STAT 154/254 Homework 5

Raymond Tsao

TOTAL POINTS

41 / 41

QUESTION 1

1 5 pts

1.1 1.1 **3 / 3**

✓ **- 0 pts** *Entirely correct*

- 1.5 pts Partial credit for Problem 1.1

- 3 pts No work shown for Problem 1.1

1.2 1.2 **2 / 2**

✓ **- 0 pts** *Entirely correct*

- 1 pts Partial credit for Problem 1.2

- 2 pts No work shown for Problem 1.2

QUESTION 2

2 16 pts

2.1 2.1 **4 / 4**

✓ **- 0 pts** *Entirely correct*

- 2 pts Partial credit for Problem 2.1

- 4 pts No work shown for Problem 2.1

2.2 2.2 **4 / 4**

✓ **- 0 pts** *Entirely correct*

- 2 pts Partial credit for Problem 2.2

- 4 pts No work shown for Problem 2.2

2.3 2.3 **4 / 4**

✓ **- 0 pts** *Entirely correct*

- 2 pts Partial credit for Problem 2.3

- 4 pts No work shown for Problem 2.3

2.4 2.4 **4 / 4**

✓ **- 0 pts** *Entirely correct*

- 2 pts Partial credit for Problem 2.4

- 4 pts No work shown for Problem 2.4

QUESTION 3

3 10 pts

3.1 3.1 **2 / 2**

✓ **- 0 pts** *Entirely correct*

- 1 pts Partial credit for Problem 3.1

- 2 pts No work shown for Problem 3.1

3.2 3.2 **2 / 2**

✓ **- 0 pts** *Entirely correct*

- 1 pts Partial credit for Problem 3.2

- 2 pts No work shown for Problem 3.2

3.3 3.3 **2 / 2**

✓ **- 0 pts** *Entirely correct*

- 1 pts Partial credit for Problem 3.3

- 2 pts No work shown for Problem 3.3

3.4 3.4 **2 / 2**

✓ **- 0 pts** *Entirely correct*

- 0.5 pts Gave accuracy instead of error.

- 1 pts Partial credit given to Problem 3.4

- **2 pts** No work shown for Problem 3.4

3.5 3.5 **2 / 2**

✓ **- 0 pts** *Entirely correct*

- **1 pts** Partial credit for Problem 3.5
- **2 pts** No work shown for Problem 3.5

QUESTION 4

4 10 pts

4.1 4.1 **1 / 1**

✓ **- 0 pts** *Correct*

- **0.5 pts** Partial credit for Problem 4.1
- **1 pts** No work shown for Problem 4.1

4.2 4.2 **3 / 3**

✓ **- 0 pts** *Entirely correct*

- **1.5 pts** Partial credit for Problem 4.2
- **3 pts** No work shown for Problem 4.2

4.3 4.3 **4 / 4**

✓ **- 0 pts** *Entirely correct*

- **2 pts** Partial credit for Problem 4.3
- **4 pts** No work shown for Problem 4.3

4.4 4.4 **2 / 2**

✓ **- 0 pts** *Entirely correct*

- **1 pts** Partial credit for Problem 4.4
- **2 pts** No work shown for Problem 4.4

Problem 1

(1)

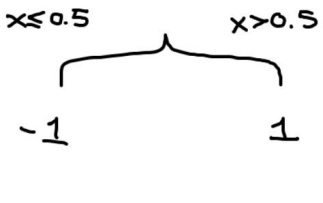
In the original data, there are 3 negatives and 1 positives ($S = [3-, 1+]$). The entropy is

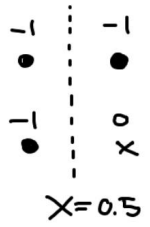
$$H([3-, 1+]) = -\frac{3}{4}\log(\frac{3}{4}) - \frac{1}{4}\log(\frac{1}{4}) = 0.2442$$

Now note that by symmetry, the only possible combinations after the cut is

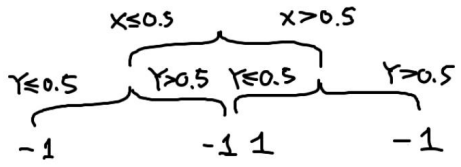
1. Cut 1: $S_{left} = [3-, 1+], S_{right} = [0-, 0+]$
2. Cut 2: $S_{left} = [2-, 0+], S_{right} = [1-, 1+]$

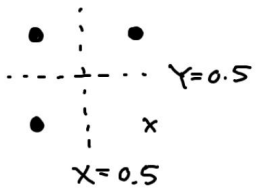
Note that the information gain in cut 1 is 0, while the information gain for cut 2 is positive (0.0937). Therefore, a depth 1 decision tree would look like





To construct a depth 2 tree, note that for each rectangles, if a vertical cut is taken again, the information gain is 0. Thus, consider taking a horizontal cut for each rectangles. This gives a depth 2 tree





Note that the classification tree perfectly classifies the training data.

(2)

We now consider using the Gini index as metric. Like before, computing the Gini index on the original dataset gives us

$$G([3-, 1+]) = 2 \cdot \frac{3}{4} \cdot \frac{1}{4} = 0.375$$

Like before, the information gain of Cut 1 is 0, while the information gain for Cut 2 is 0.125. This means that the depth 1 tree looks like

1.1 1.1 **3 / 3**

✓ **- 0 pts** *Entirely correct*

- 1.5 pts Partial credit for Problem 1.1

- 3 pts No work shown for Problem 1.1

Problem 1

(1)

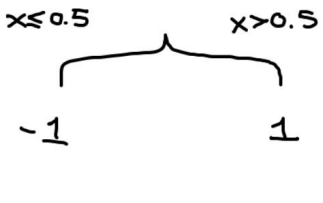
In the original data, there are 3 negatives and 1 positives ($S = [3-, 1+]$). The entropy is

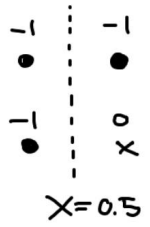
$$H([3-, 1+]) = -\frac{3}{4}\log(\frac{3}{4}) - \frac{1}{4}\log(\frac{1}{4}) = 0.2442$$

Now note that by symmetry, the only possible combinations after the cut is

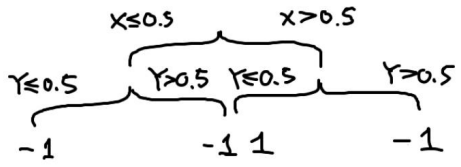
1. Cut 1: $S_{left} = [3-, 1+], S_{right} = [0-, 0+]$
2. Cut 2: $S_{left} = [2-, 0+], S_{right} = [1-, 1+]$

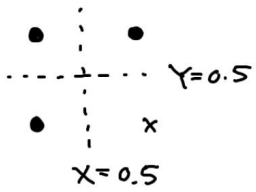
Note that the information gain in cut 1 is 0, while the information gain for cut 2 is positive (0.0937). Therefore, a depth 1 decision tree would look like





To construct a depth 2 tree, note that for each rectangles, if a vertical cut is taken again, the information gain is 0. Thus, consider taking a horizontal cut for each rectangles. This gives a depth 2 tree





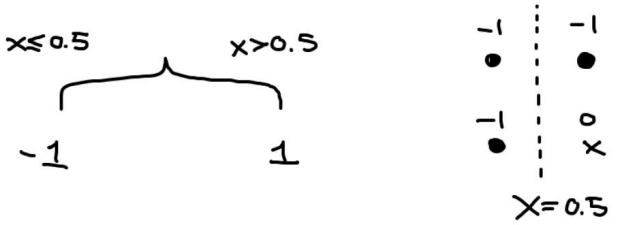
Note that the classification tree perfectly classifies the training data.

(2)

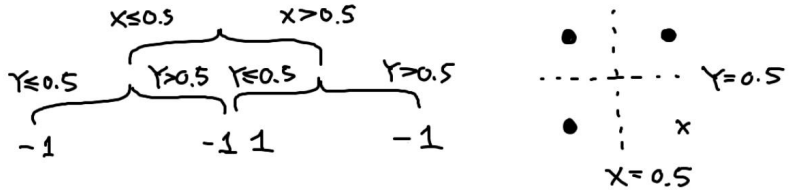
We now consider using the Gini index as metric. Like before, computing the Gini index on the original dataset gives us

$$G([3-, 1+]) = 2 \cdot \frac{3}{4} \cdot \frac{1}{4} = 0.375$$

Like before, the information gain of Cut 1 is 0, while the information gain for Cut 2 is 0.125. This means that the depth 1 tree looks like



Like before, if for each small rectangles, another vertical cut is made, then the information gain is again 0. Taking horizontal cuts therefore gives us



1.2 1.2 **2 / 2**

✓ **- 0 pts** *Entirely correct*

- 1 pts Partial credit for Problem 1.2

- 2 pts No work shown for Problem 1.2

Problem 2

(1)

Consider the first order condition

$$\frac{\partial \tilde{Q}_T}{\partial \alpha_T} = \frac{\partial}{\partial \alpha_T}\Bigg[\sum_{i=1}^{N} \underbrace{\exp\Bigg(-y_i \sum_{t=1}^{T-1} \alpha_t G_t(x_i)\Bigg)}_{w_i} \exp\Bigg(-y_i \alpha_T G_T(x_i)\Bigg)\Bigg] = 0$$

Since the bracketed term is independent of α_T, define it as the weight w_i. Decomposing based on whether $G(x_i) = y_i$ gives

$$\frac{\partial}{\partial \alpha_T}\Bigg[\sum_{i:G_T(x_i)=y_i} w_i \exp(-\alpha_T) + \sum_{i:G_T(x_i)=-y_i} w_i \exp(\alpha_T)\Bigg] = 0$$

Taking derivatives gives

$$-\sum_{i:G_T(x_i)=y_i} w_i \exp(-\alpha_T) + \sum_{i:G_T(x_i)=-y_i} w_i \exp(\alpha_T) = 0$$

Rearranging gives

$$\exp(2\alpha_T) = \frac{\sum_{i:G_T(x_i)=y_i} w_i}{\sum_{i:G_T(x_i)=-y_i} w_i}$$

Which implies that

$$\alpha_T = \frac{1}{2}\ln\left(\frac{\sum_{i:G_T(x_i)=y_i} w_i}{\sum_{i:G_T(x_i)=-y_i} w_i}\right) = \frac{1}{2}\ln\left(\frac{\sum_{i=1}^{N} w_i \mathbb{I}[G(x_i) = y_i]}{\sum_{i=1}^{N} w_i \mathbb{I}[G(x_i) = -y_i]}\right)$$

Normalizing the weights

$$w_i \mapsto \frac{w_i}{\sum_{i=1}^{N} w_i}$$

Then gives the desired result.

(2)

Recall that

$$\begin{aligned}\tilde{Q}_T &= \sum_{i:G_T(x_i)=y_i} w_i \exp(-\alpha_T) + \sum_{i:G_T(x_i)=-y_i} w_i \exp(\alpha_T) \\ &= \sum_{i=1}^{N} w_i \mathbb{I}[G(x_i) = y_i] \exp(-\alpha_T) + \sum_{i=1}^{N} w_i \mathbb{I}[G(x_i) = -y_i] w_i \exp(\alpha_T)\end{aligned}$$

2.1 2.1 **4 / 4**

✓ **- 0 pts** *Entirely correct*

- 2 pts Partial credit for Problem 2.1

- 4 pts No work shown for Problem 2.1

Problem 2

(1)

Consider the first order condition

$$\frac{\partial \tilde{Q}_T}{\partial \alpha_T} = \frac{\partial}{\partial \alpha_T}\Bigg[\sum_{i=1}^{N} \underbrace{\exp\Bigg(-y_i \sum_{t=1}^{T-1} \alpha_t G_t(x_i)\Bigg)}_{w_i} \exp\Bigg(-y_i \alpha_T G_T(x_i)\Bigg)\Bigg] = 0$$

Since the bracketed term is independent of α_T, define it as the weight w_i. Decomposing based on whether $G(x_i) = y_i$ gives

$$\frac{\partial}{\partial \alpha_T}\Bigg[\sum_{i:G_T(x_i)=y_i} w_i \exp(-\alpha_T) + \sum_{i:G_T(x_i)=-y_i} w_i \exp(\alpha_T)\Bigg] = 0$$

Taking derivatives gives

$$-\sum_{i:G_T(x_i)=y_i} w_i \exp(-\alpha_T) + \sum_{i:G_T(x_i)=-y_i} w_i \exp(\alpha_T) = 0$$

Rearranging gives

$$\exp(2\alpha_T) = \frac{\sum_{i:G_T(x_i)=y_i} w_i}{\sum_{i:G_T(x_i)=-y_i} w_i}$$

Which implies that

$$\alpha_T = \frac{1}{2}\ln\left(\frac{\sum_{i:G_T(x_i)=y_i} w_i}{\sum_{i:G_T(x_i)=-y_i} w_i}\right) = \frac{1}{2}\ln\left(\frac{\sum_{i=1}^{N} w_i \mathbb{I}[G(x_i) = y_i]}{\sum_{i=1}^{N} w_i \mathbb{I}[G(x_i) = -y_i]}\right)$$

Normalizing the weights

$$w_i \mapsto \frac{w_i}{\sum_{i=1}^{N} w_i}$$

Then gives the desired result.

(2)

Recall that

$$\begin{aligned}\tilde{Q}_T &= \sum_{i:G_T(x_i)=y_i} w_i \exp(-\alpha_T) + \sum_{i:G_T(x_i)=-y_i} w_i \exp(\alpha_T) \\ &= \sum_{i=1}^{N} w_i \mathbb{I}[G(x_i) = y_i] \exp(-\alpha_T) + \sum_{i=1}^{N} w_i \mathbb{I}[G(x_i) = -y_i] w_i \exp(\alpha_T)\end{aligned}$$

Plugging in the solution for α_T for part (1), and using the fact that

$$\mathcal{P}(G_T, \tilde{W}_N) = \frac{\sum_{i=1}^N w_i \mathbb{I}[G(x_i) = y_i]}{\sum_{i=1}^N w_i}$$

And similarly for $\mathcal{N}(G_T, \tilde{W}_N)$, we have

$$\begin{aligned}
\tilde{Q}_T &= \mathcal{P}(G_T, \tilde{W}_N)\left(\sum_{i=1}^N w_i\right) \exp\left(-\frac{1}{2}\ln\left(\frac{\mathcal{P}(G_T, \tilde{W}_N)}{\mathcal{N}(G_T, \tilde{W}_N)}\right)\right) \\
&\quad + \mathcal{N}(G_T, \tilde{W}_N)\left(\sum_{i=1}^N w_i\right) \exp\left(\frac{1}{2}\ln\left(\frac{\mathcal{P}(G_T, \tilde{W}_N)}{\mathcal{N}(G_T, \tilde{W}_N)}\right)\right) \\
&= 2\sqrt{\mathcal{P}(G_T, \tilde{W}_N)\mathcal{N}(G_T, \tilde{W}_N)}\left(\sum_{i=1}^N w_i\right)
\end{aligned}$$

To further minimize this quantity, we need to minimize the square root term. Since

$$\min 2\sqrt{\mathcal{P}(G_T, \tilde{W}_N)\mathcal{N}(G_T, \tilde{W}_N)} = \max \underbrace{\mathcal{P}(G_T, \tilde{W}_N) + \mathcal{N}(G_T, \tilde{W}_N)}_{1} - 2\sqrt{\mathcal{P}(G_T, \tilde{W}_N)\mathcal{N}(G_T, \tilde{W}_N)}$$

Note that the maximization problem, by completing the square, is equivalent as

$$\max \sqrt{\mathcal{P}(G_T, \tilde{W}_N)} - \sqrt{\mathcal{N}(G_T, \tilde{W}_N)}$$

This suggests that

$$G_T = \text{argmax}_G \sqrt{\mathcal{P}(G, \tilde{W}_N)} - \sqrt{\mathcal{N}(G, \tilde{W}_N)}$$

Thus proving the claim.

(3)

Suppose at each step

$$\sqrt{\mathcal{P}(G, \tilde{W}_N)} - \sqrt{\mathcal{N}(G, \tilde{W}_N)} > \gamma > 0$$

Since

$$(\sqrt{\mathcal{P}(G, \tilde{W}_N)} - \sqrt{\mathcal{N}(G, \tilde{W}_N)})^2 = \underbrace{\mathcal{P}(G_T, \tilde{W}_N) + \mathcal{N}(G_T, \tilde{W}_N)}_{1} - 2\sqrt{\mathcal{P}(G_T, \tilde{W}_N)\mathcal{N}(G_T, \tilde{W}_N)}$$

It follows that

$$\sqrt{\mathcal{P}(G_T, \tilde{W}_N)\mathcal{N}(G_T, \tilde{W}_N)} < 1 - \gamma^2$$

2.2 2.2 **4 / 4**

✓ **- 0 pts** *Entirely correct*

- 2 pts Partial credit for Problem 2.2

- 4 pts No work shown for Problem 2.2

Plugging in the solution for α_T for part (1), and using the fact that

$$\mathcal{P}(G_T, \tilde{W}_N) = \frac{\sum_{i=1}^N w_i \mathbb{I}[G(x_i) = y_i]}{\sum_{i=1}^N w_i}$$

And similarly for $\mathcal{N}(G_T, \tilde{W}_N)$, we have

$$\begin{aligned}
\tilde{Q}_T &= \mathcal{P}(G_T, \tilde{W}_N)\left(\sum_{i=1}^N w_i\right) \exp\left(-\frac{1}{2}\ln\left(\frac{\mathcal{P}(G_T, \tilde{W}_N)}{\mathcal{N}(G_T, \tilde{W}_N)}\right)\right) \\
&\quad + \mathcal{N}(G_T, \tilde{W}_N)\left(\sum_{i=1}^N w_i\right) \exp\left(\frac{1}{2}\ln\left(\frac{\mathcal{P}(G_T, \tilde{W}_N)}{\mathcal{N}(G_T, \tilde{W}_N)}\right)\right) \\
&= 2\sqrt{\mathcal{P}(G_T, \tilde{W}_N)\mathcal{N}(G_T, \tilde{W}_N)}\left(\sum_{i=1}^N w_i\right)
\end{aligned}$$

To further minimize this quantity, we need to minimize the square root term. Since

$$\min 2\sqrt{\mathcal{P}(G_T, \tilde{W}_N)\mathcal{N}(G_T, \tilde{W}_N)} = \max \underbrace{\mathcal{P}(G_T, \tilde{W}_N) + \mathcal{N}(G_T, \tilde{W}_N)}_{1} - 2\sqrt{\mathcal{P}(G_T, \tilde{W}_N)\mathcal{N}(G_T, \tilde{W}_N)}$$

Note that the maximization problem, by completing the square, is equivalent as

$$\max \sqrt{\mathcal{P}(G_T, \tilde{W}_N)} - \sqrt{\mathcal{N}(G_T, \tilde{W}_N)}$$

This suggests that

$$G_T = \text{argmax}_G \sqrt{\mathcal{P}(G, \tilde{W}_N)} - \sqrt{\mathcal{N}(G, \tilde{W}_N)}$$

Thus proving the claim.

(3)

Suppose at each step

$$\sqrt{\mathcal{P}(G, \tilde{W}_N)} - \sqrt{\mathcal{N}(G, \tilde{W}_N)} > \gamma > 0$$

Since

$$(\sqrt{\mathcal{P}(G, \tilde{W}_N)} - \sqrt{\mathcal{N}(G, \tilde{W}_N)})^2 = \underbrace{\mathcal{P}(G_T, \tilde{W}_N) + \mathcal{N}(G_T, \tilde{W}_N)}_{1} - 2\sqrt{\mathcal{P}(G_T, \tilde{W}_N)\mathcal{N}(G_T, \tilde{W}_N)}$$

It follows that

$$\sqrt{\mathcal{P}(G_T, \tilde{W}_N)\mathcal{N}(G_T, \tilde{W}_N)} < 1 - \gamma^2$$

To prove the claim, consider estimating the ratio

$$\begin{aligned}
\frac{\tilde{Q}_{T+1}}{\tilde{Q}_T} &= \frac{\sum_{i=1}^N w_i \exp(-y_i \alpha_{T+1} G_{T+1}(x_i))}{\sum_{i=1}^N w_i} \\
&= \sum_{i=1}^N u_i \exp(-y_i \alpha_{T+1} G_{T+1}(x_i)) \\
&= \sum_{i=1}^N u_i \mathbb{I}[G(x_i) = y_i] \exp(-\alpha_{T+1}) + \sum_{i=1}^N u_i \mathbb{I}[G(x_i) = -y_i] \exp(\alpha_{T+1}) \\
&= \mathcal{P}(G_{T+1}, \tilde{W}_N) \exp\left(-\frac{1}{2} \ln\left(\frac{\mathcal{P}(G_{T+1}, \tilde{W}_N)}{\mathcal{N}(G_{T+1}, \tilde{W}_N)} \right) \right) \\
&+ \mathcal{N}(G_{T+1}, \tilde{W}_N) \exp\left(\frac{1}{2} \ln\left(\frac{\mathcal{P}(G_{T+1}, \tilde{W}_N)}{\mathcal{N}(G_{T+1}, \tilde{W}_N)} \right) \right) \\
&= 2\sqrt{\mathcal{P}(G_{T+1}, \tilde{W}_N)\mathcal{N}(G_{T+1}, \tilde{W}_N)} \\
&< 1 - \gamma^2
\end{aligned}$$

By an inductive argument, we have

$$\frac{\tilde{Q}_{T+1}}{\tilde{Q}_1} < (1 - \gamma^2)^T$$

This proves the claim.

(4)

Note that we can write the number of errors can be expressed as

$$Q_{T+1} = \sum_{i=1}^N \mathbb{I}[y_i \sum_{t=1}^{T+1} \alpha_t G_t(x_i) < 0]$$

Since $Q_{T+1} \leq \tilde{Q}_{T+1}$ for all T, and that from part (3),

$$\tilde{Q}_{T+1} < (1 - \gamma^2)^T \tilde{Q}_1$$

If we can can show that $\gamma < 1$, then the claim follows since as $T \to \infty$,

$$Q_{T+1} \leq (1 - \gamma^2)^T \tilde{Q}_1 \to 0$$

Since Q_{T+1} takes on integer values, this proves that $Q_{T+1} = 0$ when T is large enough (when the RHS is strictly between 0 and 1).

We now justify that $0 < \gamma < 1$. This is because

$$\mathcal{P} + \mathcal{N} = 1 \implies (\sqrt{\mathcal{P}} - \sqrt{\mathcal{N}})^2 < 1 \implies \sqrt{\mathcal{P}} - \sqrt{\mathcal{N}} < 1$$

Which in turn imply that $\gamma < 1$. This completes the argument.

2.3 2.3 **4 / 4**

✓ **- 0 pts** *Entirely correct*

- 2 pts Partial credit for Problem 2.3

- 4 pts No work shown for Problem 2.3

To prove the claim, consider estimating the ratio

$$\begin{aligned}
\frac{\tilde{Q}_{T+1}}{\tilde{Q}_T} &= \frac{\sum_{i=1}^N w_i \exp(-y_i \alpha_{T+1} G_{T+1}(x_i))}{\sum_{i=1}^N w_i} \\
&= \sum_{i=1}^N u_i \exp(-y_i \alpha_{T+1} G_{T+1}(x_i)) \\
&= \sum_{i=1}^N u_i \mathbb{I}[G(x_i) = y_i] \exp(-\alpha_{T+1}) + \sum_{i=1}^N u_i \mathbb{I}[G(x_i) = -y_i] \exp(\alpha_{T+1}) \\
&= \mathcal{P}(G_{T+1}, \tilde{W}_N) \exp\left(-\frac{1}{2} \ln\left(\frac{\mathcal{P}(G_{T+1}, \tilde{W}_N)}{\mathcal{N}(G_{T+1}, \tilde{W}_N)} \right) \right) \\
&+ \mathcal{N}(G_{T+1}, \tilde{W}_N) \exp\left(\frac{1}{2} \ln\left(\frac{\mathcal{P}(G_{T+1}, \tilde{W}_N)}{\mathcal{N}(G_{T+1}, \tilde{W}_N)} \right) \right) \\
&= 2\sqrt{\mathcal{P}(G_{T+1}, \tilde{W}_N)\mathcal{N}(G_{T+1}, \tilde{W}_N)} \\
&< 1 - \gamma^2
\end{aligned}$$

By an inductive argument, we have

$$\frac{\tilde{Q}_{T+1}}{\tilde{Q}_1} < (1 - \gamma^2)^T$$

This proves the claim.

(4)

Note that we can write the number of errors can be expressed as

$$Q_{T+1} = \sum_{i=1}^N \mathbb{I}[y_i \sum_{t=1}^{T+1} \alpha_t G_t(x_i) < 0]$$

Since $Q_{T+1} \leq \tilde{Q}_{T+1}$ for all T, and that from part (3),

$$\tilde{Q}_{T+1} < (1 - \gamma^2)^T \tilde{Q}_1$$

If we can can show that $\gamma < 1$, then the claim follows since as $T \to \infty$,

$$Q_{T+1} \leq (1 - \gamma^2)^T \tilde{Q}_1 \to 0$$

Since Q_{T+1} takes on integer values, this proves that $Q_{T+1} = 0$ when T is large enough (when the RHS is strictly between 0 and 1).

We now justify that $0 < \gamma < 1$. This is because

$$\mathcal{P} + \mathcal{N} = 1 \implies (\sqrt{\mathcal{P}} - \sqrt{\mathcal{N}})^2 < 1 \implies \sqrt{\mathcal{P}} - \sqrt{\mathcal{N}} < 1$$

Which in turn imply that $\gamma < 1$. This completes the argument.

2.4 2.4 **4 / 4**

✓ **- 0 pts** *Entirely correct*

- 2 pts Partial credit for Problem 2.4

- 4 pts No work shown for Problem 2.4

In [20]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

Problem 3

(a)

In [21]:
```
path = 'iris.csv'
iris = pd.read_csv(path, header=None)
```

In [22]:
```
iris
```

Out[22]:

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [29]:
```
X = iris.iloc[:, :-1]
y = iris.iloc[:, -1]
```

In [30]:
```
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
```

In [31]:
```
X
```

Out[31]:

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...

145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [32]:
```
y
```

Out[32]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [33]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0
```

In [40]:
```
print("Training set shape:", X_train.shape, y_train.shape)
print("Test set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (112, 4) (112,)
Test set shape: (38, 4) (38,)
```

From the indices for the X_train, we see that the split is indeed random (not consecutive)

In [43]:
```
X_train
```

Out[43]:

	0	1	2	3
61	5.9	3.0	4.2	1.5
92	5.8	2.6	4.0	1.2
112	6.8	3.0	5.5	2.1
2	4.7	3.2	1.3	0.2
141	6.9	3.1	5.1	2.3
...	...	...	...	...
9	4.9	3.1	1.5	0.1
103	6.3	2.9	5.6	1.8
67	5.8	2.7	4.1	1.0
117	7.7	3.8	6.7	2.2
47	4.6	3.2	1.4	0.2

112 rows × 4 columns

(b)

In [44]:
```
from sklearn.decomposition import PCA
```

In [45]:
```
pca = PCA(n_components=2)
```

3.1 3.1 **2 / 2**

✓ **- 0 pts** *Entirely correct*

- 1 pts Partial credit for Problem 3.1

- 2 pts No work shown for Problem 3.1

145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [32]:
```
y
```

Out[32]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [33]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0
```

In [40]:
```
print("Training set shape:", X_train.shape, y_train.shape)
print("Test set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (112, 4) (112,)
Test set shape: (38, 4) (38,)
```

From the indices for the X_train, we see that the split is indeed random (not consecutive)

In [43]:
```
X_train
```

Out[43]:

	0	1	2	3
61	5.9	3.0	4.2	1.5
92	5.8	2.6	4.0	1.2
112	6.8	3.0	5.5	2.1
2	4.7	3.2	1.3	0.2
141	6.9	3.1	5.1	2.3
...	...	...	...	...
9	4.9	3.1	1.5	0.1
103	6.3	2.9	5.6	1.8
67	5.8	2.7	4.1	1.0
117	7.7	3.8	6.7	2.2
47	4.6	3.2	1.4	0.2

112 rows × 4 columns

(b)

In [44]:
```
from sklearn.decomposition import PCA
```

In [45]:
```
pca = PCA(n_components=2)
```

```
X_train_pca = pca.fit_transform(X_train)
```

In [47]:
```
print(f"PCA decomposition shape: {X_train_pca.shape}")
```

```
PCA decomposition shape: (112, 2)
```

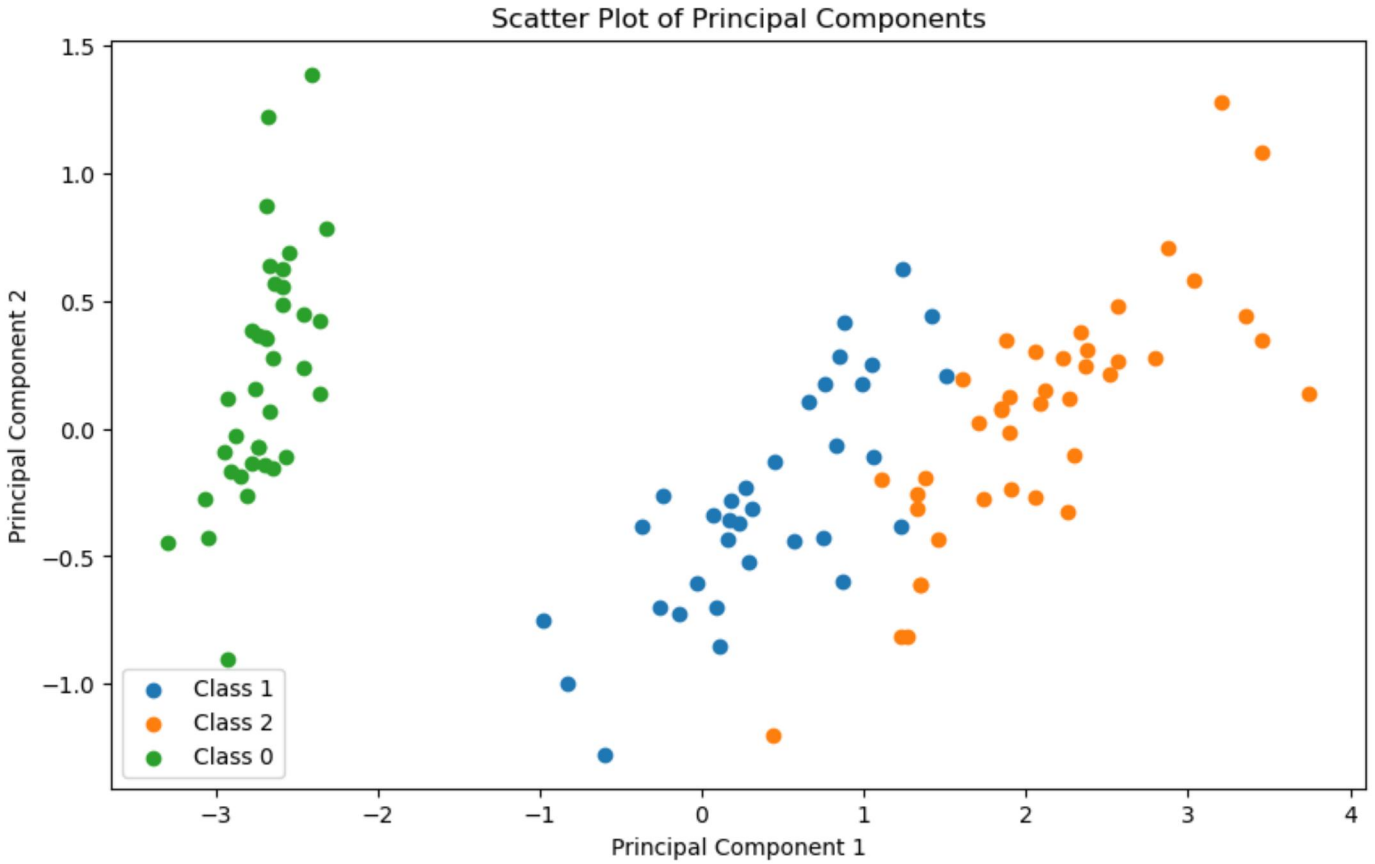
In [50]:
```
df_pca = pd.DataFrame(data=X_train_pca, columns=['Principal Component 1', 'Principal Com
df_pca['Class'] = y_train
df_pca
```

Out[50]:

	Principal Component 1	Principal Component 2	Class
0	0.451878	-0.126531	1
1	0.162624	-0.431250	1
2	2.113020	0.149036	2
3	-2.949155	-0.089098	0
4	1.875502	0.345670	2
...	...	...	...
107	-2.732974	-0.069181	0
108	1.908646	-0.234124	2
109	0.169159	-0.358948	1
110	3.456095	1.085525	2
111	-2.901679	-0.169985	0

112 rows × 3 columns

In [51]:
```
plt.figure(figsize=(10, 6))
classes = df_pca['Class'].unique()
for class_label in classes:
    plt.scatter(df_pca.loc[df_pca['Class'] == class_label, 'Principal Component 1'],
                df_pca.loc[df_pca['Class'] == class_label, 'Principal Component 2'],
                label=f'Class {class_label}')

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Scatter Plot of Principal Components')
plt.legend()
plt.show()
```



(c)

In [56]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [54]:
```
# Trained a decision tree classifier
classifier = DecisionTreeClassifier(random_state=0)
classifier.fit(X_train_pca, y_train)
```

Out[54]:
```
DecisionTreeClassifier(random_state=0)
```

In [55]:
```
# Evaluated the classfier on the test data set
X_test_pca = pca.transform(X_test)
y_pred = classifier.predict(X_test_pca)
```

Below we plot the decision boundary of the trained model

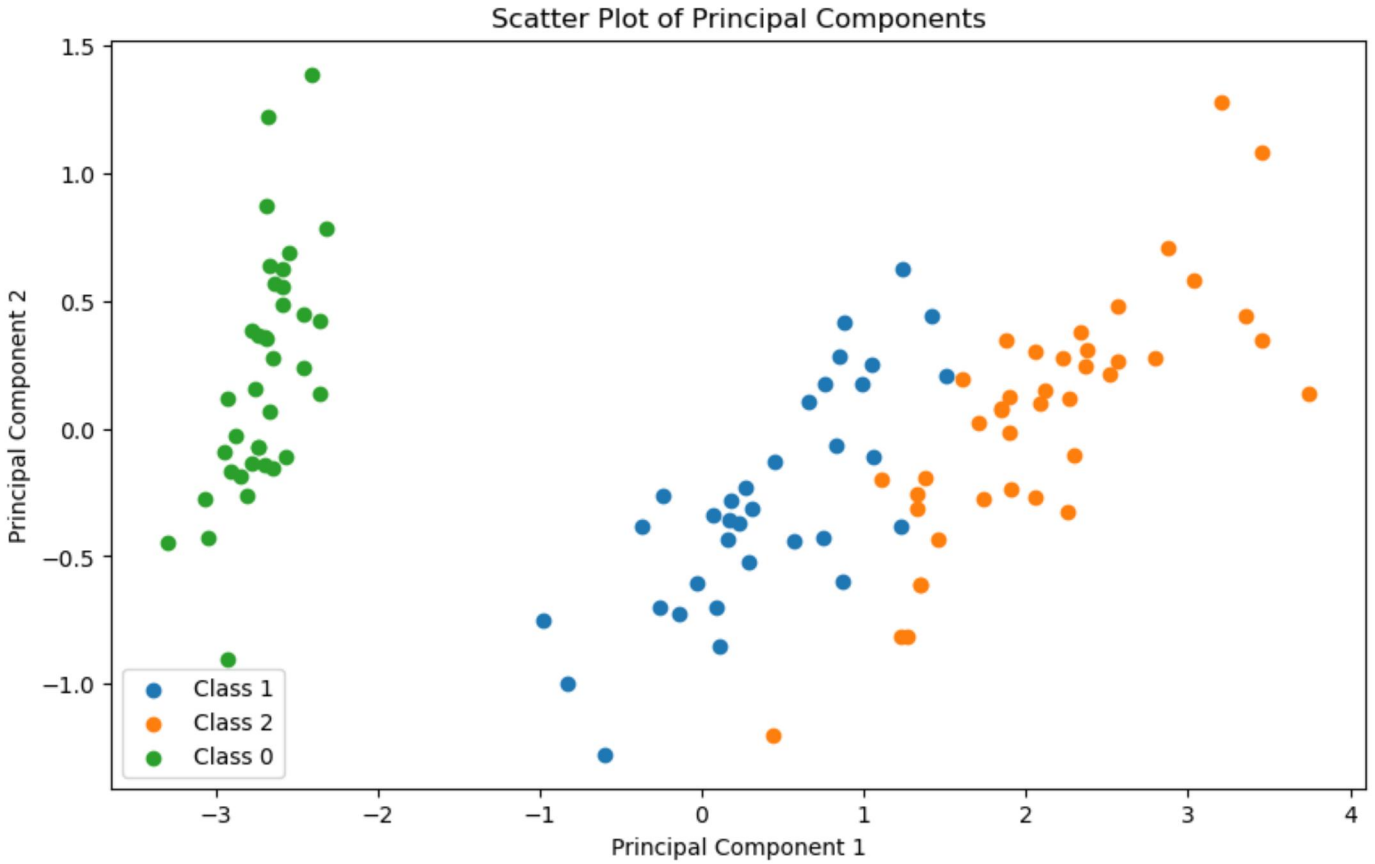
In [67]:
```
# Plot decision boundaries
plt.figure(figsize=(10, 6))
for class_label in classes:
    plt.scatter(df_pca.loc[df_pca['Class'] == class_label, 'Principal Component 1'],
                df_pca.loc[df_pca['Class'] == class_label, 'Principal Component 2'],
                label=f'Class {class_label}')

h = .02
x_min, x_max = X_train_pca[:, 0].min() - 1, X_train_pca[:, 0].max() + 1
y_min, y_max = X_train_pca[:, 1].min() - 1, X_train_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.2)
```

3.2 3.2 **2 / 2**

✓ **- 0 pts** *Entirely correct*

- 1 pts Partial credit for Problem 3.2

- 2 pts No work shown for Problem 3.2



(c)

In [56]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [54]:
```
# Trained a decision tree classifier
classifier = DecisionTreeClassifier(random_state=0)
classifier.fit(X_train_pca, y_train)
```

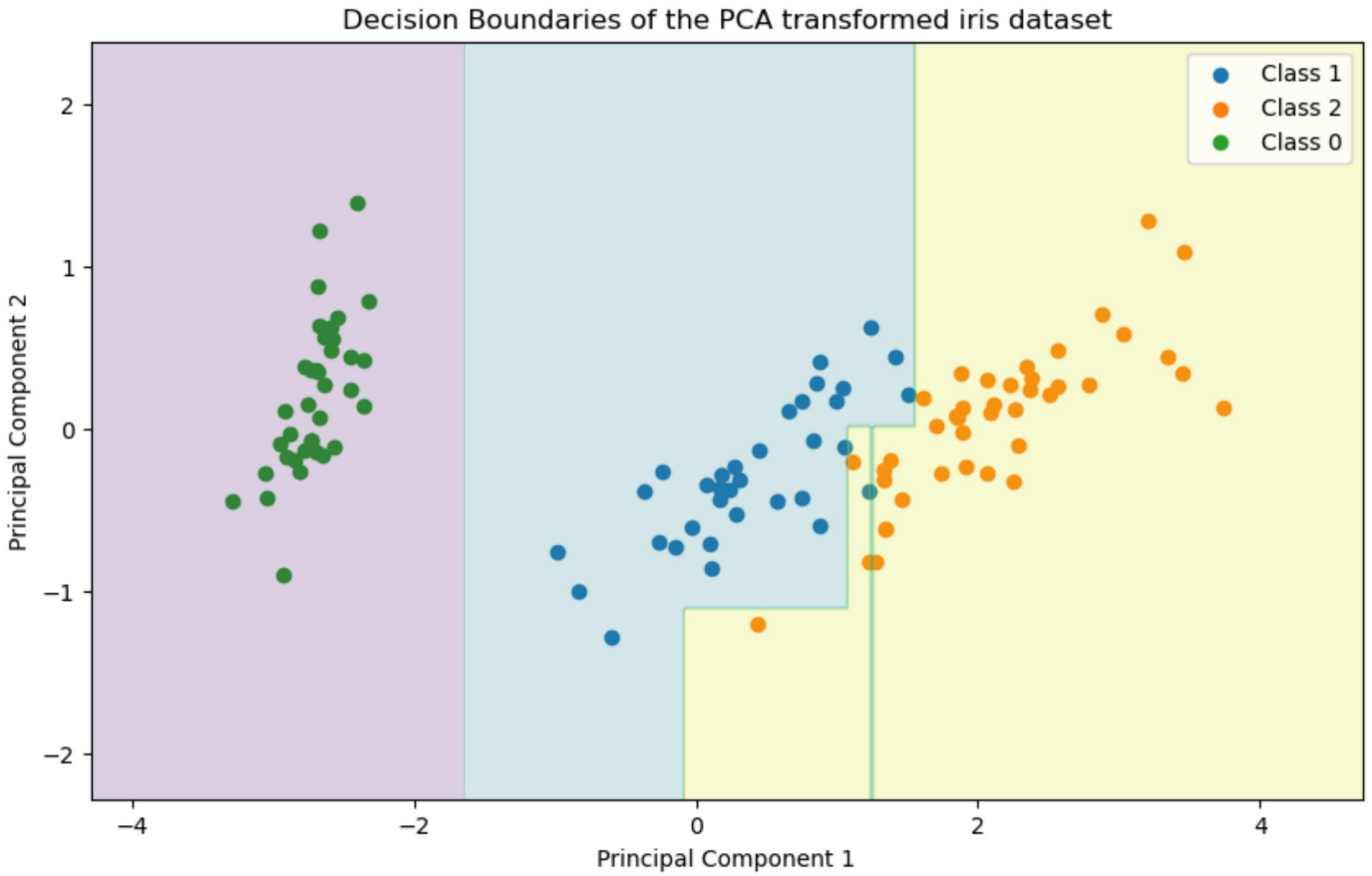
Out[54]:
```
DecisionTreeClassifier(random_state=0)
```

In [55]:
```
# Evaluated the classfier on the test data set
X_test_pca = pca.transform(X_test)
y_pred = classifier.predict(X_test_pca)
```

Below we plot the decision boundary of the trained model

In [67]:
```
# Plot decision boundaries
plt.figure(figsize=(10, 6))
for class_label in classes:
    plt.scatter(df_pca.loc[df_pca['Class'] == class_label, 'Principal Component 1'],
                df_pca.loc[df_pca['Class'] == class_label, 'Principal Component 2'],
                label=f'Class {class_label}')

h = .02
x_min, x_max = X_train_pca[:, 0].min() - 1, X_train_pca[:, 0].max() + 1
y_min, y_max = X_train_pca[:, 1].min() - 1, X_train_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.2)
```

```
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Decision Boundaries of the PCA transformed iris dataset')
plt.legend()
plt.show()
```



(d)

In [57]:
```
from sklearn.metrics import accuracy_score
```

In [59]:
```
# Evaluated the accuracy score using the accuracy_score metric in sklearn
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the Decision Tree classifier: {accuracy:.2f}')
```

```
Accuracy of the Decision Tree classifier: 0.95
```

Below we see that our prediction pretty much matches with the ground truth

In [64]:
```
print(f"Prediction: {y_pred}")
print(f"Ground truth: {y_test}")
```

```
Prediction: [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 1 2 1 0
 2]
Ground truth: [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 1]
```

(e)

In [73]:
```
alphas = np.linspace(0.0, 1, 100)
```

In [74]:
```
alpha_values = []
test_errors = []
```

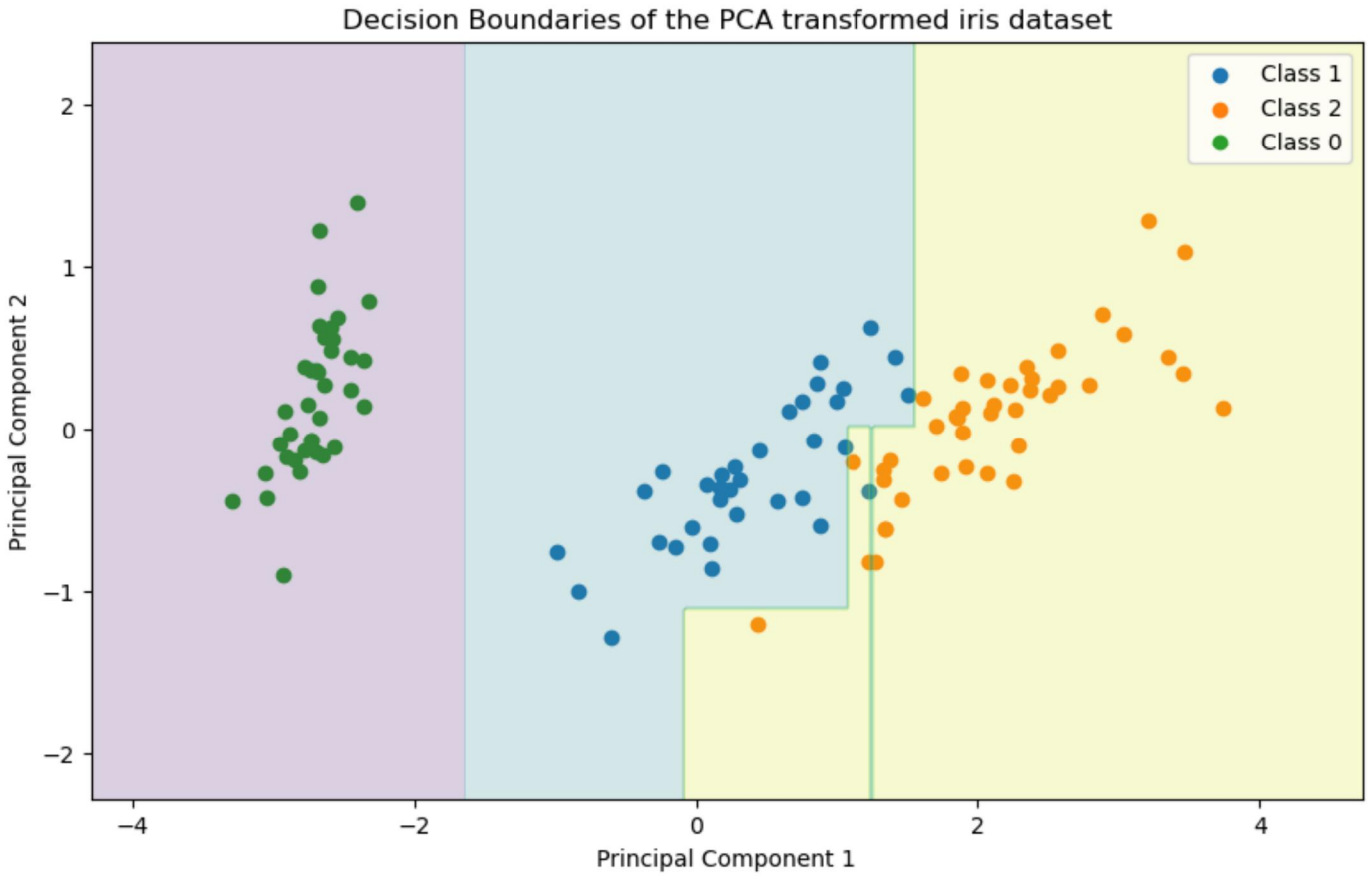
3.3 3.3 **2 / 2**

✓ **- 0 pts** *Entirely correct*

- 1 pts Partial credit for Problem 3.3

- 2 pts No work shown for Problem 3.3

```
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Decision Boundaries of the PCA transformed iris dataset')
plt.legend()
plt.show()
```



(d)

In [57]:
```
from sklearn.metrics import accuracy_score
```

In [59]:
```
# Evaluated the accuracy score using the accuracy_score metric in sklearn
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the Decision Tree classifier: {accuracy:.2f}')
```

```
Accuracy of the Decision Tree classifier: 0.95
```

Below we see that our prediction pretty much matches with the ground truth

In [64]:
```
print(f"Prediction: {y_pred}")
print(f"Ground truth: {y_test}")
```

```
Prediction: [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 1 2 1 0
 2]
Ground truth: [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 1]
```

(e)

In [73]:
```
alphas = np.linspace(0.0, 1, 100)
```

In [74]:
```
alpha_values = []
test_errors = []
```

3.4 3.4 **2 / 2**

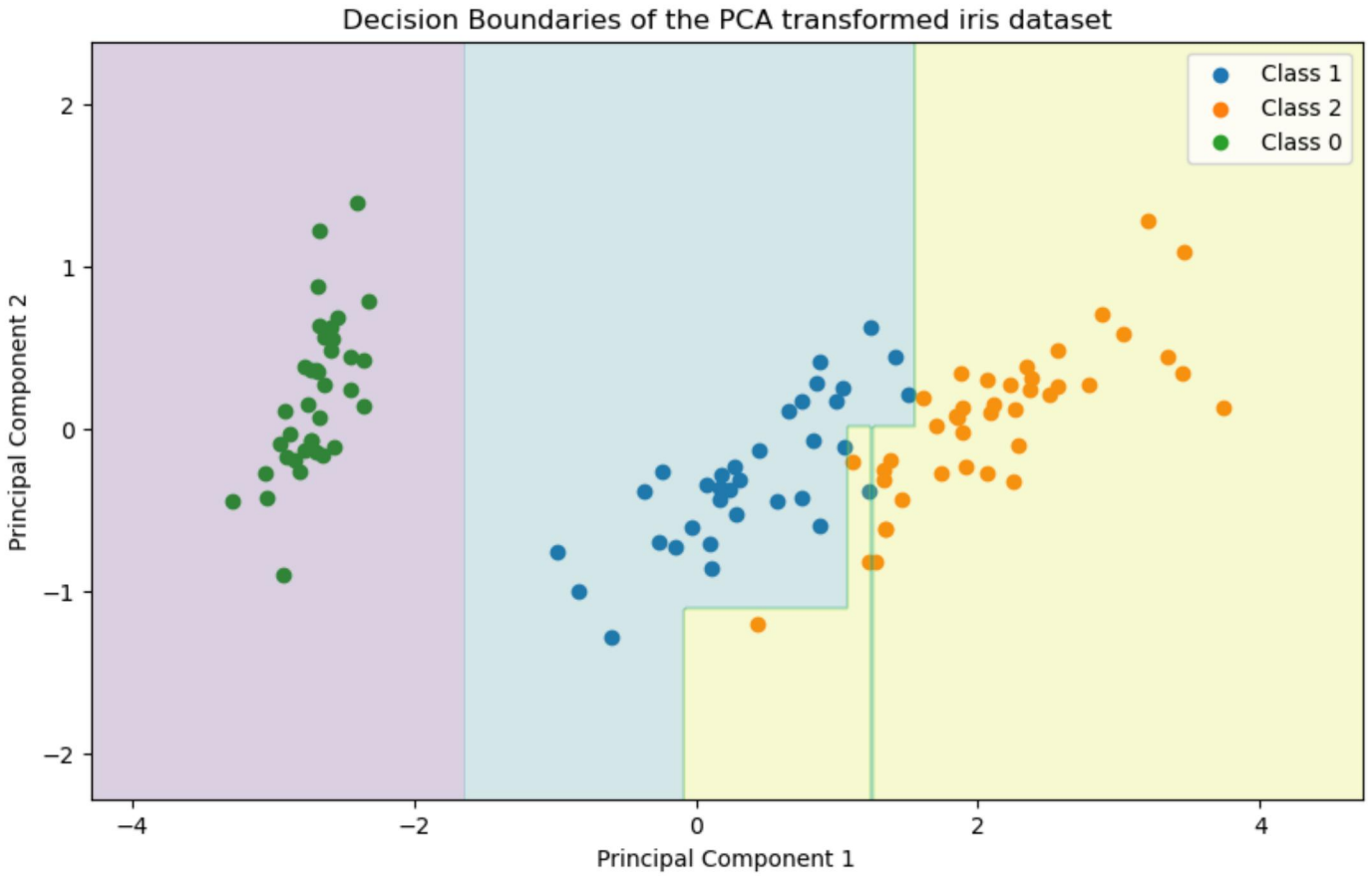
✓ **- 0 pts** *Entirely correct*

- 0.5 pts Gave accuracy instead of error.

- 1 pts Partial credit given to Problem 3.4

- 2 pts No work shown for Problem 3.4

```
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Decision Boundaries of the PCA transformed iris dataset')
plt.legend()
plt.show()
```



(d)

In [57]:
```
from sklearn.metrics import accuracy_score
```

In [59]:
```
# Evaluated the accuracy score using the accuracy_score metric in sklearn
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the Decision Tree classifier: {accuracy:.2f}')
```

```
Accuracy of the Decision Tree classifier: 0.95
```

Below we see that our prediction pretty much matches with the ground truth

In [64]:
```
print(f"Prediction: {y_pred}")
print(f"Ground truth: {y_test}")
```

```
Prediction: [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 1 2 1 0
 2]
Ground truth: [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 1]
```

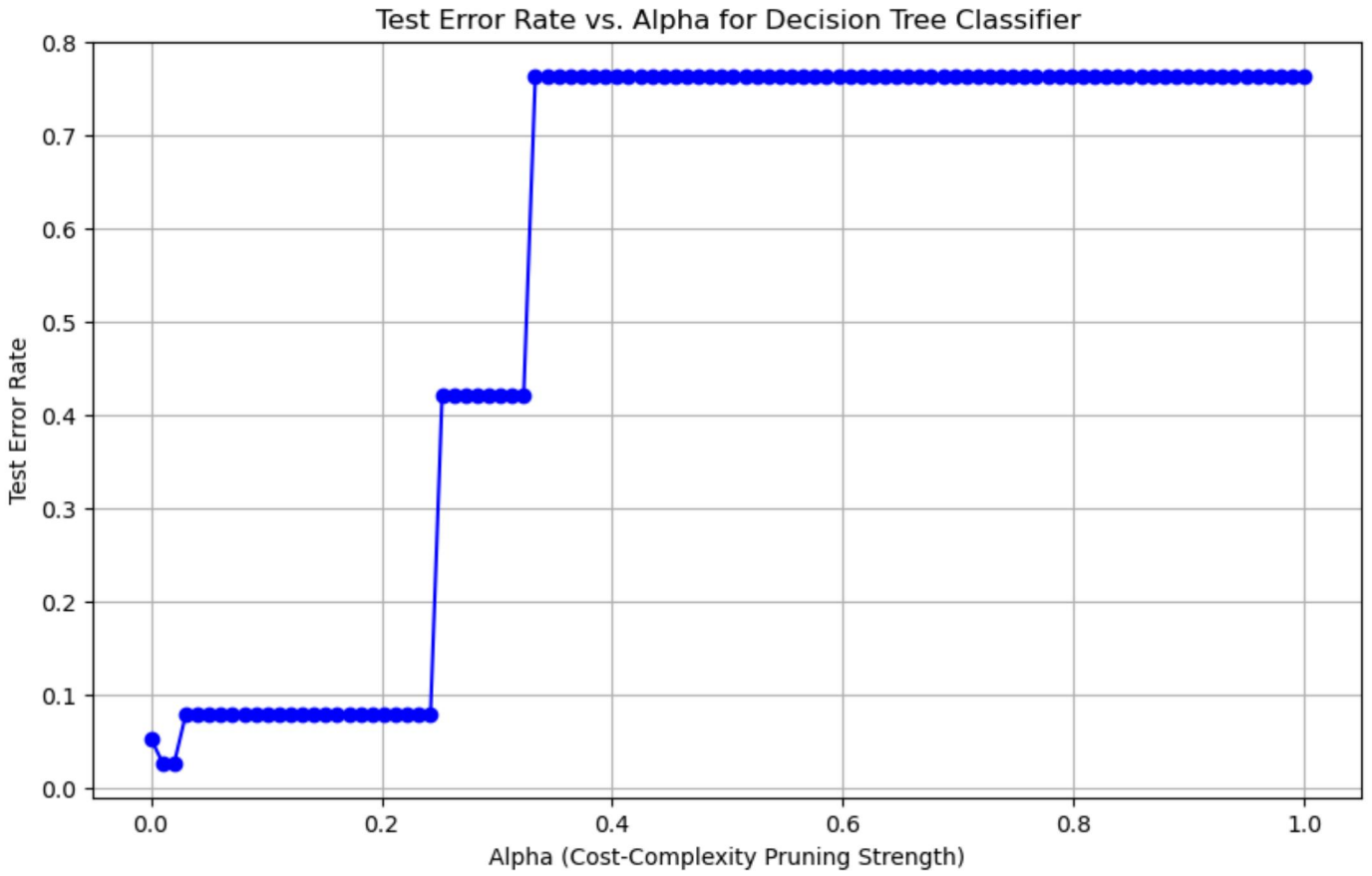
(e)

In [73]:
```
alphas = np.linspace(0.0, 1, 100)
```

In [74]:
```
alpha_values = []
test_errors = []
```

```
for alpha in alphas:
    classifier = DecisionTreeClassifier(random_state=0, ccp_alpha=alpha)
    classifier.fit(X_train_pca, y_train)
    X_test_pca = pca.transform(X_test)
    y_pred = classifier.predict(X_test_pca)
    test_error = 1 - accuracy_score(y_test, y_pred)
    alpha_values.append(alpha)
    test_errors.append(test_error)
```

In [75]:
```
plt.figure(figsize=(10, 6))
plt.plot(alpha_values, test_errors, marker='o', linestyle='-', color='b')
plt.xlabel('Alpha (Cost-Complexity Pruning Strength)')
plt.ylabel('Test Error Rate')
plt.title('Test Error Rate vs. Alpha for Decision Tree Classifier')
plt.grid(True)
plt.show()
```



From the above graph, we see that alpha between 0.01 and 0.02 minimizes the test error rate

Problem 4

(a)

In [93]:
```
path = 'iris.csv'
iris = pd.read_csv(path, header=None)
iris.columns = ["sepal_length", "sepal_width", "petal_length", "petal_width", "class"]

iris_binary = iris[iris['class'] != 'Iris-versicolor']
iris_binary
```

Out[93]:

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa

3.5 3.5 **2 / 2**

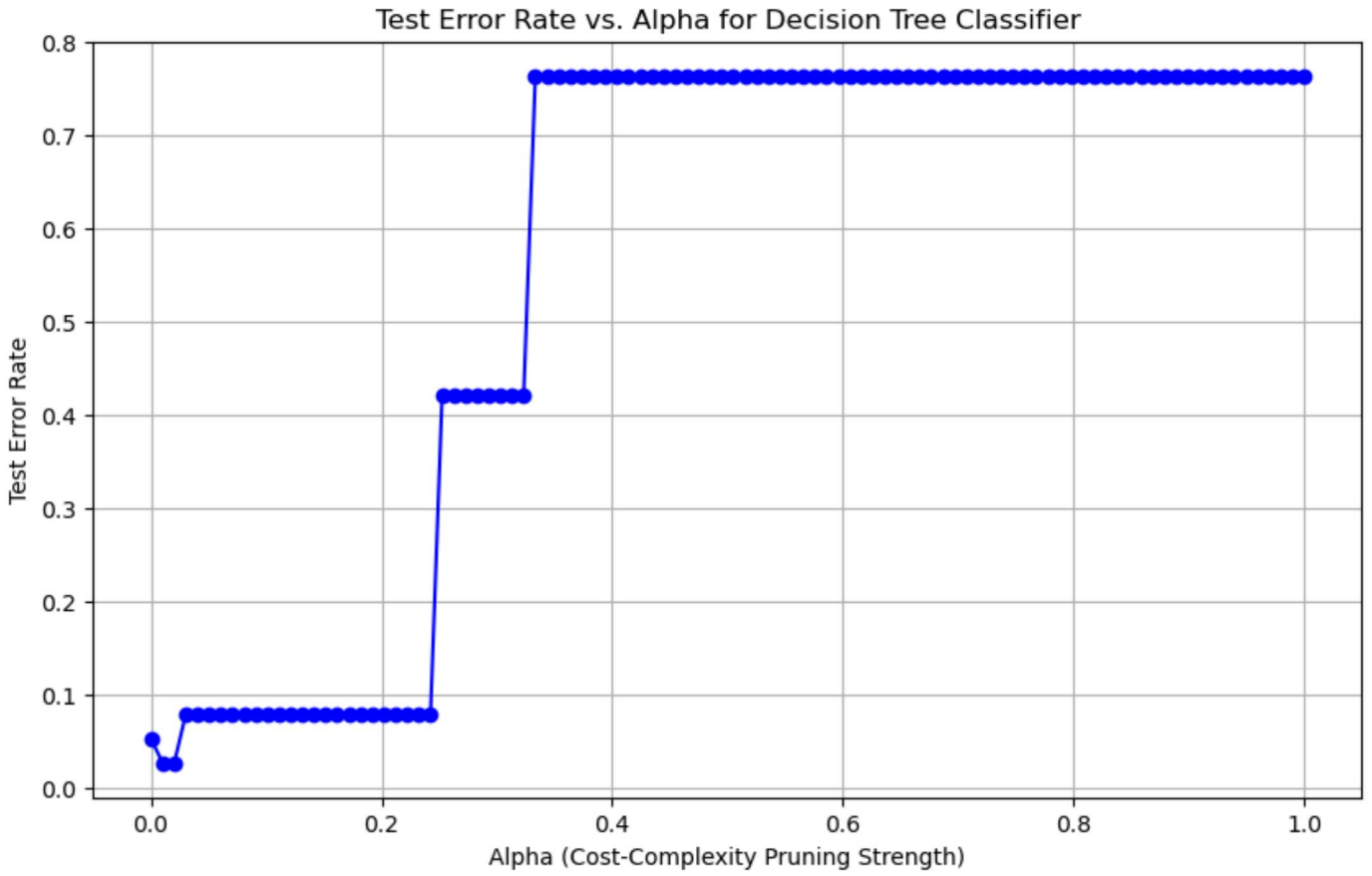
✓ **- 0 pts** *Entirely correct*

- 1 pts Partial credit for Problem 3.5

- 2 pts No work shown for Problem 3.5

```
for alpha in alphas:
    classifier = DecisionTreeClassifier(random_state=0, ccp_alpha=alpha)
    classifier.fit(X_train_pca, y_train)
    X_test_pca = pca.transform(X_test)
    y_pred = classifier.predict(X_test_pca)
    test_error = 1 - accuracy_score(y_test, y_pred)
    alpha_values.append(alpha)
    test_errors.append(test_error)
```

In [75]:
```
plt.figure(figsize=(10, 6))
plt.plot(alpha_values, test_errors, marker='o', linestyle='-', color='b')
plt.xlabel('Alpha (Cost-Complexity Pruning Strength)')
plt.ylabel('Test Error Rate')
plt.title('Test Error Rate vs. Alpha for Decision Tree Classifier')
plt.grid(True)
plt.show()
```



From the above graph, we see that alpha between 0.01 and 0.02 minimizes the test error rate

Problem 4

(a)

In [93]:
```
path = 'iris.csv'
iris = pd.read_csv(path, header=None)
iris.columns = ["sepal_length", "sepal_width", "petal_length", "petal_width", "class"]

iris_binary = iris[iris['class'] != 'Iris-versicolor']
iris_binary
```

Out[93]:

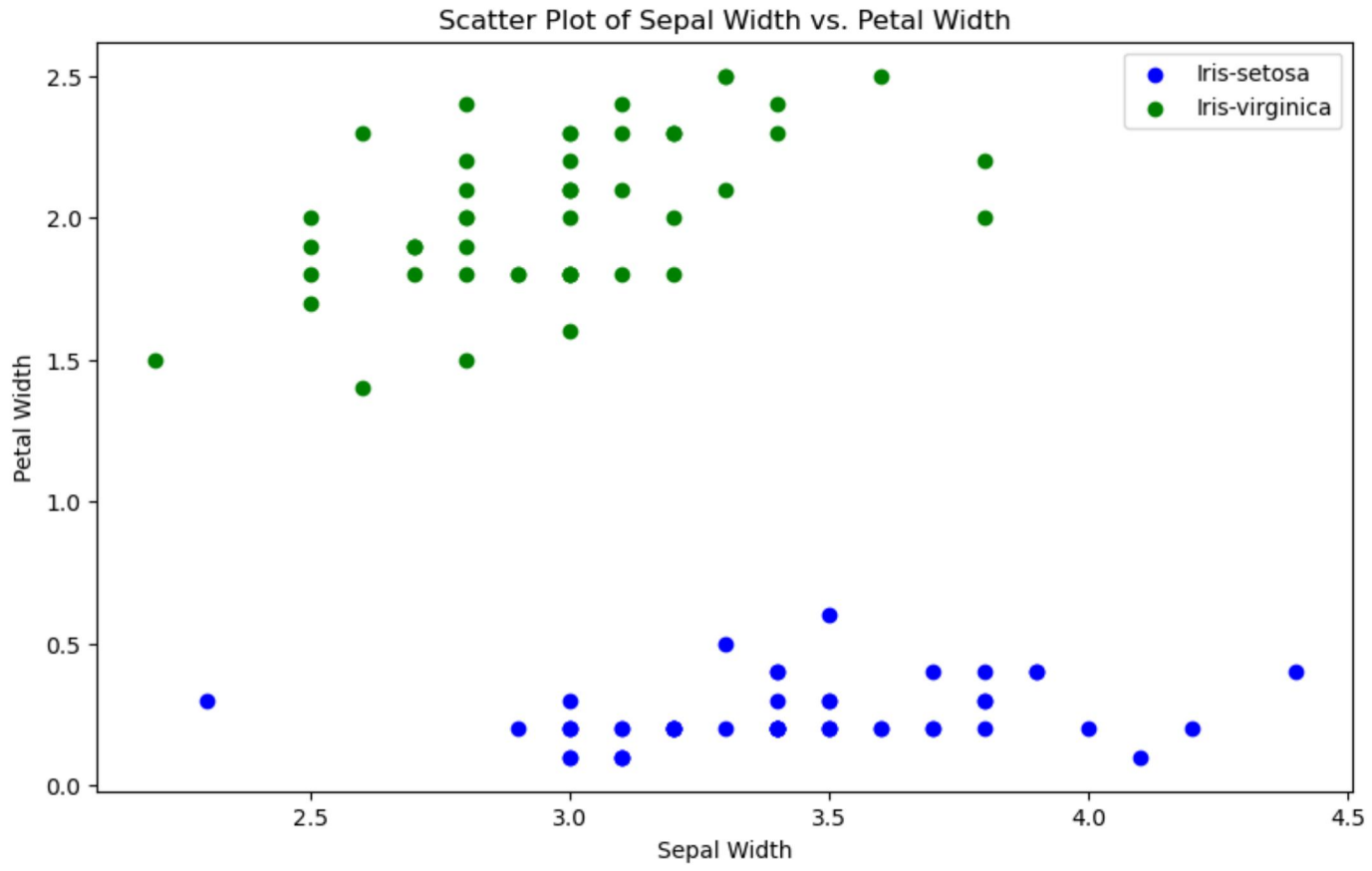
	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa

1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

100 rows × 5 columns

(b)

In [94]:
```
plt.figure(figsize=(10, 6))
colors = {'Iris-setosa': 'blue', 'Iris-virginica': 'green'}

for class_label, color in colors.items():
    class_data = iris_binary[iris_binary['class'] == class_label]
    plt.scatter(class_data['sepal_width'], class_data['petal_width'], label=f'{class_lab

plt.xlabel('Sepal Width')
plt.ylabel('Petal Width')
plt.title('Scatter Plot of Sepal Width vs. Petal Width')
plt.legend()
plt.show()
```



4.1 4.1 **1 / 1**

✓ **- 0 pts** *Correct*

- 0.5 pts Partial credit for Problem 4.1

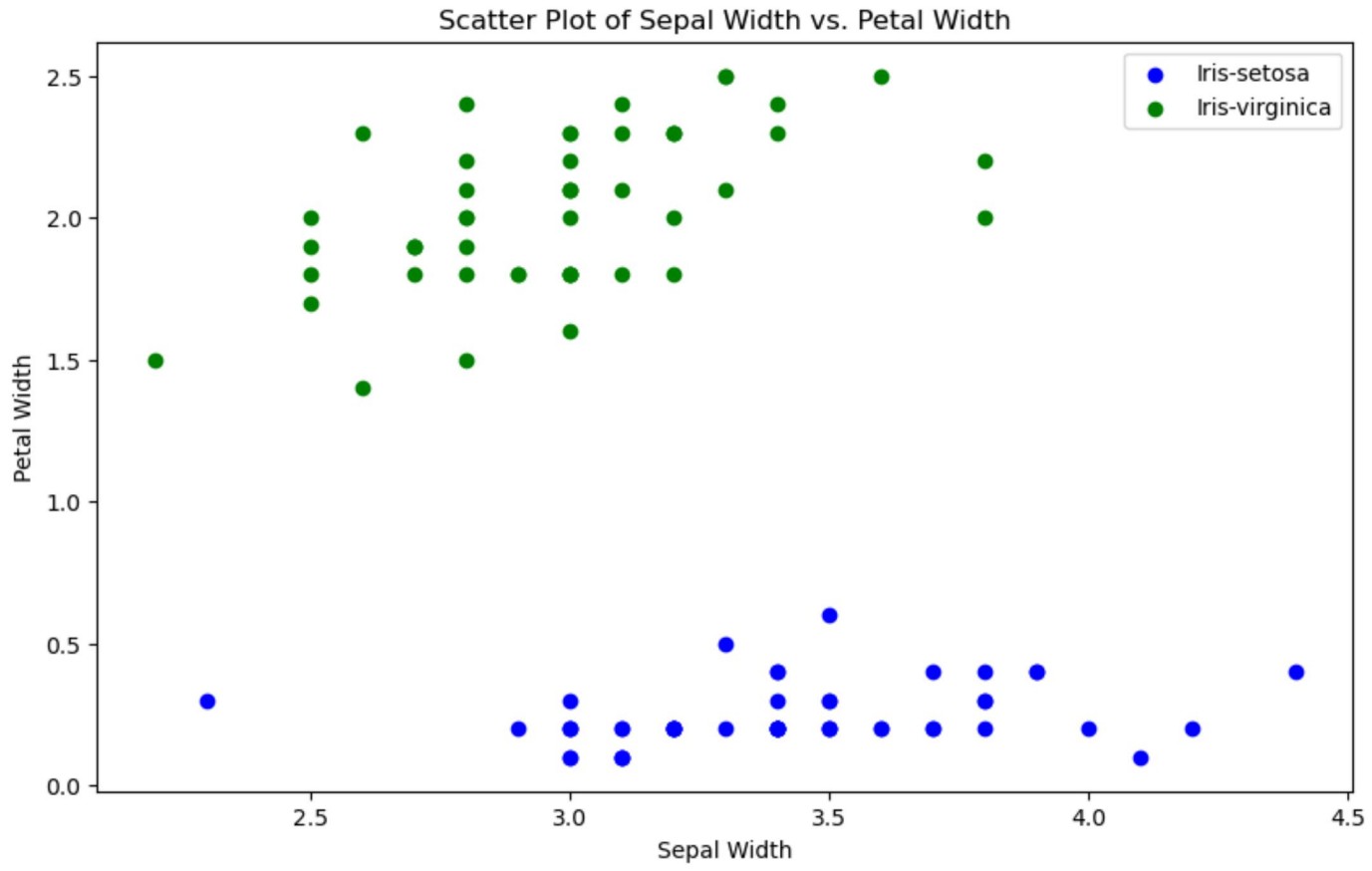
- 1 pts No work shown for Problem 4.1

1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

100 rows × 5 columns

(b)

In [94]:
```
plt.figure(figsize=(10, 6))
colors = {'Iris-setosa': 'blue', 'Iris-virginica': 'green'}

for class_label, color in colors.items():
    class_data = iris_binary[iris_binary['class'] == class_label]
    plt.scatter(class_data['sepal_width'], class_data['petal_width'], label=f'{class_lab

plt.xlabel('Sepal Width')
plt.ylabel('Petal Width')
plt.title('Scatter Plot of Sepal Width vs. Petal Width')
plt.legend()
plt.show()
```



(c)

In [120…
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
```

In [121…
```
X = iris_binary[['sepal_width', 'petal_width']]
y = iris_binary.iloc[:, -1]
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
```

In [122…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0
print("Training set shape:", X_train.shape, y_train.shape)
print("Test set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (75, 2) (75,)
Test set shape: (25, 2) (25,)
```

In [127…
```
n_estimators_values = [1, 2, 3]

n_estimator_result = []

for i, n_estimators in enumerate(n_estimators_values, 1):
    base_estimator = SVC(kernel='linear', probability=True, random_state=0)
    classifier_adaboost = AdaBoostClassifier(estimator=base_estimator, n_estimators=n_es

    classifier_adaboost.fit(X_train, y_train)

    y_pred_adaboost = classifier_adaboost.predict(X_test)

    n_estimator_result.append(y_pred_adaboost)
```

In [128…
```
plt.scatter(X_test['sepal_width'], X_test['petal_width'], c=n_estimator_result[0], cmap=
plt.title(f'AdaBoost (n_estimators={1})\nAccuracy: {accuracy_score(y_test, y_pred_adaboo
plt.xlabel('Sepal Width')
plt.ylabel('Petal Width')
plt.show()
```

4.2 4.2 **3 / 3**

✓ **- 0 pts** *Entirely correct*

- 1.5 pts Partial credit for Problem 4.2

- 3 pts No work shown for Problem 4.2

(c)

In [120…
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
```

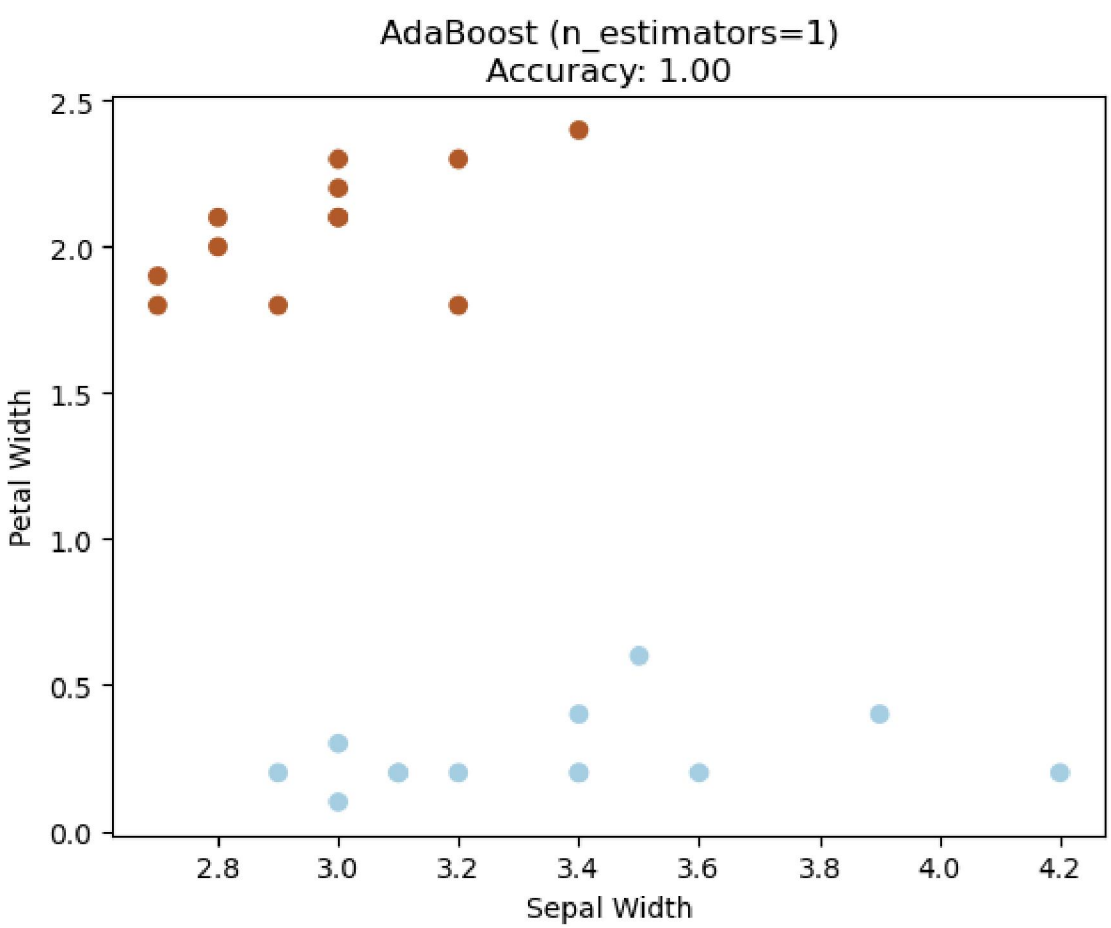
In [121…
```
X = iris_binary[['sepal_width', 'petal_width']]
y = iris_binary.iloc[:, -1]
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
```

In [122…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0
print("Training set shape:", X_train.shape, y_train.shape)
print("Test set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (75, 2) (75,)
Test set shape: (25, 2) (25,)
```

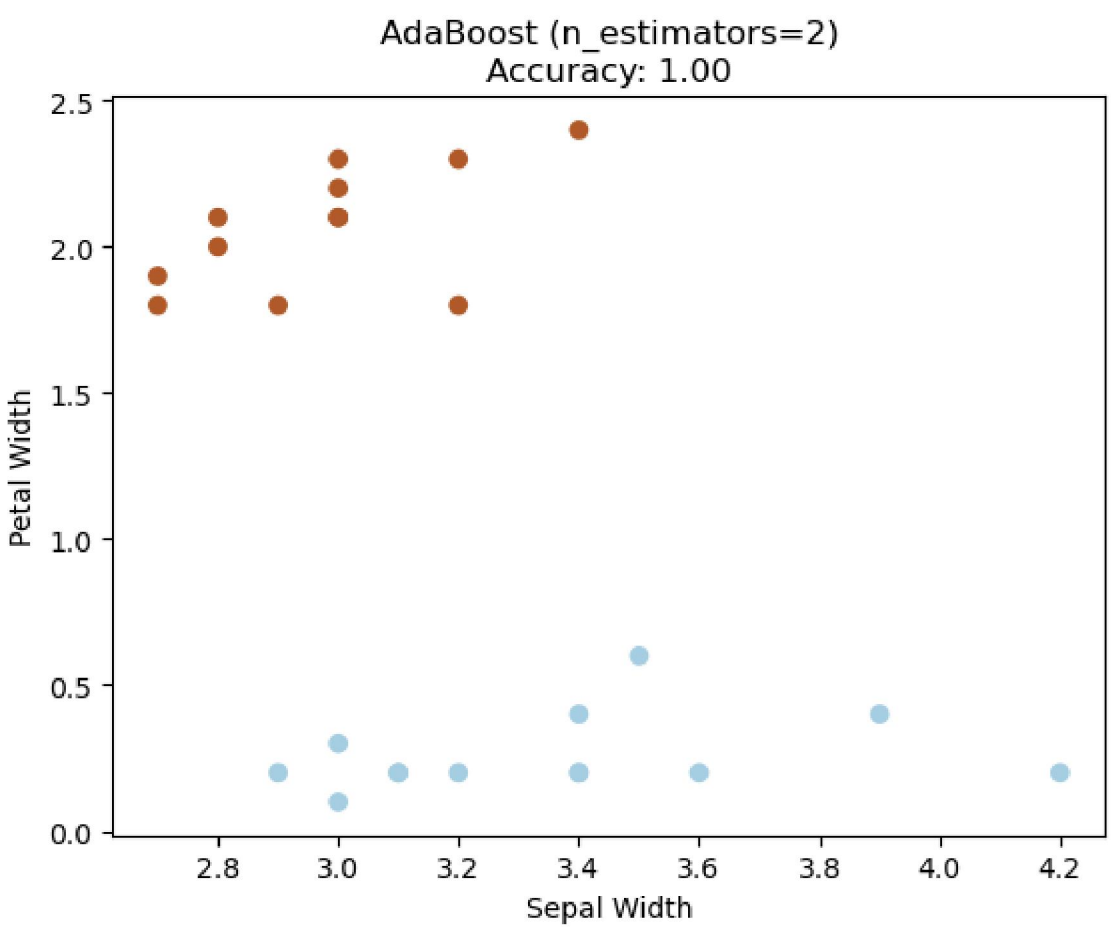
In [127…
```
n_estimators_values = [1, 2, 3]

n_estimator_result = []

for i, n_estimators in enumerate(n_estimators_values, 1):
    base_estimator = SVC(kernel='linear', probability=True, random_state=0)
    classifier_adaboost = AdaBoostClassifier(estimator=base_estimator, n_estimators=n_es

    classifier_adaboost.fit(X_train, y_train)

    y_pred_adaboost = classifier_adaboost.predict(X_test)

    n_estimator_result.append(y_pred_adaboost)
```

In [128…
```
plt.scatter(X_test['sepal_width'], X_test['petal_width'], c=n_estimator_result[0], cmap=
plt.title(f'AdaBoost (n_estimators={1})\nAccuracy: {accuracy_score(y_test, y_pred_adaboo
plt.xlabel('Sepal Width')
plt.ylabel('Petal Width')
plt.show()
```



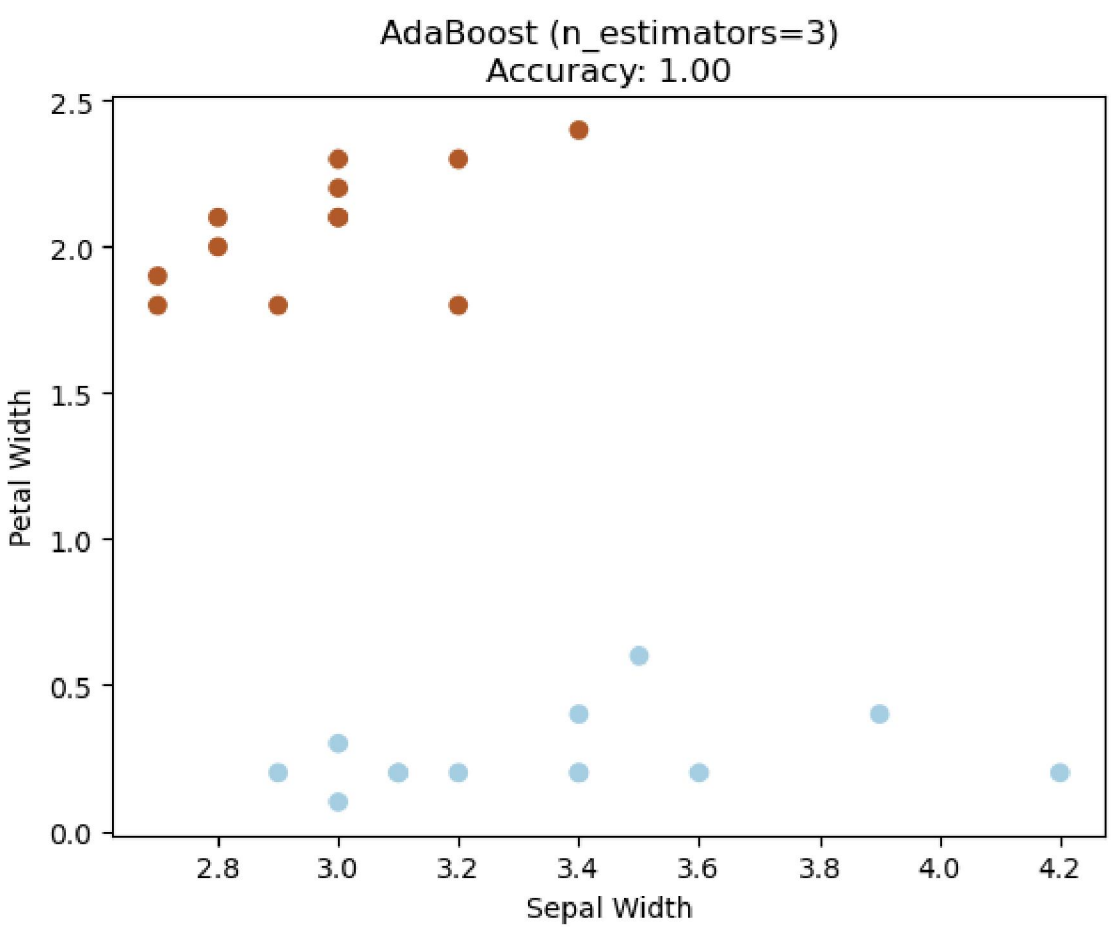
In [129…

```
plt.scatter(X_test['sepal_width'], X_test['petal_width'], c=n_estimator_result[1], cmap=
plt.title(f'AdaBoost (n_estimators={2})\nAccuracy: {accuracy_score(y_test, y_pred_adaboo
plt.xlabel('Sepal Width')
plt.ylabel('Petal Width')
plt.show()
```



In [130…

```
plt.scatter(X_test['sepal_width'], X_test['petal_width'], c=n_estimator_result[2], cmap=
plt.title(f'AdaBoost (n_estimators={3})\nAccuracy: {accuracy_score(y_test, y_pred_adaboo
plt.xlabel('Sepal Width')
plt.ylabel('Petal Width')
plt.show()
```



(d)

By comparing figure from part (b) to the three scatter plots, we see that AdaBoost gives an accurate decision boundary (the partition of the test data is the same of the separation of training data, where iris-setosa is on top and iris-virginica is at the bottom), moreover, we see that n_estimator = 1 gives decent performace already.

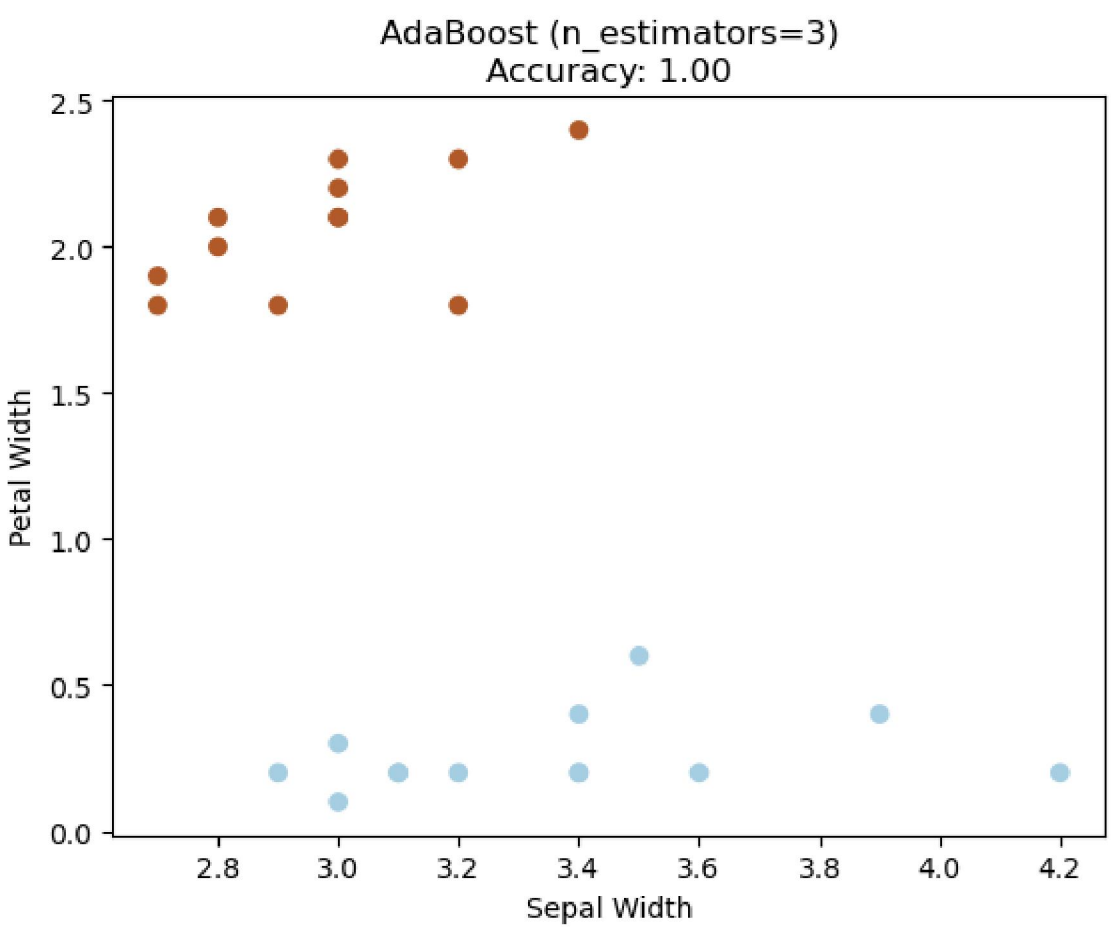
In [ ]:

4.3 4.3 **4 / 4**

✓ **- 0 pts** *Entirely correct*

- 2 pts Partial credit for Problem 4.3

- 4 pts No work shown for Problem 4.3



(d)

By comparing figure from part (b) to the three scatter plots, we see that AdaBoost gives an accurate decision boundary (the partition of the test data is the same of the separation of training data, where iris-setosa is on top and iris-virginica is at the bottom), moreover, we see that n_estimator = 1 gives decent performace already.

In [ ]:

4.4 4.4 **2 / 2**

✓ **- 0 pts** *Entirely correct*

- 1 pts Partial credit for Problem 4.4

- 2 pts No work shown for Problem 4.4