Homework Assignment 4

CSE 151A: Introduction to Machine Learning

Due: May 30th, 2023, 9:30am (Pacific Time)

Instructions: Please answer the questions below, attach your code in the document, and insert figures to create **a single PDF file**. You may search information online but you will need to write code/find solutions to answer the questions yourself.

Grade: ____ out of 100 points

1 (40 points) Naïve Bayes

In this question, we would like to build a Naïve Bayes model for a classification task. Assume there is a classification dataset $S = \{(\mathbf{x}^{(i)}, y^{(i)}), i = 1, ..., 8\}$ where each data point $(\mathbf{x}, y)$ contains a feature vector $\mathbf{x} = (x_1, x_2, x_3); x_1, x_2, x_3 \in \{0, 1\}$ and a ground-truth label $y \in \{0, 1\}$. The dataset S can be read from the table below:

i	x_1	x_2	x_3	y
1	0	0	1	1
2	0	1	1	1
3	1	1	0	1
4	0	0	1	1
5	0	1	0	0
6	1	1	0	0
7	1	0	0	0
8	0	0	1	0

In Naïve Bayes model, we use random variable $X_i \in \{0, 1\}$ to represent i-th dimension of the feature vector $\mathbf{x}$, and random variable $Y \in \{0, 1\}$ to represent the class label y. Thus, we can estimate probabilities $P(Y)$, $P(X_i|Y)$ and $P(X_i, Y)$ by counting data points in dataset S, for example:

$$P(Y = 1) = \frac{\#\{\text{data points with } y = 1\}}{\#\{\text{all data points}\}} = \frac{4}{8} = 0.5$$

$$P(X_1 = 1|Y = 0) = \frac{\#\{\text{data points with } x_1 = 1 \text{ and } y = 0\}}{\#\{\text{data points with } y = 0\}} = \frac{2}{4} = 0.5$$

$$\begin{aligned} P(X_1 = 1, Y = 1) &= P(X_1 = 1|Y = 1)P(Y = 1) \\ &= \frac{\#\{\text{data points with } x_1 = 1 \text{ and } y = 1\}}{\#\{\text{all data points}\}} = \frac{1}{8} = 0.125 \end{aligned}$$

It is noteworthy that **only** probabilities $P(Y)$, $P(X_i|Y)$ and $P(X_i, Y)$ can be **directly** estimated from dataset S in Naïve Bayes model. Other joint probabilities (e.g. $P(X_1, X_2)$ and $P(X_1, X_2, X_3)$) should **not** be estimated by directly counting the data points.

Next, we can use the probabilities $P(Y)$ and $P(X_i|Y)$ to build our Naïve Bayes model for classification: For a feature vector $\mathbf{x} = (x_1, x_2, x_3)$, we can estimate the probability $P(Y = y|X_1 = x_1, X_2 = x_2, X_3 = x_3)$ with the **conditional independence assumptions**:

$$\begin{aligned} P(Y = y|X_1 = x_1, X_2 = x_2, X_3 = x_3) &= \frac{P(X_1 = x_1, X_2 = x_2, X_3 = x_3, Y = y)}{P(X_1 = x_1, X_2 = x_2, X_3 = x_3)} \\ &= \frac{P(X_1 = x_1, X_2 = x_2, X_3 = x_3|Y = y)P(Y = y)}{P(X_1 = x_1, X_2 = x_2, X_3 = x_3)} \\ &= \frac{\Big(\prod_{i=1}^{3} P(X_i = x_i|Y = y)\Big)P(Y = y)}{P(X_1 = x_1, X_2 = x_2, X_3 = x_3)} \end{aligned}$$

where the joint probability $P(X_1 = x_1, X_2 = x_2, X_3 = x_3)$ can be calculated as:

$$\begin{aligned} P(X_1 = x_1, X_2 = x_2, X_3 = x_3) &= \sum_{y=0}^{1} P(X_1 = x_1, X_2 = x_2, X_3 = x_3, Y = y) \\ &= \sum_{y=0}^{1} \Big(P(X_1 = x_1, X_2 = x_2, X_3 = x_3|Y = y)P(Y = y)\Big) \\ &= \sum_{y=0}^{1} \left(\Big(\prod_{i=1}^{3} P(X_i = x_i|Y = y)\Big)P(Y = y)\right) \end{aligned}$$

Finally, if we find:

$$P(Y = 1|X_1 = x_1, X_2 = x_2, X_3 = x_3) > P(Y = 0|X_1 = x_1, X_2 = x_2, X_3 = x_3)$$

then we can predict the class of feature vector $\mathbf{x} = (x_1, x_2, x_3)$ to be 1, otherwise 0. It is noteworthy that although conditional independence assumptions are made in Naïve Bayes model, $P(Y = 1|X_1 = x_1, X_2 = x_2, X_3 = x_3) + P(Y = 0|X_1 = x_1, X_2 = x_2, X_3 = x_3)$ should **still be** 1.

1. (15 pts) Please estimate the following probabilities:

   $$(1)\ P(X_1 = 1, Y = 0), \quad (2)\ P(Y = 0), \quad (3)\ P(X_1 = 1|Y = 1).$$

   Note that these probabilities can be directly estimated by counting from dataset S.

   $P(X_1 = 1, Y = 0) = \frac{2}{8} = \boxed{0.25}$

   $P(Y = 0) = \frac{4}{8} = \boxed{0.5}$

   $P(X_1 = 1 \mid Y = 1) = \frac{1}{4} = \boxed{0.25}$

2. (18 pts) Please calculate the probability $P(Y = 1|X_1 = 1, X_2 = 1, X_3 = 0)$ in Naïve Bayes model using conditional independence assumptions.

$$P(Y=1 \mid X_1=1, X_2=1, X_3=0) = \frac{\left(\prod_{i=1}^{3} P(X_i = x_i \mid Y=1)\right) P(Y=1)}{P(X_1=x_1, X_2=x_2, X_3=x_3)}$$

$$\left(\prod_{i=1}^{3} P(X_i = x_i \mid Y=1)\right) P(Y=1) = \left(\frac{1}{4} \cdot \frac{1}{2} \cdot \frac{1}{4}\right) \cdot \frac{1}{2}$$

$$= \frac{1}{64}$$

$$P(X_1=x_1, X_2=x_2, X_3=x_3) = \sum_{y=0}^{1} \left(\left(\prod_{i=1}^{3} P(X_i = x_i \mid Y=y)\right) P(Y=y)\right)$$

$$= \left(\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{3}{4}\right) \cdot \frac{1}{2} + \frac{1}{64} = \frac{7}{64}$$

$$\frac{\left(\prod_{i=1}^{3} P(X_i = x_i \mid Y=1)\right) P(Y=1)}{P(X_1=x_1, X_2=x_2, X_3=x_3)} = \frac{1/64}{7/64} = \boxed{\frac{1}{7}}$$

3. (7 pts) Please calculate the probability $P(Y = 0|X_1 = 1, X_2 = 1, X_3 = 0)$ in Naïve Bayes model and predict the class of feature vector $\mathbf{x} = (1, 1, 0)$.

$$P(Y=0 \mid X_1=1, X_2=1, X_3=0) = \frac{\left(\prod_{i=1}^{3} P(X_i = x_i \mid Y=0)\right) P(Y=0)}{P(X_1=1, X_2=1, X_3=0)}$$

$$= \frac{6/64}{7/64} = \frac{6}{7}$$

$$P(Y=0 \mid X_1=1, X_2=1, X_3=0) = \frac{6}{7} > P(Y=1 \mid X_1=1, X_2=1, X_3=0) = \frac{1}{7}$$

$$\boxed{x:\ y=0}$$

2 (40 points) Decision Tree

In this question, we would like to create a decision tree model for a binary classification task. Assume there is a classification dataset $T = \{(\mathbf{x}^{(i)}, y^{(i)}), i = 1, ..., 5\}$ where each data point $(\mathbf{x}, y)$ contains a feature vector $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ and a ground-truth label $y \in \{0, 1\}$. The dataset T can be read from the table below:

i	x_1	x_2	y
1	1.0	2.0	1
2	2.0	2.0	1
3	3.0	2.0	0
4	2.0	3.0	0
5	1.0	3.0	0

To build the decision tree model, we use a simplified CART algorithm, which is a recursive procedure as follows:

- Initialize a root node with dataset T and set it as current node.
- Start a procedure for current node:
  - **Step 1**: Assume the dataset in current node is T_{cur}. Check if all data points in T_{cur} are in the same class:
    - If it is true, set current node as a *leaf node* to predict the common class in T_{cur}, and then terminate *current* procedure.
    - If it is false, continue the procedure.
  - **Step 2**: Traverse all possible splitting rules. Each splitting rule is represented by a vector (j, t), which compares feature x_j and threshold t to split the dataset T_{cur} into two subsets T_1, T_2:

    $$T_1 = \{(\mathbf{x}, y) \in T_{\text{cur}} \text{ where } x_j \leq t\},$$
    $$T_2 = \{(\mathbf{x}, y) \in T_{\text{cur}} \text{ where } x_j > t\}.$$

    We will traverse the rules over all feature dimensions $j \in \{0, 1\}$ and thresholds $t \in \{x_j | (\mathbf{x}, y) \in T_{\text{cur}}\}$.
  - **Step 3**: Decide the best splitting rule. The best splitting rule (j^*, t^*) minimizes the weighted sum of Gini indices of T_1, T_2:

    $$(j^*, t^*) = \arg\min_{j,t} \frac{|T_1|\text{Gini}(T_1) + |T_2|\text{Gini}(T_2)}{|T_1| + |T_2|}$$

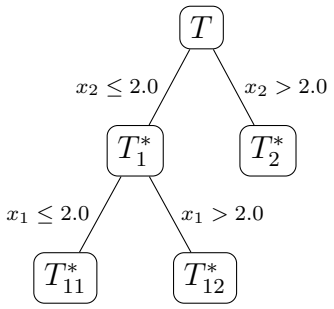
    where the $\text{Gini}(\cdot)$ is defined as:

    $$\text{Gini}(T_i) = 1 - \sum_{y=0}^{1} P(Y = y)^2,$$

    $$P(Y = y) = \frac{\#\{\text{data points with label } y \text{ in } T_i\}}{\#\{\text{data points in } T_i\}}.$$

  - **Step 4**: We split the dataset T_{cur} into two subsets T_1^*, T_2^* following the best splitting rule (j^*, t^*). Then we set current node as a *branch* node and create child nodes with the subsets T_1^*, T_2^* respectively. For each child node, start from **Step 1** again recursively.

If we run the above decision tree building procedure on dataset T and find the generated tree is shown below:



Please answer the questions:

1. (16 pts) Calculate the subsets $T_1^*, T_2^*, T_{11}^*, T_{12}^*$ using the given decision tree.

$T_1^* = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(1)}), (x^{(3)}, y^{(0)})\}$

$T_2^* = \{(x^{(4)}, y^{(0)}), (x^{(5)}, y^{(0)})\}$

$T_{11}^* = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(1)})\}$

$T_{12}^* = \{(x^{(3)}, y^{(0)})\}$

2. (12 pts) Calculate Gini(T_1^*) and Gini(T_2^*).

$$\text{Gini}(T_1^*) = 1 - \sum_{y=0}^{1} P(Y=y)^2$$
$$= 1 - \left[\left(\tfrac{1}{3}\right)^2 + \left(\tfrac{2}{3}\right)^2\right]$$
$$= 1 - \left[\tfrac{1}{9} + \tfrac{4}{9}\right] = \boxed{\tfrac{4}{9}}$$

$$\text{Gini}(T_2^*) = 1 - \sum_{y=0}^{1} P(Y=y)^2$$
$$= 1 - \left[(1)^2 + (0)^2\right]$$
$$= \boxed{0}$$

3. (12 pts) With the given tree, we can predict the class of a feature vector $\mathbf{x} = (x_1, x_2)$:

   - Start from the root node of the tree:
     - **Step 1**: If current node is a *branch* node, we evaluate conditions on branch edges with $\mathbf{x}$, choose the satisfied branch to go through, and repeat **Step 1**.
     - **Step 2**: If current node is a *leaf* node, the common class of the subset in the leaf node will be used as prediction.

   Please predict the following feature vectors using the given tree:

   (1) $\mathbf{x} = (2, 1)$,

   (2) $\mathbf{x} = (3, 1)$,

   (3) $\mathbf{x} = (3, 3)$.

$x_1 \rightarrow T_{11}^* \rightarrow y=1$

$x_2 \rightarrow T_{12}^* \rightarrow y=0$

$x_3 \rightarrow T_2^* \rightarrow y=0$

4. (**Bonus Question, 10 pts extra**) In this question, you need to implement the decision tree algorithm. Please download the Jupyter notebook `HW4_Decision_Tree.ipynb` and fill in the blanks. Note that since the same dataset T is used in the notebook, you can use the code to check if your previous answers are correct or not. Please attach your **code** and **results** in Gradescope submission.

Part I. Implement a decision tree algorithm and make predictions.

In [ ]:
```
import numpy as np
```

In [ ]:
```
class TreeNode:
    """ Node class in the decision tree. """
    def __init__(self, T):
        self.type = 'leaf' # Type of current node. Could be 'leaf' or 'branch' (at default: 'leaf').
        self.left = None   # Left branch of the tree (for leaf node, it is None).
        self.right = None  # Right branch of the tree (for leaf node, it is None).
        self.dataset = T   # Dataset of current node, which is a tuple (X, Y).
                           # X is the feature array and Y is the label vector.

    def set_as_leaf(self, common_class):
        """ Set current node as leaf node. """
        self.type = 'leaf'
        self.left = None
        self.right = None
        self.common_class = common_class

    def set_as_branch(self, left_node, right_node, split_rule):
        """ Set current node as branch node. """
        self.type = 'branch'
        self.left = left_node
        self.right = right_node
        # split_rule should be a tuple (j, t).
        #   When x_j <= t, it goes to left branch.
        #   When x_j > t, it goes to right branch.
        self.split_rule = split_rule
```

In [ ]:
```
# Prepare for dataset.
def get_dataset():
    X = np.array(
        [[1.0, 2.0],
         [2.0, 2.0],
         [3.0, 2.0],
         [2.0, 3.0],
         [1.0, 3.0]
        ])
    Y = np.array(
        [1,
         1,
         0,
         0,
         0])
    T = (X, Y) # The dataset T is a tuple of feature array X and label vector Y.
    return T

T = get_dataset()
```

In this part, you are required to implement the decision tree algorithm shown in the problem description of Q2 in HW4:

The **4 steps** are marked in comments of the following code. Please fill in the missing blanks (e.g. "...") in the TODOs:

In [ ]:
```
# Initialization.
root_node = TreeNode(T)
```

In [ ]:
```
# Procedure for current node.
def build_decision_tree_procedure(node_cur, depth=0):
    # Step 1. Check if all data points in T_cur are in the same class
    #         - If it is true, set current node as a *leaf node* to predict the common class in T_cur,
    #           and then terminate current procedure.
    #         - If it is false, continue the procedure.

    T_cur = node_cur.dataset
    X_cur, Y_cur = T_cur  # Get current feature array X_cur and label vector Y_cur.
    if (Y_cur == 1).all():
        print('    ' * depth + '+-> leaf node (predict 1).')
        print('    ' * depth + '      Gini: {:.3f}'.format(Gini(T_cur)))
        print('    ' * depth + '      samples: {}'.format(len(X_cur)))
        node_cur.set_as_leaf(1)
        return
    elif (Y_cur == 0).all():
        print('    ' * depth + '+-> leaf node (predict 0).')
        print('    ' * depth + '      Gini: {:.3f}'.format(Gini(T_cur)))
        print('    ' * depth + '      samples: {}'.format(len(X_cur)))
        node_cur.set_as_leaf(0)
        return
```

```
    # Step 2. Traverse all possible splitting rules.
    #         - We will traverse the rules over all feature dimensions j in {0, 1} and
    #           thresholds t in X_cur[:, j] (i.e. all x_j in current feature array X_cur).
    all_rules = []

    #### TODO 1 STARTS ###
    # Please traverse the rules over all feature dimensions j in {0, 1} and
    #   thresholds t in X_cur[:, j] (i.e. all x_j in current feature array X_cur),
    #   and save all rules in all_rules variable.
    # The all_rules variable should be a list of tuples such as [(0, 1.0), (0, 2.0), ... ]

    for j in (0,1):
        for t in X_cur[:,j]:
            all_rules.append((j,t))
    #### TODO 1 ENDS ###

    # print('All rules:', all_rules)   # Code for debugging.

    # Step 3. Decide the best splitting rule.
    best_rule = (_, _)
    best_weighted_sum = 1.0
    for (j, t) in all_rules:

        #### TODO 2 STARTS ###
        # For each splitting rule (j, t), we use it to split the dataset T_cur into T1 and T2.
        # Hint: You may refer to Step 4 to understand how to set inds1, X1, Y1, len_T1 and inds2, X2, Y2, len_T2.

        # - Create subset T1.
        inds1 = [x for x in range(len(T_cur[0])) if T_cur[0][x][j] <= t]                # Indices vector for those data points with
        X1 = [T_cur[0][i] for i in inds1]                   # Feature array with inds1 in X_cur.
        Y1 = [T_cur[1][i] for i in inds1]                   # Label vector with inds1 in Y_cur.
        T1 = (X1, Y1)               # Subset T1 contains feature array and label vector.
        len_T1 = len(X1)                # Size of subset T1.
        # - Create subset T2.
        inds2 = [x for x in range(len(T_cur[0])) if T_cur[0][x][j] > t]                 # Indices vector for those data points with
        X2 = [T_cur[0][i] for i in inds2]                   # Feature array with inds2 in X_cur.
        Y2 = [T_cur[1][i] for i in inds2]                   # Label vector with inds2 in Y_cur.
        T2 = (X2, Y2)               # Subset T2 contains feature array and label vector.
        len_T2 = len(X2)                # Size of subset T2.
        #### TODO 2 ENDS ###

        # Calculate weighted sum and try to find the best one.
        weighted_sum = (len_T1*Gini(T1) + len_T2*Gini(T2)) / (len_T1 + len_T2)
        # print('Rule:', (j, t), 'len_T1, len_T2:', len_T1, len_T2, 'weighted_sum:', weighted_sum)  # Code for debugging.
        if weighted_sum < best_weighted_sum:

            #### TODO 3 STARTS ####
            # Update the best rule and best weighted sum with current ones.

            best_rule = (j,t)
            best_weighted_sum = weighted_sum
            #### TODO 3 ENDS ####

    # Step 4. - We split the dataset T_cur into two subsets best_T1, best_T2 following
    #               the best splitting rule (best_j, best_t).
    #         - Then we set current node as a *branch* node and create child nodes with
    #               the subsets best_T1, best_T2 respectively.
    #         - For each child node, start from *Step 1* again recursively.

    best_j, best_t = best_rule
    # - Create subset best_T1 and corresponding child node.
    best_inds1 = X_cur[:,best_j] <= best_t
    best_X1 = X_cur[best_inds1]
    best_Y1 = Y_cur[best_inds1]
    best_T1 = (best_X1, best_Y1)
    node1 = TreeNode(best_T1)
    # - Create subset best_T2 and corresponding child node.
    best_inds2 = X_cur[:,best_j] > best_t
    best_X2 = X_cur[best_inds2]
    best_Y2 = Y_cur[best_inds2]
    best_T2 = (best_X2, best_Y2)
    node2 = TreeNode(best_T2)
    # - Set current node as branch node and create child nodes.
    node_cur.set_as_branch(left_node=node1, right_node=node2, split_rule=best_rule)
    print('    ' * depth + '+-> branch node')
    print('    ' * depth + '      Gini: {:.3f}'.format(Gini(T_cur)))
    print('    ' * depth + '      samples: {}'.format(len(X_cur)))
    # - For each child node, start from Step 1 again recursively.
    print('    ' * (depth + 1) + '|-> left branch: x_{} <= {} (with {} data point(s)).'.format(best_j, best_t, len(best_X1)))
    build_decision_tree_procedure(node1, depth+1) # Note: The depth is only used for logging.
    print('    ' * (depth + 1) + '|-> right branch: x_{} > {} (with {} data point(s)).'.format(best_j, best_t, len(best_X2)))
    build_decision_tree_procedure(node2, depth+1)

def Gini(Ti):
    """ Calculate the Gini index given dataset Ti. """
```

```
    Xi, Yi = Ti        # Get the feature array Xi and label vector Yi.
    if len(Yi) == 0:   # If the dataset Ti is empty, it simply returns 0.
        return 0

    #### TODO 4 STARTS ####
    # Implement the Gini index function.
    P_Y1 = len([x for x in Yi if x == 1]) / len(Yi)         # Estimate probability P(Y=1) in Yi
    P_Y0 = len([x for x in Yi if x == 0]) / len(Yi)         # Estimate probability P(Y=0) in Yi
    Gini_Ti = 1 - (P_Y1 ** 2) - (P_Y0 ** 2)      # Calculate Gini index: Gini_Ti = 1 - P(Y=1)^2 - P(Y=0)^2
    #### TODO 4 ENDS ####
    return Gini_Ti
```

After you finish the above code blank filling, you can use the following code to build the decision tree. The following code also shows the structure of the tree.

In [ ]:
```
# Build the decision tree.
build_decision_tree_procedure(root_node)

# If your code is correct, you should output:
#
# +-> branch node
#       Gini: 0.480
#       samples: 5
#     |-> left branch: x_1 <= 2.0 (with 3 data point(s)).
#     +-> branch node
#           Gini: 0.444
#           samples: 3
#     .....
#
# You can also use the sklearn results to validate your decision tree
# (the threshold could be slightly different but the structure of the tree should be the same).
```

```
+-> branch node
      Gini: 0.480
      samples: 5
    |-> left branch: x_1 <= 2.0 (with 3 data point(s)).
    +-> branch node
          Gini: 0.444
          samples: 3
        |-> left branch: x_0 <= 2.0 (with 2 data point(s)).
        +-> leaf node (predict 1).
              Gini: 0.000
              samples: 2
        |-> right branch: x_0 > 2.0 (with 1 data point(s)).
        +-> leaf node (predict 0).
              Gini: 0.000
              samples: 1
    |-> right branch: x_1 > 2.0 (with 2 data point(s)).
    +-> leaf node (predict 0).
          Gini: 0.000
          samples: 2
```

With the obtained decision tree, you can predict the class of new feature vectors:

In [ ]:
```
def decision_tree_predict(node_cur, x):
    if node_cur.type == 'leaf':
        return node_cur.common_class
    else:
        j, t = node_cur.split_rule
        if x[j] <= t:
            return decision_tree_predict(node_cur.left, x)
        else:
            return decision_tree_predict(node_cur.right, x)
```

In [ ]:
```
for x in [(2,1), (3,1), (3,3)]:
    y_pred = decision_tree_predict(root_node, x)
    print('Prediction of {} is {}'.format(x, y_pred))
```

```
Prediction of (2, 1) is 1
Prediction of (3, 1) is 0
Prediction of (3, 3) is 0
```

Part II. Use Scikit-learn to build the tree and make predictions.

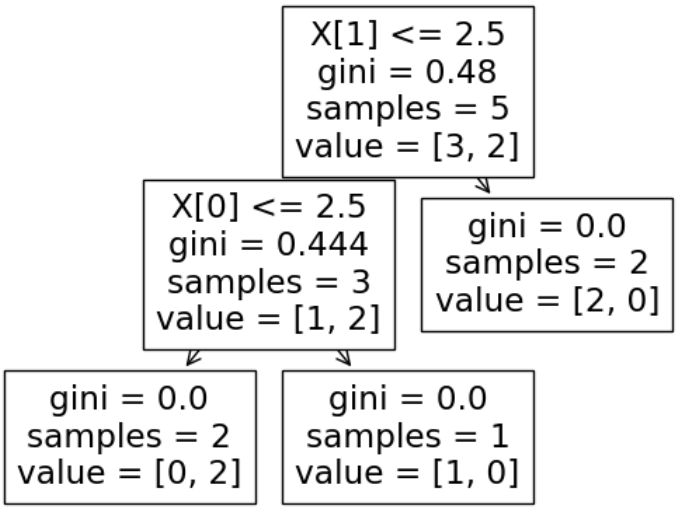
The following code uses Scikit-learn to build the decision tree. You can use it to check if your previous implementation is correct or not.

In [ ]:
```
# Ref: https://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart
from sklearn import tree
X, Y = T
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, Y)
```

The following code illustrates the obtained decision tree. It should have same structure and similar rules compared with the tree in your own implementation.

In [ ]:
```
# Plotting the tree.
tree.plot_tree(clf)
```

Out[ ]:
```
[Text(0.6, 0.8333333333333334, 'X[1] <= 2.5\ngini = 0.48\nsamples = 5\nvalue = [3, 2]'),
 Text(0.4, 0.5, 'X[0] <= 2.5\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.2, 0.16666666666666666, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.6, 0.16666666666666666, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.8, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]')]
```



The following code makes the predictions using the obtained decision tree. It should have identical results as the ones for your own implementaion.

In [ ]:
```
# Predict the class.
for x in [(2,1), (3,1), (3,3)]:
    y_pred = clf.predict(np.array([x]))[0]
    print('Prediction of {} is {}'.format(x, y_pred))
```

```
Prediction of (2, 1) is 1
Prediction of (3, 1) is 0
Prediction of (3, 3) is 0
```

3 (20 points) Bagging and Boosting

Assume we obtain T linear classifiers $\{h_t, t = 1, ..., T\}$ where each classifier $h : \mathbb{R}^2 \to \{+1, -1\}$ predicts the class $\hat{y} \in \{+1, -1\}$ with given feature vector $\mathbf{x} = (x_1, x_2)$ as follows:

$$\hat{y} = h(\mathbf{x}) = \text{sign}(w_1 x_1 + w_2 x_2 + b) \qquad \text{where} \qquad \text{sign}(a) = \begin{cases} +1 & \text{if } a \geq 0, \\ -1 & \text{if } a < 0. \end{cases}$$

where $w_1, w_2, b \in \mathbb{R}$ are the parameters.

- In a bagging model H_{bagging} of the T linear classifiers, we calculate the average prediction using classifiers $\{h_t\}$, and then use it to predict the class $\hat{y}_{\text{bagging}}$:

$$\hat{y}_{\text{bagging}} = H_{\text{bagging}}(\mathbf{x}) = \text{sign}\Big(\frac{1}{T}\sum_{t=1}^{T} h_t(\mathbf{x})\Big)$$

- In a boosting model H_{boosting} of the T linear classifiers, we calculate the weighted sum of predictions using classifiers $\{h_t\}$, and then use it to predict the class $\hat{y}_{\text{boosting}}$:

$$\hat{y}_{\text{boosting}} = H_{\text{boosting}}(\mathbf{x}) = \text{sign}\Big(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\Big)$$

  where $\{\alpha_t, t = 1, ..., T\}$ are the weight coefficients.

In this problem, suppose we have 3 linear classifiers (i.e. $T = 3$):

$$h_1(\mathbf{x}) = \text{sign}(x_1 + x_2 + 1), \quad h_2(\mathbf{x}) = \text{sign}(x_1 - x_2), \quad h_3(\mathbf{x}) = \text{sign}(x_1 - 2x_2 + 1).$$

Please answer the questions below:

1. (10 pts) Please calculate the $\hat{y}_{\text{bagging}}$ of feature vector $\mathbf{x} = (1, 2)$ using bagging on these three classifiers.

$$\hat{y}_{\text{bagging}} = \text{sign}\left(\frac{1}{3}\sum_{t=1}^{3} h_t(x)\right)$$

$$= \text{sign}\left(\frac{1}{3}(1 + (-1) + (-1)\right)$$

$$= \boxed{-1}$$

2. (10 pts) Please calculate the $\hat{y}_{\text{boosting}}$ of feature vector $\mathbf{x} = (1, 2)$ using boosting on these three classifiers. The weight coefficients are $\alpha_1 = 0.8,\ \alpha_2 = 0.2,\ \alpha_3 = 0.3$.

$$\hat{y}_{\text{boosting}} = \text{sign}\left(\sum_{t=1}^{3} \alpha_t h_t(x)\right)$$

$$= \text{sign}\left(0.8(4) + 0.2(-1) + 0.3(-1)\right)$$

$$= \boxed{+1}$$