8+CART(classification)

2019 年 3 月 10 日

1 Classification and Regresssion Trees(Classification)

1.1 一、概念

决策树是一种经典且当前应用依然十分活跃的算法，用简单的一句话概括决策树的训练过程就是不停划分变量空间最后得到函数映射。就像分段函数一样:

$$sign(x) = \begin{cases} 1 & x > 0 \\ -1 & x \leq 0 \end{cases}$$

我们可以看到这样的分类具有完备和互斥地性质，这意味着对于样本空间内的输入都有且仅有一个输出。

如果你熟悉数据结构，那么决策树是通常一个完全二叉树，如果不熟悉也没关系。

决策树有以下要点：

1. 组织形式——-树，
2. 划分准则——-信息增益、基尼系数等，本质都是衡量 “不纯度”
3. 增长方式——-通过不断地选择变量进行划分然后生成子树
4. 剪枝——-防止过拟合

譬如我们有一组数据，判断病人是否可能患病，决策树可能就会通过判断年龄是否大于 50，血压血糖是否大于某个值来进行判断，也因此决策树具有很强的可解释性，展示起来十分直观

1.2 二、训练

1.2.1 1. 组织形式

树依靠节点之间的关系来组织。因为暂时不方便传图，所以不赘述。根节点为 1，父节点和子节点之间有如下关系式：

$$parent(i) = \lfloor \frac{i}{2} \rfloor$$
$$left - child(i) = 2i$$
$$right - child(i) = 2i + 1$$

1.2.2 2. 划分准则

衡量不纯度的指标有很多，不同的决策树算法会选用不同的指标，比如经典的 ID3 树和 C4.5 选用信息熵作为划分准则，而 CART 分类树选用基尼系数。

基尼系数：假设有 K 个类，样本点属于第 i 类的概率为 p_i，概率分布的基尼系数定义为：

$$Gini(p) = \sum_{i=1}^{K} p_i(1 - p_i) = 1 - \sum_{i=1}^{K} p_i^2$$

对于给定的样本集合 D，其基尼系数为：

$$Gini(D) = 1 - \sum_{i=1}^{K} \left(\frac{|C_i|}{|D|} \right)^2$$

其中 C_i 是样本中属于第 i 类的样本子集，K 是类的个数。

如果样本集合 D 根据特征 A 是否去某一可能值 a 被分割成 D_1 和 D_2 两个子集，则在特征 A 取值 a 的情况下集合 D 的基尼系数定义为：

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

1.2.3 3. 增长方式

有了组织形式和划分准则之后，我们就要考虑如何让这棵树 ‘‘生长’’。

以二分类为例，易知基尼系数的最大值是 0.5，当且仅当取值 0 或 1 的概率都为 0.5，也就是变量取值 0 和 1 的概率一样大。此时模型的输出可以视为随机的，应用在样本集合中也就是 0 和 1 的样本一样多，此时进行预测的结果和抛硬币无异，这显然不是我们想要的。

而当基尼系数的取到最小值 0 时，可知有取值为 0 或 1 的概率中一个为 0，一个为 1，也就是变量取值是确定的，此时模型的输出是固定的，应用在样本集合中也就是只有 0 或 1 的样本，此时进行预测的结果是确定的，这才是我们希望得到的。

因此我们可以发现基尼系数越大表示样本 D 的不确定性越大，所以我们希望**随着树的生长，基尼系数以最快的速度下降**。

以下表的数据为例，变量 A 是连续变量，变量 B 是离散变量：

索引	变量 A	变量 B	标签
1	2.1	1	1
2	2.2	1	1
3	1.8	1	0

索引	变量 A	变量 B	标签
4	2.0	1	0
5	1.6	1	1
6	2.8	2	1
7	1.3	2	0
8	0.1	2	0
9	3.8	2	1
10	2.9	2	1
11	0.5	3	0
12	2.7	3	1
13	0.9	3	0
14	1.4	3	0
15	1.2	3	0

对离散变量

$$Gini(D, B=1) = \frac{5}{15} \times \left(1 - \left(\frac{3}{5}\right)^2 - \left(\frac{2}{5}\right)^2\right) + \frac{10}{15} \times \left(1 - \left(\frac{4}{10}\right)^2 - \left(\frac{6}{10}\right)^2\right) = \frac{12}{25}$$

$$Gini(D, B=2) = \frac{12}{25}$$

$$Gini(D, B=3) = \frac{32}{75} < \frac{36}{75} = \frac{12}{25}$$

于是我们将 B 是否等于 3 作为划分条件生成两个子节点，或者直接将 B 作为叶节点，B 等于 3 则预测 0，B 不等于 3 则预测 1.

对连续变量 连续变量的情况就会稍复杂一些，这由连续变量的特点决定，连续变量是稠密的，很可能一个样本集合内的某一连续变量没有两个相同的值，此时划分按照变量是否等于某个值进行则极为不合理，于是划分取值则按照变量是否小于（小于等于）某个值进行。这和大于等于（大于）是完全等价的。

决策树的特点决定，决策树从根节点到叶节点的划分变量和划分取值都**以使基尼系数下降最大为目标**，这意味着决策树是确定的，每一个划分都是最优的。而对于连续变量的最优划分，则需要遍历连续变量的每个取值（大多数情况下也是遍历整个样本），然后选出最优划分条件。

以表中数据为例，连续变量 A 的最优划分条件是 $I\{A \leq 2.0\}$（我是这么设计的...）

$$Gini(D, A \leq 2.0) = \frac{9}{15}\left(1 - \left(\frac{1}{9}\right)^2 - \left(\frac{8}{9}\right)^2\right) + \frac{6}{15}\left(1 - \left(\frac{6}{6}\right)^2 - \left(\frac{0}{6}\right)^2\right) = \frac{16}{135}$$

停止 每次划分节点时，使用一个变量，在此节点的子树下，当前划分变量不会再次出现，当划分变量用完时树的生长就会自然停止。

这样生成的树极有可能会过拟合，有两种解决方法：一是早停法，即是不把所有变量用来生成划分条件，只选择部分变量进行划分，然而这样生成的决策树不具有确定性，因为随机种子不同，选择的变量也不同；另一种解决方法则是对决策树进行剪枝。

1.2.4 4. 剪枝

剪枝相关还不熟练，暂时跳过。

1.3 三、应用

直接上真实数据集可能比较玄学，先用先前的数据集进行验证：

In [1]:
```
import copy
import numpy as np
import pandas as pd
from scipy import stats

test_data = pd.DataFrame({
    'A':[2.1, 2.2, 1.8, 2.0, 1.6, 2.8, 1.3, 0.1, 3.8, 2.9, 0.5, 2.7, 0.9, 1.4, 1.2],
    'B':[1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3],
    'label':[1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0]
})
test_data
```

Out[1]:

	A	B	label
0	2.1	1	1
1	2.2	1	1
2	1.8	1	0
3	2.0	1	0
4	1.6	1	1
5	2.8	2	1
6	1.3	2	0
7	0.1	2	0
8	3.8	2	1
9	2.9	2	1
10	0.5	3	0
11	2.7	3	1
12	0.9	3	0
13	1.4	3	0
14	1.2	3	0

