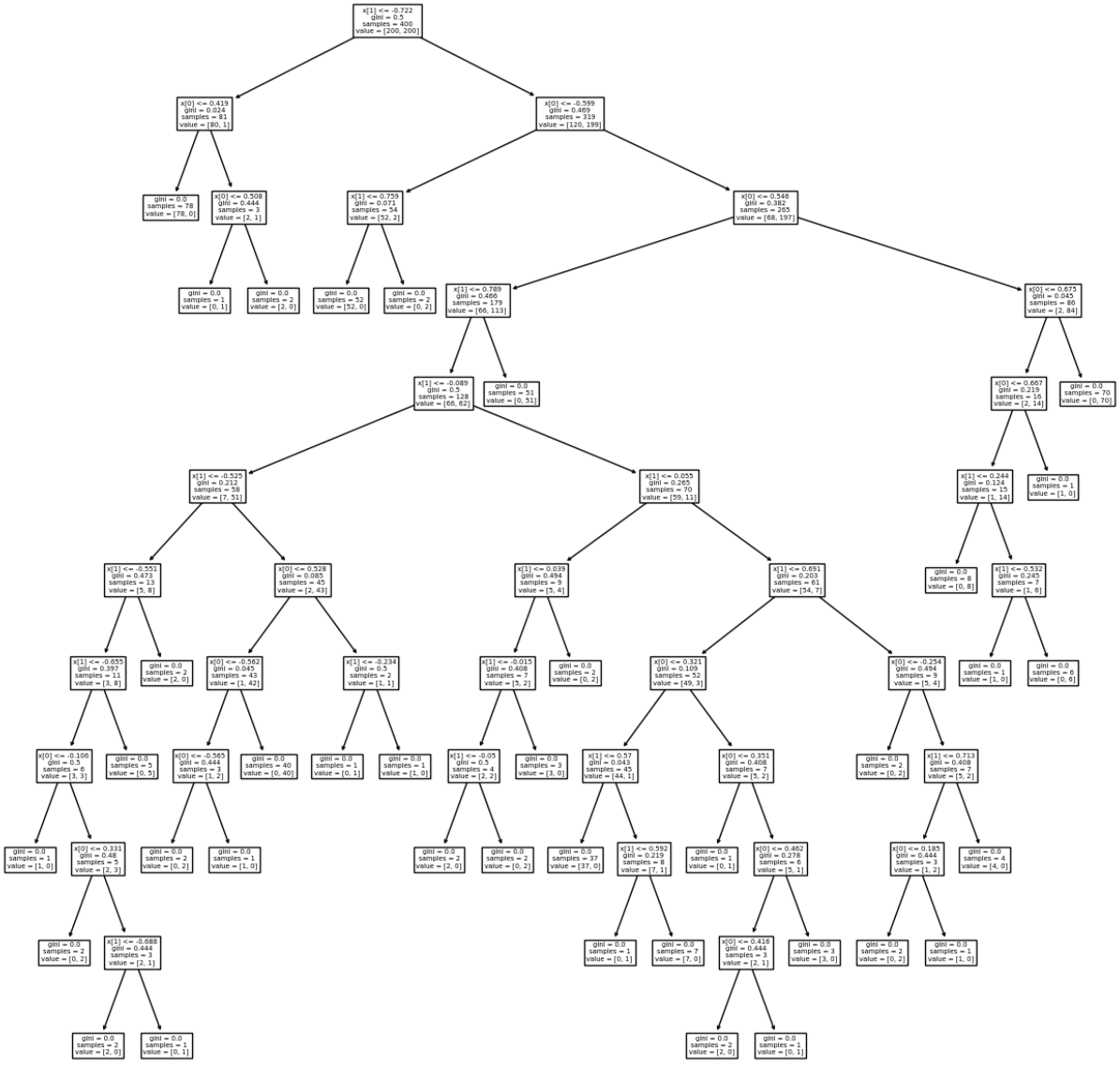
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve
from sklearn import metrics


horseshoes = np.genfromtxt('dataSetHorseshoes.csv',delimiter=',')
X = horseshoes[:,1:3]
Y = horseshoes[:,0]
```

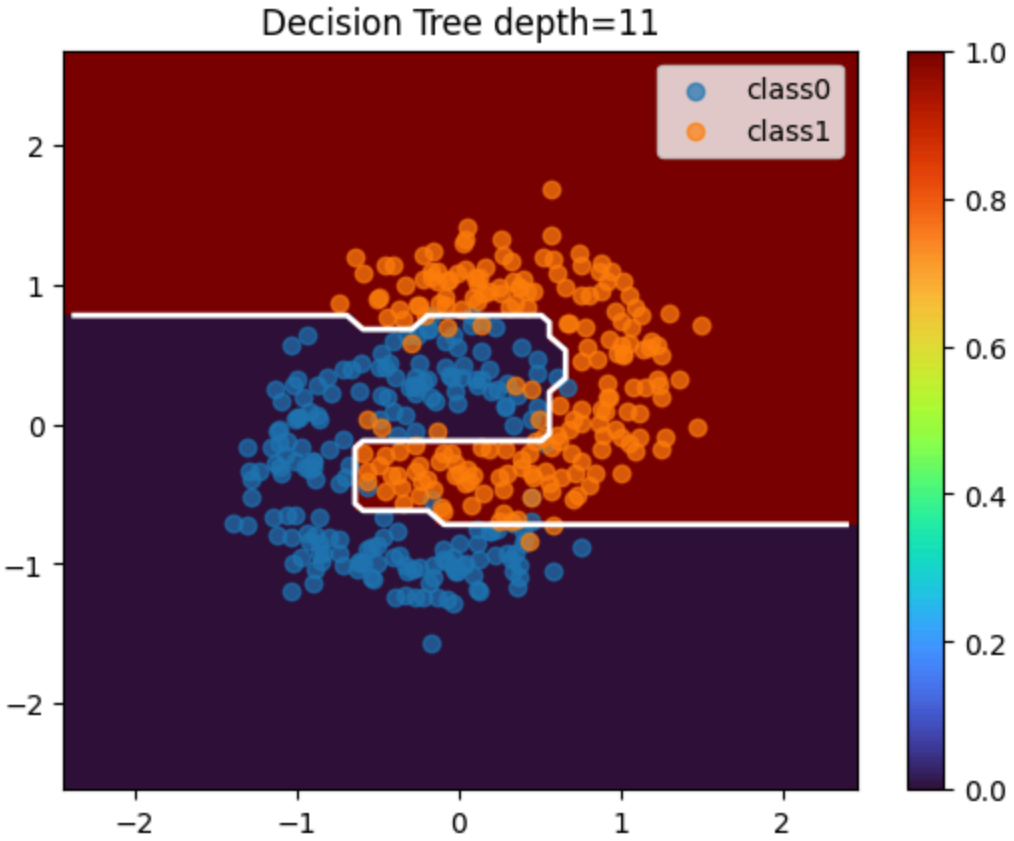
Decision trees and random forests

In [ ]:
```
def makeDecisionTree(depth=None):
    decisionTree = DecisionTreeClassifier(random_state=0, max_depth=depth)
    decisionTree.fit(X,Y)
    plt.figure(figsize=(15,15))
    plot_tree(decisionTree)
    return decisionTree

def visualizeDecisionSurface(tree, title):
    xx,yy = np.meshgrid(np.arange(np.min(X[:,0])-1, np.max(X[:,0])+1, 0.1),np.arang
    xygrid = np.c_[xx.ravel(), yy.ravel()]
    preds = tree.predict(xygrid)
    decisionstats = tree.predict_proba(xygrid)[:,1]

    fig, ax = plt.subplots()
    im = ax.pcolormesh(xx, yy, decisionstats.reshape(xx.shape), cmap="turbo")
    fig.colorbar(im)
    class0 = horseshoes[horseshoes[:,0]==0, :]
    class1 = horseshoes[horseshoes[:,0]==1, :]
    ax.scatter(class0[:,1],class0[:,2], label="class0", alpha=0.7)
    ax.scatter(class1[:,1],class1[:,2], label="class1", alpha=0.7)
    ax.contour(xx, yy, preds.reshape(xx.shape), levels=[0.5], colors="white", linew

    ax.legend()
    ax.set_title(title)
    plt.show()
```

1)

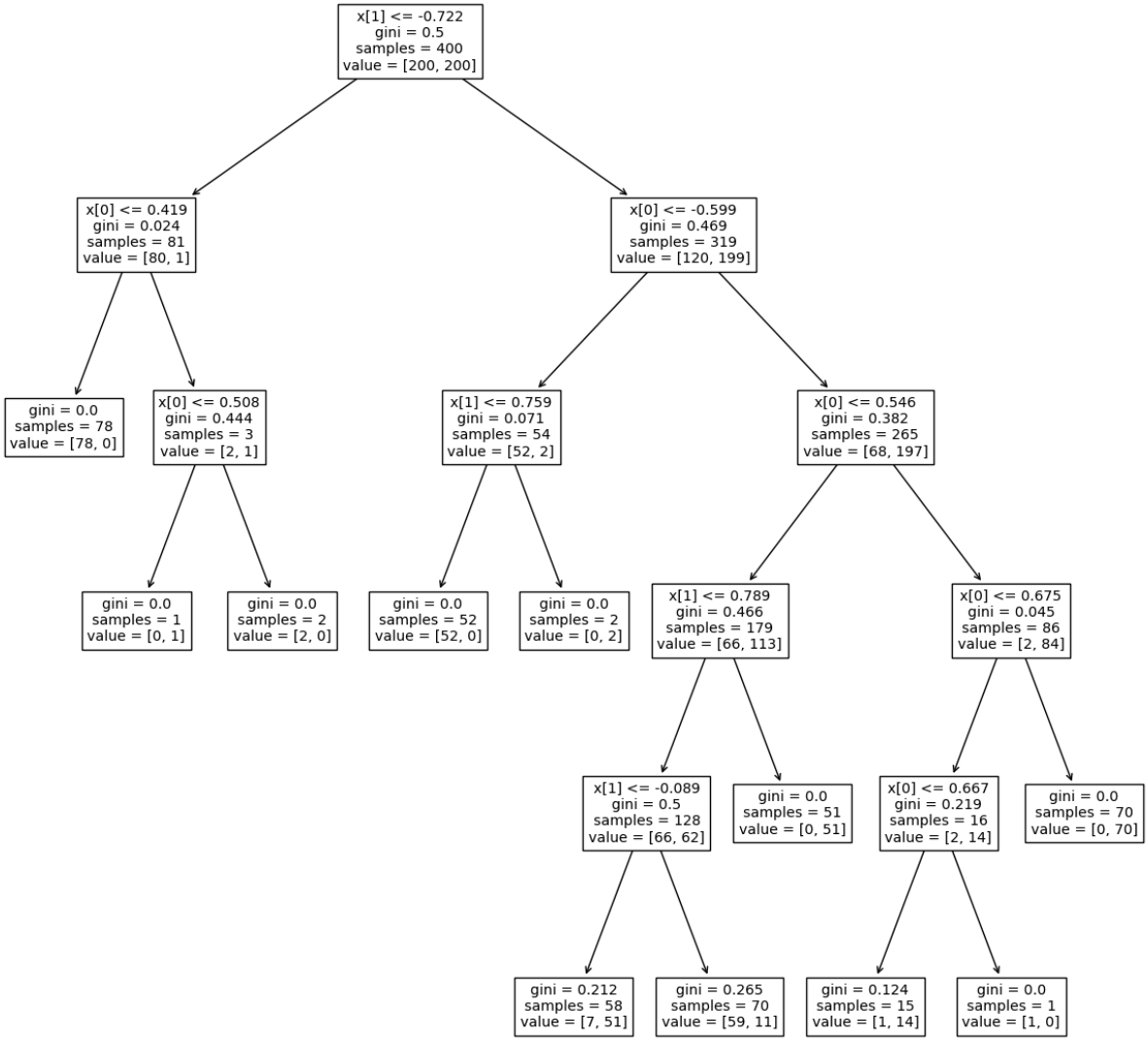
In [ ]:
```
maxTree = makeDecisionTree()
visualizeDecisionSurface(maxTree, f"Decision Tree depth={maxTree.get_depth()}")
```

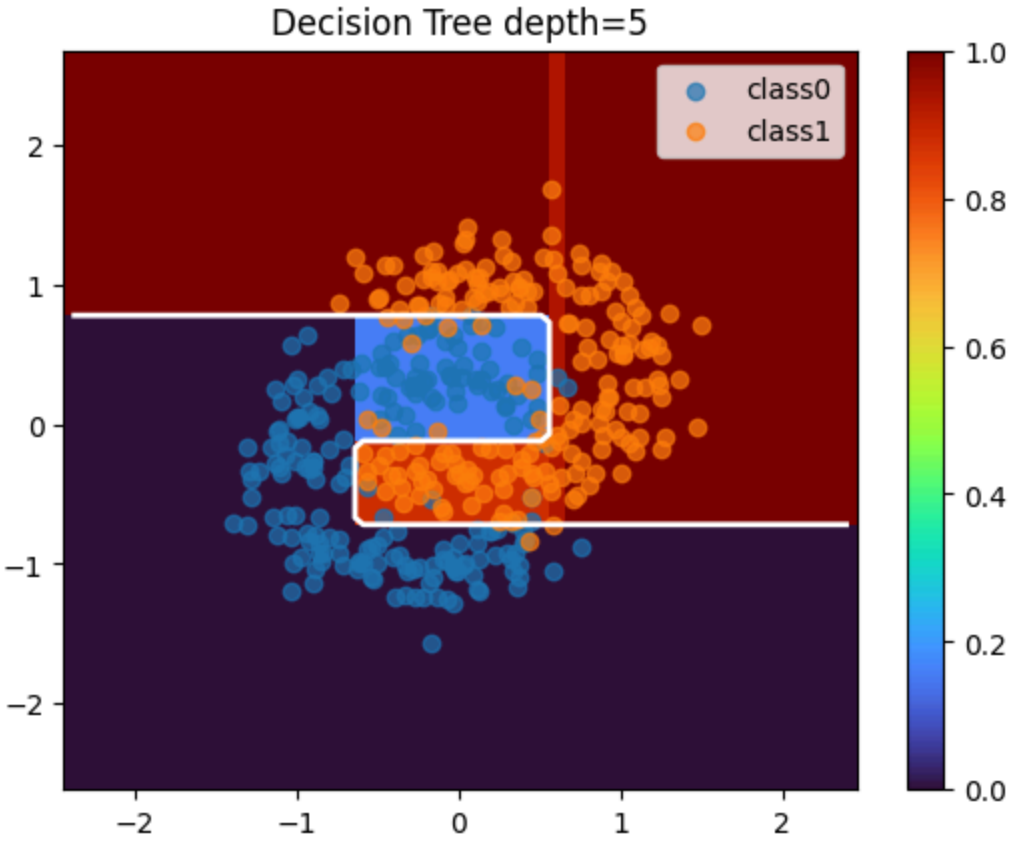

x[1] <= -0.722
gini = 0.5
samples = 400
value = [200, 200]
x[0] <= 0.419
gini = 0.024
samples = 81
value = [80, 1]
x[0] <= -0.599
gini = 0.469
samples = 319
value = [120, 199]
gini = 0.0
samples = 78
value = [78, 0]
x[0] <= 0.508
gini = 0.444
samples = 3
value = [2, 1]
x[1] <= 0.759
gini = 0.071
samples = 54
value = [52, 2]
x[0] <= 0.546
gini = 0.382
samples = 265
value = [68, 197]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 2
value = [2, 0]
gini = 0.0
samples = 52
value = [52, 0]
gini = 0.0
samples = 2
value = [0, 2]
x[1] <= 0.789
gini = 0.466
samples = 179
value = [66, 113]
x[0] <= 0.675
gini = 0.045
samples = 86
value = [2, 84]
x[1] <= -0.089
gini = 0.5
samples = 128
value = [66, 62]
gini = 0.0
samples = 51
value = [0, 51]
x[0] <= 0.667
gini = 0.219
samples = 16
value = [2, 14]
gini = 0.0
samples = 70
value = [0, 70]
x[1] <= -0.525
gini = 0.212
samples = 58
value = [7, 51]
x[1] <= 0.055
gini = 0.265
samples = 70
value = [59, 11]
x[1] <= 0.244
gini = 0.124
samples = 15
value = [1, 14]
gini = 0.0
samples = 1
value = [1, 0]
x[1] <= -0.551
gini = 0.473
samples = 13
value = [5, 8]
x[0] <= 0.528
gini = 0.085
samples = 45
value = [2, 43]
x[1] <= 0.039
gini = 0.494
samples = 9
value = [5, 4]
x[1] <= 0.691
gini = 0.203
samples = 61
value = [54, 7]
gini = 0.0
samples = 8
value = [0, 8]
x[1] <= 0.532
gini = 0.245
samples = 7
value = [1, 6]
x[1] <= -0.655
gini = 0.397
samples = 11
value = [3, 8]
gini = 0.0
samples = 2
value = [2, 0]
x[0] <= -0.562
gini = 0.045
samples = 43
value = [1, 42]
x[1] <= -0.234
gini = 0.5
samples = 2
value = [1, 1]
x[1] <= -0.015
gini = 0.408
samples = 7
value = [5, 2]
gini = 0.0
samples = 2
value = [0, 2]
x[0] <= 0.321
gini = 0.109
samples = 52
value = [49, 3]
x[0] <= -0.254
gini = 0.494
samples = 9
value = [5, 4]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 6
value = [0, 6]
x[0] <= -0.106
gini = 0.5
samples = 6
value = [3, 3]
gini = 0.0
samples = 5
value = [0, 5]
x[0] <= -0.565
gini = 0.444
samples = 3
value = [1, 2]
gini = 0.0
samples = 40
value = [0, 40]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 1
value = [1, 0]
x[1] <= -0.05
gini = 0.5
samples = 4
value = [2, 2]
gini = 0.0
samples = 3
value = [3, 0]
x[1] <= 0.57
gini = 0.043
samples = 45
value = [44, 1]
x[0] <= 0.351
gini = 0.408
samples = 7
value = [5, 2]
gini = 0.0
samples = 2
value = [2, 0]
x[1] <= 0.713
gini = 0.408
samples = 7
value = [5, 2]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 2
value = [0, 2]
gini = 0.0
samples = 2
value = [0, 2]
gini = 0.0
samples = 37
value = [37, 0]
x[1] <= 0.592
gini = 0.219
samples = 8
value = [7, 1]
gini = 0.0
samples = 1
value = [0, 1]
x[0] <= 0.462
gini = 0.278
samples = 6
value = [5, 1]
x[0] <= 0.185
gini = 0.444
samples = 3
value = [1, 2]
gini = 0.0
samples = 4
value = [4, 0]
x[0] <= 0.331
gini = 0.48
samples = 5
value = [2, 3]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 2
value = [0, 2]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 7
value = [7, 0]
x[0] <= 0.416
gini = 0.444
samples = 3
value = [2, 1]
gini = 0.0
samples = 3
value = [3, 0]
gini = 0.0
samples = 2
value = [0, 2]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 2
value = [2, 0]
x[1] <= -0.088
gini = 0.444
samples = 3
value = [2, 1]
gini = 0.0
samples = 2
value = [2, 0]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 2
value = [2, 0]
gini = 0.0
samples = 1
value = [0, 1]



2)

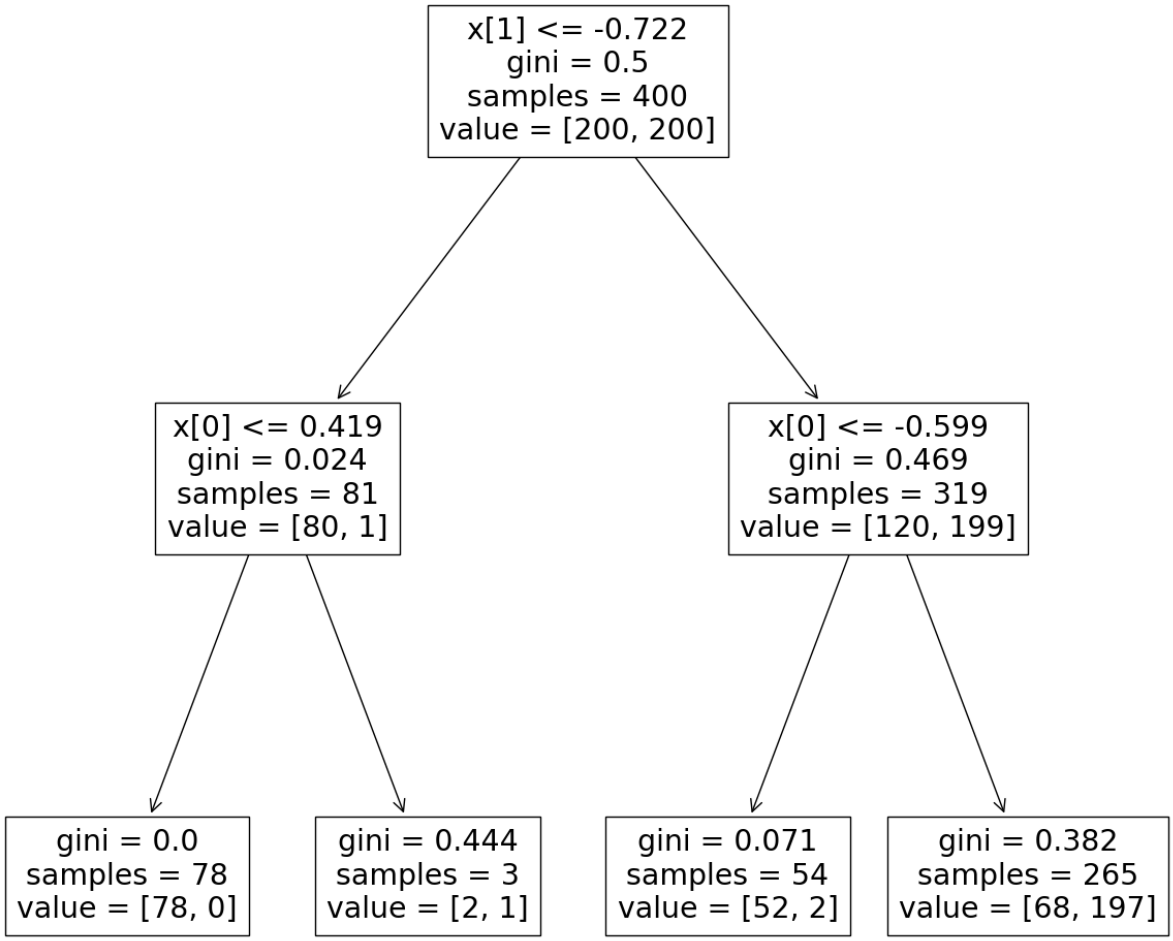
In [ ]:
```
reasonableTree = makeDecisionTree(5)
visualizeDecisionSurface(reasonableTree, f"Decision Tree depth={reasonableTree.get_
```

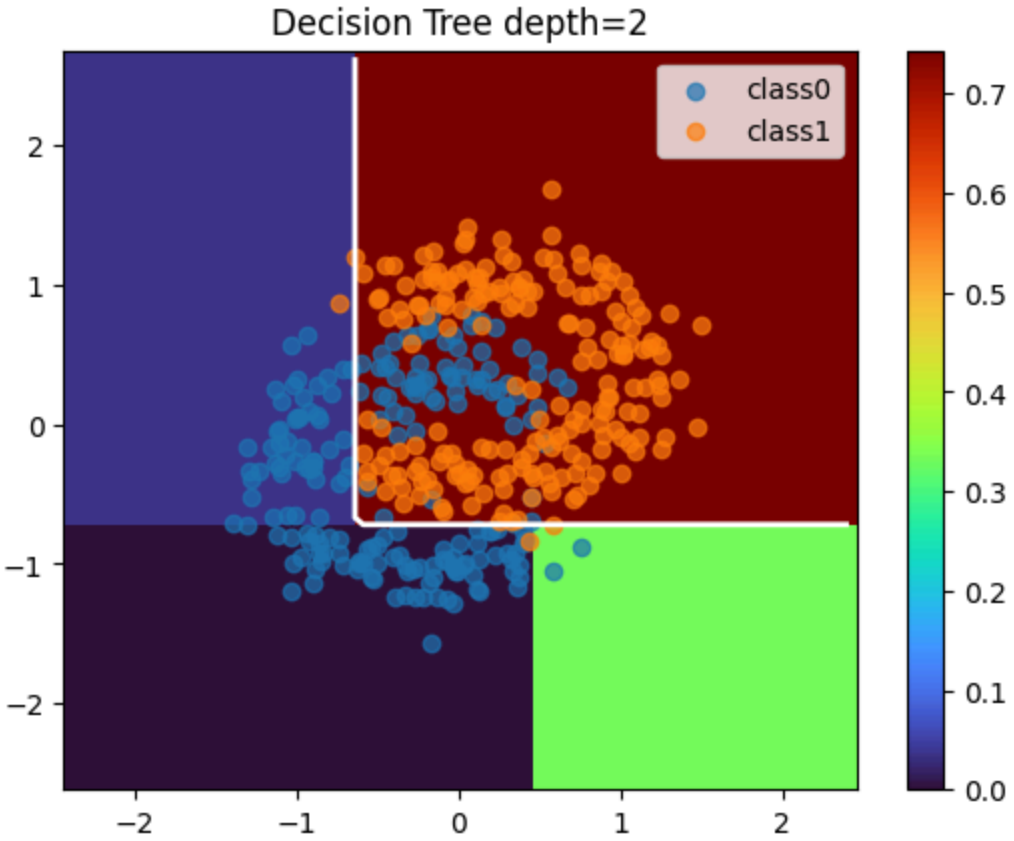

x[1] <= -0.722
gini = 0.5
samples = 400
value = [200, 200]
x[0] <= 0.419
gini = 0.024
samples = 81
value = [80, 1]
x[0] <= -0.599
gini = 0.469
samples = 319
value = [120, 199]
gini = 0.0
samples = 78
value = [78, 0]
x[0] <= 0.508
gini = 0.444
samples = 3
value = [2, 1]
x[1] <= 0.759
gini = 0.071
samples = 54
value = [52, 2]
x[0] <= 0.546
gini = 0.382
samples = 265
