Program Code: J620-002-4:2020

Program Name: FRONT-END SOFTWARE DEVELOPMENT

Title : Exe22 - Bagging and Boosting Exercise

Name: Chong Mun Chen

IC Number: 960327-07-5097

Date : 20/7/2023

Introduction : Practising on ensemble learning methods bagging and boosting.

Conclusion : Achieved the accuracy scores for the decision tree, random forest, bagging, and boosting methods. Each accuracy score is different from the other.

Bagging and Boosting Exercise

Reference: (https://www.datacamp.com/community/tutorials/ensemble-learning-python (https://www.datacamp.com/community/tutorials/ensemble-learning-python))

Bagging Method

In [5]:
```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
from sklearn.preprocessing import MinMaxScaler
```

In [6]:
```
import pandas as pd
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/breast
                   header=None)
data.columns = ['Sample code', 'Clump Thickness', 'Uniformity of Cell Size', 'Unifor
                'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei',
                'Normal Nucleoli', 'Mitoses','Class']

data = data.drop(['Sample code'],axis=1)
print('Number of instances = %d' % (data.shape[0]))
print('Number of attributes = %d' % (data.shape[1]))
data.head()
```

```
Number of instances = 699
Number of attributes = 10
```

Out[6]:

	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	5	1	1	1	2	1	3	1	1	2
1	5	4	4	5	7	10	3	2	1	2
2	3	1	1	1	2	2	3	1	1	2
3	6	8	8	1	3	4	3	7	1	2
4	4	1	1	3	2	1	3	1	1	2

In [7]:
```
data.describe()
```

Out[7]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bland Chromatin	Normal Nucleoli	Mitoses
count	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000
mean	4.417740	3.134478	3.207439	2.806867	3.216023	3.437768	2.866953	1.589413
std	2.815741	3.051459	2.971913	2.855379	2.214300	2.438364	3.053634	1.715078
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	2.000000	1.000000	1.000000	1.000000	2.000000	2.000000	1.000000	1.000000
50%	4.000000	1.000000	1.000000	1.000000	2.000000	3.000000	1.000000	1.000000
75%	6.000000	5.000000	5.000000	4.000000	4.000000	5.000000	4.000000	1.000000
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000

In [8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 10 columns):
 #   Column                       Non-Null Count  Dtype 
---  ------                       --------------  ----- 
 0   Clump Thickness              699 non-null    int64 
 1   Uniformity of Cell Size      699 non-null    int64 
 2   Uniformity of Cell Shape     699 non-null    int64 
 3   Marginal Adhesion            699 non-null    int64 
 4   Single Epithelial Cell Size  699 non-null    int64 
 5   Bare Nuclei                  699 non-null    object
 6   Bland Chromatin              699 non-null    int64 
 7   Normal Nucleoli              699 non-null    int64 
 8   Mitoses                      699 non-null    int64 
 9   Class                        699 non-null    int64 
dtypes: int64(9), object(1)
memory usage: 54.7+ KB
```

In [9]:
```
data['Bare Nuclei']
```

Out[9]:
```
0       1
1      10
2       2
3       4
4       1
       ..
694     2
695     1
696     3
697     4
698     5
Name: Bare Nuclei, Length: 699, dtype: object
```

In [10]:
```
data.replace('?',0, inplace=True)
data['Bare Nuclei']
```

Out[10]:
```
0       1
1      10
2       2
3       4
4       1
       ..
694     2
695     1
696     3
697     4
698     5
Name: Bare Nuclei, Length: 699, dtype: object
```

In [11]:
```
# Convert the DataFrame object into NumPy array otherwise you will not be able to in
values = data.values

# Now impute it

imputedData = imputer.fit_transform(values)
```

In [12]:
```
scaler = MinMaxScaler(feature_range=(0, 1))
normalizedData = scaler.fit_transform(imputedData)
```

In [25]:
```
# Bagged Decision Trees for Classification - necessary dependencies
from sklearn import model_selection
from sklearn.model_selection import train_test_split, cross_val_score, RepeatedStra
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_classification
from numpy import mean
from numpy import std

bag_clf = BaggingClassifier()
```

In [18]:
```
# Segregate the features from the labels
X = data.drop(('Class'), axis=1)
y = data['Class']
```

In [19]:
```
# accuracy score
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(bag_clf, X, y, scoring='accuracy', cv=cv, n_jobs=-1, erro
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy: 0.952 (0.025)
```

Boosting Method

In [26]:
```
from sklearn.ensemble import AdaBoostClassifier
seed = 7
num_trees = 70
kfold = model_selection.KFold(n_splits=10, random_state=seed, shuffle=True)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, y, cv=kfold)
print(results.mean())
```

```
0.9599378881987578
```

Exercise 1 Perform classification using the Titanic dataset using the classifiers that you already know (Dtree and RF)

In [27]:
```
#Preprocessing the entire Titanic dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

t_dataset = pd.read_csv(r'../data_samples2/titanic.csv')
t_dataset
```

Out[27]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500
...	...	...	...	...	...	...	...	...
882	0	2	Rev. Juozas Montvila	male	27.0	0	0	13.0000
883	1	1	Miss. Margaret Edith Graham	female	19.0	0	0	30.0000
884	0	3	Miss. Catherine Helen Johnston	female	7.0	1	2	23.4500
885	1	1	Mr. Karl Howell Behr	male	26.0	0	0	30.0000
886	0	3	Mr. Patrick Dooley	male	32.0	0	0	7.7500

887 rows × 8 columns

In [28]:
```
#drop name column
titanic_df = t_dataset.drop(('Name'), axis=1)
```

In [29]:
```
#encode categorical data into numerical value
from sklearn import preprocessing

new_sex = {'male': 1, 'female': 0}
titanic_df['Sex'] = titanic_df['Sex'].replace(new_sex)
titanic_df
```

Out[29]:

	Survived	Pclass	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	1	22.0	1	0	7.2500
1	1	1	0	38.0	1	0	71.2833
2	1	3	0	26.0	0	0	7.9250
3	1	1	0	35.0	1	0	53.1000
4	0	3	1	35.0	0	0	8.0500
...	...	...	...	...	...	...	...
882	0	2	1	27.0	0	0	13.0000
883	1	1	0	19.0	0	0	30.0000
884	0	3	0	7.0	1	2	23.4500
885	1	1	1	26.0	0	0	30.0000
886	0	3	1	32.0	0	0	7.7500

887 rows × 7 columns

In [30]:
```
#create a copy of the cleaned dataset
t_df = titanic_df.copy()
```

In [31]:
```
#define dependent variable and independent variable
X = t_df.drop(('Survived'), axis=1)
y = t_df.Survived

print(X.values)
print(y.values)
```

```
[[ 3.      1.     22.      1.      0.      7.25  ]
 [ 1.      0.     38.      1.      0.     71.2833]
 [ 3.      0.     26.      0.      0.      7.925 ]
 ...
 [ 3.      0.      7.      1.      2.     23.45  ]
 [ 1.      1.     26.      0.      0.     30.    ]
 [ 3.      1.     32.      0.      0.      7.75  ]]
[0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 0 0 1 0 0 1 1 0 0 0 1
 0 0 1 0 0 1 1 0 0 1 0 0 0 0 1 1 0 1 1 0 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 1
 0 0 0 1 1 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 1 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0
 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0
 1 1 0 0 1 0 1 1 1 1 0 0 1 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 0 0
 0 1 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 1 1
 1 0 0 0 0 0 1 1 1 0 1 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 1 1 1 1 0 0 0 0 0 0
 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 0 0 1 1 0 1 1 0 0 1 1 0 1 0 1 1 1 1 0 0 0 1
 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 1 1 1 1 0
 0 0 0 1 1 0 0 0 1 1 0 1 0 0 0 1 0 1 1 1 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 1 0
 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 1 1 0 1 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1
 1 1 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1
 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 1 1 0 1 1 0 0 1 0 1 0 1
 0 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0 0 1 1 0 1 0
 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 1 1 0 1 1 1 0 0 0 1 0 1 0 0 0 1 0 0 0
 0 1 0 0 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1 0 0 1 0
 1 0 0 1 0 0 0 0 0 1 0 1 1 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0
 1 1 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0 1 1 0 0 0 0
 1 1 1 1 1 0 1 0 0 0 1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 1 0 1 0
 1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1
 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0
 0 0 1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0
 1 1 1 1 0 0 0 1 0 0 1 1 0 0 1 0 1 0 0 1 1 0 0 0 1 1 0 0 0 0 0 0 1 0 1 0]
```

In [32]:
```
#Split the dataset into the Training and the Test set. Set the test set to 0.3
from sklearn.model_selection import train_test_split
np.random.seed(42)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

In [33]:

```
# Decision Tree object
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()

# Train Decision Tree Classifier
clf = clf.fit(X_train, y_train)

# Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct
acc_score = accuracy_score(y_test, y_pred)

print(y_pred)
print("Accuracy:", acc_score)
```

```
[1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0
 1 1 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 1 1 0 1 0 1 1 0 0 1 0 0 1 1 1 0
 1 1 1 0 1 0 0 0 0 0 1 1 1 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 1
 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0
 0 0 0 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0 1 1 1 0 0 0 1 1 1 0 1
 1 0 0 0 1 0 1 0 0 1 0 1 0 1 0 1 1 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0
 0 1 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 1 0 0 0 1 1 0
 1 0 0 1 0 0 1 0]
Accuracy: 0.7715355805243446
```

In [34]:
```
# Random forest
from sklearn.ensemble import RandomForestClassifier

# Create the model with 100 trees
rf_clf = RandomForestClassifier(n_estimators=100)

# Fit on training data
rf_clf.fit(X_train, y_train)

# Actual class predictions
y_pred_rf = rf_clf.predict(X_test)
print(y_pred_rf)

# Probabilities for each class
probabilities = rf_clf.predict_proba(X_test)
print(probabilities.flatten())

acc_score_rf = accuracy_score(y_test, y_pred_rf)
print("Accuracy:", acc_score_rf)
```

```
[1 0 0 0 0 1 0 0 1 0 1 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0
 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1 0
 0 1 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0 1
 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0
 0 0 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 0 1 0 1 1 1 1 0 0 1 1 0 0 0 0 1 1 0 1
 1 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 1 0 1 0 0 1 1 0 0 1 0 0 0 1 1 0
 1 0 1 0 0 0 1 0]
[0.42       0.58       0.92       0.08       0.93       0.07
 1.         0.         0.9        0.1        0.45       0.55
 0.69083333 0.30916667 1.         0.         0.39       0.61
 0.64       0.36       0.09       0.91       0.02       0.98
 0.43       0.57       0.92       0.08       0.62       0.38
 0.91916667 0.08083333 1.         0.         0.         1.
 0.7805     0.2195     0.83333333 0.16666667 0.97       0.03
 0.07       0.93       0.71       0.29       0.775      0.225
 0.76       0.24       0.91       0.09       0.99       0.01
 0.88       0.12       0.94       0.06       0.88       0.12
 0.5        0.5        0.64       0.36       0.44       0.56
 0.68       0.32       0.16       0.84       0.7        0.3
 0.8825     0.1175     0.11       0.89       0.         1.
 0.07       0.93       0.05       0.95       1.         0.
 0.         1.         1.         0.         0.9        0.1
 0.95       0.05       0.94       0.06       1.         0.
 0.98       0.02       0.97       0.03       0.98       0.02
 0.99       0.01       0.78       0.22       1.         0.
 0.21       0.79       0.80046429 0.19953571 0.41       0.59
 0.18       0.82       0.12       0.88       0.24       0.76
 0.92       0.08       0.16       0.84       0.89       0.11
 0.03       0.97       0.65       0.35       0.915      0.085
 0.99       0.01       0.11       0.89       0.9355     0.0645
 0.98       0.02       0.68333333 0.31666667 0.04       0.96
 0.14       0.86       0.99       0.01       0.64333333 0.35666667
 0.13       0.87       0.04       0.96       0.61       0.39
 0.04       0.96       1.         0.         0.98       0.02
 0.95633333 0.04366667 0.92       0.08       0.965      0.035
 0.07       0.93       0.01       0.99       0.61       0.39
 0.965      0.035      0.68333333 0.31666667 0.85       0.15
 0.19866667 0.80133333 0.25       0.75       0.99       0.01
 0.2        0.8        0.954      0.046      0.42       0.58
