HW #9

1. a. Linear classifiers are not as expressive as decision trees because they can only model linear decision boundaries, unless you use transformations like basis expansion.

b. SVMs are more expressive than linear classifiers, but not as expressive as decision trees. They can create quadratic/nonlinear boundaries, but are still limited to quadratic functions.

c. NN classifiers are similarly expressive because w/ a large training set and small neighborhood size, they can also represent any decision boundary. Like decision trees, however, they may not generalize well.

d. They are not as expressive because they assume each class follows a Gaussian distribution, so this assumption may be incorrect, and they wouldn't be able to create decision boundaries as expressive.

2. $d(n-1)$ possibilities

$36 = 2(0.2)(0.8) = 0.32$

4. a. Given $W = \sum_i \lambda_i$, $W_j = \sum_{i|y^{(i)}=j} \lambda_i$, using and $p_j = \frac{W_j}{W}$; cont. index

$$G = 1 - \sum_{j=1}^{k} p_j^2 = 1 - \sum_{j=1}^{k} \left(\frac{W_j}{W}\right)^2$$

$$= \left[1 - \frac{\sum_{j=1}^{k} W_j^2}{W^2}\right]$$

→ p_j now considers weights for each pt; sum of pts x weights of one class over pts x weights of every pt

b. $\pi_j = \frac{\sum_{i|y^{(i)}=j} \lambda_i}{\sum \lambda_i}$ → points w/ higher weights now contribute more

$$\mu_j = \frac{\sum_{i|y^{(i)}=j} \lambda_i x^{(i)}}{\sum_{i|y^{(i)}=j} \lambda_i}$$ → weighted average

$$\Sigma_j = \frac{\sum_{i|y^{(i)}=j} \lambda_i (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i|y^{(i)}=j} \lambda_i}$$ → data pts w/ higher weights influence COV matrix more

c. $\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$ → normal optimization

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \lambda_i \xi_i$$

↓

modified function to include weights λ_i; so now misclassification of higher weighted pts cause larger penalty, which affects the decision boundary.

5. a. False, boosting reduces training error, but does not necessarily cause zero test error, which depends on how well the final classifier generalizes to new data.

b. True, since each weak classifier has error rate of $\frac{1}{2} - \epsilon$, it will keep reducing training error till it equals zero. The algorithm weighs misclassified points more heavily, so it focuses on them until all points are correctly classified.

c. True, the final classifier is a linear combination of weak classifiers from class H, so it will also belong to class H.

6. a. This is a benefit of random forests, where each decision tree is trained independently and thus can be trained in parallel.

b. Boosted decision trees, which trains trees sequentially, improving performance between trees and focusing on misclassified points from previous trees, so they are more highly optimized.

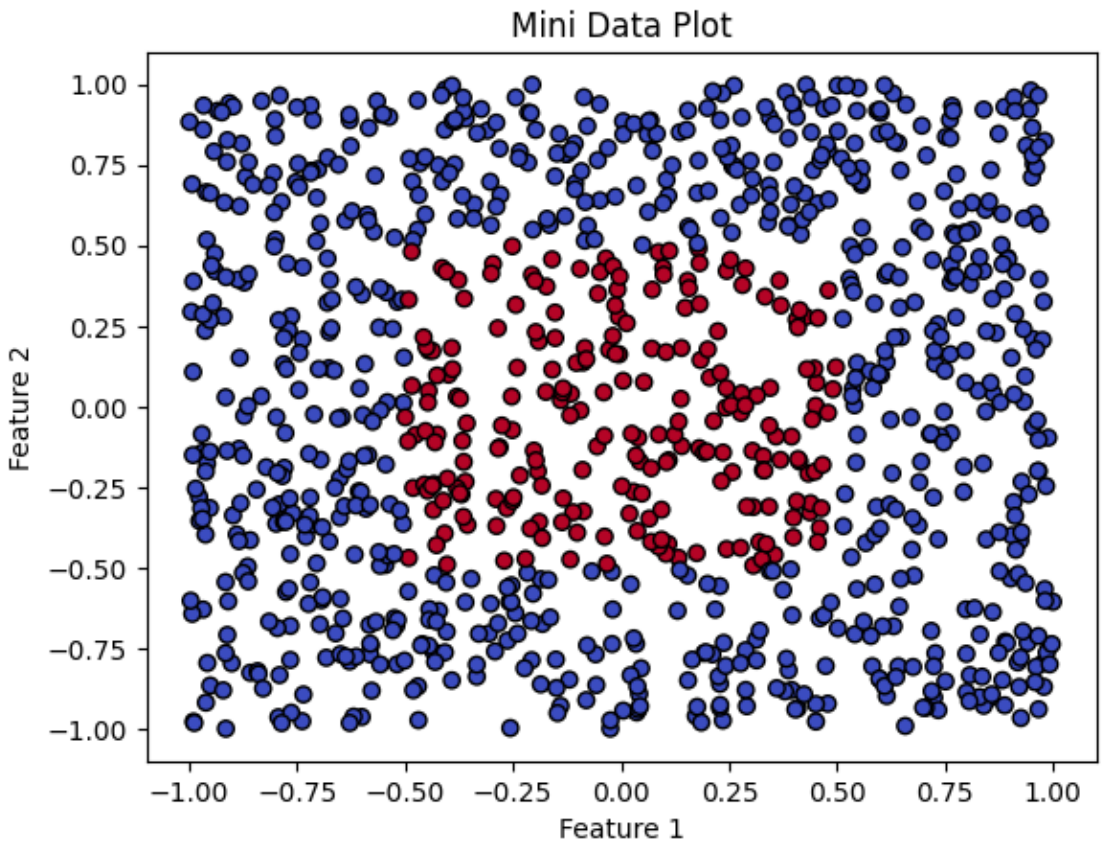
c. Boosted decision trees, because each tree improves accuracy as it reduces error from previous trees.

hw9

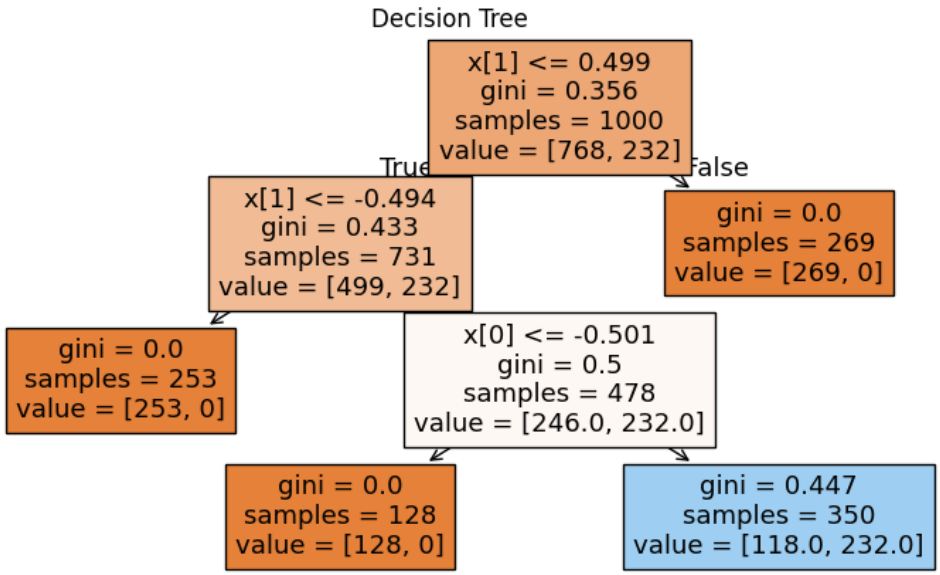
March 13, 2025

[50]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_predict
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪ConfusionMatrixDisplay
from sklearn.utils import resample
from matplotlib.colors import ListedColormap
```

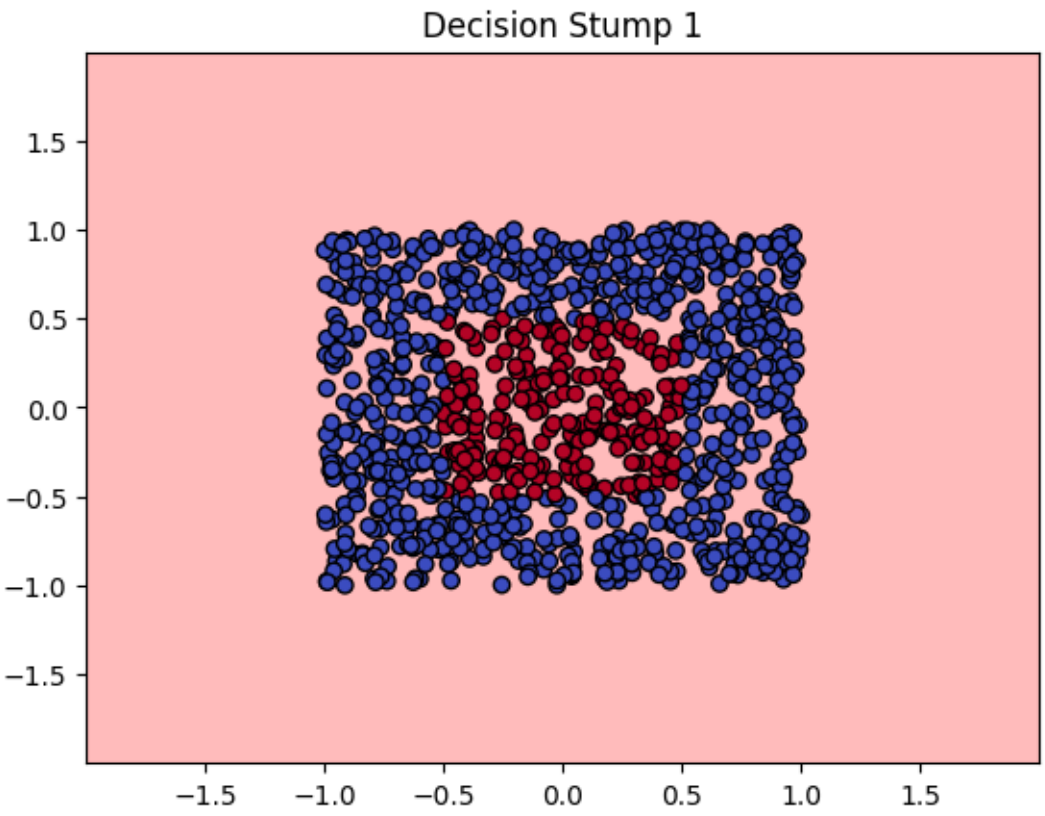
[2]:
```
data = np.loadtxt("mini-data.txt")
X, y = data[:, :2], data[:, 2]
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolors='k')
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("Mini Data Plot")
plt.show()
```

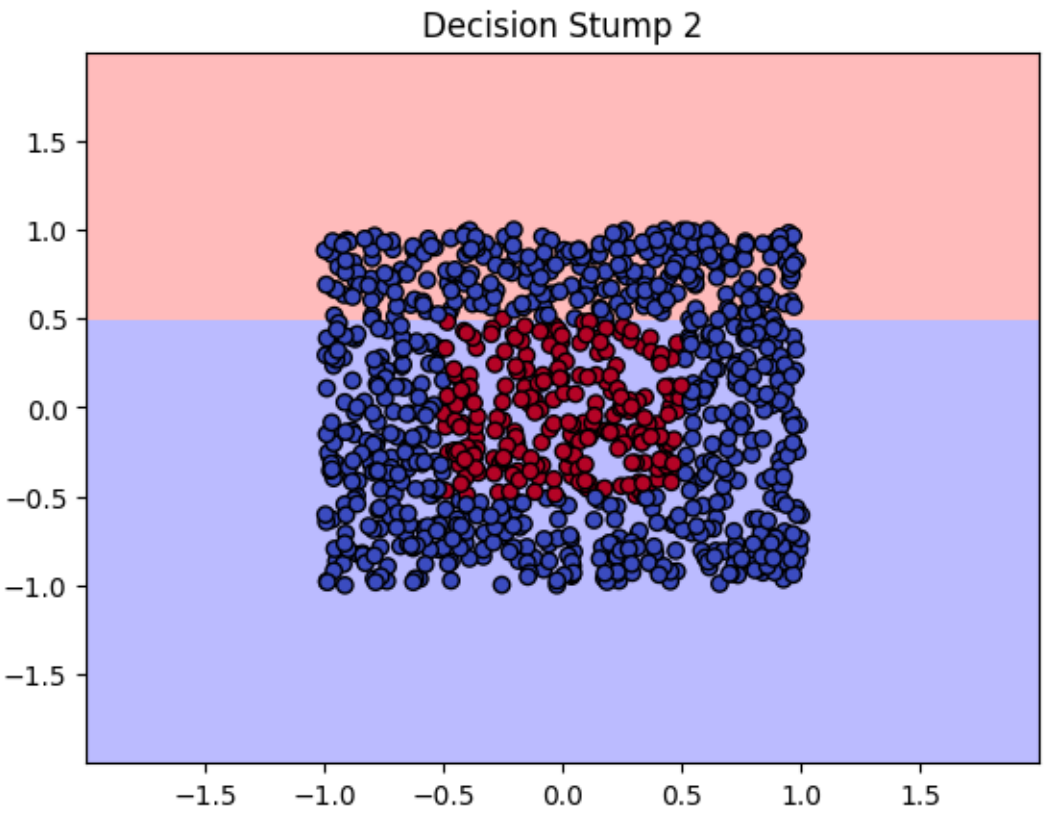


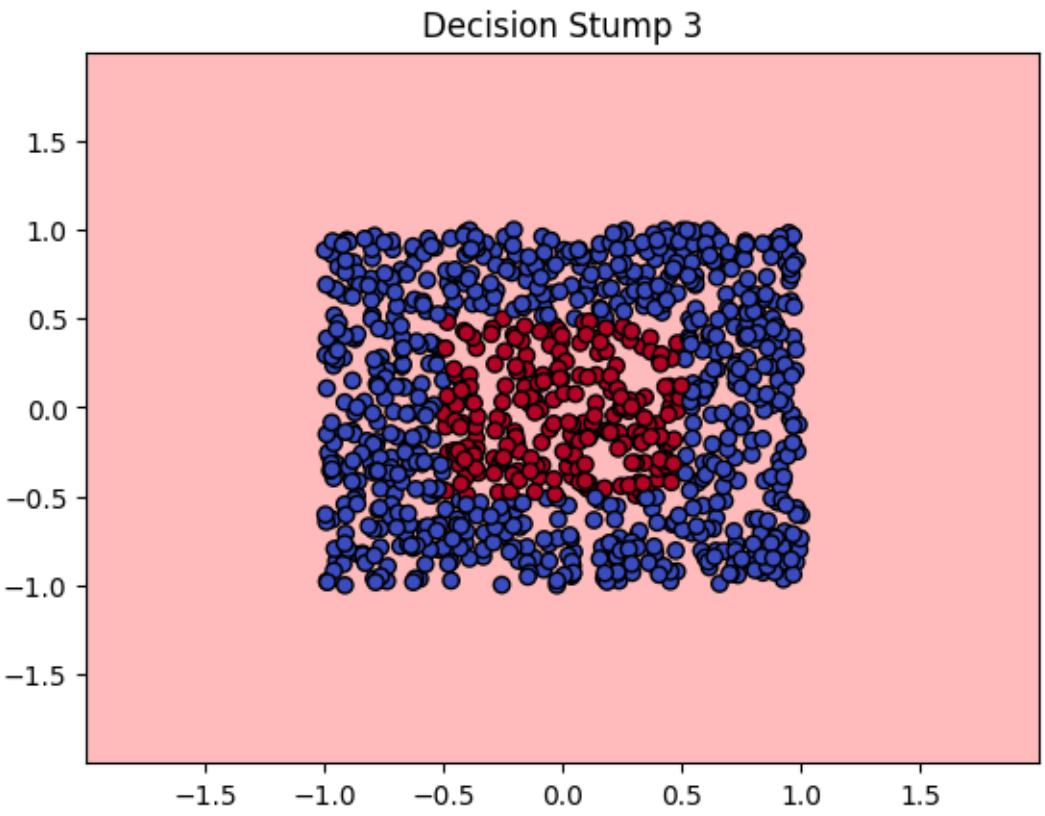
[48]:
```
# Stopping criteria used is max depth of decision tree, equal to 3
dt = DecisionTreeClassifier(max_depth=3)
dt.fit(X, y)
plt.figure(figsize=(10, 5))
plot_tree(dt, filled=True)
plt.title("Decision Tree")
plt.show()
```

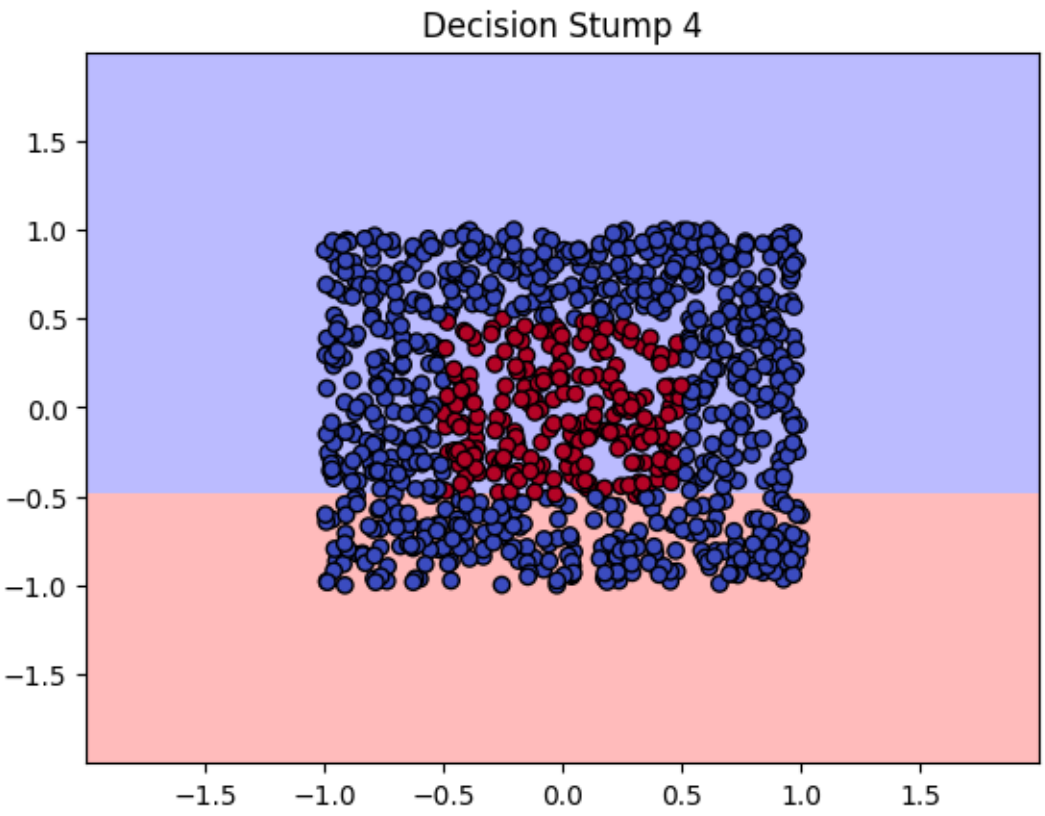


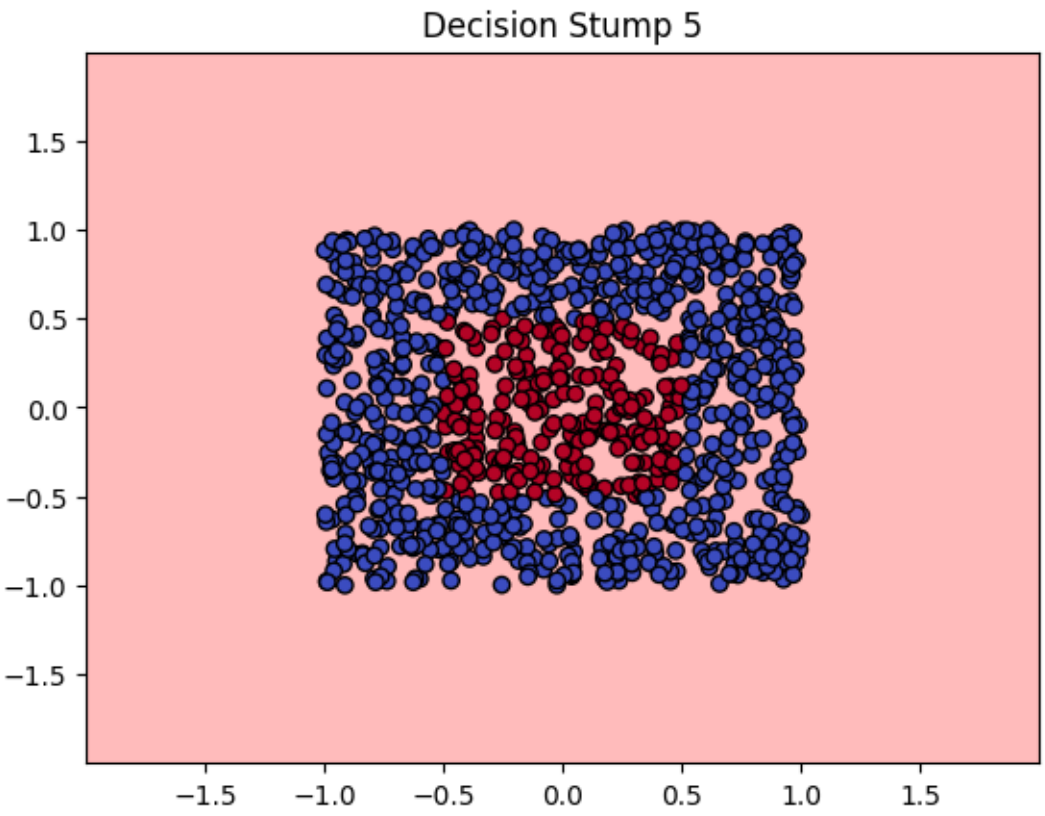
[51]:
```
ada = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=15)
ada.fit(X, y)

cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF'])
cmap_bold = ['r', 'b']
xx, yy = np.meshgrid(np.linspace(X[:, 0].min() - 1, X[:, 0].max() + 1, 100),
                     np.linspace(X[:, 1].min() - 1, X[:, 1].max() + 1, 100))

for i, stump in enumerate(ada.estimators_):
    Z = stump.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolors='k')
    plt.title(f"Decision Stump {i+1}")
    plt.show()

train_errors = [accuracy_score(y, pred) for pred in ada.staged_predict(X)]
print("Boosting Accuracy per Stump:")
for i, acc in enumerate(train_errors, start=1):
    print(f"Stump {i}: {acc:.4f}")
```

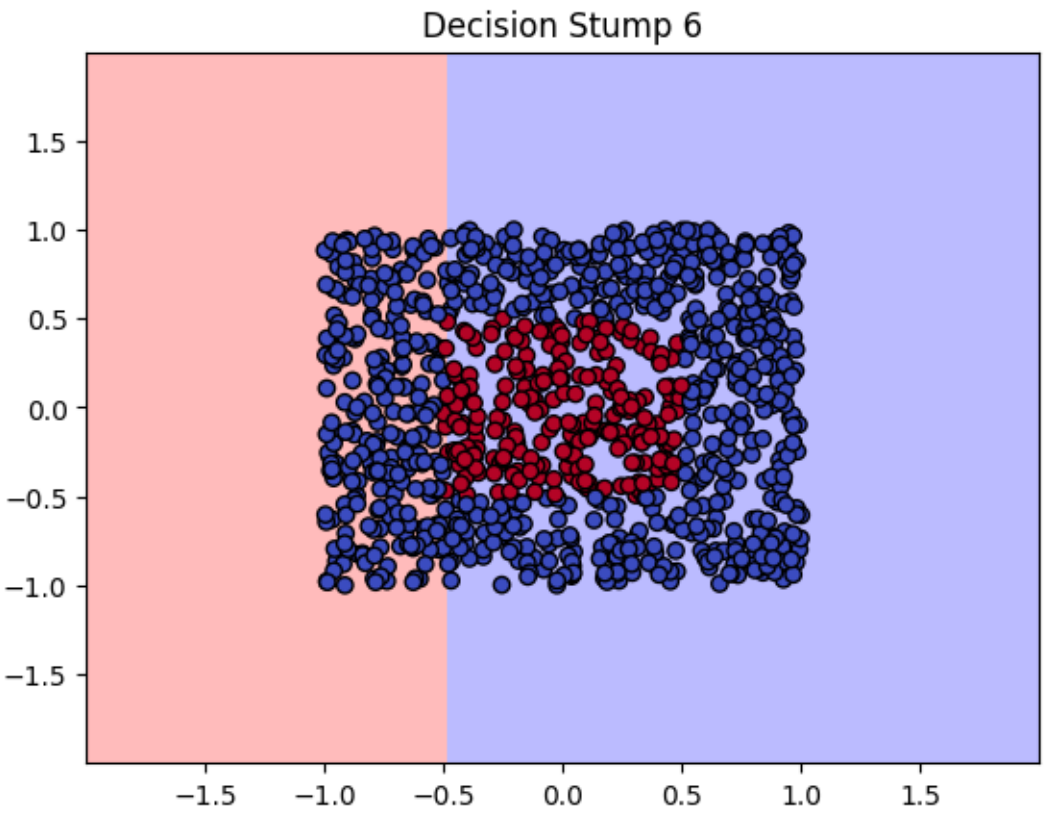

Decision Stump 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

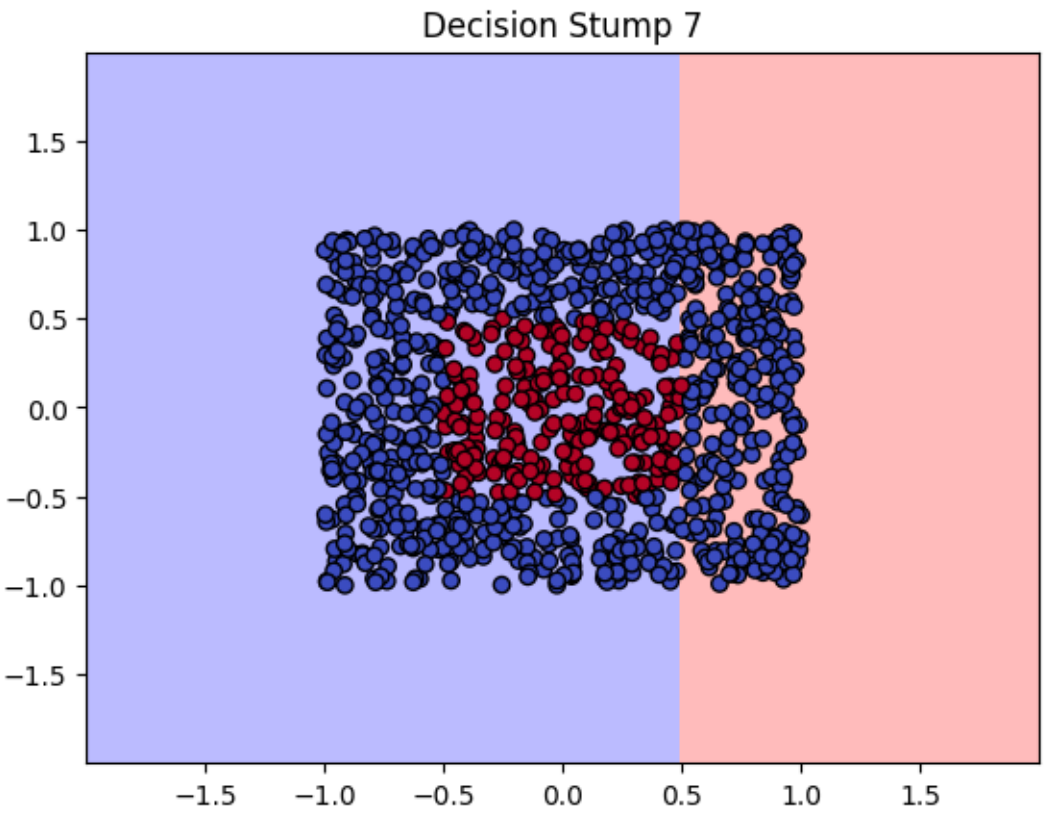

Decision Stump 2
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

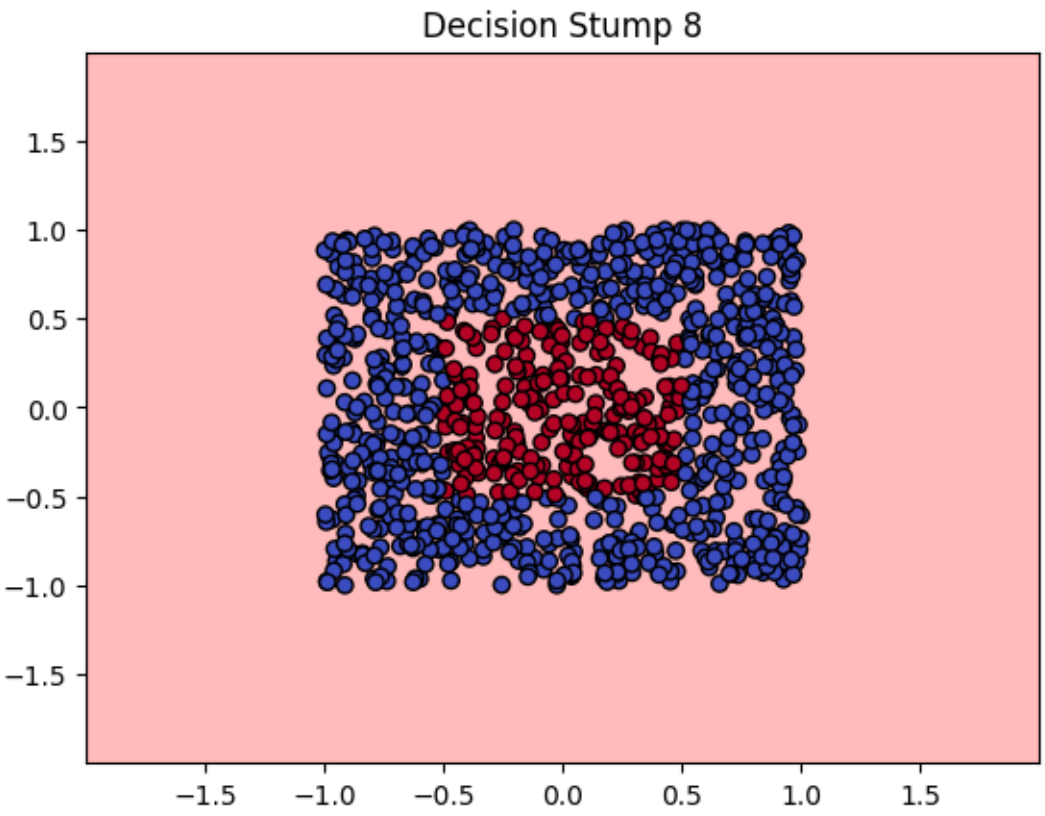

Decision Stump 3
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

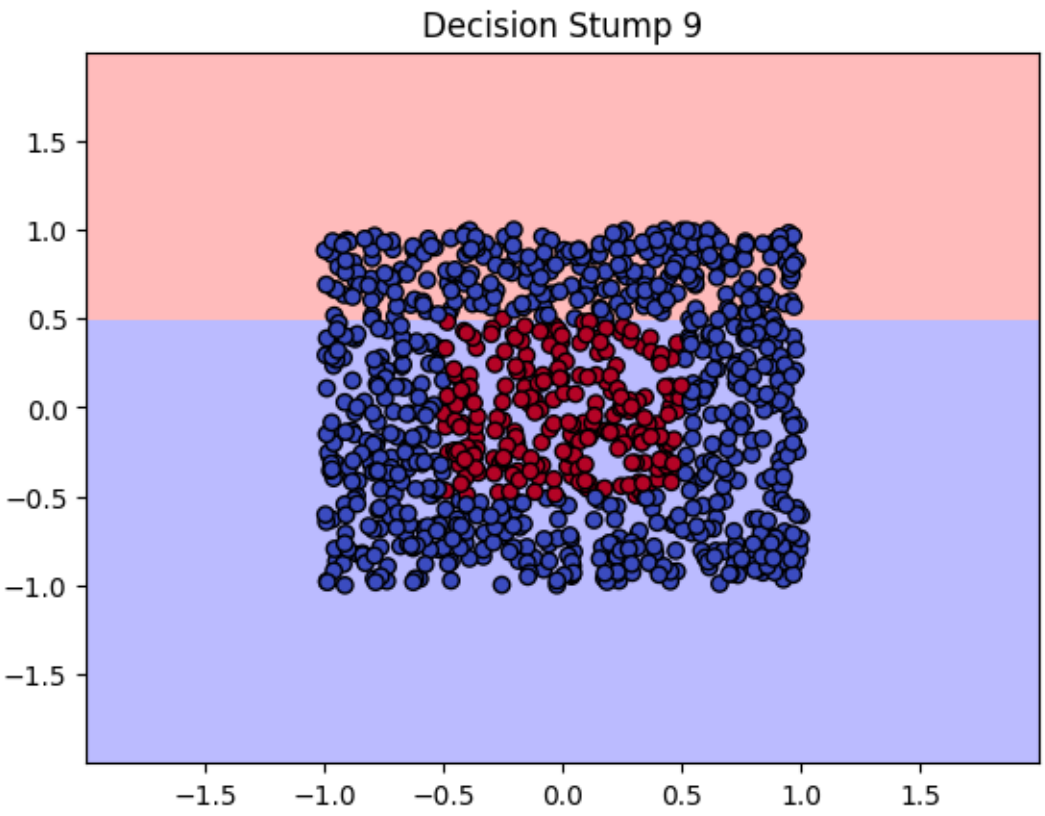

Decision Stump 4
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

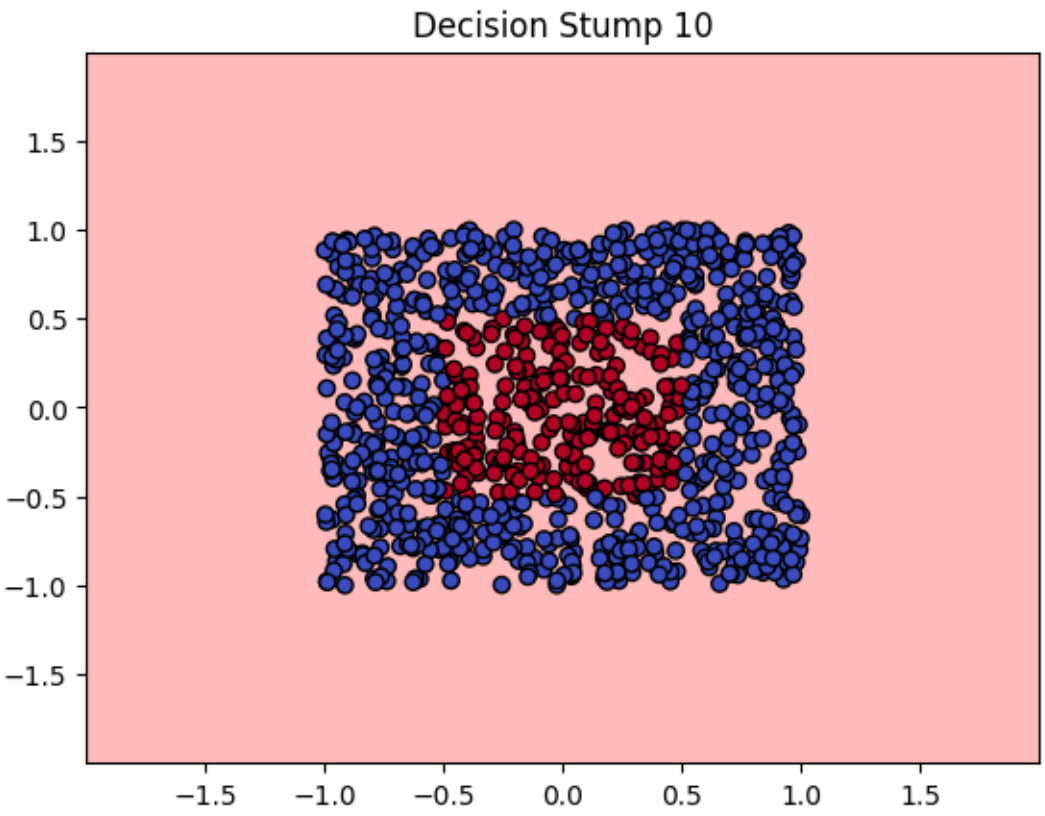

Decision Stump 5
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

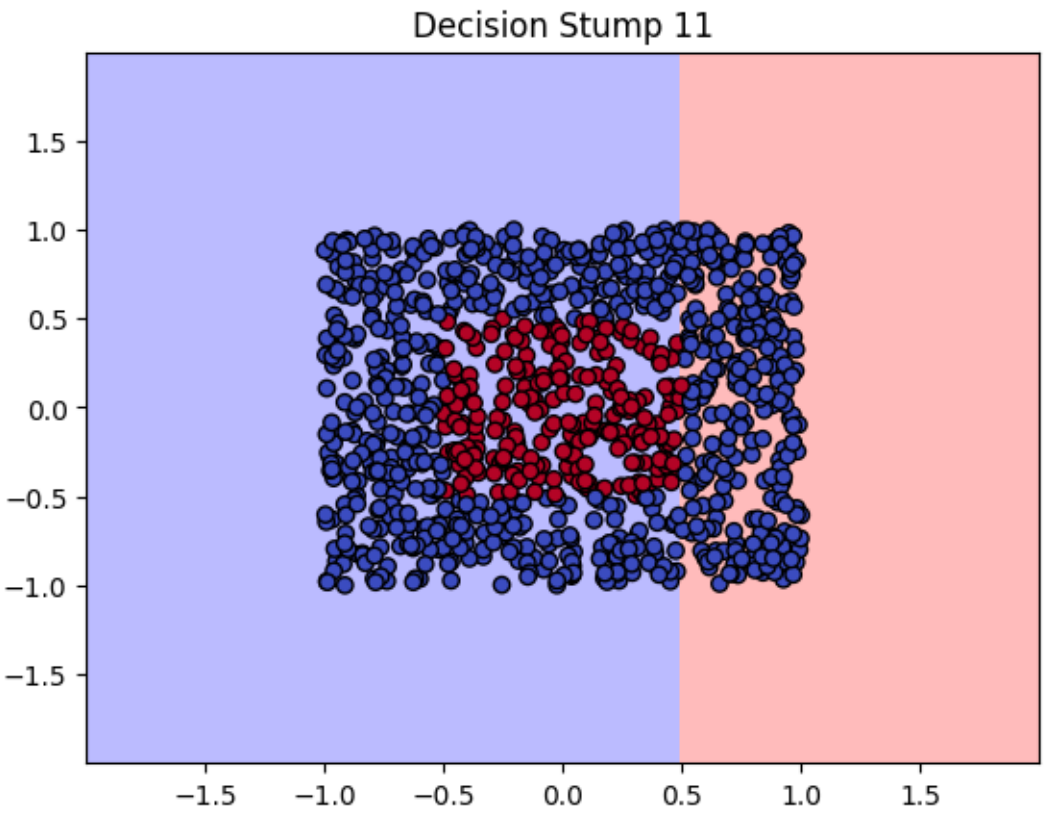

Decision Stump 6
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

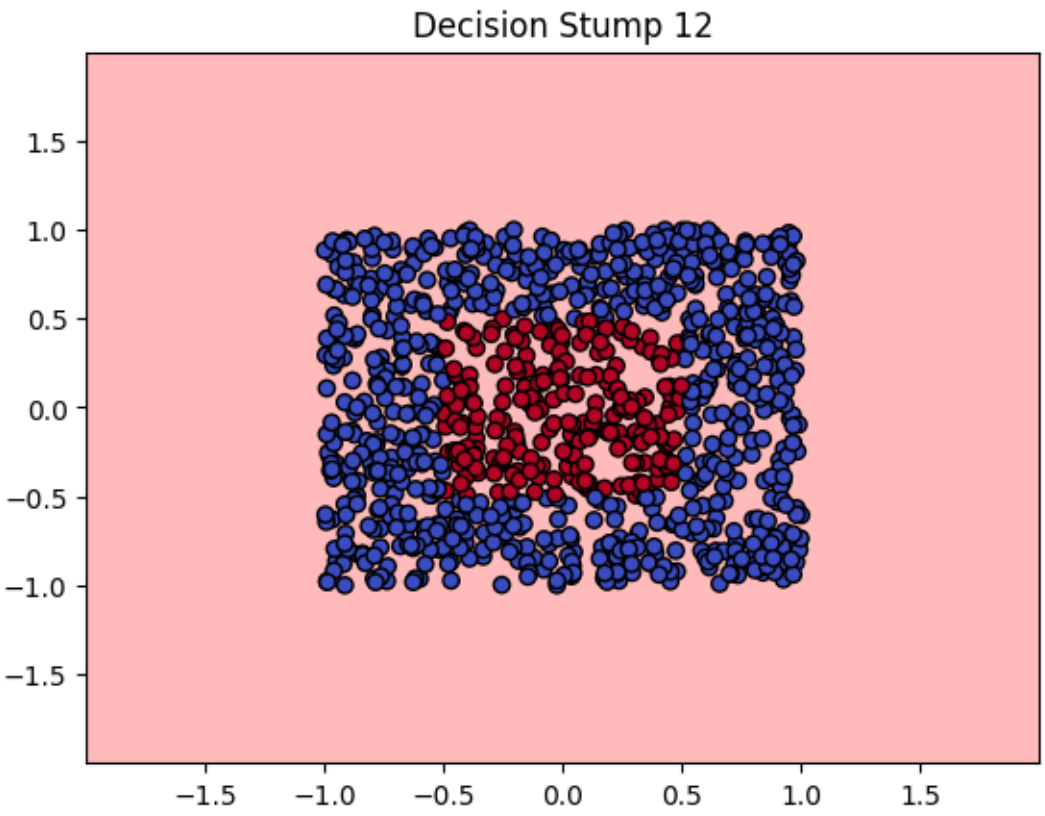

Decision Stump 7
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

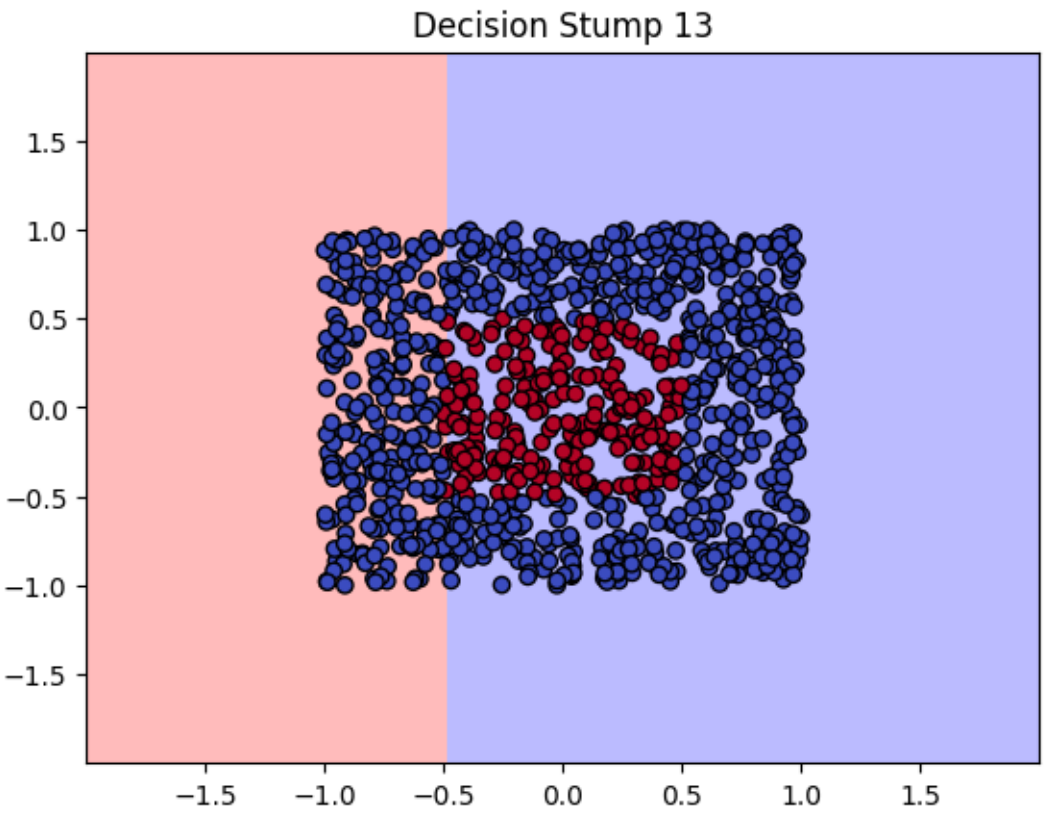

Decision Stump 8
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

