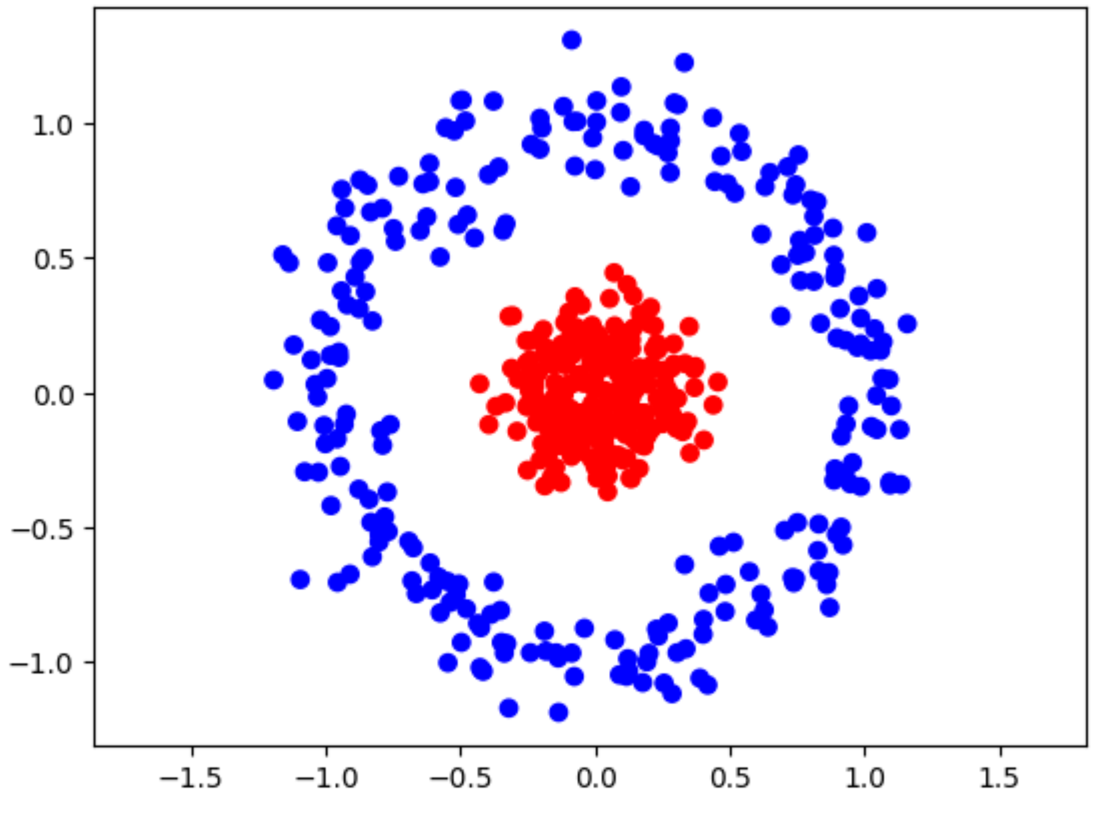
- Name: Gokulakrishnan B
- Roll No: DA24M007
- Department: Mtech DSAI
- Assignment No: 8

Importing libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from matplotlib.colors import ListedColormap
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.base import clone
```

Creating dataset (given in question)

In [2]:
```
from sklearn.datasets import make_moons, make_circles
from sklearn.model_selection import train_test_split

