Forward School

Program Code: J620-002-4:2020

Program Name: FRONT-END SOFTWARE DEVELOPMENT

Title : Exe22 - Bagging and Boosting Exercise

Name: Ooi Caaron

IC Number: 990701-07-5837

Date :21/7/23

Introduction : Decision Tree algorithm partitions the data into subsets by repeatedly asking questions about the features of the data points.

Conclusion : Still need to practice more and do revision

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 [3]:

```
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 [4]:

```
import pandas as pd
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/breast-can
                   header=None)
data.columns = ['Sample code', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity
                'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Blan
                '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[4]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mito:
0	5	1	1	1	2	1	3	1	
1	5	4	4	5	7	10	3	2	
2	3	1	1	1	2	2	3	1	
3	6	8	8	1	3	4	3	7	
4	4	1	1	3	2	1	3	1	

In [3]:

```
data.describe()
```

Out[3]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bland Chromatin	Normal Nucleoli
count	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
std	2.815741	3.051459	2.971913	2.855379	2.214300	2.438364	3.053634
min	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
50%	4.000000	1.000000	1.000000	1.000000	2.000000	3.000000	1.000000
75%	6.000000	5.000000	5.000000	4.000000	4.000000	5.000000	4.000000
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000

In [5]:

```
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 [6]:

```
data['Bare Nuclei']
```

Out[6]:

```
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 [7]:

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

Out[7]:

```
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 [8]:

```
# Convert the DataFrame object into NumPy array otherwise you will not be able to impute
values = data.values

# Now impute it

imputedData = imputer.fit_transform(values)
```

In [9]:

```
scaler = MinMaxScaler(feature_range=(0, 1))
normalizedData = scaler.fit_transform(imputedData)
```

In [10]:

```
# Bagged Decision Trees for Classification - necessary dependencies
from sklearn.datasets import make_classification
X,y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5
print(X.shape, y.shape)
from numpy import mean
from numpy import std
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import BaggingClassifier
# define dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=
# define the model
model = BaggingClassifier()
# evaluate the model
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_scor
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
(1000, 20) (1000,)
Accuracy: 0.859 (0.041)
```

In [11]:

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

In [14]:

```
from sklearn.ensemble import BaggingRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import RepeatedKFold

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_score=
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy: 0.956 (0.019)
```

Boosting Method

In [20]:

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn import model_selection
seed = 7
num_trees = 70
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
Cell In[20], line 6
      3 seed = 7
      4 num_trees = 70
----> 6 kfold = model_selection.KFold(n_splits=10, random_state=seed)
      7 model = AdaBoostClassifier(n_estimators=num_trees, random_state=se
ed)
      8 results = model_selection.cross_val_score(model, X, Y, cv=kfold)

File ~\anaconda3\envs\python-dscourse\Lib\site-packages\sklearn\model_sele
ction\_split.py:476, in KFold.__init__(self, n_splits, shuffle, random_sta
te)
    475 def __init__(self, n_splits=5, *, shuffle=False, random_state=Non
e):
--> 476     super().__init__(n_splits=n_splits, shuffle=shuffle, random_st
ate=random_state)

File ~\anaconda3\envs\python-dscourse\Lib\site-packages\sklearn\model_sele
ction\_split.py:331, in _BaseKFold.__init__(self, n_splits, shuffle, rando
m_state)
    328     raise TypeError("shuffle must be True or False; got {0}".forma
t(shuffle))
    330 if not shuffle and random_state is not None:  # None is the defaul
t
--> 331     raise ValueError(
    332         (
    333             "Setting a random_state has no effect since shuffle is
"
    334             "False. You should leave "
    335             "random_state to its default (None), or set shuffle=Tr
ue."
    336         ),
    337     )
    339 self.n_splits = n_splits
    340 self.shuffle = shuffle

ValueError: Setting a random_state has no effect since shuffle is False. Y
ou should leave random_state to its default (None), or set shuffle=True.
```

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

In [17]:

```
#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'titanic.csv')
t_dataset
```

Out[17]:

	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 [21]:

```
#drop name column
t_dataset.drop('Name', axis=1, inplace=True)
t_dataset
```

Out[21]:

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

887 rows × 7 columns

In [22]:

```
#encode categorical data into numerical value
from sklearn import preprocessing
reset = {'male':1, 'female':2}
t_dataset = t_dataset.replace({'Sex':reset})
t_dataset
```

Out[22]:

	Survived	Pclass	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	1	22.0	1	0	7.2500
1	1	1	2	38.0	1	0	71.2833
2	1	3	2	26.0	0	0	7.9250
3	1	1	2	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	2	19.0	0	0	30.0000
884	0	3	2	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 [23]:

```
#create a copy of the cleaned dataset
t_datacopy = t_dataset.copy()
t_datacopy
```

Out[23]:

	Survived	Pclass	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	1	22.0	1	0	7.2500
1	1	1	2	38.0	1	0	71.2833
2	1	3	2	26.0	0	0	7.9250
3	1	1	2	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	2	19.0	0	0	30.0000
884	0	3	2	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 [25]:

```
#define dependent variable and independent variable
dependent_variable = 'Survived'
independent_variables = ['Pclass', 'Sex', 'Age', 'Siblings/Spouses Aboard', 'Parents/Chi
X = t_datacopy[independent_variables].values
y = t_datacopy[dependent_variable].values
print(X)
print(y)
```

```
[[ 3.       1.      22.       1.       0.       7.25  ]
 [ 1.       2.      38.       1.       0.      71.2833]
 [ 3.       2.      26.       0.       0.       7.925 ]
 ...
 [ 3.       2.       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 1 1 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 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 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 [27]:

```
#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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

In [34]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
#Decision Tree object
model = DecisionTreeClassifier()

# Train Decision Tree Classifer
model.fit(X_train, y_train)

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

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

```
[0 1 1 1 0 1 0 0 1 1 0 1 0 0 1 0 1 1 1 0 1 1 1 1 0 1 1 0 1 0 0 0 0 0 1 1 0
 1 1 0 1 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 1
 1 0 0 1 1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0
 1 0 1 0 0 0 1 1 1 1 0 0 0 0 1 0 1 1 0 0 0 1 1 0 1 0 0 0 1 1 0 0 0 0 1 1 1
 0 0 1 1 1 0 1 1 0 0 0 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 1 1 1 1 1 0 0 0 0 1
 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 0 1 1 0 1
 1 0 0 0 1 1 0 0 1 0 0 0 1 1 0 1 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 0 0 1 0 1
 1 0 0 0 1 1 0 1]
Accuracy: 0.7565543071161048
```

In [38]:

```
#Random forest
from sklearn.ensemble import RandomForestClassifier

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

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

# Actual class predictions
y_pred = model.predict(X_test)

# Probabilities for each class
pro = model.predict_proba(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(y_pred)
print(pro.flatten())
print(accuracy)
```

```
[0 1 1 1 0 1 0 0 1 1 0 0 0 0 1 0 1 1 1 0 1 1 1 0 0 1 1 0 1 0 0 0 0 0 1 0 0
 1 1 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 1 0 0 1 0 0 1
 1 0 0 1 1 1 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 1 0 0 1 1 0 0
 0 0 1 0 0 0 0 1 1 1 0 0 0 0 1 0 1 1 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 0 1 1 1
 1 0 1 1 1 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 1 0 0 1 1 1 1 1 0 0 0 0 1
 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 0 1 0 1 1 0 0
 1 0 0 0 1 0 0 1 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 1 0 1
 1 0 0 0 1 0 0 1]
[0.99333333 0.00666667 0.12166667 0.87833333 0.09       0.91
 0.02       0.98       0.7        0.3        0.         1.
 0.87       0.13       0.90816667 0.09183333 0.01       0.99
 0.24       0.76       0.94       0.06       0.52       0.48
 1.         0.         0.9        0.1        0.4        0.6
 1.         0.         0.         1.         0.0825     0.9175
 0.         1.         0.6        0.4        0.31       0.69
 0.41       0.59       0.08       0.92       0.57       0.43
 0.95       0.05       0.11       0.89       0.13       0.87
 0.99       0.01       0.         1.         0.75       0.25
 0.97333333 0.02666667 0.74       0.26       0.67       0.33
 0.89       0.11       0.24       0.76       0.9        0.1
 0.96       0.04       0.15       0.85       0.         1.
 0.97       0.03       0.2        0.8        0.21       0.79
 1.         0.         0.85       0.15       1.         0.
 0.79       0.21       0.12766667 0.87233333 0.99       0.01
 1.         0.         0.88       0.12       0.29       0.71
 0.41       0.59       0.97       0.03       0.98       0.02
 0.75966667 0.24033333 0.97       0.03       0.4775     0.5225
 0.82       0.18       1.         0.         0.3        0.7
 0.31       0.69       0.99       0.01       0.78666667 0.21333333
 0.97833333 0.02166667 1.         0.         1.         0.
 0.05       0.95       0.31       0.69       0.86       0.14
 0.99       0.01       0.16       0.84       0.63       0.37
 0.76       0.24       0.08       0.92       0.34       0.66
 0.99       0.01       0.98166667 0.01833333 0.12       0.88
 0.15       0.85       0.34       0.66       0.86       0.14
 0.98166667 0.01833333 0.4        0.6        0.55       0.45
 0.07       0.93       0.83       0.17       1.         0.
 0.63       0.37       0.47       0.53       0.69       0.31
 0.61       0.39       0.99       0.01       1.         0.
 0.16       0.84       0.96       0.04       0.63       0.37
 0.01       0.99       0.98       0.02       0.005      0.995
 0.87       0.13       0.35       0.65       0.58       0.42
 0.98       0.02       0.74       0.26       0.31       0.69
 0.84       0.16       0.9        0.1        0.         1.
 0.38       0.62       0.66       0.34       1.         0.
 0.87       0.13       0.99       0.01       0.15       0.85
 0.53066667 0.46933333 0.97       0.03       0.74       0.26
 0.66       0.34       0.035      0.965      0.25       0.75
 0.07       0.93       0.76       0.24       0.52       0.48
 0.79       0.21       0.56       0.44       0.36       0.64
 0.98       0.02       0.02       0.98       0.01       0.99
 0.8625     0.1375     1.         0.         0.83       0.17
 0.03       0.97       0.76       0.24       0.8        0.2
 0.285      0.715      0.81583333 0.18416667 1.         0.
 1.         0.         0.16       0.84       0.01       0.99
 0.96666667 0.03333333 0.97       0.03       0.99       0.01
 0.929      0.071      0.08       0.92       0.03       0.97
 0.09       0.91       0.4        0.6        0.85       0.15
 0.12       0.88       0.06       0.94       0.23       0.77
 0.55       0.45       0.57       0.43       0.01       0.99
 0.97       0.03       0.72033333 0.27966667 0.76666667 0.23333333
```

```
 0.58       0.42       0.978      0.022      0.61       0.39
 0.95       0.05       0.02       0.98       0.1        0.9
 0.88       0.12       0.96       0.04       0.92       0.08
 0.03       0.97       0.91       0.09       1.         0.
 0.23       0.77       0.01       0.99       1.         0.
 1.         0.         0.04       0.96       0.02       0.98
 0.3        0.7        0.34       0.66       0.         1.
 0.94       0.06       0.89583333 0.10416667 0.98       0.02
 0.79       0.21       0.24       0.76       1.         0.
 0.84       0.16       0.99       0.01       0.5        0.5
 0.58       0.42       0.09       0.91       0.95666667 0.04333333
 0.94       0.06       0.08       0.92       0.55       0.45
 0.96       0.04       0.2815     0.7185     0.06       0.94
 0.89       0.11       1.         0.         0.6        0.4
 0.99       0.01       0.36       0.64       0.97833333 0.02166667
 0.14666667 0.85333333 0.62       0.38       0.07       0.93
 0.29       0.71       0.989      0.011      0.74       0.26
 0.95       0.05       0.97       0.03       1.         0.
 0.99       0.01       0.894      0.106      0.9        0.1
 0.14       0.86       0.83       0.17       0.17       0.83
 0.         1.         0.9575     0.0425     0.7        0.3
 0.06       0.94       0.74       0.26       0.99       0.01
 1.         0.         0.14       0.86       0.52       0.48
 0.54       0.46       0.34       0.66       0.09       0.91
 0.978      0.022      0.94       0.06       0.9        0.1
 0.01       0.99       0.04       0.96       1.         0.
 0.13       0.87       0.43       0.57       0.3        0.7
 1.         0.         0.52       0.48       0.655      0.345
 0.13       0.87       0.98       0.02       0.89       0.11
 0.01       0.99       0.97       0.03       0.4        0.6
 0.19       0.81       0.79       0.21       0.96       0.04
 0.92       0.08       0.08       0.92       1.         0.
 0.87       0.13       0.16       0.84       1.         0.
 0.27       0.73       0.         1.         0.79       0.21
 0.64       0.36       0.74       0.26       0.11       0.89
 0.79       0.21       0.98       0.02       0.         1.        ]
0.7752808988764045
```

