目录

[（a）ID3 2](#_Toc15441)

[（b）CART 6](#_Toc5632)

[（c） 13](#_Toc31437)

[Additional questions 13](#_Toc26253)

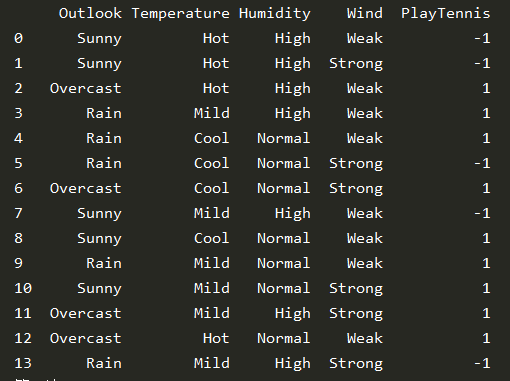
[代码 14](#_Toc17862)

[ID3代码 14](#_Toc11743)

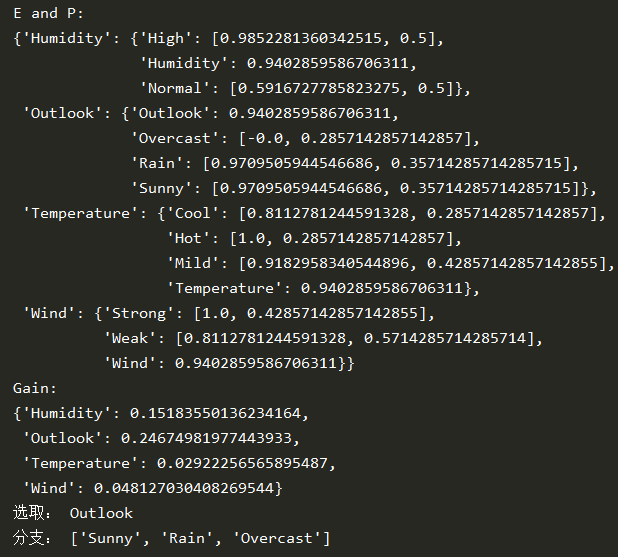
[CART代码 17](#_Toc23736)

### （a）ID3

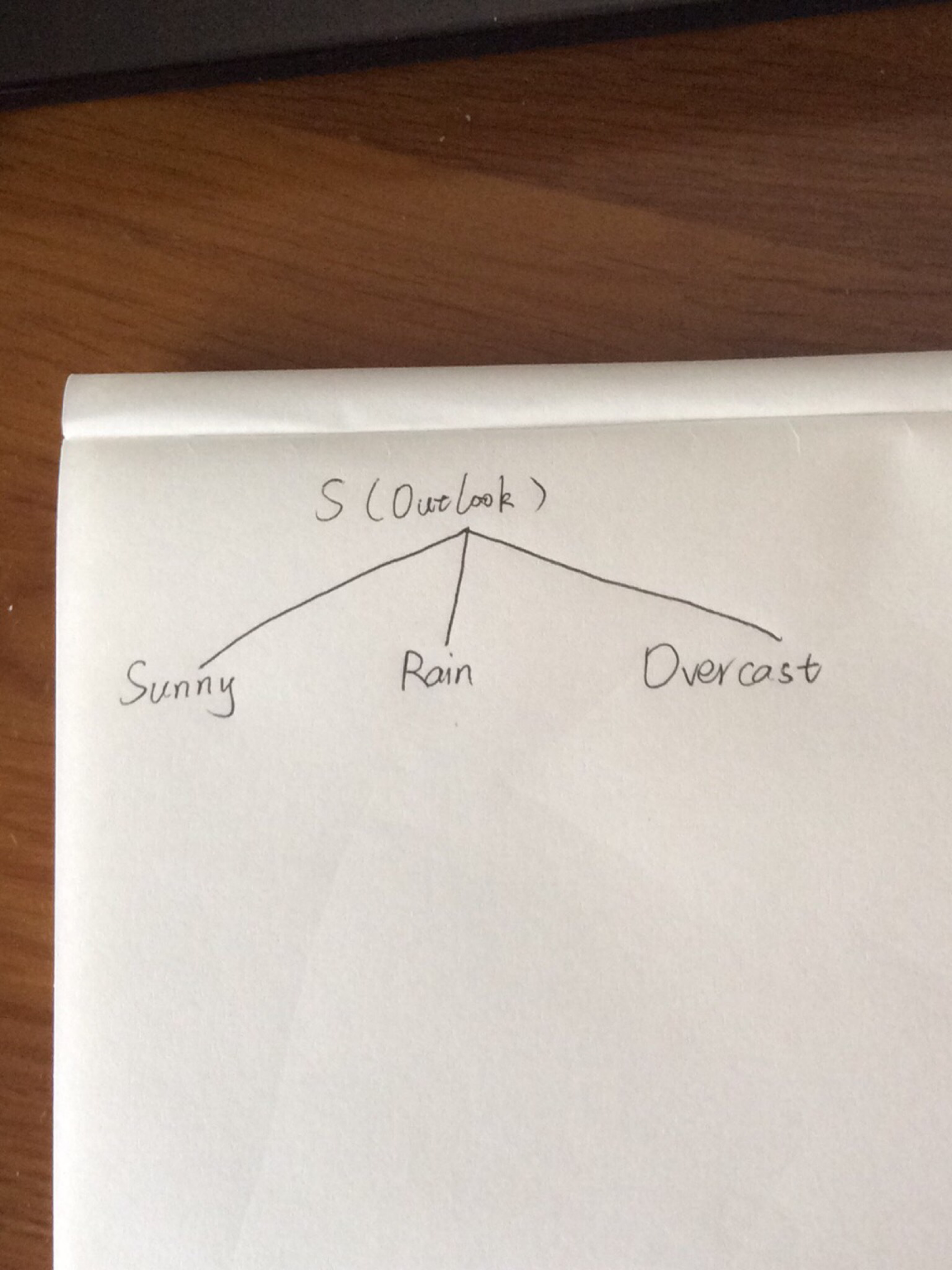
Data：



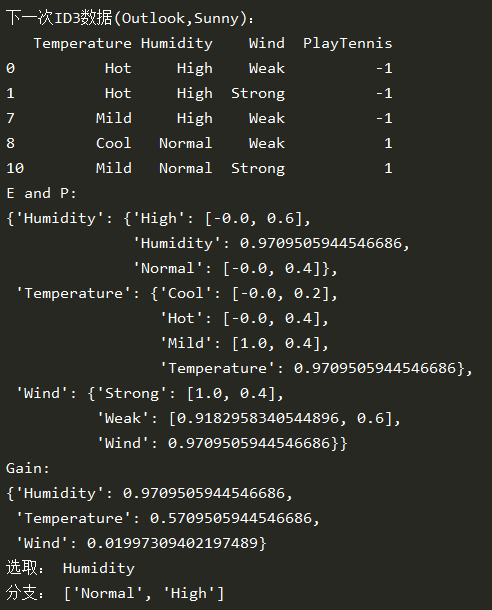
First calculation：



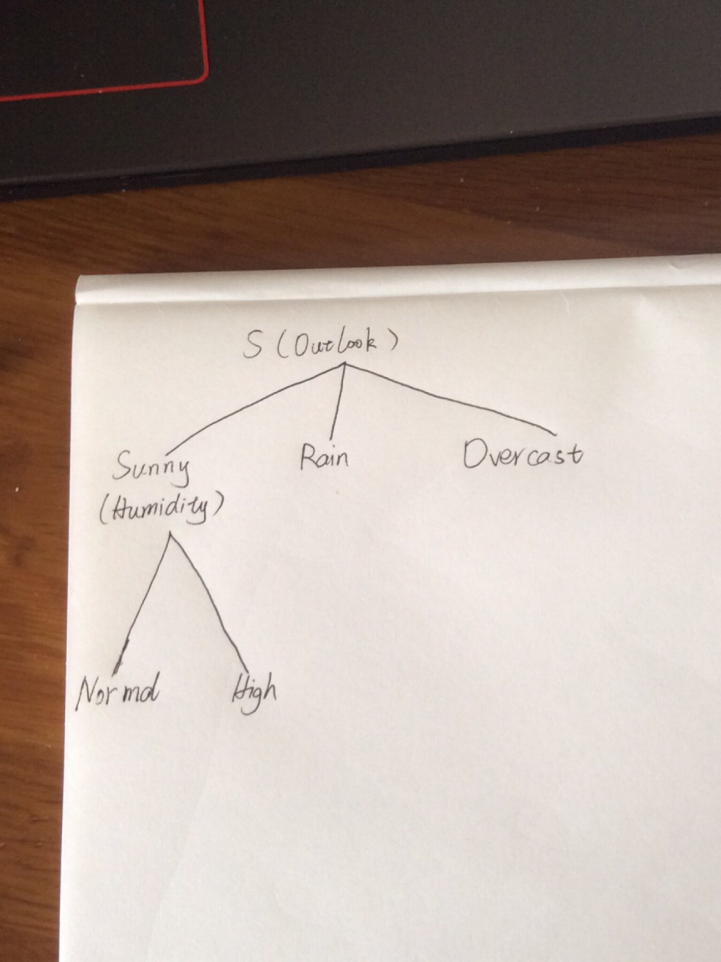
Get decision tree：



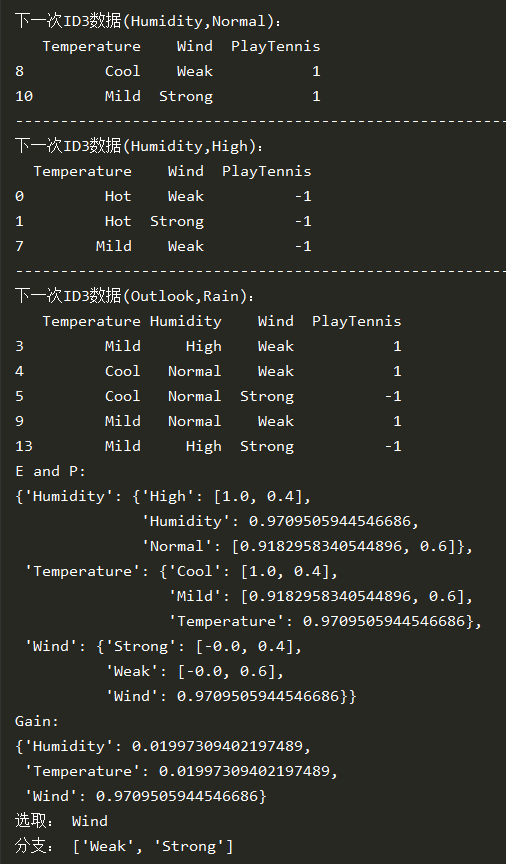
Next calculation:



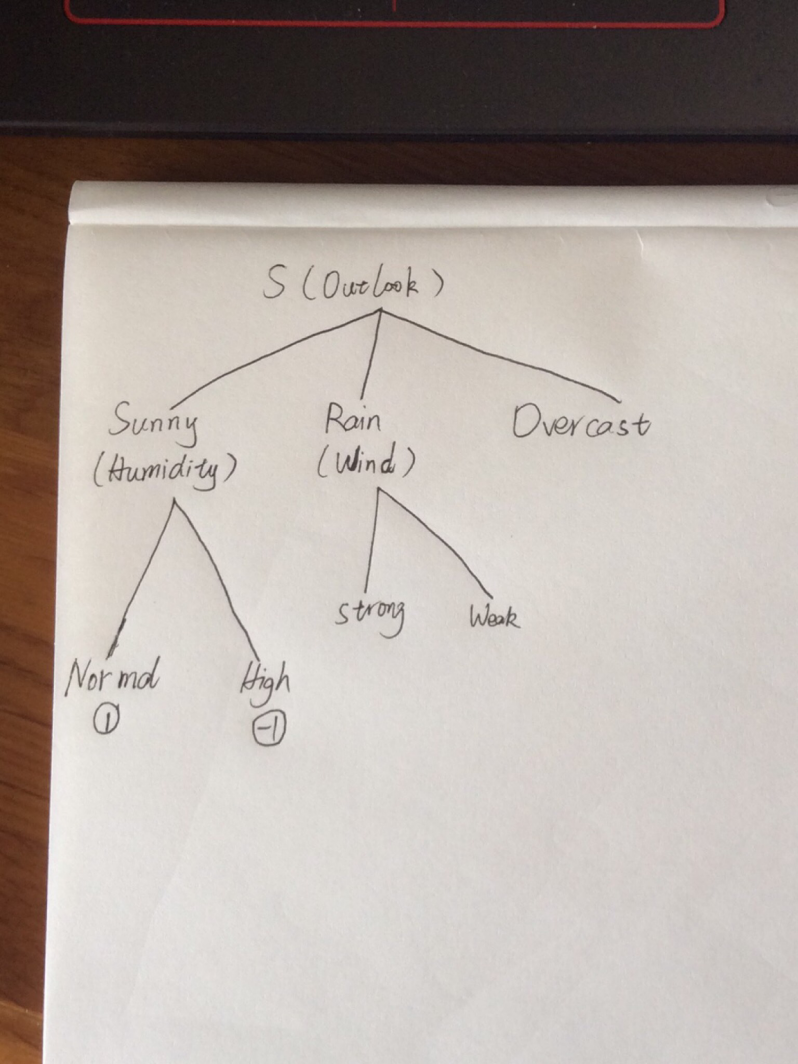
Get decision tree：



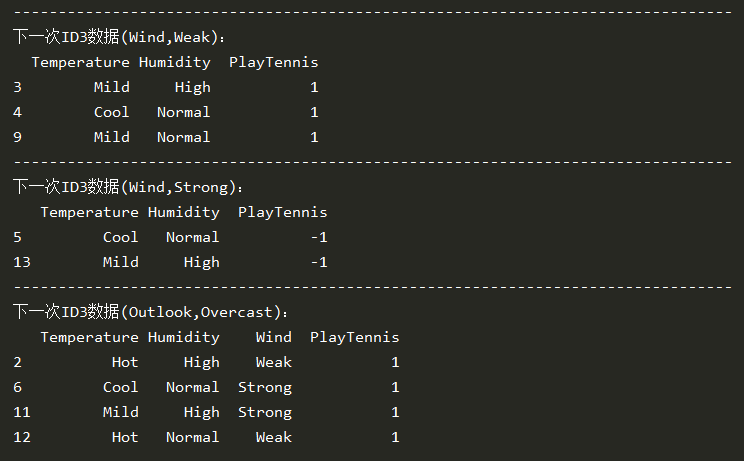
Future calculation:



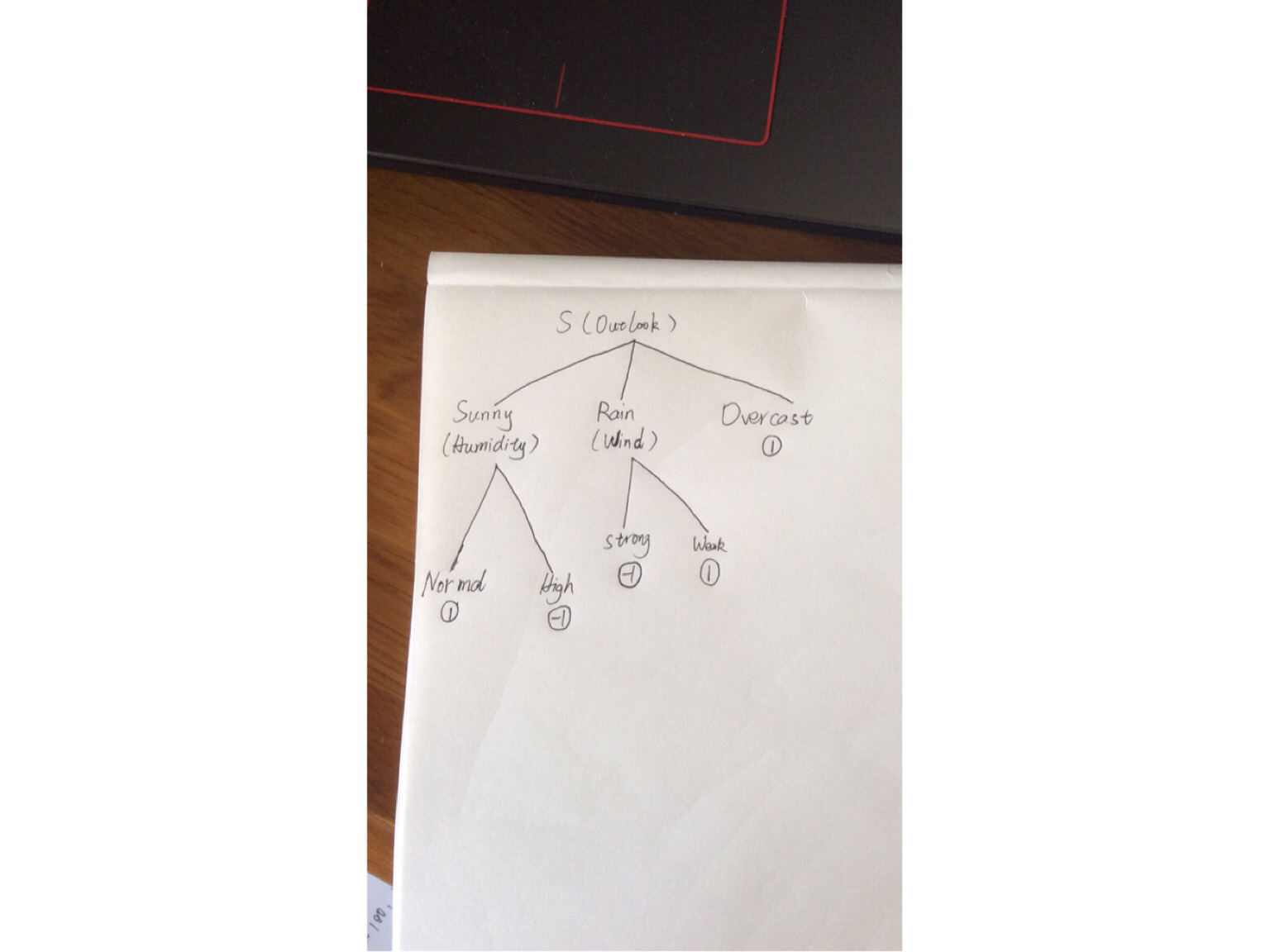
Get decision tree：



Future calculation:



Get final decision tree：

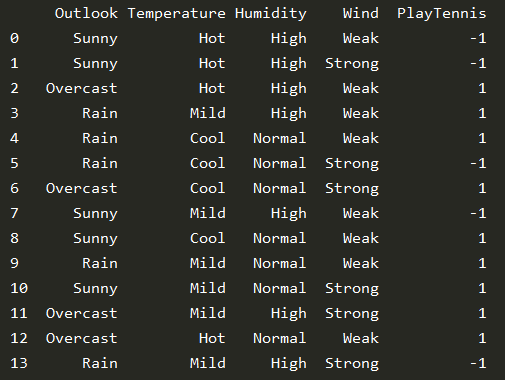


### （b）CART

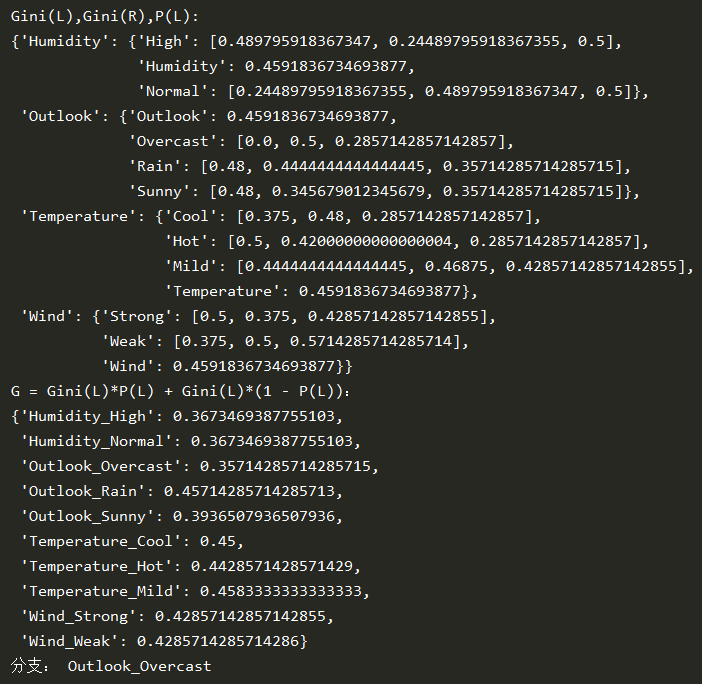
 



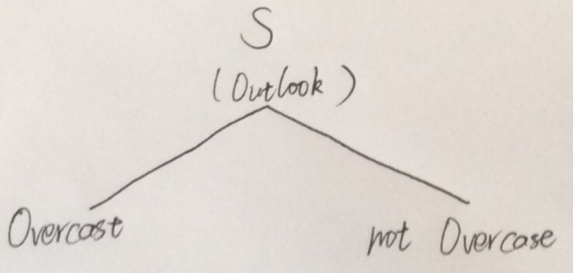
Data:



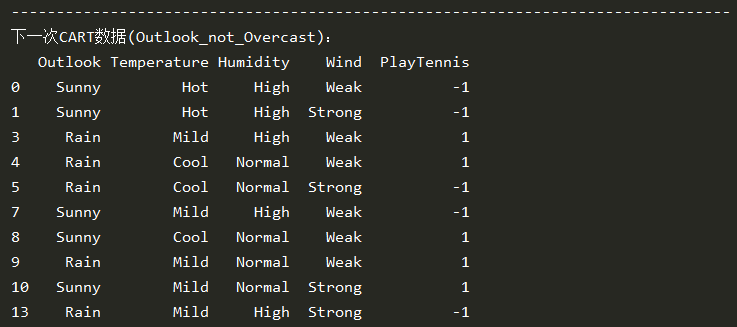
First calculation：

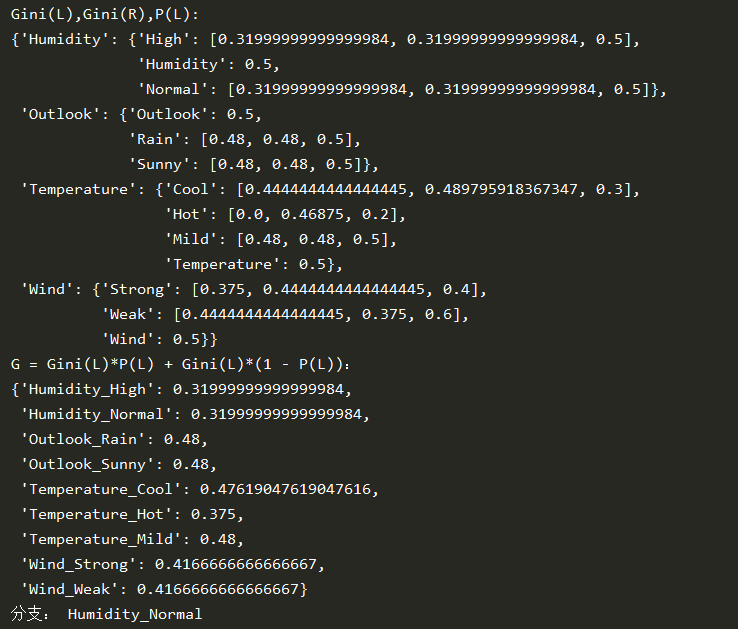


Get decision tree：

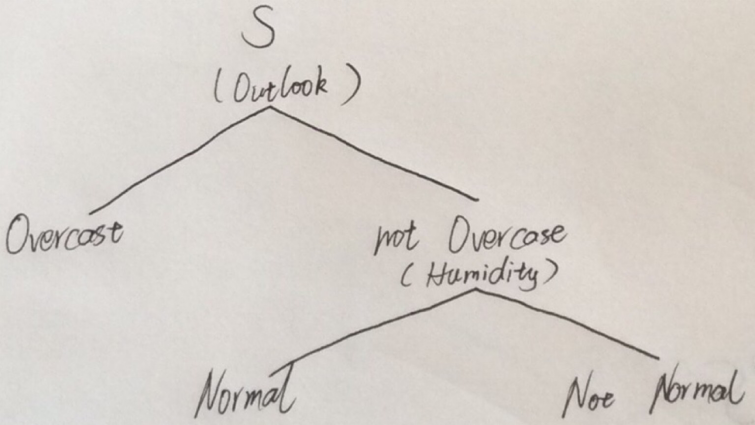


Next calculation:

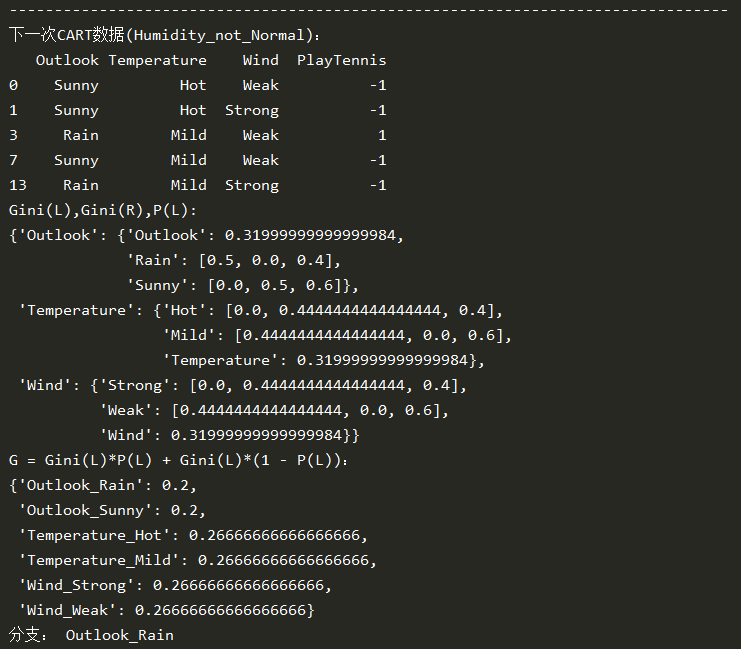




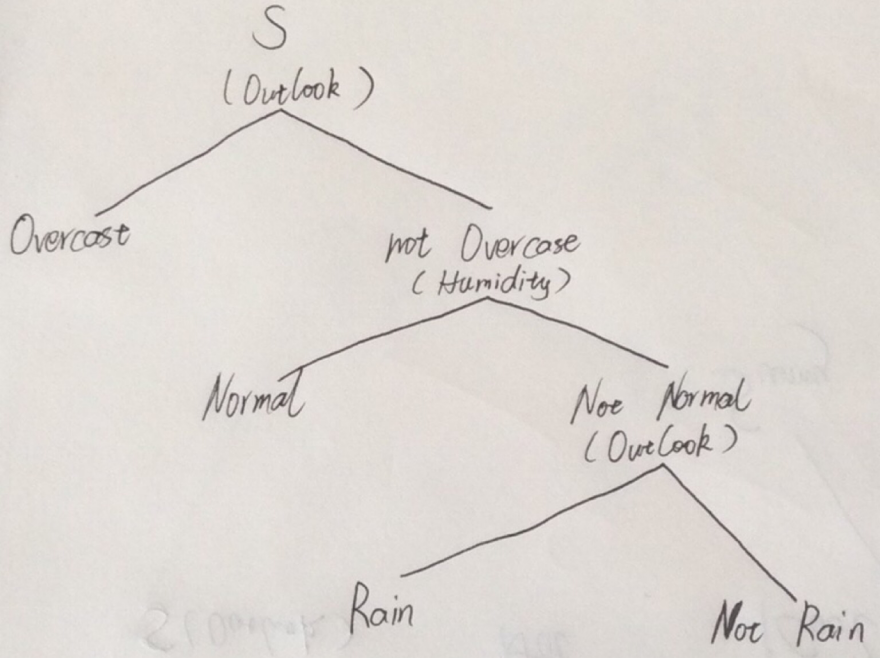
Get decision tree：



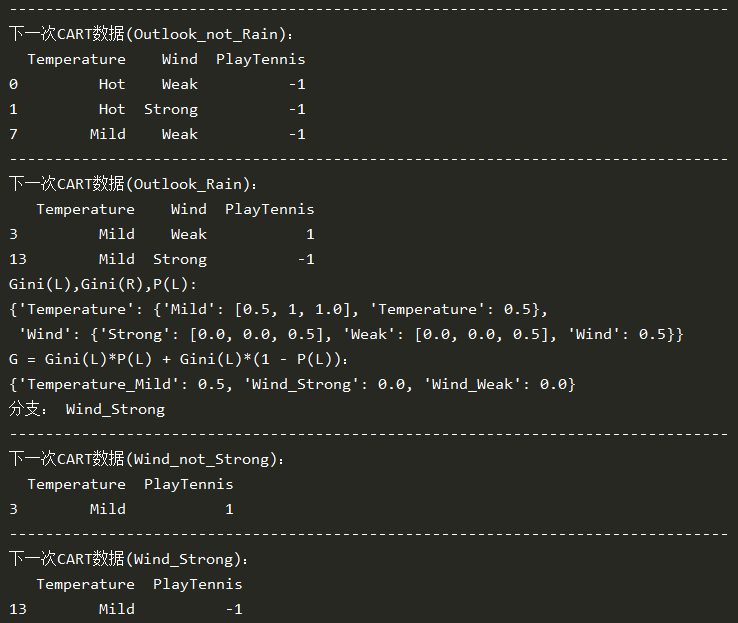
Next calculation:



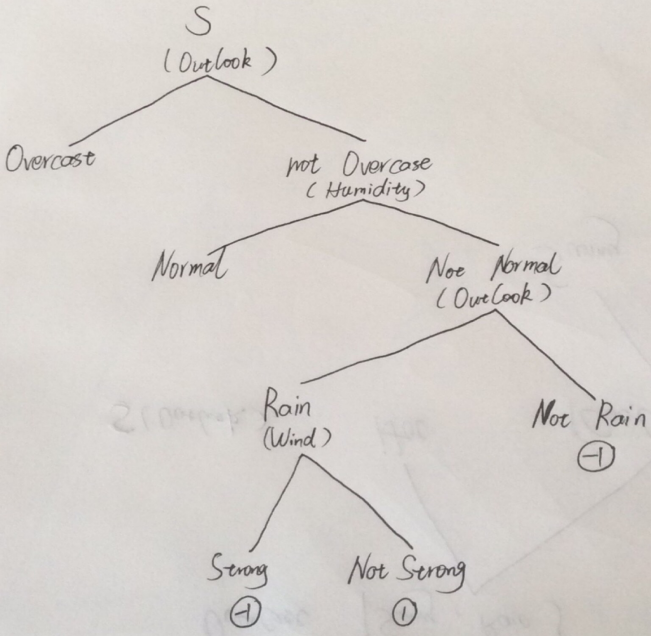
Get decision tree：



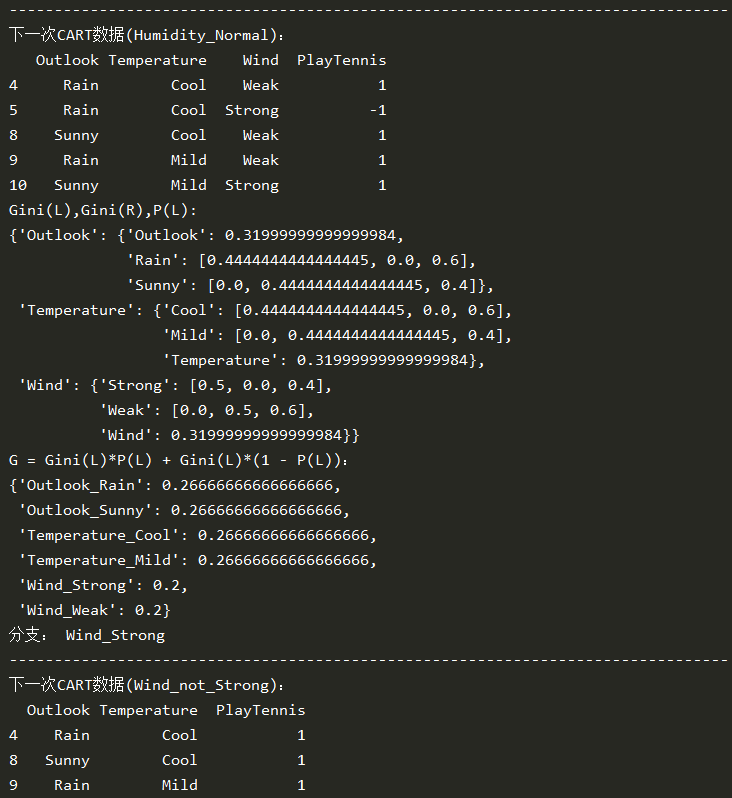
Future calculation：



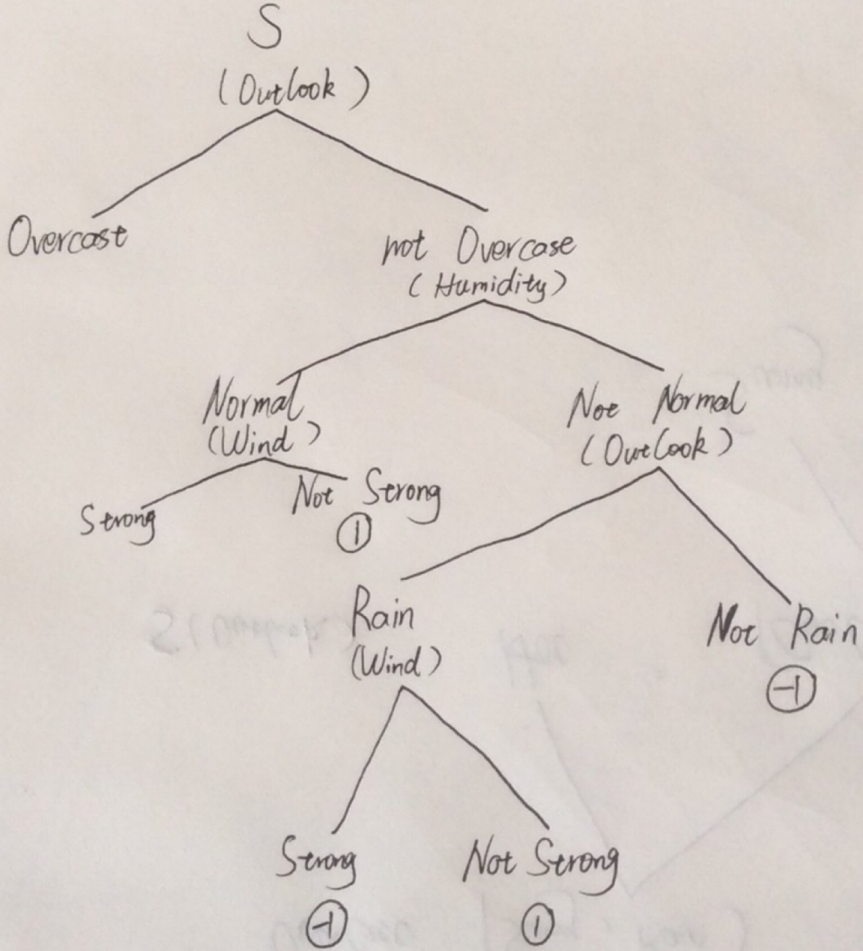
Get decision tree：



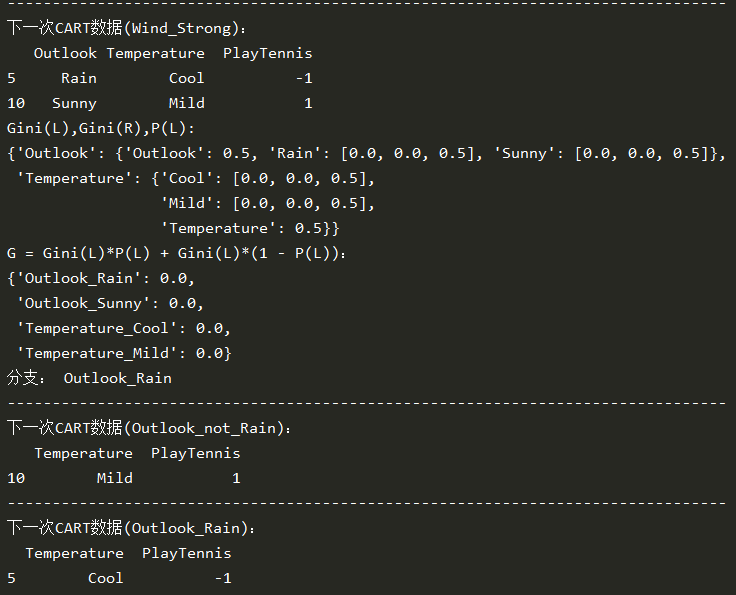
Future calculation：



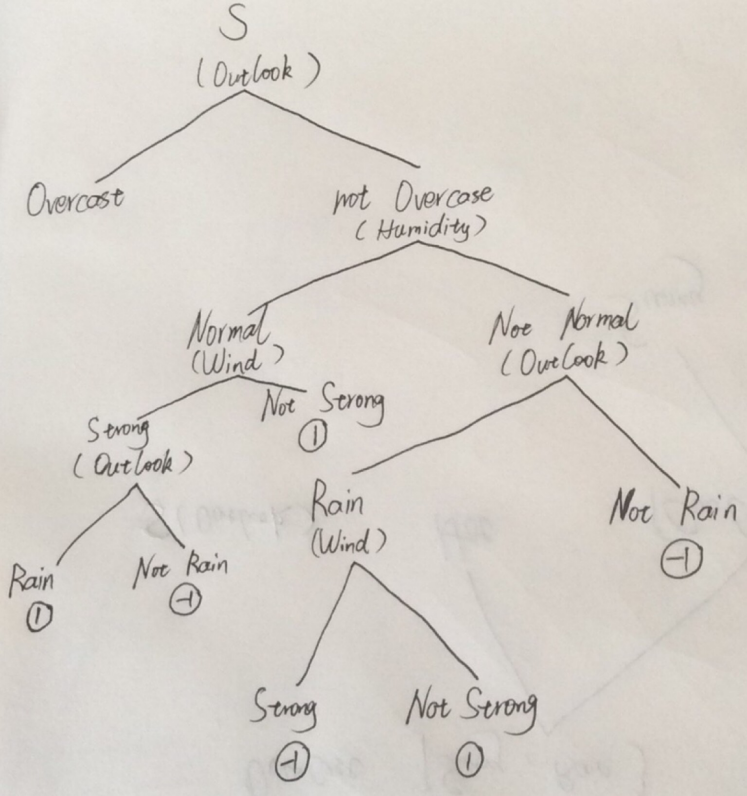
Get decision tree：



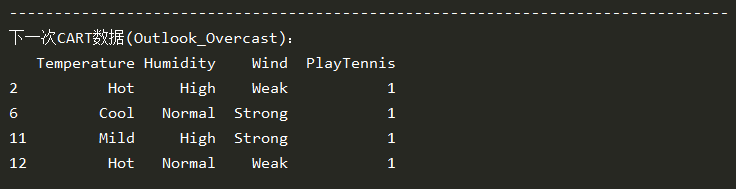
Future calculation：



Get decision tree：



Future calculation：



Get final decision tree：



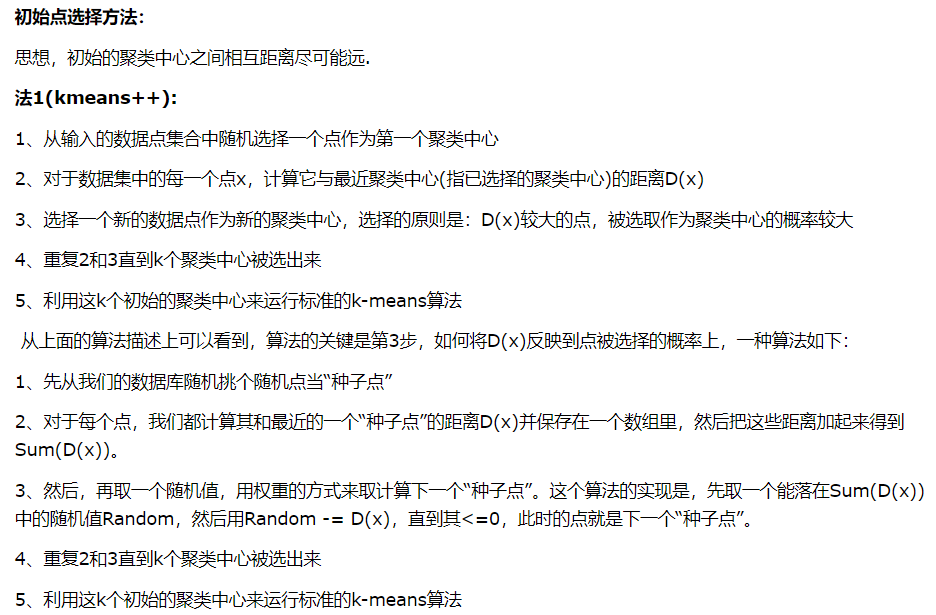
### （c）

1. They use different impurty.
2. ID3 nodes can have multiple child nodes.
3. ID3 can only process classification variables, CART can process continuous and classification variables.
4. CART tree is deeper.

### Additional questions

1. **How K-means choose the initial point?**

Picture from blog : <https://www.cnblogs.com/dudumiaomiao/p/5839905.html>



1. **How to update the point can make the result better?**

In order to solve the problem that only the local optimal solution (equivalent to greedy algorithm) can be obtained, limited local update can be selected during update.

In order to solve the problem of nonconvex clustering, select a distance measure that better matches the dataset when updating(like some kernel functions).

In order to solve the problem of sensitive to noise points, the cluster can be deleted properly when updating.

### 代码

使用Python3.7编程，没有真正的构建决策树，只是使用Python迭代进行各个分支的计算

#### ID3代码

*import* numpy *as* np  
*import* pandas *as* pd  
*from* pprint *import* pprint  
pd.set\_option('display.unicode.ambiguous\_as\_wide', *True*)  
pd.set\_option('display.unicode.east\_asian\_width', *True*)  
*def* ID3(*data*):  
 # 计算ID3的E和Gain，并迭代进行各个分支计算  
  
 # 输入data为dataframe数据，最后一列是判断值  
 # 例子，data：  
 # Outlook Temperature Humidity Wind PlayTennis  
 # 0 Sunny Hot High Weak -1  
 # 1 Sunny Hot High Strong -1  
 # 2 Overcast Hot High Weak 1  
  
 *def* calculate\_iN(*target*):  
 # 计算i(N)，输入为一列dataframe数据  
 # 例子：  
 # target = PlayTennis（列名）  
 # 1  
 # -1  
 # 1  
 # 输出为i(N) = -(1/3)log(1/3)-(2/3)log(2/3)  
 target\_list = *target*.tolist()  
 unique\_target\_list = list(set(target\_list))  
 E = [0] \* len(unique\_target\_list)  
 *for* i *in* range(len(target\_list)):  
 target\_index = unique\_target\_list.index(target\_list[i])  
 E[target\_index] += 1  
 *for* i *in* range(len(unique\_target\_list)):  
 E[i] /= len(target\_list)  
 E[i] = E[i] \* np.log2(E[i])  
 *return* -sum(E)  
  
 *def* get\_unique\_name(*column*):  
 # 得到无重复元素的集合，输入为一列dataframe数据  
 # 例子：  
 # column = Humidity（列名）  
 # High  
 # Normal  
 # Normal  
 # 输出为[High, Normal]  
 column\_list = *column*.tolist()  
 unique\_column\_list = list(set(column\_list))  
 *return* unique\_column\_list  
  
 *def* get\_P(*column*, *name*):  
 # 得到指定元素的比例，输入为一列dataframe数据  
 # 例子：  
 # name = High  
 # column = Humidity（列名）  
 # High  
 # Normal  
 # Normal  
 # 输出为1/3  
 column\_list = *column*.tolist()  
 P = 0  
 *for* i *in* column\_list:  
 *if* i == *name*:  
 P += 1  
 P\_ = 1 *if* P == 0 *else* P / len(column\_list)  
 *return* P\_  
  
 # 得到列名  
 column\_name = *data*.columns.tolist()  
 column\_len = len(column\_name)  
 target\_name = column\_name[column\_len - 1]  
 *del* column\_name[column\_len - 1]  
 # 分别计算E(i(N))和Gain(theta\_i(N))  
 E = {}  
 theta\_iN = {}  
 *for* i *in* column\_name:  
 E[i] = {}  
 E[i][i] = calculate\_iN(*data*[target\_name])  
 theta\_iN[i] = E[i][i]  
 unique\_name = get\_unique\_name(*data*.loc[:, i])  
 *for* j *in* unique\_name:  
 i\_j\_data = *data*[*data*[i] == j][target\_name]  
 E[i][j] = [calculate\_iN(i\_j\_data), get\_P(*data*[i], j)]  
 *for* j *in* unique\_name:  
 theta\_iN[i] -= (E[i][j][0] \* E[i][j][1])  
 # 输出E和Gain  
 print('E and P:')  
 pprint(E)  
 print('Gain:')  
 pprint(theta\_iN)  
 # 选取最大的Gain  
 max\_name = list(theta\_iN.keys())[0]  
 *for* i *in* theta\_iN.keys():  
 *if* theta\_iN[i] > theta\_iN[max\_name]:  
 max\_name = i  
 print('选取：', max\_name)  
 print('分支：', get\_unique\_name(*data*[max\_name]))  
 *if* max\_name *in* list(*data*.columns) *and* len(*data*.columns) != 2:  
 # 对每个分支进行类似操作  
 *for* i *in* get\_unique\_name(*data*[max\_name]):  
 i\_data = *data*[*data*[max\_name] == i]  
 *del* i\_data[max\_name]  
 # 输出分支下的dataframe数据  
 print('-' \* 80)  
 print('下一次ID3数据({},{})：'.format(max\_name, i))  
 pprint(i\_data)  
 # 判断是否需要继续计算,如果剩下的数据都是同一类，就不需要计算了  
 *if* len(get\_unique\_name(i\_data[target\_name])) != 1:  
 # 进行下一次ID3的计算  
 ID3(i\_data)  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 *import* pandas *as* pd  
 raw\_data = {  
 'Outlook':['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain'],  
 'Temperature':['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],  
 'Humidity':['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High'],  
 'Wind':['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Strong'],  
 'PlayTennis':[-1,-1,1,1,1,-1,1,-1,1,1,1,1,1,-1]  
 }  
 data = pd.DataFrame.from\_dict(raw\_data)  
 pprint(data)  
 print('第一次ID3：')  
 ID3(data)

#### CART代码

*import* numpy *as* np  
*import* pandas *as* pd  
*from* pprint *import* pprint  
pd.set\_option('display.unicode.ambiguous\_as\_wide', *True*)  
pd.set\_option('display.unicode.east\_asian\_width', *True*)  
*def* CART(*data*):  
 # 计算ID3的E和Gain，并迭代进行各个分支计算  
  
 # 输入data为dataframe数据，最后一列是判断值  
 # 例子，data：  
 # Outlook Temperature Humidity Wind PlayTennis  
 # 0 Sunny Hot High Weak -1  
 # 1 Sunny Hot High Strong -1  
 # 2 Overcast Hot High Weak 1  
  
 *def* calculate\_iN(*target*):  
 # 计算Gini(i(N))，输入为一列dataframe数据  
 # 例子：  
 # target = PlayTennis（列名）  
 # 1  
 # -1  
 # 1  
 # 输出为i(N) = (1 - (1/3)^2 - (2/3)^2)  
 target\_list = *target*.tolist()  
 unique\_target\_list = list(set(target\_list))  
 E = [0] \* len(unique\_target\_list)  
 *for* i *in* range(len(target\_list)):  
 target\_index = unique\_target\_list.index(target\_list[i])  
 E[target\_index] += 1  
 iN = 1  
 *for* i *in* range(len(unique\_target\_list)):  
 E[i] /= len(target\_list)  
 iN -= E[i] \*\* 2  
 *return* iN  
  
 *def* get\_unique\_name(*column*):  
 # 得到无重复元素的集合，输入为一列dataframe数据  
 # 例子：  
 # column = Humidity（列名）  
 # High  
 # Normal  
 # Normal  
 # 输出为[High, Normal]  
 column\_list = *column*.tolist()  
 unique\_column\_list = list(set(column\_list))  
 *return* unique\_column\_list  
  
 *def* get\_P(*column*, *name*):  
 # 得到指定元素的比例，输入为一列dataframe数据  
 # 例子：  
 # name = High  
 # column = Humidity（列名）  
 # High  
 # Normal  
 # Normal  
 # 输出为1/3  
 column\_list = *column*.tolist()  
 P = 0  
 *for* i *in* column\_list:  
 *if* i == *name*:  
 P += 1  
 P\_ = 1 *if* P == 0 *else* P / len(column\_list)  
 *return* P\_  
 *if* len(*data*.columns) == 1 *or data*.empty:  
 *return* # 得到列名  
 column\_name = *data*.columns.tolist()  
 column\_len = len(column\_name)  
 target\_name = column\_name[column\_len - 1]  
 *del* column\_name[column\_len - 1]  
 # 分别计算Gini(L)和Gini(R)和G  
 Gini = {}  
 theta\_iN = {}  
 *for* i *in* column\_name:  
 Gini[i] = {}  
 Gini[i][i] = calculate\_iN(*data*[target\_name])  
 unique\_name = get\_unique\_name(*data*.loc[:, i])  
 *for* j *in* unique\_name:  
 i\_j\_data = *data*[*data*[i] == j][target\_name]  
 i\_not\_j\_data = *data*[*data*[i] != j][target\_name]  
 Gini[i][j] = [calculate\_iN(i\_j\_data), calculate\_iN(i\_not\_j\_data), get\_P(*data*[i], j)]  
 *for* j *in* unique\_name:  
 theta\_iN[i + '\_' + j] = (Gini[i][j][0] \* Gini[i][j][2]) + (Gini[i][j][1] \*(1 - Gini[i][j][2]))  
 # 输出E和Gain  
 print('Gini(L),Gini(R),P(L):')  
 pprint(Gini)  
 print('G = Gini(L)\*P(L) + Gini(L)\*(1 - P(L))：')  
 pprint(theta\_iN)  
 # 选取最大的Gain  
 min\_name = list(theta\_iN.keys())[0]  
 *for* i *in* theta\_iN.keys():  
 *if* theta\_iN[i] < theta\_iN[min\_name]:  
 min\_name = i  
 print('分支：', min\_name)  
 # 获得下次数据  
 i = min\_name.split('\_')[0]  
 j = min\_name.split('\_')[1]  
 i\_not\_j\_data = *data*[*data*[i] != j]  
 i\_j\_data = *data*[*data*[i] == j]  
  
 *if* len(get\_unique\_name(i\_not\_j\_data[i])) == 1:  
 *del* i\_not\_j\_data[i]  
 *if* len(get\_unique\_name(i\_j\_data[i])) == 1:  
 *del* i\_j\_data[i]  
  
 print('-' \* 80)  
 print('下一次CART数据({}\_not\_{})：'.format(i, j))  
 pprint(i\_not\_j\_data)  
 *if* len(get\_unique\_name(i\_not\_j\_data[target\_name])) != 1:  
 CART(i\_not\_j\_data)  
  
 print('-' \* 80)  
 print('下一次CART数据({}\_{})：'.format(i, j))  
 pprint(i\_j\_data)  
 *if* len(get\_unique\_name(i\_j\_data[target\_name])) != 1:  
 CART(i\_j\_data)  
  
  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 *import* pandas *as* pd  
 raw\_data = {  
 'Outlook':['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain'],  
 'Temperature':['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],  
 'Humidity':['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High'],  
 'Wind':['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Strong'],  
 'PlayTennis':[-1,-1,1,1,1,-1,1,-1,1,1,1,1,1,-1]  
 }  
 data = pd.DataFrame.from\_dict(raw\_data)  
 pprint(data)  
 CART(data)