```
#perceptron

import numpy as np


class Perceptron:
    def __init__(self, max_iters=100):
        self.max_iters = max_iters

    def fit(self, X, y):
        X, y = np.asarray(X), np.asarray(y)
        iters = 0

        X = np.concatenate((X, np.asarray([[1] * X.shape[0]]).T), axis=1)

        ω = np.random.random(X.shape[1])

        for _ in range(self.max_iters):
            y_pred_all = []
            for idx in range(X.shape[0]):
                x_sample, y_sample = X[idx], y[idx]
                y_pred = int(np.sum(ω * x_sample) >= 0.5)
                if y_pred == y_sample:
                    pass
                elif y_pred == 0 and y_sample == 1:
                    ω = ω + x_sample
                elif y_pred == 1 and y_sample == 0:
                    ω = ω - x_sample

                y_pred_all.append(y_pred)

            iters += 1
            if np.equal(np.array(y_pred_all), y).all():
                break

        self.iters, self.ω = iters, ω

    def predict(self, X):
        X = np.asarray(X)
        X = np.concatenate((X, np.asarray([[1] * X.shape[0]]).T), axis=1)

        return (X @ self.ω > 0.5).astype(int)


clf = Perceptron()
clf.fit([[1], [2], [3]], [0, 0, 1])


clf.iters
```

```
     6


clf.predict([[1], [2], [3]])


     array([0, 0, 1])


clf = Perceptron()
clf.fit([[1], [2], [3]], [0, 1, 0])


clf.iters

     100


clf.predict([[1], [2], [3]])


     array([1, 0, 0])


#adaline

def adaline(outputs, weights, bias):
    total_error = 1
    counter = 0
    while total_error != 0 and counter < 10:

        total_error = 0
        counter += 1
        for i in range(len(outputs)):
            sum = INPUTS[i].dot(weights) + bias
            prediction = step_function(sum)

            total_error += outputs[i] - prediction
            error = outputs[i] - sum

            if outputs[i] != prediction:
                weights[0] = weights[0] + (LEARNING_RATE * error * INPUTS[i][0])
                weights[1] = weights[1] + (LEARNING_RATE * error * INPUTS[i][1])
                bias = bias + (LEARNING_RATE * error)
                print("Weight updated: " + str(weights[0]))
                print("Weight updated: " + str(weights[1]))
                print("Bias updated`: " + str(bias))
                print("----------------------------------------")

        print("Total error: " + str(total_error))
        print("----------------------------------------")
```

```
return weights, bias
```