 0.98133333 0.01866667 1.         0.         0.03       0.97
 0.11       0.89       0.86       0.14       0.97283333 0.02716667
 0.32833333 0.67166667 0.02       0.98       1.         0.
 1.         0.         0.7        0.3        0.54       0.46
 0.95       0.05       0.81       0.19       0.05       0.95
 0.78       0.22       1.         0.         0.79       0.21
 0.94916667 0.05083333 0.96333333 0.03666667 0.47       0.53
 0.45       0.55       0.86       0.14       0.707      0.293
 0.48       0.52       0.78060714 0.21939286 0.67       0.33
 0.96       0.04       0.43       0.57       0.88       0.12
 0.91       0.09       0.         1.         0.03       0.97
 0.06       0.94       0.9        0.1        0.32       0.68
 0.955      0.045      0.98       0.02       0.90116667 0.09883333
 0.66333333 0.33666667 0.99       0.01       0.99       0.01
 0.8825     0.1175     0.78       0.22       0.         1.
 0.96       0.04       0.32       0.68       0.78       0.22
 0.46       0.54       0.02       0.98       0.92       0.08
 0.79716667 0.20283333 0.99       0.01       1.         0.
```

```
 0.9        0.1        0.19       0.81       0.63       0.37
 0.93       0.07       0.89       0.11       0.14       0.86
 0.94       0.06       0.1        0.9        0.24       0.76
 0.9675     0.0325     0.98       0.02       0.37       0.63
 1.         0.         0.69       0.31       0.92       0.08
 0.03       0.97       0.77       0.23       0.05       0.95
 0.93       0.07       0.04       0.96       0.04       0.96
 0.         1.         0.42683333 0.57316667 0.96       0.04
 0.61       0.39       0.22       0.78       0.1        0.9
 1.         0.         0.70588095 0.29411905 0.94       0.06
 0.63       0.37       0.         1.         0.04       0.96
 1.         0.         0.073      0.927      0.45       0.55
 1.         0.         0.95       0.05       0.75       0.25
 0.77       0.23       0.84       0.16       0.01       0.99
 1.         0.         0.96333333 0.03666667 0.18       0.82
 0.85       0.15       0.57       0.43       0.85       0.15
 0.03       0.97       0.63083333 0.36916667 0.02       0.98
 0.03       0.97       0.19       0.81       0.9775     0.0225
 0.88       0.12       0.75       0.25       0.742      0.258
 1.         0.         0.         1.         0.66       0.34
 0.64       0.36       0.76       0.24       0.91333333 0.08666667
 0.05       0.95       0.         1.         0.99666667 0.00333333
 0.79       0.21       0.82       0.18       0.18       0.82
 0.78       0.22       0.93       0.07       1.         0.
 1.         0.         0.68333333 0.31666667 0.92       0.08
 0.74       0.26       1.         0.         0.94083333 0.05916667
 0.99       0.01       0.93       0.07       0.94       0.06
 0.65       0.35       0.45       0.55       0.49       0.51
 0.30716667 0.69283333 0.42       0.58       0.45233333 0.54766667
 0.43       0.57       0.98       0.02       0.687      0.313
 0.75       0.25       0.91       0.09       0.67       0.33
 0.05       0.95       0.99       0.01       0.25       0.75
 0.99       0.01       0.97283333 0.02716667 0.02       0.98
 0.01       0.99       0.95333333 0.04666667 0.93130952 0.06869048
 0.02       0.98       0.925      0.075      0.97       0.03
 0.99       0.01       0.01       0.99       0.45       0.55
 0.80046429 0.19953571 0.41       0.59       0.89       0.11
 0.38       0.62       0.54       0.46       0.91       0.09
 0.63       0.37       0.16       0.84       0.99       0.01      ]
Accuracy: 0.7940074906367042
```

Exercise 2 Perform classification using the Titanic dataset using the classifiers that you already know and with feature selection and dimension reduction. Which gives you the best result?

In [37]:
```
#StandardScaler
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_scaled = sc.fit_transform(X)

#PCA & Pick up from the point where dataset has been split
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
X_pca

print(pca.explained_variance_ratio_)

#Decision Tree object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.3, random_
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
print(y_pred)

# Model Accuracy, how often is the classifier correct?
acc_score = accuracy_score(y_test, y_pred)
print("Accuracy:", acc_score)
```

```
[0.29874926 0.29350524]
[0 1 0 1 0 1 0 0 0 0 1 1 1 0 0 0 0 1 1 0 0 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 0
 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0
 0 1 1 0 1 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 1 1 0 0 0 1 0 1 1
 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 0 1 0 1 0 1
 1 0 0 0 0 0 1 1 0 1 1 0 0 1 0 1 0 1 0 0 0 1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 1
 1 0 0 1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 0 0 0 0 0 1 0 1 0 1 1 0 0 1 0 1 1 0 0
 0 0 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0
 0 0 0 1 0 1 1 0]
Accuracy: 0.6966292134831461
```

In [38]:
```
#rebuild analytical dataset & create a copy of the cleaned dataset
titanic_df
t_df = titanic_df.copy()

#define dependent variable and independent variable
X = t_df.drop(('Survived'), axis=1)
y = t_df.Survived

#Split the dataset into the Training and the Test set. Set the test set to 0.3
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

In [39]:
```
# RF Feature Selector
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score

# Create a random forest classifier (10000 trees)
rf_clf = RandomForestClassifier(n_estimators=10000)

# Train the classifier
rf_clf.fit(X_train, y_train)

# Create a selector object that will use the random forest classifier to identify
# features that have an importance of more than 0.15
selector = SelectFromModel(rf_clf, threshold = 0.15)

# Train the selector
selector.fit(X_train, y_train)

# Transform the data to create a new dataset containing only the most important feat
# Note: We have to apply the transform to both the training X and test X data.
important_feature_indices = selector.get_support(indices=True)
important_feature_names = X.columns[important_feature_indices]

# Create a new random forest classifier for the most important features
new_rf_clf = RandomForestClassifier(n_estimators=10000)

# Train the new classifier on the new dataset containing the most important features
X = t_df[important_feature_names]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
new_rf_clf.fit(X_train, y_train)

# Apply The Limited Classifier To The Test Data
y_pred = new_rf_clf.predict(X_test)

# View The Accuracy Of Our Limited Feature (2 Features) Model
acc_score = accuracy_score(y_test, y_pred)
print("Accuracy:", acc_score)
```

```
Accuracy: 0.7715355805243446
```

Exercise 3 Perform classification using the Titanic dataset using bagging and boosting (choose 1 bagging and 1 boosting algo)

In [42]:
```
# Bagging
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import BaggingClassifier

model = BaggingClassifier(base_estimator=KNeighborsClassifier())

titanic_df
t_df = titanic_df.copy()

#define dependent variable and independent variable
X = t_df.drop(('Survived'), axis=1)
y = t_df.Survived

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy: 0.710 (0.044)
```

In [44]:
```
# Boosting
#create a copy of the cleaned dataset
from xgboost import XGBClassifier

titanic_df
t_df = titanic_df.copy()

#define dependent variable and independent variable
X = t_df.drop(('Survived'), axis=1)
y = t_df.Survived

#Split the dataset into the Training and the Test set. Set the test set to 0.3
np.random.seed(42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Apply Xgboost
model = XGBClassifier()

#fit model
model.fit(X_train, y_train)

# make predictions for test data
y_pred = model.predict(X_test)

# evaluate predictions
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.797752808988764
```

Out of all 3 approaches, which gives you the best result?

XGBoost classification gives me the best result out of all 3 approaches.