Breast Cancer Detection using ML

In [ ]:

```
##########################Breast Cancer Detection project using Machine
Learning ##############
```

In [431]:

```
import tensorflow as tf
from tensorflow import keras
```

In [432]:

```
import numpy as np
```

In [433]:

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

In [261]:

```
print(tf.__version__)
```

```
1.14.0
```

In [ ]:

```
# The sklearn.preprocessing package provides several common utility functions
and transformer classes to change raw feature vectors into a representation
that is more suitable for the downstream estimators.
```

In [434]:

```
from sklearn import preprocessing
import pandas as pd
```

In [ ]:

In [435]:

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

In [436]:

```
from sklearn.datasets import load_breast_cancer
```

In [437]:

```
data=load_breast_cancer
```

In [438]:

```
import zipfile
```

```
zipfilePath = ("./breast-cancer-wisconsin-data.zip")
zip = zipfile.ZipFile(zipfilePath)
zip.extractall(".")
zip.close()
```

In [439]:

```
dataset = pd.read_csv('data 3.csv')
X = dataset.iloc[:, 1:31].values
Y = dataset.iloc[:, 31].values
```

In [440]:

```
dataset.head()
```

Out[440]:

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	fractal_dimension_worst	Unnamed: 32

5 rows × 33 columns

In [441]:

```
print("Cancer data set dimensions : {}".format(dataset.shape))
```

```
Cancer data set dimensions : (569, 33)
```

In [442]:

```
dataset = pd.read_csv("data 3.csv", usecols = ['diagnosis'])
print(dataset)
```

```
   diagnosis
0          M
1          M
2          M

[569 rows x 1 columns]
```

In [443]:

```
dataset.isnull().sum()
dataset.isna().sum() # used to count NaN values in a dataset
```

Out[443]:

```
diagnosis    0
dtype: int64
```

In [444]:

```
#Encoding categorical data values
from sklearn.preprocessing import LabelEncoder
labelencoder_Y = LabelEncoder()
Y = labelencoder_Y.fit_transform(Y)
```

In [445]:

```
dataset['diagnosis'].unique()
```

Out[445]:

```
array(['M', 'B'], dtype=object)
```

In [446]:

```
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'diagnosis'.
df['diagnosis']= label_encoder.fit_transform(df['diagnosis'])

df['diagnosis'].unique()
```

Out[446]:

```
array([569,   0,   1, 112, 223, 334, 445, 525, 536, 547, 558,   2,  13,
        24,  35,  46,  57,  68,  79,  90, 101, 113, 124, 135, 146, 157,
       168, 179, 190, 201, 212, 224, 235, 246, 257, 268, 279, 290, 301,
       312, 323, 335, 346, 357, 368, 379, 390, 401, 412, 423, 434, 446,
       457, 468, 479, 490, 501, 512, 522, 523, 524, 526, 527, 528, 529,
       530, 531, 532, 533, 534, 535, 537, 538, 539, 540, 541, 542, 543,
       544, 545, 546, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557,
       559, 560, 561, 562, 563, 564, 565, 566, 567, 568,   3,   4,   5,
       121, 122, 123, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134,
```

In [448]:

```
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'diagnosis'.
dataset['diagnosis']= label_encoder.fit_transform(dataset['diagnosis'])

dataset['diagnosis'].unique()
```

Out[448]:

```
array([1, 0])
```

In [449]:

```
print(dataset['diagnosis'])
```

```
0      1
1      1
560    0
561    0
562    1
563    1
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int64
```

In [450]:

```
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)
```

In [451]:

```
dataset.head()

# Initialise the Scaler for feature scaling
scaler = StandardScaler()

# To scale data
scaler.fit(dataset)
```

Out[451]:

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [452]:

```
from sklearn.preprocessing import StandardScaler
dataset.head()

# Initialise the Scaler
scaler = StandardScaler()

# To scale data
scaler.fit(dataset)
```

Out[452]:

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [453]:

```
print(dataset)
```

```
     diagnosis
0            1
1            1
2            1
3            1
4            1
563          1
564          1
565          1
566          1
567          1
568          0

[569 rows x 1 columns]
```

```
df=pd.read_csv('data 3.csv') df.head()
```

Initialise the Scaler¶

```
scaler = StandardScaler()
```

```
scaler.fit(df)
```

In [ ]:

In [455]:

```
df=pd.read_csv("Data 3.csv",converters={"Diagnosis":int})
```

In [ ]:

In [457]:

```
datafram= pd.read_csv("Data 3.csv",converters={"Diagnosis":int})
```

In [ ]:

In [459]:

```
print(datafram)
```

```
           id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  
\
0      842302         M       17.990         10.38          122.80     1001.0  
1      842517         M       20.570         17.77          132.90     1326.0  
2    84300903         M       19.690         21.25          130.00     1203.0  
3    84348301         M       11.420         20.38           77.58      386.1  
4    84358402         M       20.290         14.34          135.10     1297.0  
5      843786         M       12.450         15.70           82.57      477.1  
```

```
563          NaN
564          NaN
565          NaN
566          NaN
567          NaN
568          NaN

[569 rows x 34 columns]
```

In [466]:

```
dataset = pd.read_csv("data 3.csv", usecols = ['diagnosis'])
```

In [467]:

```
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'diagnosis'.
dataset['diagnosis']= label_encoder.fit_transform(dataset['diagnosis'])

dataset['diagnosis'].unique()
```

Out[467]:

```
array([1, 0])
```

In [468]:

```
print(dataset)
```

```
     diagnosis
0            1
1            1
2            1
3            1
4            1
5            1
6            1
7            1
8            1
9            1
10           1
11           1
12           1
13           1
14           1
15           1
```

In [469]:

```
dataset.to_csv("UpdatedData.csv")
```

In [470]:

```
print(dataset)
```

```
     diagnosis
0            1
1            1
2            1
3            1
4            1
5            1
```

In [471]:

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

In [472]:

```
print(dataset)
```

```
     Unnamed: 0  diagnosis
0             0          1
1             1          1
2             2          1
3             3          1
4             4          1
5             5          1
```

In [473]:

```
dataset.to_csv('Data.csv')
```

In [474]:

```
dataset=pd.read_csv('Data 3.csv')
```

In [475]:

```
print(dataset)
```

```
           id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean
\
0      842302         M       17.990         10.38          122.80     1001.0
1      842517         M       20.570         17.77          132.90     1326.0
2    84300903         M       19.690         21.25          130.00     1203.0
3    84348301         M       11.420         20.38           77.58      386.1
4    84358402         M       20.290         14.34          135.10     1297.0
5      843786         M       12.450         15.70           82.57      477.1
```

In [476]:

```
#encoding the values

df=pd.read_csv('data 3.csv')
```

In [ ]:

In [478]:

```
df.dtypes
```

Out[478]:

```
id                          int64
diagnosis                  object
radius_mean               float64
texture_mean              float64
perimeter_mean            float64
area_mean                 float64
smoothness_mean           float64
compactness_mean          float64
concavity_mean            float64
concave points_mean       float64
symmetry_mean             float64
fractal_dimension_mean    float64
```

In [480]:

```
obj_df = df.select_dtypes(include=['object']).copy()
obj_df.head()
```

Out[480]:

	diagnosis
0	M
1	M
2	M
3	M
4	M

In [481]:

```
obj_df[obj_df.isnull().any(axis=1)]
```

Out[481]:

diagnosis

In [482]:

```
obj_df["diagnosis"].value_counts()
```

Out[482]:

```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [483]:

```
obj_df = obj_df.fillna({"diagnosis": "B"})
```

In [484]:

```
cleanup_nums = {"diagnosis":     {"B": 0, "M": 1}}
```

In [485]:

```
obj_df.replace(cleanup_nums, inplace=True)
obj_df.head()
```

Out[485]:

```
   diagnosis
0  1
1  1
2  1
3  1
4  1
```

In [486]:

```
obj_df.dtypes
```

Out[486]:

```
diagnosis    int64
dtype: object
```

In [487]:

```
print(obj_df)
```

```
[569 rows x 1 columns]
```

In [488]:

```
headers =
["id","diagnosis","radius_mean","texture_mean","perimeter_mean","area_mean","
smoothness_mean","compactness_mean","concavity_mean","concave
points_mean","symmetry_mean","fractal_dimension_mean","radius_se","texture_se
","perimeter_se","area_se","smoothness_se","compactness_se","concavity_se","c
oncave
points_se","symmetry_se","fractal_dimension_se","radius_worst","texture_worst
","perimeter_worst","area_worst","smoothness_worst","compactness_worst","conc
avity_worst","concave
points_worst","symmetry_worst","fractal_dimension_worst"]

# Read in the CSV file and convert "?" to NaN
df = pd.read_csv('data 3.csv',
                  header=None, names=headers, na_values="?" )
df.head()
```

In [ ]:

In [490]:

```
df.dtypes
```

Out[490]:

```
id                         object
diagnosis                  object
radius_mean                object
texture_mean               object
perimeter_mean             object
area_mean                  object
smoothness_mean            object
compactness_mean           object
concavity_mean             object
concave points_mean        object
symmetry_mean              object
fractal_dimension_mean     object
radius_se                  object
texture_se                 object
```

In [491]:

```
obj_df = df.select_dtypes(include=['object']).copy()
obj_df.head()
```

Out[491]:

5 rows × 31 columns

In [492]:

```
obj_df[obj_df.isnull().any(axis=1)]
```

Out[492]:

0 rows × 31 columns

In [493]:

```
obj_df["diagnosis"].value_counts()
```

Out[493]:

```
Name: diagnosis, Length: 457, dtype: int64
```

In [495]:

```
obj_df = obj_df.fillna({"diagnosis": "B"})
```

In [496]:

```
obj_df["diagnosis"].value_counts()
```

In [497]:

```
cleanup_nums = {"diagnosis":     {"M": 1, "B": 0}
                }
```

In [498]:

```
obj_df.replace(cleanup_nums, inplace=True)
obj_df.head()
```

Out[498]:

id

842302

842517

84300903

84348301

5 rows × 31 columns

In [ ]:

In [500]:

```
obj_df["diagnosis"] = obj_df["diagnosis"].astype('category')
obj_df.dtypes
```

In [501]:

```
obj_df["diagnosis_cat"] = obj_df["diagnosis"].cat.codes
obj_df.head()
```

Out[501]:

	id						
id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_ ean
842302	M	17.99	10.38	122.8	1001	0.1184	0.2776
842517	M	20.57	17.77	132.9	1326	0.08474	0.07864

84300903	M	19.69	21.25	130	1203	0.1096	0.1599
84348301	M	11.42	20.38	77.58	386.1	0.1425	0.2839

5 rows × 32 columns

In [ ]:

In [ ]:

In [504]:

```
from sklearn.preprocessing import LabelEncoder

lb_make = LabelEncoder()
obj_df["diagnosis_code"] = lb_make.fit_transform(obj_df["diagnosis"])
obj_df[["diagnosis", "diagnosis_code"]].head(11)
```

Out[504]:

	diagnosis	diagnosis_code
id	radius_mean	456
842302	17.99	325
842517	20.57	381
84300903	19.69	361
84348301	11.42	53
84358402	20.29	373
843786	12.45	114
844359	18.25	331
84458202	13.71	188
844981	13	147
84501001	12.46	115

In [505]:

```
from sklearn.preprocessing import LabelEncoder

lb_make = LabelEncoder()
obj_df["id_code"] = lb_make.fit_transform(obj_df["id"])
obj_df[["id", "id_code"]].head(11)
```

Out[505]:

	id	id_code

id	diagnosis	2
842302	M	1
842517	M	1
84300903	M	1
84348301	M	1
84358402	M	1
843786	M	1
844359	M	1
84458202	M	1
844981	M	1
84501001	M	1

In [506]:

```
from sklearn.preprocessing import LabelEncoder

lb_make = LabelEncoder()
obj_df["id_code"] = lb_make.fit_transform(obj_df["id"])
obj_df[["id", "id_code"]].head(11)
```

Out[506]:

	id	id_code
id	diagnosis	2
842302	M	1
842517	M	1
84300903	M	1
84348301	M	1
84358402	M	1
843786	M	1
844359	M	1
84458202	M	1
844981	M	1
84501001	M	1

In [507]:

```
print(obj_df)
```

```
                 id    diagnosis   radius_mean    texture_mean
perimeter_mean]
```

In [508]:

```
df=pd.read_csv('data 3.csv')
```

In [ ]:

In [ ]:

In [511]:

```
from sklearn.preprocessing import LabelEncoder
number=LabelEncoder()
df['diagnosis']=number.fit_transform(df['diagnosis'].astype('str'))
df.head(5)
```

```
4 84   1 2  1  1  1  0.  0.  0.  0.  . 1  1  1  0.  0.  0.  0.  0.  0.  N
  35     0. 4. 3  2  10  13  1   10  . 6. 5  5  1   2   4   1   2   07  a
  84     2  3  5. 9  03  28  9   43  . 6  2. 7  3   0   0   6   3   67  N
  02     9  4  1  7. 0   0   8   0     7  2  5. 7   5   0   2   6   8
               0  0          0            0  0  4   0   0   5   4
```

5 rows × 33 columns

In [512]:

```
headers =
["id","diagnosis","radius_mean","texture_mean","perimeter_mean","area_mean","
smoothness_mean","compactness_mean","concavity_mean","concave
points_mean","symmetry_mean","fractal_dimension_mean","radius_se","texture_se
","perimeter_se","area_se","smoothness_se","compactness_se","concavity_se","c
oncave
points_se","symmetry_se","fractal_dimension_se","radius_worst","texture_worst
","perimeter_worst","area_worst","smoothness_worst","compactness_worst","conc
avity_worst","concave
points_worst","symmetry_worst","fractal_dimension_worst"]

# Read in the CSV file and convert "?" to NaN
df = pd.read_csv('data 3.csv',header=None, names=headers, na_values="?" )
df.head()

from sklearn.preprocessing import LabelEncoder
number=LabelEncoder()
df['diagnosis']=number.fit_transform(df['diagnosis'].astype('str'))
df.head(5)
```

```
84   M 5 2  7  3  0.  0.  0.  0.  0.  . 2  9  5  0.  0.  0.  0.  0.  0.  N
34     3 0. 7. 8  1   2   2   1   2   . 6  8. 6  2   8   6   2   6   1   a
83       3  5  6. 4   8   4   0   5   . .  8  7. 0   6   8   5   6   7   N
01       8  8  1  2   3   1   5   9     5  7  7  9   6   6   7   3   3
                  5   9   4   2   7              8   3   9   5   8
```

5 rows × 32 columns

In [513]:

```
print(df)
```

```
                 id  diagnosis   radius_mean    texture_mean perimeter_mean
\
id        diagnosis        456  texture_mean  perimeter_mean      area_mean
```

In [ ]:

In [ ]:

In [528]:

```
headers =
["diagnosis","radius_mean","texture_mean","perimeter_mean","area_mean","smoot
hness_mean","compactness_mean","concavity_mean","concave
points_mean","symmetry_mean","fractal_dimension_mean","radius_se","texture_se
","perimeter_se","area_se","smoothness_se","compactness_se","concavity_se","c
oncave
points_se","symmetry_se","fractal_dimension_se","radius_worst","texture_worst
","perimeter_worst","area_worst","smoothness_worst","compactness_worst","conc
avity_worst","concave
points_worst","symmetry_worst","fractal_dimension_worst"]

# Read in the CSV file and convert "?" to NaN
df = pd.read_csv('data 3.csv',header=None, names=headers, na_values="?" )
df.head()

from sklearn.preprocessing import LabelEncoder
number=LabelEncoder()
df['diagnosis']=number.fit_transform(df['diagnosis'].astype('str'))
df.head(5)
```

Out[528]:

In [529]:

```
import pandas as pd
```

In [530]:

```
headers =
["id","diagnosis","radius_mean","texture_mean","perimeter_mean","area_mean","
smoothness_mean","compactness_mean","concavity_mean","concave
points_mean","symmetry_mean","fractal_dimension_mean","radius_se","texture_se
","perimeter_se","area_se","smoothness_se","compactness_se","concavity_se","c
oncave
points_se","symmetry_se","fractal_dimension_se","radius_worst","texture_worst
","perimeter_worst","area_worst","smoothness_worst","compactness_worst","conc
avity_worst","concave
points_worst","symmetry_worst","fractal_dimension_worst"]

# Read in the CSV file and convert "?" to NaN
df = pd.read_csv('data 3.csv',header=None, names=headers, na_values="?" )
df.head()

from sklearn.preprocessing import LabelEncoder
number=LabelEncoder()
df['diagnosis']=number.fit_transform(df['diagnosis'].astype('str'))
df.head(5)

print(df['radius_mean'])
```

In [ ]:

In [532]:

```
df['radius_mean'].plot(kind='hist',bins=50,figsize=(12,6))
```

Out[532]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a42f0c2b0>
```

In [533]:

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

In [534]:

```
df['radius_mean'].plot(kind='hist',bins=50,figsize=(12,6))
```

Out[534]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a40763828>
```

In [535]:

```
df['radius_mean'].plot(kind='hist',bins=50,figsize=(12,6))
```

Out[535]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a408525f8>
```

In [536]:

```
df['perimeter_mean'].plot(kind='hist',bins=50,figsize=(12,6))
```

Out[536]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a408fdda0>
```

In [537]:

```
df['texture_mean'].plot(kind='hist',bins=50,figsize=(12,6))
```

Out[537]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a43406780>
```

In [538]:

```
# 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)
```

In [539]:

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

In [540]:

```
from sklearn import preprocessing
```

In [541]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25,
random_state = 0)
```

In [542]:

```
headers =
["id","diagnosis","radius_mean","texture_mean","perimeter_mean","area_mean","
smoothness_mean","compactness_mean","concavity_mean","concave
points_mean","symmetry_mean","fractal_dimension_mean","radius_se","texture_se
","perimeter_se","area_se","smoothness_se","compactness_se","concavity_se","c
oncave
points_se","symmetry_se","fractal_dimension_se","radius_worst","texture_worst
","perimeter_worst","area_worst","smoothness_worst","compactness_worst","conc
avity_worst","concave
points_worst","symmetry_worst","fractal_dimension_worst"]

# Read in the CSV file and convert "?" to NaN
df = pd.read_csv('data 3.csv',header=None, names=headers, na_values="?" )
df.head()

from sklearn.preprocessing import LabelEncoder
number=LabelEncoder()
df['diagnosis']=number.fit_transform(df['diagnosis'].astype('str'))
df.head(5)
```

Out[542]:

In [543]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25,
random_state = 0)
```

In [ ]:

In [545]:

```
#defining the training and testing datasets
Y = df.diagnosis
X = df.drop('diagnosis', axis=1)
```

In [546]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25,
random_state = 0)
```

In [547]:

```
#defining the training and testing datasets
Y = df.diagnosis                      #label to be predicted
X = df.drop('diagnosis', axis=1)   #features used for prediction
```

In [548]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25,
random_state = 0)
```

In [ ]:

```
Name: diagnosis, Length: 143, dtype: int64
```

In [554]:

```
print(X_test.shape)
```

```
(143, 31)
```

In [555]:

```
print(Y_test.shape)
```

```
(143,)
```

In [556]:

```
print(X_train.shape)
```

```
(426, 31)
```

In [557]:

```
print(Y_train.shape)
```

```
(426,)
```

In [558]:

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

In [559]:

```
#Using Logistic Regression Algorithm to the Training Set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, Y_train)
```

```
/Users/sushmapriya/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

In [560]:

```
Y_pred = classifier.predict(X_test)
```

In [561]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)
```

In [562]:

```
print(cm)
```

```
[[87  3]
 [ 3 50]]
```

In [563]:

```
accuracy= (87+50)/(87+50+3+3)
print(accuracy)
```

```
0.958041958041958
```

In [564]:

```
accuracy_logisticreg= (87+50)/(87+50+3+3)*100
print("Accuracy from logistic regression :" ,accuracy_logisticreg)
```

```
Accuracy from logistic regression : 95.8041958041958
```

