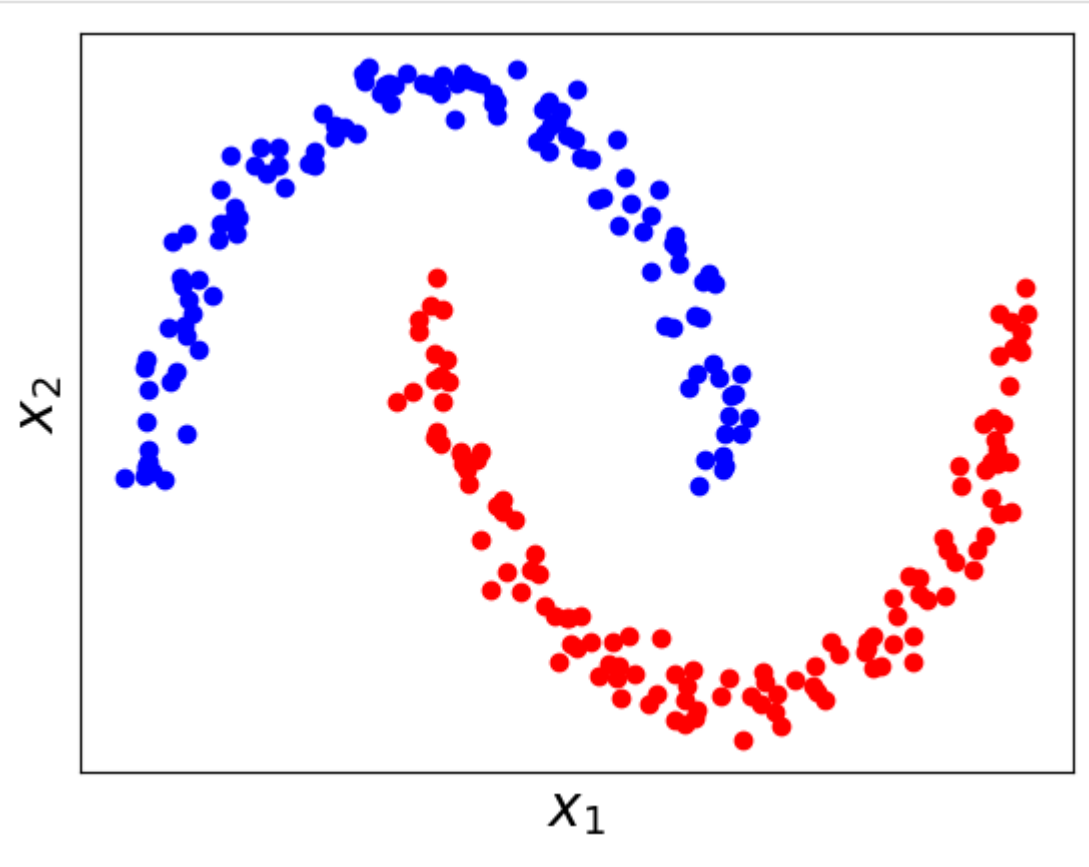
In [ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```python
#getting the dataset
from sklearn.datasets import make_moons

X,y  = make_moons(n_samples=250,noise=0.05,random_state=42)
y = 2*y -1
```

In [ ]:
```python
# plotting the data

plt.plot(X[:, 0][y==1], X[:, 1][y==1], "ro")
plt.plot(X[:, 0][y!=1], X[:, 1][y!=1], "bo")

plt.xlabel(r"$x_1$", fontsize=20)
plt.ylabel(r"$x_2$", fontsize=20)


plt.xticks([])
plt.yticks([])

plt.show()
```



AdaBoost

For $m = 1$ to M

1. Select and extract from the pool of classifiers the classifier k_m which minimizes
$$W_e = \sum_{y_i \neq k_m(x_i)} w_i^{(m)}$$
2. Set the weight α_m of the classifier to
$$\alpha_m = \frac{1}{2}\ln\left(\frac{1-e_m}{e_m}\right)$$
where $e_m = W_e/W$
3. Update the weights of the data points for the next iteration. If $k_m(x_i)$ is a miss, set
$$w_i^{(m+1)} = w_i^{(m)} e^{\alpha_m} = w_i^{(m)} \sqrt{\frac{1-e_m}{e_m}}$$
otherwise
$$w_i^{(m+1)} = w_i^{(m)} e^{-\alpha_m} = w_i^{(m)} \sqrt{\frac{e_m}{1-e_m}}$$

Setting the weights for each datapoint

In [ ]:
```python
w = [1/len(X) for _ in X]
```

We will be using decision stumps (Decision Trees with depth = 1) as weak learners

In [ ]:
```python
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:
```python
def calculate_error(y, y_pred, weights):
    # Function to calculte the error for a particular*prediction
    I = np.not_equal(y,y_pred).astype(int)
    #print(I)
    return (sum(weights * I))/sum(weights)
```

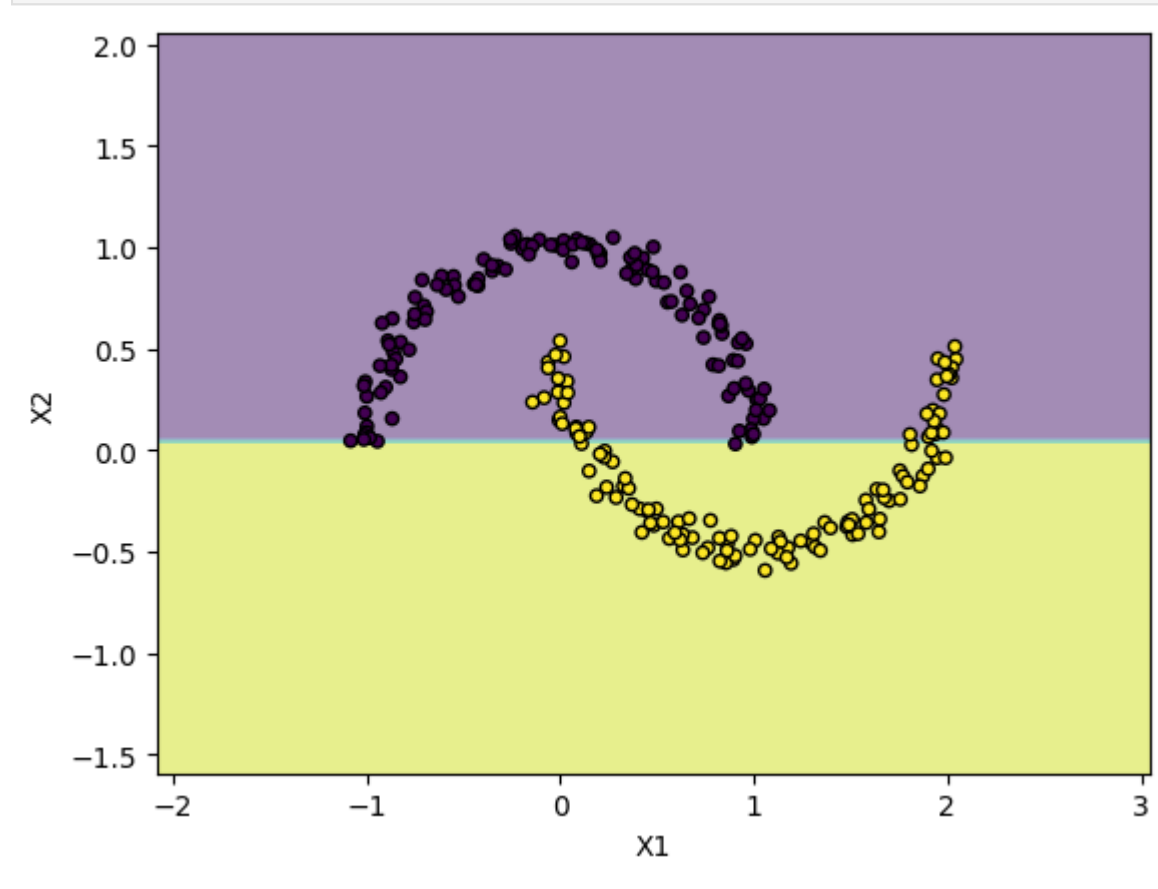
In [ ]:
```python
def calculate_alpha(error):
    # Function to Calculate the weight of a weak classifier in the majority vote of the final classifier (alpha)
    return 0.5 * np.log((1 - error) / error)
```

In [ ]:
```python
def update_weights(alpha, weights, y , y_pred):
    # function to update the weights after each iteration
    I = np.not_equal(y,y_pred)
    return weights * np.exp(alpha * (np.not_equal(y, y_pred)).astype(int))
```