value = [68, 197]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 2
value = [2, 0]
gini = 0.0
samples = 52
value = [52, 0]
gini = 0.0
samples = 2
value = [0, 2]
x[1] <= 0.789
gini = 0.466
samples = 179
value = [66, 113]
x[0] <= 0.675
gini = 0.045
samples = 86
value = [2, 84]
x[1] <= -0.089
gini = 0.5
samples = 128
value = [66, 62]
gini = 0.0
samples = 51
value = [0, 51]
x[0] <= 0.667
gini = 0.219
samples = 16
value = [2, 14]
gini = 0.0
samples = 70
value = [0, 70]
gini = 0.212
samples = 58
value = [7, 51]
gini = 0.265
samples = 70
value = [59, 11]
gini = 0.124
samples = 15
value = [1, 14]
gini = 0.0
samples = 1
value = [1, 0]



3)

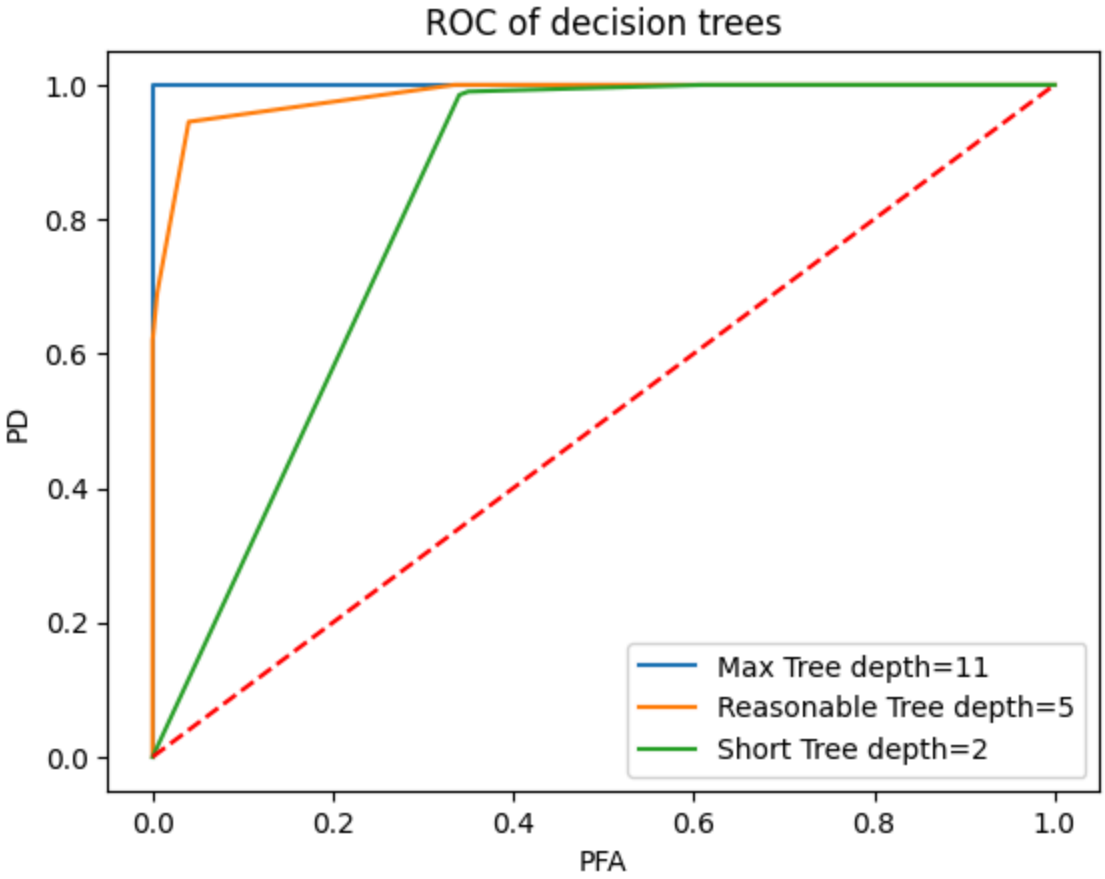
In [ ]:
```
shortTree = makeDecisionTree(2)
visualizeDecisionSurface(shortTree, f"Decision Tree depth={shortTree.get_depth()}")
```


x[1] <= -0.722
gini = 0.5
samples = 400
value = [200, 200]
x[0] <= 0.419
gini = 0.024
samples = 81
value = [80, 1]
x[0] <= -0.599
gini = 0.469
samples = 319
value = [120, 199]
gini = 0.0
samples = 78
value = [78, 0]
gini = 0.444
samples = 3
value = [2, 1]
gini = 0.071
samples = 54
value = [52, 2]
gini = 0.382
samples = 265
value = [68, 197]



4)

In [ ]:
```
maxTreeDecisionStats = maxTree.predict_proba(X)[:,1]
reasonableTreeDecisionStats = reasonableTree.predict_proba(X)[:,1]
shortTreeDecisionStats = shortTree.predict_proba(X)[:,1]

d = {"Max Tree depth=11":maxTreeDecisionStats, "Reasonable Tree depth=5":reasonable

fig, ax = plt.subplots()
for label, decision_stats in d.items():
    pFA, pD, _ = roc_curve(Y, decision_stats)
    ax.plot(pFA, pD, label=label)

ax.plot([0, 1], [0, 1],'r--')
ax.set_ylabel('PD')
ax.set_xlabel('PFA')
ax.set_title('ROC of decision trees')
ax.legend()
plt.show()
```



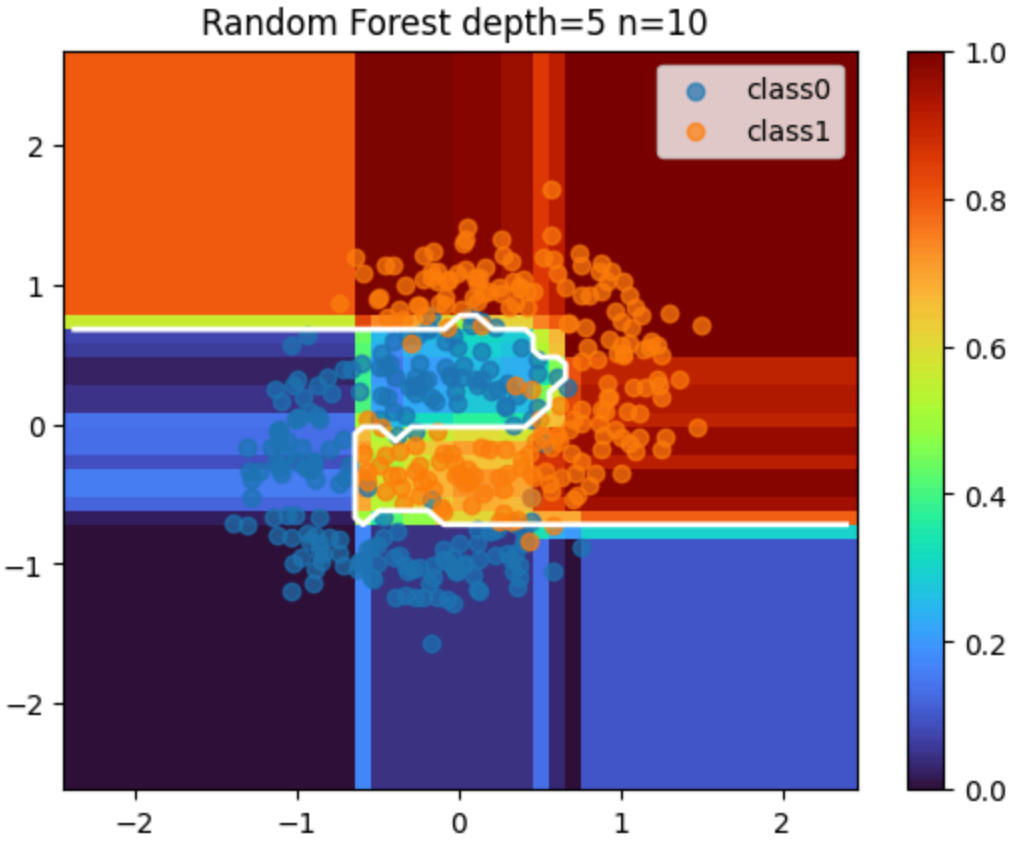
5)

As depth increases, the variance of the model increases and bias decreases. There is more variance because the model is more overfitted to the data and thus will likely perform significantly different when new data is introduced. We can see the effect of increased depth on decreasing bias in the ROC. Since the ROC was generated from testing using the training data, the performance of high depth trees is very good as seen by a perfect ROC for the max tree.

As depth decreases, the bias of the model increases and variance decreases. With lower variance, it's likely that the decision statistic generated by new data will be similar to the decision statistics generated by the training data. With higher bias, there is more systematic errors in the predictions. This is evident in the decision statistic surfaces plots since we can see more blue (class 0) data points in areas that are in the more orange (class 1) decision regions as depth decreases. This signifies more incorrect predictions.

6)

In [ ]:
```
randomForest = RandomForestClassifier(max_depth=5, n_estimators=10, random_state=0)
randomForest.fit(X,Y)
visualizeDecisionSurface(randomForest, f"Random Forest depth=5 n=10 ")
```



7)

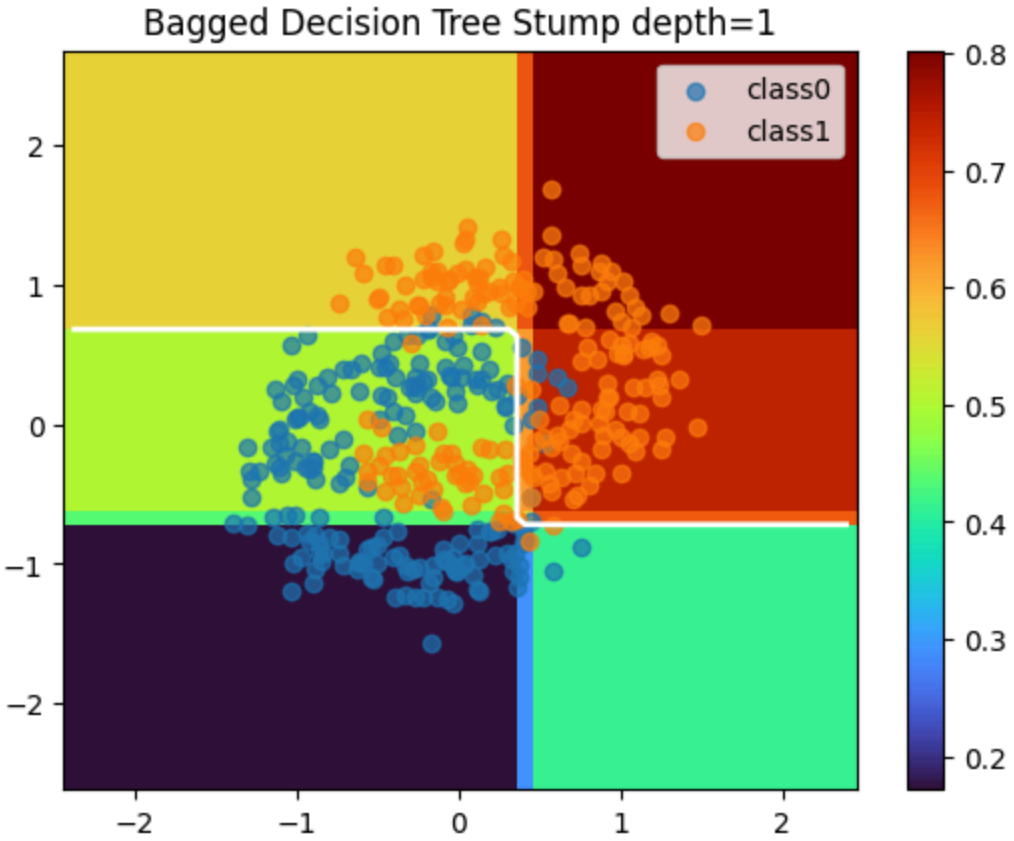
The random forest has more variance and less bias than the decision trees. We can see from the decision statistic surface that there are way more decisions being made (seen by the increase in the variation of colors). Since the decisions are based off of training data, this shows that there is more variance in this model as different data will likely produce a different set of decisions. There is less bias compared to the decision trees and this is seen by the decision bountary separating the classes better. There are less orange dots (class 1) in the region to the left of the decision boundary (shown in white) and there are less blue dots (class 0) in the region to the right of the decision boundary.

Ensemble methods

9)

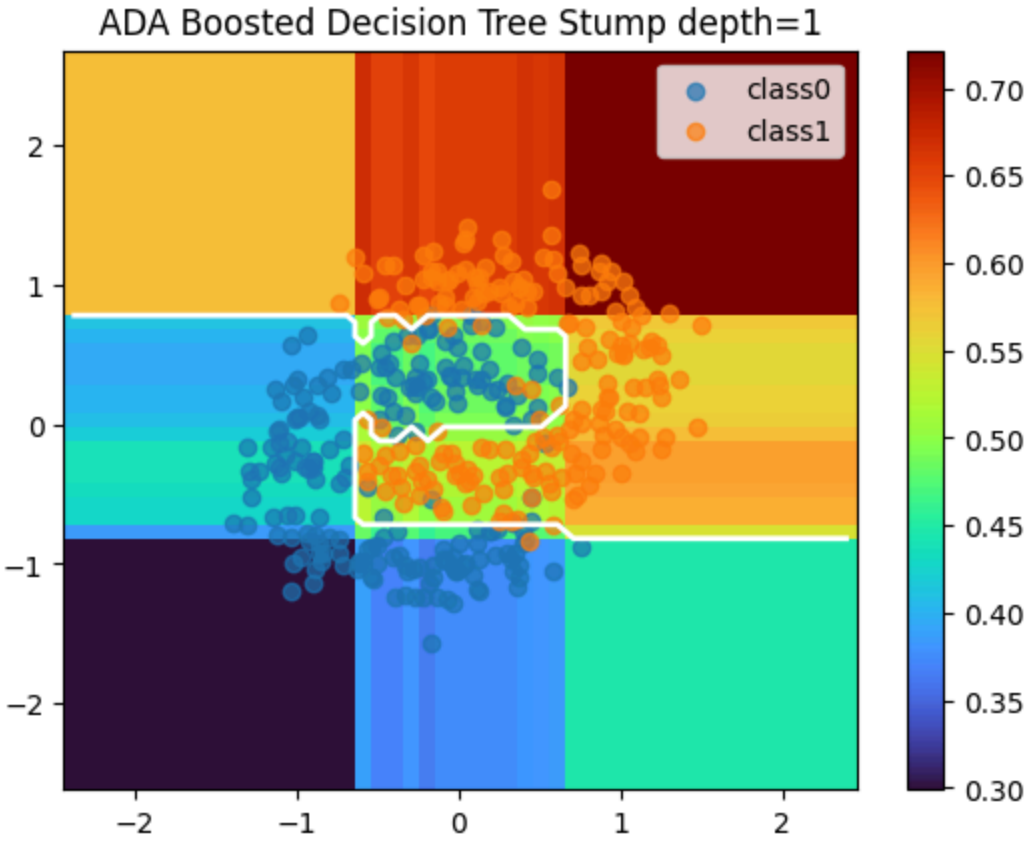
9a)

In [ ]:
```
from sklearn.ensemble import BaggingClassifier
bagged_stump = BaggingClassifier(DecisionTreeClassifier(max_depth=1, random_state=0
bagged_stump.fit(X,Y)
visualizeDecisionSurface(bagged_stump, f"Bagged Decision Tree Stump depth=1")
```



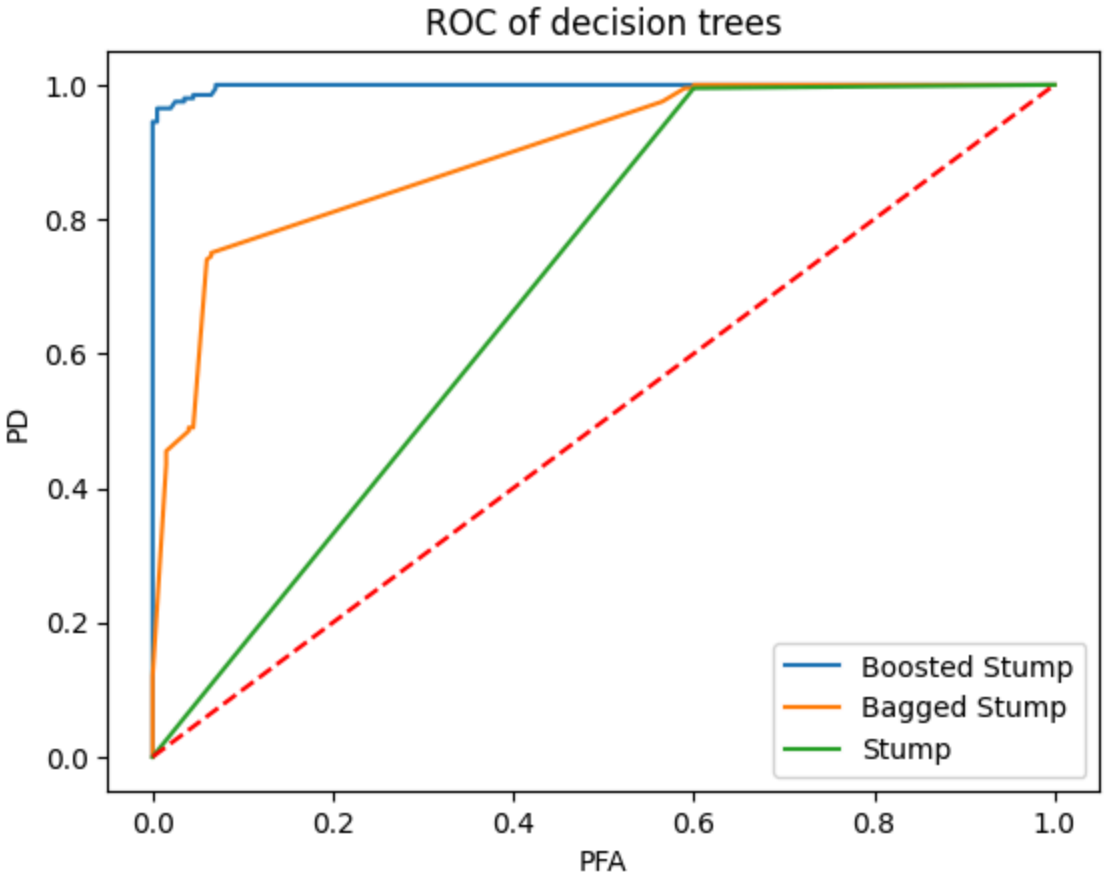
9b)

In [ ]:
```
from sklearn.ensemble import AdaBoostClassifier
boosted_stump = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1, random_state
boosted_stump.fit(X,Y)
