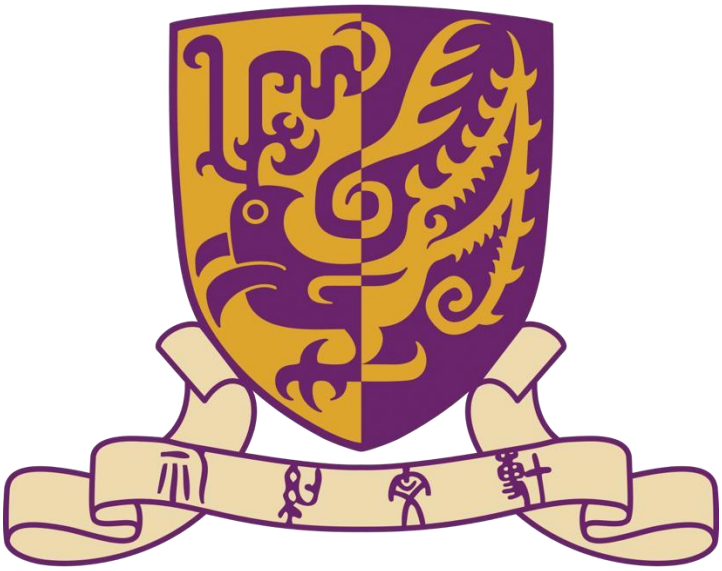

The Chinese University of Hong Kong, Shenzhen

Report for Wine Qualification Based on Classification and Regression Tree

Author: Student Number:

Zhen Tong 120090694

December 6th, 2021

Contents

Abstract

The key problem of this project is to build a CART tree for wine quality estimation, which is to classify which quality class the testing wine belongs to. The result of the classification is at most 87%. This report also analyzed whether the tree size will affect the accuracy and what determines the best pre-pruning strategy.

1 Introduction

The target problem is to train a classification and regression tree by the training data and automatically classify the testing wine with unknown quality from 3 to 8. There are two complex problems. The first problem is that classifying quality from 3 to 8 is a multi-classification problem, which means the tree will be much more giant than the traditional binary classification. The second problem is how to effectively pruning to avoid overfitting. The code to implement the decision tree without third-party CART tree packages. In the end, we will compare the accuracy of big trees and small trees and compare the correctness of the tree without pruning and the tree after pruning.

2 Theory

The CART algorithm uses the Gini coefficient to select features, representing the model's impurity. The smaller the Gini coefficient, the lower the impurity and the better the feature is, which is the opposite of information gain. The purity of dataset D can be measured by the Gini value

$$\text{Gini}(D) = \sum_{i-1}^{n} p(x_i) \times p\big(1 - p(x_i)\big) = 1 - \sum_{i-1}^{n} p(x_i)^2 \tag{1}$$

In the formula, $p(x_i)$ is the probability of occurrence of classification x_i, and n is the number of classification. Gini(D) reflects the probability that two samples randomly selected from dataset D have inconsistent category markers. Therefore, the smaller Gini(D) is, the higher the purity of dataset D is.

Denote the sample number as | D |. Sample D is divided into two-part, D_1 and D_2, by particular value a of feature category A. Therefore, the CART algorithm builds a binary tree.

$$D_1 = (\text{x, y}) \in \text{D} | \text{A(x)} = a \tag{2}$$

$$\text{D}_2 = D - D_1 \tag{3}$$

Under the condition of A = a, the Gini coefficient of sample set D is defined as

$$\text{Gini(D|A = a)} = \frac{D_1}{D} Gini(D_1) + \frac{D_2}{D} Gini(D_2) \tag{4}$$

When the CART tree is too specific, it will over-fit the noise data, so pruning should be used to solve the problem. Pruning includes pre-pruning and post-pruning. Pre-pruning means setting rules for which nodes can be cut while constructing the tree. Post-pruning

is the process of examining which sub-trees can be pruned in a complete decision tree. In this project, we used the cost complexity pruning method.