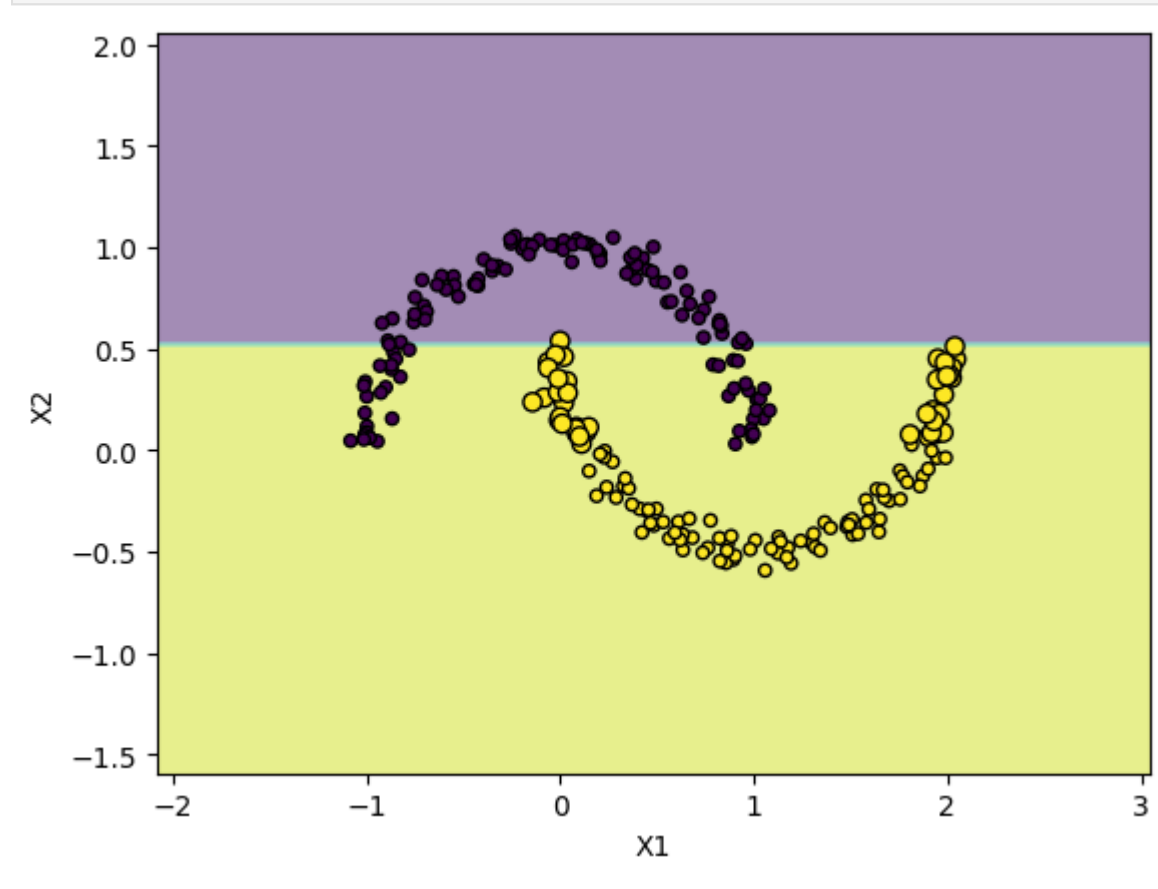
For the purpose of this tutorial, we will consider 10 rounds of boosting

In [ ]:
```python
classifiers = []
weights = []
alphas = []
#print(w)
weights.append(w)
num_iters = 100

for i in range(num_iters):
    # fit a weak classifier (decision stump) to the weighted dataset
    clf = DecisionTreeClassifier(max_depth=1)
    clf.fit(X, y, sample_weight = w)
    y_pred = clf.predict(X)

    classifiers.append(clf)

    error = calculate_error(y,y_pred,w)

    alpha = calculate_alpha(error)

    w = update_weights(alpha, w, y, y_pred)

    weights.append(w)
    alphas.append(alpha)
```

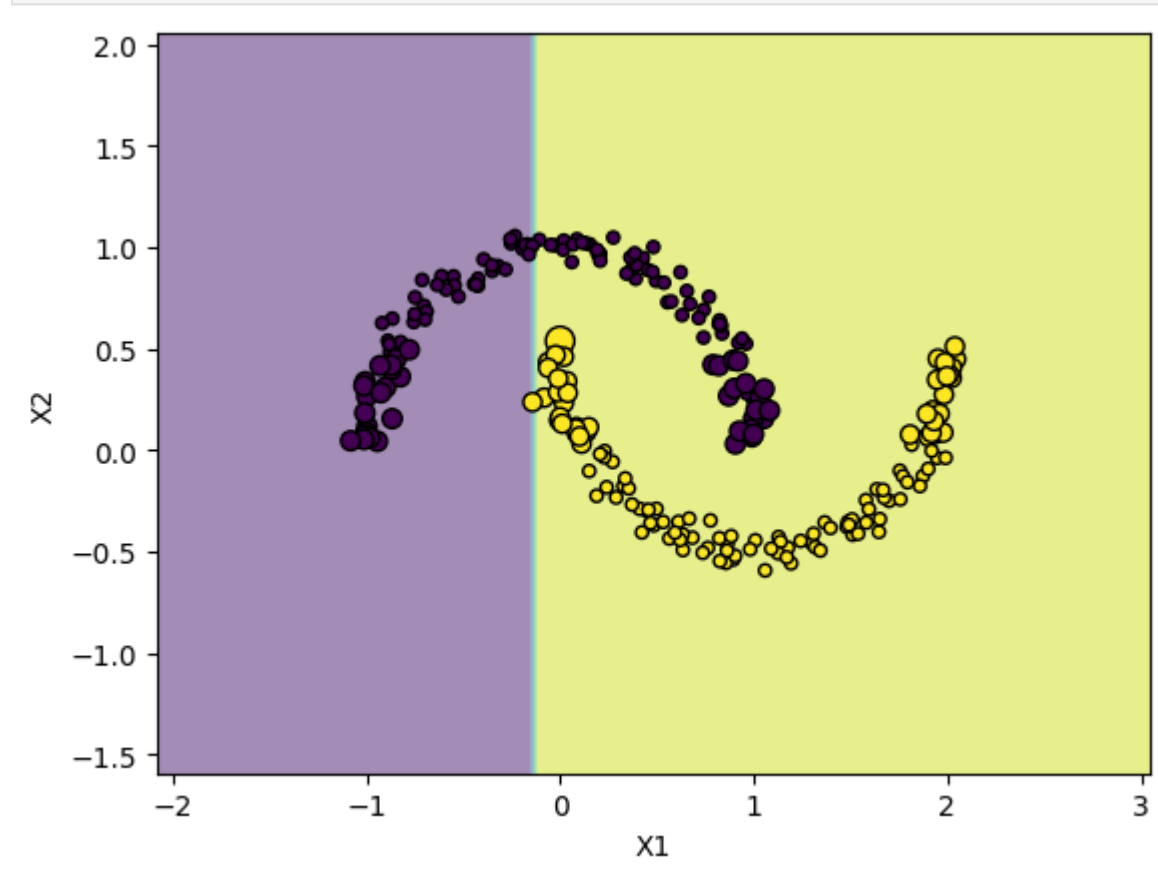
In [ ]:
```python
#visualize the nth classifier and weights
n = 0
from sklearn.inspection import DecisionBoundaryDisplay
disp = DecisionBoundaryDisplay.from_estimator(classifiers[n], X,
                                              response_method="predict",
                                              xlabel='X1', ylabel='X2',
                                              alpha=0.5,                                     )
disp.ax_.scatter(X[:, 0], X[:, 1],s = np.array(weights[n])*5000 ,c=y, edgecolor="black")

plt.show()
```



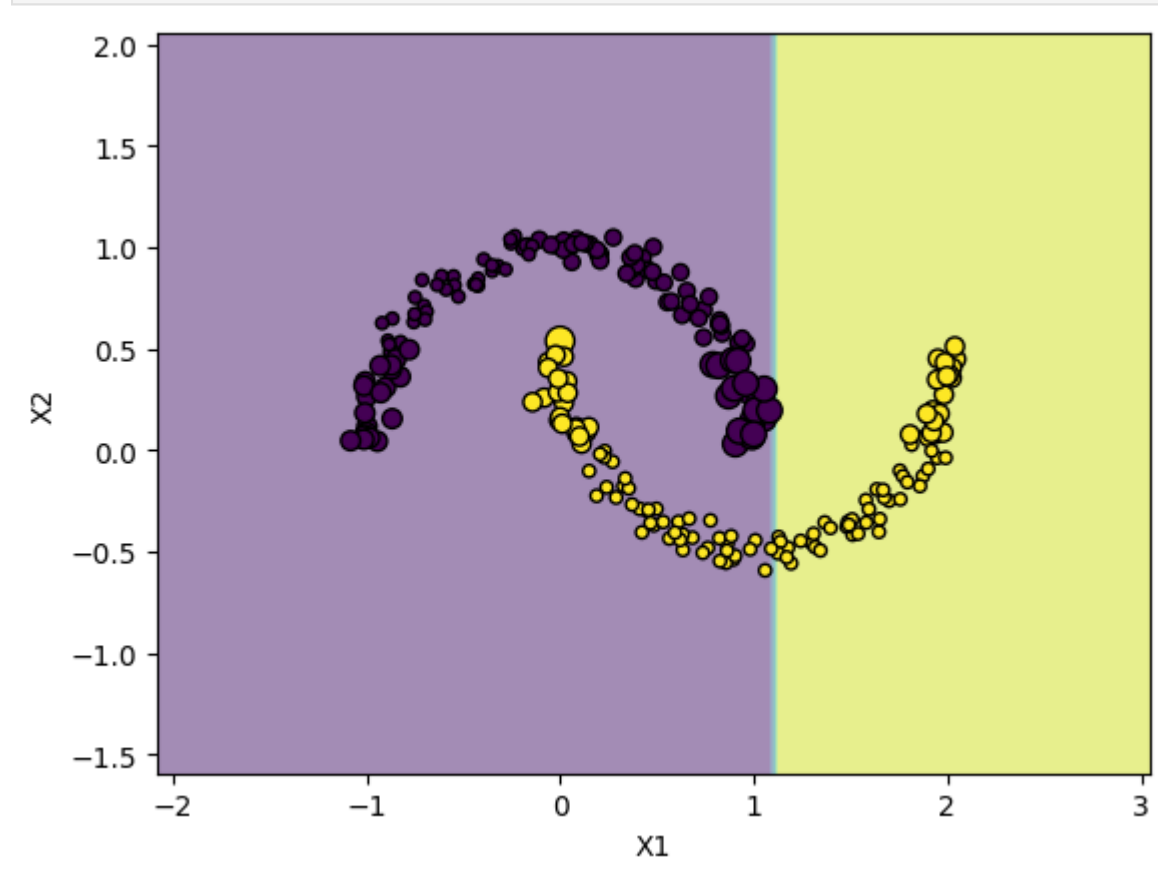
In [ ]:
```python
#visualize the nth classifier and weights
n = 1
from sklearn.inspection import DecisionBoundaryDisplay
disp = DecisionBoundaryDisplay.from_estimator(classifiers[n], X,
                                              response_method="predict",
                                              xlabel='X1', ylabel='X2',
                                              alpha=0.5,                                     )
disp.ax_.scatter(X[:, 0], X[:, 1],s = np.array(weights[n])*5000 ,c=y, edgecolor="black")

plt.show()
```



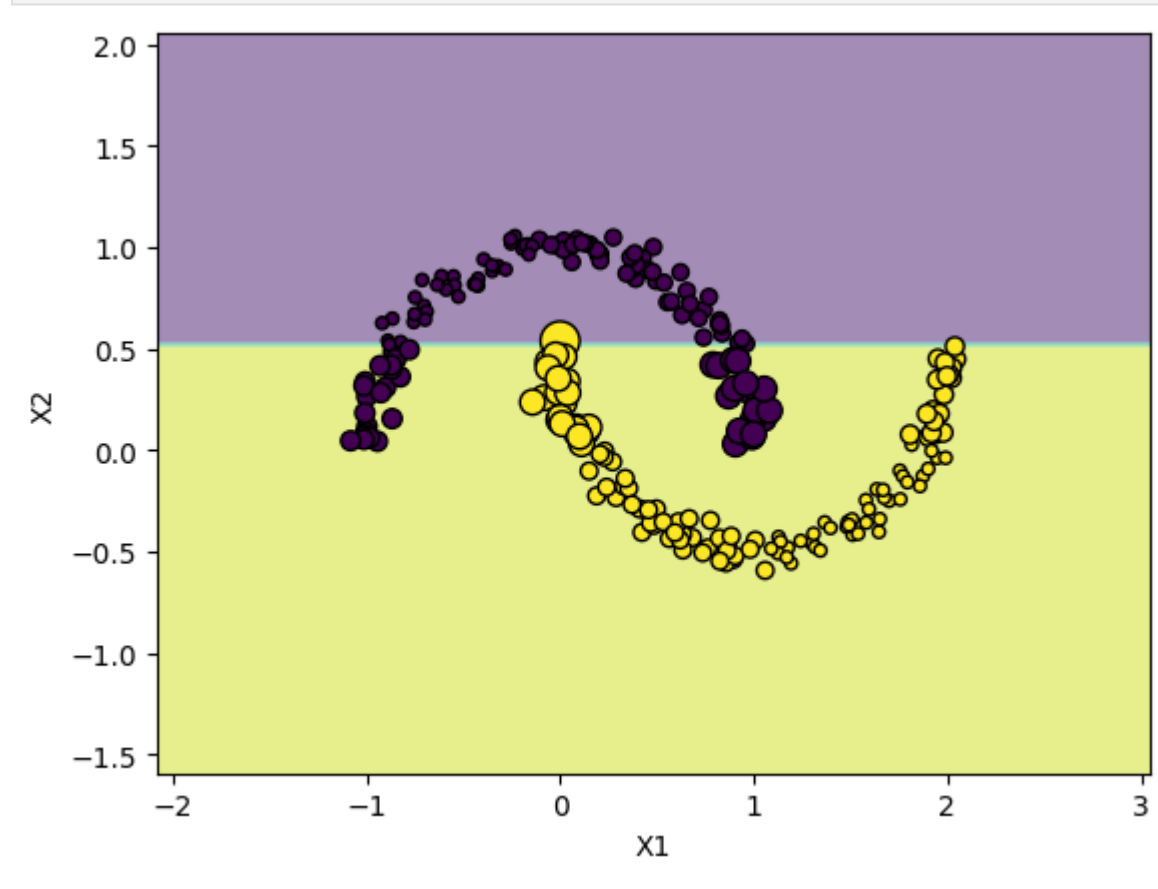
In [ ]:
```python
#visualize the nth classifier and weights
n = 2
from sklearn.inspection import DecisionBoundaryDisplay
disp = DecisionBoundaryDisplay.from_estimator(classifiers[n], X,
                                              response_method="predict",
                                              xlabel='X1', ylabel='X2',
                                              alpha=0.5,                                     )
disp.ax_.scatter(X[:, 0], X[:, 1],s = np.array(weights[n])*5000 ,c=y, edgecolor="black")

plt.show()
```



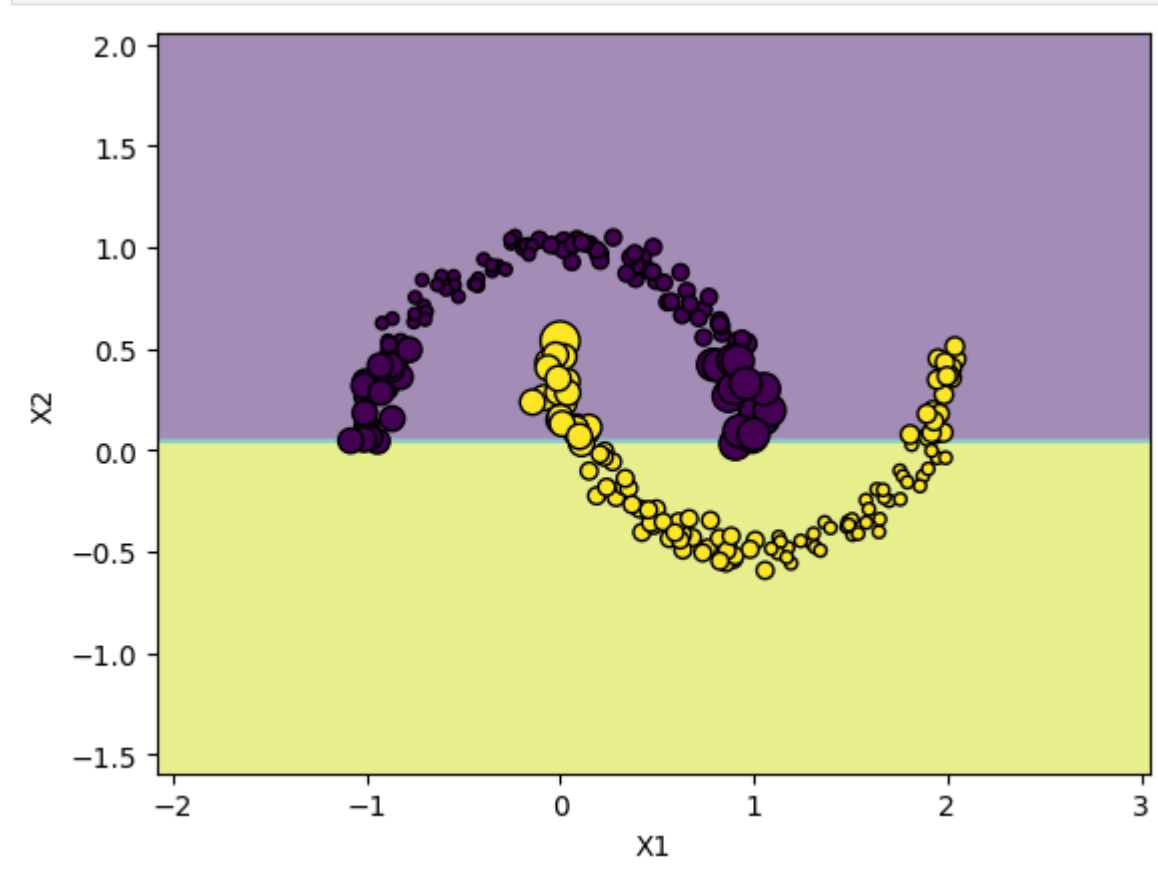
In [ ]:
```python
#visualize the nth classifier and weights
n = 3
from sklearn.inspection import DecisionBoundaryDisplay
disp = DecisionBoundaryDisplay.from_estimator(classifiers[n], X,
                                              response_method="predict",
                                              xlabel='X1', ylabel='X2',
                                              alpha=0.5,                                     )
disp.ax_.scatter(X[:, 0], X[:, 1],s = np.array(weights[n])*5000 ,c=y, edgecolor="black")

plt.show()
```



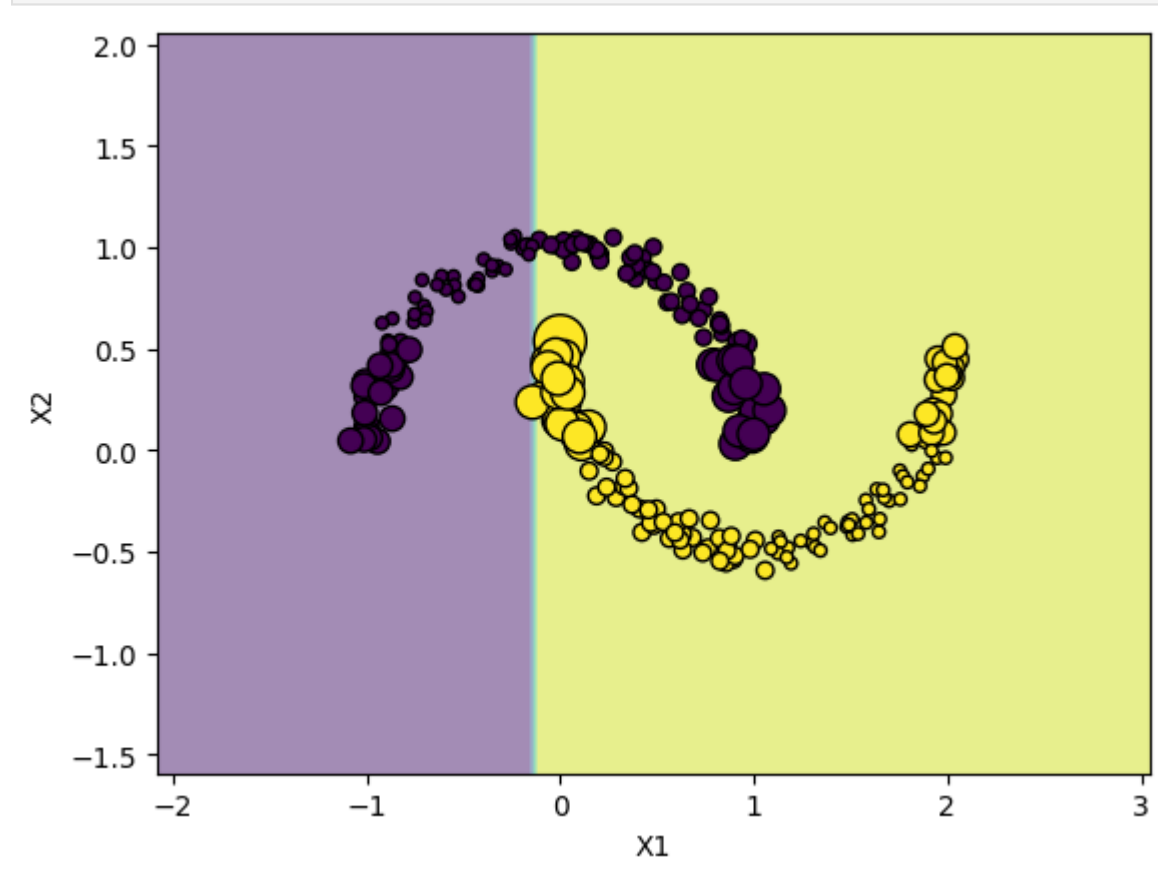
In [ ]:
```python
#visualize the nth classifier and weights
n = 4
from sklearn.inspection import DecisionBoundaryDisplay
disp = DecisionBoundaryDisplay.from_estimator(classifiers[n], X,
                                              response_method="predict",
                                              xlabel='X1', ylabel='X2',
                                              alpha=0.5,                                     )
disp.ax_.scatter(X[:, 0], X[:, 1],s = np.array(weights[n])*5000 ,c=y, edgecolor="black")

plt.show()
```



In [ ]:
```python
#visualize the nth classifier and weights
n = 5
from sklearn.inspection import DecisionBoundaryDisplay
disp = DecisionBoundaryDisplay.from_estimator(classifiers[n], X,
                                              response_method="predict",
                                              xlabel='X1', ylabel='X2',
                                              alpha=0.5,                                     )
disp.ax_.scatter(X[:, 0], X[:, 1],s = np.array(weights[n])*5000 ,c=y, edgecolor="black")

plt.show()
```

