Modify the Bagging scratch code in our lecture such that:

- Calculate for oob evaluation for each bootstrapped dataset, and also the average score
- Change the code to "without replacement"
- Put everything into a class `Bagging`. It should have at least two methods, `fit(X_train, y_train)`, and `predict(X_test)`
- Modify the code from above to randomize features. Set the number of features to be used in each tree to be `sqrt(n)`, and then select a subset of features for each tree. This can be easily done by setting our DecisionTreeClassifier `max_features` to 'sqrt'

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [95]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score


iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
                test_size=0.3, shuffle=True, random_state=42)
```

In [162]:
```
class Bagging:

    def __init__(self, boostrap_ratio, without_replacement = True, B = 5):
        self.B = B
        self.boostrap_ratio = boostrap_ratio
        self.without_replacement = without_replacement
        self.tree_params = {'max_depth': 2, 'criterion':'gini', 'min_samples_split': 5}
        self.models = [DecisionTreeClassifier(**self.tree_params) for _ in range(B)]

    def fit(self, X_train, y_train):
        m, n = X_train.shape

        #sample size for each tree
        sample_size = int(self.boostrap_ratio * len(X_train))

        xsamples = np.zeros((B, sample_size, n))
        ysamples = np.zeros((B, sample_size))


        xoob = []
        yoob = []

        #subsamples for each model
        for i in range(self.B):
            idxes = []
            oob_idxes = []
            ##sampling with replacement; i.e., sample can occur more than once
            #for the same predictor
            for j in range(sample_size):
                idx = random.randrange(m)    #<----with replacement #change so no repetition

                # If there is replacement create indices of oob for future
                if (self.without_replacement):
                    # If idx is already in oob_idxs, redo
                    while idx in oob_idxes:
                        idx = random.randrange(m)
                idxes.append(idx)
                oob_idxes.append(idx)
                xsamples[i, j, :] = X_train[idx]
                ysamples[i, j] = y_train[idx]
                    #keep track of idx that i did not use for ith tree

            # if bootstrap is less than 0, we don't take the whole X
            # Here I create an array the size of X with False values
            mask = np.zeros((m), dtype=bool)
            # oob_idx will not have all X's m values, so we are creating a mask that will select indice
s we have
            mask[oob_idxes] = True # mask = [True, False, False, True ... etc]
            # Here flip the incides
            x_mask = X_train[~mask]
            y_mask = y_train[~mask]
            xoob.append(x_mask)
            yoob.append(y_mask)


        #fitting each estimator
        oob_score = 0
        total_score = 0
        for i, model in enumerate(self.models):
            _X = xsamples[i, :]
            _y = ysamples[i, :]
            model.fit(_X, _y)

            # calculating oob score here
            # oob can now be used as a validation set! Because the model didn't see it
            _X_test = np.asarray(xoob[i])
            _y_test = np.asarray(yoob[i])
            _yhat = model.predict(_X_test)
            acc_score = accuracy_score(_y_test, _yhat)
            total_score += acc_score
            print(f"Accuracy score in three {i}: {acc_score}")
        # calculate total oob score! finally!!
        total_oob_score = total_score / len(models)
        print(f"Total oob score is: {total_oob_score}")

    def predict(self, X_test):
        #make prediction and return the probabilities
        predictions = np.zeros((self.B, X_test.shape[0]))
        for i, model in enumerate(self.models):
            yhat = model.predict(X_test)
            predictions[i, :] = yhat

        yhat = stats.mode(predictions)[0][0]
        return yhat
```

In [165]:
```
model = Bagging(boostrap_ratio = 0.7, without_replacement=True, B = 5)

model.fit(X_train, y_train)
yhat = model.predict(X_test)
```

```
Accuracy score in three 0: 0.9375
Accuracy score in three 1: 0.84375
Accuracy score in three 2: 0.9375
Accuracy score in three 3: 0.75
Accuracy score in three 4: 0.96875
Total oob score is: 0.8875
```

In [166]:
```
print(classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

In [85]:
```
# Test
aaa = np.array([1,2,3,4,5])
bbb = np.array([1,2,3,4,5])
print(~aaa)
```

```
[-2 -3 -4 -5 -6]
```

In [97]:
```
oob_idx = [96, 67, 0, 91, 38, 26, 93, 48, 32, 6, 40, 14, 63, 78, 12, 36, 94, 4, 56, 53, 5, 41, 51, 72,
44, 19, 46, 34, 33, 43, 74, 92, 61, 85, 57, 22, 2, 30, 24, 82, 65, 70, 27, 95, 97, 76, 29, 39, 83, 80,
18, 73, 101, 103, 13, 47, 54, 3, 64, 99, 20, 81, 69, 7, 23, 89, 37, 98, 59, 45, 77, 11, 35]

mask = np.zeros((m), dtype=bool)
mask[oob_idx] = True
xxx = X_train[~mask]

print(xxx.shape)
```

```
(32, 4)
```

In [ ]:
```

```