Data Mining (Δ02): Exercise Set 2: 2.1

Name: Nefeli Eleftheria Sextou

Student ID: 503

E-mail: pcs00503@uoi.gr, nsekstou@cs.uoi.gr

In [1]:
```
#general
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#data preprocessing
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MaxAbsScaler
from sklearn.utils import shuffle

#classifiers
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


#to ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

Data Preprocessing

This is done in the same manner as in the first excercise set. The preprocessing strategy is explained in detail in the corresponding ipynb file.

In [2]:
```
#LOAD DATA-----------------------------------------

# Load the xlsx file without the header
data = pd.read_excel(r"C:\Users\Nefeli\Desktop\dm_msc\\DM_Homework2_2024\Dataset_503_2.x
# Generate column names based on the column index
col_names = [f'f_{i}' for i in range(len(data.columns))]

# Assing the generated column names to the column names on the dataframe
data.columns = col_names

# ENCODE LABELS-------------------------------------

encoder = LabelEncoder()
data['f_13']=encoder.fit_transform(data['f_13'])

#FEATURE-TARGET SPLIT------------------------------

X = data.iloc[:, :13].copy() # feature data
y = data['f_13'].copy() #target variable - class/category column

#SCALE DATA----------------------------------------
```

```
scaler = MaxAbsScaler()
X_scaled = scaler.fit_transform(X)

#SHUFFLE DATA-----------------------------------------

# shuffle the data rows
X_shuffled, y_shuffled = shuffle(X_scaled, y, random_state=42)

#Train-Test split
X_train, X_test, y_train, y_test = train_test_split(X_shuffled, y_shuffled, test_size=0.
```

Bagging with Decesion Trees

In [3]:
```
#number of estimators
n_est= [25, 50, 75, 100]

#results dictionary
bagging_results = {}
bagging_results_err={}

# train and eval
for n in n_est:

    bagging_clf = BaggingClassifier(base_estimator=DecisionTreeClassifier(min_samples_le
    bagging_clf.fit(X_train, y_train)

    bagging_results_err[n] = 1-bagging_clf.oob_score_
    bagging_results[n] = bagging_clf.oob_score_


print("Bagging Results (OOB scores):", bagging_results)
print("Bagging Results (OOB error):", bagging_results_err)
```

```
Bagging Results (OOB scores): {25: 0.79125, 50: 0.8425, 75: 0.8725, 100: 0.88}
Bagging Results (OOB error): {25: 0.20875, 50: 0.15749999999999997, 75: 0.12749999999999
995, 100: 0.12}
```

Random Forest

In [4]:
```
#number of estimators
n_est = [25, 50, 75, 100]

#results dictionary
random_forest_results = {}
random_forest_results_err = {}

#train and eval
for n in n_est:

    rf_clf = RandomForestClassifier(n_estimators=n, min_samples_leaf=5, oob_score=True,
    rf_clf.fit(X_train, y_train)

    random_forest_results_err[n] = 1-rf_clf.oob_score_
    random_forest_results[n]=rf_clf.oob_score_



print("Random Forest Results (OOB Score):", random_forest_results)
print("Random Forest Results (OOB Error):", random_forest_results_err)
```

```
Random Forest Results (OOB Score): {25: 0.69, 50: 0.75625, 75: 0.775, 100: 0.79}
```

```
Random Forest Results (OOB Error): {25: 0.31000000000000005, 50: 0.24375000000000002, 7
5: 0.22499999999999998, 100: 0.20999999999999996}
```

Both classifiers' mean out-of-bag (OOB) error increases as the number of estimators is decreased. A higher OOB error signifies worse performance. Both classifiers perform their best for **number of estimators = 100** with the Bagging classifier having the highest OOB Score of **0.88** and the lowest OOB Error of **0.12**.

Note: **oob_score_** is not the OOB error but the OOB score which is calculated using the samples that are not used in the training of the model (out-of-bag samples). These samples are used to provide an unbiased estimate of the model's performance. It is equal to the Accuracy score and is used to evaluate the model's generalization capability. To obtain the OOB Error calculate: *1- OOB Score* . A high OOB Score (a low OOB error) is equivalent to a high accuracy score.

In comparison to results obtained for the same dataset in the first excercise set, both these implementations, at their best found parameterization, perform significantly better.

In [ ]: