Breast Cancer Prediction

Problem Statement

Breast cancer became the major source of mortality between women. The accessibility of healthcare datasets and data analysis promote the researchers to apply study in extracting unknown pattern from healthcare datasets. The intention of this study is to design a prediction system that can predict the incidence of the breast cancer at early stage by analyzing smallest set of attributes that has been selected from the clinical dataset. Wisconsin breast cancer dataset (WBCD) have been used to conduct the proposed experiment. The potential of the proposed method is obtained using classification accuracy which was obtained by comparing actual to predicted values. This project is proposed with boosted accuracy to predict the breast cancer patient. The framework is composed of the following important phases:

- Dataset Selection
- Data Preprocessing
- Learning by Classifier (Training) i.e. Random Forest,Naive Bayes,Decision Tree SVC, Logistic Regression and KNN.
- Achieving trained model with highest accuracy
- Using trained model for prediction

Importing the libraries

In [55]:

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

Importing the dataset

In [56]:

```
dataset = pd.read_csv('C:/Users/User/Desktop/DataSets/Data_selection.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [57]:

```
dataset.head(10)
```

Out[57]:

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2
5	1017122	8	10	10	8	7	10	9	7	1	4
6	1018099	1	1	1	1	2	10	3	1	1	2
7	1018561	2	1	2	1	2	1	3	1	1	2
8	1033078	2	1	1	1	2	1	1	1	5	2
9	1033078	4	2	1	1	2	1	2	1	1	2

In [58]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 683 entries, 0 to 682
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Sample code number           683 non-null    int64
 1   Clump Thickness              683 non-null    int64
 2   Uniformity of Cell Size      683 non-null    int64
 3   Uniformity of Cell Shape     683 non-null    int64
 4   Marginal Adhesion            683 non-null    int64
 5   Single Epithelial Cell Size  683 non-null    int64
 6   Bare Nuclei                  683 non-null    int64
 7   Bland Chromatin              683 non-null    int64
 8   Normal Nucleoli              683 non-null    int64
 9   Mitoses                      683 non-null    int64
 10  Class                        683 non-null    int64
dtypes: int64(11)
memory usage: 58.8 KB
```

In [59]:

```
dataset.shape
```

Out[59]:

```
(683, 11)
```

In [60]:

```
dataset.describe()
```

Out[60]:

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	
count	6.830000e+02	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000	683.000000	683
mean	1.076720e+06	4.442167	3.150805	3.215227	2.830161	3.234261	3.544656	3.445095	2.869693	1.603221	2
std	6.206440e+05	2.820761	3.065145	2.988581	2.864562	2.223085	3.643857	2.449697	3.052666	1.732674	0
min	6.337500e+04	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	2
25%	8.776170e+05	2.000000	1.000000	1.000000	1.000000	2.000000	1.000000	2.000000	1.000000	1.000000	2
50%	1.171795e+06	4.000000	1.000000	1.000000	1.000000	2.000000	1.000000	3.000000	1.000000	1.000000	2
75%	1.238705e+06	6.000000	5.000000	5.000000	4.000000	4.000000	6.000000	5.000000	4.000000	1.000000	4
max	1.345435e+07	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	4

Splitting the dataset into the Training set and Test set

In [61]:

```
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 [62]:

```
print(X_train)
```

```
[[ 142932       7       6 ...       9      10       2]
 [1120559       8       3 ...       8       9       8]
 [1254538       8      10 ...      10      10       1]
 ...
 [1214092       1       1 ...       1       1       1]
 [1303489       3       1 ...       2       1       1]
 [ 378275      10       9 ...       7       7      1]]
```

In [63]:

```
print(y_train)
```

```
[4 4 4 2 2 2 4 2 2 4 4 2 4 2 2 4 4 2 2 2 2 2 2 2 2 2 2 4 2 2 2 2 2 4 4 2 4
 2 2 2 4 2 2 2 2 4 4 2 2 4 4 2 2 4 4 2 4 2 4 4 2 2 2 4 2 4 2 4 2 2 2 2 2 4
 2 2 4 2 2 4 2 2 2 2 2 4 2 2 4 2 4 2 2 4 4 4 2 2 2 2 2 2 4 4 2 2 2 2 2 2 2
 4 2 2 4 2 2 2 2 2 2 2 4 2 2 2 4 4 2 4 2 2 2 4 2 2 2 4 4 2 4 2 2 4 2 2 2 2
 2 2 2 4 4 4 4 2 4 2 4 2 4 4 4 2 2 4 2 2 2 2 4 4 2 2 2 4 2 2 4 2 2 2 2 4 4
 2 2 2 2 2 2 2 2 2 2 2 4 2 4 2 2 2 2 2 2 4 4 4 4 2 2 4 2 4 2 4 2 2 2 2 4 2
 4 2 2 2 2 4 2 2 4 2 2 2 2 2 2 2 4 2 4 2 2 2 4 2 2 2 2 2 4 2 2 2 2 4 2 2 4
 2 2 2 2 4 4 2 2 2 2 4 2 2 4 2 2 2 2 4 4 2 4 2 4 2 2 2 4 4 4 2 2 2 2 2 2 2
 2 4 4 2 2 2 2 2 2 2 4 4 2 2 2 2 4 4 4 2 4 2 4 2 2 2 2 2 4 2 4 4 2 2 2 2 2
 2 2 4 2 2 2 4 2 2 4 4 4 2 4 4 4 2 2 2 4 2 4 2 2 4 2 4 4 4 2 2 2 4 2 4 4 4
 2 2 2 4 2 4 2 2 2 2 4 4 2 2 2 4 4 2 2 4 2 2 2 4 4 2 2 2 4 2 2 2 4 2 4 2 2
 4 2 2 2 2 4 4 2 2 4 4 2 2 4 4 4 2 2 4 2 2 2 2 2 4 2 4 4 2 2 2 2 4 2 2 2 2
 4 2 4 2 4 2 2 4 2 2 2 2 4 2 2 2 4 2 2 4 4 4 4 2 4 4 2 4 4 2 2 2 2 2 2 4 2
 2 2 2 4 4 2 4 4 4 2 2 4 4 2 2 2 2 2 2 4 2 2 4 2 2 4 2 2 2 2 4]
```

In [64]:

