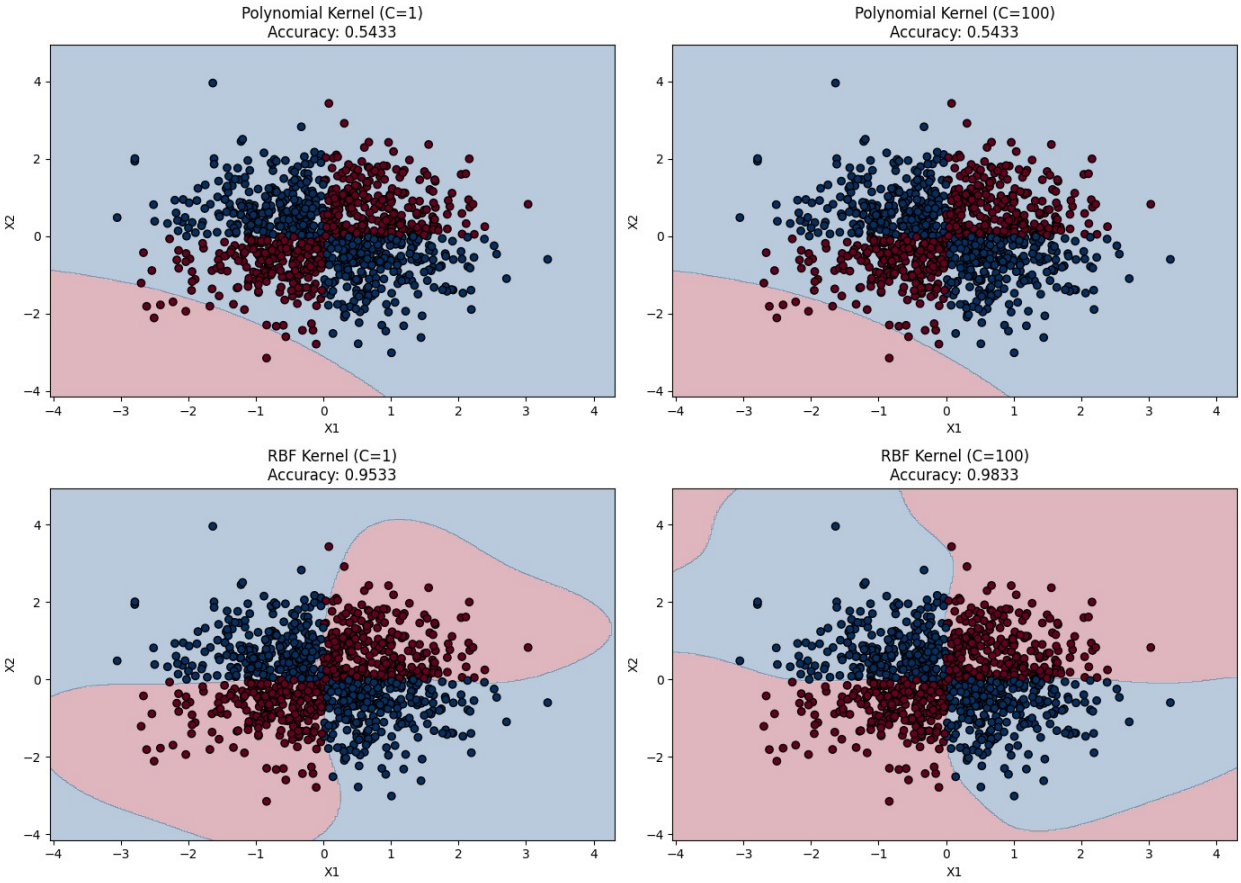
```python
# ==== XOR Problem with Nonlinear Support Vector Machine ====

import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

# --- Generate XOR dataset with 1000 samples ---
np.random.seed(1)
X_xor = np.random.randn(1000, 2)
y_xor = np.logical_xor(X_xor[:, 0] > 0, X_xor[:, 1] > 0)
y_xor = np.where(y_xor, 1, -1)

# --- Splitting data into 70% training and 30% test data ---
X_train, X_test, y_train, y_test = train_test_split(
    X_xor, y_xor, test_size=0.3, random_state=42)

# --- Standardizing the features ---
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

# --- Build SVM models with polynomial kernel (two regularization
strengths) ---
svm_poly_weak = SVC(kernel="poly", degree=3, C=1, random_state=42)
svm_poly_weak.fit(X_train_std, y_train)

svm_poly_strong = SVC(kernel="poly", degree=3, C=100, random_state=42)
svm_poly_strong.fit(X_train_std, y_train)

# --- Build SVM models with RBF kernel (two regularization strengths)
---
svm_rbf_weak = SVC(kernel="rbf", gamma='scale', C=1, random_state=42)
svm_rbf_weak.fit(X_train_std, y_train)

svm_rbf_strong = SVC(kernel="rbf", gamma='scale', C=100,
random_state=42)
svm_rbf_strong.fit(X_train_std, y_train)

# --- Test the SVM models ---
y_pred_poly_weak = svm_poly_weak.predict(X_test_std)
accuracy_poly_weak = accuracy_score(y_test, y_pred_poly_weak)
print(f"Polynomial Kernel (C=1) Accuracy: {accuracy_poly_weak:.4f}")

y_pred_poly_strong = svm_poly_strong.predict(X_test_std)
accuracy_poly_strong = accuracy_score(y_test, y_pred_poly_strong)
print(f"Polynomial Kernel (C=100) Accuracy:
```

```
{accuracy_poly_strong:.4f}")

y_pred_rbf_weak = svm_rbf_weak.predict(X_test_std)
accuracy_rbf_weak = accuracy_score(y_test, y_pred_rbf_weak)
print(f"RBF Kernel (C=1) Accuracy: {accuracy_rbf_weak:.4f}")

y_pred_rbf_strong = svm_rbf_strong.predict(X_test_std)
accuracy_rbf_strong = accuracy_score(y_test, y_pred_rbf_strong)
print(f"RBF Kernel (C=100) Accuracy: {accuracy_rbf_strong:.4f}")

# --- Function to plot decision boundaries ---
def plot_decision_boundary(X, y, model, title, ax):
    # Create a mesh grid
    h = 0.02  # Step size in the mesh
    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, h), np.arange(y_min,
y_max, h))

    # Apply standardization to the mesh grid
    mesh_points = np.c_[xx.ravel(), yy.ravel()]
    mesh_points_std = sc.transform(mesh_points)

    # Predict on the mesh grid
    Z = model.predict(mesh_points_std)
    Z = Z.reshape(xx.shape)

    # Plot the decision boundary
    ax.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.RdBu)

    # Plot the data points
    ax.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdBu,
edgecolors='k')

    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_title(title)
    ax.set_xlabel('X1')
    ax.set_ylabel('X2')

# --- Plot the decision boundaries ---
fig, axs = plt.subplots(2, 2, figsize=(14, 10))
axs = axs.flatten()

plot_decision_boundary(
    X_xor, y_xor,
    svm_poly_weak,
    f"Polynomial Kernel (C=1)\nAccuracy: {accuracy_poly_weak:.4f}",
    axs[0]
)
```

```
plot_decision_boundary(
    X_xor, y_xor,
    svm_poly_strong,
    f"Polynomial Kernel (C=100)\nAccuracy:
{accuracy_poly_strong:.4f}",
    axs[1]
)

plot_decision_boundary(
    X_xor, y_xor,
    svm_rbf_weak,
    f"RBF Kernel (C=1)\nAccuracy: {accuracy_rbf_weak:.4f}",
    axs[2]
)

plot_decision_boundary(
    X_xor, y_xor,
    svm_rbf_strong,
    f"RBF Kernel (C=100)\nAccuracy: {accuracy_rbf_strong:.4f}",
    axs[3]
)

plt.tight_layout()
plt.show()
```

```
Polynomial Kernel (C=1) Accuracy: 0.5433
Polynomial Kernel (C=100) Accuracy: 0.5433
RBF Kernel (C=1) Accuracy: 0.9533
RBF Kernel (C=100) Accuracy: 0.9833
```


Polynomial Kernel (C=1)
Accuracy: 0.5433
Polynomial Kernel (C=100)
Accuracy: 0.5433
RBF Kernel (C=1)
Accuracy: 0.9533
RBF Kernel (C=100)
Accuracy: 0.9833
X1
X2

```python
# ==== XOR Problem with Decision Tree and Random Forest ====

import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from matplotlib.colors import ListedColormap

# --- Import plot_decision_regions function ---
def plot_decision_regions(X, y, classifier, test_idx=None,
resolution=0.02):
    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],
                    y=X[y == cl, 1],
                    alpha=0.8,
                    c=colors[idx],
                    marker=markers[idx],
                    label=cl,
                    edgecolor='black')

    # highlight test samples
    if test_idx:
        # plot all samples
        X_test, y_test = X[test_idx, :], y[test_idx]

        plt.scatter(X_test[:, 0],
                    X_test[:, 1],
                    c='white',
                    edgecolor='black',
                    alpha=0.4,
                    linewidth=1,
```

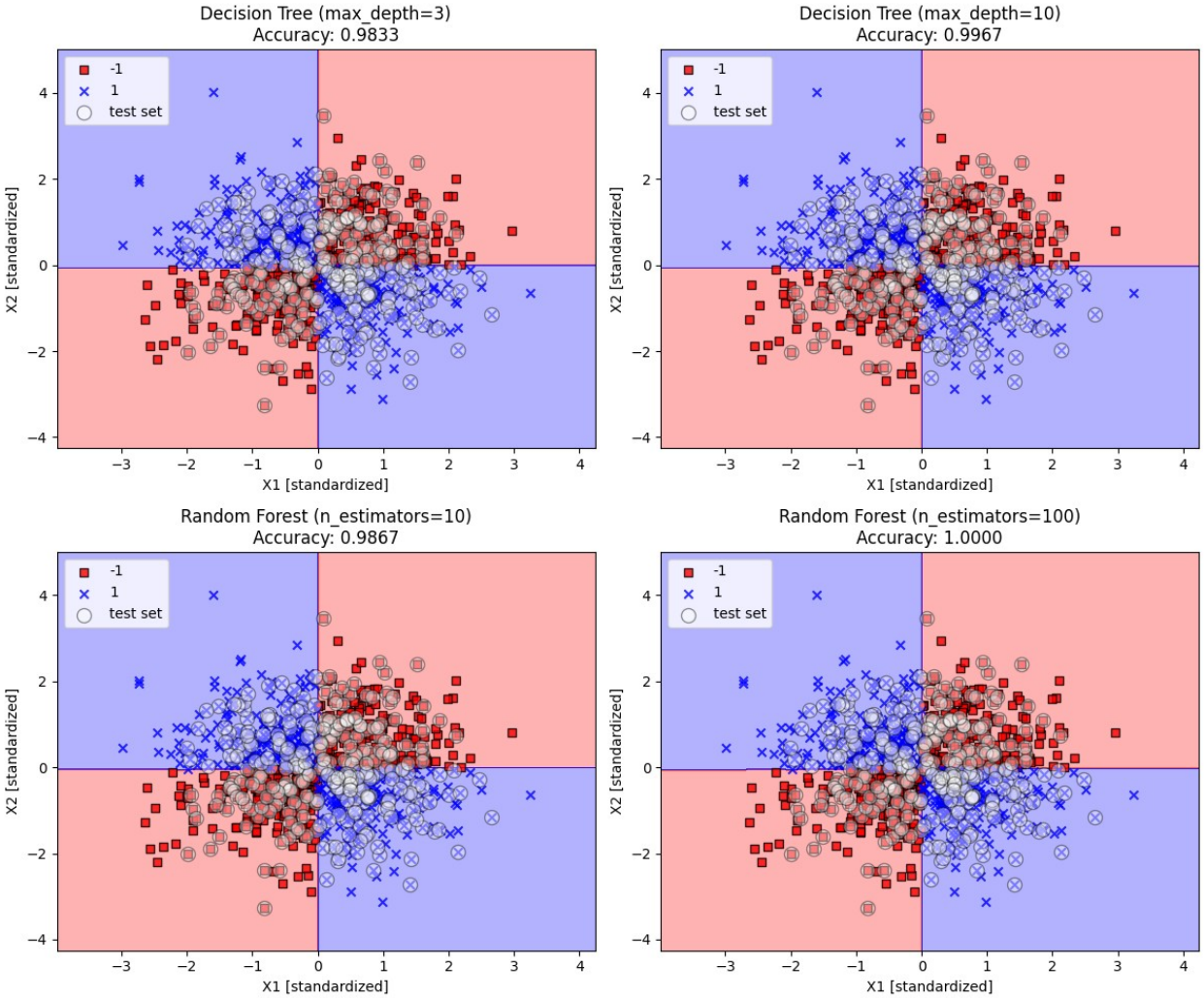
```
                    marker='o',
                    s=100,
                    label='test set')

# --- Generate XOR dataset with 1000 samples ---
np.random.seed(1)
X_xor = np.random.randn(1000, 2)
y_xor = np.logical_xor(X_xor[:, 0] > 0, X_xor[:, 1] > 0)
y_xor = np.where(y_xor, 1, -1)

# --- Splitting data into 70% training and 30% test data ---
X_train, X_test, y_train, y_test = train_test_split(
    X_xor, y_xor, test_size=0.3, random_state=42)

# --- Standardizing the features ---
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

# --- Combine training and test data for plotting ---
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))
test_idx = range(len(X_train_std), len(X_combined_std))

# --- Build Decision Tree models with two different depths ---
dt_shallow = DecisionTreeClassifier(max_depth=3, random_state=42)
dt_shallow.fit(X_train_std, y_train)

dt_deep = DecisionTreeClassifier(max_depth=10, random_state=42)
dt_deep.fit(X_train_std, y_train)

# --- Build Random Forest models with two different settings ---
rf_few = RandomForestClassifier(n_estimators=10, max_depth=5,
random_state=42)
rf_few.fit(X_train_std, y_train)

rf_many = RandomForestClassifier(n_estimators=100, max_depth=5,
random_state=42)
rf_many.fit(X_train_std, y_train)

# --- Test the models ---
y_pred_dt_shallow = dt_shallow.predict(X_test_std)
accuracy_dt_shallow = accuracy_score(y_test, y_pred_dt_shallow)
print(f"Decision Tree (max_depth=3) Accuracy:
{accuracy_dt_shallow:.4f}")

y_pred_dt_deep = dt_deep.predict(X_test_std)
accuracy_dt_deep = accuracy_score(y_test, y_pred_dt_deep)
print(f"Decision Tree (max_depth=10) Accuracy:
```

```
{accuracy_dt_deep:.4f}")

y_pred_rf_few = rf_few.predict(X_test_std)
accuracy_rf_few = accuracy_score(y_test, y_pred_rf_few)
print(f"Random Forest (n_estimators=10) Accuracy:
{accuracy_rf_few:.4f}")

y_pred_rf_many = rf_many.predict(X_test_std)
accuracy_rf_many = accuracy_score(y_test, y_pred_rf_many)
print(f"Random Forest (n_estimators=100) Accuracy:
{accuracy_rf_many:.4f}")

# --- Plot the decision boundaries ---
plt.figure(figsize=(12, 10))

# Decision Tree (max_depth=3)
plt.subplot(2, 2, 1)
plot_decision_regions(X_combined_std, y_combined,
classifier=dt_shallow, test_idx=test_idx)
plt.title(f'Decision Tree (max_depth=3)\nAccuracy:
{accuracy_dt_shallow:.4f}')
plt.xlabel('X1 [standardized]')
plt.ylabel('X2 [standardized]')
plt.legend(loc='upper left')

# Decision Tree (max_depth=10)
plt.subplot(2, 2, 2)
plot_decision_regions(X_combined_std, y_combined, classifier=dt_deep,
test_idx=test_idx)
plt.title(f'Decision Tree (max_depth=10)\nAccuracy:
{accuracy_dt_deep:.4f}')
plt.xlabel('X1 [standardized]')
plt.ylabel('X2 [standardized]')
plt.legend(loc='upper left')

# Random Forest (n_estimators=10)
plt.subplot(2, 2, 3)
plot_decision_regions(X_combined_std, y_combined, classifier=rf_few,
test_idx=test_idx)
plt.title(f'Random Forest (n_estimators=10)\nAccuracy:
{accuracy_rf_few:.4f}')
plt.xlabel('X1 [standardized]')
plt.ylabel('X2 [standardized]')
plt.legend(loc='upper left')

# Random Forest (n_estimators=100)
plt.subplot(2, 2, 4)
plot_decision_regions(X_combined_std, y_combined, classifier=rf_many,
test_idx=test_idx)
plt.title(f'Random Forest (n_estimators=100)\nAccuracy:
```

```python
{accuracy_rf_many:.4f}')
plt.xlabel('X1 [standardized]')
plt.ylabel('X2 [standardized]')
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()
```

```
Decision Tree (max_depth=3) Accuracy: 0.9833
Decision Tree (max_depth=10) Accuracy: 0.9967
Random Forest (n_estimators=10) Accuracy: 0.9867
Random Forest (n_estimators=100) Accuracy: 1.0000
```

```
<ipython-input-1-408c174f82d5>:31: UserWarning: You passed a
edgecolor/edgecolors ('black') for an unfilled marker ('x').
Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
behavior may change in the future.
  plt.scatter(x=X[y == cl, 0],
<ipython-input-1-408c174f82d5>:31: UserWarning: You passed a
edgecolor/edgecolors ('black') for an unfilled marker ('x').
Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
behavior may change in the future.
  plt.scatter(x=X[y == cl, 0],
<ipython-input-1-408c174f82d5>:31: UserWarning: You passed a
edgecolor/edgecolors ('black') for an unfilled marker ('x').
Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
behavior may change in the future.
  plt.scatter(x=X[y == cl, 0],
<ipython-input-1-408c174f82d5>:31: UserWarning: You passed a
edgecolor/edgecolors ('black') for an unfilled marker ('x').
Matplotlib is ignoring the edgecolor in favor of the facecolor.  This
behavior may change in the future.
  plt.scatter(x=X[y == cl, 0],
```


Decision Tree (max_depth=3)
Accuracy: 0.9833
Decision Tree (max_depth=10)
Accuracy: 0.9967
Random Forest (n_estimators=10)
Accuracy: 0.9867
Random Forest (n_estimators=100)
Accuracy: 1.0000
-1
1
test set
X1 [standardized]
X2 [standardized]