

## **DEPARTMENT OF INFORMATION TECHNOLOGY**

<b>Course Title: Machine Learning Algorithms (UITP204)</b>		
<b>Teaching Scheme:</b> <b>PR: 02 Hours/Week</b>	<b>Credit</b> <b>01</b>	<b>Examination Scheme:</b> <b>Cont. Ass: 25 Marks</b> <b>Ext.: --</b> <b>Total: 25 Marks</b>

### **Course Objectives:**

1. The objectives of the course is to introduce students to the basic machine learning algorithms.
2. To understand nature of the problem and apply suitable machine learning algorithm

### **Course Outcomes:**

After the completion of course the student should be able to

1. Understand complexity of Machine Learning algorithms and their limitations
2. Understand modern notions in machine learning and computing
3. Be capable of confidently applying common Machine Learning algorithms in practice and implementing their own
4. Be capable of performing experiments in Machine Learning using real-world data.
5. Implement different Machine Learning Models

## DEPARTMENT OF INFORMATION TECHNOLOGY

### LIST OF EXPERIMENTS

Sr. No.	List of Practical	CO's
1	Calculating Evaluation using Sklearn library (Regression Matrix): 1. Mean Absolute Error 2. Mean Squared Error 3. R squared Error	CO1, CO5
2	Write a python program to evaluate a Confusion Matrix from given Dataset	CO2, CO3, CO4
3	Write a python program to evaluate a Linear Regression on Diabetic Dataset	CO3, CO4, CO5
4	Write a python program to evaluate a Decision tree on iris Dataset	CO3, CO4, CO5
5	Write a python program to evaluate a Over fitting and Under fitting on custom dataset	CO3, CO4, CO5
6	Write a python program to evaluate a Applying Logistic Regression on iris Dataset	CO3, CO4, CO5
7	Write a python program to evaluate an Applying gaussian Naïve Bayes learning on iris Dataset	CO3, CO4, CO5
8	Write a python program to implement K mean clustering in python	CO3, CO4, CO5
9	Write a python program to evaluate a Apply PCA Algorithm on Iris Dataset	CO3, CO4, CO5
10	Introduction to Fundamental of Fuzzy Logic and Basic Operations (Through Virtual Lab)	CO3, CO4, CO5
11	Color Image Processing (Through Virtual Lab)	CO4, CO5
12	Mean and Covariance (Through Virtual Lab)	CO4, CO5

	Content Beyond Syllabus	
1	Write a python program to predicting if a customer with certain age and Salary will purchase a product or not using support vector machine.	CO3, CO4, CO5
2	Associative Rule Mining: Implementation of Apriori algorithm	CO3, CO4, CO5

# Assignment No. 1

**Aim:** Calculating Evaluation using Sklearn library (Regression Matrix):

1. Mean Absolute Error
2. Mean Squared Error
3. R squared Error

**Theory:**

## Regression Predictive Modeling

Predictive modeling is the problem of developing a model using historical data to make a prediction on new data where we do not have the answer.

Predictive modeling can be described as the mathematical problem of approximating a mapping function (f) from input variables (X) to output variables (y). This is called the problem of function approximation.

## Evaluating Regression Models

The skill or performance of a regression model must be reported as an error in those predictions.

This makes sense if you think about it. If you are predicting a numeric value like a height or a dollar amount, you don't want to know if the model predicted the value exactly (this might be intractably difficult in practice); instead, we want to know how close the predictions were to the expected values.

Error addresses exactly this and summarizes on average how close predictions were to their expected values.

There are three error metrics that are commonly used for evaluating and reporting the performance of a regression model; they are:

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE).
- Root Mean Squared Error (RMSE).

### 1. Mean Absolute Error or MAE

We know that an error basically is the absolute difference between the actual or true values and the values that are predicted. Absolute difference means that if the result has a negative sign, it is ignored.

Hence, **MAE = True values – Predicted values**

MAE takes the **average** of this error from every sample in a dataset and gives the output.

This can be implemented using **sklearn's mean\_absolute\_error** method:

```
from sklearn.metrics import mean_absolute_error
```

```
# predicting home prices in some area

predicted_home_prices = mycity_model.predict(X)

mean_absolute_error(y, predicted_home_prices)
```

he concepts of underfitting and overfitting can be pondered over, from here:

**Underfitting**: The scenario when a machine learning model almost exactly matches the training data but performs very poorly when it encounters new data or validation set.

**Overfitting**: The scenario when a machine learning model is unable to capture the important patterns and insights from the data, which results in the model performing poorly on training data itself.

## 2. Mean Squared Error or MSE

MSE is calculated by taking the average of the square of the difference between the original and predicted values of the data.

$$\frac{1}{N} \sum_{i=1}^n (\text{actual values} - \text{predicted values})^2$$

Hence, MSE =

Here N is the total number of observations/rows in the dataset. The sigma symbol denotes that the difference between actual and predicted values taken on every i value ranging from 1 to n.

This can be implemented using **sklearn**'s mean\_squared\_error method:

```
from sklearn.metrics import mean_squared_error

actual_values = [2, -0.6, 2, 8]

predicted_values = [2.5, 0.0, 2, 7]

mean_squared_error(actual_values, predicted_values)
```

### 3. Root Mean Squared Error or RMSE

RMSE is the standard deviation of the errors which occur when a prediction is made on a dataset. This is the same as MSE (Mean Squared Error) but the root of the value is considered while determining the accuracy of the model.

```
from sklearn.metrics import mean_squared_error

from math import sqrt

actual_values = [2, -0.6, 2, 8]

predicted_values = [2.5, 0.0, 2, 7]

a=mean_squared_error(actual_values, predicted_values)

# taking root of mean squared error

root_mean_squared_error = sqrt(a)

print(root_mean_squared_error)
```

## Assignment No. 2

**Aim:** Write a python program to evaluate a Confusion Matrix from given Dataset

**Theory:**

A Confusion matrix is an  $N \times N$  matrix used for evaluating the performance of a classification model, where  $N$  is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

For a binary classification problem, we would have a  $2 \times 2$  matrix as shown below with 4 values:

		ACTUAL VALUES	
		POSITIVE	NEGATIVE
PREDICTED VALUES	POSITIVE	TP	FP
	NEGATIVE	FN	TN

review	true_label	predicted_label
sample review 1	0	1
sample review 2	0	0
sample review 3	1	0
sample review 4	1	1
sample review 5	1	1
sample review 6	0	1
sample review 7	1	1
sample review 8	0	1
sample review 9	0	0
sample review 10	0	0
sample review 11	1	0
sample review 12	1	1
sample review 13	0	0
sample review 14	1	1
sample review 15	0	0
sample review 16	0	1
sample review 17	1	0
sample review 18	1	1
sample review 19	1	1
sample review 20	0	0

In this dataset, **0** means it's a negative review, and **1** means it's a positive review.

**TP** (True Positive)

**TN** (True Negative)

**FP** (False Positive)

**FN** (False Negative)

### **True Positive (TP)**

When the actual label (`true_label`) is positive (1) and your machine learning model also predicts that label as positive (1).

### True Negative (TN)

When the actual label (`true_label`) is negative (0) and your machine learning model also predicts that label as negative (0).

### False Positive (FP)

When the actual label (`true_label`) is negative (0) but your machine learning model predicts that label as positive (1).

### False Negative (FN)

When the actual label is positive (1) but your machine learning model predicts that label as negative (0).

<b>review</b>	<b>true_label</b>	<b>predicted_label</b>	<b>TP/TN/FP/FN</b>
sample review 1	0	1	FP
sample review 2	0	0	TN
sample review 3	1	0	FN
sample review 4	1	1	TP
sample review 5	1	1	TP
sample review 6	0	1	FP
sample review 7	1	1	TP
sample review 8	0	1	FP
sample review 9	0	0	TN
sample review 10	0	0	TN
sample review 11	1	0	FN
sample review 12	1	1	TP
sample review 13	0	0	TN
sample review 14	1	1	TP
sample review 15	0	0	TN
sample review 16	0	1	FP
sample review 17	1	0	FN
sample review 18	1	1	TP
sample review 19	1	1	TP
sample review 20	0	0	TN



Total value of TP : 7  
Total value of TN: 6  
Total value of FP: 4  
Total value of FN: 3

### Calculate Accuracy

Accuracy represents the number of correctly classified data instances over the total number of data instances.

The formula for calculating accuracy of your model:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

formula for calculating accuracy

If you place the values for these terms in the above formula and calculate the simple math, you will get the accuracy number. In our case, it is: **0.65**

Which means the accuracy is 65%.

### Calculate Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

The formula for calculating precision of your model:

$$\frac{TP}{TP + FP}$$

formula for calculating precision

Replace the values of these terms, and calculate the simple math, it would be **0.636**

## Calculate Recall

Recall is the ratio of correctly predicted positive observations to the all observations in actual class.

$$\frac{TP}{TP + FN}$$

Replace the values of these terms, and calculate the simple math, it would be **0.70**

## Calculate the F1 Score

F1 Score is the weighted average of Precision and Recall.

$$\frac{2TP}{2TP + FP + FN}$$

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

Replace the values of these terms, and calculate the simple math, it would be **0.667**

## Program:

```
#
confusion
matrix in
sklearn

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# actual values
actual = [1,0,0,1,0,0,1,0,0,1]
# predicted values
predicted = [1,0,0,1,0,0,0,1,0,0]
# confusion matrix
matrix = confusion_matrix(actual,predicted, labels=[1,0])
print('Confusion matrix : \n',matrix)
# outcome values order in sklearn
tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)
# classification report for precision, recall f1-score and accuracy
matrix = classification_report(actual,predicted,labels=[1,0])
```

```

print('Classification report : \n',matrix)

Confusion matrix :
[[2 2]
 [1 5]]
Outcome values :
2 2 1 5
Classification report :

```

	precision	recall	f1-score	support
1	0.67	0.50	0.57	4
0	0.71	0.83	0.77	6
micro avg	0.70	0.70	0.70	10
macro avg	0.69	0.67	0.67	10
weighted avg	0.70	0.70	0.69	10

Sklearn has two great functions: **confusion\_matrix()** and **classification\_report()**.

- Sklearn [confusion\\_matrix\(\)](#) returns the values of the Confusion matrix. The output is, however, slightly different from what we have studied so far. It takes the rows as Actual values and the columns as Predicted values. The rest of the concept remains the same.
- Sklearn [classification\\_report\(\)](#) outputs precision, recall and f1-score for each target class. In addition to this, it also has some extra values: **micro avg**, **macro avg**, and **weighted avg**

**Micro average** is the precision/recall/f1-score calculated for all the classes.

$$\text{Micro avg Precision} = \frac{TP1 + TP2}{TP1 + TP2 + FP1 + FP2}$$

**Macro average** is the average of precision/recall/f1-score.

$$\text{Macro avg Precision} = \frac{P1 + P2}{2}$$

**Weighted average** is just the weighted average of precision/recall/f1-score.

## Assignment No. 3

**Aim:** Write a python program to evaluate a Linear Regression on Diabetic Dataset

### Theory:

Linear regression is a statistical approach for modeling relationship between a dependent variable with a given set of independent variables.

### Simple Linear Regression

Simple linear regression is an approach for predicting a **response** using a **single feature**.

It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).

Let us consider a dataset where we have a value of response y for every feature x:

For generality, we define:

x as **feature vector**, i.e  $x = [x_1, x_2, \dots, x_n]$ ,

y as **response vector**, i.e  $y = [y_1, y_2, \dots, y_n]$

for **n** observations (in above example,  $n=10$ ).

A scatter plot of above dataset looks like:-

Now, the task is to find a **line which fits best** in above scatter plot so that we can predict the response for any new feature values. (i.e a value of x not present in dataset)

This line is called **regression line**.

The equation of regression line is represented as:

Here,

- $h(x_i)$  represents the **predicted response value** for ith observation.
- $b_0$  and  $b_1$  are regression coefficients and represent **y-intercept** and **slope** of regression line respectively.

To create our model, we must “learn” or estimate the values of regression coefficients  $b_0$  and  $b_1$ . And once we’ve estimated these coefficients, we can use the model to predict responses

### Get the Dataset:

- Find out the correlation between salary & years of experience

- For this we will create a model i.e. Simple Linear Regression Model which will tell us what is the best fitting line for this relationship

$$Y = b_0 + b_1 * X_1$$

What is Regression ? How does it work ?

Regression is a parametric technique used to predict continuous (dependent) variable given a set of independent variables. It is parametric in nature because it makes certain assumptions (discussed next) based on the data set. If the data set follows those assumptions, regression gives incredible results. Otherwise, it struggles to provide convincing accuracy.

Mathematically, regression uses a linear function to approximate (predict) the dependent variable given as:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

where,

Y	-	Dependent	variable
X	-	Independent	variable
$\beta_0$	-		Intercept
$\beta_1$	-		Slope
$\epsilon$ - Error			

$\beta_0$  and  $\beta_1$  are known as coefficients. This is the equation of simple linear regression. It's called 'linear' because there is just one independent variable (X) involved. In multiple regression, we have many independent variables (Xs). If you recall, the equation above is nothing but a line equation ( $y = mx + c$ ) we studied in schools. Let's understand what these parameters say:

Y - This is the variable we predict  
X - This is the variable we use to make a prediction  
 $\beta_0$  - This is the intercept term. It is the prediction value you get when  $X = 0$   
 $\beta_1$  - This is the slope term. It explains the change in Y when X changes by 1 unit.  $\epsilon$  - This represents the residual value, i.e. the difference between actual and predicted values.

Error is an inevitable part of the prediction-making process. No matter how powerful the algorithm we choose, there will always remain an ( $\epsilon$ ) irreducible error which reminds us that the "future is uncertain."

Yet, we humans have a unique ability to persevere, i.e. we know we can't completely eliminate the ( $\epsilon$ ) error term, but we can still try to reduce it to the lowest. Right? To do this, regression uses a technique known as **Ordinary Least Square(OLS)**.

So the next time when you say, I am using *linear /multiple regression*, you are actually referring to the *OLS technique*. Conceptually, OLS technique tries to reduce the sum of squared errors  $\sum [\text{Actual}(y) - \text{Predicted}(y')]^2$  by finding the best possible value of regression coefficients ( $\beta_0$ ,  $\beta_1$ , etc).

Is OLS the only technique regression can use? No! There are other techniques such as Generalized Least Square, Percentage Least Square, Total Least Squares, Least absolute deviation, and many more. Then, why OLS? Let's see.

1. It uses squared error which has nice mathematical properties, thereby making it easier to differentiate and compute gradient descent.
2. OLS is easy to analyze and computationally faster, i.e. it can be quickly applied to data sets having 1000s of features.
3. Interpretation of OLS is much easier than other regression techniques.

Let's understand OLS in detail using an example:

We are given a data set with 100 observations and 2 variables, namely Height and Weight. We need to predict weight(y) given height(x1). The OLS equation can be written as:

$$Y = \beta_0 + \beta_1(\text{Height}) + \epsilon$$

When using R, Python or any computing language, you don't need to know how these coefficients and errors are calculated. As a matter of fact, most people don't care. But you must know, and that's how you'll get close to becoming a master.

The formula to calculate these coefficients is easy. Let's say you are given the data, and you don't have access to any statistical tool for computation. Can you still make any prediction? Yes!

The most intuitive and closest approximation of Y is **mean of Y**, i.e. even in the worst case scenario our predictive model should at least give higher accuracy than mean prediction. The formula to calculate coefficients goes like this:

$$\beta_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} \text{ where } i = 1 \text{ to } n \text{ (no. of obs.)}$$

$$\beta_0 = \bar{y} - \beta_1(\bar{x})$$

Now you know  $\bar{y}$  plays a crucial role in determining regression coefficients and furthermore accuracy. In OLS, the error estimates can be divided into three parts:

**Residual Sum of Squares (RSS)** -  $\sum [\text{Actual}(y) - \text{Predicted}(y)]^2$

**Explained Sum of Squares (ESS)** -  $\sum [\text{Predicted}(y) - \text{Mean}(\bar{y})]^2$

**Total Sum of Squares (TSS)** -  $\sum [\text{Actual}(y) - \text{Mean}(\bar{y})]^2$

The most important use of these error terms is used in the calculation of the Coefficient of Determination ( $R^2$ ).

$$R^2 = 1 - (\text{SSE}/\text{TSS})$$

$R^2$  metric tells us the amount of variance explained by the independent variables in the model. In the upcoming section, we'll learn and see the importance of this coefficient and more metrics to compute the model's accuracy.

What are the assumptions made in regression ?

As we discussed above, regression is a parametric technique, so it makes assumptions. Let's look at the assumptions it makes:

1. There exists a **linear** and **additive** relationship between dependent (DV) and independent variables (IV). By linear, it means that the change in DV by 1 unit change in IV is constant. By additive, it refers to the effect of X on Y is independent of other variables.
2. There must be no correlation among independent variables. Presence of correlation in independent variables lead to **Multicollinearity**. If variables are correlated, it becomes extremely difficult for the model to determine the true effect of IVs on DV.
3. The error terms must possess constant variance. Absence of constant variance leads to **heteroskedestacity**.
4. The error terms must be uncorrelated i.e. error at  $t$  must not indicate the error at  $t+1$ . Presence of correlation in error terms is known as **Autocorrelation**. It drastically affects the regression coefficients and standard error values since they are based on the assumption of uncorrelated error terms.
5. The dependent variable and the error terms must possess a **normal distribution**.

Presence of these assumptions make regression quite restrictive. By **restrictive** I meant, the performance of a regression model is conditioned on fulfillment of these assumptions.

### Steps:

Prepare your data preprocessing template

Co-relate salaries with experience

Carry out prediction

Verify the values of prediction

Prediction on test set

### Program:

```
# Simple Linear Regression
```

```
# Importing the libraries
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import pandas as pd
```

```
# Importing the dataset
```

```
dataset = pd.read_csv('Salary_Data.csv')
```

```
X = dataset.iloc[:, :-1].values
```

```
y = dataset.iloc[:, 1].values
```

```
# Splitting the dataset into the Training set and Test set
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)
```

```
# Feature Scaling
```

```
"""from sklearn.preprocessing import StandardScaler
```

```
sc_X = StandardScaler()
```

```
X_train = sc_X.fit_transform(X_train)
```

```
X_test = sc_X.transform(X_test)
```

```
sc_y = StandardScaler()
```

```
y_train = sc_y.fit_transform(y_train)"""
```

```
# Fitting Simple Linear Regression to the Training set
```

```
from sklearn.linear_model import LinearRegression
```

```
regressor = LinearRegression()
```

```
regressor.fit(X_train, y_train)
```

```
# Predicting the Test set results
```

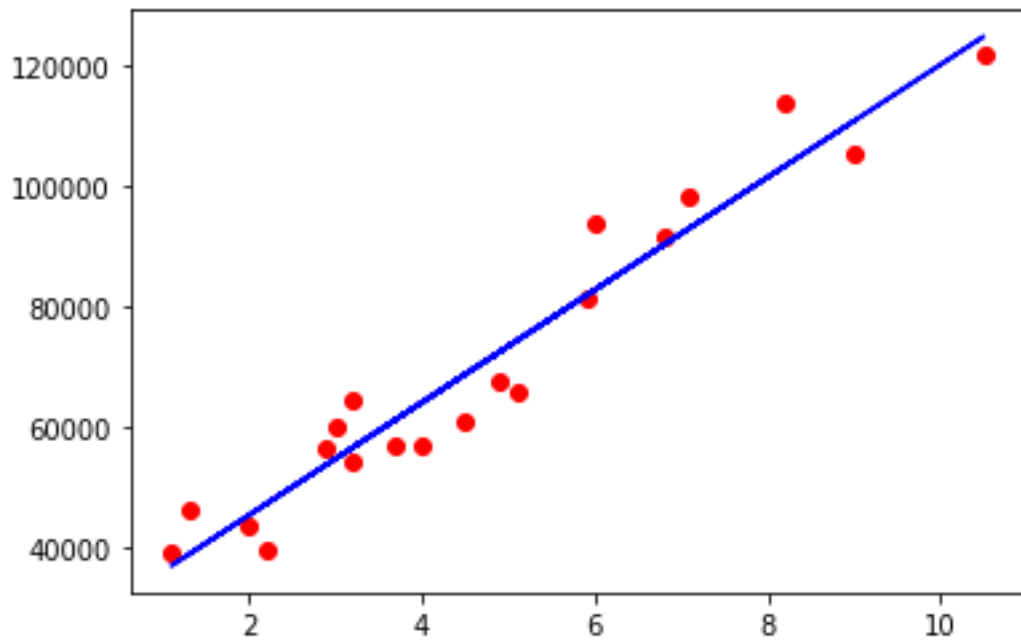
```
y_pred = regressor.predict(X_test)
```

```
# Visualising the Training set results
```

```
plt.scatter(X_train, y_train, color = 'red')
```

```
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
```





# Visualising the Test set results

```
plt.scatter(X_test, y_test, color = 'red')
```

```
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
```

```
plt.title('Salary vs Experience (Test set)')
```

```
plt.xlabel('Years of Experience')
```

```
plt.ylabel('Salary')
```

```
plt.show()
```

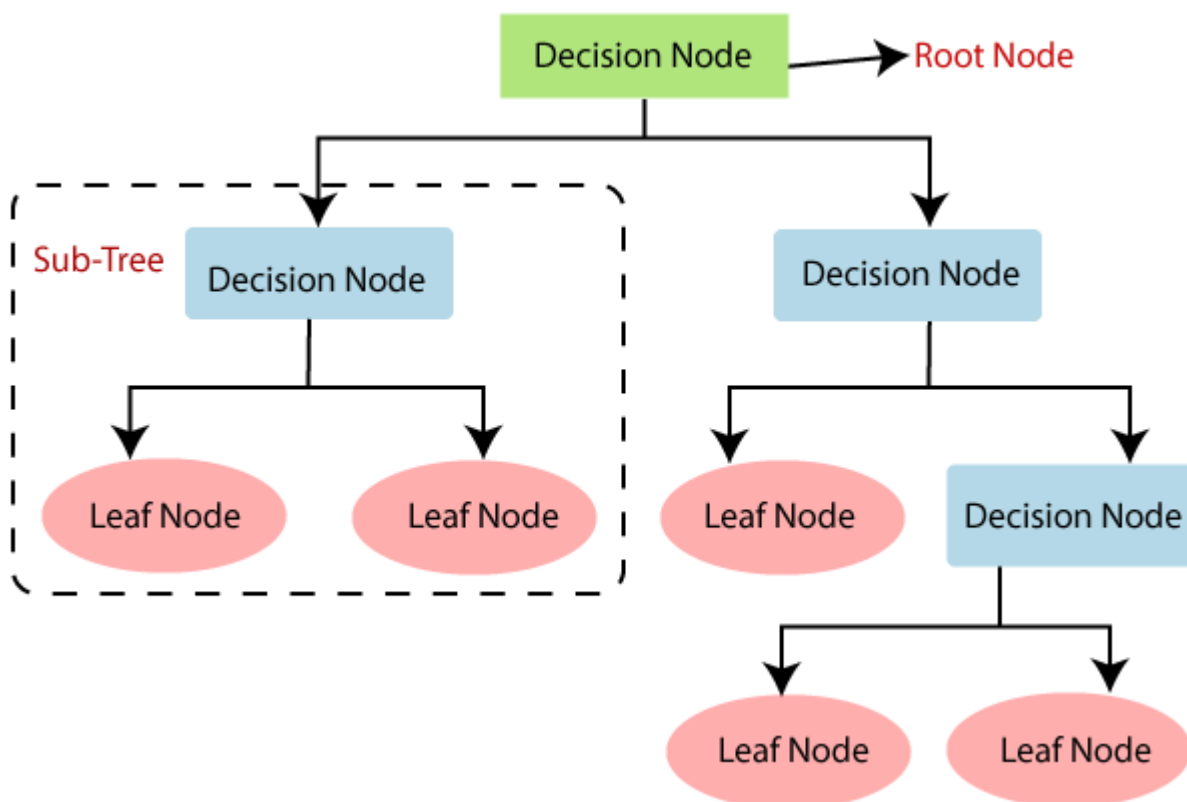


## Assignment No. 4

**Aim:** Write a python program to evaluate a Decision tree on iris Dataset

### Theory:

Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome**. In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node**. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure. In order to build a tree, we use the **CART algorithm**, which stands for **Classification and Regression Tree algorithm**. A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees. Below diagram explains the general structure of a decision tree:



## Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

## Decision Tree Terminologies

**Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

**Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.

**Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

**Branch/Sub Tree:** A tree formed by splitting the tree.

**Pruning:** Pruning is the process of removing the unwanted branches from the tree.

**Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

## How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM)**.
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.

- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

## Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM**. By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

### 1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

Information Gain= Entropy(S)- [(Weighted Avg) \*Entropy(each feature)]

**Entropy:** Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

Entropy(s)=  $-P(\text{yes})\log_2 P(\text{yes}) - P(\text{no})\log_2 P(\text{no})$

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

### 2. Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.

- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$

### Pruning: Getting an Optimal Decision tree

*Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.*

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

- **Cost Complexity Pruning**
- **Reduced Error Pruning.**

#### Program:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0
)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
#Fitting Decision Tree classifier to the training set
```

```

From sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
#Predicting the test set result
y_pred= classifier.predict(x_test)
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)

#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
    1, stop = x_set[:, 0].max() + 1, step =0.01),
    nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.
    shape),
    alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
fori, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()

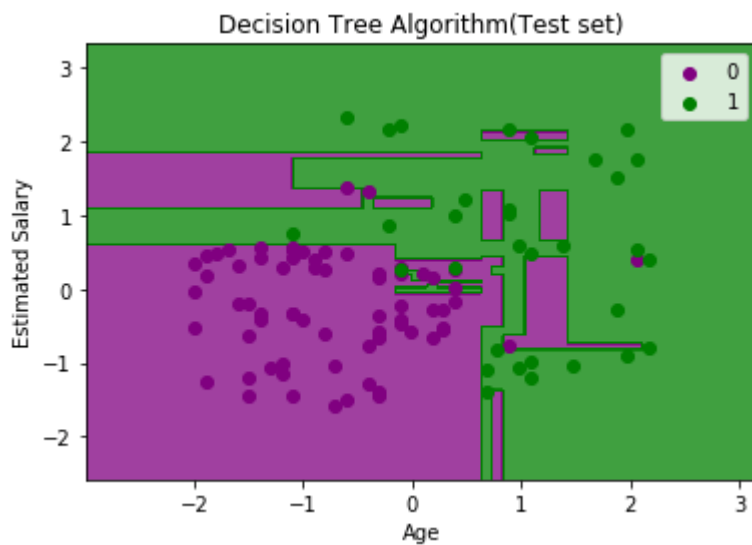
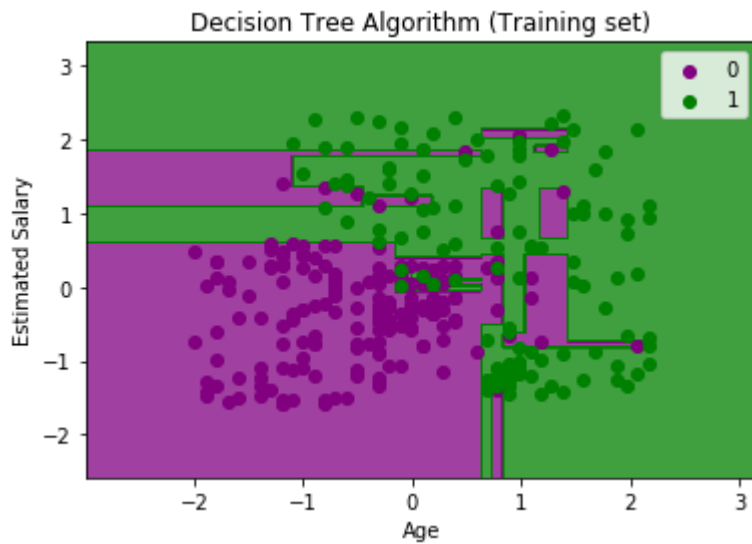
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
    1, stop = x_set[:, 0].max() + 1, step =0.01),
    nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.
    shape),
    alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())

```

```

for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
                c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()

```



## Assignment No. 5

**Aim:** Write a python program to evaluate a Over fitting and Under fitting on custom dataset

### Theory:

Overfitting and Underfitting are the two main problems that occur in machine learning and degrade the performance of the machine learning models.

The main goal of each machine learning model is to **generalize well**. Here **generalization** defines the ability of an ML model to provide a suitable output by adapting the given set of unknown input. It means after providing training on the dataset, it can produce reliable and accurate output. Hence, the underfitting and overfitting are the two terms that need to be checked for the performance of the model and whether the model is generalizing well or not.

Before understanding the overfitting and underfitting, let's understand some basic term that will help to understand this topic well:

- **Signal:** It refers to the true underlying pattern of the data that helps the machine learning model to learn from the data.
- **Noise:** Noise is unnecessary and irrelevant data that reduces the performance of the model.
- **Bias:** Bias is a prediction error that is introduced in the model due to oversimplifying the machine learning algorithms. Or it is the difference between the predicted values and the actual values.
- **Variance:** If the machine learning model performs well with the training dataset, but does not perform well with the test dataset, then variance occurs.

### Overfitting

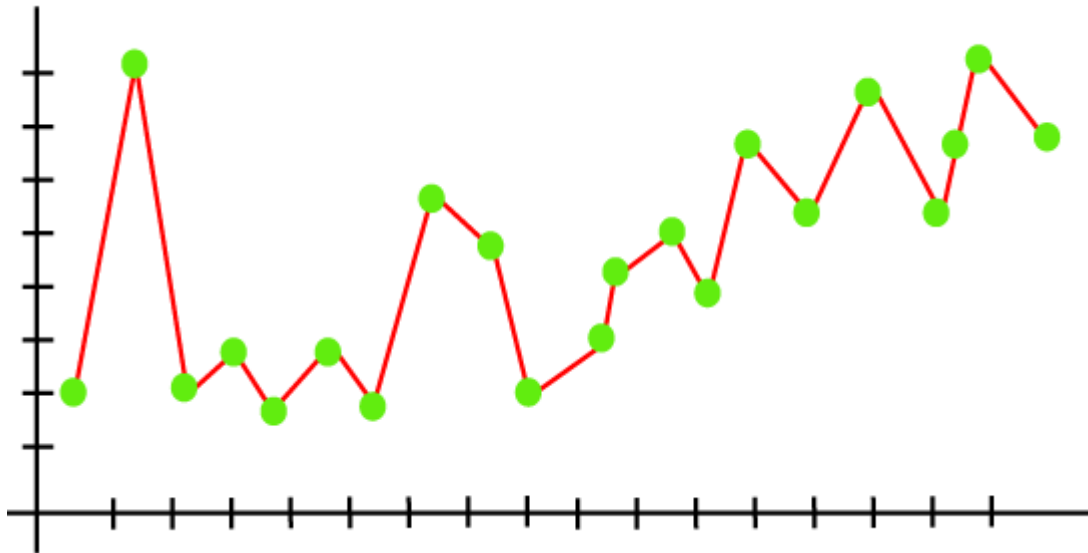
Overfitting occurs when our machine learning model tries to cover all the data points or more than the required data points present in the given dataset. Because of this, the model starts caching noise and inaccurate values present in the dataset, and all these factors reduce the efficiency and accuracy of the model. The overfitted model has **low bias** and **high variance**.

The chances of occurrence of overfitting increase as much we provide training to our model. It means the more we train our model, the more chances of occurring the overfitted model.

Overfitting is the main problem that occurs in supervised learning.

**Example:** The concept of the overfitting can be understood by the below graph of the linear regression output:





As we can see from the above graph, the model tries to cover all the data points present in the scatter plot. It may look efficient, but in reality, it is not so. Because the goal of the regression model to find the best fit line, but here we have not got any best fit, so, it will generate the prediction errors.

### How to avoid the Overfitting in Model

Both overfitting and underfitting cause the degraded performance of the machine learning model. But the main cause is overfitting, so there are some ways by which we can reduce the occurrence of overfitting in our model.

- **Cross-Validation**
- **Training with more data**
- **Removing features**
- **Early stopping the training**
- **Regularization**
- **Ensembling**

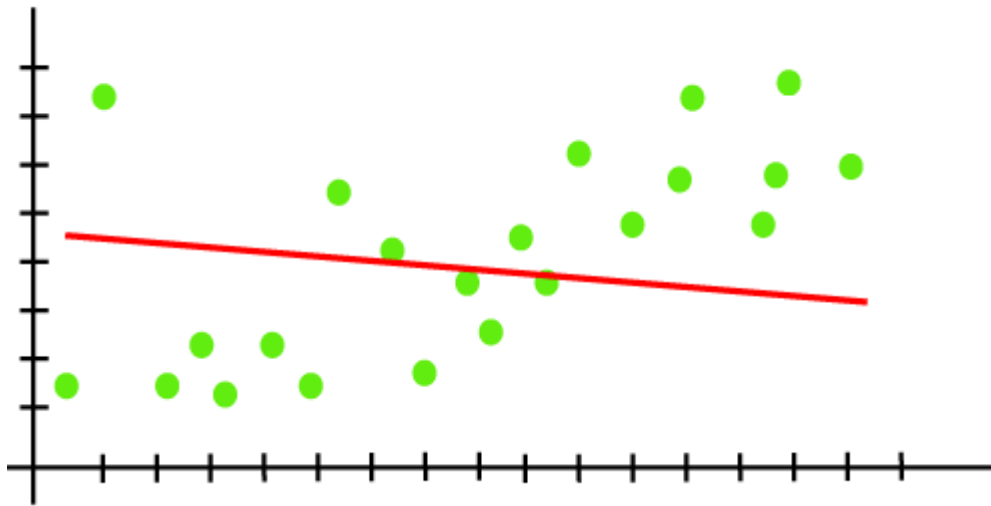
### Underfitting

Underfitting occurs when our machine learning model is not able to capture the underlying trend of the data. To avoid the overfitting in the model, the fed of training data can be stopped at an early stage, due to which the model may not learn enough from the training data. As a result, it may fail to find the best fit of the dominant trend in the data.

In the case of underfitting, the model is not able to learn enough from the training data, and hence it reduces the accuracy and produces unreliable predictions.

An underfitted model has high bias and low variance.

**Example:** We can understand the underfitting using below output of the linear regression model:



As we can see from the above diagram, the model is unable to capture the data points present in the plot.

#### How to avoid underfitting:

- By increasing the training time of the model.
- By increasing the number of features.

#### Goodness of Fit

The "Goodness of fit" term is taken from the statistics, and the goal of the machine learning models to achieve the goodness of fit. In statistics modeling, *it defines how closely the result or predicted values match the true values of the dataset.*

The model with a good fit is between the underfitted and overfitted model, and ideally, it makes predictions with 0 errors, but in practice, it is difficult to achieve it.

As when we train our model for a time, the errors in the training data go down, and the same happens with test data. But if we train the model for a long duration, then the performance of the model may decrease due to the overfitting, as the model also learn the noise present in the dataset. The errors in the test dataset start increasing, *so the point, just before the raising of errors, is the good point, and we can stop here for achieving a good model.*

There are two other methods by which we can get a good point for our model, which are the **resampling method** to estimate model accuracy and **validation dataset**.

#### Program:

```

from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error
import numpy as np
from sklearn.model_selection import validation_curve
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
np.random.seed(0)
plt.style.use('ggplot')

```

```

iris = load_iris()
X, y = iris.data, iris.target

```

```

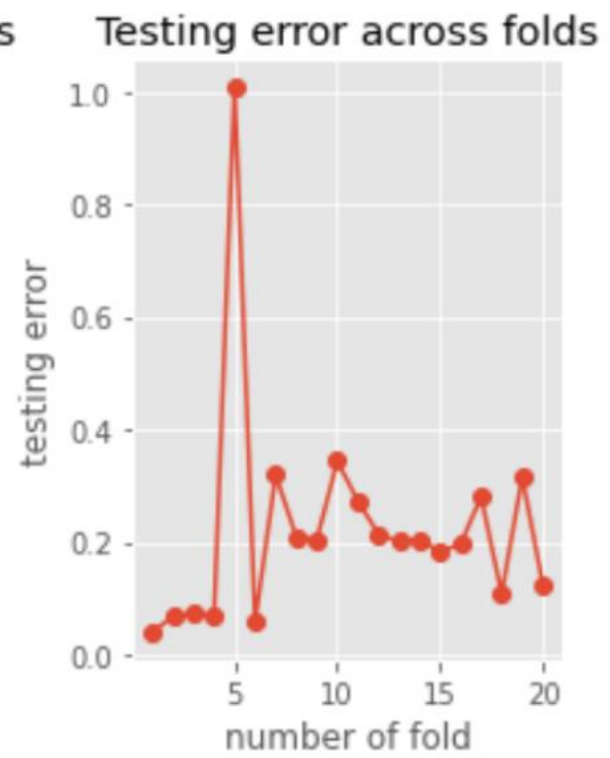
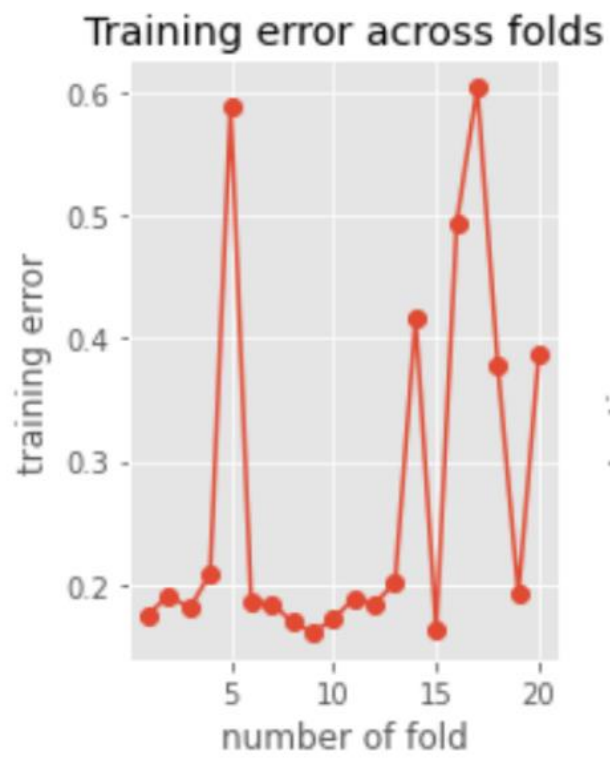
kf = KFold(n_splits=20)
list_training_error = []
list_testing_error = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    model = MLPRegressor() # We have used a multilayer perceptron (MLP) regressor model. A MLP is a
    model.fit(X_train, y_train)
    y_train_data_pred = model.predict(X_train)
    y_test_data_pred = model.predict(X_test)
    fold_training_error = mean_absolute_error(y_train, y_train_data_pred)
    fold_testing_error = mean_absolute_error(y_test, y_test_data_pred)
    list_training_error.append(fold_training_error)
    list_testing_error.append(fold_testing_error)

```

```

plt.subplot(1,2,1)
plt.plot(range(1, kf.get_n_splits() + 1), np.array(list_training_error).ravel(), 'o-')
plt.xlabel('number of fold')
plt.ylabel('training error')
plt.title('Training error across folds')
plt.tight_layout()
plt.subplot(1,2,2)
plt.plot(range(1, kf.get_n_splits() + 1), np.array(list_testing_error).ravel(), 'o-')
plt.xlabel('number of fold')
plt.ylabel('testing error')
plt.title('Testing error across folds')
plt.tight_layout()
plt.show()

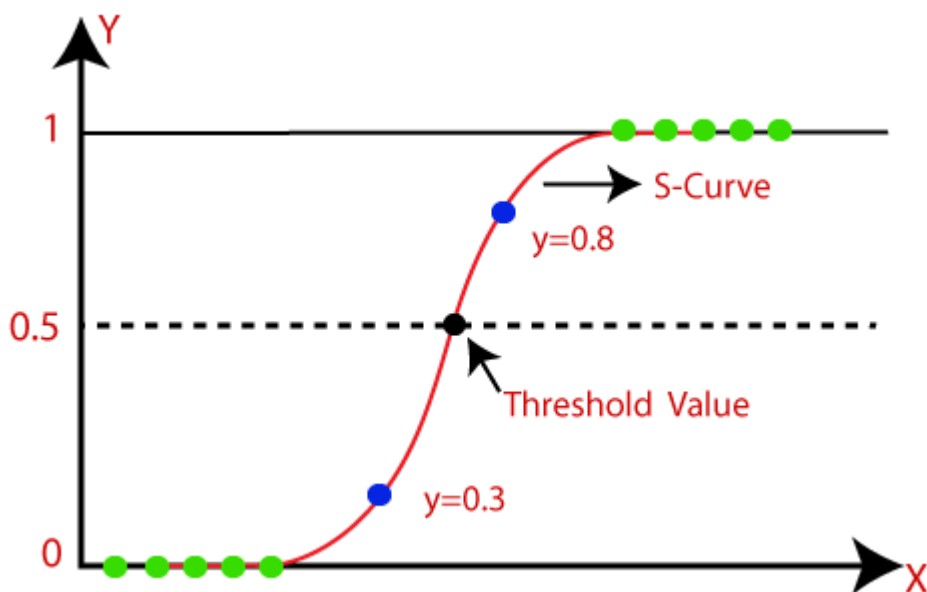
```



## Assignment No. 6

**Aim:** Write a python program to evaluate a Applying Logistic Regression on iris Dataset  
**Theory:**

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**. Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**. In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1). The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc. Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets. Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



### Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.

- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

### Assumptions for Logistic Regression:

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.

### Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- We know the equation of the straight line can be written as:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}; 0 \text{ for } y=0, \text{ and infinity for } y=1$$

- But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$\log \left[ \frac{y}{1-y} \right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

### Type of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

## Program:

```
#Data Pre-processing Step
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)

#Fitting Logistic Regression to the training set
from sklearn.linear_model import LogisticRegression
classifier= LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)

#Predicting the test set result
y_pred= classifier.predict(x_test)

#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix()
```

```

#Visualizing the training set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
1, stop = x_set[:, 0].max() + 1, step = 0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple', 'green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
               c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Logistic Regression (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()

```

```

#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
1, stop = x_set[:, 0].max() + 1, step = 0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple', 'green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
               c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Logistic Regression (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()

```



## Assignment No. 7

**Aim:** Write a python program to evaluate an Applying gaussian Naïve Bayes learning on iris Dataset

### Theory:

Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems. It is mainly used in *text classification* that includes a high-dimensional training dataset. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.** Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**. The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

**Naïve:** It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.

**Bayes:** It is called Bayes because it depends on the principle of [Bayes' Theorem](#).

### Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

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

Where,

**P(A|B) is Posterior probability:** Probability of hypothesis A on the observed event B.

**P(B|A) is Likelihood probability:** Probability of the evidence given that the probability of a hypothesis is true.

**P(A) is Prior Probability:** Probability of hypothesis before observing the evidence.

**P(B) is Marginal Probability:** Probability of Evidence.

## Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.
4. **Problem:** If the weather is sunny, then the Player should play or not?
5. **Solution:** To solve this, first consider the below dataset:

	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

## Frequency table for the Weather Conditions:

Weather	Yes	No
---------	-----	----

Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

#### Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	$5/14 = 0.35$
Rainy	2	2	$4/14 = 0.29$
Sunny	2	3	$5/14 = 0.35$
All	$4/14 = 0.29$	$10/14 = 0.71$	

#### Applying Bayes'theorem:

$$P(\text{Yes}|\text{Sunny}) = P(\text{Sunny}|\text{Yes}) * P(\text{Yes}) / P(\text{Sunny})$$

$$P(\text{Sunny}|\text{Yes}) = 3/10 = 0.3$$

$$P(\text{Sunny}) = 0.35$$

$$P(\text{Yes}) = 0.71$$

$$\text{So } P(\text{Yes}|\text{Sunny}) = 0.3 * 0.71 / 0.35 = \mathbf{0.60}$$

$$P(\text{No}|\text{Sunny}) = P(\text{Sunny}|\text{No}) * P(\text{No}) / P(\text{Sunny})$$

$$P(\text{Sunny}|\text{NO}) = 2/4 = 0.5$$

$$P(\text{No}) = 0.29$$

$$P(\text{Sunny}) = 0.35$$

$$\text{So } P(\text{No}|\text{Sunny}) = 0.5 * 0.29 / 0.35 = \mathbf{0.41}$$

So as we can see from the above calculation that  $P(\text{Yes}|\text{Sunny}) > P(\text{No}|\text{Sunny})$

**Hence on a Sunny day, Player can play the game.**

#### Types of Naïve Bayes Model:

There are three types of Naive Bayes Model, which are given below:

- **Gaussian:** The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.
- **Multinomial:** The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc. The classifier uses the frequency of words for the predictors.
- **Bernoulli:** The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

### Program:

Importing the libraries

```
import numpy as nm
```

```
import matplotlib.pyplot as mtp
```

```
import pandas as pd
```

**# Importing the dataset**

```
dataset = pd.read_csv('user_data.csv')
```

```
x = dataset.iloc[:, [2, 3]].values
```

```
y = dataset.iloc[:, 4].values
```

**# Splitting the dataset into the Training set and Test set**

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)
```

**# Feature Scaling**

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
x_train = sc.fit_transform(x_train)
```

```
x_test = sc.transform(x_test)
```

**# Fitting Naive Bayes to the Training set**

```
from sklearn.naive_bayes import GaussianNB
```

```
classifier = GaussianNB()
```

```
classifier.fit(x_train, y_train)
```

```
# Predicting the Test set results
```

```
y_pred = classifier.predict(x_test)
```

```
# Making the Confusion Matrix
```

```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(y_test, y_pred)
```

```
# Visualising the Training set results
```

```
from matplotlib.colors import ListedColormap
```

```
x_set, y_set = x_train, y_train
```

```
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -  
1, stop = x_set[:, 0].max() + 1, step = 0.01),
```

```
nm.arange(start = x_set[:, 1].min() -
```

```
1, stop = x_set[:, 1].max() + 1, step = 0.01))
```

```
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.  
shape),
```

```
alpha = 0.75, cmap = ListedColormap(('purple', 'green')))
```

```
mtp.xlim(X1.min(), X1.max())
```

```
mtp.ylim(X2.min(), X2.max())
```

```
for i, j in enumerate(nm.unique(y_set)):
```

```
mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
```

```
c = ListedColormap(('purple', 'green'))(i), label = j)
```

```
mtp.title('Naive Bayes (Training set)')
```

```
mtp.xlabel('Age')
```

```
mtp.ylabel('Estimated Salary')
```

```
mtp.legend()
```

```
mtp.show()
```

```
# Visualising the Test set results
```

```
from matplotlib.colors import ListedColormap
```

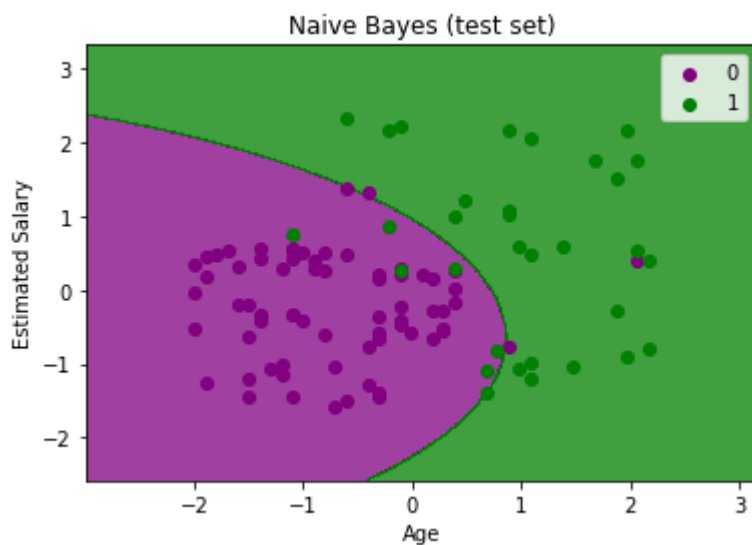
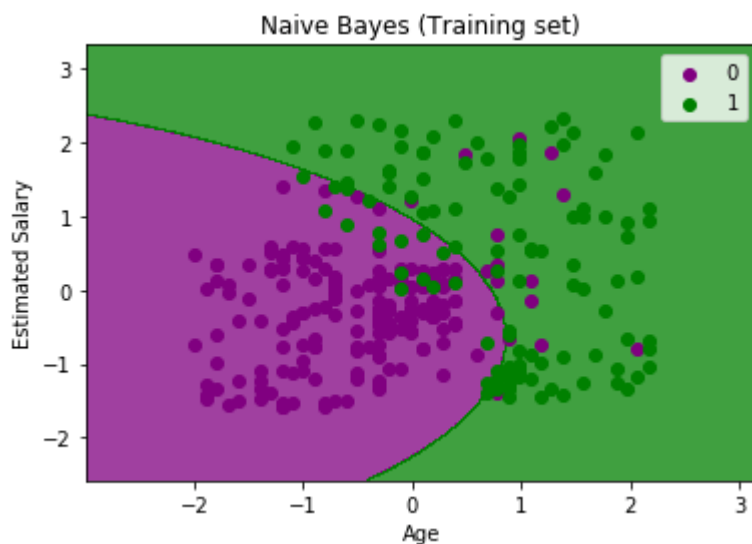
```
x_set, y_set = x_test, y_test
```

```
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -  
1, stop = x_set[:, 0].max() + 1, step = 0.01),
```

```

nm.arange(start = x_set[:, 1].min() -
1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.
shape),
alpha = 0.75, cmap = ListedColormap(('purple', 'green')))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()

```



## Assignment No. 8

**Aim:** Write a python program to implement K mean clustering in python

### Theory:

K-Means clustering is an unsupervised learning algorithm that, as the name hints, finds a fixed number ( $k$ ) of clusters in a set of data. A *cluster* is a group of data points that are grouped together due to similarities in their features. When using a K-Means algorithm, a cluster is defined by a *centroid*, which is a point (either imaginary or real) at the center of a cluster. Every point in a data set is part of the cluster whose centroid is most closely located. To put it simply, K-Means finds  $k$  number of centroids, and then assigns all data points to the closest cluster, with the aim of keeping the centroids small.

### The Algorithm

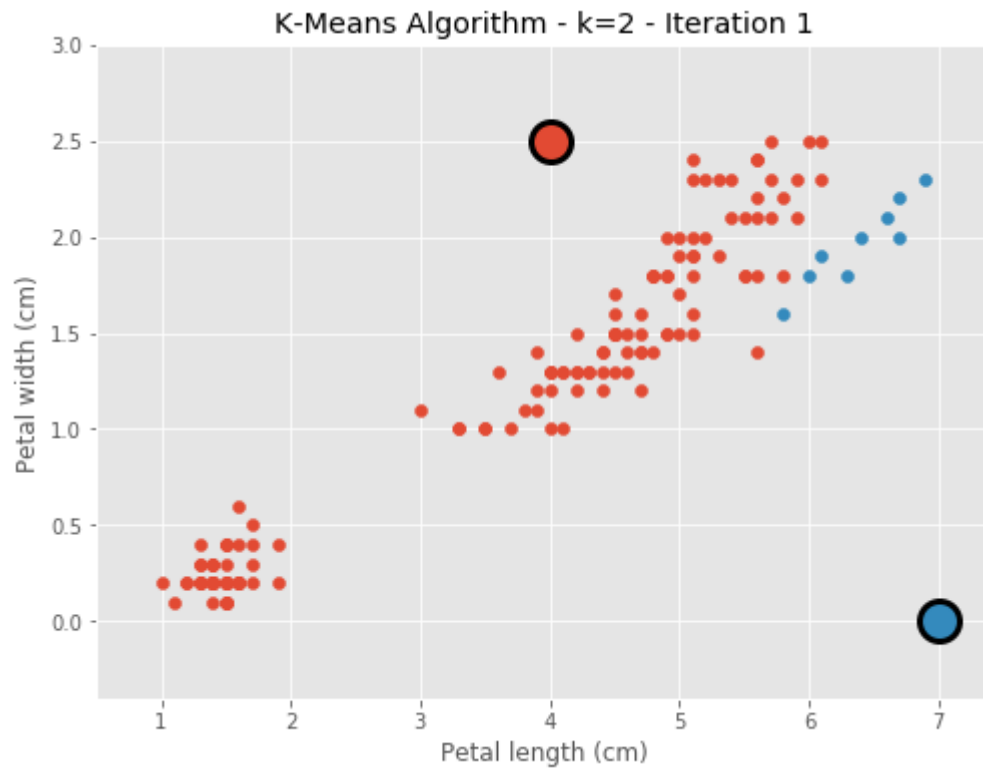
K-Means starts by randomly defining  $k$  centroids. From there, it works in iterative (repetitive) steps to perform two tasks:

1. Assign each data point to the closest corresponding centroid, using the standard Euclidean distance. In layman's terms: the straight-line distance between the data point and the centroid.
2. For each centroid, calculate the mean of the values of all the points belonging to it. The mean value becomes the new value of the centroid.

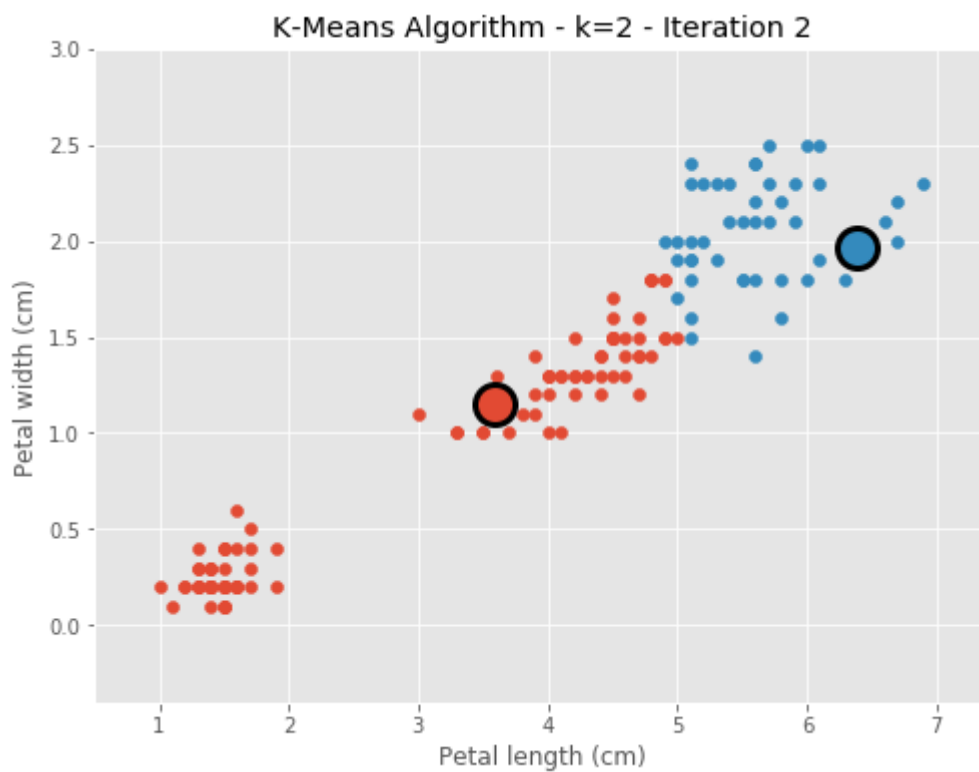
Once step 2 is complete, all of the centroids have new values that correspond to the means of all of their corresponding points. These new points are put through steps one and two producing yet another set of centroid values. This process is repeated over and over until there is no change in the centroid values, meaning that they have been accurately grouped. Or, the process can be stopped when a previously determined maximum number of steps has been met.

### Applying the K-Means Algorithm

**Iteration 1:** First, we create two randomly generated centroids and assign each data point to the cluster of the closest centroid. In this case, because we are using two centroids, our  $k$  value is 2.

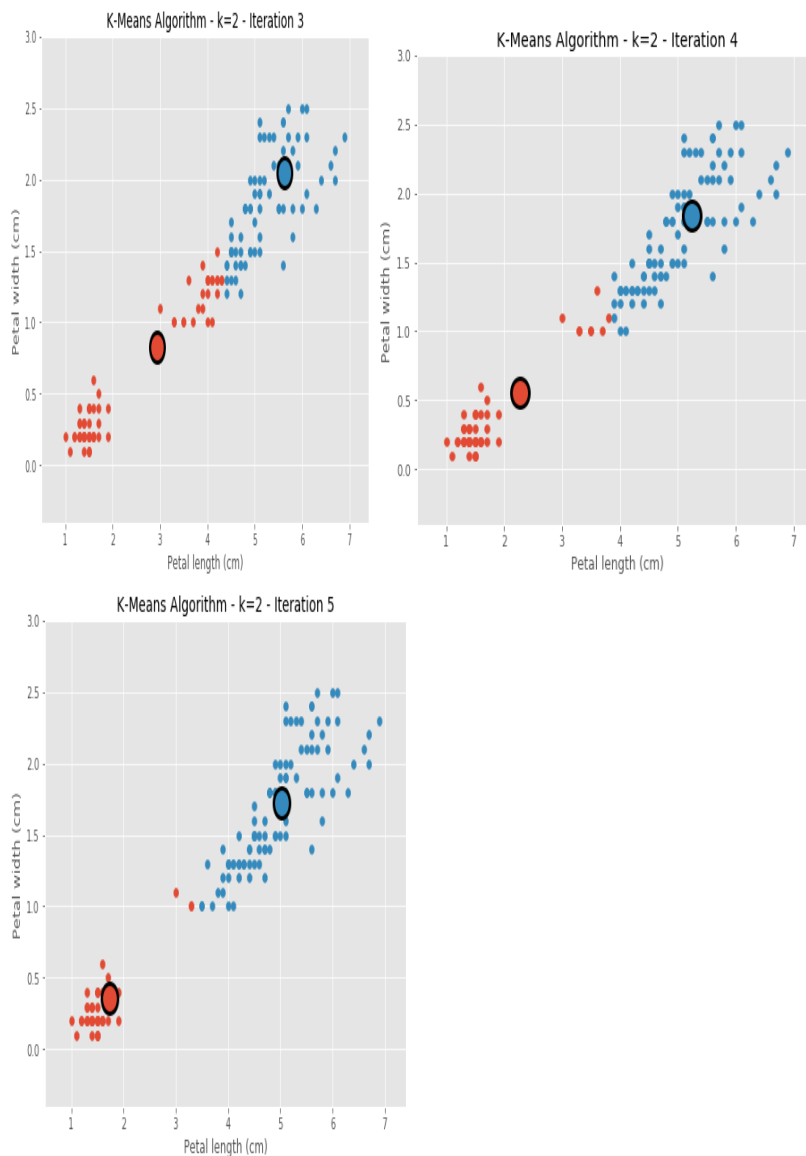


**Iteration 2:** As you can see above, the centroids are not evenly distributed. In the second iteration of the algorithm, the average values of each of the two clusters are found and become the new centroid values.





**Iterations 3-5:** We repeat the process until there is no further change in the value of the centroids.



Finally, after iteration 5, there is no further change in the clusters.

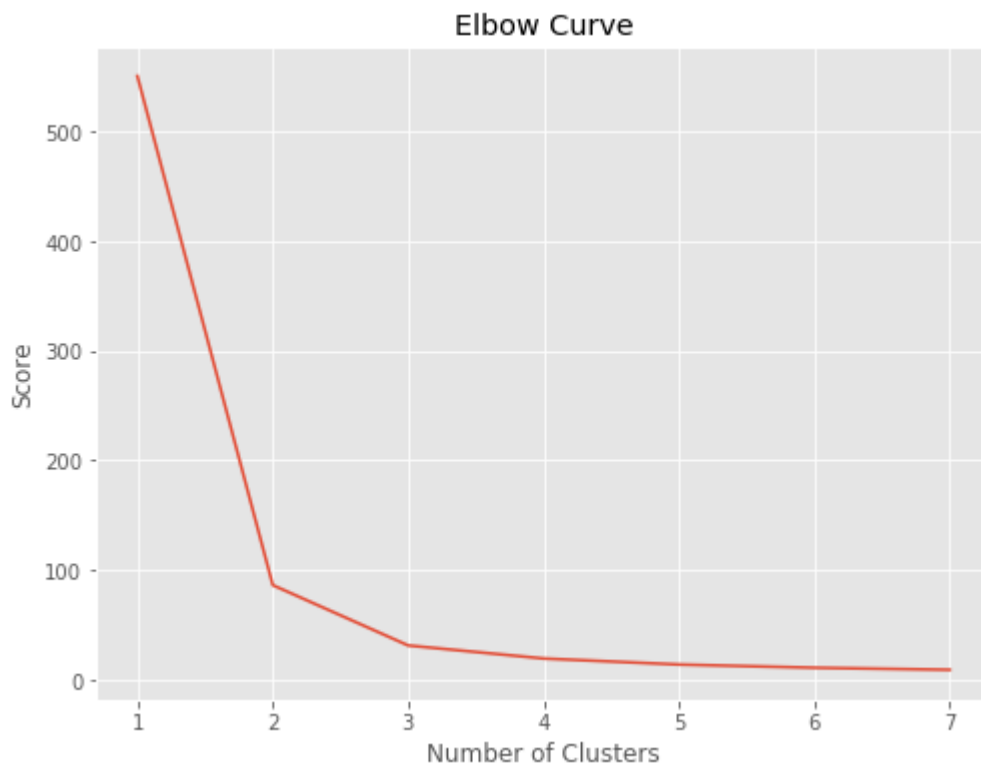
## Choosing K

The algorithm explained above finds clusters for the number  $k$  that we chose. So, how do we decide on that number?

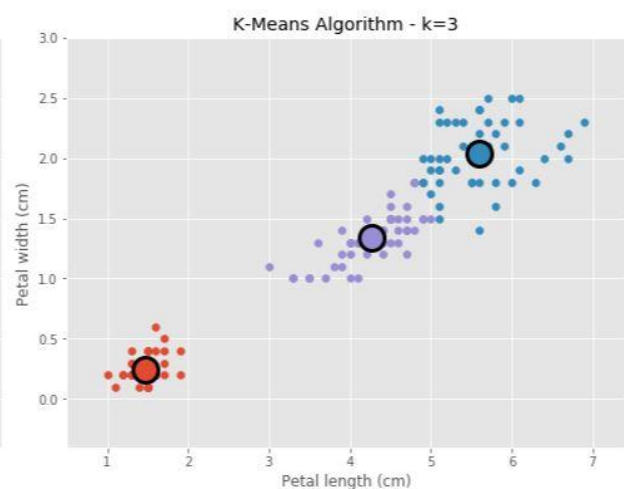
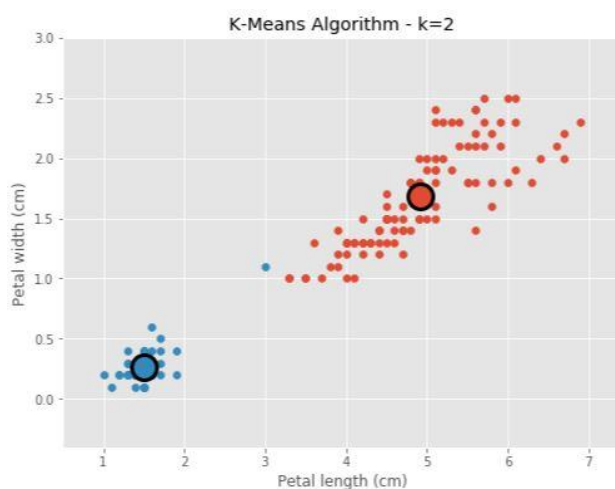
To find the best  $k$  we need to measure the quality of the clusters. The most traditional and straightforward method is to start with a random  $k$ , create centroids, and run the algorithm as we explained above. A sum is given based on the distances between each point and its closest centroid. As an increase in clusters correlates with smaller groupings and distances, this sum

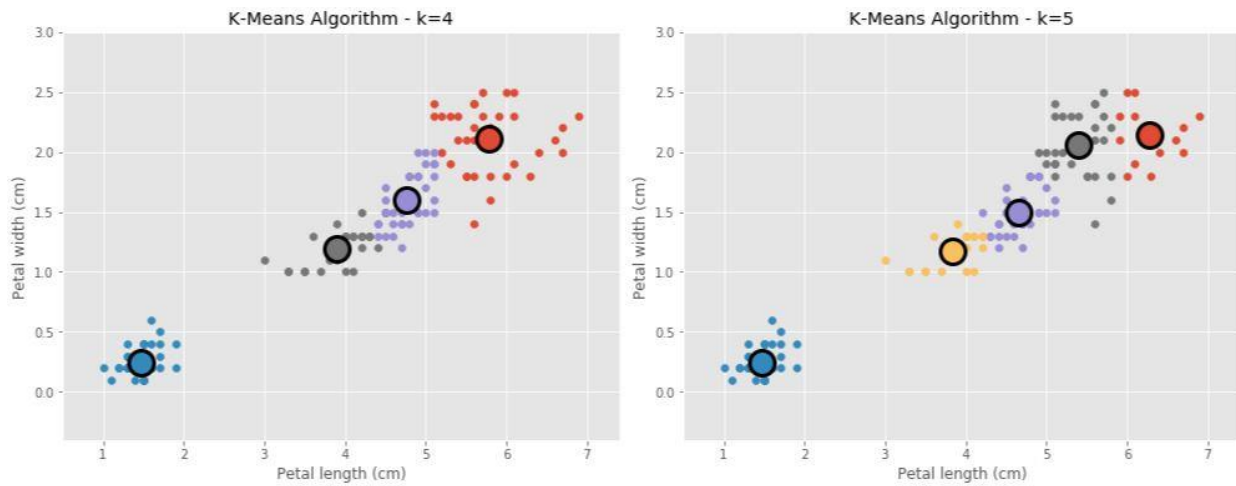
will always decrease when  $k$  increases; as an extreme example, if we choose a  $k$  value that is equal to the number of data points that we have, the sum will be zero.

The goal with this process is to find the point at which increasing  $k$  will cause a very small decrease in the error sum, while decreasing  $k$  will sharply increase the error sum. This sweet spot is called the “elbow point.” In the image below, it is clear that the “elbow” point is at  $k$ -3.



Now, let's take a look at the visual differences between using two, three, four, or five clusters.





The different graphs also show that three is the most appropriate  $k$  value, which makes sense when taking into account that the data contains three types of iris flowers.

The K-Means algorithm is a great example of a simple, yet powerful algorithm. For example, it can be used to cluster phishing attacks with the objective of discovering common patterns and even discovering new phishing kits. It can also be used to understand fraudulent patterns in financial transactions.

## Summary

Advantages of K-Means:

- Widely used method for cluster analysis
- Easy to understand
- Trains quickly

Disadvantages of K-Means:

- Euclidean distance is not ideal in many applications
- Performance is (generally) not competitive with the best clustering methods
- Small variations in the data can result in a completely different clusters (high variance)
- Clusters are assumed to have a spherical shape and be evenly sized

**Program:**

```
# K MEANS CLUSTERING
```

```

# Importing the Libraries

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Import Mall Dataset
dataset = pd.read_csv('3_Clustering/Mall_Customers.csv')
# Clients that subscribe to Membership card
# Maintains the Purchase history
# Score is Dependent on INCOME,
# No. times in week the show up in Mall, total expense in same mall
# YOU ARE!!
# TO Segment Clients into Different Groups based on Income & Score
# CLUSTERING PROBLEM
X = dataset.iloc[:, [3, 4]].values
# We have no Idea to look for
# We don't know the Optimal no. of Clusters

# USE ELBOW METHOD
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    # Fit values into KM
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,
random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title("ELBOW METHOD")
plt.xlabel("No. of Clus")
plt.ylabel("WCSS")
plt.plot()

# Applying K-means to Mall

```

```

kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=10,
random_state=0)
y_kmeans = kmeans.fit_predict(X)

# Visualising the Clusters
plt.scatter(X[y_kmeans==0, 0], X[y_kmeans==0, 1], s= 100, c = 'red', label='Cluster1')
plt.scatter(X[y_kmeans==1, 0], X[y_kmeans==0, 1], s= 100, c = 'blue', label='Cluster2')
plt.scatter(X[y_kmeans==2, 0], X[y_kmeans==0, 1], s= 100, c = 'yellow', label='Cluster3')
plt.scatter(X[y_kmeans==3, 0], X[y_kmeans==0, 1], s= 100, c = 'green', label='Cluster4')
plt.scatter(X[y_kmeans==4, 0], X[y_kmeans==0, 1], s= 100, c = 'cyan', label='Cluster5')
plt.scatter(kmeans.cluster_centers_[0, 0], kmeans.cluster_centers_[0, 1], s=300, c = 'red',
label='Cluster1')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()

```

## Assignment No. 9

**Aim:** Write a python program to evaluate a Apply PCA Algorithm on Iris Dataset

### Theory:

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the **Principal Components**. It is one of the popular tools that is used for exploratory data analysis and predictive modeling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.

PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are **image processing, movie recommendation system, optimizing the power allocation in various communication channels**. It is a feature extraction technique, so it contains the important variables and drops the least important variable.

The PCA algorithm is based on some mathematical concepts such as:

- Variance and Covariance
- Eigenvalues and Eigen factors

Some common terms used in PCA algorithm:

- **Dimensionality:** It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset.
- **Correlation:** It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
- **Orthogonal:** It defines that variables are not correlated to each other, and hence the correlation between the pair of variables is zero.
- **Eigenvectors:** If there is a square matrix  $M$ , and a non-zero vector  $v$  is given. Then  $v$  will be eigenvector if  $Av$  is the scalar multiple of  $v$ .
- **Covariance Matrix:** A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

## Principal Components in PCA

As described above, the transformed new features or the output of PCA are the Principal Components. The number of these PCs are either equal to or less than the original features present in the dataset. Some properties of these principal components are given below:

- The principal component must be the linear combination of the original features.
- These components are orthogonal, i.e., the correlation between a pair of variables is zero.
- The importance of each component decreases when going to 1 to n, it means the 1 PC has the most importance, and n PC will have the least importance.

### Steps for PCA algorithm

- 1. Getting the dataset**  
Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.
- 2. Representing data into a structure**  
Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.
- 3. Standardizing the data**  
In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance.  
If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.
- 4. Calculating the Covariance of Z**  
To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.
- 5. Calculating the Eigen Values and Eigen Vectors**  
Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.

#### 6. **Sorting the Eigen Vectors**

In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as  $P^*$ .

#### 7. **Calculating the new features Or Principal Components**

Here we will calculate the new features. To do this, we will multiply the  $P^*$  matrix to the Z. In the resultant matrix  $Z^*$ , each observation is the linear combination of original features. Each column of the  $Z^*$  matrix is independent of each other.

#### 8. **Remove less or unimportant features from the new dataset.**

The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

### **Applications of Principal Component Analysis**

- PCA is mainly used as the dimensionality reduction technique in various AI applications such as **computer vision, image compression, etc.**
- It can also be used for finding hidden patterns if data has high dimensions. Some fields where PCA is used are Finance, data mining, Psychology, etc.



## Assignment No. 10

**Aim:** Introduction to Fundamental of Fuzzy Logic and Basic Operations (Through Virtual Lab)

### Theory:

In recent times, engineers have very well accepted soft computing tools such as Fuzzy Computing, Neuro-Computing, Evolutionary Computing, Probabilistic Computing, and Immunological Computing etc. for carrying out various numerical simulation studies. In last two decades, these tools independently as well as in hybrid forms has been successfully applied to varieties of problems.

The main objective of the proposed virtual lab is to introduce students about the latest Computational Intelligence Tools ( also known as soft computing tools). The training of these tools will be useful to develop rigorous applications in the engineering domain.

The experiments cover following broad areas:

#### Fuzzy Logic

1. Fuzzy Logic Fundamentals and Basic Operations
2. Fuzzy Inference System(FIS)
3. Fuzzy Weighted Average
4. Fuzzy Control

#### Artificial Neural Networks

5. Neural Networks and Perceptron
6. Multilayer Perceptron
7. Radial Basis Function
8. Probabilistic Neural Networks

#### Evolutionary Algorithms (EA)

9. Introduction to EA
10. Binary and Real Coded genetic Algorithms
11. Genetic Expression Programming

#### Probabilistic Reasoning

12. Introduction to Probabilistic Reasoning and Baayesian Networks Application



# Assignment No. 11

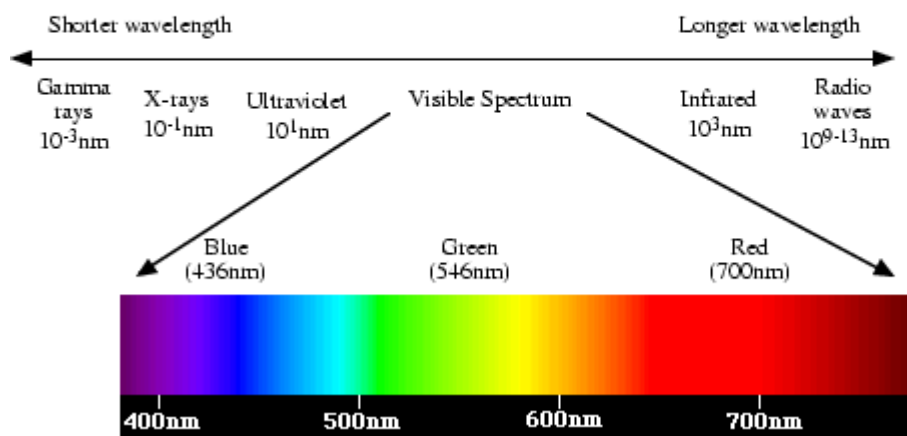
**Aim:** Color Image Processing (Through Virtual Lab)

**Theory:**

Colour Image Processing

The human visual system can distinguish hundreds of thousands of different colour shades and intensities, but only around 100 shades of grey. Therefore, in an image, a great deal of extra information may be contained in the colour, and this extra information can then be used to simplify image analysis, e.g. object identification and extraction based on colour.

Three independent quantities are used to describe any particular colour. The *hue* is determined by the dominant wavelength. Visible colours occur between about 400nm (violet) and 700nm (red) on the electromagnetic spectrum, as shown in figure [1](#).

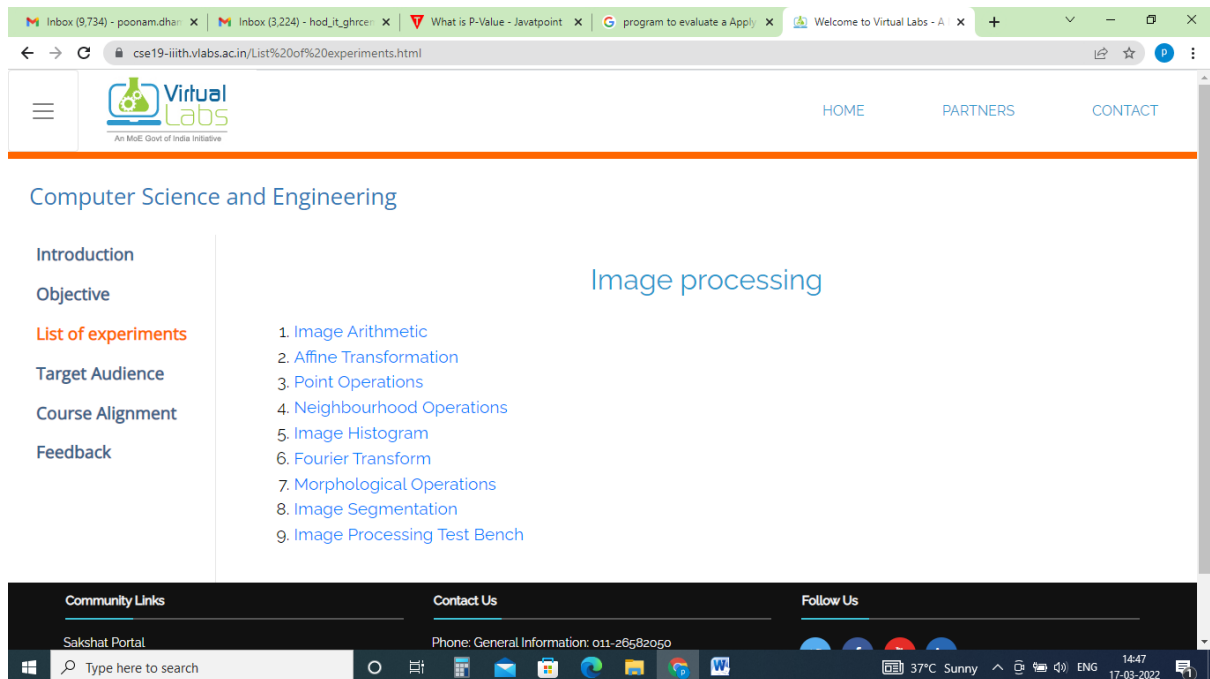


The *saturation* is determined by the excitation purity, and depends on the amount of white light mixed with the hue. A pure hue is fully saturated, i.e. no white light mixed in. Hue and saturation together determine the *chromaticity* for a given colour. Finally, the *intensity* is determined by the actual amount of light, with more light corresponding to more intense colours.

*Achromatic* light has no colour - its only attribute is quantity or intensity. Greylevel is a measure of intensity. The *intensity* is determined by the energy, and is therefore a physical quantity. On the other hand, *brightness* or *luminance* is determined by the perception of the colour, and is therefore psychological. Given equally intense blue and green, the blue is perceived as much darker than the green. Note also that our perception of intensity is nonlinear, with changes of normalised intensity from 0.1 to 0.11 and from 0.5 to 0.55 being perceived as equal changes in brightness.

Colour depends primarily on the reflectance properties of an object. We see those rays that are reflected, while others are absorbed. However, we also must consider the colour of the

light source, and the nature of human visual system. For example, an object that reflects both red and green will appear green when there is green but no red light illuminating it, and conversely it will appear red in the absence of green light. In pure white light, it will appear yellow (= red + green).



## Assignment No. 12

**Aim:** Mean and Covariance (Through Virtual Lab)

**Theory:**

Given samples of a class,  $x_1, x_2, \dots, x_n$ , each being an  $d$ -dimensional vector, we define the mean,  $\mu$ , and the covariance matrix,  $\Sigma$  are defined as:

$$\mu = \frac{1}{N} \sum_{i=1}^N x_i$$
$$\Sigma = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)(x_i - \mu)^t$$

**Properties of  $\mu$  and  $\Sigma$**

1. Note that  $x_i$ s are  $d$ -dimensional column vectors, and hence the mean,  $\mu$ , is also a  $d$ -dimensional column vector. The covariance matrix,  $\Sigma$ , is a  $d \times d$  symmetric square matrix.
2. The mean,  $\mu$  is the centroid of the samples in the feature space.
3. The covariance matrix,  $\Sigma$  is a positive semi-definite matrix.

The pane on the right allows you to specify the mean and covariance matrices for a set of samples in 2-dimensions.

**Stage 1:**

1. Set the default parameters and click the generate button; Observe the dataset for multiple instances of samples.
2. Click the estimate button to compute the mean and covariance of the generated samples; Observe the mismatch between the two. Repeat the process for multiple sets of samples generated from the same distribution and observe the variations in the estimate.

**Stage 2:**

1. Repeat the above procedure for the different values for the number of generated samples.
2. Plot a graph between the average error, and number of samples.
3. Note down your inferences regarding the graph.

**Stage 3:**

1. Generate your own dataset by clicking on the plot area. Note that the distribution need not be similar to a normal density.

2. Estimate the mean and covariance matrices.
3. Note down your inferences regarding the error committed if you assume the distribution to be normal.

The screenshot displays a web browser window with multiple tabs. The active tab is 'Virtual Labs', showing the URL 'cse20-iiith.vlabs.ac.in/exp/mean-and-covariance/procedure.html'. The website header includes the 'Virtual Labs' logo and navigation links for 'HOME', 'PARTNERS', and 'CONTACT'. The main content area is titled 'Computer Science and Engineering > Pattern recognition > Experiments' and 'Mean and Covariance'. A left sidebar contains a list of links: 'Aim', 'Theory', 'Objective', 'Procedure' (highlighted in orange), 'Pretest', 'Simulation', 'Assignment', 'References', and 'Feedback'. The main text area describes the experiment's purpose and provides two stages of instructions. Stage 1 involves setting parameters and observing the dataset. Stage 2 involves repeating the procedure for different sample sizes and plotting a graph of average error versus the number of samples. The bottom of the image shows a Windows taskbar with various application icons and system information like '37°C Sunny' and the date '17-03-2022'.

Virtual Labs  
An MoE Govt of India Initiative

HOME PARTNERS CONTACT

Computer Science and Engineering > Pattern recognition > Experiments

**Mean and Covariance**

The pane on the right allows you to specify the mean and covariance matrices for a set of samples in 2-dimensions.

**Stage 1:**

1. Set the default parameters and click the generate button; Observe the dataset for multiple instances of samples.
2. Click the estimate button to compute the mean and covariance of the generated samples; Observe the mismatch between the two. Repeat the process for multiple sets of samples generated from the same distribution and observe the variations in the estimate.

**Stage 2:**

1. Repeat the above procedure for the different values for the number of generated samples.
2. Plot a graph between the average error, and number of samples.
3. Note down your inferences regarding the graph.

# Assignment No. 1

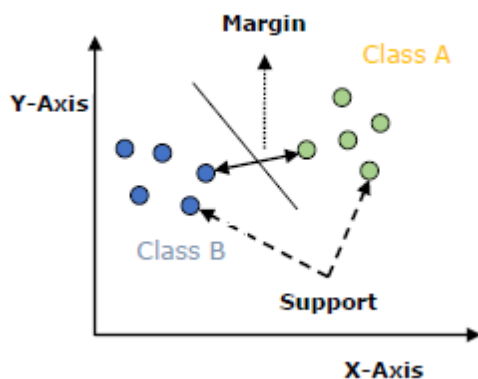
**Aim:** Write a python program to predicting if a customer with certain age and Salary will purchase a product or not using support vector machine.

## Theory:

Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems. In 1960s, SVMs were first introduced but later they got refined in 1990. SVMs have their unique way of implementation as compared to other machine learning algorithms. Lately, they are extremely popular because of their ability to handle multiple continuous and categorical variables.

## Working of SVM

An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).



The followings are important concepts in SVM –

- **Support Vectors** – Datapoints that are closest to the hyperplane is called support vectors. Separating line will be defined with the help of these data points.
- **Hyperplane** – As we can see in the above diagram, it is a decision plane or space which is divided between a set of objects having different classes.
- **Margin** – It may be defined as the gap between two lines on the closet data points of different classes. It can be calculated as the perpendicular distance from the line to the support vectors. Large margin is considered as a good margin and small margin is considered as a bad margin.

The main goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH) and it can be done in the following two steps –

- First, SVM will generate hyperplanes iteratively that segregates the classes in best way.
- Then, it will choose the hyperplane that separates the classes correctly.

## Implementing SVM in Python

- For implementing SVM in Python – We will start with the standard libraries import as follows –

### SVM Kernels

In practice, SVM algorithm is implemented with kernel that transforms an input data space into the required form. SVM uses a technique called the kernel trick in which kernel takes a low dimensional input space and transforms it into a higher dimensional space. In simple words, kernel converts non-separable problems into separable problems by adding more dimensions to it. It makes SVM more powerful, flexible and accurate. The following are some of the types of kernels used by SVM.

#### Linear Kernel

It can be used as a dot product between any two observations. The formula of linear kernel is as below –

$$K(x, x_i) = \sum (x * x_i) \quad K(x, x_i) = \sum (x * x_i)$$

From the above formula, we can see that the product between two vectors say  $x$  &  $x_i$  is the sum of the multiplication of each pair of input values.

#### Polynomial Kernel

It is more generalized form of linear kernel and distinguish curved or nonlinear input space. Following is the formula for polynomial kernel –

$$k(X, X_i) = 1 + \sum (X * X_i)^d \quad k(X, X_i) = 1 + \sum (X * X_i)^d$$

Here  $d$  is the degree of polynomial, which we need to specify manually in the learning algorithm.

#### Radial Basis Function (RBF) Kernel

RBF kernel, mostly used in SVM classification, maps input space in indefinite dimensional space. Following formula explains it mathematically –

$$K(x, x_i) = \exp(-\gamma * \sum (x - x_i)^2) \quad K(x, x_i) = \exp(-\gamma * \sum (x - x_i)^2)$$

Here,  $\gamma$  ranges from 0 to 1. We need to manually specify it in the learning algorithm. A good default value of  $\gamma$  is 0.1.

As we implemented SVM for linearly separable data, we can implement it in Python for the data that is not linearly separable. It can be done by using kernels.

### Program:

```
#Importing the libraries
import numpy as np
import pandas as pd

#Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
```



```

X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

dataset.head()

X

y

#Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size =
0.25, random_state = 0)

#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

#Fitting SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)

#Predicting the Test set results
y_pred = classifier.predict(X_test)
classifier.score(X_test,y_test)

from sklearn.metrics import confusion_matrix,classification_report
print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test,y_pred))

```

## Assignment No. 2

**Aim:** Associative Rule Mining: Implementation of Apriori algorithm.

**Theory:**

The Apriori algorithm uses frequent itemsets to generate association rules, and it is designed to work on the databases that contain transactions. With the help of these association rule, it determines how strongly or how weakly two objects are connected. This algorithm uses a **breadth-first search** and **Hash Tree** to calculate the itemset associations efficiently. It is the iterative process for finding the frequent itemsets from the large dataset.

This algorithm was given by the **R. Agrawal** and **Srikant** in the year **1994**. It is mainly used for *market basket analysis* and helps to find those products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

**What is Frequent Itemset?**

Frequent itemsets are those items whose support is greater than the threshold value or user-specified minimum support. It means if A & B are the frequent itemsets together, then individually A and B should also be the frequent itemset.

Suppose there are the two transactions: A= {1,2,3,4,5}, and B= {2,3,7}, in these two transactions, 2 and 3 are the frequent itemsets.

### Steps for Apriori Algorithm

Below are the steps for the apriori algorithm:

**Step-1:** Determine the support of itemsets in the transactional database, and select the minimum support and confidence.

**Step-2:** Take all supports in the transaction with higher support value than the minimum or selected support value.

**Step-3:** Find all the rules of these subsets that have higher confidence value than the threshold or minimum confidence.

**Step-4:** Sort the rules as the decreasing order of lift.

### Apriori Algorithm Working

We will understand the apriori algorithm using an example and mathematical calculation:

**Example:** Suppose we have the following dataset that has various transactions, and from this dataset, we need to find the frequent itemsets and generate the association rules using the Apriori algorithm:

TID	ITEMSETS
T1	A, B
T2	B, D
T3	B, C
T4	A, B, D
T5	A, C
T6	B, C
T7	A, C
T8	A, B, C, E
T9	A, B, C

**Given: Minimum Support= 2, Minimum Confidence= 50%**

**Solution:**

### Step-1: Calculating C1 and L1:

- In the first step, we will create a table that contains support count (The frequency of each itemset individually in the dataset) of each itemset in the given dataset. This table is called the **Candidate set or C1**.

Itemset	Support_Count
A	6
B	7
C	5
D	2
E	1

Now, we will take out all the itemsets that have the greater support count than the Minimum Support (2). It will give us the table for the **frequent itemset L1**. Since all the itemsets have greater or equal support count than the minimum support, except the E, so E itemset will be removed.

Itemset	Support_Count
A	6
B	7
C	5
D	2

### Step-2: Candidate Generation C2, and L2:

- In this step, we will generate C2 with the help of L1. In C2, we will create the pair of the itemsets of L1 in the form of subsets.

- After creating the subsets, we will again find the support count from the main transaction table of datasets, i.e., how many times these pairs have occurred together in the given dataset. So, we will get the below table for C2:

Itemset	Support_Count
{A, B}	4
{A, C}	4
{A, D}	1
{B, C}	4
{B, D}	2
{C, D}	0

Again, we need to compare the C2 Support count with the minimum support count, and after comparing, the itemset with less support count will be eliminated from the table C2. It will give us the below table for L2

Itemset	Support_Count
{A, B}	4
{A, C}	4
{B, C}	4
{B, D}	2

**A, B, C, D**

### Step-3: Candidate generation C3, and L3

For C3, we will repeat the same two processes, but now we will form the C3 table with subsets of three itemsets together, and will calculate the support count from the dataset. It will give the below table:

Itemset	Support_Count
{A, B, C}	2
{B, C, D}	1
{A, C, D}	0
{A, B, D}	0

- Now we will create the L3 table. As we can see from the above C3 table, there is only one combination of itemset that has support count equal to the minimum support count. So, the L3 will have only one combination, i.e., **{A, B, C}**.

## Step-4: Finding the association rules for the subsets:

To generate the association rules, first, we will create a new table with the possible rules from the occurred combination {A, B.C}. For all the rules, we will calculate the Confidence using formula  $\frac{\text{sup}(A \wedge B)}{\text{sup}(A)}$ . After calculating the confidence value for all rules, we will exclude the rules that have less confidence than the minimum threshold(50%).

Consider the below table:

Rules	Support	Confidence
$A \wedge B \rightarrow C$	2	$\frac{\text{Sup}\{(A \wedge B) \wedge C\}}{\text{sup}(A \wedge B)} = \frac{2}{4} = 0.5 = 50\%$
$B \wedge C \rightarrow A$	2	$\frac{\text{Sup}\{(B \wedge C) \wedge A\}}{\text{sup}(B \wedge C)} = \frac{2}{4} = 0.5 = 50\%$
$A \wedge C \rightarrow B$	2	$\frac{\text{Sup}\{(A \wedge C) \wedge B\}}{\text{sup}(A \wedge C)} = \frac{2}{4} = 0.5 = 50\%$
$C \rightarrow A \wedge B$	2	$\frac{\text{Sup}\{(C \wedge (A \wedge B))\}}{\text{sup}(C)} = \frac{2}{5} = 0.4 = 40\%$
$A \rightarrow B \wedge C$	2	$\frac{\text{Sup}\{(A \wedge (B \wedge C))\}}{\text{sup}(A)} = \frac{2}{6} = 0.33 = 33.33\%$
$B \rightarrow B \wedge C$	2	$\frac{\text{Sup}\{(B \wedge (B \wedge C))\}}{\text{sup}(B)} = \frac{2}{7} = 0.28 = 28\%$

As the given threshold or minimum confidence is 50%, so the first three rules  $A \wedge B \rightarrow C$ ,  $B \wedge C \rightarrow A$ , and  $A \wedge C \rightarrow B$  can be considered as the strong association rules for the given problem.

## Advantages of Apriori Algorithm

- This is easy to understand algorithm
- The join and prune steps of the algorithm can be easily implemented on large datasets.

## Disadvantages of Apriori Algorithm

- The apriori algorithm works slow compared to other algorithms.
- The overall performance can be reduced as it scans the database for multiple times.
- The time complexity and space complexity of the apriori algorithm is  $O(2^D)$ , which is very high. Here D represents the horizontal width present in the database.

**Program:**

# Market Basket Analysis of Store Data

## Dataset Description

- Different products given 7500 transactions over the course of a week at a French retail store.
- We have library(**apyori**) to calculate the association rule using Apriori.

---

[6]

```
!pip install apyori #This installs the Apyori package for using the Association Mining Apriori algorithm

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from apyori import apriori
```

Collecting apyori

Downloading apyori-1.1.2.tar.gz (8.6 kB)

Building wheels for collected packages: apyori

Building wheel for apyori (setup.py) ... done

Created wheel for apyori: filename=apyori-1.1.2-py3-none-any.whl size=5974  
sha256=716a556bac1d2c5d32dbcb182e8a9c5556b91f2ebe30b66a3d73a379aa554df7

Stored in directory:

/root/.cache/pip/wheels/cb/f6/e1/57973c631d27efd1a2f375bd6a83b2a616c4021f24aab84080

Successfully built apyori

Installing collected packages: apyori

Successfully installed apyori-1.1.2

---

## Import the Library

## Read data and Display

---

[2]

```
store_data = pd.read_csv("store_data.csv", header=None)

display(store_data.head())

print(store_data.shape)
```

---

## Preprocessing on Data

- Here we need a data in form of list for Apriori Algorithm.

---

[3]

```
records = []  
for i in range(1, 7501):  
    records.append([str(store_data.values[i, j]) for j in range(0, 20)])
```

---

[4]

```
print(type(records))
```

```
<class 'list'>
```























---

## Apriori Algorithm

- Now time to apply algorithm on data.
- We have provide `min_support`, `min_confidence`, `min_lift`, and `min length` of sample-set for find rule.

Measure 1: Support.

This says how popular an itemset is, as measured by the proportion of transactions in which an itemset appears. In Table 1 below, the support of {apple} is 4 out of 8, or 50%. Itemsets can also contain multiple items. For instance, the support of {apple, beer, rice} is 2 out of 8, or 25%.

Transaction 1	   
Transaction 2	  
Transaction 3	 
Transaction 4	 
Transaction 5	   
Transaction 6	  
Transaction 7	 
Transaction 8	 

If you discover that sales of items beyond a certain proportion tend to have a significant impact on your profits, you might consider using that proportion as your support threshold. You may then identify itemsets with support values above this threshold as significant itemsets.

Measure 2: Confidence.

This says how likely item Y is purchased when item X is purchased, expressed as {X → Y}. This is measured by the proportion of transactions with item X, in which item Y also appears. In Table 1, the confidence of {apple → beer} is 3 out of 4, or 75%.

$$\text{Confidence } \{\text{apple} \rightarrow \text{beer}\} = \frac{\text{Support } \{\text{apple}, \text{beer}\}}{\text{Support } \{\text{apple}\}}$$

One drawback of the confidence measure is that it might misrepresent the importance of an association. This is because it only accounts for how popular apples are, but not beers. If beers are also very popular in general, there will be a higher chance that a transaction containing apples will also contain beers, thus inflating the confidence measure. To account for the base popularity of both constituent items, we use a third measure called lift.

Measure 3: Lift.

This says how likely item Y is purchased when item X is purchased, while controlling for how popular item Y is. In Table 1, the lift of {apple → beer} is 1, which implies no association between items. A lift value greater than 1 means that item Y is likely to be bought if item X is bought, while a value less than 1 means that item Y is unlikely to be bought if item X is bought.



bought.

$$\text{Lift} \{ \text{🍎} \rightarrow \text{🍺} \} = \frac{\text{Support} \{ \text{🍎}, \text{🍺} \}}{\text{Support} \{ \text{🍎} \} \times \text{Support} \{ \text{🍺} \}}$$

---

[7]

2s

```
association_rules = apriori(records, min_support=0.0045, min_confidence=0.2, min_lift=3, min_length=2)
association_results = list(association_rules)
```

---

## How many relation derived

---

[8]

0s

```
print("There are {} Relation derived.".format(len(association_results)))
```

There are 48 Relation derived.

---

## Association Rules Derived

---

[9]

0s

```
for i in range(0, len(association_results)):
    print(association_results[i][0])
```

```
frozenset({'chicken', 'light cream'})
frozenset({'mushroom cream sauce', 'escalope'})
frozenset({'pasta', 'escalope'})
frozenset({'herb & pepper', 'ground beef'})
frozenset({'ground beef', 'tomato sauce'})
frozenset({'olive oil', 'whole wheat pasta'})
frozenset({'shrimp', 'pasta'})
frozenset({'chicken', 'nan', 'light cream'})
frozenset({'chocolate', 'shrimp', 'frozen vegetables'})
frozenset({'spaghetti', 'ground beef', 'cooking oil'})
frozenset({'mushroom cream sauce', 'nan', 'escalope'})
frozenset({'nan', 'pasta', 'escalope'})
frozenset({'spaghetti', 'ground beef', 'frozen vegetables'})
frozenset({'olive oil', 'milk', 'frozen vegetables'})
frozenset({'shrimp', 'mineral water', 'frozen vegetables'})
frozenset({'olive oil', 'spaghetti', 'frozen vegetables'})
frozenset({'shrimp', 'spaghetti', 'frozen vegetables'})
```

```

frozenset({'spaghetti', 'tomatoes', 'frozen vegetables'})
frozenset({'spaghetti', 'ground beef', 'grated cheese'})
frozenset({'herb & pepper', 'ground beef', 'mineral water'})
frozenset({'nan', 'herb & pepper', 'ground beef'})
frozenset({'spaghetti', 'herb & pepper', 'ground beef'})
frozenset({'olive oil', 'ground beef', 'milk'})
frozenset({'nan', 'ground beef', 'tomato sauce'})
frozenset({'shrimp', 'spaghetti', 'ground beef'})
frozenset({'olive oil', 'spaghetti', 'milk'})
frozenset({'olive oil', 'mineral water', 'soup'})
frozenset({'olive oil', 'nan', 'whole wheat pasta'})
frozenset({'shrimp', 'nan', 'pasta'})
frozenset({'olive oil', 'spaghetti', 'pancakes'})
frozenset({'chocolate', 'nan', 'shrimp', 'frozen vegetables'})
frozenset({'nan', 'ground beef', 'cooking oil', 'spaghetti'})
frozenset({'nan', 'ground beef', 'spaghetti', 'frozen vegetables'})
frozenset({'spaghetti', 'milk', 'mineral water', 'frozen vegetables'})
frozenset({'olive oil', 'nan', 'milk', 'frozen vegetables'})
frozenset({'shrimp', 'nan', 'mineral water', 'frozen vegetables'})
frozenset({'olive oil', 'nan', 'spaghetti', 'frozen vegetables'})
frozenset({'shrimp', 'nan', 'spaghetti', 'frozen vegetables'})
frozenset({'tomatoes', 'nan', 'spaghetti', 'frozen vegetables'})
frozenset({'nan', 'ground beef', 'spaghetti', 'grated cheese'})
frozenset({'nan', 'herb & pepper', 'ground beef', 'mineral water'})
frozenset({'nan', 'herb & pepper', 'ground beef', 'spaghetti'})
frozenset({'olive oil', 'nan', 'ground beef', 'milk'})
frozenset({'shrimp', 'nan', 'ground beef', 'spaghetti'})
frozenset({'olive oil', 'nan', 'milk', 'spaghetti'})
frozenset({'olive oil', 'nan', 'mineral water', 'soup'})
frozenset({'olive oil', 'nan', 'pancakes', 'spaghetti'})
frozenset({'spaghetti', 'milk', 'frozen vegetables', 'mineral water', 'nan'})

```

---

## Rules Generated

---

[10]

```

for item in association_results:

    # first index of the inner list

    # Contains base item and add item

    pair = item[0]

    items = [x for x in pair]

    print("Rule: " + items[0] + " -> " + items[1])

    # second index of the inner list

    print("Support: " + str(item[1]))

    # third index of the list located at 0th

    # of the third index of the inner list

    print("Confidence: " + str(item[2][0][2]))

```

```
print("Lift: " + str(item[2][0][3]))

print("=====")
```

Rule: chicken -> light cream  
 Support: 0.004533333333333334  
 Confidence: 0.2905982905982906  
 Lift: 4.843304843304844

=====

Rule: mushroom cream sauce -> escalope  
 Support: 0.005733333333333333  
 Confidence: 0.30069930069930073  
 Lift: 3.7903273197390845

=====

Rule: pasta -> escalope  
 Support: 0.005866666666666667  
 Confidence: 0.37288135593220345  
 Lift: 4.700185158809287

=====

Rule: herb & pepper -> ground beef  
 Support: 0.016  
 Confidence: 0.3234501347708895  
 Lift: 3.2915549671393096

=====

Rule: ground beef -> tomato sauce  
 Support: 0.005333333333333333  
 Confidence: 0.37735849056603776  
 Lift: 3.840147461662528

=====

Rule: olive oil -> whole wheat pasta  
 Support: 0.008  
 Confidence: 0.2714932126696833  
 Lift: 4.130221288078346

=====

Rule: shrimp -> pasta  
 Support: 0.005066666666666666  
 Confidence: 0.3220338983050848  
 Lift: 4.514493901473151

=====

Rule: chicken -> nan  
 Support: 0.004533333333333334  
 Confidence: 0.2905982905982906  
 Lift: 4.843304843304844

=====

Rule: chocolate -> shrimp  
 Support: 0.005333333333333333  
 Confidence: 0.23255813953488372  
 Lift: 3.260160834601174

=====

Rule: spaghetti -> ground beef  
 Support: 0.0048  
 Confidence: 0.5714285714285714  
 Lift: 3.281557646029315

=====

Rule: mushroom cream sauce -> nan  
 Support: 0.005733333333333333  
 Confidence: 0.30069930069930073  
 Lift: 3.7903273197390845

=====

Rule: nan -> pasta  
 Support: 0.005866666666666667  
 Confidence: 0.37288135593220345  
 Lift: 4.700185158809287

=====

Rule: spaghetti -> ground beef  
Support: 0.008666666666666666  
Confidence: 0.3110047846889952  
Lift: 3.164906221394116

Rule: olive oil -> milk  
Support: 0.0048  
Confidence: 0.20338983050847456  
Lift: 3.094165778526489

Rule: shrimp -> mineral water  
Support: 0.0072  
Confidence: 0.3068181818181818  
Lift: 3.2183725365543547

Rule: olive oil -> spaghetti  
Support: 0.005733333333333333  
Confidence: 0.20574162679425836  
Lift: 3.1299436124887174

Rule: shrimp -> spaghetti  
Support: 0.006  
Confidence: 0.21531100478468898  
Lift: 3.0183785717479763

Rule: spaghetti -> tomatoes  
Support: 0.006666666666666667  
Confidence: 0.23923444976076555  
Lift: 3.497579674864993

Rule: spaghetti -> ground beef  
Support: 0.005333333333333333  
Confidence: 0.3225806451612903  
Lift: 3.282706701098612

Rule: herb & pepper -> ground beef  
Support: 0.006666666666666667  
Confidence: 0.390625  
Lift: 3.975152645861601

Rule: nan -> herb & pepper  
Support: 0.016  
Confidence: 0.3234501347708895  
Lift: 3.2915549671393096

Rule: spaghetti -> herb & pepper  
Support: 0.0064  
Confidence: 0.3934426229508197  
Lift: 4.003825878061259

Rule: olive oil -> ground beef  
Support: 0.004933333333333333  
Confidence: 0.22424242424242424  
Lift: 3.411395906324912

Rule: nan -> ground beef  
Support: 0.005333333333333333  
Confidence: 0.37735849056603776  
Lift: 3.840147461662528

Rule: shrimp -> spaghetti  
Support: 0.006  
Confidence: 0.5232558139534884  
Lift: 3.004914704939635

=====  
Rule: olive oil -> spaghetti  
Support: 0.0072  
Confidence: 0.20300751879699247  
Lift: 3.0883496774390333  
=====

Rule: olive oil -> mineral water  
Support: 0.0052  
Confidence: 0.2254335260115607  
Lift: 3.4295161157945335  
=====

Rule: olive oil -> nan  
Support: 0.008  
Confidence: 0.2714932126696833  
Lift: 4.130221288078346  
=====

Rule: shrimp -> nan  
Support: 0.005066666666666666  
Confidence: 0.3220338983050848  
Lift: 4.514493901473151  
=====

Rule: olive oil -> spaghetti  
Support: 0.005066666666666666  
Confidence: 0.20105820105820105  
Lift: 3.0586947422647217  
=====

Rule: chocolate -> nan  
Support: 0.005333333333333333  
Confidence: 0.23255813953488372  
Lift: 3.260160834601174  
=====

Rule: nan -> ground beef  
Support: 0.0048  
Confidence: 0.5714285714285714  
Lift: 3.281557646029315  
=====

Rule: nan -> ground beef  
Support: 0.008666666666666666  
Confidence: 0.3110047846889952  
Lift: 3.164906221394116  
=====

Rule: spaghetti -> milk  
Support: 0.004533333333333334  
Confidence: 0.28813559322033905  
Lift: 3.0224013274860737  
=====

Rule: olive oil -> nan  
Support: 0.0048  
Confidence: 0.20338983050847456  
Lift: 3.094165778526489  
=====

Rule: shrimp -> nan  
Support: 0.0072  
Confidence: 0.3068181818181818  
Lift: 3.2183725365543547  
=====

Rule: olive oil -> nan  
Support: 0.005733333333333333  
Confidence: 0.20574162679425836  
Lift: 3.1299436124887174  
=====

Rule: shrimp -> nan  
Support: 0.006  
Confidence: 0.21531100478468898

Lift: 3.0183785717479763

Rule: tomatoes -> nan

Support: 0.006666666666666667

Confidence: 0.23923444976076555

Lift: 3.497579674864993

Rule: nan -> ground beef

Support: 0.005333333333333333

Confidence: 0.3225806451612903

Lift: 3.282706701098612

Rule: nan -> herb & pepper

Support: 0.006666666666666667

Confidence: 0.390625

Lift: 3.975152645861601

Rule: nan -> herb & pepper

Support: 0.0064

Confidence: 0.3934426229508197

Lift: 4.003825878061259

Rule: olive oil -> nan

Support: 0.004933333333333333

Confidence: 0.22424242424242424

Lift: 3.411395906324912

Rule: shrimp -> nan

Support: 0.006

Confidence: 0.5232558139534884

Lift: 3.004914704939635

Rule: olive oil -> nan

Support: 0.0072

Confidence: 0.20300751879699247

Lift: 3.0883496774390333

Rule: olive oil -> nan

Support: 0.0052

Confidence: 0.2254335260115607

Lift: 3.4295161157945335

Rule: olive oil -> nan

Support: 0.005066666666666666

Confidence: 0.20105820105820105

Lift: 3.0586947422647217

Rule: spaghetti -> milk

Support: 0.004533333333333334

Confidence: 0.28813559322033905

Lift: 3.0224013274860737