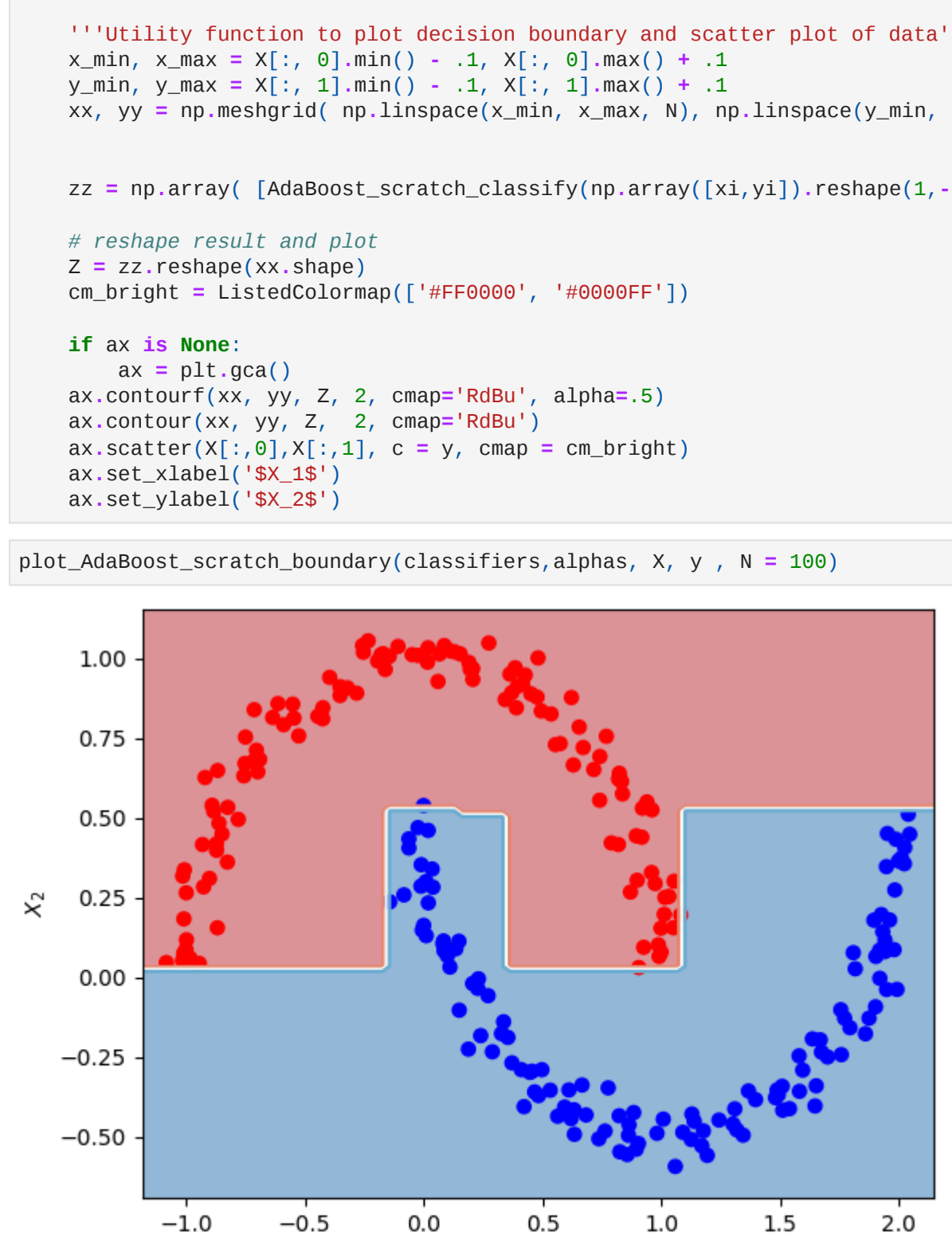


In [ ]:
```python
#visualize the nth classifier and weights
n = 6
from sklearn.inspection import DecisionBoundaryDisplay
disp = DecisionBoundaryDisplay.from_estimator(classifiers[n], X,
                                              response_method="predict",
                                              xlabel='X1', ylabel='X2',
                                              alpha=0.5,                                     )
disp.ax_.scatter(X[:, 0], X[:, 1],s = np.array(weights[n])*5000 ,c=y, edgecolor="black")

plt.show()
```



In [ ]:
```python
def plot_AdaBoost_scratch_boundary(estimators,estimator_weights, X, y, N = 10,ax = None ):

    def AdaBoost_scratch_classify(x_temp, est,est_weights ):
        '''Return classification prediction for a given point X and a previously fitted AdaBoost'''
        temp_pred = np.asarray( [ (e.predict(x_temp)).T* w for e, w in zip(est,est_weights )]  ) / sum(est_weights)
        return np.sign(temp_pred.sum(axis = 0))


    '''Utility function to plot decision boundary and scatter plot of data'''
    x_min, x_max = X[:, 0].min() - .1, X[:, 0].max() + .1
    y_min, y_max = X[:, 1].min() - .1, X[:, 1].max() + .1
    xx, yy = np.meshgrid( np.linspace(x_min, x_max, N), np.linspace(y_min, y_max, N))


    zz = np.array( [AdaBoost_scratch_classify(np.array([xi,yi]).reshape(1,-1), estimators,estimator_weights ) for  xi, yi in zip(np.ravel(xx)

    # reshape result and plot
    Z = zz.reshape(xx.shape)
    cm_bright = ListedColormap(['#FF0000', '#0000FF'])

    if ax is None:
        ax = plt.gca()
    ax.contourf(xx, yy, Z, 2, cmap='RdBu', alpha=.5)
    ax.contour(xx, yy, Z,  2, cmap='RdBu')
    ax.scatter(X[:,0],X[:,1], c = y, cmap = cm_bright)
    ax.set_xlabel('$X_1$')
    ax.set_ylabel('$X_2$')
```

In [ ]:
```python
plot_AdaBoost_scratch_boundary(classifiers,alphas, X, y , N = 100)
```



In [ ]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

In [ ]:
```python
!jupyter nbconvert --to html "/content/drive/MyDrive/Colab Notebooks/tutorial_VI_decision_tree.ipynb"
```