Modify the AdaBoost scratch code in our lecture such that:

- Notice that if `err` = 0, then α will be undefined, thus attempt to fix this by adding some very small value to the lower term
- Notice that sklearn version of AdaBoost has a parameter `learning_rate`. This is in fact the $\frac{1}{2}$ in front of the α calculation. Attempt to change this $\frac{1}{2}$ into a parameter called `eta`, and try different values of it and see whether accuracy is improved. Note that sklearn default this value to 1.
- Observe that we are actually using sklearn DecisionTreeClassifier. If we take a look at it closely, it is actually using weighted gini index, instead of weighted errors that we learn above. Attempt to write your own class of `class Stump` that actually uses weighted errors, instead of weighted gini index. To check whether your stump really works, it should give you still relatively the same accuracy. In addition, if you do not change y to -1, it will result in very bad accuracy. Unlike sklearn version of DecisionTree, it will STILL work even y is not change to -1 since it uses gini index
- Put everything into a class

In [1]:
```
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

X, y = make_classification(n_samples=500, random_state=1)
y = np.where(y==0,-1,1)  #change our y to be -1 if it is 0, otherwise 1

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

In [33]:
```
class Stump():
    def __init__(self):
        self.polarity = 1
        self.feature_index = None
        self.threshold = None
        self.alpha = None
```

In [42]:
```
class AdaBoost():

    def __init__(self, S = 5, eta = 0.5):
        self.S = S
        self.eta = eta

    def fit(self, X_train, y_train):
        #initially, we set our weight to 1/m
        m, n = X_train.shape
        W = np.full(m, 1/m)

        # Try learning rates (or clf?)
        self.classifiers = []



        for _ in range(self.S):
            classifier = Stump()

            # Minimum error to infinity to identify first "stump?"
            min_err = np.inf

            # Loop through features n
            for feature in range(n):
                feature_vals = np.sort(np.unique(X_train[:, feature]))
                tresholds = (feature_vals[:-1] + feature_vals[1:]) / 2
                for treshold in tresholds:
                    for pol in [1, -1]:
                        yhat = np.ones(len(y_train))
                        yhat[pol * X_train[:, feature] < pol * treshold] = - 1
                        err = W[(yhat != y_train)].sum()

                    # Savethe best error
                    if err < min_err:
                        classifier.polarity = pol
                        classifier.treshold = treshold
                        classifier.feature_index = feature
                        min_err = err

            #compute the predictor weight a_j
            error_fix = 0.00000000000001
            classifier.alpha = self.eta * (np.log ((1 - err) / err + error_fix) / 2)
            self.classifiers.append(classifier)

            #update sample weight; divide sum of W to normalize
            W = (W * np.exp(-classifier.alpha * y_train * yhat))
            W = W / sum (W)

    def predict(self, X_test):
        m, n = X_test.shape
        yhat = np.zeros(m)
        for clf in self.classifiers:
            pred = np.ones(m)
            pred[clf.polarity * X_test[:, clf.feature_index] < clf.polarity * clf.treshold] = -1
            yhat = clf.alpha * pred
        return np.sign(yhat)
```

In [43]:
```
model = AdaBoost(eta = 0.5)
model.fit(X_train, y_train)
yhat = model.predict(X_test)
print("========================= Done =========================")
# print(yhat)
print(classification_report(y_test, yhat))
```

```
========================= Done =========================
              precision    recall  f1-score   support

          -1       0.95      0.23      0.37        79
           1       0.53      0.99      0.69        71

    accuracy                           0.59       150
   macro avg       0.74      0.61      0.53       150
weighted avg       0.75      0.59      0.52       150
```

In [44]:
```
# Test stuff

test = np.array([5, 1, 7, 4, 9])
test2 = np.array([5, 1, 7, 4, 2])
testsorted = np.sort(test)
print(testsorted)

print(testsorted[1:])
print(testsorted[:-1])
z = testsorted[:-1]+testsorted[1:]
print(z)
```

```
[1 4 5 7 9]
[4 5 7 9]
[1 4 5 7]
[ 5  9 12 16]
```

In [24]:
```
print(test < test2)
```

```
[False False False False False]
```