X, y = make_circles(n_samples=500, noise=0.1, random_state=42, factor=0.2)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
y_train = np.where(y_train == 0, -1, y_train)
y_test = np.where(y_test == 0, -1, y_test)
plt.scatter(X[:,0], X[:,1], c=y, cmap=colors.ListedColormap(["blue", "red"]))
plt.axis('equal')
plt.show()
```



In [3]:
```
pd.Series(y).value_counts()
```

Out[3]:
```
1    250
0    250
Name: count, dtype: int64
```

The dataset is balanced.

Implementing AdaBoost Algorithm from scratch

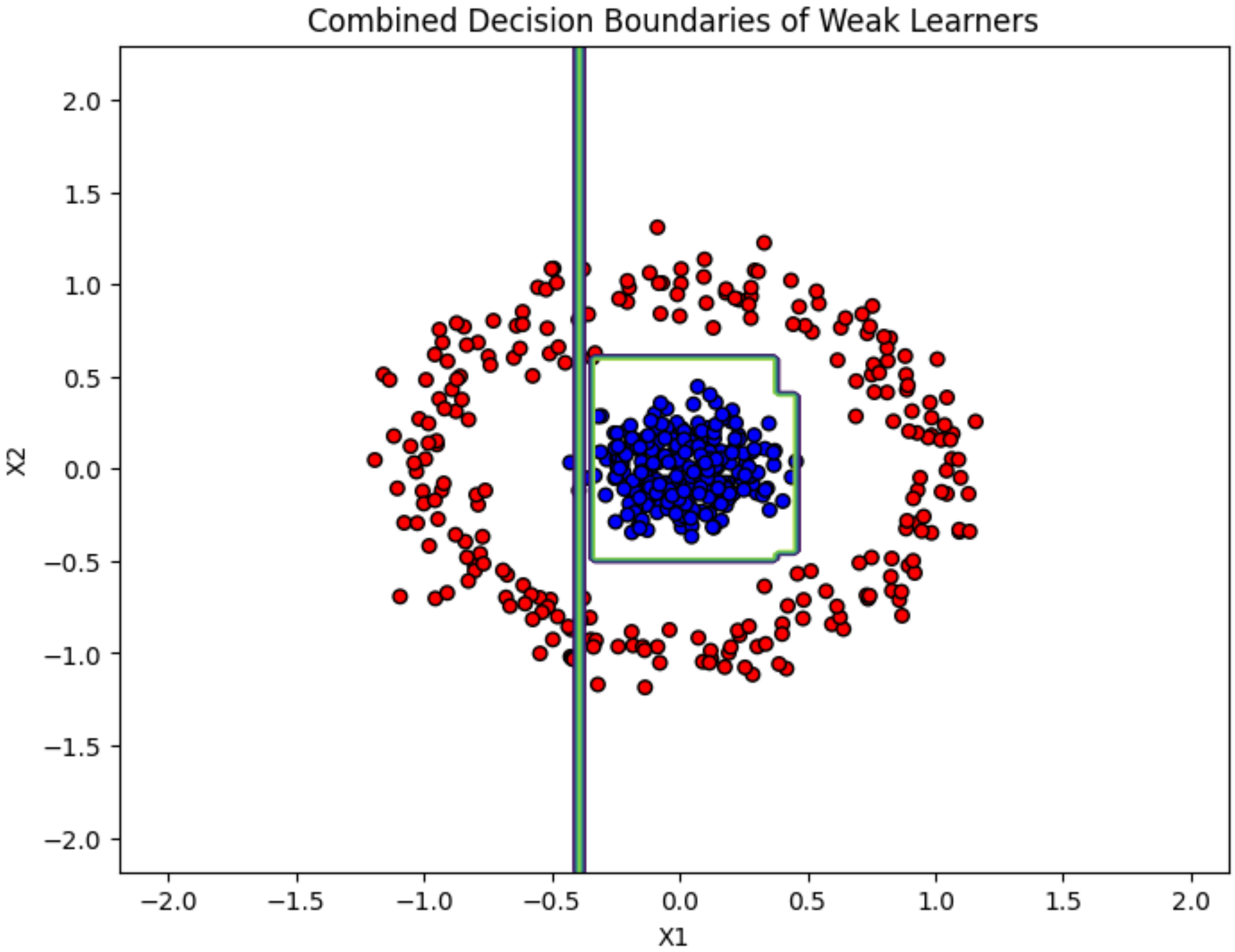
In [4]:
```
class Adaboost:
    def __init__(self, base_learner, n_learners, eta=0.5, **learner_params):
        self.base_learner = base_learner
        self.n_learners = n_learners
        self.eta = eta
        self.learners = []
        self.alphas = []
        self.errors = []
        self.learner_params = learner_params
```

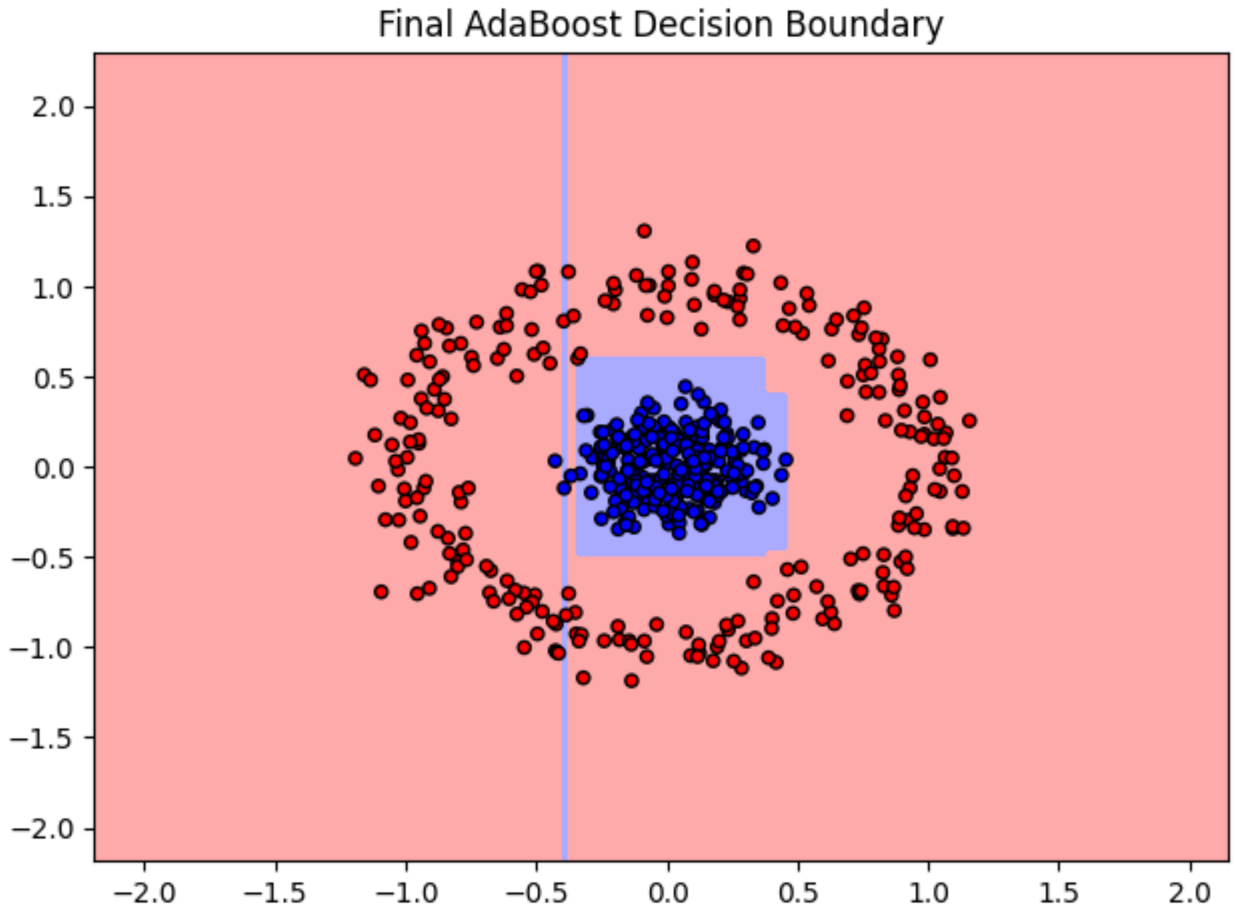
```python
    def fit(self, X, y):
        n_samples, n_features = X.shape
        w = np.ones(n_samples) / n_samples

        for _ in range(self.n_learners):
            learner = self.base_learner(**self.learner_params)
            learner.fit(X, y, sample_weight=w)
            y_pred = learner.predict(X)
            error = np.sum(w[y != y_pred]) / np.sum(w)

            alpha = self.eta * 0.5 * np.log((1 - error) / (error + 1e-10))

            w = w * np.exp(-alpha * y_pred * y)
            w = w / np.sum(w)

            self.alphas.append(alpha)
            self.learners.append(learner)
            self.errors.append(error)

    def predict(self, X):
        predictions = np.zeros(X.shape[0])

        for alpha, learner in zip(self.alphas, self.learners):
            predictions += alpha * learner.predict(X)

        predictions = np.sign(predictions)

        return predictions

    def plot_decision_boundaries(self, X, y):
        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.arange(x_min, x_max, 0.02),
                             np.arange(y_min, y_max, 0.02))


        cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF'])
        cmap_bold = ['#FF0000', '#0000FF']

        plt.figure(figsize=(8, 6))

        for i, learner in enumerate(self.learners):
            Z = learner.predict(np.c_[xx.ravel(), yy.ravel()])
            Z = Z.reshape(xx.shape)

            plt.contour(xx, yy, Z, alpha=0.3, linewidths=1)


        plt.scatter(X[:, 0], X[:, 1], c=y, cmap=ListedColormap(cmap_bold), edgecolor='k', s=30)


        plt.xlim(xx.min(), xx.max())
        plt.ylim(yy.min(), yy.max())
        plt.title('Combined Decision Boundaries of Weak Learners')
        plt.xlabel('X1')
        plt.ylabel('X2')

        plt.show()


    def plot_final_decision_boundary(self, X, y):
        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.arange(x_min, x_max, 0.02),
                              np.arange(y_min, y_max, 0.02))


        Z = self.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)

        cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF'])
        cmap_bold = ['#FF0000', '#0000FF']

        plt.contourf(xx, yy, Z, cmap=cmap_light)
        plt.scatter(X[:, 0], X[:, 1], c=y, cmap=ListedColormap(cmap_bold), edgecolor='k', s=20)
        plt.title('Final AdaBoost Decision Boundary')
        plt.xlim(xx.min(), xx.max())
        plt.ylim(yy.min(), yy.max())
        plt.tight_layout()
        plt.show()
```

Hyperparameter tuning on Adaboost algorithm

In [5]:
```
from sklearn.metrics import accuracy_score
# Model - Decision stump

eta_values = np.arange(0,1,0.1)
n_learners_values=np.arange(1,10,2)
baseClassifier = DecisionTreeClassifier

best_eta=None
best_n_learner = None
max_score = float('-inf')

for eta in eta_values:
    for n_learners in n_learners_values:
        model = Adaboost(baseClassifier,n_learners=n_learners, eta=eta,max_depth=1)
        model.fit(X_train,y_train)
        y_pred = model.predict(X_test)
        score = accuracy_score(y_test,y_pred)