In [2]:
```
def Gini(data, labels, index, target='label'):
    data = data.loc[index]
```

```
            n = data.shape[0]

            gini = 1

            for label in labels:
                p = np.sum(data[target] == label)/n
                gini -= p*p

            return gini

        print(Gini(data=test_data, labels=[0, 1], index=np.linspace(0, 14, 15)))
```

```
0.4977777777777777
```

```
In [3]: def conditional_gini(data, labels, split_variable,
                               split_value, target='label', discrete=True):
            n = data.shape[0]
            if discrete:
                index1 = data[data[split_variable]==split_value].index
                index2 = data[data[split_variable]!=split_value].index

                gini1 = Gini(data=data, labels=labels, index=index1, target=target)
                gini2 = Gini(data=data, labels=labels, index=index2, target=target)

                gini = len(index1)/n*gini1 + len(index2)/n*gini2

                return gini
            else:
                index1 = data[data[split_variable]<=split_value].index
                index2 = data[data[split_variable]>split_value].index

                gini1 = Gini(data=data, labels=labels, index=index1, target=target)
                gini2 = Gini(data=data, labels=labels, index=index2, target=target)

                gini = len(index1)/n*gini1 + len(index2)/n*gini2

                return gini

        print(conditional_gini(data=test_data, labels=[0,1],
                               split_variable='B', split_value=3))
```

```
        print(32/75)
        print(conditional_gini(data=test_data, labels=[0,1],
                               split_variable='A', split_value=2, discrete=False))
        print(16/135)
```

```
0.4266666666666665
0.4266666666666667
0.11851851851851854
0.11851851851851852
```

```
In [4]: def split_value(data, variable, target, discrete=True):
            labels = np.unique(data[target])
            #data = data.loc[index]
            gini = 1

            if discrete:
                values = np.unique(data[variable])
            else:
                values = data[variable].copy()
            split_value = values[0]
            for value in values:
                tmp = conditional_gini(data=data, labels=labels, split_variable=variable,
                                       split_value=value, target=target, discrete=discrete)
                if tmp < gini:
                    gini = tmp
                    split_value = value
                else:
                    continue

            return gini, split_value

        a, b = split_value(data=test_data, variable='A', target='label', discrete=False)
        print(split_value(data=test_data, variable='A', target='label', discrete=False))
        print(split_value(data=test_data, variable='B', target='label', discrete=True))
        a, b
```

```
c:\users\37922\appdata\local\programs\python\python36\lib\site-packages\ipykernel_launcher.py:8: R
```

```
(0.11851851851851854, 2.0)
```

```
(0.4266666666666665, 3)
```

Out[4]: (0.11851851851851854, 2.0)

In [5]:
```
def split_variable(data, index, target, variable_set):
    gini = 1
    #data = data.loc[index]
    to_split_value = None
    to_split_variable = None

    tmp_gini = None
    tmp_value = None
    var_type = None

    for dtype in variable_set:
        discrete = dtype=='discrete'
        for variable in variable_set[dtype]:
            tmp_gini, tmp_value = split_value(data, variable,
                                              target, discrete)
            #print(tmp_gini)
            #print(tmp_value)
            if gini > tmp_gini:
                gini = tmp_gini
                to_split_value = tmp_value
                to_split_variable = variable
                var_type = discrete

    return gini, to_split_variable, to_split_value, var_type

variable_set = {
    'discrete':['B'],
    'continuous':['A']
}
split_variable(data=test_data, index=np.linspace(0, 14, 15),
               target='label', variable_set=variable_set)
```

```
c:\users\37922\appdata\local\programs\python\python36\lib\site-packages\ipykernel_launcher.py:8: R
```

Out[5]: (0.11851851851851854, 'A', 2.0, False)

这次选的数据集是汽车数据集：http://archive.ics.uci.edu/ml/datasets/Car+Evaluation
一共有六个变量：

- buying：购买的价格，有四个取值，vhigh、high、med、low
- maint：保养的价格，有四个取值，vhigh、high、med、low
- doors：门的数量，有四个取值，2、3、4、5more
- persons：载荷人数，有三个取值，2、4、more
- lug_boot：后备箱的尺寸，有三个取值，small、med、big
- safety：安全性，有三个取值，low、med、high

还有一个标签：

- acceptability：可接受程度，有四个取值，unacc、acc、good、vgood

就直觉而言，门的数量和载荷人数按照连续变量进行处理更为合适，也有助于演示连续变量的划分。

```
In [6]: title = ['buying','maint','doors','persons','lug_boot','safety','acc']
        car = pd.read_csv('car.data', header=None, names=title)
        car.head()
```

Out[6]:

	buying	maint	doors	persons	lug_boot	safety	acc
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

计算训练集和测试集的基尼系数

```
In [7]: car = car.replace(['more', '5more'], 6)
        n = car.shape[0]

        np.random.seed(2099)
        index = np.random.permutation(n)
        train_index = index[0: int(0.7*n)]
        test_index = index[int(0.7*n): n]

        labels = np.unique(car['acc'])
        print(Gini(data=car, labels=labels, index=train_index, target='acc'))
        print(Gini(data=car, labels=labels, index=test_index, target='acc'))
```

```
0.46404653272499263
0.4407096795749942
```

计算以 buying 为划分变量，'vhigh' 为划分值的基尼系数

```
In [8]: conditional_gini(car.loc[train_index], labels,
                         split_variable='buying', split_value='vhigh', target='acc')
```

```
0.45587846574195173
```

寻找以 buying 为划分变量时最好的划分值

```
In [9]: split_value(data=car.loc[train_index], variable='buying',
                    target='acc', discrete=True)
```

```
Out[9]: (0.45587846574195173, 'vhigh')
```

```
In [10]: variable_set={
             'discrete':['buying', 'maint', 'lug_boot', 'safety'],
             'continuous':['doors', 'persons']
         }
```

```
In [11]: car['doors'] = car['doors'].astype('int')
         car['persons'] = car['persons'].astype('int')

         split_variable(data=car, index=train_index,
                        target='acc', variable_set=variable_set)
```

```
c:\users\37922\appdata\local\programs\python\python36\lib\site-packages\ipykernel_launcher.py:8: R
```

```
Out[11]: (0.386157126093107, 'safety', 'low', True)
```

定义 *build_tree* 递归生成一棵树，遇到困难的地方就是传递可分割变量的字典变量，因为我们按照连续和离散划分变量，针对不同类型的变量划分方法也不一样。离散是按是否等于某个值进行判断，而连续变量则是按是否小于等于某个值，因此我们传递的可划分变量是以字典组织的。

而 python 传递实例时是传地址，因此一个节点对字典进行删除元素的操作会对所有的所有的节点产生影响。

一般这样处理是用 copy() 方法即可，但对字典并不可行，于是经过查阅发现 deepcopy() 可行。

```
In [41]: def build_tree(data, index, variable_set, tree, target, node=1):
             """
```

```
    data: all the data
    index: the data on which shall be splited, data won't change in recursion but index
    variable_set: variables to be split
    tree: a dataframe with node,split_variable,split_value,discrete,leaf,left and right p
    target: the label variable
    node: the node of the tree
    """
    tmp = data.iloc[index]
    leaf = len(variable_set[str(node)]['discrete'])==0
    leaf = leaf and len(variable_set[str(node)]['continuous'])==0
    leaf = leaf or len(np.unique(tmp[target]))==1

    print(str(node)+":"+str(leaf))

    if not leaf:
        gini, variable, value, discrete = split_variable(data, index, target,
                                                          variable_set[str(node)])
        prediction = stats.mode(tmp[target])[0][0]
        if discrete:
            variable_set[str(node)]['discrete'].remove(variable)
            variable_set[str(node*2)] = copy.deepcopy(variable_set[str(node)])
            variable_set[str(node*2+1)] = copy.deepcopy(variable_set[str(node)])
            #variable_set['continuous'].remove(variable)
            #left_set = variable_set.copy()
            #right_set = variable_set.copy()

            index1 = tmp[tmp[variable] == value].index
            left_prediction = stats.mode(data.iloc[index1][target])[0][0]
            index2 = tmp[tmp[variable] != value].index
            right_prediction = stats.mode(data.iloc[index2][target])[0][0]
        else:
            variable_set[str(node)]['continuous'].remove(variable)
            variable_set[str(node*2)] = copy.deepcopy(variable_set[str(node)])
            variable_set[str(node*2+1)] = copy.deepcopy(variable_set[str(node)])
            #variable_set['continuous'].remove(variable)
            #left_set = variable_set.copy()
            #right_set = variable_set.copy()

            index1 = tmp[tmp[variable] <= value].index
```

```
                    left_prediction = stats.mode(data.iloc[index1][target])[0][0]
                    index2 = tmp[tmp[variable] > value].index
                    right_prediction = stats.mode(data.iloc[index2][target])[0][0]

                tree.loc[node] = [variable, value, discrete, left_prediction, right_prediction]

                build_tree(data=data, index=index1, variable_set=variable_set,
                           tree=tree, target='acc', node=node*2)
                build_tree(data=data, index=index2, variable_set=variable_set,
                           tree=tree, target='acc', node=node*2+1)


        tree = {
            'split_variable':[None],
            'split_value':[None],
            'discrete':[None],
            'left_prediction': [None],
            'right_prediction':[None]
        }
        tree = pd.DataFrame(tree)
        variables={
            'discrete':['buying', 'maint', 'lug_boot', 'safety'],
            'continuous':['doors', 'persons']
        }
        variable_set={
            '1':variables
        }
        build_tree(data=car, index=train_index, variable_set=variable_set,
                   tree=tree, target='acc')
```

```
1:False
```

```
c:\users\37922\appdata\local\programs\python\python36\lib\site-packages\ipykernel_launcher.py:8: R
```

```
2:True
3:False
6:True
7:False
```

```
14:False
28:True
29:False
58:False
116:True
117:True
59:False
118:True
119:True
15:False
30:False
60:False
120:True
121:True
61:False
122:True
123:True
31:False
62:False
124:True
125:True
63:False
126:True
127:True
```

定义预测的函数为 *climb*（爬树嘻嘻），主要作用是按照树的条件进行查询和判断。

```
In [64]: def climb(data, tree):
             n = data.shape[0]
             prediction = pd.Series()
             for i in range(n):
                 tmp = data.iloc[i]
                 leaf = False
                 node = 1
                 while not leaf:
                     variable = tree.loc[node]['split_variable']
                     discrete = tree.loc[node]['discrete']
                     if discrete:
                         left = tmp[variable] == tree.loc[node]['split_value']
                     else:
```

```
                    left = tmp[variable] <= tree.loc[node]['split_value']
                if left:
                    prediction.loc[i] = tree.loc[node]['left_prediction']
                    node = node*2
                else:
                    prediction.loc[i] = tree.loc[node]['right_prediction']
                    node = node*2+1
                leaf = not (node in tree.index)
            return prediction

        prediction = climb(car.iloc[test_index], tree)
```

```
In [72]: prediction.index = test_index
```

```
        rate = np.sum(prediction == car.iloc[test_index]['acc'])/len(prediction)
        print('the accuracy rate is: '+str(rate))
```

```
the accuracy rate is: 0.8342967244701349
```

```
In [53]: tree
```

Out[53]:

	split_variable	split_value	discrete	left_prediction	right_prediction
0	None	None	None	None	None
1	safety	low	True	unacc	unacc
3	persons	2	False	unacc	acc
7	buying	vhigh	True	unacc	acc
14	maint	vhigh	True	unacc	acc
29	lug_boot	small	True	unacc	acc
58	doors	2	False	unacc	unacc
59	doors	2	False	acc	acc
15	maint	vhigh	True	acc	acc
30	lug_boot	small	True	unacc	acc
60	doors	2	False	unacc	unacc
61	doors	2	False	acc	acc
31	lug_boot	small	True	acc	acc
62	doors	2	False	unacc	acc
63	doors	2	False	acc	acc

```
In [ ]:
```