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 [35]:

```
#StandardScaler
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
clf = DecisionTreeClassifier()
clf.fit(X_train_pca, y_train)
y_pred = clf.predict(X_test_pca)
explained_variance_ratio = pca.explained_variance_ratio_
print(explained_variance_ratio)
print(y_pred)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

```
[0.31208962 0.29640549]
[0 0 0 1 0 1 0 0 1 1 0 0 0 0 0 0 1 1 0 1 1 1 1 0 0 0 1 0 1 0 0 0 1 0 1 1 0
 1 1 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1 1 0 1
 1 0 0 0 1 1 1 0 1 1 1 0 0 0 1 0 0 0 0 1 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0 1 0
 0 0 1 0 0 1 0 1 1 0 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1
 0 0 0 1 1 1 0 1 0 0 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 0 1 1 1 0 1 0 0 0 0 1
 0 1 0 0 0 1 0 0 1 0 0 1 1 0 0 0 1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1
 0 0 0 0 1 0 1 1 1 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 1 0 1
 1 1 1 0 1 0 0 1]
0.6891385767790262
```

In [25]:

```
#rebuild analytical dataset & create a copy of the cleaned dataset


#define dependent variable and independent variable


#Split the dataset into the Training and the Test set. Set the test set to 0.3
```

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

from sklearn.feature_selection import SelectFromModel
num_trees = 10000
model = RandomForestClassifier(n_estimators=num_trees)
# Create a selector object that will use the random forest classifier to identify
# features that have an importance of more than 0.15
feature_selector = SelectFromModel(estimator=model, threshold=0.15)
# Train the selector
feature_selector.fit(X_train, y_train)
# Transform the data to create a new dataset containing only the most important features
# Note: We have to apply the transform to both the training X and test X data.
X_train_selected = feature_selector.transform(X_train)
X_test_selected = feature_selector.transform(X_test)
# Create a new random forest classifier for the most important features
model_selected = RandomForestClassifier(n_estimators=num_trees)
# Train the new classifier on the new dataset containing the most important features
model_selected.fit(X_train_selected, y_train)
# Apply the limited classifier to the test data
y_pred_selected = model_selected.predict(X_test_selected)
# View the accuracy of our limited feature (selected features) model
accuracy_selected = accuracy_score(y_test, y_pred_selected)
print("Accuracy with selected features:", accuracy_selected)
```

```
Accuracy with selected features: 0.7453183520599251
```

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

In [42]:

```
!pip install xgboost
```

```
0:00:18
     ---                                    5.4/70.9 MB 3.7 MB/s eta
0:00:18
     ---                                    5.5/70.9 MB 3.7 MB/s eta
0:00:18
     ---                                    5.6/70.9 MB 3.6 MB/s eta
0:00:19
     ---                                    5.7/70.9 MB 3.5 MB/s eta
0:00:19
     ---                                    5.9/70.9 MB 3.5 MB/s eta
0:00:19
     ---                                    5.9/70.9 MB 3.5 MB/s eta
0:00:19
     ---                                    6.0/70.9 MB 3.5 MB/s eta
0:00:19
     ---                                    6.1/70.9 MB 3.4 MB/s eta
0:00:19
     ---                                    6.3/70.9 MB 3.4 MB/s eta
0:00:20
     ---                                    6.4/70.9 MB 3.4 MB/s eta
```

In [43]:

```
from xgboost import XGBClassifier
#create a copy of the cleaned dataset
model = XGBClassifier()
#define dependent variable and independent variable
model.fit(X_train, y_train)
#Split the dataset into the Training and the Test set. Set the test set to 0.3
# Apply Xgboost

#fit model

# make predictions for test data
y_pred = model.predict(X_test)
# evaluate predictions
predictions = [round(value) for value in y_pred]
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 79.40%
```

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