        if score > max_score:
            max_score = score
            best_eta=eta
            best_n_learner = n_learners

print(f"Best eta value : {best_eta}")
print(f"Best n_learners value : {best_n_learner}")
print(f"Accuracy score on best parameters: {max_score}")

ada = Adaboost(baseClassifier,best_n_learner,best_eta)
ada.fit(X_train,y_train)
ada.plot_decision_boundaries(X,y)
ada.plot_final_decision_boundary(X,y)

print(classification_report(y_test,ada.predict(X_test)))
```

```
Best eta value : 0.8
Best n_learners value : 9
Accuracy score on best parameters: 0.904
```





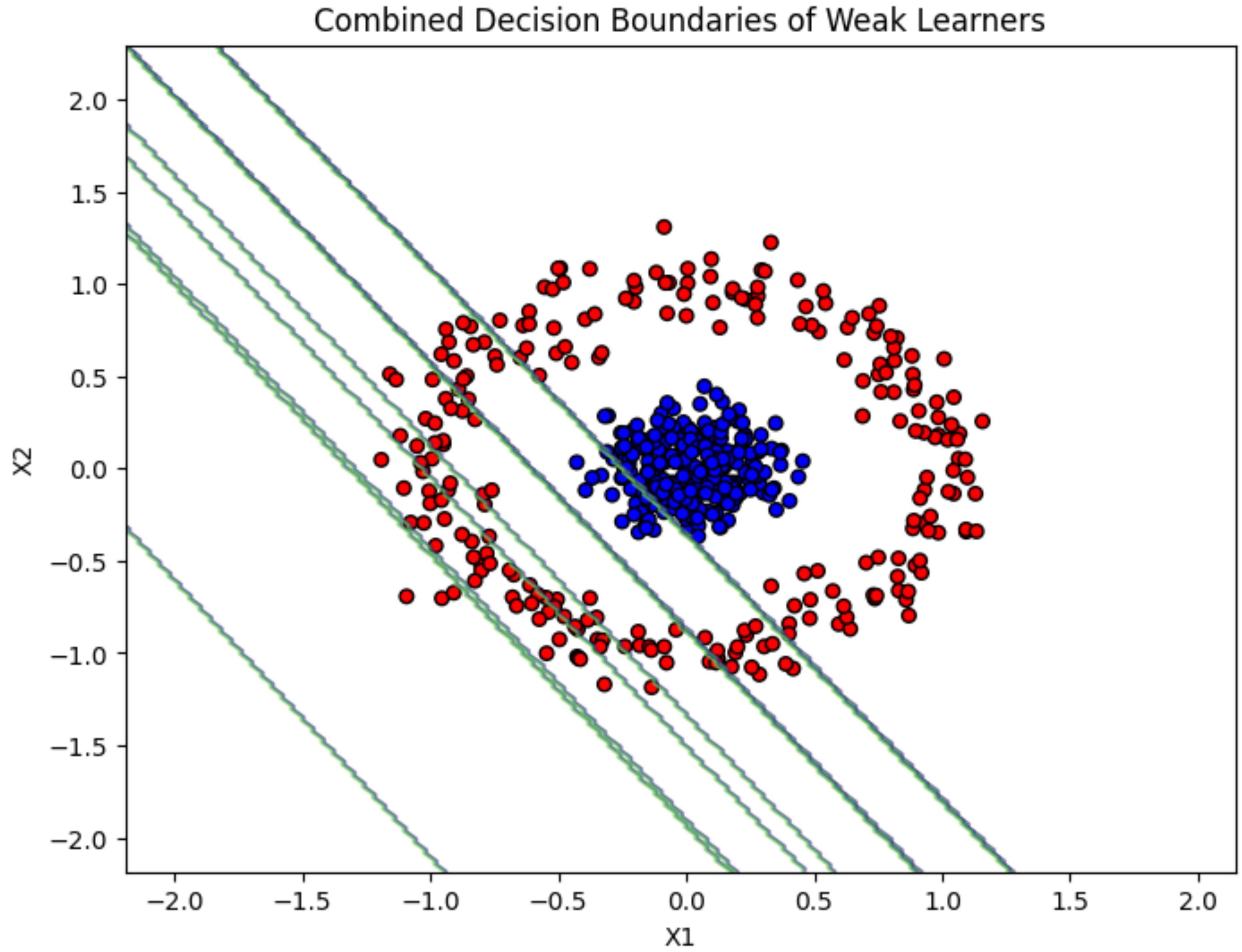

```
              precision    recall  f1-score   support

          -1       0.97      1.00      0.98        61
           1       1.00      0.97      0.98        64

    accuracy                           0.98       125
   macro avg       0.98      0.98      0.98       125
weighted avg       0.98      0.98      0.98       125
```

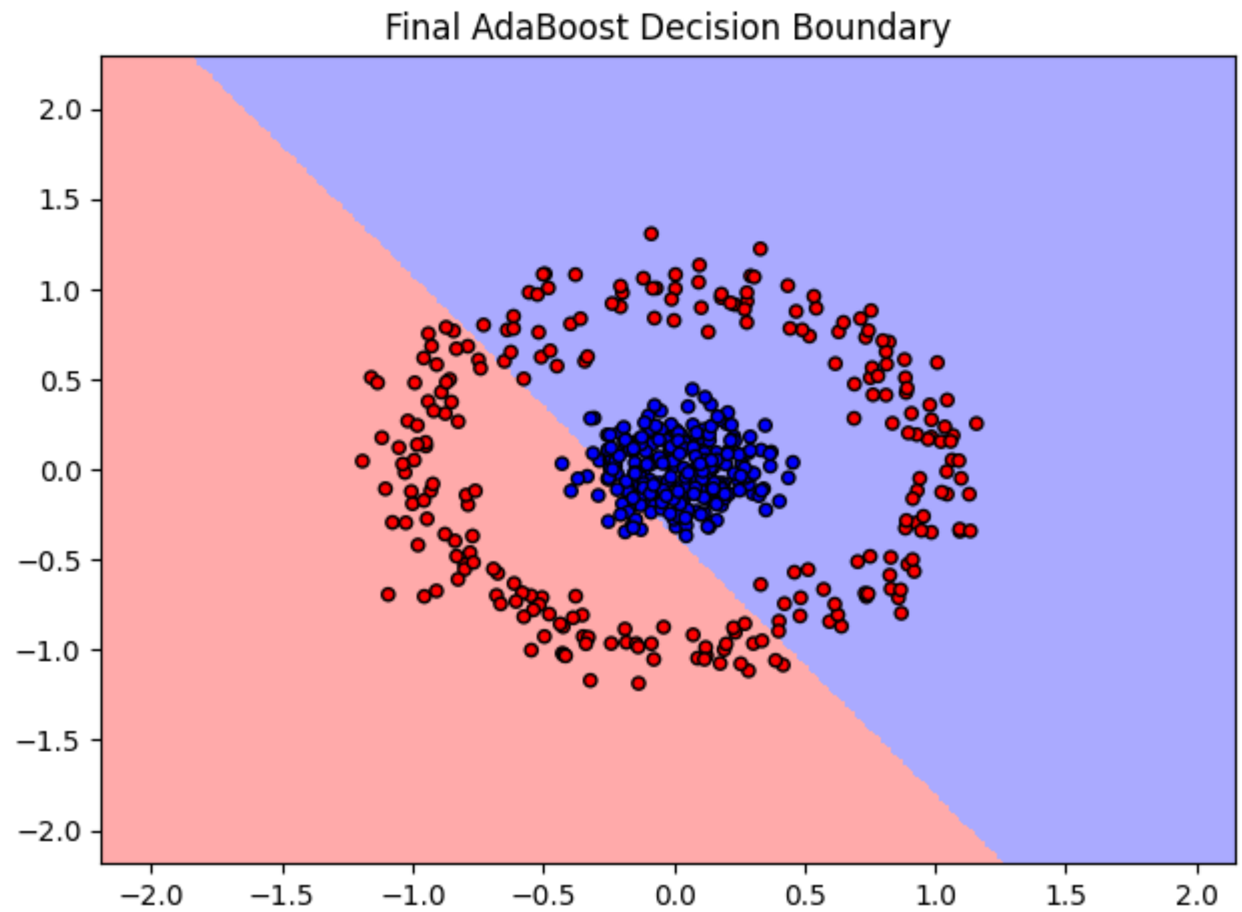
Here, we used a decision stump (a decision tree with just one node) as the weak learner. As shown in the images above, the boosted model is able to recognize the pattern in the data more effectively than the individual weak learners, although it doesn't capture the relationship perfectly. To enhance its performance, we can experiment with different models as weak learners.

In [6]:

```
from sklearn.metrics import accuracy_score
# Model - Logistic Regression

eta_values = np.arange(0,1,0.1)
n_learners_values=np.arange(1,10,2)
param_grid_log_penalty = ['l1','l2']
param_grid_log_c = [1,3,5]
baseClassifier = LogisticRegression

best_eta=None
best_n_learner = None
max_score = float('-inf')
best_penalty=None
best_c=None

for eta in eta_values:
    for n_learners in n_learners_values:
        for penalty in param_grid_log_penalty:
            for c in param_grid_log_c:
                model = Adaboost(baseClassifier,n_learners=n_learners, eta=eta,solver='liblinear', random_state=42,
                model.fit(X_train,y_train)
                y_pred = model.predict(X_test)
                score = accuracy_score(y_test,y_pred)

                if score > max_score:
                    max_score = score
                    best_eta=eta
                    best_n_learner = n_learners
                    best_penalty=penalty
                    best_c=c

print(f"Best eta value : {best_eta}")
print(f"Best n_learners value : {best_n_learner}")
print(f"Accuracy score on best parameters: {max_score}")

ada = Adaboost(baseClassifier,n_learners=n_learners, eta=eta,solver='liblinear', random_state=42,max_iter=10000,pen
ada.fit(X_train,y_train)
ada.plot_decision_boundaries(X,y)
ada.plot_final_decision_boundary(X,y)

print(classification_report(y_test,ada.predict(X_test)))
```

```
Best eta value : 0.8
Best n_learners value : 3
Accuracy score on best parameters: 0.736
```





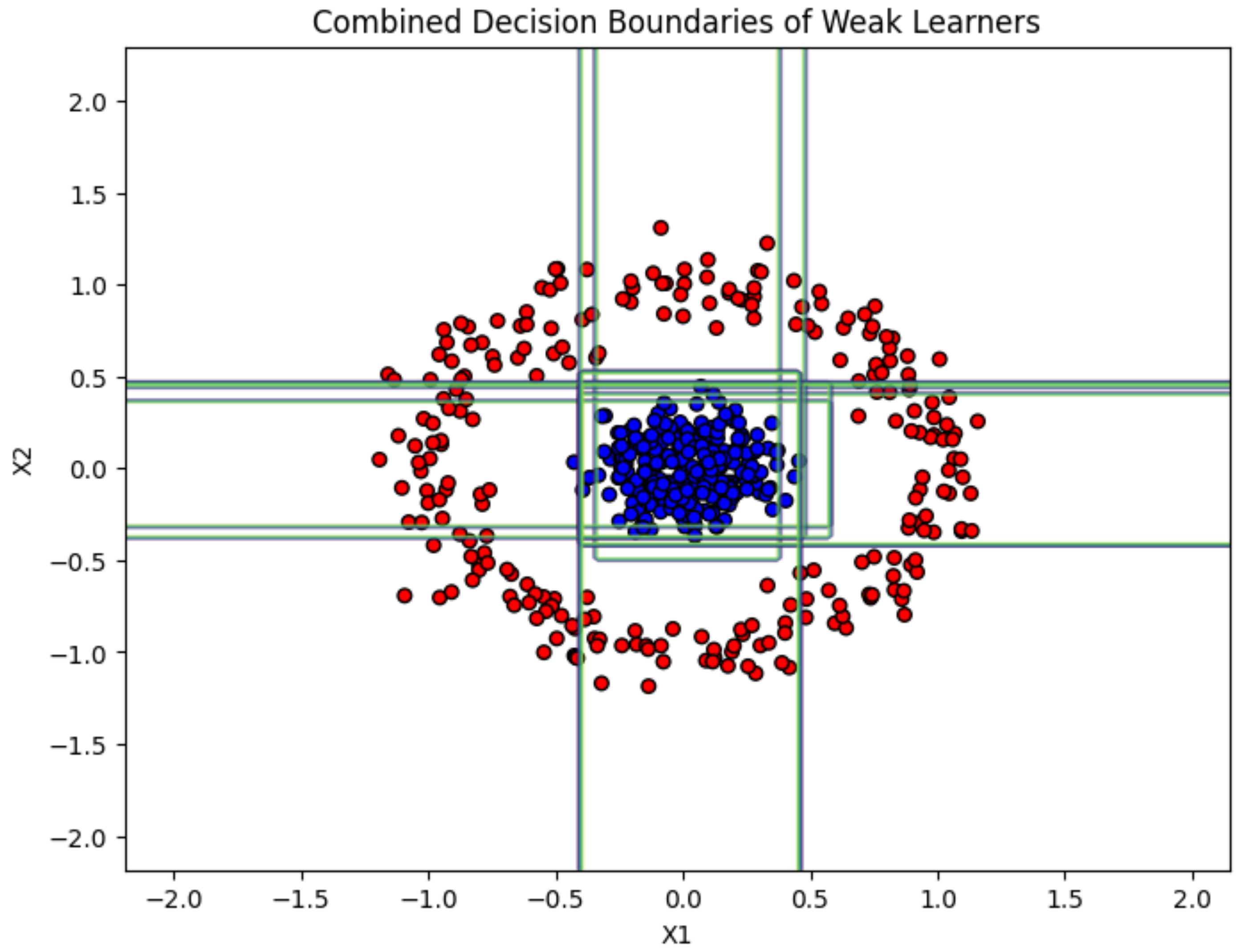

```
              precision    recall  f1-score   support

          -1       0.85      0.46      0.60        61
           1       0.64      0.92      0.76        64

    accuracy                           0.70       125
   macro avg       0.74      0.69      0.68       125
weighted avg       0.74      0.70      0.68       125
```

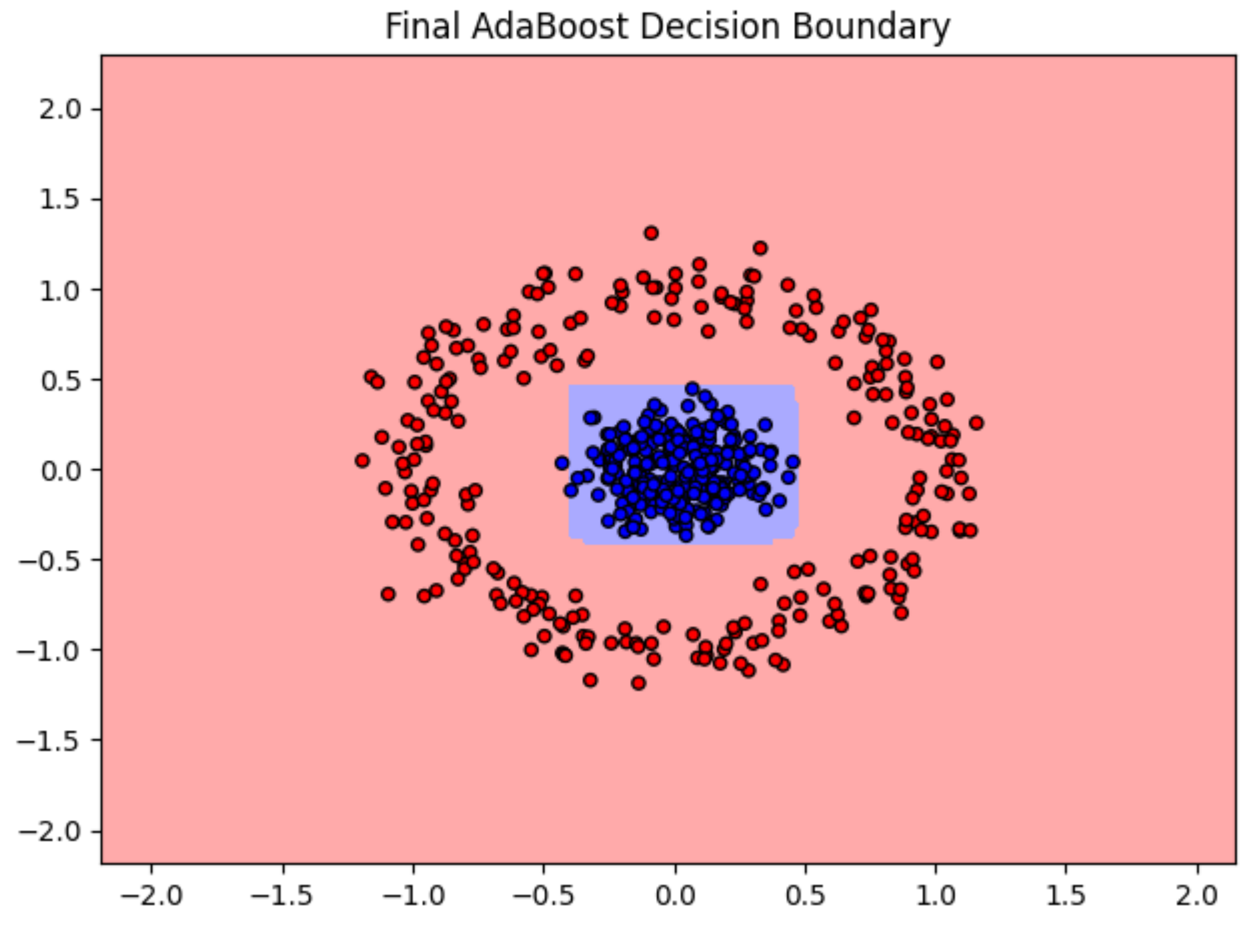
In [7]:

```
from sklearn.metrics import accuracy_score
# Model - Decision Tree

eta_values = np.arange(0,1,0.1)
n_learners_values=np.arange(1,10,2)
param_grid_dt = [3,5,7,9] # min sample split
baseClassifier = DecisionTreeClassifier

best_eta=None
best_n_learner = None
max_score = float('-inf')
best_dt = None

for eta in eta_values:
    for n_learners in n_learners_values:
        for dt in  param_grid_dt:
            model = Adaboost(baseClassifier,n_learners=n_learners, eta=eta,min_samples_split=dt,max_depth=3)
```

```
            model.fit(X_train,y_train)
            y_pred = model.predict(X_test)
            score = accuracy_score(y_test,y_pred)

            if score > max_score:
                max_score = score
                best_eta=eta
                best_n_learner = n_learners
                best_dt = dt

print(f"Best eta value : {best_eta}")
print(f"Best n_learners value : {best_n_learner}")
print(f"Accuracy score on best parameters: {max_score}")

ada = Adaboost(baseClassifier,n_learners=n_learners, eta=eta,min_samples_split=best_dt,max_depth=3)
ada.fit(X_train,y_train)
ada.plot_decision_boundaries(X,y)
ada.plot_final_decision_boundary(X,y)

