Homework 5 Written Part

Exercise 1a.

$$P(x=1) = \underline{\frac{1}{3}}$$

$$P(y=1) = \underline{\frac{2}{3}}$$

$$P(x=1, y=1) = \underline{\frac{1}{3}}$$

$$P(x=1|y=1) = \underline{\frac{1}{2}}$$

$$P(y=1|x=1) = \underline{1}$$

$$H(x=1) = -P(x=1)\log_2(P(x=1)) - P(x=0)\log_2(P(x=0)) = \underline{0.918}$$

$$H(y=1) = -P(y=1)\log_2(P(y=1)) - P(y=0)\log_2(P(y=0)) = \underline{0.918}$$

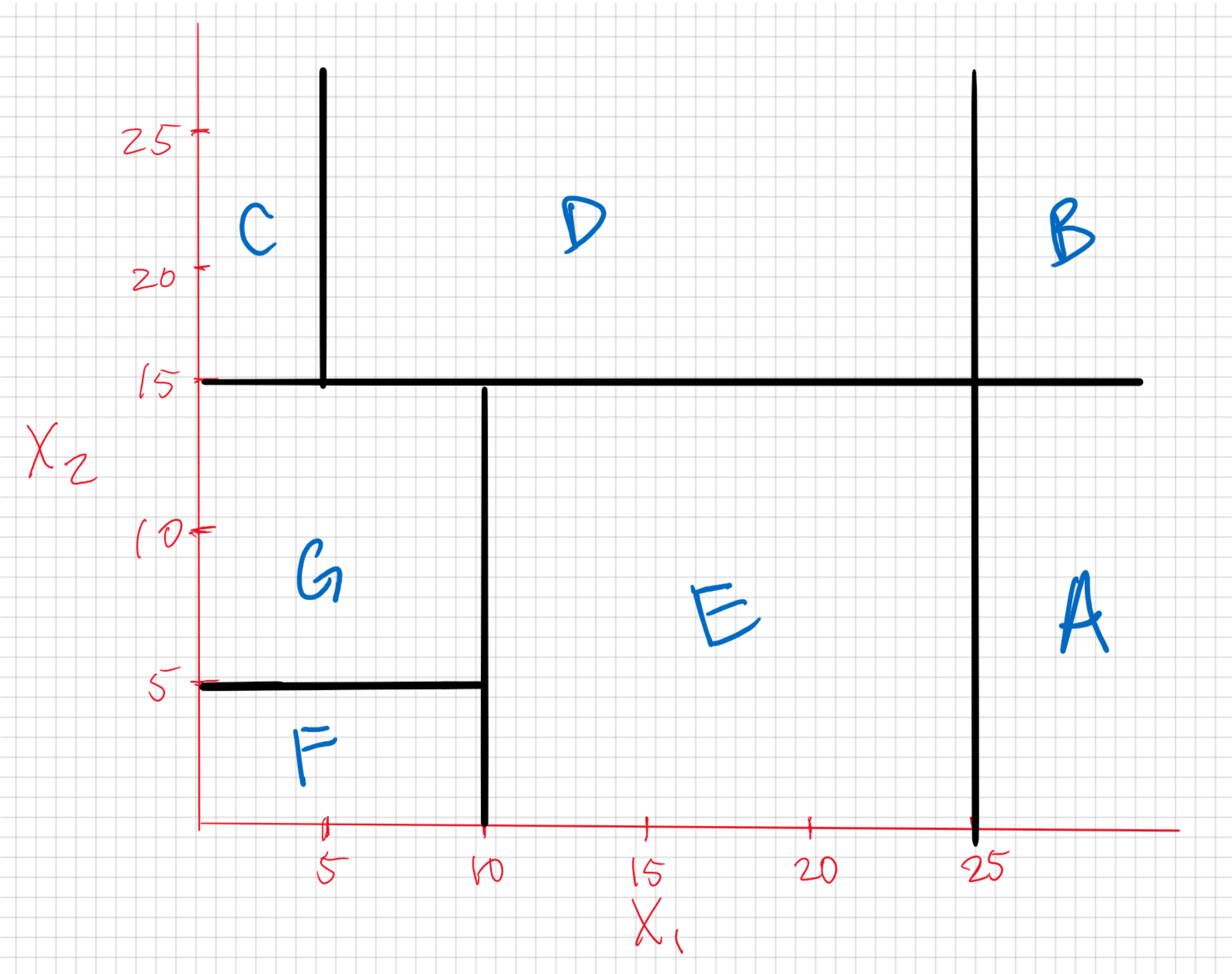
$$\begin{aligned} H(x=1|y=1) &= -P(y=1)[\frac{P(x=1,y=1)}{P(y=1)}\log_2(\frac{P(x=1,y=1)}{P(y=1)}) + \frac{P(x=0,y=1)}{P(y=1)}\log_2(\frac{P(x=0,y=1)}{P(y=1)})] \\ &-P(y=0)[\frac{P(x=1,y=0)}{P(y=0)}\log_2(\frac{P(x=1,y=0)}{P(y=0)}) + \frac{P(x=0,y=0)}{P(y=0)}\log_2(\frac{P(x=0,y=0)}{P(y=0)})] \\ &= -\frac{2}{3}\left(\frac{1}{2}\log_2\left(\frac{1}{2}\right) + \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right) - \frac{1}{3}\left(0\log_2(0) + \log_2(1)\right) = \underline{0.667} \end{aligned}$$

$$\begin{aligned} H(y=1|x=1) &= -P(x=1)[\frac{P(x=1,y=1)}{P(x=1)}\log_2(\frac{P(x=1,y=1)}{P(x=1)}) + \frac{P(x=1,y=0)}{P(x=1)}\log_2(\frac{P(x=1,y=0)}{P(x=1)})] \\ &-P(x=0)[\frac{P(x=0,y=1)}{P(x=0)}\log_2(\frac{P(x=0,y=1)}{P(x=0)}) + \frac{P(x=0,y=0)}{P(x=0)}\log_2(\frac{P(x=0,y=0)}{P(x=0)})] \\ &= -\frac{1}{3}\left(\log_2(1) + 0\log_2(0)\right) - \frac{2}{3}\left(\frac{1}{2}\log_2\left(\frac{1}{2}\right) + \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right) = \underline{0.667} \end{aligned}$$

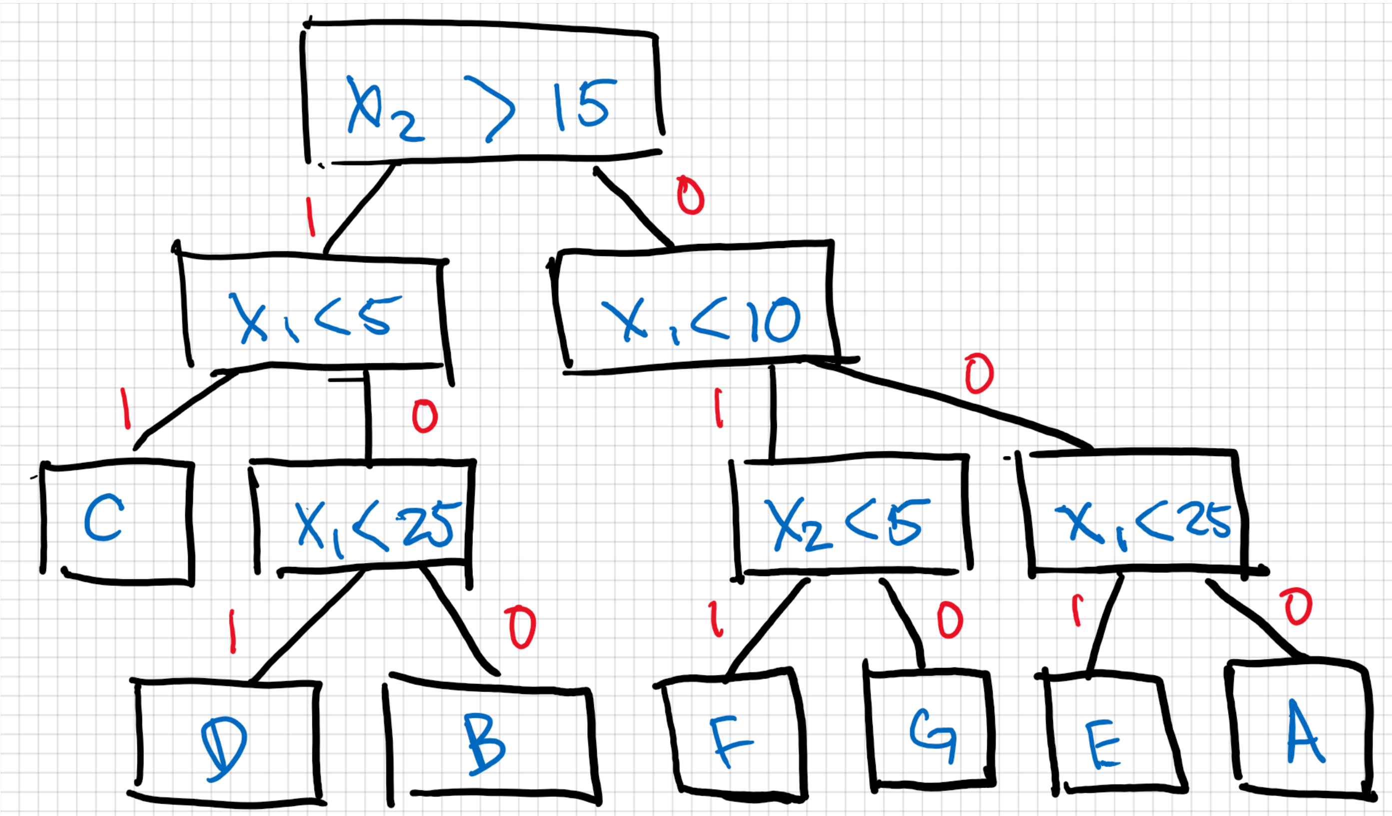
$$IG(y=1|x=1) = H(y=1) - H(y=1|x=1) = 0.918 - 0.667 = \underline{0.251}$$

P(x)	0.333	H(x)	0.918
P(y)	0.667	H(y)	0.918
P(x, y)	0.333	H(x|y)	0.667
P(x|y)	0.5	H(y|x)	0.667
P(y|x)	1	IG(y|x)	0.251

Exercise 1b.



Exercise 1c.



Exercise 2a.

$$H(Y) = [4^+, 4^-] = -\frac{4}{8}\log_2\left(\frac{4}{8}\right) - \frac{4}{8}\log_2\left(\frac{4}{8}\right) = 1$$

Exercise 2b.

$$H(Y=1|X_1=1) = -P(X_1=1)[\frac{P(X_1=1,Y=1)}{P(X_1=1)}\log_2(\frac{P(X_1=1,Y=1)}{P(X_1=1)}) + \frac{P(X_1=1,Y=0)}{P(X_1=1)}\log_2(\frac{P(X_1=1,Y=0)}{P(X_1=1)})]$$

$$-P(X_1=0)[\frac{P(X_1=0,Y=1)}{P(X_1=0)}\log_2(\frac{P(X_1=0,Y=1)}{P(X_1=0)}) + \frac{P(X_1=0,Y=0)}{P(X_1=0)}\log_2(\frac{P(X_1=0,Y=0)}{P(X_1=0)})]$$

$$= -\frac{1}{2}\left[\frac{1}{2}\log_2\left(\frac{1}{2}\right) + \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right] - \frac{1}{2}\left[\frac{1}{2}\log_2\left(\frac{1}{2}\right) + \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right] = 1$$

$$H(Y=1|X_2=1) = -P(X_2=1)[\frac{P(X_2=1,Y=1)}{P(X_2=1)}\log_2(\frac{P(X_2=1,Y=1)}{P(X_2=1)}) + \frac{P(X_2=1,Y=0)}{P(X_2=1)}\log_2(\frac{P(X_2=1,Y=0)}{P(X_2=1)})]$$

$$-P(X_2=0)[\frac{P(X_2=0,Y=1)}{P(X_2=0)}\log_2(\frac{P(X_2=0,Y=1)}{P(X_2=0)}) + \frac{P(X_2=0,Y=0)}{P(X_2=0)}\log_2(\frac{P(X_2=0,Y=0)}{P(X_2=0)})]$$

$$= -\frac{1}{2}\left[\frac{1}{2}\log_2\left(\frac{1}{2}\right) + \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right] - \frac{1}{2}\left[\frac{1}{2}\log_2\left(\frac{1}{2}\right) + \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right] = 1$$

$$H(Y=1|X_3=1) = -P(X_3=1)[\frac{P(X_3=1,Y=1)}{P(X_3=1)}\log_2(\frac{P(X_3=1,Y=1)}{P(X_3=1)}) + \frac{P(X_3=1,Y=0)}{P(X_3=1)}\log_2(\frac{P(X_3=1,Y=0)}{P(X_3=1)})]$$

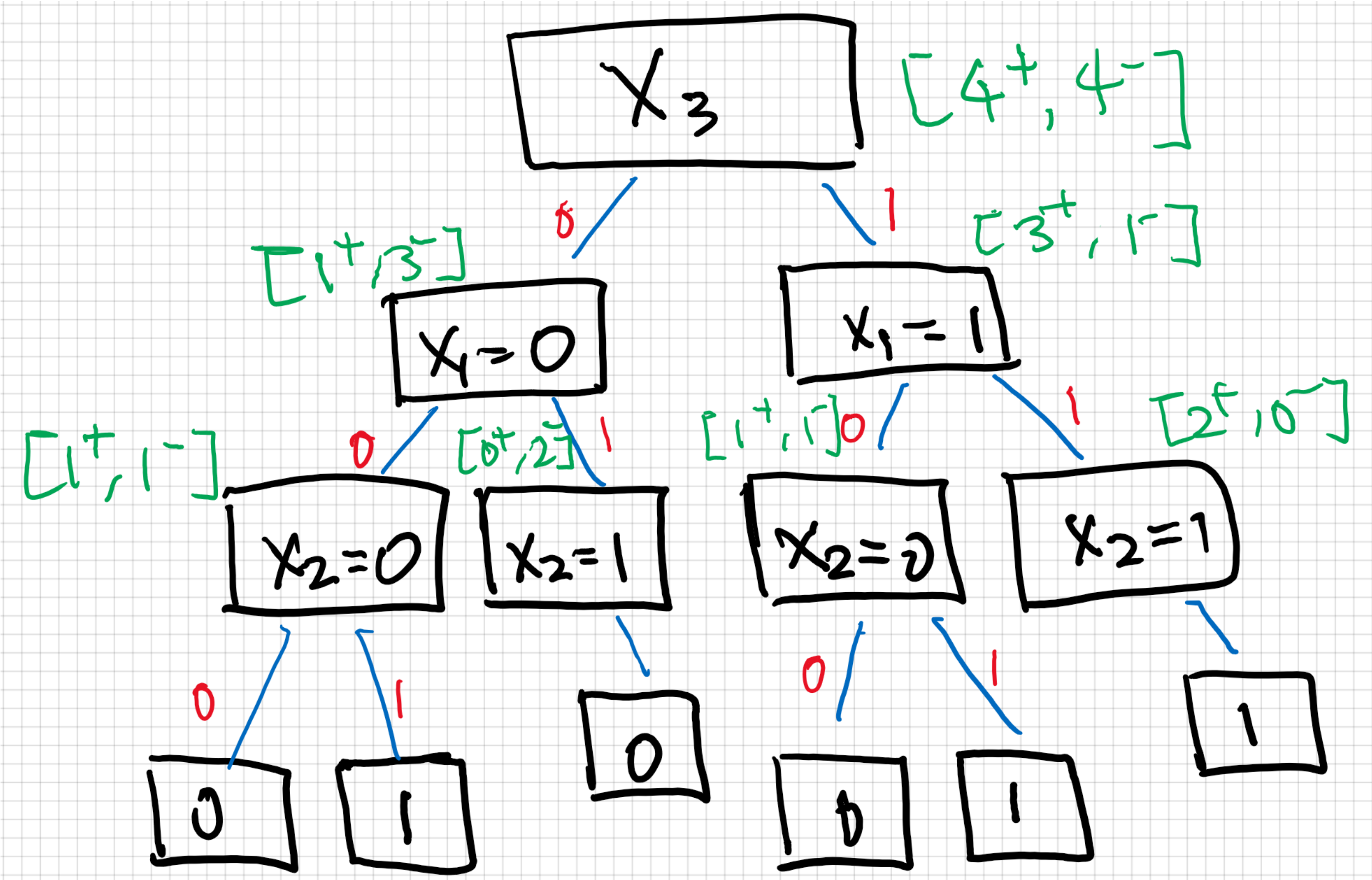
$$-P(X_3=0)[\frac{P(X_3=0,Y=1)}{P(X_3=0)}\log_2(\frac{P(X_3=0,Y=1)}{P(X_3=0)}) + \frac{P(X_3=0,Y=0)}{P(X_3=0)}\log_2(\frac{P(X_3=0,Y=0)}{P(X_3=0)})]$$

$$= -\frac{1}{2}\left[\frac{3}{4}\log_2\left(\frac{3}{4}\right) + \frac{1}{4}\log_2\left(\frac{1}{4}\right)\right] - \frac{1}{2}\left[\frac{1}{4}\log_2\left(\frac{1}{4}\right) + \frac{3}{4}\log_2\left(\frac{3}{4}\right)\right] = 0.8112$$

$H(Y\|X_1)$	1	$IG(Y\|X_1) = H(Y) - H(Y\|X_1)$	0
$H(Y\|X_2)$	1	$IG(Y\|X_2) = H(Y) - H(Y\|X_2)$	0
$H(Y\|X_3)$	0.8112	$IG(Y\|X_3) = H(Y) - H(Y\|X_3)$	0.1888

Exercise 2c.
The branches of the tree should stop growing when the particular branch reaches highest purity (sample is perfectly classified). The tree should stop growing altogether when all of the branches reaches the aforementioned purity.

Exercise 2d.



Training error is 0% since everything is perfectly classified.

Exercise 2e.
Following the decision tree in previous section, the result of the testing instances are as below:

Instance	X_1	X_2	X_3	Y
9	1	1	1	1
10	1	0	0	0
11	0	1	1	1

Exercise 2f.
Decision tree intrinsically tends to overfit when there is a large number of nodes. To combat overfitting in a decision tree, we can set a maximum feature to consider for a split, this way we can ensure there is a control on the nodes being created as descendant of the previous feature.

Exercise 2g.
The structure of the tree will change since X_3 will not be the best feature to split anymore. The leaf nodes are also affected because by simply observing path $(0 \rightarrow 1)$ and $(1 \rightarrow 1)$ (referencing from Exercise 2d), these two paths now leads to ambiguous results, whereas before it leads to a perfect labeling of 0 and 1 respectively. Additionally, no further training can be done to resolve this since Instance 6 shares the same features as Instance 5, and Instance 7 to Instance 8 in a similar fashion.

Exercise 2h.
See output plot from sklearn below:

In [1]:
```
from sklearn import tree
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
X_train = np.array([[0, 0, 0],
                    [0, 0, 1],
                    [0, 1, 0],
                    [0, 1, 1],
                    [1, 0, 1],
                    [1, 0, 1],
                    [1, 1, 0],
                    [1, 1, 0]])
y_train = np.array([0, 0, 1, 1, 1, 1, 0, 0])
```

In [3]:
```
clf = tree.DecisionTreeClassifier().fit(X_train, y_train)
```

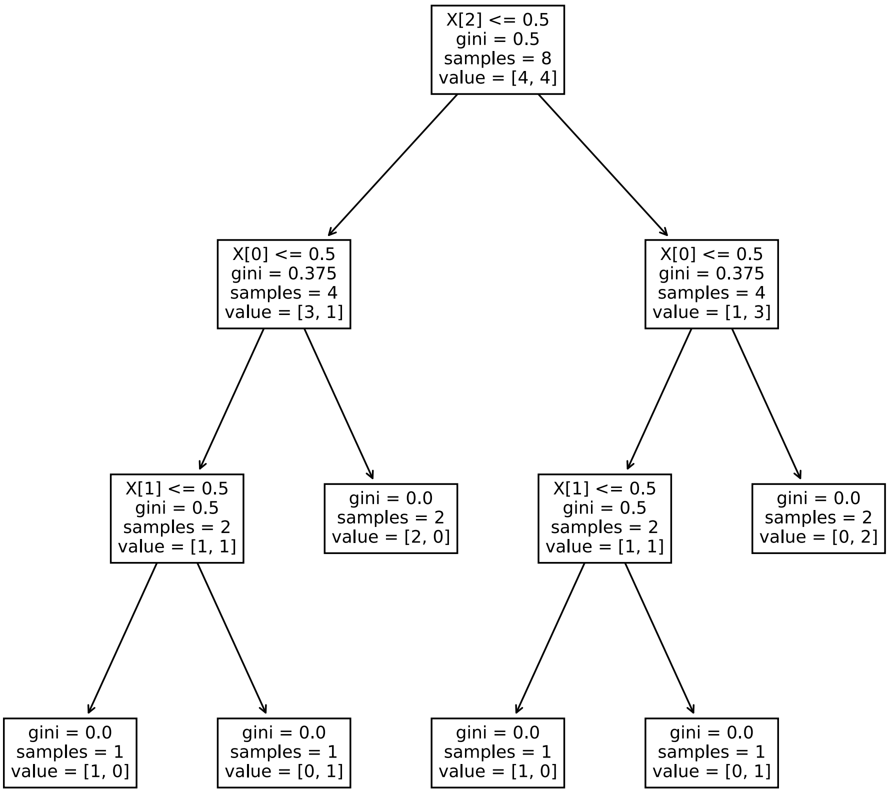
In [4]:
```
X_test = np.array([[1, 1, 1],
                   [1, 0, 0],
                   [0, 1, 1]])
```

In [5]:
```
y_test = clf.predict(X_test)
print(y_test)
```

```
[1 0 1]
```

In [6]:
```
plt.figure(figsize=(10,10))
tree.plot_tree(clf, fontsize=10)
```

Out[6]:
```
[Text(310.0, 475.65000000000003, 'X[2] <= 0.5\ngini = 0.5\nsamples = 8\nvalue = [4,
4]'),
 Text(186.0, 339.75, 'X[0] <= 0.5\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(124.0, 203.85000000000002, 'X[1] <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1,
1]'),
 Text(62.0, 67.94999999999999, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(186.0, 67.94999999999999, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(248.0, 203.85000000000002, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(434.0, 339.75, 'X[0] <= 0.5\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
 Text(372.0, 203.85000000000002, 'X[1] <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1,
1]'),
 Text(310.0, 67.94999999999999, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(434.0, 67.94999999999999, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(496.0, 203.85000000000002, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]')]
```


X[2] <= 0.5
gini = 0.5
samples = 8
value = [4, 4]
X[0] <= 0.5
gini = 0.375
samples = 4
value = [3, 1]
X[0] <= 0.5
gini = 0.375
samples = 4
value = [1, 3]
X[1] <= 0.5
gini = 0.5
samples = 2
value = [1, 1]
gini = 0.0
samples = 2
value = [2, 0]
X[1] <= 0.5
gini = 0.5
samples = 2
value = [1, 1]
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 = 1
value = [1, 0]
gini = 0.0
samples = 1
value = [0, 1]

Exercise 3a. The Gini impurity of the weather in Pittsburgh is calculated as below:

$$G = 1 - \left(\left(\frac{6}{31}\right)^2 + \left(\frac{10}{31}\right)^2 + \left(\frac{10}{31}\right)^2 + \left(\frac{2}{31}\right)^2 + \left(\frac{3}{31}\right)^2\right) = 0.74$$

Exercise 3b.

cart.py

```python
"""Implementation of the CART algorithm to train decision tree classifiers."""
import numpy as np
import tree
import graphviz


class DecisionTreeClassifier:
    def __init__(self, max_depth=None):
        self.max_depth = max_depth

    def fit(self, X, y):
        """Build decision tree classifier."""

        self.n_classes_ = len(set(y))  # classes are assumed to go from 0 to n-1
        self.n_features_ = X.shape[1]

        self.tree_ = self._grow_tree(X, y)

    def predict(self, X):
        """Predict class for X."""
        return [self._predict(inputs) for inputs in X]

    def debug(self, feature_names, class_names, show_details=True):
        """Print ASCII visualization of decision tree."""
        self.tree_.debug(feature_names, class_names, show_details)

    def _gini(self, y):
        """Compute Gini impurity of a non-empty node.
        Gini impurity is defined as sum p(1-p) over all classes, with p the frequency of a
        class within the node. Since sum p = 1, this is equivalent to 1 - sum p^2.
        """
        m = y.size
        # 1. Your code goes here (fill in variable for self._grow_tree)
        # you will need the variable self.n_classes_ which is the number of classes you need
        # to calculate the gini impurity
        gini_impurity = 1.0 - sum((y[i] / sum(y)) ** 2 for i in range(m))
        # 1. Your code goes here above
        return gini_impurity

    def _best_split(self, X, y):
        """Find the best split for a node.

        "Best" means that the average impurity of the two children, weighted by their
        population, is the smallest possible. Additionally it must be less than the
        impurity of the current node.

        To find the best split, we loop through all the features, and consider all the
        midpoints between adjacent training samples as possible thresholds. We compute
        the Gini impurity of the split generated by that particular feature/threshold
        pair, and return the pair with smallest impurity.

        Returns:
            best_idx: Index of the feature for best split, or None if no split is found.
            best_thr: Threshold to use for the split, or None if no split is found.
        """
        # Need at least two elements to split a node.
        m = y.size
        if m <= 1:
            return None, None

        # Count of each class in the current node.
        num_parent = [np.sum(y == c) for c in range(self.n_classes_)]
```

Exercise 3b.


```python
        # Gini of current node.
        best_gini = 1.0 - sum((n / m) ** 2 for n in num_parent)
        best_idx, best_thr = None, None

        # Loop through all features.
        for idx in range(self.n_features_):
            # Sort data along selected feature.
            thresholds, classes = zip(*sorted(zip(X[:, idx], y)))

            # We could actually split the node according to each feature/threshold pair
            # and count the resulting population for each class in the children, but
            # instead we compute them in an iterative fashion, making this for loop
            # linear rather than quadratic.
            num_left = [0] * self.n_classes_
            num_right = num_parent.copy()
            for i in range(1, m):  # possible split positions
                c = classes[i - 1]
                num_left[c] += 1
                num_right[c] -= 1
                gini_left = 1.0 - sum(
                    (num_left[x] / i) ** 2 for x in range(self.n_classes_)
                )
                gini_right = 1.0 - sum(
                    (num_right[x] / (m - i)) ** 2 for x in range(self.n_classes_)
                )

                # The Gini impurity of a split is the weighted average of the Gini
                # impurity of the children.
                gini = (i * gini_left + (m - i) * gini_right) / m

                # The following condition is to make sure we don't try to split two
                # points with identical values for that feature, as it is impossible
                # (both have to end up on the same side of a split).
                if thresholds[i] == thresholds[i - 1]:
                    continue

                if gini < best_gini:
                    best_gini = gini
                    best_idx = idx
                    best_thr = (thresholds[i] + thresholds[i - 1]) / 2  # midpoint

        return best_idx, best_thr

    def _grow_tree(self, X, y, depth=0):
        """Build a decision tree by recursively finding the best split."""
        # Population for each class in current node. The predicted class is the one with
        # largest population.
        num_samples_per_class = [np.sum(y == i) for i in range(self.n_classes_)]

        # Your code goes here to get the predicted class (should be just 1-2 lines)
        predicted_class = np.argmax(num_samples_per_class)

        node = tree.Node(
            gini=self._gini(y),
            num_samples=y.size,
            num_samples_per_class=num_samples_per_class,
            predicted_class=predicted_class,
        )

        # Split recursively until maximum depth is reached.
        if depth < self.max_depth:
            idx, thr = self._best_split(X, y)
            if idx is not None:
                indices_left = X[:, idx] < thr
```

Exercise 3b.


```
                X_left, y_left = X[indices_left], y[indices_left]
                X_right, y_right = X[~indices_left], y[~indices_left]
                node.feature_index = idx
                node.threshold = thr
                # 2. Your code goes here (fill in variable for self._grow_tree)
                depth += 1
                node.left = self._grow_tree(X_left, y_left, depth)
                node.right = self._grow_tree(X_right, y_right, depth)
                # 2. Your code goes here above

        return node

    def _predict(self, inputs):
        """Predict class for a single sample."""
        node = self.tree_
        while node.left:
            if inputs[node.feature_index] < node.threshold:
                node = node.left
            else:
                node = node.right
        return node.predicted_class


if __name__ == "__main__":
    import argparse
    import pandas as pd
    from sklearn.datasets import load_breast_cancer, load_iris
    from sklearn.tree import DecisionTreeClassifier as SklearnDecisionTreeClassifier
    from sklearn.tree import export_graphviz
    from sklearn.utils import Bunch

    parser = argparse.ArgumentParser(description="Train a decision tree.")
    parser.add_argument("-f", "--fff", help="a dummy argument to fool ipython", default="1")
    parser.add_argument("--dataset", choices=["breast", "iris", "wifi"], default="wifi")
    parser.add_argument("--max_depth", type=int, default=2)
    parser.add_argument("--hide_details", dest="hide_details", action="store_true")
    parser.set_defaults(hide_details=True)
    parser.add_argument("--use_sklearn", dest="use_sklearn", action="store_true")
    parser.set_defaults(use_sklearn=False)
    args = parser.parse_args()

    # 1. Load dataset.
    if args.dataset == "breast":
        dataset = load_breast_cancer()
    elif args.dataset == "iris":
        dataset = load_iris()
    elif args.dataset == "wifi":
        # https://archive.ics.uci.edu/ml/datasets/Wireless+Indoor+Localization
        df = pd.read_csv("wifi_localization.txt", delimiter="\t")
        data = df.to_numpy()
        dataset = Bunch(
            data=data[:, :-1],
            target=data[:, -1] - 1,
            feature_names=["Wifi {}".format(i) for i in range(1, 8)],
            target_names=["Room {}".format(i) for i in range(1, 5)],
        )
    X, y = dataset.data, dataset.target

    # 2. Fit decision tree.
    if args.use_sklearn:
        clf = SklearnDecisionTreeClassifier(max_depth=args.max_depth)
    else:
        clf = DecisionTreeClassifier(max_depth=args.max_depth)
    clf.fit(X, y)
```

Exercise 3b.


```
    # 3. Predict.
    if args.dataset == "iris":
        #         input = [0, 0, 5.0, 1.5]
        input = [0, 0, 3.0, 2.5]

    elif args.dataset == "wifi":
        input = [-70, 0, 0, 0, -40, 0, 33]
    elif args.dataset == "breast":
        input = [np.random.rand() for _ in range(30)]
    pred = clf.predict([input])[0]
    print("Input: {}".format(input))
    print("Prediction: " + dataset.target_names[pred])

    # 4. Visualize.
    if args.use_sklearn:
        dot_data = export_graphviz(
            clf,
            out_file=None,  # "tree.dot",
            feature_names=dataset.feature_names,
            class_names=dataset.target_names,
            rounded=True,
            filled=True,
        )
        graph = graphviz.Source(dot_data, format='png')
        graph
        print("Done. To convert to PNG, run: dot -Tpng tree.dot -o tree.png")
    else:
        clf.debug(
            list(dataset.feature_names),
            list(dataset.target_names),
            not args.hide_details,
        )
```

Exercise 3c.

Q3 CART from scratch and sklearn

b. Code Completion

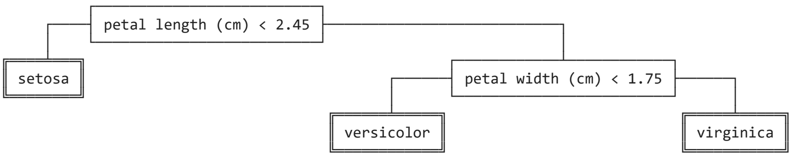
See "cart.py"

c. Experimenting your models

(1) Dataset iris with Python Decision tree

In [1]:
```
%run cart.py --dataset="iris" --max_depth=2 --hide_details
```

```
Input: [0, 0, 3.0, 2.5]
Prediction: virginica
```




```
c:\Users\ivanw\Documents\College & Grad School\Carnegie Mellon\24-787 Machine Learning\HW5\cart.py:35: RuntimeWarning: in
valid value encountered in long_scalars
  gini_impurity = 1.0 - sum((y[i] / sum(y)) ** 2 for i in range(m))
```

(2) Dataset iris with sklearn Decision tree

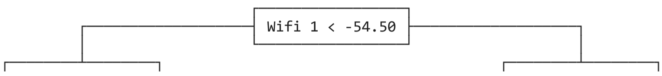
In [2]:
```
%run cart.py --dataset="iris" --max_depth=2 --hide_details --use_sklearn
```

```
Input: [0, 0, 3.0, 2.5]
Prediction: virginica
Done. To convert to PNG, run: dot -Tpng tree.dot -o tree.png
```

(3) Dataset wifi with Python Decision tree

In [3]:
```
%run cart.py --dataset="wifi" --max_depth=2 --hide_details
```

```
Input: [-70, 0, 0, 0, -40, 0, 33]
Prediction: Room 4
```

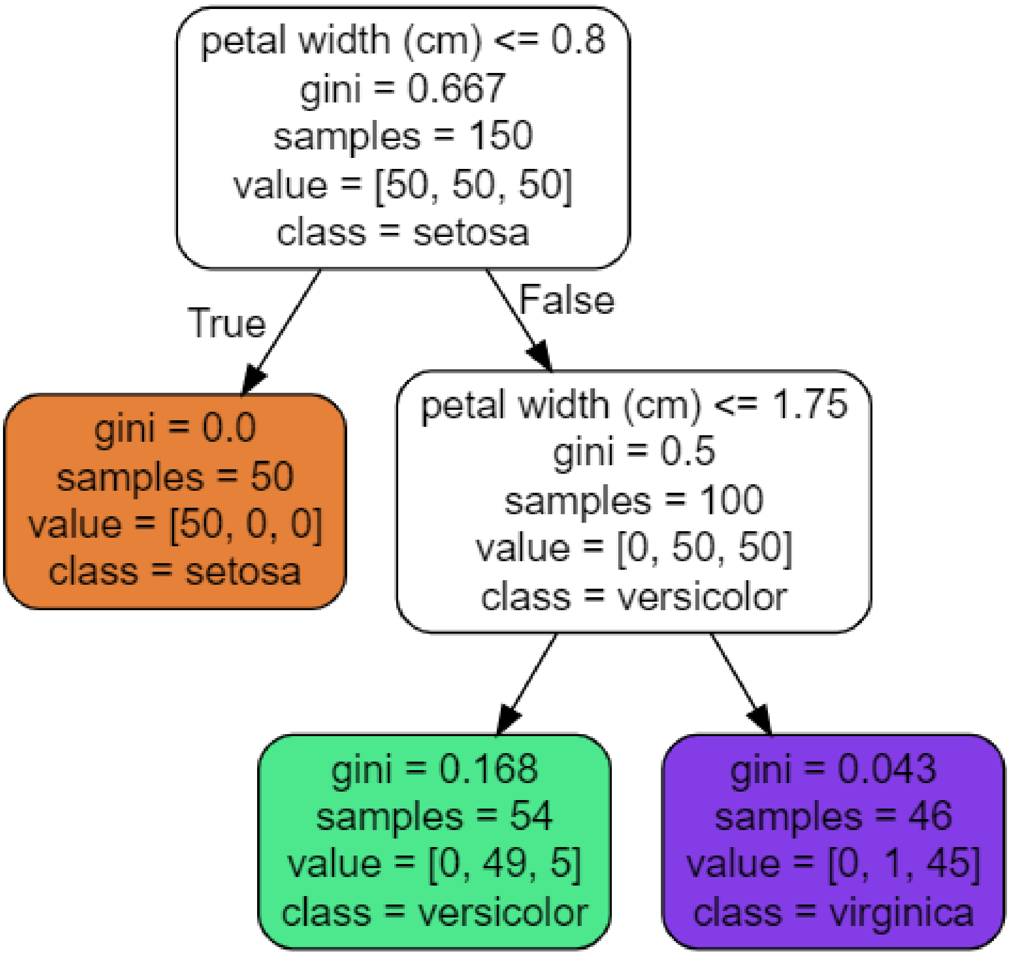


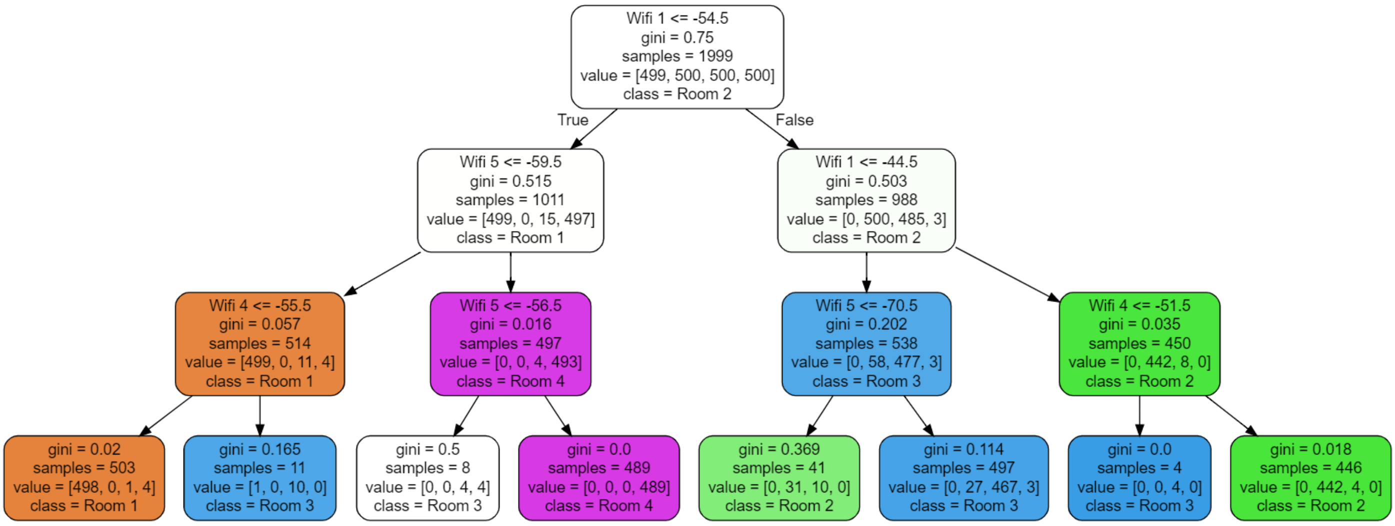
Exercise 3c.

(4) Dataset wifi with sklearn Decision tree

In [4]:
```
%run cart.py --dataset="wifi" --max_depth=3 --use_sklearn
```

```
Input: [-70, 0, 0, 0, -40, 0, 33]
Prediction: Room 4
Done. To convert to PNG, run: dot -Tpng tree.dot -o tree.png
```





The IRIS samples are the same for our manual CART algorithm and sklearn library. However, the Wifi samples are different because one uses a maximum depth of 2 and the other uses a maximum depth of 3, therefore, this may cause more splitting within the model and will affect the terminating leaf nodes at the maximum depth.