```
print(X_test)
```

```
[[1173347       1       1 ...       1       1       1]
 [1156017       3       1 ...       2       1       1]
 [ 706426       5       5 ...       4       3       1]
 ...
 [ 764974       5       1 ...       3       1       2]
 [1137156       2       2 ...       7       1       1]
 [1160476       2       1 ...       3       1       1]]
```

In [65]:

```
print(y_test)
```

```
[2 2 4 4 2 2 2 4 2 2 4 2 4 2 2 2 4 4 4 2 2 2 4 2 4 4 2 2 2 4 2 4 4 2 2 2 4
 4 2 4 2 2 2 2 2 2 2 4 2 2 4 2 4 2 2 2 4 2 2 4 2 2 2 2 2 2 2 2 4 4 2 2 2 2
 2 2 4 2 2 2 4 2 4 2 2 4 2 2 4 2 4 2 4 4 4 2 4 4 4 2 2 2 4 4 2 2 4 4 2 2 4
 2 2 4 2 2 2 4 2 2 2 4 2 2 4 4 2 4 2 4 2 2 4 2 2 4 2 4 2 2 2 4 2 2 2 2 4 2
 4 2 4 4 2 2 2 2 4 4 2 4 4 4 4 4 4 2 2 2 2 2 2]
```

Feature Scaling

In [66]:

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

In [67]:

```
#print(X_train)
```

In [68]:

```
#print(X_test)
```

Training the Logistic Regression model on the Training set

In [69]:

```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(random_state = 0)
```

```
LR = LogisticRegression(random_state = 0)
LR.fit(X_train, y_train)
```

Out[69]:

```
LogisticRegression(random_state=0)
```

Predicting the Test set results

In [70]:

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

Training the K-NN model on the Training set

In [71]:

```
from sklearn.neighbors import KNeighborsClassifier
knn= KNeighborsClassifier(n_neighbors= 5, metric= 'minkowski', p = 2)
knn.fit(X_train, y_train)

y_pred1 = knn.predict(X_test)
```

Training the Kernel SVC model on the Training set

In [72]:

```
from sklearn.svm import SVC
svc= SVC(kernel='rbf', random_state= 0) # Or Kernel= 'linear' for linear kernels
svc.fit(X_train, y_train)

y_pred2 = svc.predict(X_test)
```

Training the Naive Bayes model on the Training set

In [73]:

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train, y_train)

y_pred3 = nb.predict(X_test)
```

Training the Decision Tree model on the Training set

In [74]:

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

y_pred4 = dtc.predict(X_test)
```

Training the Random Forest model on the Training set

In [78]:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=10 , random_state= 0)
rfc.fit(X_train, y_train)

y_pred5 = rfc.predict(X_test)
```

```
y_pred5 = rfc.predict(X_test)
```

Making the Confusion Matrix

In [80]:

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred) # for Logistic Regression
cm1 = confusion_matrix(y_test, y_pred1) # for KNN
cm2 = confusion_matrix(y_test, y_pred2) # for SVC
cm3 = confusion_matrix(y_test, y_pred3) # for Naive Bayes
cm4 = confusion_matrix(y_test, y_pred4) # for Decision Tree
cm5 = confusion_matrix(y_test, y_pred5) # for Random Forest

print("Accuracy score for LR: {} and KNN: {} and SVC:{} and Naive_Bayes:{} and Decision_Tree:{} an
d Random_forest:{}".format(accuracy_score(y_test, y_pred),accuracy_score(y_test, y_pred1),accuracy_
score(y_test, y_pred2),
accuracy_score(y_test, y_pred3),accuracy_score(y_test, y_pred4),accuracy_score(y_test, y_pred5)))
```

```
Accuracy score for LR: 0.9473684210526315 and KNN: 0.9473684210526315 and SVC:0.9532163742690059 a
nd Naive_Bayes:0.9415204678362573 and Decision_Tree:0.9590643274853801 and
Random_forest:0.9473684210526315
```

Observation:

</p> **We can see that Decision Tree model best fits the data more accurately with an accuracy score of 0.9590643274853801**

In [ ]:

```

```