For each non-leaf node in the classification and regression tree, its error ratio gain value α is

$$\alpha = \frac{C(t) - C(T_t)}{|T_t| - 1} \quad (5)$$

$$C(t) = r(t) \times p(t) \quad (6)$$

This formula denotes the number of leaf nodes contained in the sub-tree as$|T_t|$. The error cost $C(t)$ of the single-node tree, whose node is pruned. $r(t)$ represents the error rate of node t. $p(t)$ represents the ratio of node number on T to the number of all the nodes. The error cost of the sub-tree with the root t is $C(T_t)$. Find the non-leaf node with the smallest value of α and make its left and right child nodes NULL. When the α value of many non-leaf nodes reaches the minimum simultaneously, take the maximum value of $|T_t|$ for pruning.

3 Method

3.1 Introduction to the Data Set

The data is from the CSC1002 course midterm project. The data set contains 1120 wines, each with 11 features, including total sulfur dioxide, pH, density, and alcohol. The given quality is an integer from 3 to 8. The number of each quality is almost the same.

3.2 Find Partition Value

Because the feature value of a wine is continuous, there are at most 1120 different numbers for one feature $f_k \in F$ among all the wines. First, we need to sort the 1120 value from small to large. Then we iterate the sorted 1120 values and get the average of two values each time. If there are K categories of features, the time complexity is $O(K \times N)$. K = 11, N = $|D_i|$ in this project.

$$v_{partition} = \frac{v_i + v_{i+1}}{2}$$

We iterated all the partition values and divided the data set into two parts according to the partition value. Among all the data x_i whose feature value is bigger than the partition value VJ we calculate $p_j(x_i)$, which is the probability of occurrence of classification x_i. Similarly, for all the data y_i whose feature value is smaller than the partition value VJ, we calculate $q_j(y_i)$. Substitute the $Gini(D)$ in equation 4 and minimize the Gini coefficient of the sample set Dunder partition value v_j

$$v_k = arg\,min\,Gini(D|v_j \in V, f_k) = \frac{D_1}{D}1 - \sum_{i-1}^{n} p_j(x_i)^2 + \frac{D_2}{D} q_j(y_i) \quad (7)$$

After iteration of all possible partition values, find the v_k that can divide the data set into two groups with the lowest Gini coefficient, indicating the highest purity. Discrete attributes cannot be reused after being used once, but continuous attributes can also be used as partition attributes of their descendant nodes.

3.3 Find the Division Feature

We only consider estimating the data set into two different kinds in a traditional classification problem. The challenging part of this project is to use the CART binary decision tree to classify six categories (quality ∈ {3,4,5,6,7,8}).

The solution of this project is each time assume there is a standard $s_l \in \{3.5,4.5,5.5,6.5,7.5\}$ that divides the quality into two-part parts, D_1 for those data whose quality is below s_l, D_2 for the data whose quality is higher than s_l. Each time we needed to create a new node for division, we tried a standardss_l, so we never tried it before, and calculated the minimum Gini coefficient.

$$Gini(D|s_l) = \min Gini(D|v_j \in V, f_k \in F, s_l)$$

$$s(D_i) = arg \min Gini\left(D\middle|v_j \in V, f_k \in F, s_l\right), s_l \in S_{un\ used}$$

In this way, the CART tree will have at least six leaves, and each indicates a category.

3.4 Building Tree and Pre-pruning

The building process is a depth first search recursion. We set the threshold to stop the recursion process as a pre-pruning. We set the $|D_{set}|$ as the smallest number of data a node allowed to have. The pseudo code is as below:

```
Divide Node (D_i):
  If (max min Gini(D_i|f_k ∈ F_i) > threshold) and F_i ≠ ∅ and |D_i| > |D_set|{
//Find the best division condition ( f_k, s_l, v_j) for data D_i by min Gini(D_i| f_k, s_l, v_j)
    For f_k ∈ F_i {
      For s_l ∈ S{
        For v_j ∈ V{
          Calculate and compare Gini(D_i|f_k, s_l, v_j), find minimum
        }
      }
    }
  Divide D_i into D_i+1 and D_i+2, remove f_k from F_i+1 and F_i+2
  Divide Node (D_i+1)
  Divide Node (D_i+2)
  }
  Else{
```

```
        Node[D_i][quality] = the majority of the data's quality
        return
    }
```

We record the accuracy of the raw tree without post-pruning.

3.5 Post-pruning

It is not enough to directly build a full tree. We need to do pruning to avoid over-fitting. The non-leaf nodes are investigated from the bottom up. If changing the sub-tree corresponding to this node into a leaf node improves accuracy, the sub-tree should be replaced with a leaf node. The pseudo-code of post-pruning is as below:

```
Post Pruning(Tree):
//Find the minimized error ratio gain value α
    For every non-leaf node of the Tree{
        //Check the sub-tree of the non-leaf node into a leaf
        Calculate α and find min α = (C(t)−C(T_t)) / (|T_t|−1) // Equation (5)
    }
    Change the sub-tree of the non-leaf node into a leaf
```

The time complexity of the method is $O(N)$.

4 Result and Discussion

A part of output estimated quality is printed as follows:

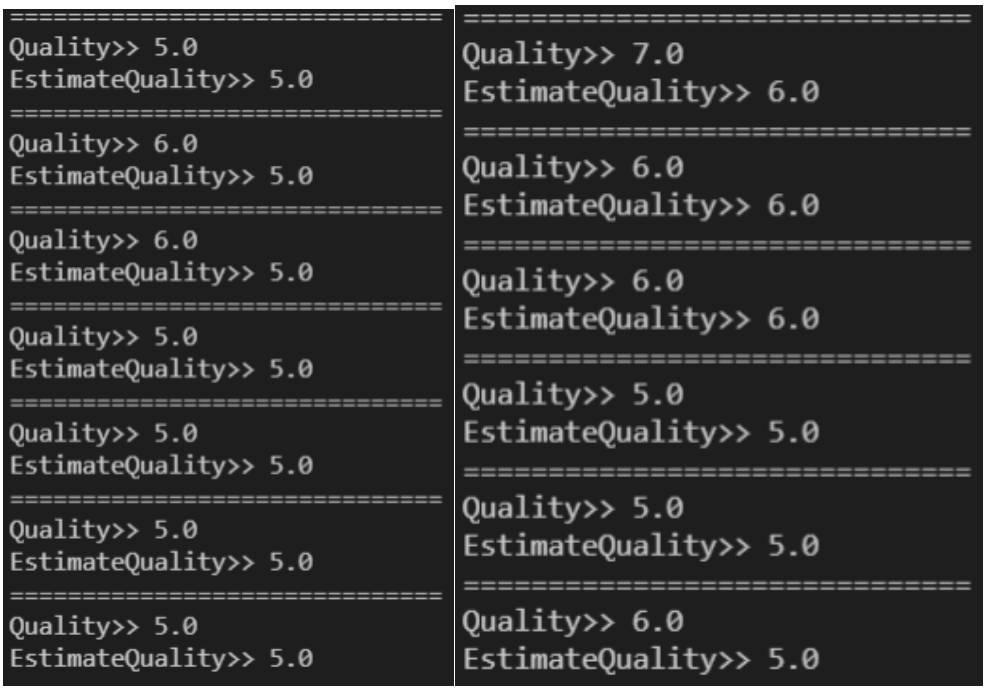

Figure 1 part of the sample output

4.1 Changing the Minimum Data Size

We randomly choose 500 data in the whole 1120 training data as a smaller new training $TrainSet = \{Train_1, Train_2, Train_3\}$, $|T_i|$ =500. Similarly, we selected 100 data from in the whole 481 testing data, $estSet = \{Test_1, Test_2, Test_3\}$. After selecting training data and test data, we set the minimum volume of branch nodes $|D_{set}|$ from 0 to 40, and the Gini threshold equals 0.05 unchanged. The relationship is shown in the following table: Number of experiments.

Train and Test No.1					
The least node $\|D_{set}\|$	0	10	20	30	40
Total node number	91	83	63	38	31
Leaf node number	50	39	34	25	11
Accuracy	83%	84%	85 %	82%	83%

Train and Test No.2					
The least node $\|D_{set}\|$	0	10	20	30	40
Total node number	105	73	64	38	31
Leaf node number	64	35	35	21	10
Accuracy	88%	88%	88%	86%	86%

Train and Test No.3					
The least node $\|D_{set}\|$	0	10	20	30	40
Total node number	63	38	33	25	15
Leaf node number	23	14	12	9	5
Accuracy	67%	77 %	87%	82%	72%

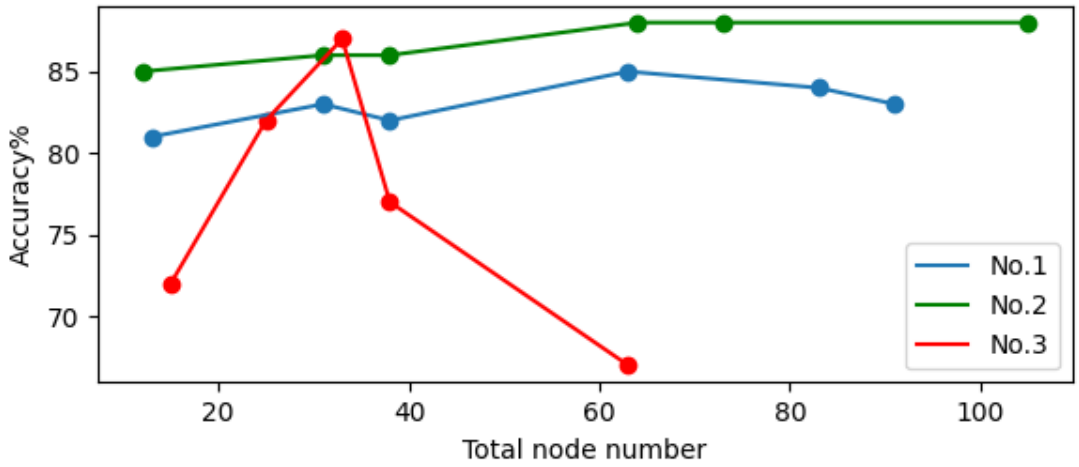

Figure 2 Relationship between the accuracy and tree size

From these experimental results, we can draw the following conclusions:

(1) There is no strong correlation between tree size and accuracy. For some data, reducing the tree size will improve the classification accuracy. However, for other data, reducing the size of the tree reduces the accuracy of the classification. A more accurate decision tree model can be obtained by increasing training data as much as possible.

(2) Judging from the results, when the set minimum volume of branch nodes $|D_{set}|$ is restricted from 0 to 20, the size of all trees is reduced to a certain extent, while the accuracy of a judgment is not changed. Simple trees are better than complex ones at the same rate of accuracy. Therefore, pruning the decision tree is necessary.

(3)To sum up the experimental results, the pruning method is to change some branch nodes, which does not improve leaf nodes' accuracy.

4.2 Changing the Gini Threshold of Pre-pruning

We set the minimum number of each node equals to 10 and changed the Gini Threshold from 0.005 to 0.1

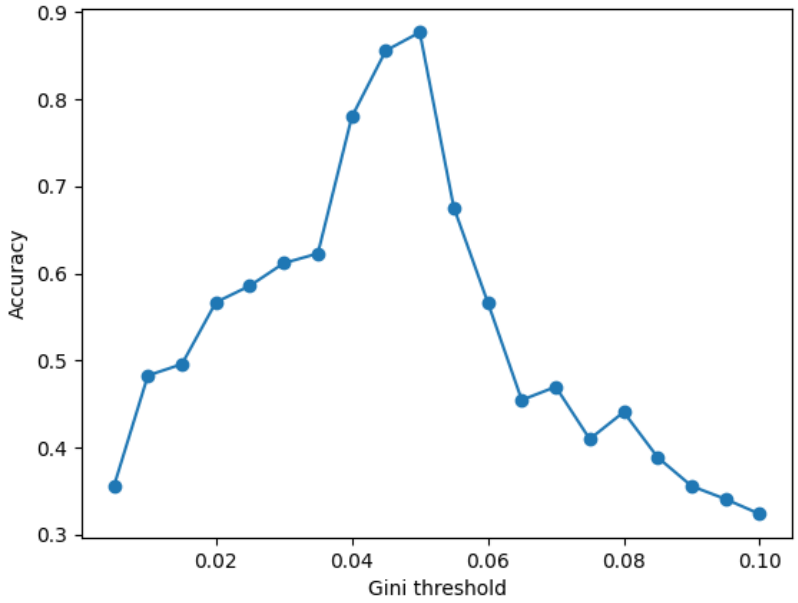

Figure 3 Relationship between accuracy and Gini treshold

From the graph, when the Gini threshold is 0.055, the accuracy reached a peak (87.7%). A possible reason for it is 0.055 can describe how the purity of a data set in a sub-tree should be.

(1) When the Gini threshold is too small, the CART tree is over-fitted because the recursion of the building tree only stops when the purity is very high. Most of the leaf node data amount is near to 10 because it is the minimum number of data a node should have as we set.

(2) The CART classification accuracy drops rapidly when the Gini threshold is bigger than 0.055. The classification process does not work when the Gini coefficient is big, because the majority quality of a leaf node is close to the number of the other quality in that node. The classification process stopped at a very vague stage.

(3) It is reasonable that the positive Gini threshold deviation can largely decrease the accuracy while the negative Gini threshold deviation won't, because the bad consequence of over-fitting is not as terrible as incomplete classification.

(4) The best Gini threshold of a CART is always close to 0.055 because the size and value of data will not change the purity requirement of the CART model as long as the train data and test data are from a random sample.

5 Conclusion

For the feature of continuous value, we successfully find the best partition to distinguish data by sorting, iterating the average, and finding minimum Gini. To divide the data set into multiple categories, we also used the binary division method and checked the best

partition value for categories value. The best classification accuracy is up to 88% with optimized pruning. We found out that the tree size and the accuracy are not always correlated. When the minimum volume of branch nodes $|D_{set}|$ changed from 0 to 20, the accuracy increased, and the time complexity decreased. For the Gini threshold of pre-pruning, the perfect value is around 0.055. We also found out that the lower Gini threshold deviation will better classify the data.

6 Reference

6.1 Python 3 Code

```
import csv
from os import close
import numpy as np

f = open('train.csv', 'r', encoding="utf-8")
reader = csv.reader(f)
L = list()
L = [row for row in reader]
L = np.array(L)

print(len(L))
pure_data = L[1:, :].astype(np.float32)


class Node:
    def __init__(self, data: np.ndarray):
        self.data = data
        # define left and right tree
        self.left = None
        self.right = None
        self.unused_ftNum = None  # unused feature
        self.div = None
        self.div_ftNum = None

        self.quality = None

    def binaryDataDivide(self):
        if CART_chooseBestFeatureToSplit(self) == None:
            return
        [MiniGini, div, quality, ftNum] = CART_chooseBestFeatureToSplit(self)

        if div == -1:
            return
        # Get segmentation information
        self.div = div  # Example Set the node split value
        self.div_ftNum = ftNum  # Set the node segmentation feature
        left_data = np.zeros((1, 12))
        right_data = np.zeros((1, 12))
        for wine in self.data:
            if(wine[ftNum] <= div):
                left_data = np.r_[left_data, [wine]]
            else:
                right_data = np.r_[right_data, [wine]]
        left_data = left_data[1:, :]
        right_data = right_data[1:, :]

        leftNode = Node(left_data)
        rightNode = Node(right_data)
        newFtNum = []
```

```
49.         for i in self.unused_ftNum:
50.             newFtNum.append(i)
51.         newFtNum.remove(ftNum)
52.         leftNode.unused_ftNum = newFtNum
53.         rightNode.unused_ftNum = newFtNum
54.
55.         self.left = leftNode
56.         self.right = rightNode
57.         # print("Left>>",self.left.data.shape,"Right>>",self.right.data.shape)
58.
59.         # Generates two child nodes
60.
61.     def has_feature(self):
62.         if(len(self.unused_ftNum) == 0):
63.             # print("hasNo_feature")
64.             return False
65.         else:
66.             # print("has_feature")
67.             return True
68.
69.     def is_higher_threshold(self, threshold):
70.         if CART_chooseBestFeatureToSplit(self) == None:
71.             return False
72.         gini = CART_chooseBestFeatureToSplit(self)[0]
73.         if gini > threshold:
74.             print("lower than threshold")
75.             return False
76.         #print("higher than threshold")
77.         return True
78.
79.     def getMajority(self):
80.         dataList = self.data[:, -1].tolist()
81.         maxlabel = max(dataList, key=dataList.count)
82.         print('maxlabel>>>', maxlabel)
83.         self.quality = maxlabel
84.         return maxlabel
85.
86.     def hasChild(self):
87.         if(self.left == None and self.right == None):
88.             return False
89.         return True
90.
91.
92. class Tree:
93.     def __init__(self, data: np.ndarray, threshold):
94.         rootNode = Node(data)
95.         rootNode.unused_ftNum = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
96.
97.         self.data = data
98.         self.root = rootNode
99.         self.threshold = threshold
100.
101.
102. def get_Min_gini(dataset: np.ndarray, ftNum, quality):
103.
104.     quality = float(quality)
105.     div_list = []
106.     # This step is more accurate
107.     # colm_data = sorted(dataset[:,ftNum].tolist())      #Get a column of so
    rted data
108.     # print(colm_data)
109.     # for i in range(len(colm_data)-
    2):                  #Generate a set of median split
110.     #     j = (colm_data[i]+colm_data[i+1])/2
111.     #     div_list.append(j)
```

```
112.
113.     # This step is more fast
114.     arrMin = min(dataset[:, ftNum])
115.     arrMax = max(dataset[:, ftNum])
116.     # print(arrMin,arrMax)
117.     div_list = np.arange(arrMin, arrMax, 10)
118.     # print(div_list)
119.     #div_list = div_list[1:-2]
120.     gini = 10000
121.     target_div = -1
122.     for i in range(len(div_list)):  # Calculate gini coefficients by dividin
    g them in a div list
123.         div = div_list[i]
124.
125.         D1 = 0
126.         D2 = 0
127.         d1_count = 0
128.         d2_count = 0
129. # Div divides the data into two parts
130.         for wine in dataset:
131.             if wine[ftNum] < div:
132.                 D1 += 1
133.                 if wine[-1] > quality:
134.                     d1_count += 1
135.             else:
136.                 D2 += 1
137.                 if wine[-1] > quality:
138.                     d2_count += 1
139.         if D1 == 0:
140.             p1 = 0
141.         else:
142.             p1 = d1_count/D1
143.         if D2 == 0:
144.             p2 = 0
145.         else:
146.             p2 = d2_count/D2
147.
148.         temp_gini = (D1/(D1+D2))*2*p1*(1-p1) + (D2/(D1+D2))*2*p2*(1-p2)
149.         if temp_gini < gini:
150.             # Check whether temp_gini is minimum and greater than the thresh
    old value
151.             gini = temp_gini
152.             target_div = div
153.     # Print the value of the minimum gini, the segmentation condition,
154.     #print("the Min Gini>>",gini,"target div>>",target_div)
155.     return [gini, target_div, quality, ftNum]
156.
157.
158. def CART_chooseBestFeatureToSplit(node: Node):
159.
160.     GiniList = []
161.     quality_list = [3.5, 4.5, 5.5, 6.5, 7.5]
162.     for quality in quality_list:
163.         # Take quality as the segmentation condition
164.         for ftNum in node.unused_ftNum:
165.             GiniList.append(get_Min_gini(node.data, ftNum, quality))
166.     # print(GiniList)
167.     gini = 10000
168.     index = -1
169.     for i in range(len(GiniList)):
170.         if GiniList[i][0] < gini:
171.             gini = GiniList[i][0]
172.             index = i
173.     if index == -1:
174.         return
175.     else:
```

```
176.         # print("MiniGini>>",GiniList[index][0],"div>>",GiniList[index][1],
177.         # "quality>>",GiniList[index][2],"FeatureNumber>>",GiniList[index][3
    ])
178.         return GiniList[index]  # Returns the smallest Gini split group
179.
180.
181. def divideNode(node: Node):
182.     if node.has_feature() and node.is_higher_threshold(0.1) and node.data.sh
    ape[0] > 2:
183.         node.binaryDataDivide()
184.         print("FeatureNumber>>", node.div_ftNum, "Div_Value>>", node.div)
185.         divideNode(node.right)
186.         divideNode(node.left)
187.     else:
188.         node.getMajority()
189.         return
190.
191.
192. print(min(L[:, -1]), ",", max(L[1:, -1]))  # show the max and min
193.
194. tree = Tree(pure_data, 0.01)
195. divideNode(tree.root)
196.
197. # Build input detection
198. f_test = open('test.csv', 'r', encoding="utf-8")
199. reader = csv.reader(f_test)
200. test_ls = [row for row in reader]
201. test_ls = np.array(test_ls)
202. # L the raw data with the first row
203. print(len(test_ls))
204. pure_test_data = test_ls[1:, :].astype(np.float32)
205. print(pure_test_data.shape)
206. # pure raw data without the first row
207.
208.
209. def printData(data: np.ndarray):  # Input a row of wine vectors, output a pr
    int
210.     print("================================")
211.     #print("fixed acidity>>",data[0])
212.     #print("volatile acidity>>",data[1])
213.     #print(" citric acid",data[2])
214.     print("Quality>>", data[-1])
215.     return
216.
217.
218. def testifyTREE(tree: Tree, data: np.ndarray):
219.     true_count = 0
220.     for wine in data:
221.         print("============================")
222.         print("Quality>>", wine[-1])
223.         estimate_quality = evl_quality(tree, wine)
224.         print("EstimateQuality>>", estimate_quality)
225.         if wine[-1] == estimate_quality:
226.             true_count += 1
227.
228.     print("correctness>>", true_count, "/",
229.           data.shape[0], "=", true_count/data.shape[0])
230.     return true_count/data.shape[0]
231.
232.
233. def evl_quality(tree: Tree, data):
234.     tempNode = tree.root
235.     while(tempNode.right != None and tempNode.left != None):
236.         #print("Left and Right>>", tempNode.left.data.shape,tempNode.right.d
    ata.shape)
```

```
237.         ftNum = tempNode.div_ftNum
238.         divVal = tempNode.div
239.         if data[ftNum] <= divVal:
240.             tempNode = tempNode.left
241.         else:
242.             tempNode = tempNode.right
243.     tempNode.getMajority()
244.     # print(tempNode.quality)
245.     return tempNode.quality
246.
247.
248. testifyTREE(tree, pure_test_data)
```