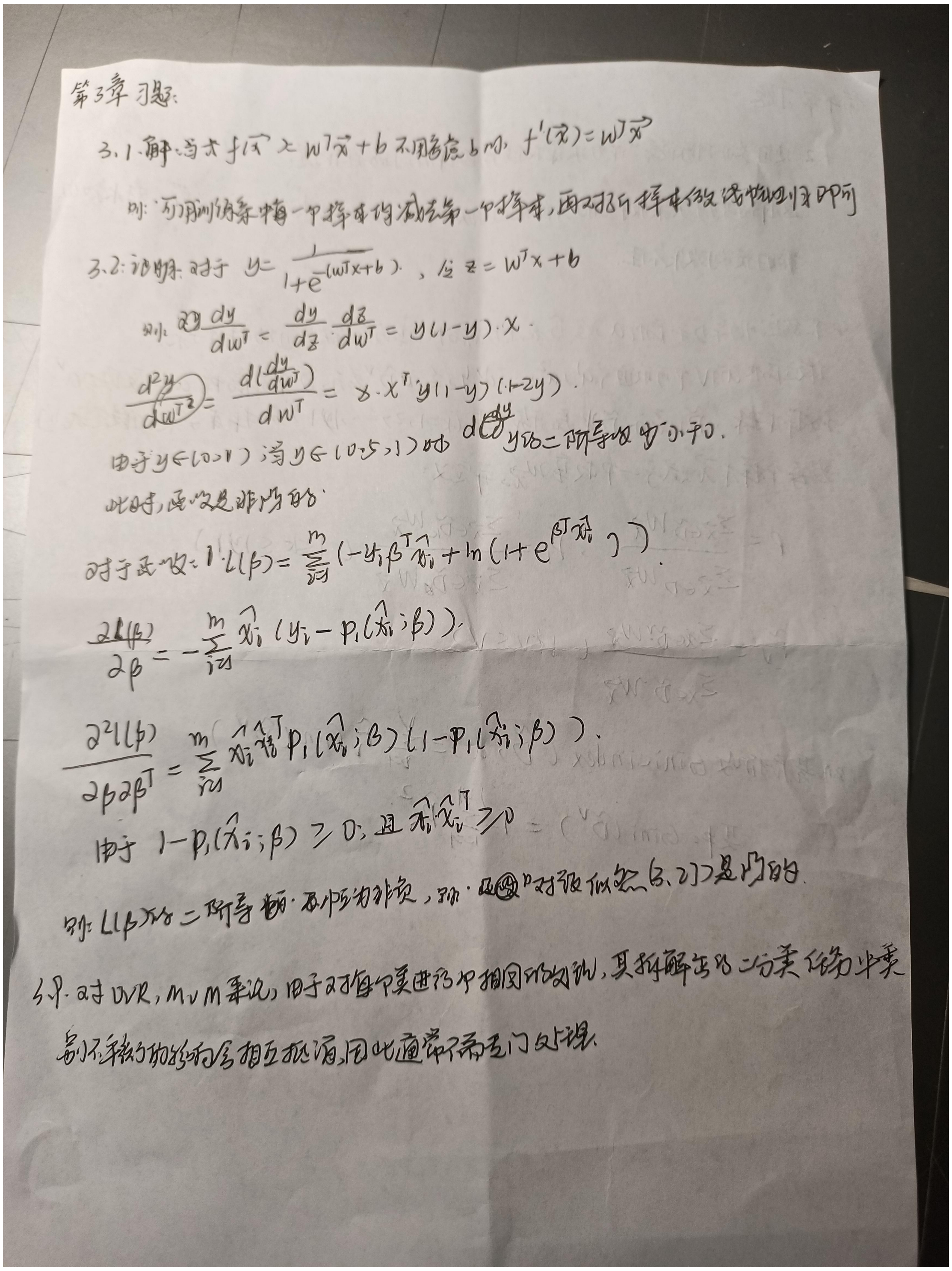


第3章 习题:

3.1. 解: 当式 $f(\vec{x}) = w^T\vec{x} + b$ 不用考虑 b 时, $f'(\vec{x}) = w^T\vec{x}$

则: 可用训练集中每一个样本均减去第一个样本, 再对新样本做线性回归即可

3.2: 证明: 对于 $y = \frac{1}{1+e^{-(w^Tx+b)}}$, 令 $z = w^Tx + b$

则: $\frac{dy}{dw^T} = \frac{dy}{dz}\frac{dz}{dw^T} = y(1-y)\cdot x$.

$\frac{d^2y}{dw^{T2}} = \frac{d(\frac{dy}{dw^T})}{dw^T} = x\cdot x^T y(1-y)(1-2y)$.

由于 $y\in(0,1)$ 当 $y\in(0.5,1)$ 时 $d(^{dy}$ y的二阶导数小于0.

此时, 函数是非凸的.

对于函数: $l(\beta) = \sum_{i=1}^{m}(-y_i\beta^T\hat{x}_i + \ln(1+e^{\beta^T\hat{x}_i}))$.

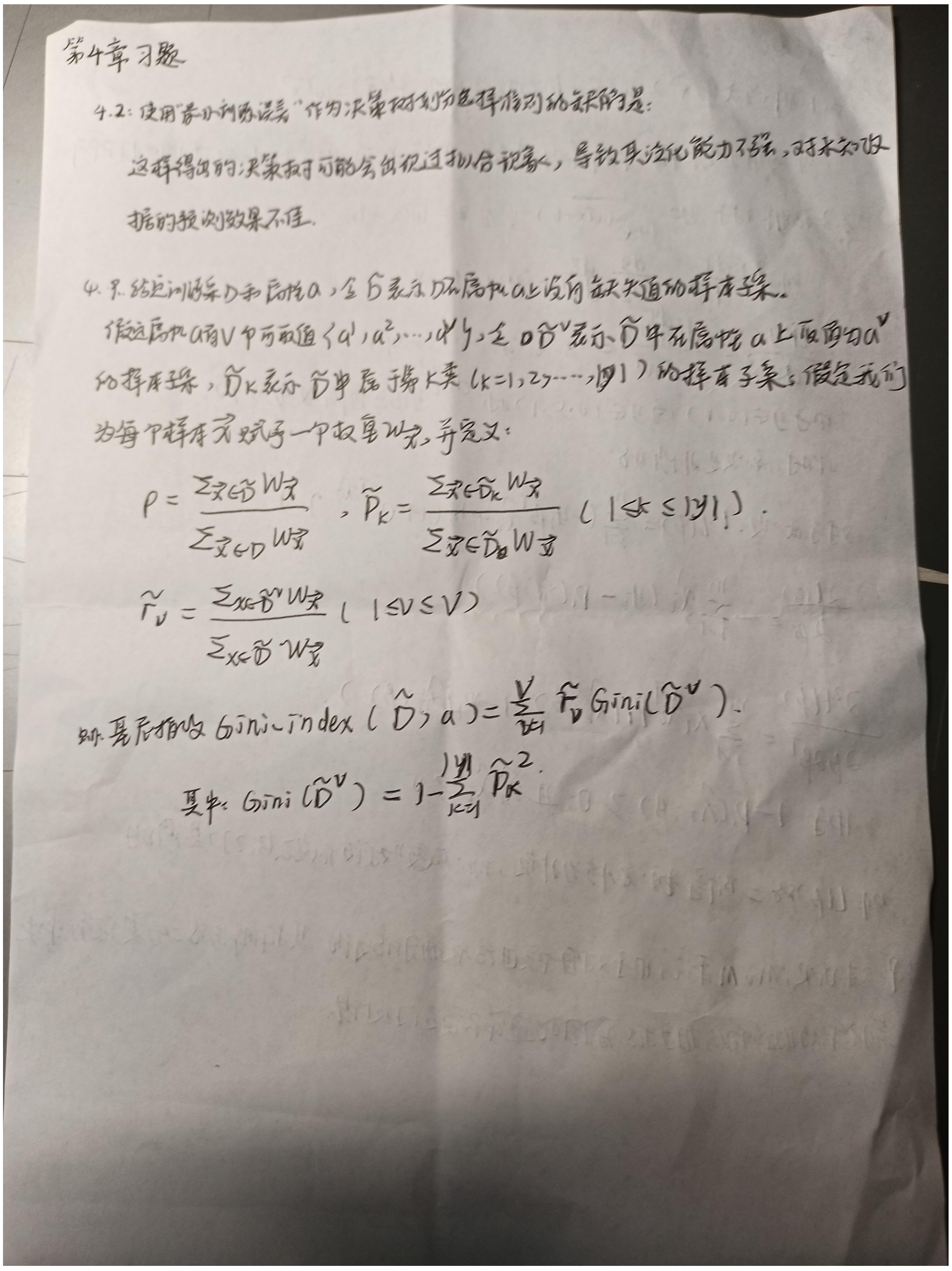
$\frac{\partial l(\beta)}{\partial\beta} = -\sum_{i=1}^{m}\hat{x}_i(y_i - p_1(\hat{x}_i;\beta))$.

$\frac{\partial^2 l(\beta)}{\partial\beta\partial\beta^T} = \sum_{i=1}^{m}\hat{x}_i\hat{x}_i^T p_1(\hat{x}_i;\beta)(1-p_1(\hat{x}_i;\beta))$.

由于 $1-p_1(\hat{x}_i;\beta) \geq 0$, 且 $\hat{x}_i\hat{x}_i^T \geq 0$

则: $l(\beta)$ 的二阶导数恒为非负, 所以 对数似然 (3.27) 是凸的.

3.8. 对 OvR, MvM 来说, 由于对每个类进行了相同的处理, 其拆解出的二分类任务中类别不平衡的影响会相互抵消, 因此通常不需专门处理.



第4章 习题

4.2: 使用"最小训练误差"作为决策树划分选择准则的缺陷是:

这样得出的决策树可能会出现过拟合现象，导致其泛化能力不强，对未知数据的预测效果不佳.

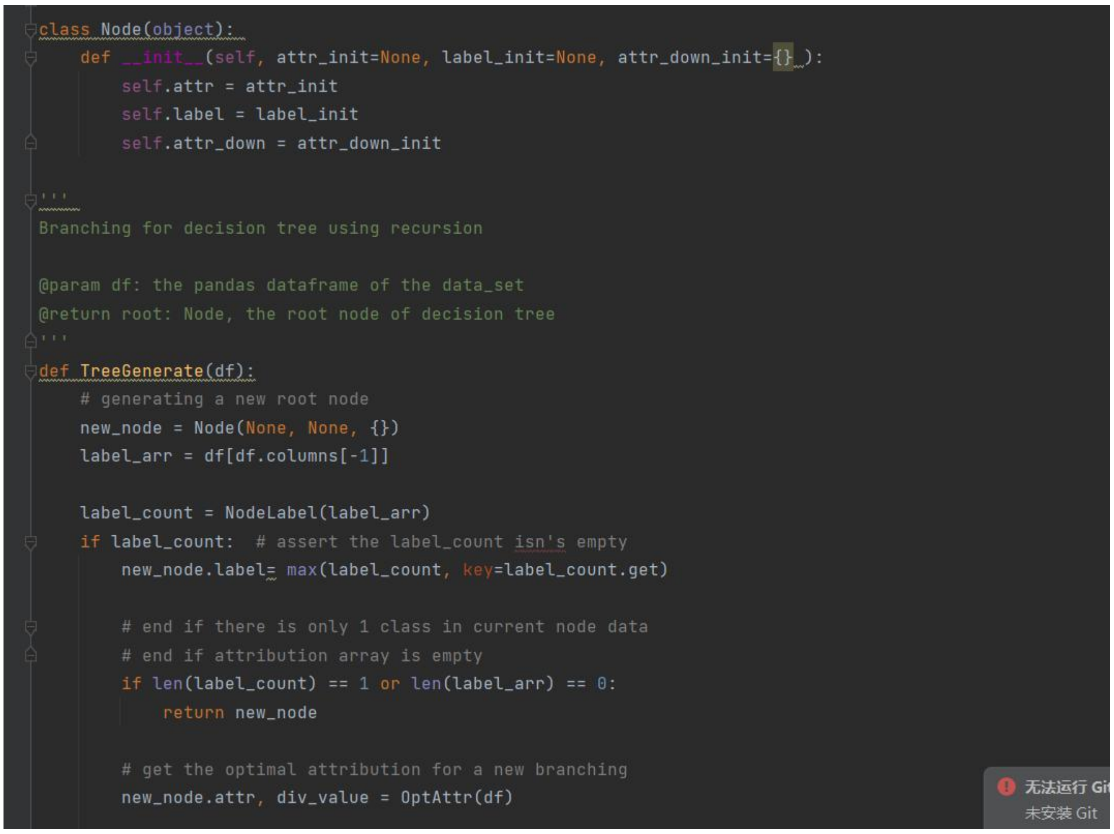
4.9. 给定训练集D和属性a，令$\tilde{D}$表示D在属性a上没有缺失值的样本子集。
假设属性a有V个可取值$\{a^1, a^2, \ldots, a^V\}$，令$\tilde{D}^v$表示$\tilde{D}$中在属性a上取值为$a^v$的样本子集，$\tilde{D}_k$表示$\tilde{D}$中属于第k类$(k=1,2,\ldots,|y|)$的样本子集；假定我们为每个样本$\vec{x}$赋予一个权重$w_{\vec{x}}$，并定义:

$$\rho = \frac{\sum_{\vec{x}\in\tilde{D}} w_{\vec{x}}}{\sum_{\vec{x}\in D} w_{\vec{x}}}, \quad \tilde{p}_k = \frac{\sum_{\vec{x}\in\tilde{D}_k} w_{\vec{x}}}{\sum_{\vec{x}\in\tilde{D}} w_{\vec{x}}} \ (1 \le k \le |y|).$$

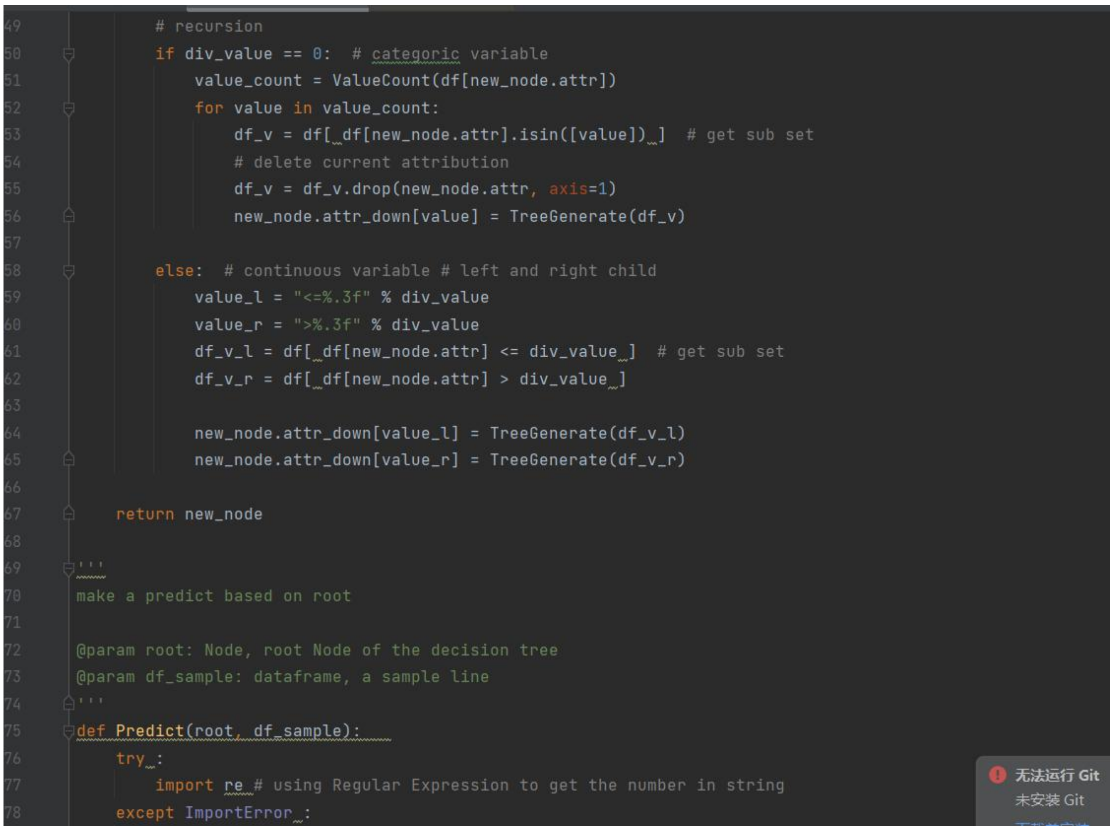
$$\tilde{r}_v = \frac{\sum_{x\in\tilde{D}^v} w_{\vec{x}}}{\sum_{x\in\tilde{D}} w_{\vec{x}}} \ (1 \le v \le V)$$

则基尼指数 $\text{Gini\_index}(\tilde{D}, a) = \sum_{v=1}^{V} \tilde{r}_v \, \text{Gini}(\tilde{D}^v)$.

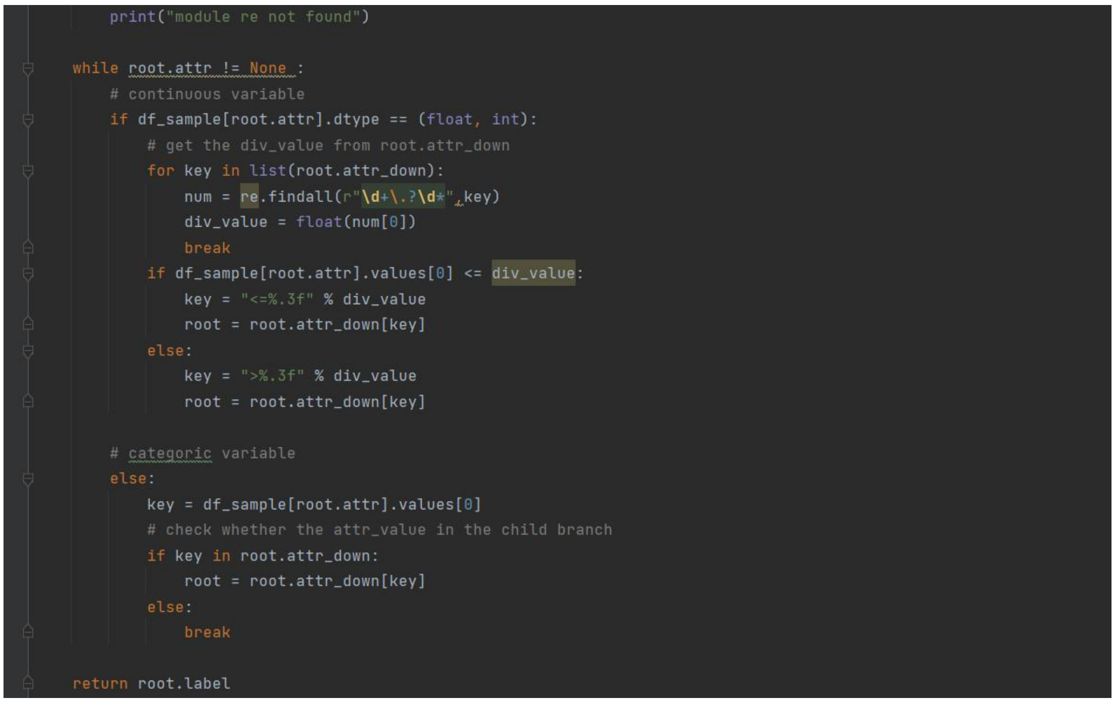
其中: $\text{Gini}(\tilde{D}^v) = 1 - \sum_{k=1}^{|y|} \tilde{p}_k^2$.



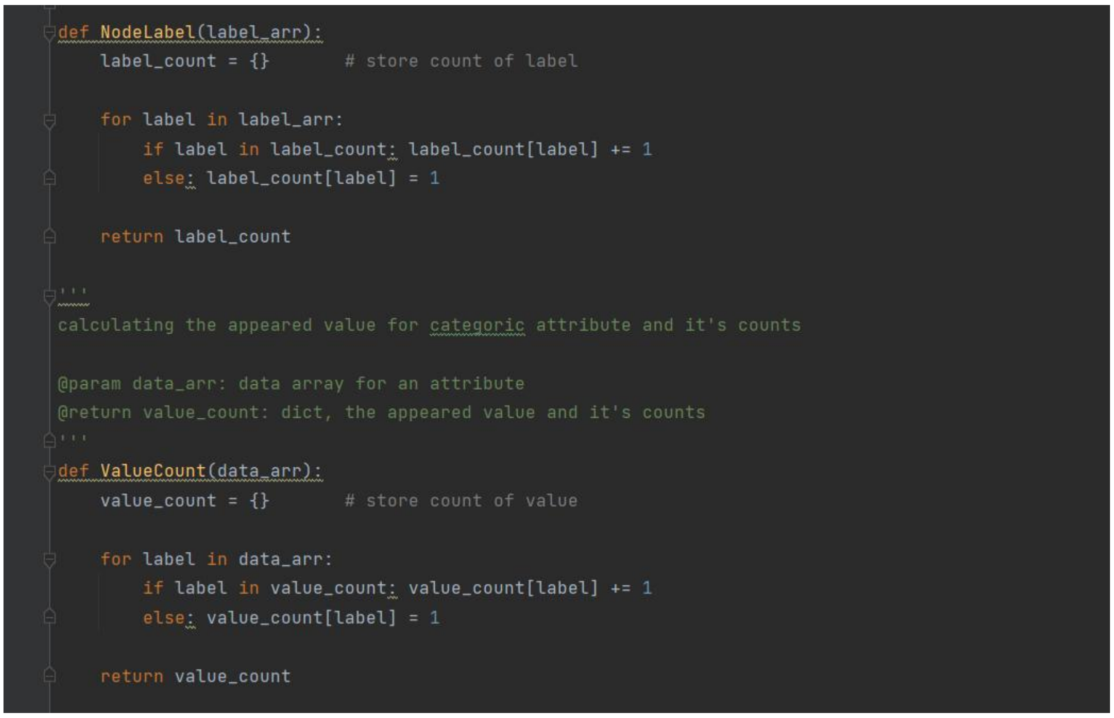
```python
class Node(object):
    def __init__(self, attr_init=None, label_init=None, attr_down_init={} ):
        self.attr = attr_init
        self.label = label_init
        self.attr_down = attr_down_init

'''
Branching for decision tree using recursion

@param df: the pandas dataframe of the data_set
@return root: Node, the root node of decision tree
'''
def TreeGenerate(df):
    # generating a new root node
    new_node = Node(None, None, {})
    label_arr = df[df.columns[-1]]

    label_count = NodeLabel(label_arr)
    if label_count:  # assert the label_count isn's empty
        new_node.label= max(label_count, key=label_count.get)

        # end if there is only 1 class in current node data
        # end if attribution array is empty
        if len(label_count) == 1 or len(label_arr) == 0:
            return new_node

        # get the optimal attribution for a new branching
        new_node.attr, div_value = OptAttr(df)
```



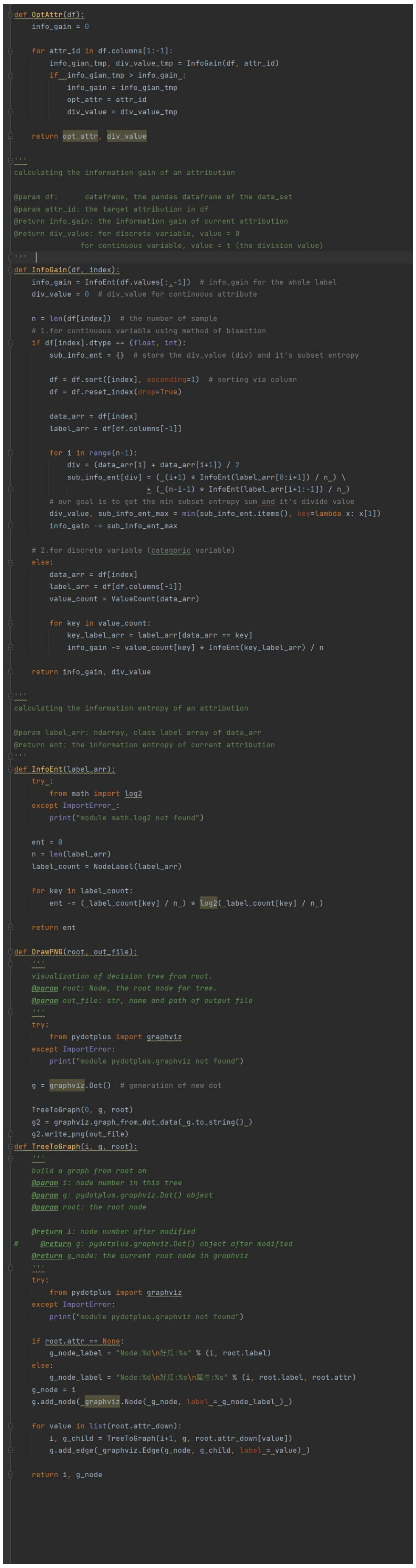
```python
        # recursion
        if div_value == 0:  # categoric variable
            value_count = ValueCount(df[new_node.attr])
            for value in value_count:
                df_v = df[ df[new_node.attr].isin([value]) ]  # get sub set
                # delete current attribution
                df_v = df_v.drop(new_node.attr, axis=1)
                new_node.attr_down[value] = TreeGenerate(df_v)

        else:  # continuous variable # left and right child
            value_l = "<=%.3f" % div_value
            value_r = ">%.3f" % div_value
            df_v_l = df[ df[new_node.attr] <= div_value ]  # get sub set
            df_v_r = df[ df[new_node.attr] > div_value ]

            new_node.attr_down[value_l] = TreeGenerate(df_v_l)
            new_node.attr_down[value_r] = TreeGenerate(df_v_r)

    return new_node

'''
make a predict based on root

@param root: Node, root Node of the decision tree
@param df_sample: dataframe, a sample line
'''
def Predict(root, df_sample):
    try :
        import re # using Regular Expression to get the number in string
    except ImportError :
```



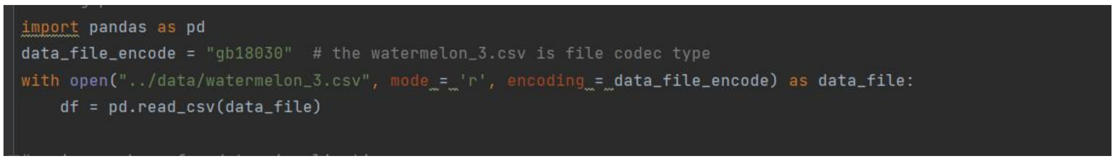
```python
        print("module re not found")

    while root.attr != None :
        # continuous variable
        if df_sample[root.attr].dtype == (float, int):
            # get the div_value from root.attr_down
            for key in list(root.attr_down):
                num = re.findall(r"\d+\.?\d*",key)
                div_value = float(num[0])
                break
            if df_sample[root.attr].values[0] <= div_value:
                key = "<=%.3f" % div_value
                root = root.attr_down[key]
            else:
                key = ">%.3f" % div_value
                root = root.attr_down[key]

        # categoric variable
        else:
            key = df_sample[root.attr].values[0]
            # check whether the attr_value in the child branch
            if key in root.attr_down:
                root = root.attr_down[key]
            else:
                break

    return root.label
```



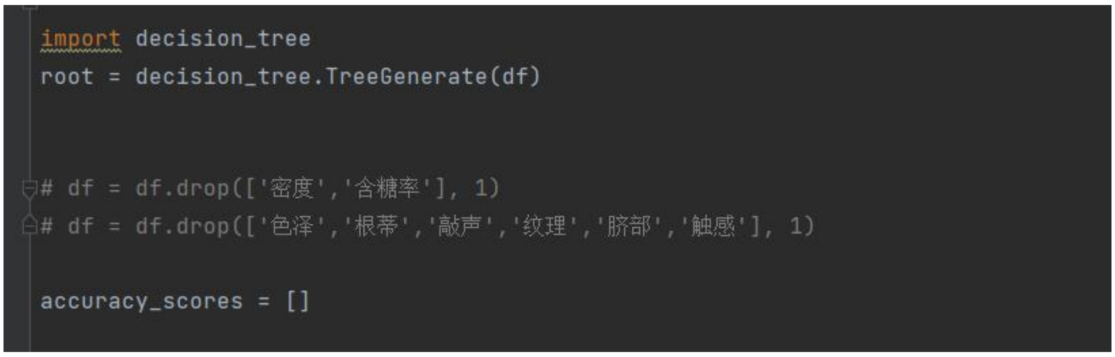
```python
def NodeLabel(label_arr):
    label_count = {}       # store count of label

    for label in label_arr:
        if label in label_count: label_count[label] += 1
        else: label_count[label] = 1

    return label_count

'''
calculating the appeared value for categoric attribute and it's counts

@param data_arr: data array for an attribute
@return value_count: dict, the appeared value and it's counts
'''
def ValueCount(data_arr):
    value_count = {}       # store count of value

    for label in data_arr:
        if label in value_count: value_count[label] += 1
        else: value_count[label] = 1

    return value_count
```



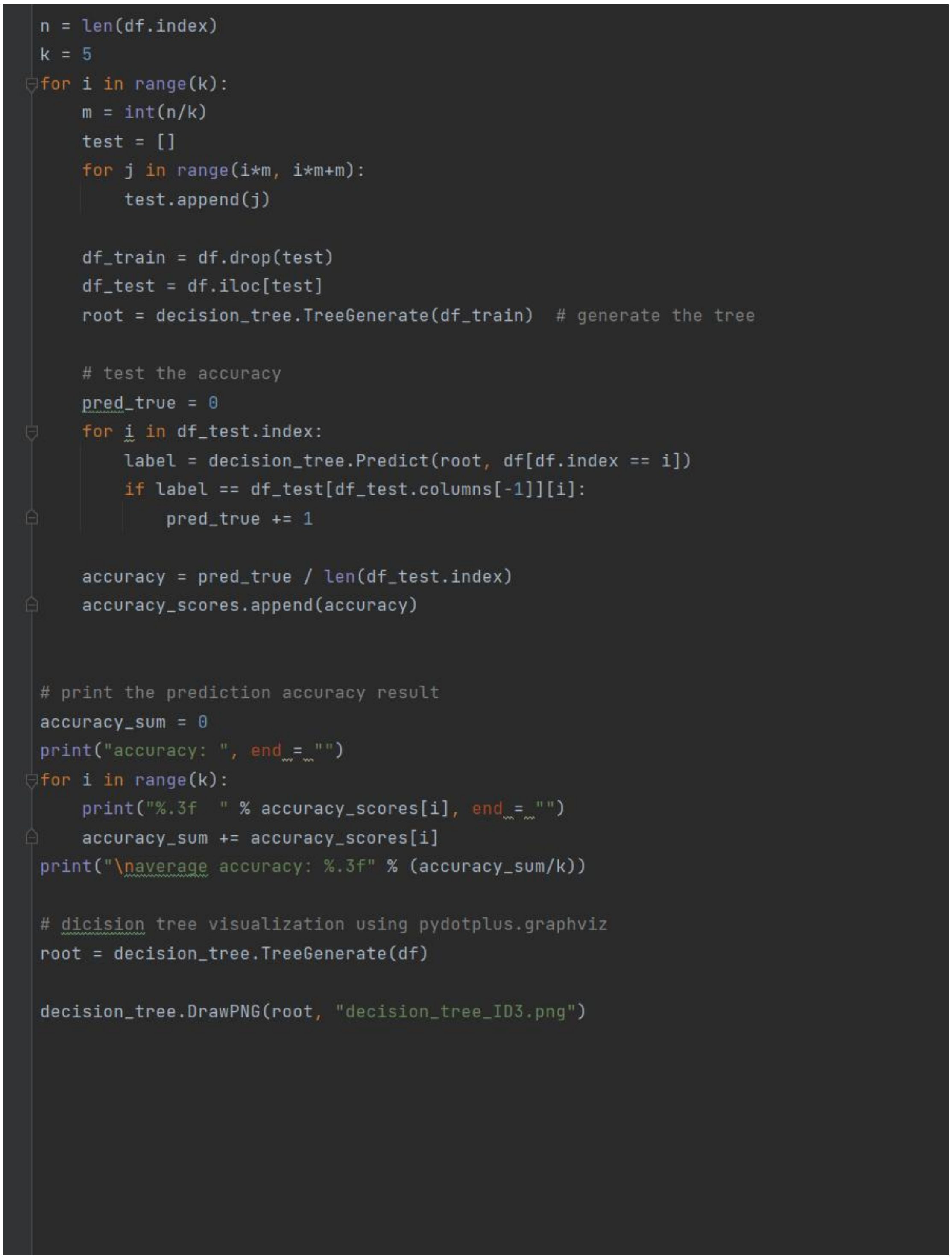
```python
def OptAttr(df):
    info_gain = 0

    for attr_id in df.columns[1:-1]:
        info_gian_tmp, div_value_tmp = InfoGain(df, attr_id)
        if  info_gian_tmp > info_gain :
            info_gain = info_gian_tmp
            opt_attr = attr_id
            div_value = div_value_tmp

    return opt_attr, div_value

'''
calculating the information gain of an attribution

@param df:      dataframe, the pandas dataframe of the data_set
@param attr_id: the target attribution in df
@return info_gain: the information gain of current attribution
@return div_value: for discrete variable, value = 0
               for continuous variable, value = t (the division value)
'''
def InfoGain(df, index):
    info_gain = InfoEnt(df.values[:,-1])  # info_gain for the whole label
    div_value = 0  # div_value for continuous attribute

    n = len(df[index])  # the number of sample
    # 1.for continuous variable using method of bisection
    if df[index].dtype == (float, int):
        sub_info_ent = {}  # store the div_value (div) and it's subset entropy

        df = df.sort([index], ascending=1)  # sorting via column
        df = df.reset_index(drop=True)

        data_arr = df[index]
        label_arr = df[df.columns[-1]]

        for i in range(n-1):
            div = (data_arr[i] + data_arr[i+1]) / 2
            sub_info_ent[div] = ( (i+1) * InfoEnt(label_arr[0:i+1]) / n ) \
                                + ( (n-i-1) * InfoEnt(label_arr[i+1:-1]) / n )
        # our goal is to get the min subset entropy sum and it's divide value
        div_value, sub_info_ent_max = min(sub_info_ent.items(), key=lambda x: x[1])
        info_gain -= sub_info_ent_max

    # 2.for discrete variable (categoric variable)
    else:
        data_arr = df[index]
        label_arr = df[df.columns[-1]]
        value_count = ValueCount(data_arr)

        for key in value_count:
            key_label_arr = label_arr[data_arr == key]
            info_gain -= value_count[key] * InfoEnt(key_label_arr) / n

    return info_gain, div_value

'''
calculating the information entropy of an attribution

@param label_arr: ndarray, class label array of data_arr
@return ent: the information entropy of current attribution
'''
def InfoEnt(label_arr):
    try :
        from math import log2
    except ImportError :
        print("module math.log2 not found")

    ent = 0
    n = len(label_arr)
    label_count = NodeLabel(label_arr)

    for key in label_count:
        ent -= ( label_count[key] / n ) * log2( label_count[key] / n )

    return ent

def DrawPNG(root, out_file):
    '''
    visualization of decision tree from root.
    @param root: Node, the root node for tree.
    @param out_file: str, name and path of output file
    '''
    try:
        from pydotplus import graphviz
    except ImportError:
        print("module pydotplus.graphviz not found")

    g = graphviz.Dot()  # generation of new dot

    TreeToGraph(0, g, root)
    g2 = graphviz.graph_from_dot_data( g.to_string() )
    g2.write_png(out_file)
def TreeToGraph(i, g, root):
    '''
    build a graph from root on
    @param i: node number in this tree
    @param g: pydotplus.graphviz.Dot() object
    @param root: the root node

    @return i: node number after modified
#     @return g: pydotplus.graphviz.Dot() object after modified
    @return g_node: the current root node in graphviz
    '''
    try:
        from pydotplus import graphviz
    except ImportError:
        print("module pydotplus.graphviz not found")

    if root.attr == None:
        g_node_label = "Node:%d\n好瓜:%s" % (i, root.label)
    else:
        g_node_label = "Node:%d\n好瓜:%s\n属性:%s" % (i, root.label, root.attr)
    g_node = i
    g.add_node( graphviz.Node( g_node, label = g_node_label ) )

    for value in list(root.attr_down):
        i, g_child = TreeToGraph(i+1, g, root.attr_down[value])
        g.add_edge( graphviz.Edge(g_node, g_child, label = value) )

    return i, g_node
```



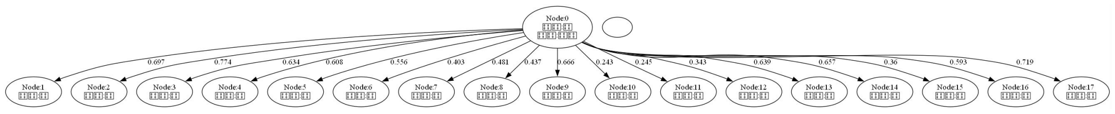
```python
import pandas as pd
data_file_encode = "gb18030"  # the watermelon_3.csv is file codec type
with open("../data/watermelon_3.csv", mode = 'r', encoding = data_file_encode) as data_file:
    df = pd.read_csv(data_file)
```



```python
import decision_tree
root = decision_tree.TreeGenerate(df)


# df = df.drop(['密度','含糖率'], 1)
# df = df.drop(['色泽','根蒂','敲声','纹理','脐部','触感'], 1)

accuracy_scores = []
```



```python
n = len(df.index)
k = 5
for i in range(k):
    m = int(n/k)
    test = []
    for j in range(i*m, i*m+m):
        test.append(j)

    df_train = df.drop(test)
    df_test = df.iloc[test]
    root = decision_tree.TreeGenerate(df_train)  # generate the tree

    # test the accuracy
    pred_true = 0
    for i in df_test.index:
        label = decision_tree.Predict(root, df[df.index == i])
        if label == df_test[df_test.columns[-1]][i]:
            pred_true += 1

    accuracy = pred_true / len(df_test.index)
    accuracy_scores.append(accuracy)


# print the prediction accuracy result
accuracy_sum = 0
print("accuracy: ", end = "")
for i in range(k):
    print("%.3f  " % accuracy_scores[i], end = "")
    accuracy_sum += accuracy_scores[i]
print("\naverage accuracy: %.3f" % (accuracy_sum/k))

# dicision tree visualization using pydotplus.graphviz
root = decision_tree.TreeGenerate(df)

decision_tree.DrawPNG(root, "decision_tree_ID3.png")
```



Node:0
0.697
0.774
0.634
0.608
0.556
0.403
0.481
0.437
0.666
0.243
0.245
0.343
0.639
0.657
0.36
0.593
0.719
Node:1
Node:2
Node:3
Node:4
Node:5
Node:6
Node:7
Node:8
Node:9
Node:10
Node:11
Node:12
Node:13
Node:14
Node:15
Node:16
Node:17