In [565]:

```
#Using KNeighborsClassifier Method of neighbors class to use Nearest Neighbor
algorithm
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p =
2)
classifier.fit(X_train, Y_train)
```

Out[565]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [566]:

```
cm = confusion_matrix(Y_test, Y_pred)
```

In [567]:

```
print(cm)
```

```
[[87  3]
 [ 3 50]]
```

In [568]:

```
Y_pred = classifier.predict(X_test)
```

In [569]:

```
cm = confusion_matrix(Y_test, Y_pred)
```

In [ ]:

In [570]:

```
print(cm)
```

```
[[89  1]
 [ 6 47]]
```

In [571]:

```
accuracy_KNN=(89+47)/(89+47+6+1) *100
```

In [572]:

```
print(accuracy_KNN)
```

```
95.1048951048951
```

In [573]:

```
#Using SVC method of svm class to use Kernel SVM Algorithm
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, Y_train)
```

Out[573]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=0,
    shrinking=True, tol=0.001, verbose=False)
```

In [574]:

```
Y_pred = classifier.predict(X_test)
```

In [575]:

```
cm = confusion_matrix(Y_test, Y_pred)
```

In [576]:

```
print(cm)
```

```
[[87  3]
 [ 3 50]]
```

In [577]:

```
cm1=confusion_matrix(Y_test, Y_pred)
```

In [578]:

```
print(cm1)
```

```
[[87  3]
 [ 3 50]]
```

In [579]:

```
Y_pred = classifier.predict(X_test)
```

In [580]:

```
cm1=confusion_matrix(Y_test, Y_pred)
```

In [581]:

```
print(cm1)
```

```
[[87  3]
 [ 3 50]]
```

In [582]:

```
accuracy_SVM= (87+50)/(87+50+3+3)*100
```

In [583]:

```
print("Accuracy with svm : ", accuracy_SVM)
```

```
Accuracy with svm :  95.8041958041958
```

In [584]:

```
#Using GaussianNB method of naïve_bayes class to use Naïve Bayes Algorithm
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, Y_train)
```

Out[584]:

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [585]:

```
Y_pred=classifier.predict(X_test)
```

In [586]:

```
cm=confusion_matrix(Y_test,Y_pred)
```

In [587]:

```
print(cm)
```

```
[[84  6]
 [ 6 47]]
```

In [588]:

```
accuracy_NB= (84+47)/(84+47+6+6)
```

In [589]:

```
print("Accuracy from Guassian Naive Bayes :", accuracy_NB)
```

```
Accuracy from Guassian Naive Bayes : 0.916083916083916
```

In [590]:

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, Y_train)
```

Out[590]:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy',
max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

In [ ]:

In [592]:

```
Y_pred=classifier.predict(X_test)
```

In [593]:

```
cm=confusion_matrix(Y_test,Y_pred)
```

In [594]:

```
print(cm)
```

```
[[85  5]
 [ 2 51]]
```

In [595]:

```
accuracy_DT= (85+21)/(85+21+5+5)*100
```

In [596]:

```
print("Accuracy from Decision Tree :", accuracy_DT)
```

```
Accuracy from Decision Tree : 91.37931034482759
```

In [597]:

```
#Using RandomForestClassifier method of ensemble class to use Random Forest
Classification algorithm

from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy',
random_state = 0)
classifier.fit(X_train, Y_train)
```

Out[597]:

```
RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='entropy',
                       max_depth=None, max_features='auto',
max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=0,
verbose=0,
                       warm_start=False)
```

In [598]:

```
Y_pred=classifier.predict(X_test)
```

In [599]:

```
cm=confusion_matrix(Y_test,Y_pred)
```

In [600]:

```
print(cm)
```

```
[[90  0]
 [ 2 51]]
```

In [601]:

```
accuracy_RF= (90+51)/(90+51+2+0)*100
```

In [602]:

```
print("Accuracy with Random forest :" ,accuracy_RF)
```

```
Accuracy with Random forest : 98.6013986013986
```

In [1]:

```
print("Highest accuracy is achieved with Random Forest for this dataset")
```

```
Highest accuracy is achieved with Random Forest for this dataset
```

In [ ]: