Question 2.a

Consider a data set comprising 400 data points from class C1 and 400 data points from class C2. Suppose that a decision stump model A splits these into two leaves at the root node; one containing (300,100) and the other containing (100,300) where (n,m) denotes n points are from class C1 and m points are from class C2. Similarly a second decision stump model B splits the examples as (200,400) and (200,0). Calculate the reduction in cost using misclassification rate for models A and B. Which is the preferred split (model A or model B) according to the cost calculations?

$$\text{cost}(D) = \frac{1}{|D|} \sum_{(x,y)\in D} I(y \neq \hat{y})$$

$$\hat{y} = \text{majority label in D}$$

ANSWER:

For Modal A

misclassification rate $= \text{C2}, \frac{400/800}{=} 0.50$

Found with 400 points from C1 and 400 points from C2, where C2 is the misclassification

with the first split, we find 100 misclassified points from C1 and C2, for the first and second leaf, respectively - so,

First Leaf

$C1 = \frac{100}{400} = 0.25$ and

Second Leaf

$C2 = \frac{100}{400} = 0.25$

this gives us an overall misclassification rate of

$\frac{0.25+0.25}{2} = 0.25$

and a reduction in cost of

$0.50 - 0.25 = 0.25$

For Modal B

misclassification rate is the same as in Model A $= \frac{400/800}{=} 0.50$

with the first split, we find 0 misclassified points from C1 on the second leaf and 200 from C2 on the first leaf - so,

First Leaf

$C1 = \frac{200}{600} = 0.33 \text{ and}$

Second Leaf

$C2 = \frac{0}{600} = 0.0$

this gives us an overall misclassification rate of

$\frac{0.33+0.0}{2} = 0.167$

and a reduction in cost of

$0.50 - 0.167 = 0.33$

Preferred Split

$\text{Greateest Reduction in Cost} = \text{Model B} = 0.33$

Question 2.b

If using the entropy to measure the cost, what is the answer to the question 2.a ?

Entropy

$$\text{Entropy}(S) = -p_{positive} log_2 p_{positive} - p_{negative} log_2 p_{negative}$$

ANSWER

Model A, Leaf 1

$p_{C1} = \frac{300}{400}$ and $p_{C2} = \frac{100}{400}$

$\text{Entropy } = -\frac{300}{400} log_2 \frac{300}{400} - \frac{100}{400} log_2 \frac{100}{400}$

$\text{Entropy } = 0.81$

Model A, Leaf 2

$p_{C1} = \frac{100}{400}$ and $p_{C2} = \frac{300}{400}$

$$\text{Entropy } = -\frac{100}{400}log_2\frac{100}{400} - \frac{300}{400}log_2\frac{300}{400}$$

$\text{Entropy } = 0.81

$$\text{Model B, Leaf 1}$$

$$p_{C1} = \frac{200}{600} \text{ and } p_{C2} = \frac{400}{600}$$

$$\text{Entropy } = -\frac{200}{600}log_2\frac{200}{600} - \frac{400}{600}log_2\frac{400}{600}$$

$\text{Entropy } = 0.91

$$\text{Model B, Leaf 2}$$

$$p_{C1} = \frac{200}{200} \text{ and } p_{C2} = \frac{0}{200}$$

$$\text{Entropy } = -\frac{200}{200}log_2\frac{200}{200} - \frac{0}{200}log_2\frac{0}{200}$$

$$\text{Entropy } = 0.0$$

Average Entropy of Model A = 0.81

Average Entropy of Model B = 0.46

A lower entropy number means that there is less uncertainty in the data after the split. Because of this, Model B would still be the correct choice.

In [ ]:
```
import numpy as np

# calculating Entropy
def entropy(p):
    # Handle the case where probability is 0, which would result in NaN in the log2 function
    return -p * np.log2(p) if p != 0 else 0

# Model A leaves
p_A1_C1 = 300 / 400
p_A1_C2 = 100 / 400
entropy_A1 = entropy(p_A1_C1) + entropy(p_A1_C2)

p_A2_C1 = 100 / 400
p_A2_C2 = 300 / 400
entropy_A2 = entropy(p_A2_C1) + entropy(p_A2_C2)

# Average entropy for Model A
average_entropy_A = (entropy_A1 + entropy_A2) / 2

# Model B leaves
p_B1_C1 = 200 / 600
```

```
p_B1_C2 = 400 / 600
entropy_B1 = entropy(p_B1_C1) + entropy(p_B1_C2)

p_B2_C1 = 1  # All C1
p_B2_C2 = 0  # No C2
entropy_B2 = entropy(p_B2_C1) + entropy(p_B2_C2)

# Average entropy for Model B
average_entropy_B = (entropy_B1 + entropy_B2) / 2

print(f'Entropy Model A, Leaf 1: {entropy_A1}')
print(f'Entropy Model A, Leaf 2: {entropy_A2}')
print(f'Entropy Model B, Leaf 1: {entropy_B1}')
print(f'Entropy Model B, Leaf 2: {entropy_B2}\n')

print(f'Avg Entropy for Model A {round(average_entropy_A, 2)}\nAvg Entropy for Model B {round(average_entropy_B, 2)}')
```

```
Entropy Model A, Leaf 1: 0.8112781244591328
Entropy Model A, Leaf 2: 0.8112781244591328
Entropy Model B, Leaf 1: 0.9182958340544896
Entropy Model B, Leaf 2: 0.0

Avg Entropy for Model A 0.81
Avg Entropy for Model B 0.46
```

Question 2.c

If using the Gini index to measure the cost, what is the answer to the question 2.a

$$\text{Gini Index}$$
$$cost(D) = 2p(1-p)$$

ANSWER

Model A, Leaf 1

$p_{C1} =$p_{C1} = \frac{300}{400} \text{ and } p_{C2} = \frac{100}{400}$

$\text{Gini Index} =$p_{C1} = 2\frac{300}{400}(\frac{100}{400})$

$\text{Gini Index} = 0.375$

Model A, Leaf 2

$p_{C1} = \frac{100}{400}$ and $p_{C2} = \frac{300}{400}$

$\text{Gini Index } =$p_{C1} = 2\frac{100}{400}(\frac{300}{400})$

$\text{Gini Index } = 0.375

Model B, Leaf 1

$p_{C1} = \frac{200}{600}$ and $p_{C2} = \frac{400}{600}$

$\text{Gini Index } =$p_{C1} = 2\frac{200}{600}(\frac{400}{600})$

$\text{Gini Index } = 0.44

Model B, Leaf 2

$p_{C1} = \frac{200}{200}$ and $p_{C2} = \frac{0}{200}$

$\text{Gini Index } =$p_{C1} = 2\frac{200}{200}(\frac{0}{200})$

$\text{Gini Index } = 0.0$

Average Gini Index of Model A = 0.375

Average Gini Index of Model B = 0.22

Gini Index determines which model has a lower "impurity" or disorder. So, we will choose model B, as the resulting 0.22 is the lowest and displays more order and purity in the data. This order/purity indicates similar class distributions.

In [ ]:
```
import matplotlib.pyplot as plt

import pandas as pd
```

In [ ]:
```
dataset = pd.read_csv("./play_tennis.csv")
```

In [ ]:
```
dataset
```

Out[ ]:

	day	outlook	temp	humidity	wind	play
0	D1	Sunny	Hot	High	Weak	No
1	D2	Sunny	Hot	High	Strong	No
2	D3	Overcast	Hot	High	Weak	Yes
3	D4	Rain	Mild	High	Weak	Yes
4	D5	Rain	Cool	Normal	Weak	Yes
5	D6	Rain	Cool	Normal	Strong	No
6	D7	Overcast	Cool	Normal	Strong	Yes
7	D8	Sunny	Mild	High	Weak	No
8	D9	Sunny	Cool	Normal	Weak	Yes
9	D10	Rain	Mild	Normal	Weak	Yes
10	D11	Sunny	Mild	Normal	Strong	Yes
11	D12	Overcast	Mild	High	Strong	Yes
12	D13	Overcast	Hot	Normal	Weak	Yes
13	D14	Rain	Mild	High	Strong	No

In [ ]:

```
categorical_cols = ["outlook", "temp", "humidity", "wind"]
for column in categorical_cols:
    dataset[column] = pd.factorize(dataset[column])[0]
dataset
```

Out[ ]:

	day	outlook	temp	humidity	wind	play
0	D1	0	0	0	0	No
1	D2	0	0	0	1	No
2	D3	1	0	0	0	Yes
3	D4	2	1	0	0	Yes
4	D5	2	2	1	0	Yes
5	D6	2	2	1	1	No
6	D7	1	2	1	1	Yes
7	D8	0	1	0	0	No
8	D9	0	2	1	0	Yes
9	D10	2	1	1	0	Yes
10	D11	0	1	1	1	Yes
11	D12	1	1	0	1	Yes
12	D13	1	0	1	0	Yes
13	D14	2	1	0	1	No

In [ ]:
```
dataset["play"] = dataset["play"].replace("Yes", 1)
dataset["play"] = dataset["play"].replace("No", 0)
target = dataset["play"]
dataset = dataset.drop("play", axis = 1)
dataset = dataset.drop("day", axis = 1)
```

In [ ]:
```
dataset
```

Out[ ]:

	outlook	temp	humidity	wind
0	0	0	0	0
1	0	0	0	1
2	1	0	0	0
3	2	1	0	0
4	2	2	1	0
5	2	2	1	1
6	1	2	1	1
7	0	1	0	0
8	0	2	1	0
9	2	1	1	0
10	0	1	1	1
11	1	1	0	1
12	1	0	1	0
13	2	1	0	1

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(dataset, target, test_size = 0.33, random_state = 42)
```

Question 1 Implement Decision Tree

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
# TODO :: define a sklearn DecisionTreeClassifier and set the max_depth to 3, expect 1 line of code
decision_tree_binary_classifier = DecisionTreeClassifier(max_depth=3, random_state=1)

# TODO :: fit the classifier your defined earlier, and fit on training data
decision_tree_binary_classifier.fit(X_train, y_train)
```

Out[ ]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, random_state=1)
```

In [ ]:

```
import numpy as np

y_pred_train = decision_tree_binary_classifier.predict(X_train)
```

```
num_correct = np.sum(y_pred_train == y_train)
print("accuracy on test set : {}".format(num_correct / float(len(y_train))))
```

```
accuracy on test set : 1.0
```

In [ ]:
```
y_pred_test = decision_tree_binary_classifier.predict(X_test)
num_correct = np.sum(y_pred_test == y_test)
print("accuracy on test set : {}".format(num_correct / float(len(y_pred_test))))
```

```
accuracy on test set : 0.6
```

Question 2 Tune the depth of the tree

Tune the max_depth parameter of DecisionTreeClassifier. How does the training accuracy and test accuracy change when you vary the value of max_depth?

In [ ]:
```
decision_tree_binary_classifier_a = DecisionTreeClassifier(max_depth=1, random_state=1)
decision_tree_binary_classifier_a.fit(X_train, y_train)

y_pred_train = decision_tree_binary_classifier_a.predict(X_train)
num_correct = np.sum(y_pred_train == y_train)
print("accuracy on test set : {}".format(num_correct / float(len(y_train))))

y_pred_test = decision_tree_binary_classifier_a.predict(X_test)
num_correct = np.sum(y_pred_test == y_test)
print("accuracy on test set : {}".format(num_correct / float(len(y_pred_test))))
```

```
accuracy on test set : 0.7777777777777778
accuracy on test set : 0.6
```

In [ ]:
```
decision_tree_binary_classifier_b = DecisionTreeClassifier(max_depth=2, random_state=1)
decision_tree_binary_classifier_b.fit(X_train, y_train)

y_pred_train = decision_tree_binary_classifier_b.predict(X_train)
num_correct = np.sum(y_pred_train == y_train)
print("accuracy on test set : {}".format(num_correct / float(len(y_train))))

y_pred_test = decision_tree_binary_classifier_b.predict(X_test)
num_correct = np.sum(y_pred_test == y_test)
print("accuracy on test set : {}".format(num_correct / float(len(y_pred_test))))
```

```
accuracy on test set : 0.8888888888888888
accuracy on test set : 0.8
```

In [ ]:
```
decision_tree_binary_classifier_c = DecisionTreeClassifier(max_depth=4, random_state=1)
decision_tree_binary_classifier_c.fit(X_train, y_train)

y_pred_train = decision_tree_binary_classifier_c.predict(X_train)
num_correct = np.sum(y_pred_train == y_train)
print("accuracy on test set : {}".format(num_correct / float(len(y_train))))
```

```
y_pred_test = decision_tree_binary_classifier_c.predict(X_test)
num_correct = np.sum(y_pred_test == y_test)
print("accuracy on test set : {}".format(num_correct / float(len(y_pred_test))))
```

```
accuracy on test set : 1.0
accuracy on test set : 0.6
```

ANSWER

Tuning the max depth parameter determines if our solution will be representative of a shallow or deep tree. A low max depth parameter seems to lower the training accuracy but at some points can increase test accuracy. However, high max depth can increase training accuracy but looks to lower test accuracy to a constant minimum, sub optimal point.