print(classification_report(y_test,ada.predict(X_test)))
```

```
Best eta value : 0.1
Best n_learners value : 3
Accuracy score on best parameters: 1.0
```

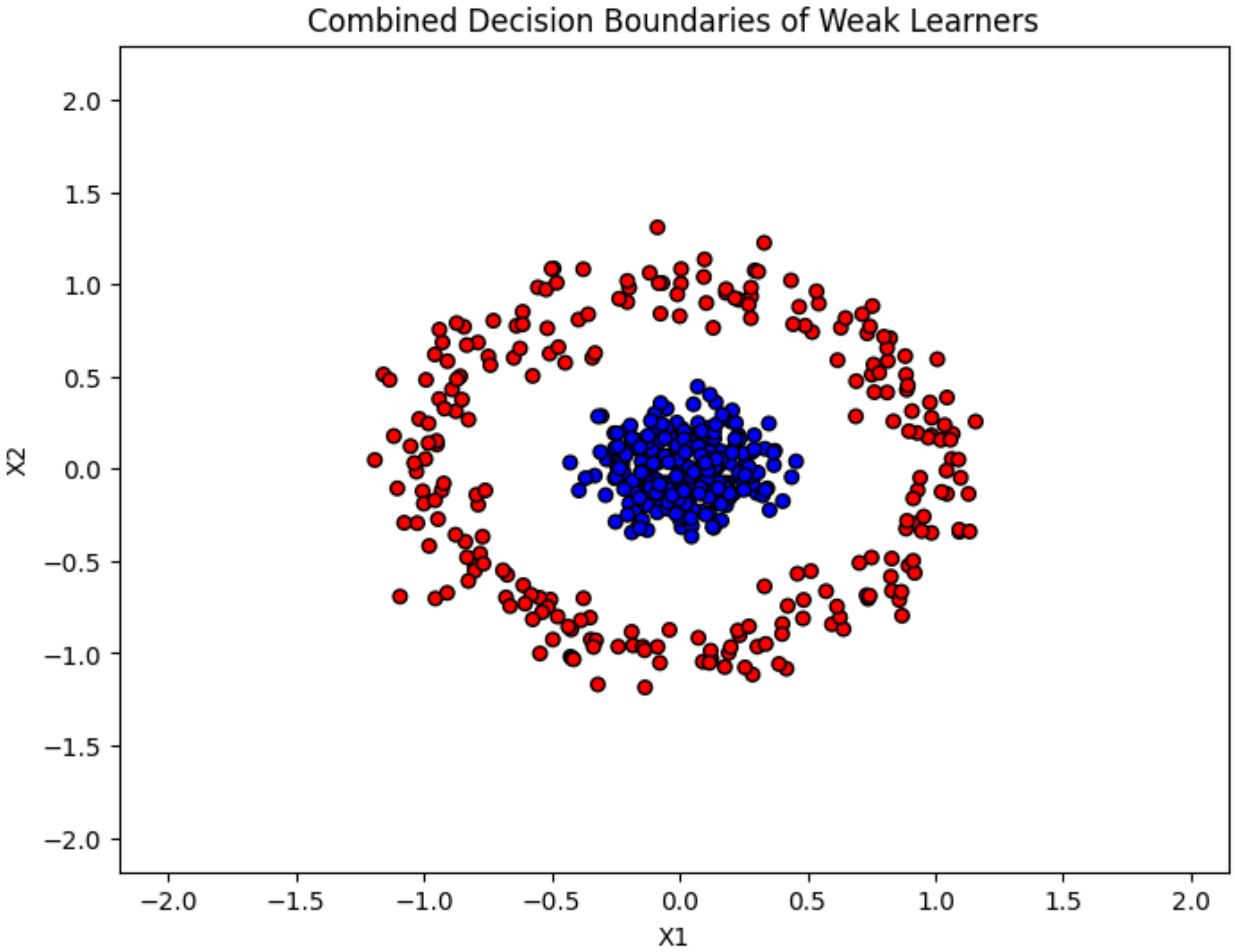


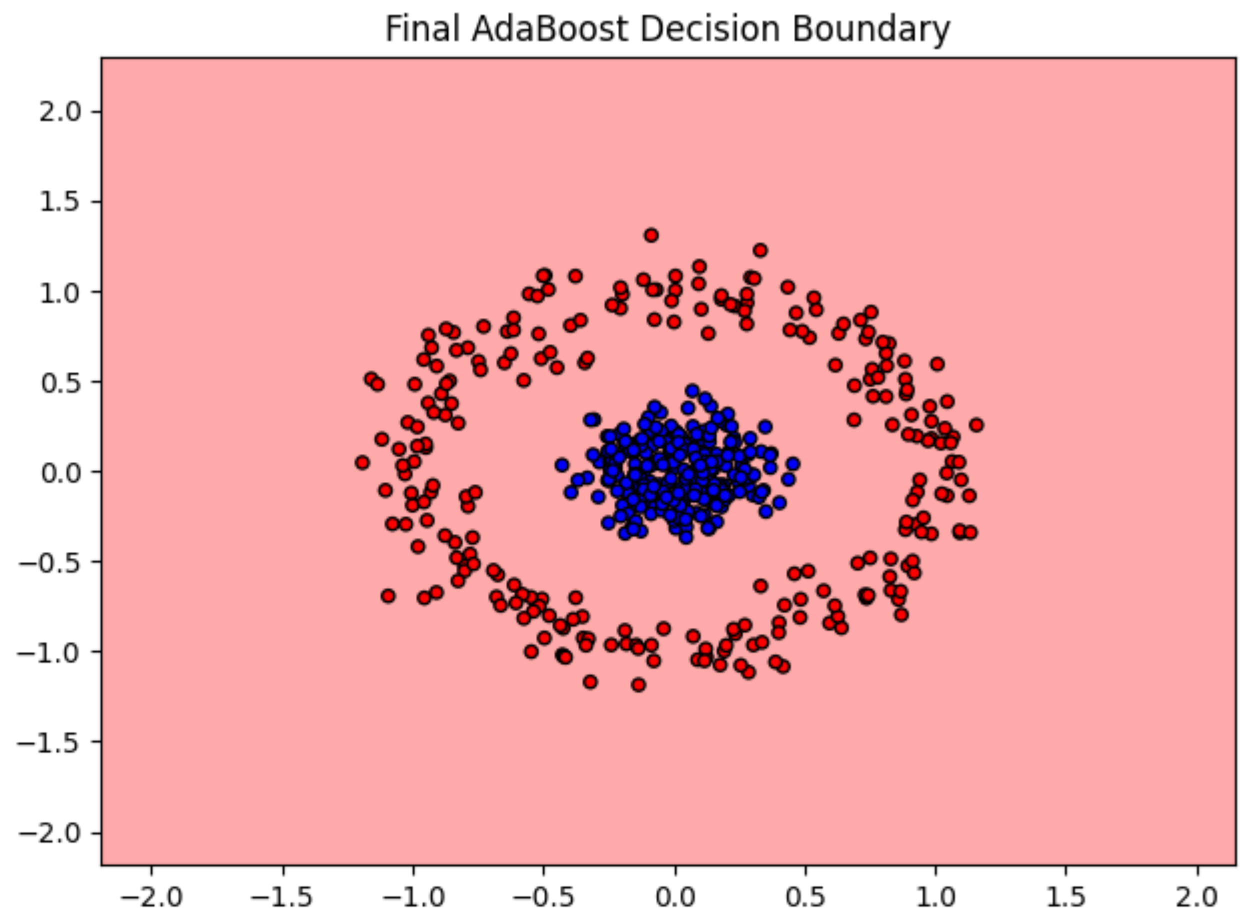


```
              precision    recall  f1-score   support

          -1       0.98      1.00      0.99        61
           1       1.00      0.98      0.99        64

    accuracy                           0.99       125
   macro avg       0.99      0.99      0.99       125
weighted avg       0.99      0.99      0.99       125
```

In [8]:
```python
from sklearn.metrics import accuracy_score
# Model - Linear SVM

eta_values = np.arange(0,1,0.1)
n_learners_values=np.arange(1,10,2)
param_grid_svc_c =  [3,5,7,9]
baseClassifier = SVC

best_eta=None
best_n_learner = None
max_score = float('-inf')
best_c = None

for eta in eta_values:
    for n_learners in n_learners_values:
        for c in  param_grid_svc_c:
            model = Adaboost(baseClassifier,n_learners=n_learners, eta=eta,C=c,kernel='linear')
            model.fit(X_train,y_train)
            y_pred = model.predict(X_test)
            score = accuracy_score(y_test,y_pred)

            if score > max_score:
                max_score = score
                best_eta=eta
                best_n_learner = n_learners
                best_c = c

print(f"Best eta value : {best_eta}")
print(f"Best n_learners value : {best_n_learner}")
print(f"Accuracy score on best parameters: {max_score}")

ada = Adaboost(baseClassifier,n_learners=n_learners, eta=eta,C=best_c,kernel='linear')
ada.fit(X_train,y_train)
ada.plot_decision_boundaries(X,y)
ada.plot_final_decision_boundary(X,y)

print(classification_report(y_test,ada.predict(X_test)))
```

```
Best eta value : 0.1
Best n_learners value : 1
Accuracy score on best parameters: 0.488
```






```
              precision    recall  f1-score   support

          -1       0.49      1.00      0.66        61
           1       0.00      0.00      0.00        64

    accuracy                           0.49       125
   macro avg       0.24      0.50      0.33       125
weighted avg       0.24      0.49      0.32       125
```

```
/home/gokul/miniconda3/envs/dsai/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/gokul/miniconda3/envs/dsai/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/gokul/miniconda3/envs/dsai/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Incorporating Various weak models into a single Adaboost model

In [9]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.base import clone
```

In [10]:
```python
from sklearn.base import BaseEstimator, ClassifierMixin

class NewAdaboost(BaseEstimator, ClassifierMixin):
    def __init__(self, base_learners, n_learners=50, eta=0.5):
        self.base_learners = base_learners
        self.n_learners = n_learners
        self.eta = eta
        self.learners = []
        self.alphas = []
        self.errors=[]

    def fit(self, X, y):
        n_samples = X.shape[0]
        w = np.ones(n_samples) / n_samples

        for i in range(self.n_learners):
            learner = clone(self.base_learners[i % len(self.base_learners)])
            learner.fit(X, y, sample_weight=w)
            y_pred = learner.predict(X)

            error = np.sum(w * (y_pred != y)) / np.sum(w)

            alpha = self.eta * np.log((1 - error) / error)

            w = w * np.exp(-alpha * y * y_pred)
            w = w / np.sum(w)
```

```python
            self.learners.append(learner)
            self.alphas.append(alpha)
            self.errors.append(error)

    def predict(self, X):
        final_predictions = np.zeros(X.shape[0])
        for alpha, learner in zip(self.alphas, self.learners):
            final_predictions += alpha * learner.predict(X)

        return np.sign(final_predictions)

    def plot_decision_boundaries(self, X, y):
        # Define grid for plotting
        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.arange(x_min, x_max, 0.02),
                             np.arange(y_min, y_max, 0.02))

        # Set up the plot
        cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF'])
        cmap_bold = ['#FF0000', '#0000FF']

        plt.figure(figsize=(8, 6))

        # Loop over each weak learner and plot its decision boundary
        for i, learner in enumerate(self.learners):
            Z = learner.predict(np.c_[xx.ravel(), yy.ravel()])
            Z = Z.reshape(xx.shape)

            # Plot decision boundary (contour lines)
            plt.contour(xx, yy, Z, alpha=0.3, linewidths=1)  # Alpha and linewidth adjust the visibility of the con

        # Plot the data points
        plt.scatter(X[:, 0], X[:, 1], c=y, cmap=ListedColormap(cmap_bold), edgecolor='k', s=30)

        # Labels and limits
        plt.xlim(xx.min(), xx.max())
        plt.ylim(yy.min(), yy.max())
        plt.title('Combined Decision Boundaries of Weak Learners')
        plt.xlabel('X1')
        plt.ylabel('X2')

        plt.show()



    def plot_final_decision_boundary(self, X, y):
        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.arange(x_min, x_max, 0.02),
                              np.arange(y_min, y_max, 0.02))


        Z = self.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)

        cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF'])
        cmap_bold = ['#FF0000', '#0000FF']

        plt.contourf(xx, yy, Z, cmap=cmap_light)
        plt.scatter(X[:, 0], X[:, 1], c=y, cmap=ListedColormap(cmap_bold), edgecolor='k', s=20)
        plt.title('Final AdaBoost Decision Boundary')
        plt.xlim(xx.min(), xx.max())
        plt.ylim(yy.min(), yy.max())
        plt.tight_layout()
        plt.show()
```

In [11]:
```python
weak_learners = [
    DecisionTreeClassifier(max_depth=1, random_state=42),
    DecisionTreeClassifier(max_depth=3, random_state=42),
    LogisticRegression(solver='liblinear', random_state=42,max_iter=10000),
    SVC(kernel='linear', random_state=42)
]
```

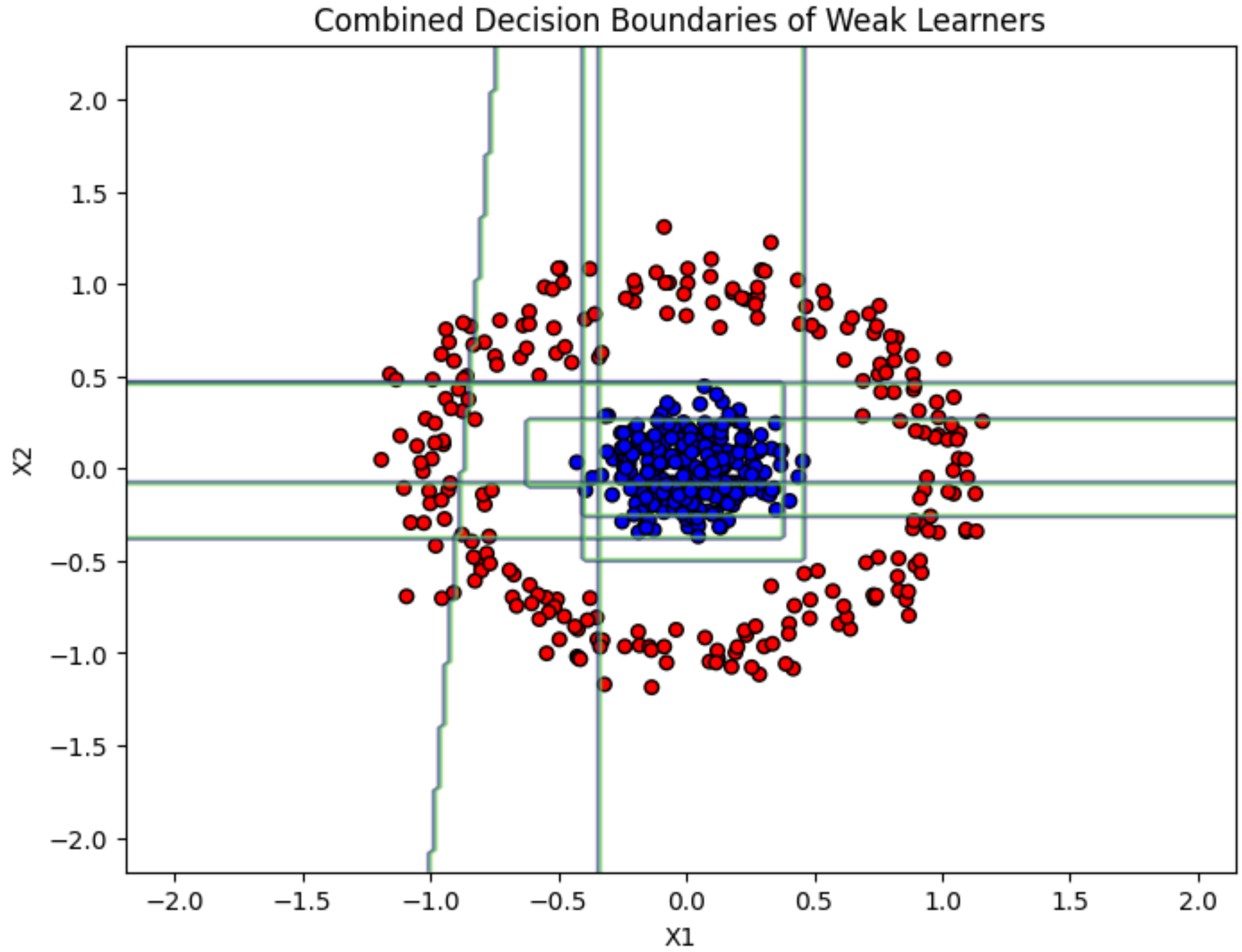
Hyperparameter tuning on various parameters of weak learners.

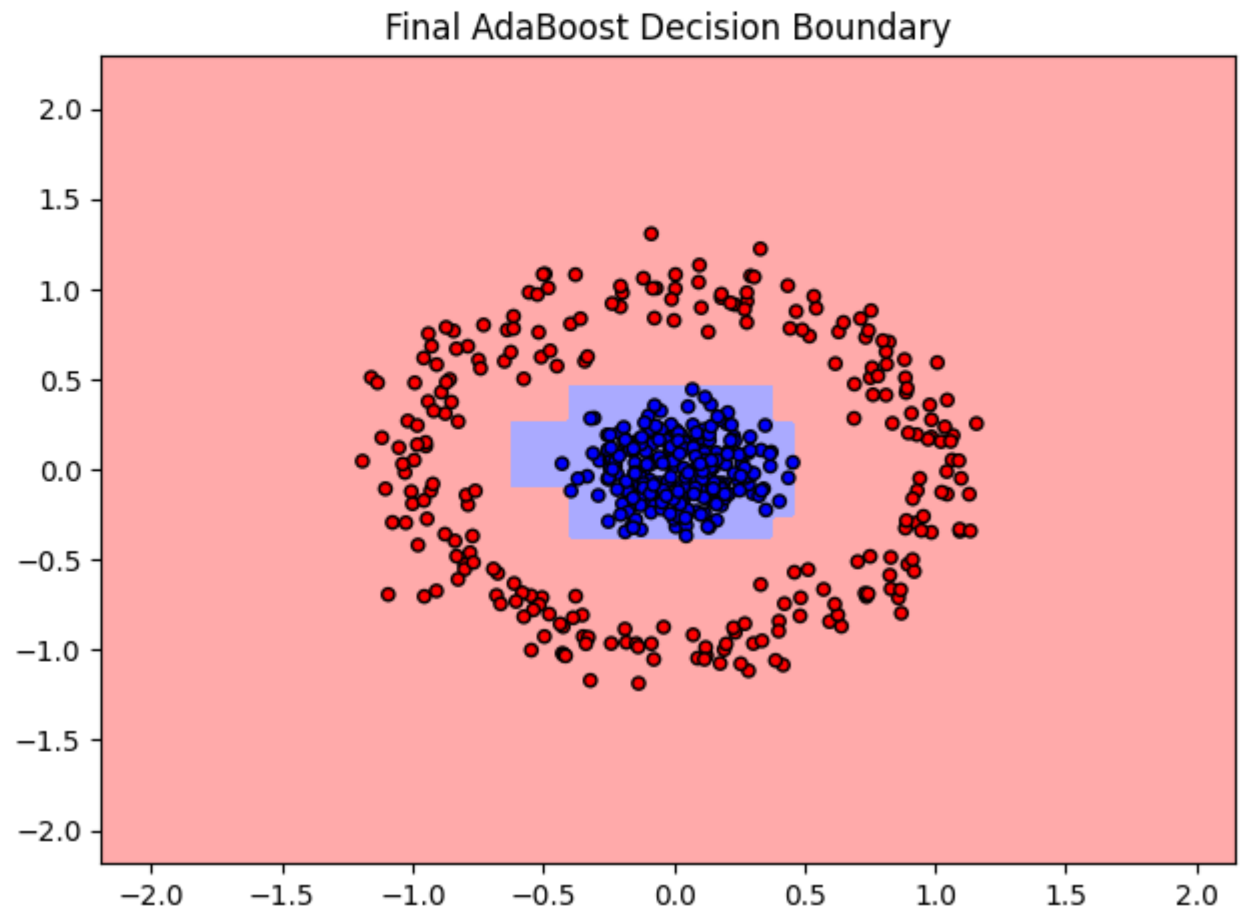
In [12]:
```python
eta_values = np.arange(0,1,0.3)
n_learners_values=np.arange(1,20,5)

param_grid_dt = [3,5] # min samples split
param_grid_log_penalty = ['l1','l2']  # penalty
param_grid_log_c = [1,3,5] # c value in Logistic Regression
param_grid_svc_c =  [1,3,5] # c value in SVC
```

In [13]:
```
from sklearn.metrics import accuracy_score

best_eta=None
best_n_learner = None
best_dt1 = None
best_dt2= None
best_log_penalty = None
best_log_c = None
best_svc_c = None
max_score = float('-inf')

for eta in eta_values:
    for n_learners in n_learners_values:
        for dt1 in param_grid_dt:
            for dt2 in param_grid_dt:
                for log_penalty in param_grid_log_penalty:
                    for log_c in param_grid_log_c:
                        for svc_c in param_grid_svc_c:

                            weak_learners = [
                                DecisionTreeClassifier(max_depth=1, random_state=42,min_samples_split=dt1),
                                DecisionTreeClassifier(max_depth=3, random_state=42,min_samples_split=dt2),
                                LogisticRegression(solver='liblinear', random_state=42,max_iter=10000, penalty=log_
                                SVC(kernel='linear', random_state=42,C=svc_c)
                            ]

                            model = NewAdaboost(base_learners=weak_learners,n_learners=n_learners, eta=eta)
                            model.fit(X_train,y_train)
                            y_pred = model.predict(X_test)
                            score = accuracy_score(y_test,y_pred)

                            if score > max_score:
                                max_score = score
                                best_eta=eta
                                best_n_learner = n_learners
                                best_dt1=dt1
                                best_dt2=dt2
                                best_log_penalty=log_penalty
                                best_log_c = log_c
                                best_svc_c=svc_c

print(f"Best eta value : {best_eta}")
print(f"Best n_learners value : {best_n_learner}")
print(f"Best dt1 value : {best_dt1}")
print(f"Best dt2 value : {best_dt2}")
print(f"Best penalty for log reg : {best_log_penalty}")
print(f"Best c value for log reg : {best_log_c}")
print(f"Best c value for SVC : {best_svc_c}")
print(f"Accuracy score on best parameters: {max_score}")
```

```
Best eta value : 0.6
Best n_learners value : 11
Best dt1 value : 3
Best dt2 value : 3
Best penalty for log reg : l1
Best c value for log reg : 5
Best c value for SVC : 3
Accuracy score on best parameters: 1.0
```

In [14]:
```
weak_learners = [
                                DecisionTreeClassifier(max_depth=1, random_state=42,min_samples_split=best_dt1),
                                DecisionTreeClassifier(max_depth=3, random_state=42,min_samples_split=best_dt2),
                                LogisticRegression(solver='liblinear', random_state=42,max_iter=10000, penalty=best
                                SVC(kernel='linear', random_state=42,C=best_svc_c)
                            ]

ada = NewAdaboost(weak_learners,best_n_learner,best_eta)
ada.fit(X_train,y_train)
ada.plot_decision_boundaries(X,y)
ada.plot_final_decision_boundary(X,y)
```





In [15]:
```
print(classification_report(y_test, ada.predict(X_test)))
```

```
              precision    recall  f1-score   support

          -1       1.00      1.00      1.00        61
           1       1.00      1.00      1.00        64

    accuracy                           1.00       125
   macro avg       1.00      1.00      1.00       125
weighted avg       1.00      1.00      1.00       125
```

Here, we observe a similar outcome to using only decision stumps as weak learners, but this approach has outperformed using a single type of weak learner. By utilizing multiple weak learners, we achieved 100% accuracy.

In [16]:
```
print('Alpha Values')
print(*[float(i) for i in ada.alphas])
print('')
print('Error Values')
print(*[float(i) for i in ada.errors])
```

```
Alpha Values
0.48944970159136664 1.8448097080522923 0.9414683502754397 0.9078059743192963 0.5095112286745234 1.8806960805637407 0
.8156227255665596 0.3521439023874767 0.8155256001830995 1.9069936651164854 0.3117266741365387

Error Values
0.30666666666666664 0.044163726705591294 0.17234274654376913 0.1804931459753935 0.2996037703412075 0.041706645437150
51 0.20434251625702005 0.3573449603046424 0.20436883633488606 0.03998967379547085 0.372958761069171
```

Here, we can observe that, the alpha values are lower for weak learners that have high errors, and vice versa.

In [ ]: