**CHAPTER-4**

**IMPLEMENTATION**

Here, in our project, feature selection is made from 30 features by using different wrapper feature selection methods of Machine learning. Models are constructed with the selected features that are generated from different selection methods by using Random Forest Classifier Algorithm. Accuracy is calculated for each model and the best model with best features is selected as result based on high accuracy.

**4.1 Libraries Used**

Python is increasingly being used as a scientific language. Matrix and vector manipulation are extremely important for scientific computations. Both NumPy and Pandas have emerged to be essential libraries for any scientific computation, including machine learning, in python due to their intuitive syntax and high-performance matrix computation capabilities.

**NumPy:**

NumPy stands for ‘Numerical Python’ or ‘Numeric Python’. It is an open source module of Python which provides fast mathematical computation on arrays and matrices. Since, arrays and matrices are an essential part of the Machine Learning ecosystem, NumPy along with Machine Learning modules like Scikit-learn, Pandas, Matplotlib, TensorFlow, etc. complete the Python Machine Learning Ecosystem.

NumPy provides the essential multi-dimensional array-oriented computing functionalities designed for high-level mathematical functions and scientific computation. NumPy can be imported into the notebook using

import numpy as np.

**Pandas:**

Similar to NumPy, Pandas is one of the most widely used python libraries in data science. It provides high-performance, easy to use structures and data analysis tools. Pandas provides in-memory 2d table object called Data frame. It is like a spreadsheet with column names and row labels.

Hence, with 2d tables, pandas are capable of providing many additional functionalities like creating pivot tables, computing columns based on other columns and plotting graphs. Pandas can be imported into Python using:

import pandas as pd.

**pip:**

The pip command is a tool for installing and managing Python packages, such as those found in the Python Package Index. It's a replacementfor easy install. The easiest way to install the nfl\* python modules and keep them up-to-date is with a Python-based package manager called [Pip](http://en.wikipedia.org/wiki/Pip_(package_manager)).

**Sklearn:**

Skikit-learn is a free software machine library for Python programming language.It features various classification , regression and clustering algorithms including support vector machine, random forest, k-means and gradient boosting. In our project we have used different features.

* **from** **sklearn.ensemble** **import** RandomForestClassifier:

Used for Random Forest Classifier algorithm.

* **from** **sklearn.model\_selection** **import** train\_test\_split:

Used for Splitting the dataset into Training and Testing.

* **from** **sklearn.metrics** **import** accuracy\_score **as** acc

Used for calculating the Accuracy.

**Mlxtend:**

MLxtend is a library that implements a variety of core algorithms and utilities for machine learning and data mining. It implements a large variety of functions, highlights include sequential feature selection algorithms, implementations of stacked generalization for classifcation and re-gression, and algorithms for frequent pattern mining. The sequential feature selection algorithms cover forward, backward, forward floating, and backward floating selection and leverage scikit-learn’s cross-validation to ensure satisfactory generalization performance upon constructing and selecting feature subsets. It is imported as:

**from** **mlxtend.feature\_selection** **import**SequentialFeatureSelector **as** sfs

**4.2 Implementation:**

After importing the .CSV dataset file into the Google collaborator as we mentioned in Chapter 3.

**Implementing Libraries:**

import pandas as pd

import numpy as np

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score as acc

from sklearn.preprocessing import StandardScaler

from mlxtend.feature\_selection import SequentialFeatureSelector as sfs

from sklearn.metrics import confusion\_matrix

**Input:**

data = pd.read\_csv('CancerDataset.csv')

Reading the dataset file using pandas library.

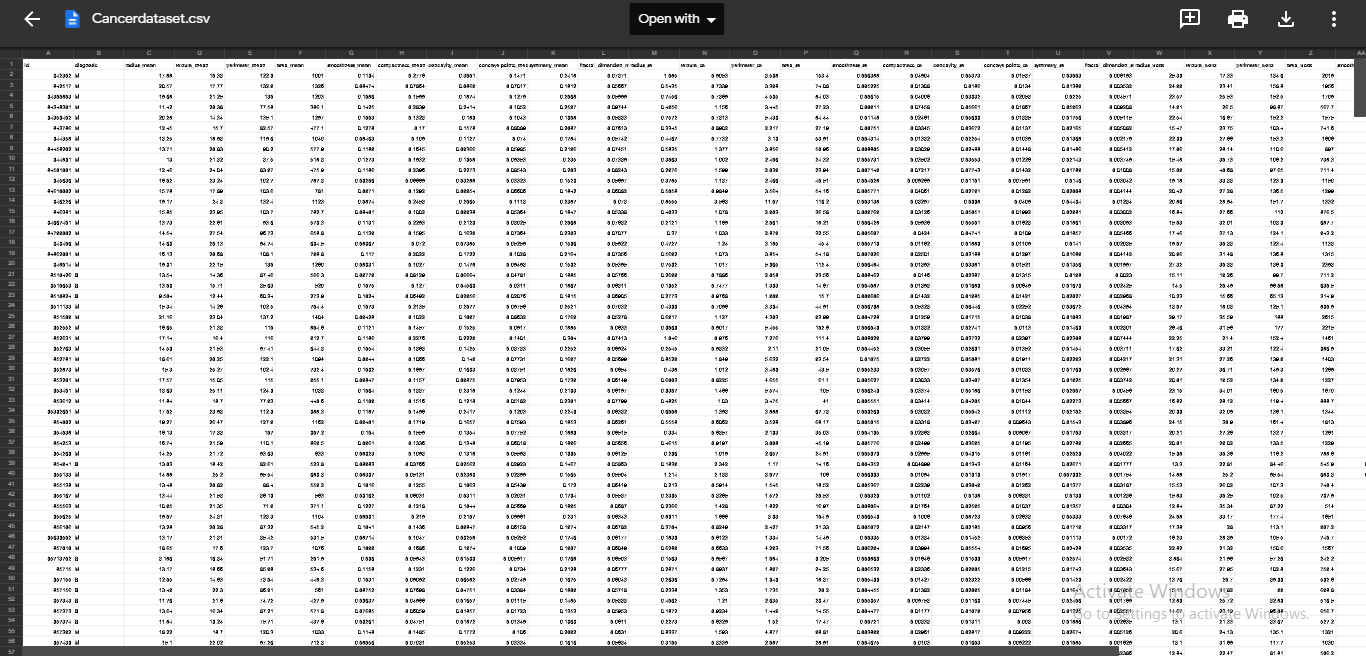
****

Fig 4.1: input dataset of 30 features.

**4.2.1 Feature Selection using Sequential feature selection(SFS):**

Sequential feature selection starts with the evaluation of each individual feature, and selects that which results in the best performing selected algorithm model. Best is depends entirely on the defined evaluation criteria (AUC, prediction accuracy, RMSE, etc. It the simplest greedy search algorithm. ­Starting from the empty set, sequentially add the feature x+ that results in the highest objective function J(Yk+x+ ) when combined with the features Yk that have already been selected .

**Algorithm:**

1. Start with the empty set Y0={φ}
2. Select the next best feature X+ =argmax[J(Yk+X)];x¢Yk
3. Update Yk+1=Yk+ X+ ; k=k+1
4. Goto 2
5. Process continues until the specified number of features are selected.

**Code:**

clf=RandomForestClassifier(n\_estimators=100,n\_jobs=-1)

sfs1=sfs(clf,k\_features=9,forward=True,floating=False,scoring='accuracy',cv=5,verbosve=2)

sfs1=sfs1.fit(X\_train,Y\_train)

Verbose: Displays the information of building tree.

Forward:when it is true then it is a SFS , else it is SBE.

Cv:Cross Validation.

**Output:**

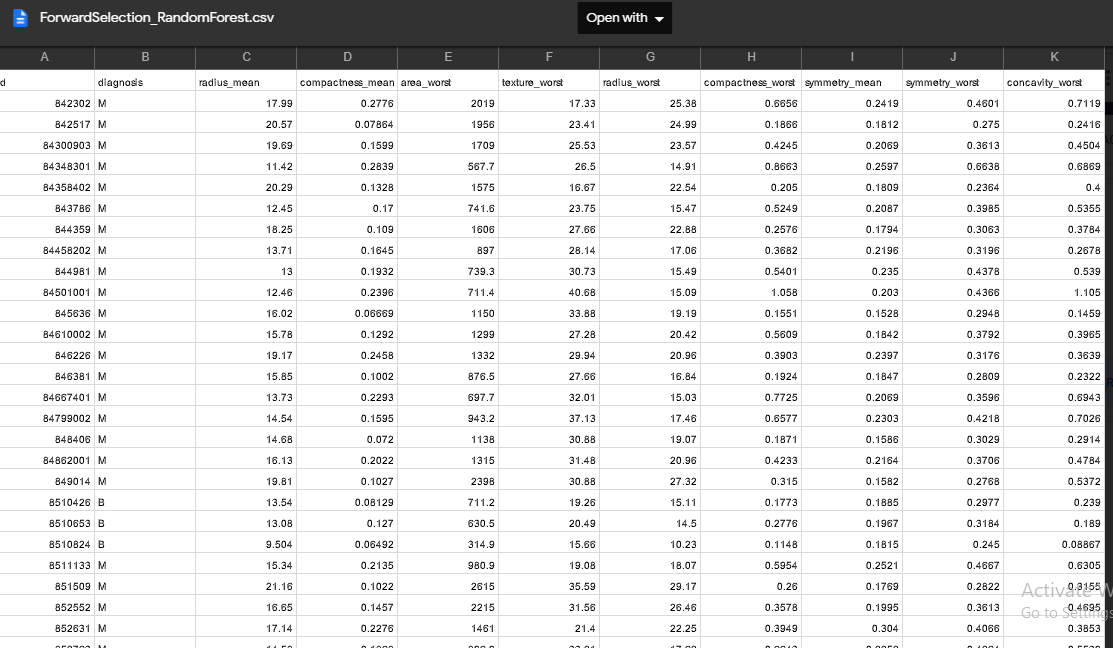
****

Fig 4.2: Selected features using SFS.

**4.2.2 Features selection using Sequential Backward Elimination(SBE):**

Sequential Backward Elimination works in the opposite direction of SFS. Also referred to as SBS (Sequential Backward Selection). Starting from the full set, sequentially remove the feature x− that results in the smallest decrease in the value of the objective function J(Y-x−). Notice that removal of a feature may actually lead to an increase in the objective function J(Yk-x−)>J(Yk). Such functions are said to be non-monotonic.

**Algorithm:**

1. Start with the full set Y0=X
2. Remove the worst feature X- =argmax[J(Yk-X)];x­Yk
3. Update Yk+1=Yk- X- ; k=k+1
4. Goto 2
5. Process continues until the specified number of features are selected.

SBS works best when the optimal feature subset has a large number of features, since SBS spends most of its time visiting large subsets.

**Code:**

clf=RandomForestClassifier(n\_estimators=100,n\_jobs=-1)

sfs1=sfs(clf,k\_features=9,forward=False,floating=False,scoring='accuracy',cv=5,verbose=2)

sfs1=sfs1.fit(X\_train,Y\_train)

.

When forward is equal to False it is a “Sequential backward elimination.”

**Output:**

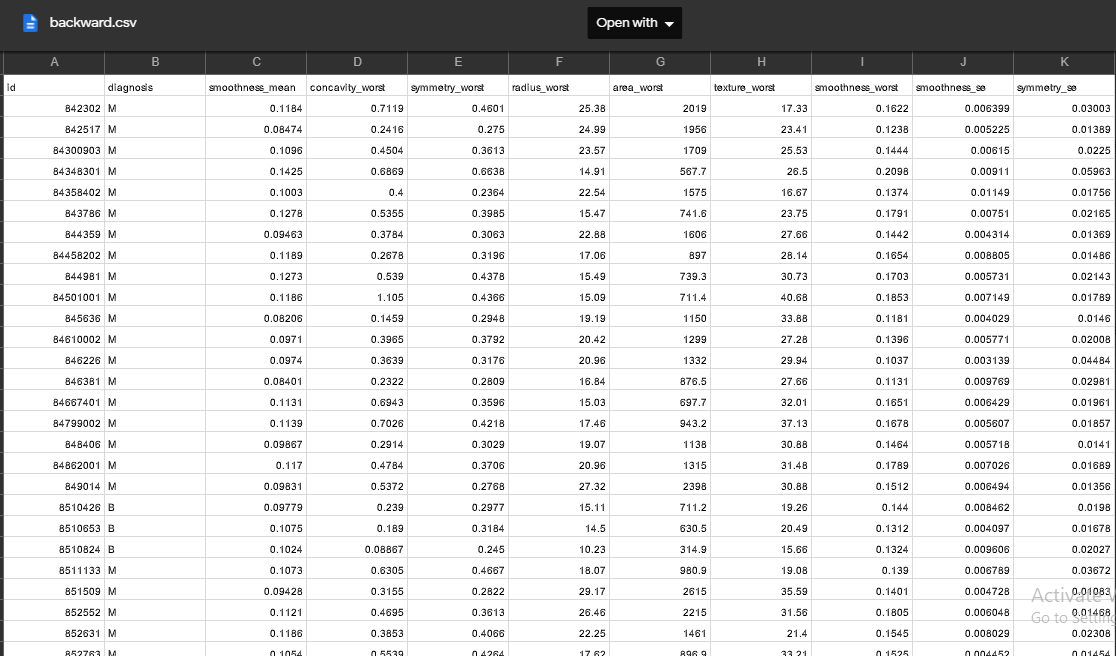
****

Fig 4.3: Output of Sequential backward elimination.

**4.2.3 Model construction using Random Forest Classifier:**

**Random Forest Classifier:**The random forest combines hundreds or thousands of decision trees, trains each one on a slightly different set of the observations, splitting nodes in each tree considering a limited number of the features. The final predictions of the random forest are made by averaging the predictions of each individual tree.This model uses [two key concepts](https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf) that gives it the name random:

**Algorithm:**

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



Fig 4.4: process of randomly selected features.

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

1. Takes the **test features** and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target).
2. Calculate the **votes** for each predicted target.
3. Consider the **high voted** predicted target as the **final prediction** from the random forest algorithm.

Function: clf = RandomForestClassifier(n\_estimators=100, random\_state=1, max\_depth=4,verbose=2)

Where,

X\_train: Traing input data set.

X\_test: Testing input dataset.

Y\_trian: Training ouput label.

Y\_test: Testing output label.

Test\_size: Testing dataset size.

N\_estimators: Number of trees in random forest.

Random\_state: Selects the particular data rows for each execution.

Max\_depth: Maximum depth of a tree.

**Model:** Model is a system that answers the question of a problem statement and this model is created via a process called “training”. The goal of training is to create an accurate model that answers our questions correctly most of the time.

**Splitting of data:** Splitting of data is dividing your data set into two subsets:

* **training set**—a subset to train a model which is used for fit and tune the model.
* **testing set**—a subset to test the trained model which is used to evaluate the model of unseen data.

Training dataset Testing dataset

Fig 4.5 Data splitting.

**4.2.3(a) Model construction with SFS features:**

**Input:**

data = pd.read\_csv('ForwardSelection\_RandomForest.csv')

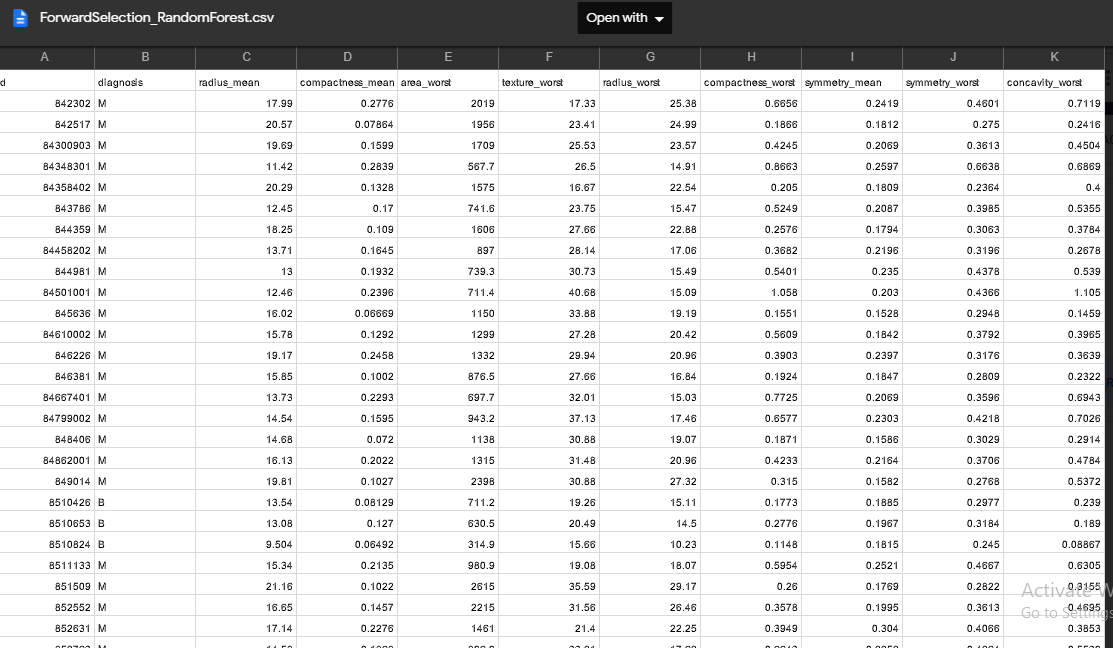
****

Fig 4.6: Selected features using SFS.

**Code:**

X=data.iloc[:,2:]

Y=data.iloc[:,1].values

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,Y,test\_size=0.25,random\_state=1)

sc=StandardScaler()

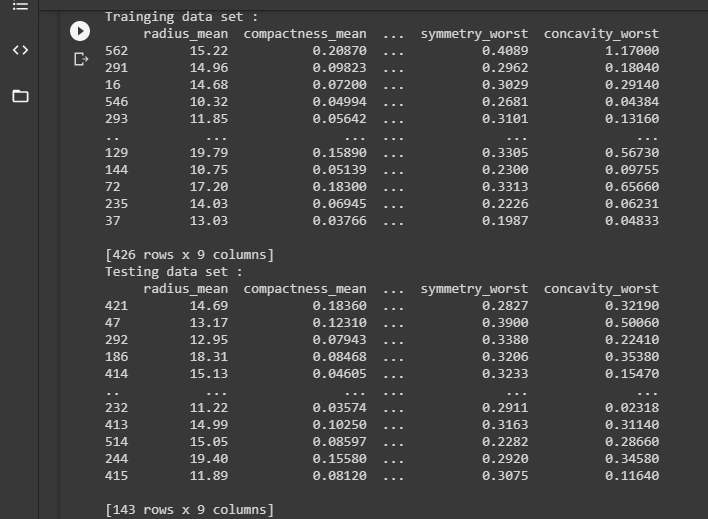
X\_train=sc.fit\_transform(X\_train)

X\_test=sc.fit\_transform(X\_test)

clf = RandomForestClassifier(n\_estimators=100, random\_state=1, max\_depth=4,verbose=2)

clf.fit(X\_train, Y\_train)

**Displaying Training and Testing dataset:**

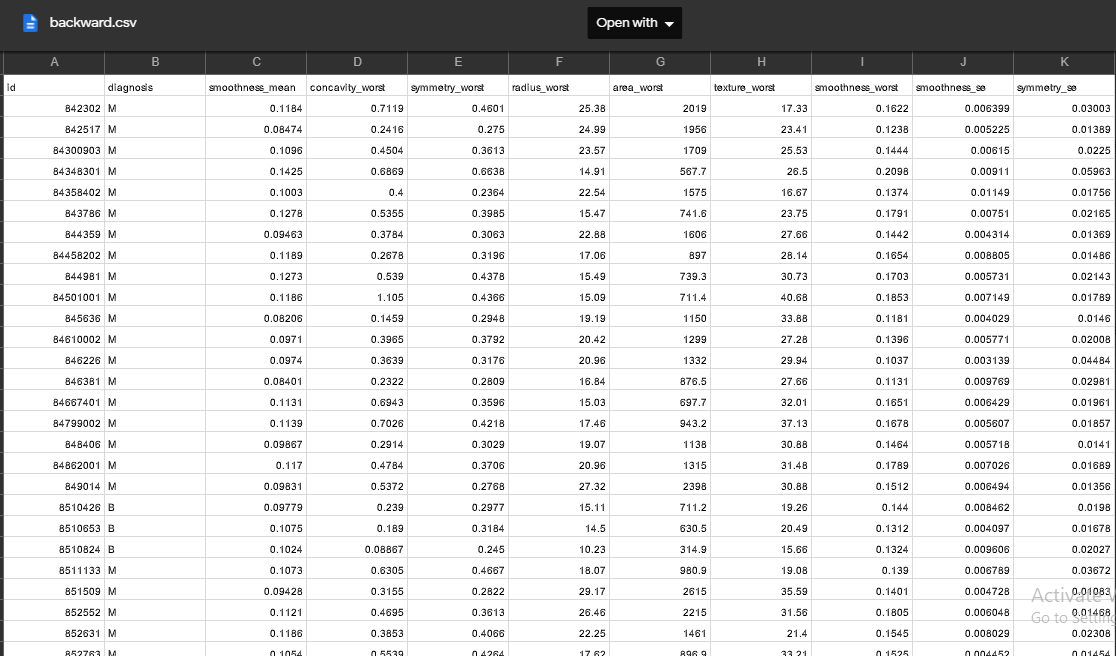
****

**4.7.: training ans testing dataset**

**4.2.3(b) Model construction with SBE features:**

**Input:**

Data=pd.read(“forwardselection.csv”)

****

**Fig 4.8:** selected features using SBE.

**Code:**

X=data.iloc[:,2:]

Y=data.iloc[:,1].values

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,Y,test\_size=0.25,random\_state=1)

sc=StandardScaler()

X\_train=sc.fit\_transform(X\_train)

X\_test=sc.fit\_transform(X\_test)

clf = RandomForestClassifier(n\_estimators=100, random\_state=1, max\_depth=4,verbose=2)

**Displaying Training and Testing dataset:**

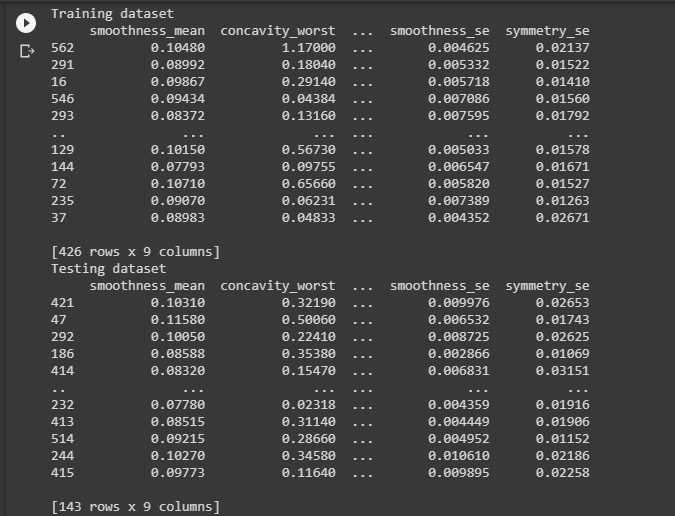
****

Fig 4.9 : Training and Testing dataset.

**4.2.4 Predictions and Metrics:**

Prediction refers to the output of an algorithm after it has been trained on a dataset and applied to new data when forecasting the likelihood of a particular outcome, such as a patient has a cancer or not.

Function used for prediction:

Predict();

y\_train\_pred = clf.predict(X\_train)

**Metrics:**

**Accuracy**:Accuracy is one metric for evaluating classification models. It is the Number of correct predictions.

Accuracy = TP+TN/TP+FP+FN+TN.

**Confuson Matrix**:  A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known.

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

Precision = TP/TP+FP.

**Recall**(Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class

Recall = TP/TP+FN.

**F1 score** - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it’s better to look at both Precision and Recall.

F1 Score = 2\*(Recall \* Precision) / (Recall + Precision).

cm=confusion\_matrix(Y\_test,clf.predict(X\_test))

tn=cm[0][0]

tp=cm[1][1]

fn=cm[1][0]

fp=cm[0][1]

print("Confusion matrix: ")

cm=tp+tn/tp+tn+fn+fp.

print(“Testing Accuracy”cm)

print("Precision of testing data=",tp/(tp+fp))

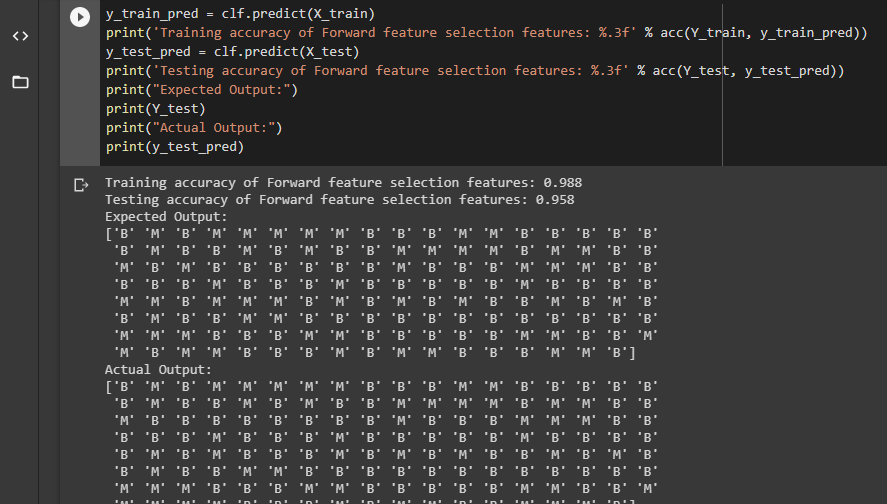
print("Recall of testing data=",tp/(tp+fn))

P=tp/(tp+fp)

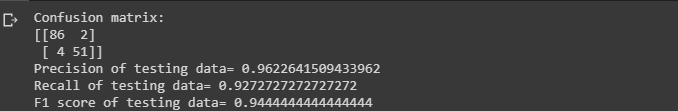
R=tp/(tp+fn)

print("F1 score of testing data=",2\*((P\*R)/(P+R)))

**4.2.3(a)Predictions and metrics for SFS :**

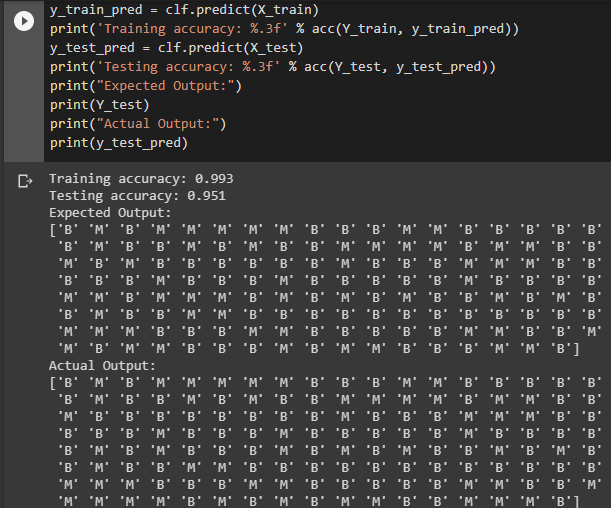
****

**Fig 4.10 : Prediction of SFS**

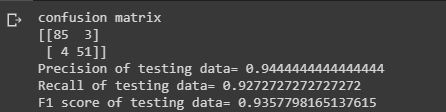
****

**Fig 4.11 :Metric of SFS.**

**4.2.4(b) Predictions and metrics for SBE:**

****

**Fig 4.12 : Prediction of SBE**

****

**Fig 4.13: Accuracy of SBE**