visualizeDecisionSurface(boosted_stump, f"ADA Boosted Decision Tree Stump depth=1")
```



10)

In [ ]:
```
stump = DecisionTreeClassifier(max_depth=1, random_state=0)
stump.fit(X,Y)
stump_decisionStats = stump.predict_proba(X)[:,1]
boosted_stump_decisionStats = boosted_stump.predict_proba(X)[:,1]
bagged_stump_decisionStats = bagged_stump.predict_proba(X)[:,1]

d = {"Boosted Stump":boosted_stump_decisionStats, "Bagged Stump":bagged_stump_decis

fig, ax = plt.subplots()
for label, decision_stats in d.items():
    pFA, pD, _ = roc_curve(Y, decision_stats)
    ax.plot(pFA, pD, label=label)

ax.plot([0, 1], [0, 1],'r--')
ax.set_ylabel('PD')
ax.set_xlabel('PFA')
ax.set_title('ROC of decision trees')
ax.legend()
plt.show()
```



11)

Bagging works by creating many variations of the decision tree stump by sampling the data with replacement, and then averaging the decision statistics. This effectively decreases variance as the model is created to fit many different data sets. However, since the different datasets are all sampled from the same training dataset, there is no guarantee that the model will predict brand new data not sampled from the training set well.

Boosting works by iteratively weighting incorrect classifications higher and correct classifications lower. This reduces bias because after each iteration, a new decision region is made based on the newly weighted data points. This results in the model fitting the training data very well as with more and more iterations, more decision regions will be made to classify the training data well.