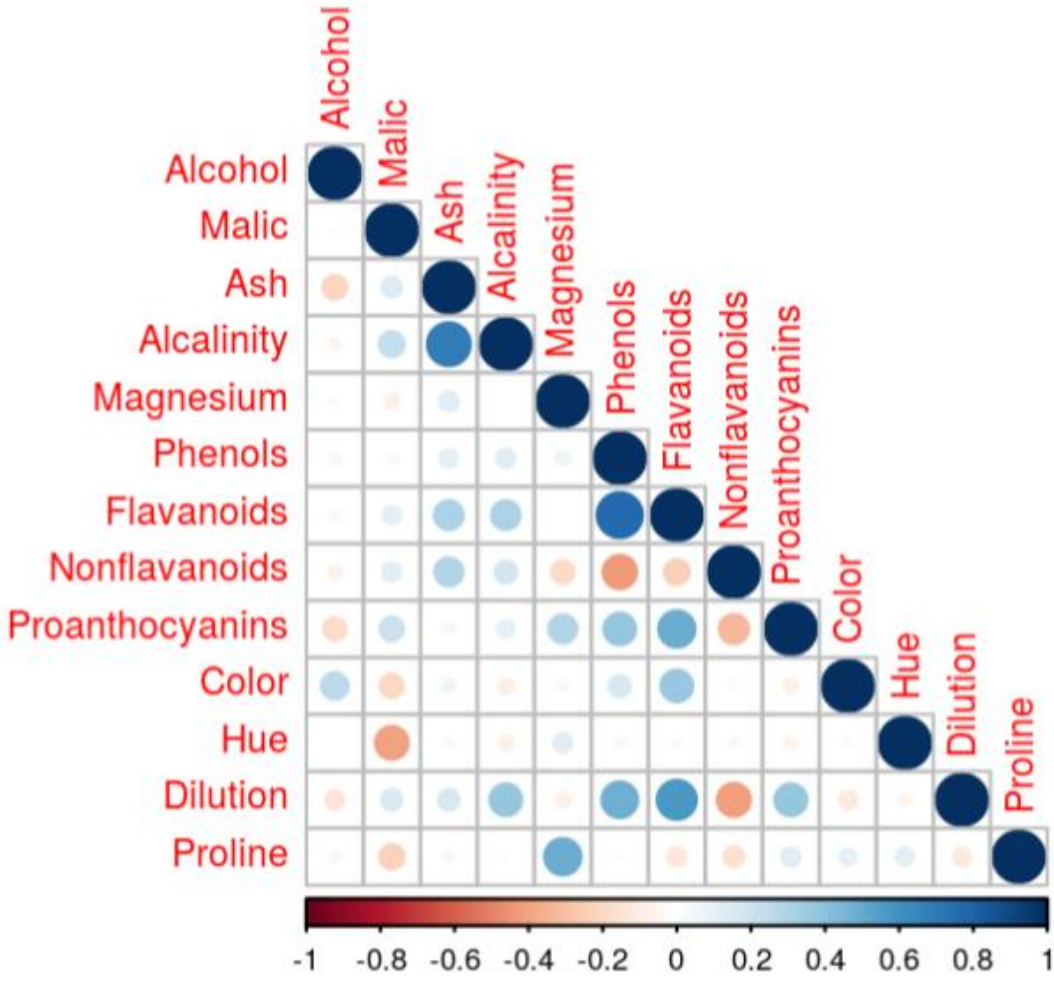
In the correlation plots of the 3 Types of wine shown in the next 2 pages positive correlations are displayed in blue and negative correlations in red color. Color intensity and size of the circle are proportional to the correlation coefficients. Below the correlogram, the legend color shows the correlation coefficients and the corresponding colors.

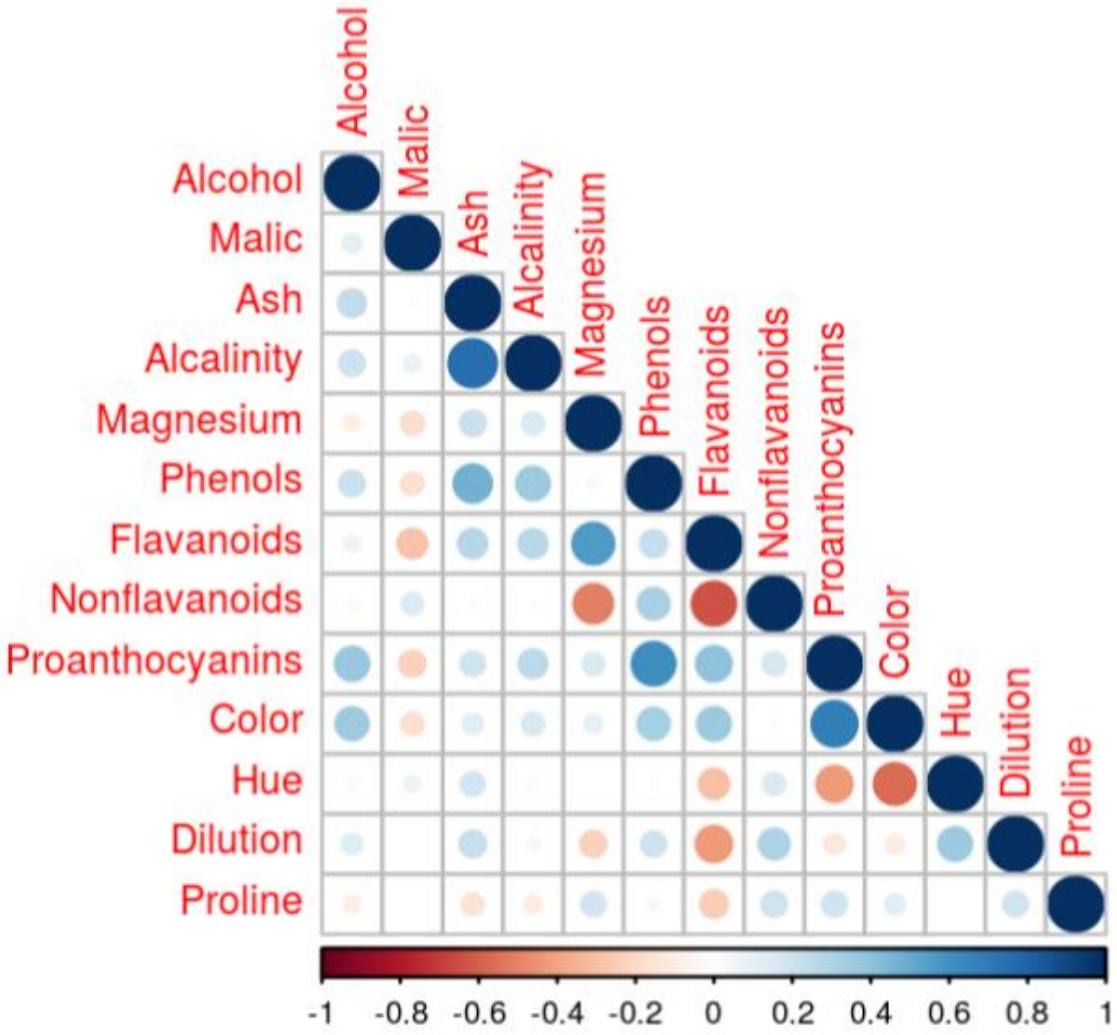
The correlation plots of the covariates separately drawn for the 3 groups (3 Types of wine) show the extent of positive correlation and darker the red higher the extent of negative correlation. For example: In the diagram of correlation plot of Type 1 wine the chemicals flavanoids and phenols have high positive correlation similarly in the diagram of correlation plot of Type 3 wine the chemicals flavanoids and non flavanoids have high negative correlation.



CORRELATION PLOT OF TYPE 1 WINE

CORRELATION PLOT OF TYPE 2 WINE





CORRELATION PLOT OF TYPE 3 WINE

TESTING FORTHE EQUALITY OF CORRELATION MATRICES

Now we test for the pairwise equality of the correlation matrices of Type 1,Type 2 and Type 3 of wine using the R function “cortest.mat” from the package psych. The cortest.mat function performs Chi Square Tests of whether a single Matrix is an Identity Matrix, or a pair of matrices are equal. Here the null hypothesis of testing is the equality of the matrices considered for testing.

On performing the tests on the correlation matrices of Type 1 on Type 2,Type 2 on Type 3 and Type 3 on Type 1, we observe in all the cases the null hypothesis is rejected, i.e the correlation matrices are not equal.

GENERAL INFERENCEFROM THE EDA

From Exploratory Data Analysis I have found that the variance covariance matrices(correlation matrices) of the predictors of the three types of wine are not equal. Thus linear discriminant analysis is not expected to work well with this wine dataset. We even observe that the PCA plot for the three types of wine overlap a lot and are not separated by any hyperplanes so any linear classifier like Logistic Regression, Linear Discriminant Analysis or Probit regression will yield poor results for this dataset. So I am building a three class classification model using the chemical properties as the predictors. So I will be training the following classifiers and comparing their efficiencies to get the best predictive classifier :

1. **K Nearest Neighbour Classifier**
2. **Naïve Bayes Classifer**
3. **Decision tree Classifier**
4. **Random Forest Classifier**

REQUIREMENTS FOR APPLICATION OF VARIOUS CLASSIFICATION TECHNIQUES

1. We first randomly split the dataset into training and testing sets. The training set will be used to train the classifiers and the testing set will be used to test and compare their accuracies. We will use 20% of the dataset as testing data.

2. Before proceeding to apply any classifier let us standardize the training data so that all the variables have mean 0 and variance 1 in the training data. This transformation makes sure that the effect of any specific variable does not overshadow the effect of other variables while building the classifiers. We will also fit this training data standardization to the testing data.

3. Before training classifiers we need to learn about K-fold Cross validation technique which is used to estimate the skill or efficiency of the machine learning models. It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods. A detailed description of K-fold cross validation is given below. This method is also used in estimating the value of K for the K Nearest Neighbor classifier for a dataset.

K-FOLD CROSSVALIDATION TECHNIQUE

Cross-validation is a re-sampling procedure used to evaluate machine learning models on a limited data sample.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:

   3.1. Take the group as a hold out or test data set

   3.2. Take the remaining groups as a training data set

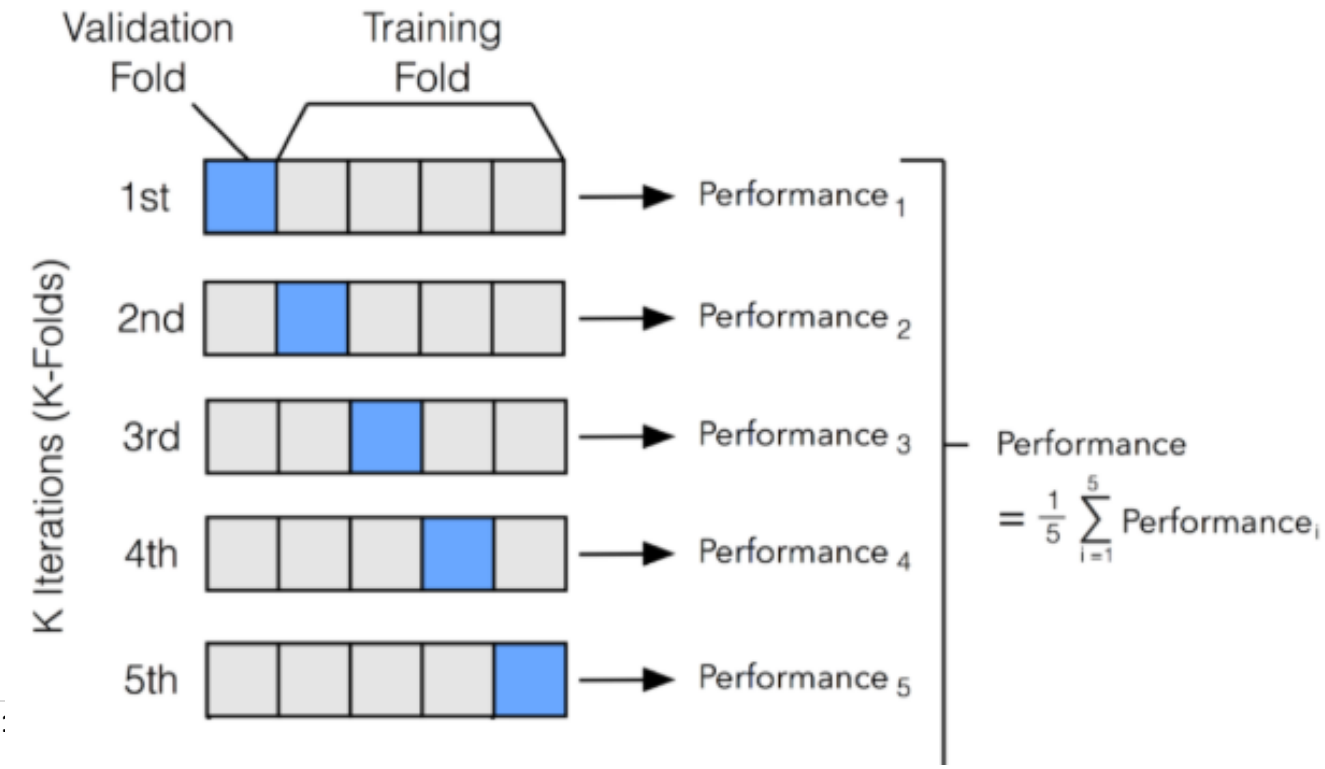
3.3. Fit a model on the training set and evaluate it on the test set

3.4. Retain the evaluation score and discard the model

4. Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given theopportunity to be used in the hold out set 1 time and used to train the model k-1 times. The choice of k is usually 5 or 10, but there is no formal rule. As k gets larger, the difference in size between the training set and the re-sampling subsets gets smaller. As this difference decreases, the bias of the technique becomes smaller

In training the classifiers and for comparing the skills of each of the classifier we make use of a **5-fold Cross Validation technique**(k = 5).The choice of k = 5 is made keeping in mind the limited sample size(178 observations) of the "wine" dataset. The following diagram will give a clear overview of the 5-fold Cross Validation technique.



K NEAREST NEIGHBOR CLASSIFIER

Theory :

K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection.It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data.KNN stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).If K=1 then the case will be simply assigned to the nearest neighbor. The functions which are generally used for measuring the distance of the new cases from the K neighbors when the variables are continuous are :

Distance functions

Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

Manhattan $\sum_{i=1}^{k}|x_i - y_i|$

Minkowski $\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$

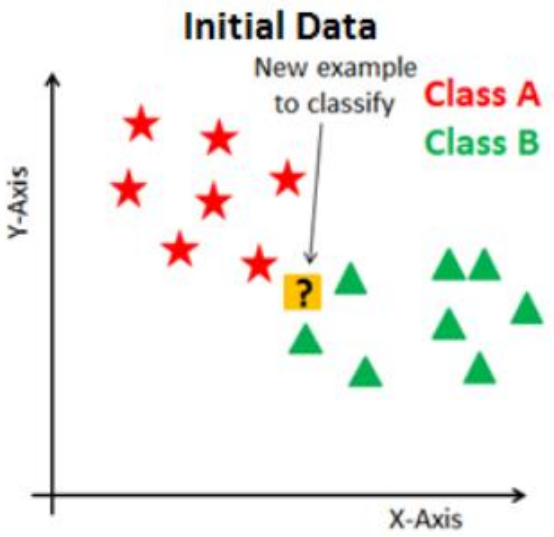
The KNN Algorithm

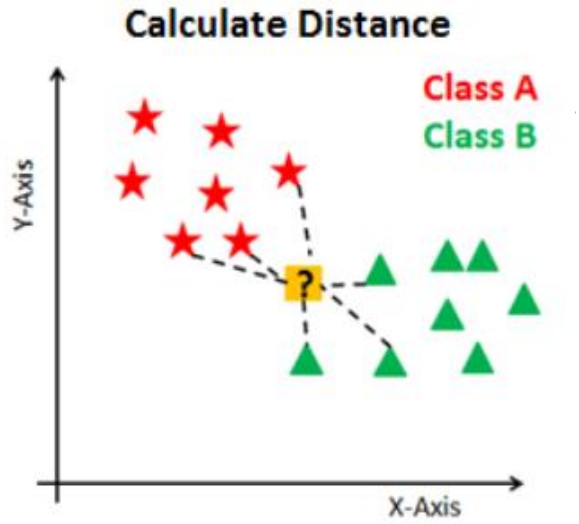
1. Load the data
2. Initialize K to your chosen number of neighbors
3. For each example in the data

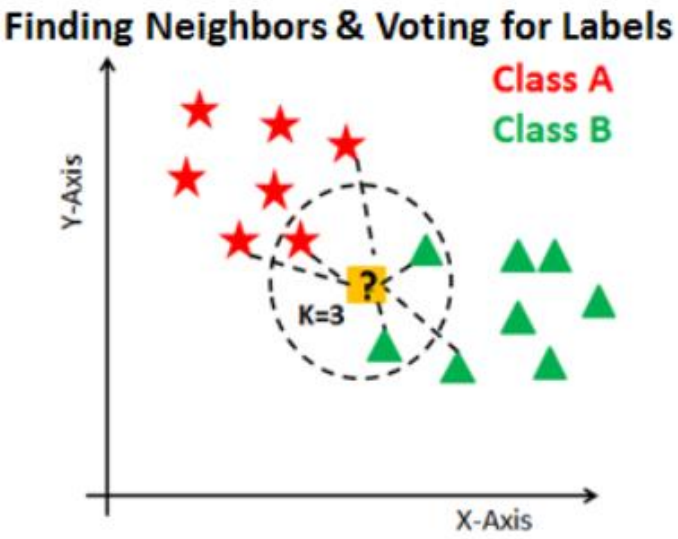
3.1. Calculate the distance between the query example and the current example from the data.

3.2. Add the distance and the index of the example to an ordered collection

4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
5. Pick the first K entries from the sorted collection
6. Return the mode of the K labels







ANALYSIS:

We will firstly apply 5-fold cross validation technique in the training dataset to obtain the optimal value of k.The algorithm for the 5 fold Cross Validation technique is mentioned above in details. From this technique using R programming we obtain the optimal value of **K = 18.**

So we will train a 18 - Nearest Neighbour Classifier in the training dataset and use that classifier to predict the testing dataset. Now let's see the results:

As we can see we get the **Confusion Matrix**, the **accuracy** as 1, **95% CI** as (0.9026,1) and the **Statistics by Class**.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3
##          1 11  0  0
##          2  0 16  0
##          3  0  0  9
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9026, 1)
##     No Information Rate : 0.4444
##     P-Value [Acc > NIR] : 2.096e-13
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            1.0000   1.0000     1.00
## Specificity            1.0000   1.0000     1.00
## Pos Pred Value         1.0000   1.0000     1.00
## Neg Pred Value         1.0000   1.0000     1.00
## Prevalence             0.3056   0.4444     0.25
## Detection Rate         0.3056   0.4444     0.25
## Detection Prevalence   0.3056   0.4444     0.25
## Balanced Accuracy      1.0000   1.0000     1.00
```

TEST ACCURACY BY 5- FOLD CROSS VALIDATION FOR KNN IS :0.9719212

NAÏVE BAYES CLASSIFIER

Theory:

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem.

Bayes Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Using Bayes theorem, we can find the probability of **A** happening, given that **B** has occurred. Here, **B** is the evidence and **A** is the hypothesis. The assumption made here is that the predictors/features are independent. That is presence of one particular feature does not affect the other. Hence it is called naive.Example:Let us take an example to get some better intuition. Consider the problem of playing golf. The dataset is represented as below.

	OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	PLAY GOLF
0	Rainy	Hot	High	False	No
1	Rainy	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Sunny	Mild	High	False	Yes
4	Sunny	Cool	Normal	False	Yes
5	Sunny	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Rainy	Mild	High	False	No
8	Rainy	Cool	Normal	False	Yes
9	Sunny	Mild	Normal	False	Yes
10	Rainy	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Sunny	Mild	High	True	No

We classify whether the day is suitable for playing golf, given the features of the day. The columns represent these features and the rows represent individual entries. If we take the first row of the dataset, we can observe that is not suitable for playing golf if the outlook is rainy, temperature is hot, humidity is high and it is not windy. We make two assumptions here, one as stated above we consider that these predictors are independent. That is, if the temperature is hot, it does not necessarily mean that the humidity is high. Another assumption made here is that all the predictors have an equal effect on the outcome. That is, the day being windy does not have more importance in deciding to play golf or not.

According to this example, Bayes theorem can be rewritten as:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

The variable **y** is the class variable(play golf), which represents if it is suitable to play golf or not given the conditions. Variable **X** represent the parameters/features.**X** is given as,

$$X = (x_1, x_2, x_3, ..., x_n)$$

Here x_1,x_2….x_n represent the features, i.e they can be mapped to outlook, temperature, humidity and windy. By substituting for **X** and expanding using the chain rule we get,

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

Now, you can obtain the values for each by looking at the dataset and substitute them into the equation. For all entries in the dataset, the denominator does not change, it remain static. Therefore, the denominator can be removed and a proportionality can be introduced.

$$P(y|x_1,...,x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$$

In our case, the class variable(**y**) has only two outcomes, yes or no. There could be cases where the classification could be multivariate. Therefore, we need to find the class **y** with maximum probability.

$$y = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y)$$

Using the above function, we can obtain the class, given the predictors.

Naive Bayes algorithms are mostly used in sentiment analysis, spam filtering, recommendation systems etc. They are fast and easy to implement but their biggest disadvantage is that the requirement of predictors to be independent. In most of the real life cases, the predictors are dependent, this hinders the performance of the classifier.

ANALYSIS:

We will be training Naïve Bayes classifier without Kernel on the training dataset.Now let's see the results:

As we can see we get the **Confusion Matrix**, the **accuracy** as 1, **95% CI** as (0.9026,1) and the **Statistics by Class**.

TEST ACCURACY BY 5- FOLD CROSS VALIDATION FOR NAÏVE BAYES CLASSIFIER WITHOUT KERNEL IS :**0.9645320**

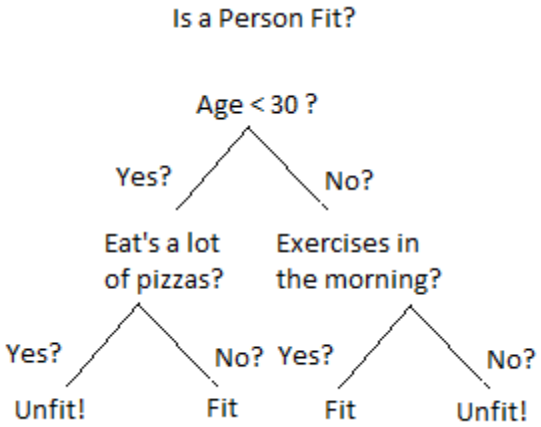
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3
##          1 11  0  0
##          2  0 16  0
##          3  0  0  9
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9026, 1)
##     No Information Rate : 0.4444
##     P-Value [Acc > NIR] : 2.096e-13
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            1.0000   1.0000     1.00
## Specificity            1.0000   1.0000     1.00
## Pos Pred Value         1.0000   1.0000     1.00
## Neg Pred Value         1.0000   1.0000     1.00
## Prevalence             0.3056   0.4444     0.25
## Detection Rate         0.3056   0.4444     0.25
## Detection Prevalence   0.3056   0.4444     0.25
## Balanced Accuracy      1.0000   1.0000     1.00
```

DECISION TREE CLASSIFIER

Theory: A Decision Tree is a simple representation for classifying examples. It is a Supervised Machine Learning where the data is continuously split according to a certain parameter. Classification tree is a specific type of Decision Tree where the decision variable is categorical or discrete. Such a tree is built through a process known as binary recursive partitioning. This is an iterative process of splitting the data into partitions, and then splitting it up further on each of the branches.

Classification Tree consists of :

1. **Nodes** : Test for the value of a certain attribute.
2. **Edges/ Branch** : Correspond to the outcome of a test and connect to the next node or leaf.
3. **Leaf nodes** : Terminal nodes that predict the outcome (represent class labels or class distribution).



To understand the concept of Classification Tree consider the above example. Let's say you want to predict whether a person is fit or unfit, given their information like age, eating habits, physical activity, etc. The decision nodes are the questions like 'What's the age?', 'Does he exercise?', 'Does he eat a lot of pizzas'? And the leaves represent outcomes like either 'fit', or 'unfit'.

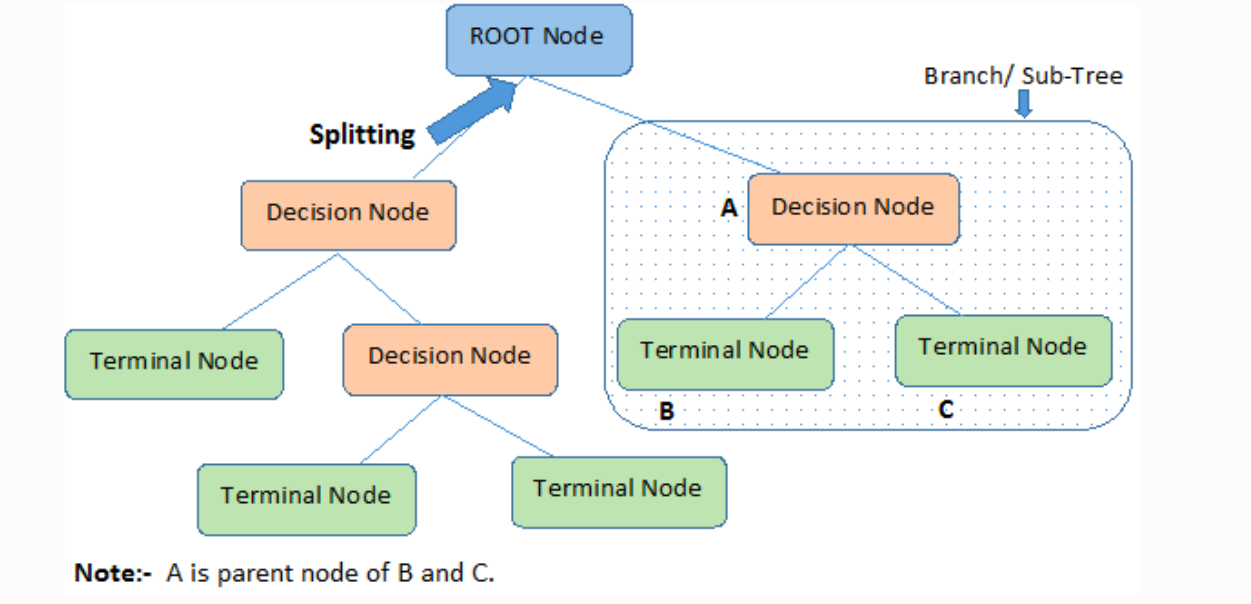
Decision Tree Algorithm

The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data(training data).

In Decision Trees, for predicting a class label for a record we start from the **root** of the tree. We compare the values of the root attribute with the record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

Important Terminology related to Decision Trees

1. **Root Node:** It represents the entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called the decision node.
4. **Leaf / Terminal Node:** Nodes do not split is called Leaf or Terminal node.
5. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.
6. **Branch / Sub-Tree:** A subsection of the entire tree is called branch or sub-tree.
7. **Parent and Child Node:** A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.



Decision trees classify the examples by sorting them down the tree from the root to some leaf/terminal node, with the leaf/terminal node providing the classification of the example.Each node in the tree acts as a test case for some attribute, and each edge descending from the node corresponds to the possible answers to the test case. This process is recursive in nature and is repeated for every subtree rooted at the new node.

Assumptions while creating Decision Tree

Below are some of the assumptions we make while using Decision tree:

- In the beginning, the whole training set is considered as the **root.**
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- Records are **distributed recursively** on the basis of attribute values.
- Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

Decision Trees follow **Sum of Product (SOP) r**epresentation. The Sum of product (SOP) is also known as **Disjunctive Normal Form**. For a class, every branch from the root of the tree to a leaf node having the same class is conjunction (product) of values, different branches ending in that class form a disjunction (sum).

The primary challenge in the decision tree implementation is to identify which attributes do we need to consider as the root node and each level. Handling this is to know as the attributes selection. We have different attributes selection measures to identify the attribute which can be considered as the root note at each level.

Learning a CART Model From Data

Creating a CART model involves selecting input variables and split points on those variables until a suitable tree is constructed.The selection of which input variable to use and the specific split or cut-point is chosen using a greedy algorithm to minimize a cost function. Tree construction ends using a predefined stopping criterion, such as a minimum number of training instances assigned to each leaf node of the tree.

Greedy Splitting

Creating a binary decision tree is actually a process of dividing up the input space. A greedy approach is used to divide the space called **recursive binary splitting.**

This is a numerical procedure where all the values are lined up and different split points are tried and tested using a cost function. The split with the best cost (lowest cost because we minimize cost) is selected.

All input variables and all possible split points are evaluated and chosen in a greedy manner (e.g. the very best split point is chosen each time).

For classification the Gini index function is used which provides an indication of how "pure" the leaf nodes are (how mixed the training data assigned to each node is).

G = sum(pk * (1 – pk))

Where G is the Gini index over all classes, pk are the proportion of training instances with class k in the rectangle of interest. A node that has all classes of the same type (perfect class purity) will have G=0, where as a G that has a 50-50 split of classes for a binary classification problem (worst purity) will have a G=0.5.For a binary classification problem, this can be re-written as:

G = 2 * p1 * p2

or

G = 1 – (p1^2 + p2^2)

The Gini index calculation for each node is weighted by the total number of instances in the parent node. The Gini score for a chosen split point in a binary classification problem is therefore calculated as follows:

G = ((1 – (g1_1^2 + g1_2^2)) * (ng1/n)) + ((1 – (g2_1^2 + g2_2^2)) * (ng2/n))

Where G is the Gini index for the split point, g1_1 is the proportion of instances in group 1 for class 1, g1_2 for class 2, g2_1 for group 2 and class 1, g2_2 group 2 class 2, ng1 and ng2 are the total number of instances in group 1 and 2 and n are the total number of instances we are trying to group from the parent node.

Stopping Criterion

The recursive binary splitting procedure described above needs to know when to stop splitting as it works its way down the tree with the training data.The most common stopping procedure is to use a minimum count on the number of training instances assigned to each leaf node. If the count is less than some minimum then the split is not accepted and the node is taken as a final leaf node.The count of training members is tuned to the dataset, e.g. 5 or 10. It defines how specific to the training data the tree will be. Too specific (e.g. a count of 1) and the tree will overfit the training data and likely have poor performance on the test set.

Pruning The Tree

The stopping criterion is important as it strongly influences the performance of your tree. You can use **pruning** after learning your tree to further lift performance.

The complexity of a decision tree is defined as the number of splits in the tree. Simpler trees are preferred. They are easy to understand (you can print them out and show them to subject matter experts), and they are less likely to overfit your data.The fastest and simplest pruning method is to work through each leaf node in the tree and evaluate the effect of removing it using a hold-out test set. Leaf nodes are removed only if it results in a drop in the overall cost function on the entire test set. You stop removing nodes when no further improvements can be made.More sophisticated pruning methods can be used such as cost complexity pruning (also called weakest link pruning) where a learning parameter (alpha) is used to weigh whether nodes can be removed based on the size of the sub-tree.

ANALYSIS:

We will be training the Decision tree classifier to the training dataset. For that firstly we need to compute the complexity parameter by the aforementioned method of 5-fold Cross Validation. From applying the 5-fold Cross Validation we get the **optimal complexity parameter = 0.**

Now after fitting the Decision tree with the optimal complexity parameter = 0, we get :

1. The **Confusion Matrix**

2. **Accuracy (0.9167)**

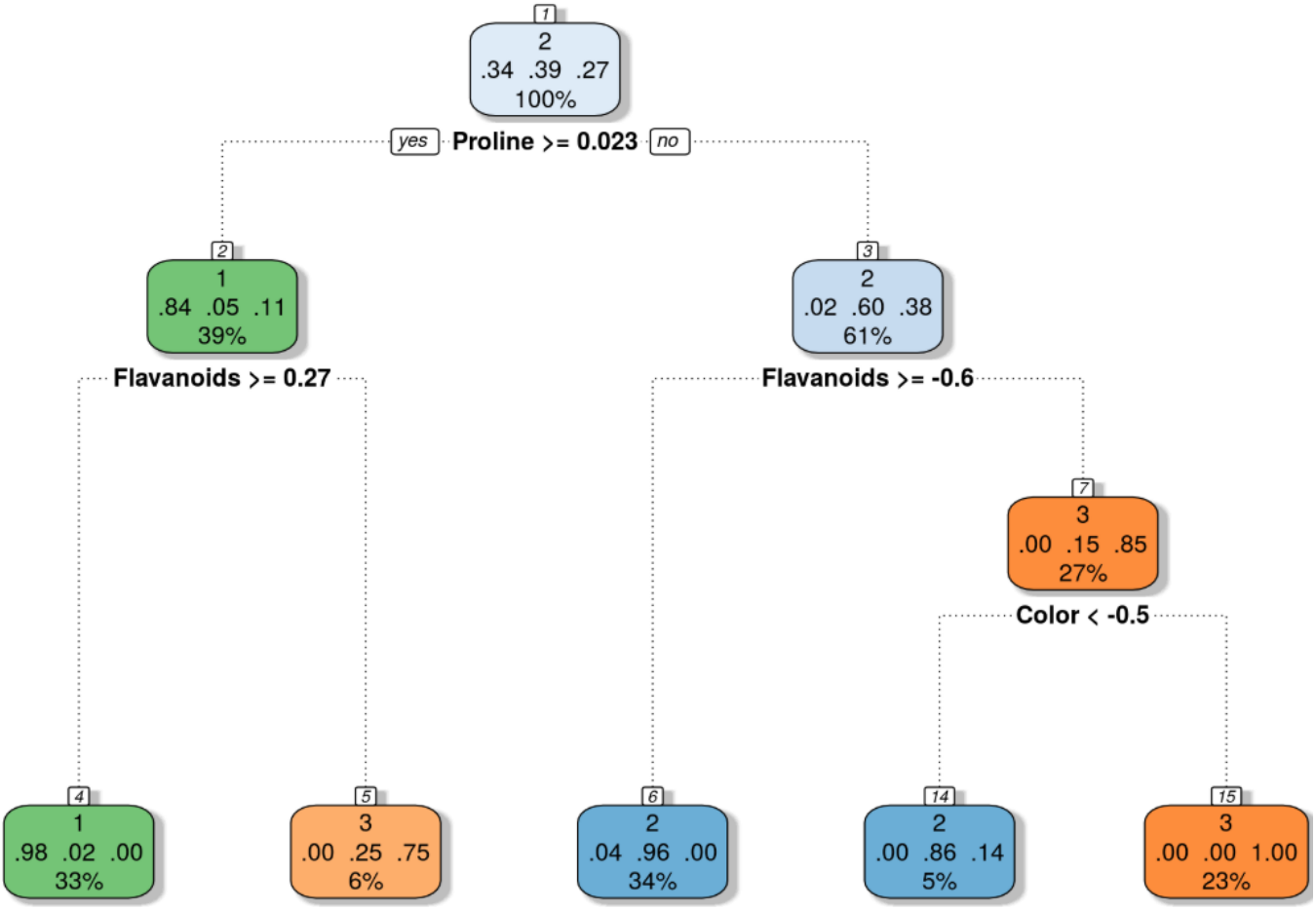
3. **95% Confidence Interval (0.7753,0.9825)**

4. **Statistics by class**

TEST ACCURACY BY 5- FOLD CROSS VALIDATION FOR DECISION TREE CLASSIFIER IS: 0.9091133

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3
##          1 10  0  0
##          2  0 15  1
##          3  1  1  8
##
## Overall Statistics
##
##                Accuracy : 0.9167
##                  95% CI : (0.7753, 0.9825)
##     No Information Rate : 0.4444
##     P-Value [Acc > NIR] : 3.139e-09
##
##                   Kappa : 0.8714
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.9091   0.9375   0.8889
## Specificity            1.0000   0.9500   0.9259
## Pos Pred Value         1.0000   0.9375   0.8000
## Neg Pred Value         0.9615   0.9500   0.9615
## Prevalence             0.3056   0.4444   0.2500
## Detection Rate         0.2778   0.4167   0.2222
## Detection Prevalence   0.2778   0.4444   0.2778
## Balanced Accuracy      0.9545   0.9437   0.9074
```

ON VISUALIZING THE DECISION TREE WE GET:



RANDOM FOREST CLASSIFIER

Since we know Random Forest is the culmination of many Decision trees. We would like to know :

Why a Forest is better than One Tree?

You might be tempted to ask why not just use one decision tree? It seems like the perfect classifier since it did not make any mistakes! A critical point to remember is that the tree made no mistakes **on the training data**. We expect this to be the case since we gave the tree the answers and didn't limit the max depth (number of levels). The objective of a machine learning model is to generalize well to **new data** it has never seen before.

Over fitting occurs when we have a very flexible model (the model has a high capacity) which essentially **memorizes** the training data by fitting it closely. The problem is that the model learns not only the actual relationships in the training data, but also any noise that is present. A flexible model is said to have high **variance** because the learned parameters (such as the structure of the decision tree) will vary considerably with the training data.

On the other hand, an inflexible model is said to have high **bias** because it makes **assumptions** about the training data (it's biased towards pre-conceived ideas of the data.) For example, a linear classifier makes the assumption that the data is linear and does not have the flexibility to fit non-linear relationships. An inflexible model may not have the capacity to fit even the training data and in both cases — high variance and high bias — the model is not able to generalize well to new data.

The balance between creating a model that is so flexible it memorizes the training data versus an inflexible model that can't learn the training data is known as the **bias-variance tradeoff**and is a foundational concept in machine learning.

The reason the decision tree is prone to **over fitting** when we don't limit the maximum depth is because it has unlimited flexibility, meaning that it can keep growing until it has exactly one leaf node for every single observation, perfectly classifying all of them. If you go back to the image of the decision tree and limit the maximum depth to 2 (making only a single split), the classifications are no longer 100% correct. We have reduced the variance of the decision tree but at the cost of increasing the bias.

As an alternative to limiting the depth of the tree, which reduces variance (good) and increases bias (bad), we can combine many decision trees into a single ensemble model known as the random forest.

RANDOM FOREST

The random forest is a model made up of many decision trees. Rather than just simply averaging the prediction of trees (which we could call a "forest"), this model uses two key concepts that gives it the name *random*:

1. Random sampling of training data points when building trees

2. Random subsets of features considered when splitting nodes

Random sampling of training observations

When training each tree in a random forest learns from a **random** sample of the data points. The samples are **drawn with replacement**, known as **bootstrapping**, which means that some samples will be used multiple times in a single tree. The idea is that by training each tree on different samples, although each tree might have high variance with respect to a particular set of the training data, overall, the entire forest will have lower variance but not at the cost of increasing the bias.

At test time, predictions are made by averaging the predictions of each decision tree. This procedure of training each individual learner on different bootstrapped subsets of the data and then averaging the predictions is known as *bagging*, short for **bootstrap aggregating**.

Random Subsets of features for splitting nodes

The other main concept in the random forest is that only a subset of all the features are considered for splitting each node in each decision tree. Generally this is set to sqrt(n_features) for classification meaning that if there are 16 features, at each node in each tree, only 4 random features will be considered for splitting the node. (The random forest can also be trained considering all the features at every node)

If you can comprehend a single decision tree, the idea of **bagging**, and random subsets of features, then you have a pretty good understanding of how a random forest works:

The random forest combines hundreds or thousands of decision trees, trains each one on a slightly different set of the observations, splitting nodes in each tree considering a limited number of the features. The final predictions of the random forest are made by averaging the predictions of each individual tree.

To understand why a random forest is better than a single decision tree imagine the following scenario: you have to decide whether Tesla stock will go up and you have access to a dozen analysts who have no prior knowledge about the company. Each analyst has low bias because they don't come in with any assumptions, and is allowed to learn from a dataset of news reports.

This might seem like an ideal situation, but the problem is that the reports are likely to contain noise in addition to real signals. Because the analysts are basing their predictions entirely on the data — they have high flexibility — they can be swayed by irrelevant information. The analysts might come up with differing predictions from the same dataset. Moreover, each individual analyst has high variance and would come up with drastically different predictions if given a **different** training set of reports.

The solution is to not rely on any one individual, but pool the votes of each analyst. Furthermore, like in a random forest, allow each analyst access to only a section of the reports and hope the effects of the noisy information will be cancelled out by the sampling. In real life, we rely on multiple sources , and therefore, not only is a decision tree intuitive, but so is the idea of combining them in a random forest.

Features of Random Forests

- It is unexcelled in accuracy among current algorithms.
- It runs efficiently on large data bases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing error in class population unbalanced data sets.
- Generated forests can be saved for future use on other data.
- Prototypes are computed that give information about the relation between the variables and the classification.
- It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data.
- The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
- It offers an experimental method for detecting variable interactions.

How Random Forest algorithm works?

There are two stages in Random Forest algorithm, one is random forest creation, the other is to make a prediction from the random forest classifier created in the first stage. The whole process is shown below, and it's easy to understand using the figure.

Here we firstly show the Random Forest creation pseudo code:

1. Randomly select "**K**" features from total "**m**" features where **k << m**
2. Among the "**K**" features, calculate the node "**d**" using the best split point
3. Split the node into **daughter nodes** using the **best split**
4. Repeat the **a to c** steps until "l" number of nodes has been reached
5. Build forest by repeating steps **a to d** for "n" number times to create **"n" number of trees**

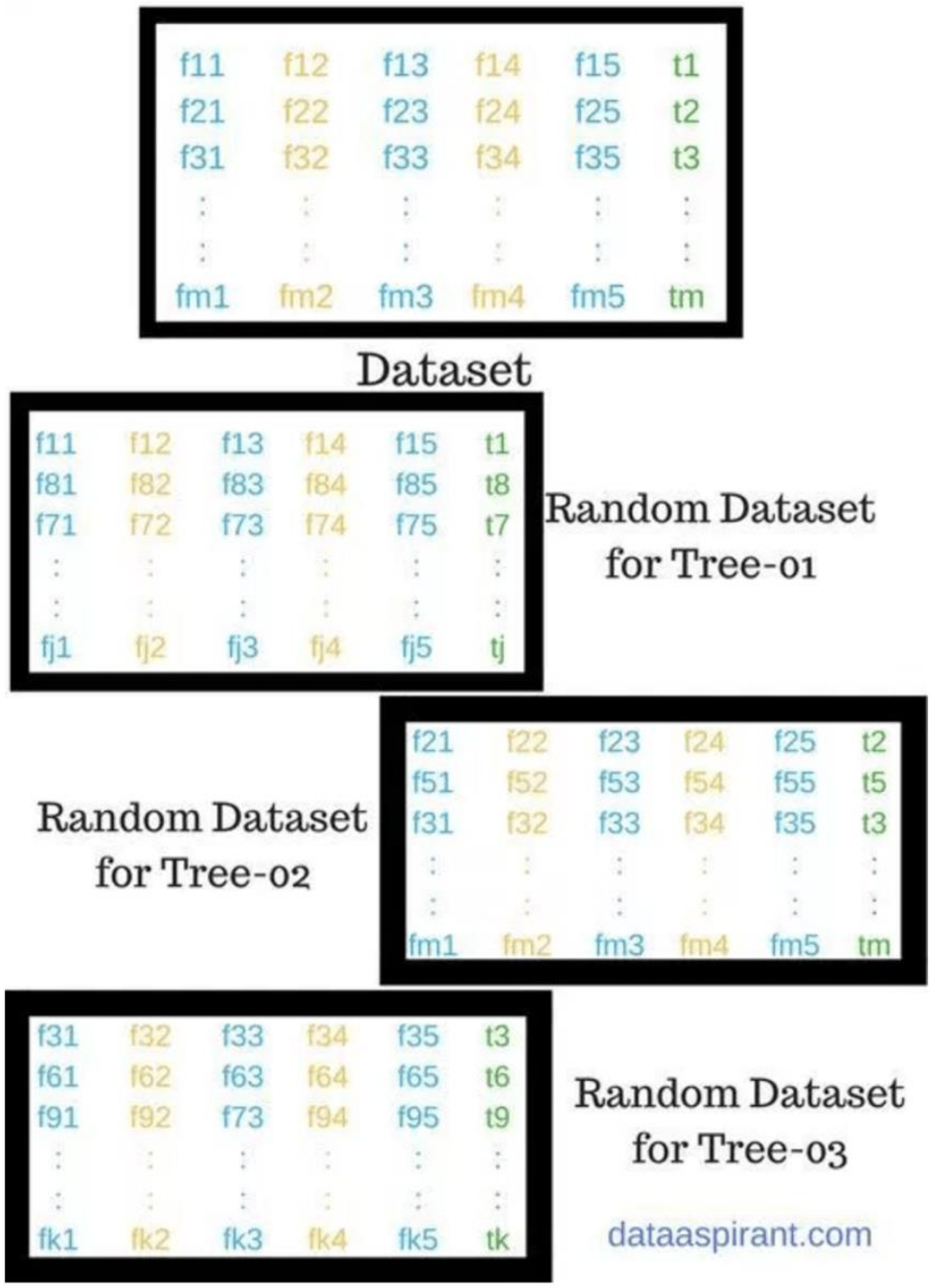
The figure given in the next page shows the process of randomly selecting features

In the next stage, with the random forest classifier created, we will make the prediction. The random forest prediction pseudocode is shown below:

1. Takes the **test features** and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target)
2. Calculate the **votes** for each predicted target

3. Consider the **high voted** predicted target as the **final prediction** from the random forest algorithm

The process is easy to understand, but it's somehow efficient.



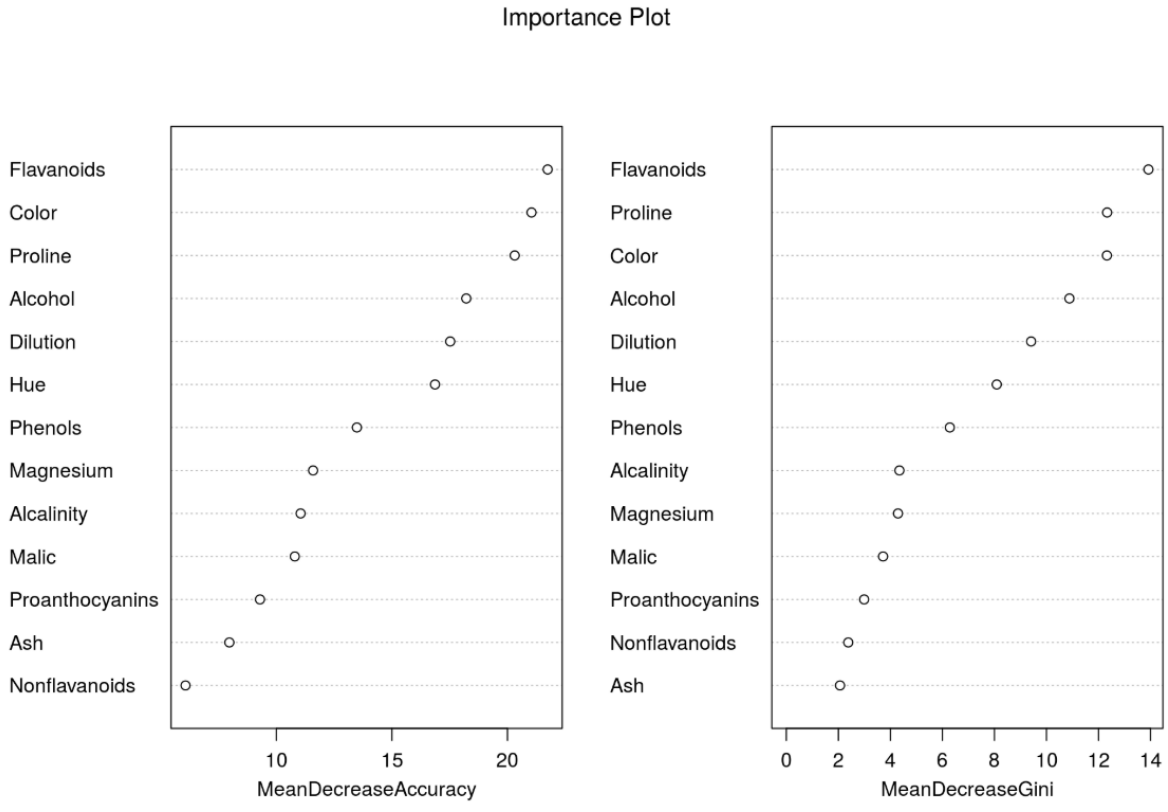
ANALYSIS:

We will now train a random forest classifier to the training dataset. But before that we need to find the optimal number of predictors to be sampled during the formation of each tree by the method of 5-fold Cross Validation. By applying 5-Fold Cross Validation in R programming we compute the value of **the optimal number of predictors to be sampled = 2.**

Now we proceed for building a random forest classifier with two predictors to be used at each stage of tree formation. Also we will test for its accuracy. We get:

```
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 1.41%
## Confusion matrix:
##    1  2  3 class.error
## 1 48  0  0  0.00000000
## 2  1 53  1  0.03636364
## 3  0  0 39  0.00000000
```

Now we will check for the importance of the predictors for the sake of predicting using "varImpPlot" function in R from the package randomForest. We get the following diagram:



Now on testing the test accuracy of the Random forest classifier we get – the **Confusion Matrix, Accuracy (1), 95%CI (0.9026,1), Statistics by Class.**

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3
##          1 11  0  0
##          2  0 16  0
##          3  0  0  9
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9026, 1)
##     No Information Rate : 0.4444
##     P-Value [Acc > NIR] : 2.096e-13
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            1.0000   1.0000     1.00
## Specificity            1.0000   1.0000     1.00
## Pos Pred Value         1.0000   1.0000     1.00
## Neg Pred Value         1.0000   1.0000     1.00
## Prevalence             0.3056   0.4444     0.25
## Detection Rate         0.3056   0.4444     0.25
## Detection Prevalence   0.3056   0.4444     0.25
## Balanced Accuracy      1.0000   1.0000     1.00
```

TEST ACCURACY BY 5- FOLD CROSS VALIDATION FOR RANDOM FOREST CLASSIFIER IS: 0.9790640

CONCLUSION

On comparing the skills of each of the classifier using the 5-Fold Cross Validation method we get that the **Random Forest Classifier will be the best predictive model to predict the type of the wine.**

Classifier	Accuracy
19 - Nearest Neighbour	0.9719212
Naive Bayes without Kernel	0.9645320
Classification Tree	0.9091133
Random Forest	0.9790640

And secondly from the Random Forest Classifier and the importance plot we can conclude that

Proline and Flavanoids are the two predictors most useful in predicting the type of the wine.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude towards my supervisor and mentor professor Pallabi Ghosh for her endless support .Without her encouragement and guidance it would have been impossible for me to finish this project .Under her supervision I could easily grasp topics like Exploratory Data Analysis, Decision Tree and Random Forest.

I would also like to thank all the professors of the Department of Statistics, St. Xaviers College , Kolkata for their support and guidance.

Performing this project has not only been a great experience, but has helped me gain knowledge about Machine Learning and has made me more passionate about the field of statistics and data analysis.

BIBLIOGRAPHY

I would like to cite the following sources that helped in the completion of my dissertation project:

- An Introduction to Statistical Learningwith Applications in R written by - Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani.
- From websites like www.machinelearningmastery.com ,www.stat.berkeley.edu, wikipedia and etc.

R - PROGRAMMING CODING FILE

1. THE DATASET:

```
library(rattle.data)
head(wine, 10)
```

2. THE SUMMARY OF THE DATASET:

```
library(Hmisc)
describe(wine)
```

3.DATA VISUALIZATION :

3.1. DENSITY PLOT:

```
library(reshape)
wine.melt<- melt(wine)
str(wine.melt)
library(lattice)
densityplot(~ value | variable, wine.melt, groups = Type, plot.points = FALSE
, auto.key = TRUE, scales = "free")
```

3.2. PRINCIPAL COMPONENT ANALYSIS PLOT

```
library(ggfortify)
autoplot(prcomp(wine[ , -1]), data = wine, colour = 'Type')
```

3.3. CORRELATION PLOTS:

```
library(corrplot)
par(mfrow = c(1, 3))
corrplot(cor(wine[wine$Type == 1 , -1]), method = 'circle', type = 'lower'
)
corrplot(cor(wine[wine$Type == 2 , -1]), method = 'circle', type = 'lower'
)
corrplot(cor(wine[wine$Type == 3 , -1]), method = 'circle', type = 'lower'
)
```

4. TESTING FOR EQUALITY OF THE CORRELATION MATRICES

4.1. TYPE-1 VS TYPE-2

```
library(psych)
cortest.mat(R1 = cor(wine[wine$Type == 1 , -1]), R2 = cor(wine[wine$Typ
e == 2 , -1]), n1 = length(which(wine$Type == 1)), n2 = length(which(wi
ne$Type == 2)))
```

4.2. TYPE-2 VS TYPE-3

```
cortest.mat(R1 = cor(wine[wine$Type == 2 , -1]), R2 = cor(wine[wine$Typ
e == 3 , -1]), n1 = length(which(wine$Type == 2)), n2 = length(which(wi
ne$Type == 3)))
```

4.3. TYPE-3 VS TYPE-1

```
cortest.mat(R1 = cor(wine[wine$Type == 3 , -1]), R2 = cor(wine[wine$Typ
e == 1 , -1]), n1 = length(which(wine$Type == 3)), n2 = length(which(wi
ne$Type == 1)))
```

5. APPLYING VARIOUS CLASSIFICATION TECHNIQUES

```
set.seed(123)
index<- sample(1:nrow(wine), 0.8*nrow(wine), replace = FALSE)
wine.train<- wine[index, ]
wine.test<- wine[-index, ]
library(dataPreparation)
```

```
scales<- build_scales(dataSet = wine.train[ , -1], verbose = TRUE)

wine.train[ , -1] <- fastScale(dataSet = wine.train[ , -1], scales = sc
ales, verbose = TRUE)

wine.test[ , -1] <- fastScale(dataSet = wine.test[ , -1], scales = scal
es, verbose = TRUE)
```

5.1. APPLYING K – NEAREST NEIGHBOR CLASSIFIER

```
library(caret)

library(e1071)

set.seed(123)

trControl<- trainControl(method  = "cv", number  = 5)

train(Type ~ ., method = "knn", tuneGrid = expand.grid(k = 1:25), trCon
trol = trControl, metric = "Accuracy", data = wine.train)

library(class)

predict.test.knn<- knn(train = wine.train[ , -1], test = wine.test[ , -
1], cl = wine.train[ , 1], k = 18)

caret::confusionMatrix(data = predict.test.knn, reference = wine.test[
, 1])
```

5.2. APPLYING NAIVE BAYES CLASSIFIER

```
train(Type ~ ., method = "nb", trControl = trControl, metric = "Accurac
y", data = wine.train)

library(klaR)

wine.nB<- klaR::NaiveBayes(Type ~ ., data = wine.train, usekernel = FAL
SE)

predict.test.nB<- predict(wine.nB, newdata = wine.test[ , -1])$class

caret::confusionMatrix(data = predict.test.nB, reference = wine.test[ ,
 1])
```

5.3. APPLYING CLASSIFICATION TREE CLASSIFIER

```
train(Type ~., data = wine.train, method = "rpart", trControl = trContr
ol, tuneLength = 10, parms=list(split='gini'))

library(rpart)

wine.tree<- rpart(formula = Type ~ ., data = wine.train, cp = 0, parms=
list(split='gini'))

wine.tree
```

```
predict.test.tree<- predict(wine.tree, newdata = wine.test[ , -1], type
 = 'class')

caret::confusionMatrix(data = predict.test.tree, reference = wine.test[
 , 1])

library(rattle)

fancyRpartPlot(wine.tree)
```

5.4. APPLYING RANDOM FOREST CLASSIFIER

```
library(randomForest)

tuneGrid<- expand.grid(.mtry=c(2:13))

train(Type ~ ., data = wine.train, method = "rf", metric = 'Accuracy',
tuneGrid = tuneGrid, trControl = trControl)

set.seed(123)

wine.randomForest<- randomForest(Type ~ ., data = wine.train, mtry = 2,
 importance = TRUE)

wine.randomForest

varImpPlot(wine.randomForest, main = 'Importance Plot')

predict.test.randomForest<- predict(wine.randomForest, newdata = wine.t
est[ , -1])

caret::confusionMatrix(data = predict.test.randomForest, reference = wi
ne.test[ , 1])
```

6. COMPARISON OF PERFORMANCES OF VARIOUS CLASSIFIERS

6.1. TEST ACCURACY BY VALIDATION SET APPROACH

```
Classifier <- c('19 - Nearest Neighbour', 'Naive Bayes without Kernel',
 'Classification Tree', 'Random Forest')

Accuracy_1 <- c(1, 1, 0.9167, 1)

df_1 <- data.frame(Classifier, Accuracy_1)

colnames(df_1) <- c('Classifier', 'Accuracy')

df_1
```

6.2. TEST ACCURACY BY 5-FOLD CROSS VALIDATION

```
Accuracy_2 <- c(0.9719212, 0.964532, 0.9091133, 0.9790640)
df_2 <- data.frame(Classifier, Accuracy_2)
colnames(df_2) <- c('Classifier', 'Accuracy')
df_2
```