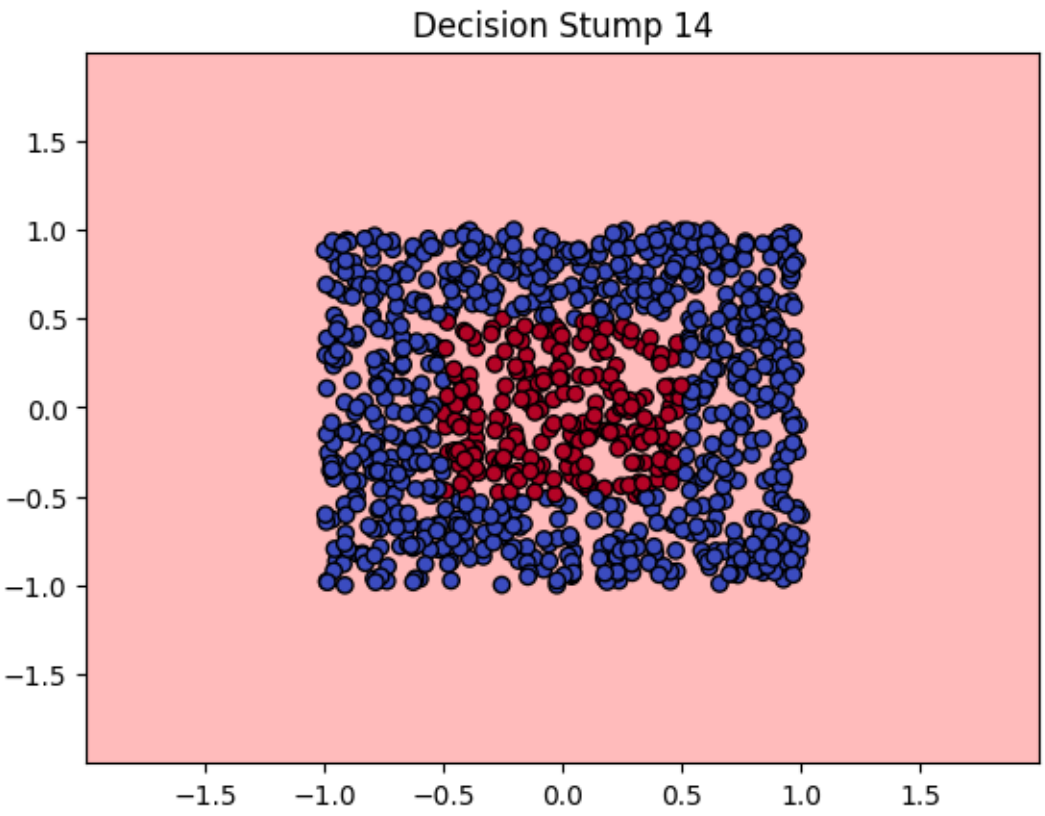

Decision Stump 9
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

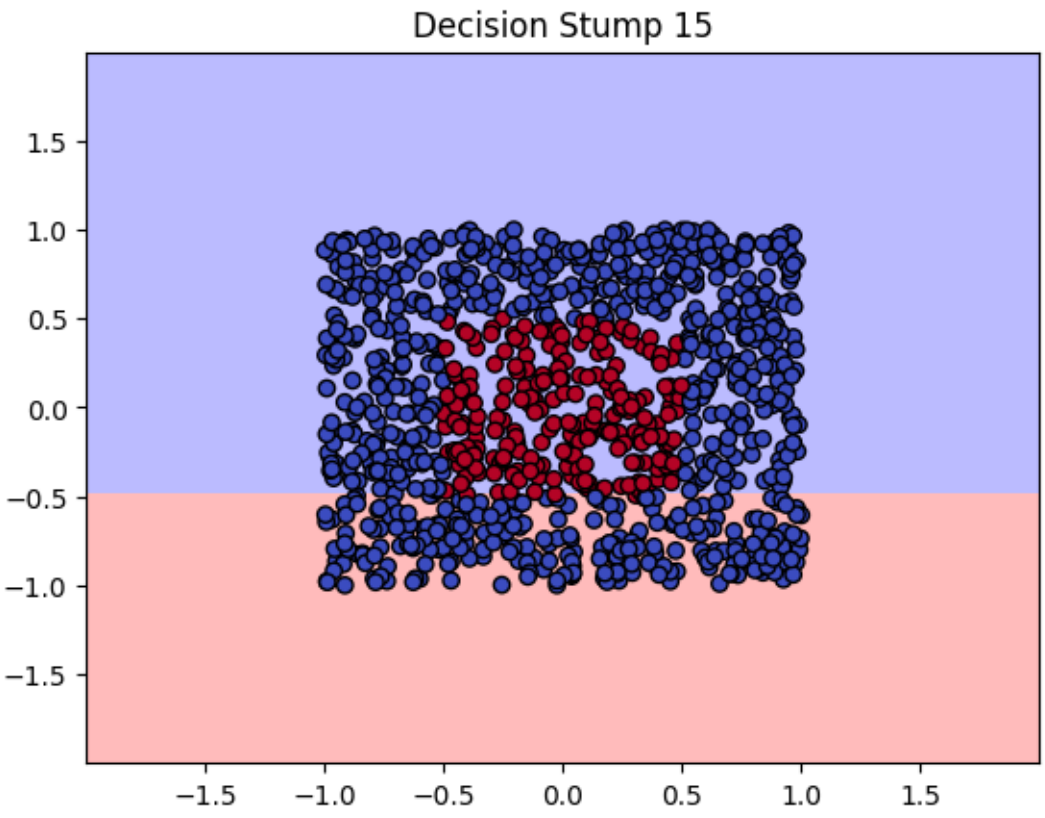

Decision Stump 10
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Decision Stump 11
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Decision Stump 12
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Decision Stump 13
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Decision Stump 14
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5




```
Boosting Accuracy per Stump:
Stump 1: 0.7680
Stump 2: 0.7680
Stump 3: 0.7680
Stump 4: 0.7680
Stump 5: 0.7680
Stump 6: 0.8820
Stump 7: 1.0000
Stump 8: 0.7680
Stump 9: 1.0000
Stump 10: 0.7680
Stump 11: 1.0000
Stump 12: 1.0000
Stump 13: 1.0000
Stump 14: 1.0000
Stump 15: 1.0000
```

[46]:
```
df = pd.read_csv("creditcard.csv")
print(df['Class'].value_counts())
```

```
Class
0    284315
```

```
1       492
Name: count, dtype: int64
```

0.1 Problem 7a. There are 492 fradulent cases. This can be problematic because models focus on maximizing overall accuracy, which may lead to poor performance on classifying the class with less examples, in this dataset the fradulent cases.

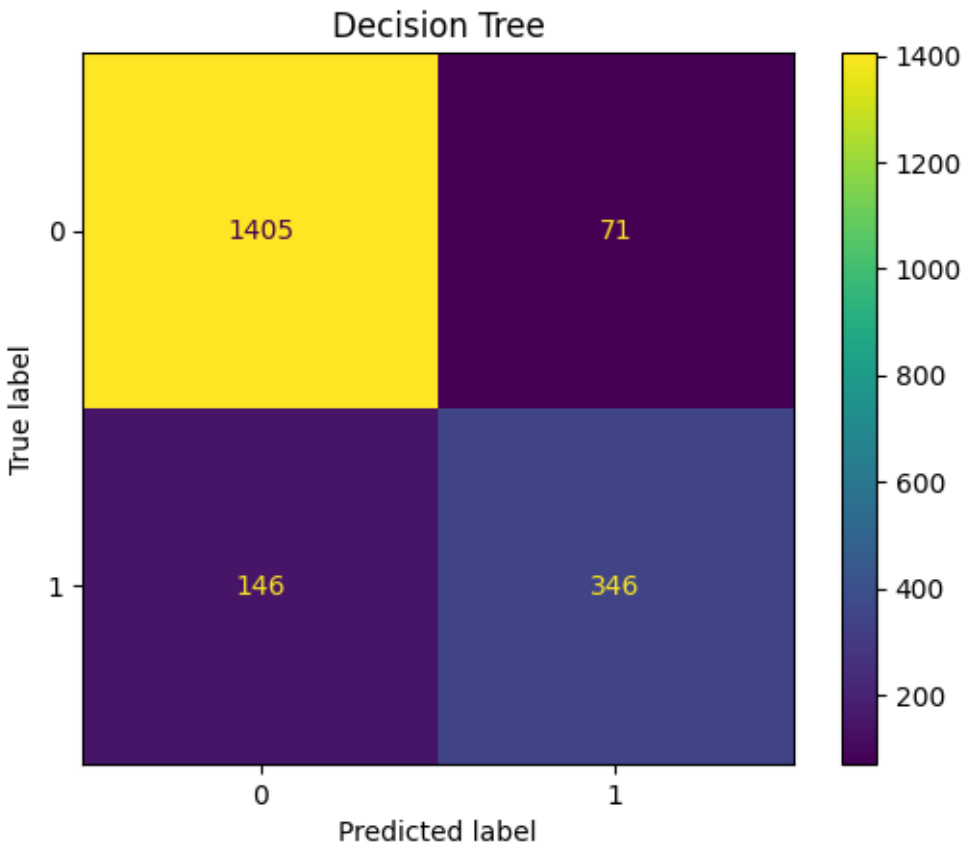
[52]:
```
legit = df[df['Class'] == 0]
fraud = df[df['Class'] == 1]
legit_downsampled = resample(legit, replace=False, n_samples=len(fraud) * 3,␣
 ↪random_state=42)
df_balanced = pd.concat([legit_downsampled, fraud])
X_balanced = df_balanced.drop(columns=['Class'])
y_balanced = df_balanced['Class']
print(df_balanced['Class'].value_counts())
```

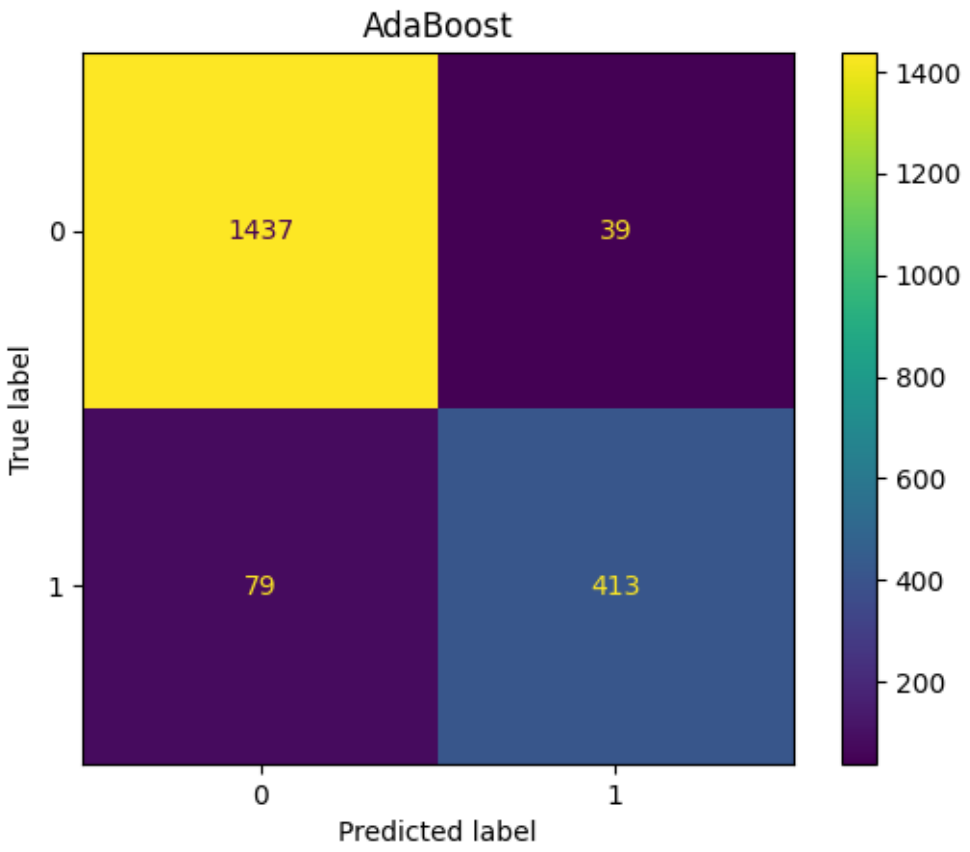
```
Class
0    1476
1     492
Name: count, dtype: int64
```

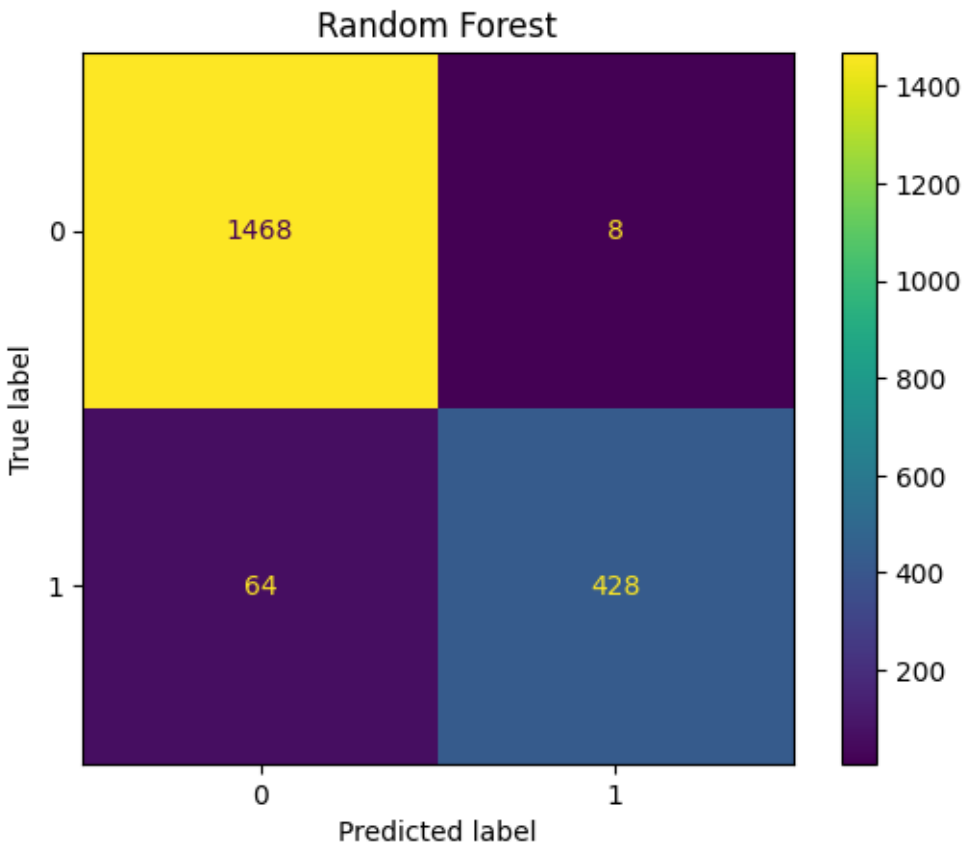
[6]:
```
X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced,␣
 ↪test_size=0.2, random_state=42)
```

[18]:
```
dt_fraud = DecisionTreeClassifier()
ada_fraud = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),␣
 ↪n_estimators=4)
rf_fraud = RandomForestClassifier()
```

[19]:
```
dt_pred = cross_val_predict(dt_fraud, X_balanced, y_balanced, cv=5)
ada_pred = cross_val_predict(ada_fraud, X_balanced, y_balanced, cv=5)
rf_pred = cross_val_predict(rf_fraud, X_balanced, y_balanced, cv=5)

for name, pred in zip(["Decision Tree", "AdaBoost", "Random Forest"], [dt_pred,␣
 ↪ada_pred, rf_pred]):
    cm = confusion_matrix(y_balanced, pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot()
    plt.title(name)
    plt.show()
```


Decision Tree
1405
71
146
346
0
1
True label
0
1
Predicted label
1400
1200
1000
800
600
400
200


AdaBoost
1437
39
79
413
0
1
True label
0
1
Predicted label
1400
1200
1000
800
600
400
200


Random Forest
1468
8
64
428
0
1
True label
0
1
Predicted label
1400
1200
1000
800
600
400
200

[ ]: