Python编程与人工智能实践

算法篇：CART分类树

(Classification And Regression Tree)

于泓
鲁东大学
信息与电气工程学院

2019.11.13

CART 树

- CART(Classification And Regression Tree)算法采用一种二分递归分割的技术，将当前 的样本集分为两个子样本集，使得生成的子节点都有两个分支。因此，CART算法生成的决策树是结构简洁的二叉树

CART分类树算法

1 按照**基尼增益最大**的原则，从**某一个特征维度** (ind_fea) 中寻找一个**阈值** **threshold** 把数据分为**2簇 （left，right)**

2 递归建立决策树，直到叶子节点内样本种类单一，
或者叶子节点内样本的数目小于阈值

当叶子节点内样本类别不单一时，
选取类别最多的类别作为叶子节点标签进行输出

基尼系数：表示数据集的纯度，越小数据集越纯

当集和内只有一类数据时 基尼系数为0

$$Gini(D) = \sum_{i=1}^{n} p(x_i) * (1 - p(x_i)) = 1 - \sum_{i=1}^{n} p(x_i)^2$$

$p(x_i)$ 是分类 x_i 出现的概率，

1.5	Iris-setosa
5.6	Iris-virginica
1.3	Iris-setosa
5.1	Iris-virginica
3.9	Iris-versicolor
4.5	Iris-versicolor
4.7	Iris-versicolor
1.5	Iris-setosa
1.5	Iris-setosa
1.3	Iris-setosa

用4 作为阈值 将数据分为2组

<4

1.5	Iris-setosa
1.3	Iris-setosa
3.9	Iris-versicolor
1.5	Iris-setosa
1.5	Iris-setosa
1.3	Iris-setosa

$1 - ((5/6)^2+(1/6)^2+(0/6)^2)$

=0.28

>4

5.6	Iris-virginica
5.1	Iris-virginica
4.5	Iris-versicolor
4.7	Iris-versicolor

$1 - ((2/4)^2+(2/4)^2+(0/4)^2)$

=0.5

基尼增益 $Gini(D) - \left(\frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2) \right)$

1.5	Iris-setosa
5.6	Iris-virginica
1.3	Iris-setosa
5.1	Iris-virginica
3.9	Iris-versicolor
4.5	Iris-versicolor
4.7	Iris-versicolor
1.5	Iris-setosa
1.5	Iris-setosa
1.3	Iris-setosa

$1 - ((5/10)^2+(2/10)^2+(3/10)^2)$

$= 0.62$

<4

1.5	Iris-setosa
1.3	Iris-setosa
3.9	Iris-versicolor
1.5	Iris-setosa
1.5	Iris-setosa
1.3	Iris-setosa

$1 - ((5/6)^2+(1/6)^2+(0/6)^2)$

$=0.28$

>4

5.6	Iris-virginica
5.1	Iris-virginica
4.5	Iris-versicolor
4.7	Iris-versicolor

$1 - ((2/4)^2+(2/4)^2+(0/4)^2)$

$=0.5$

基尼增益：**0.62 –(6/10)*0.28 – (4/10)*0.5** 越大越好

阈值选取?

1.5	Iris-setosa
5.6	Iris-virginica
1.3	Iris-setosa
5.1	Iris-virginica
3.9	Iris-versicolor
4.5	Iris-versicolor
4.7	Iris-versicolor
1.5	Iris-setosa
1.5	Iris-setosa
1.3	Iris-setosa

排序后，取相邻2数的中值，作为分割阈值

	1.3	Iris-setosa
1.3	1.3	Iris-setosa
1.4	1.5	Iris-setosa
1.5	1.5	Iris-setosa
1.5	1.5	Iris-setosa
2.2	3.9	Iris-versicolor
4.2	4.5	Iris-versicolor
4.6	4.7	Iris-versicolor
4.9	5.1	Iris-virginica
5.3	5.6	Iris-virginica

遍历所有的分割阈值，找到基尼增益最大的阈值，将数据分成2簇

代码实现:

```
# 从datas 的第 ind_fea 维特征中获取所有可能得分割阈值
def get_possible_splits(datas , ind_fea ):
    feas =datas[:,ind_fea]
    feas = np.unique(feas)
    feas = np.sort(feas)
    splits =[]
    for i in range(len(feas)-1):
        th = (feas[i]+feas[i+1])/2
        splits.append(th)

    return np.array(splits)
```

	1.3	Iris-setosa
1.3	1.3	Iris-setosa
1.4	1.5	Iris-setosa
1.5	1.5	Iris-setosa
1.5	1.5	Iris-setosa
2.2	3.9	Iris-versicolor
4.2	4.5	Iris-versicolor
4.6	4.7	Iris-versicolor
4.9	5.1	Iris-virginica
5.3	5.6	Iris-virginica

计算基尼系数

```
def gini_impurity( labs ):
    unique_labs = np.unique(labs)
    gini = 0
    for lab in unique_labs:
        n_pos = np.where(labs==lab)[0].shape[0]
        prob_pos = n_pos/len(labs)
        gini += prob_pos**2

    gini = 1-gini
    return gini
```

计算基尼增益

$$Gini(D) - \left(\frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2) \right)$$

```
# 利用 split 对 datas 的 ind_fea 维进行分割
# 计算该分割的基尼增益
def eval_split(datas,labs,ind_fea,split):
    mask = datas[:,ind_fea]<=split
    index_l = np.where(mask==0)[0]
    index_r = np.where(mask==1)[0]
    labs_l = labs[index_l]
    labs_r = labs[index_r]

    weight_left = float(len(labs_l)/len(labs))
    weight_right = 1- weight_left

    gini_parent = gini_impurity(labs)
    gini_left = gini_impurity(labs_l)
    gini_right = gini_impurity(labs_r)

    weighted_gini = gini_parent - (weight_left*gini_left + weight_right*gini_right)

    return weighted_gini
```

节点类

```python
class node:
    def __init__(self, datas, labs, parent):
        self.parent = parent
        self.datas = datas
        self.labs = labs

        # 当前节点的gini纯度
        self.gini = gini_impurity( self.labs )

        # tree nodes left and right
        self.left = None
        self.right = None

        # 当前节点的分割条件
        self.splitting_ind_fea = None
        self.threshold = 0

        # set leaf parameters to None
        self.leaf = False
        self.label = None
        self.confidence = None
```

```python
# 设置当前节点的分割条件
def set_splitting_criteria( self, ind_fea, threshold ):
    self.splitting_ind_fea = ind_fea
    self.threshold = threshold

# stopping_sz 剩下的数据小于stopping_sz 停止分割
def is_leaf( self, stopping_sz ):
    if len(self.labs) <= stopping_sz or self.gini == 0.0:
        return True
    else:
        return False
```

```
# 找到当前节点 最佳的分割 ind_fea 以及其相应的阈值
def find_splitting_criterion( self ):

    max_score = -1.0

    best_ind_fea = None
    threshold = 0.0

    N_fea = np.shape(self.datas)[-1]

    for i in range(N_fea):
        splits = get_possible_splits( self.datas, i )

        for split in splits:
            split_score = eval_split( self.datas, self.labs, i, split)
            if split_score > max_score:
                max_score = split_score
                best_ind_fea = i
                threshold = split

    return max_score, best_ind_fea, threshold
```

```python
# 对当前的节点进行分割
def split( self, ind_fea, threshold ):

    mask = self.datas[:,ind_fea]<=threshold
    index_l = np.where(mask==1)[0]
    index_r = np.where(mask==0)[0]
    labs_l = self.labs[index_l]
    labs_r = self.labs[index_r]
    datas_l = self.datas[index_l,:]
    datas_r = self.datas[index_r,:]

    print("Splitting %d samples into %d and %d samples by %d th =%.2f"% \
          (len(self.labs),len(labs_l),len(labs_r),ind_fea,threshold))

    left = node( datas_l , labs_l,self )
    right = node(datas_r , labs_r,self )

    return left, right


# 将当前节点设为叶子节点
def set_as_leaf ( self ):

    # set leaf parameters
    self.leaf = True
    # 设置该节点的标签为，所剩数据中标签最多的数据
    labs = self.labs.tolist()
    self.label =  max(labs,key=labs.count)
    n_pos= len(np.where(self.labs == self.label)[0])
    self.confidence = float( n_pos/len(self.labs))
```

```python
class tree:

    def __init__( self, datas, labs ,stopping_sz ):

        self.root = None
        self.datas = datas
        self.labs = labs
        self.stopping_sz = stopping_sz
```

```python
    def __build_tree( self, root ):

        # 如果是叶子节点则返回
        if root.is_leaf(self.stopping_sz):
            root.set_as_leaf()
            return

        # 找到最佳分割
        max_score, best_ind_fea, threshold = root.find_splitting_criterion()

        if best_ind_fea == None:
            return

        # 设置分割条件
        root.set_splitting_criteria( best_ind_fea, threshold )

        # 对当前节点进行分割
        left, right = root.split( best_ind_fea, threshold )
        root.left = left
        root.right = right

        self.__build_tree(root.left)
        self.__build_tree(root.right)
        return
```

```
def fit( self ):
    if self.root == None:
        self.root = node( self.datas, self.labs, None )
        self.__build_tree(self.root)
```

预测部分

```
def predict ( self , sample ):

    current = self.root
    while ( not current.leaf ):
        # check for split criterion
        if sample[current.splitting_ind_fea] <= current.threshold:
            current = current.left
        else:
            current = current.right

    return current.label
```

把树保存成字典进行打印

```
    def __print_tree(self,root):

        if root.leaf:

            return(root.label)

        ret_Tree = {}
        str_root= 'dim%d th=%.2f'%(root.splitting_ind_fea, root.threshold)
        ret_Tree[str_root]={}

        str_left = "dim %d<%.2f"%(root.splitting_ind_fea, root.threshold)
        str_right = "dim %d>%.2f"%(root.splitting_ind_fea, root.threshold)

        ret_Tree[str_root][str_left] = self.__print_tree(root.left)
        ret_Tree[str_root][str_right] = self.__print_tree(root.right)
        return ret_Tree

    def print_tree(self):

        return self.__print_tree(self.root)
```

测试部分

```
import numpy as np
from cart import tree
def indexSplit(N,train_ratio):
    N_train = int(N*train_ratio)
    index_random = np.random.permutation(N)
    index_train = index_random[:N_train]
    index_test = index_random[N_train:]

    return index_train,index_test


if __name__ == "__main__":
    # iris 数据处理
    file_data = 'iris.data'

    # 数据读取
    datas = np.loadtxt(file_data,dtype = float, delimiter = ',',usecols=(0,1,2,3))
    labs = np.loadtxt(file_data,dtype = str, delimiter = ',',usecols=(4))
    N,D = np.shape(datas)


    # 分为训练集和测试集和
    index_train,index_test = indexSplit(N,train_ratio=0.6)
```

数据分割过程

```
train_datas = datas[index_train,:]
train_labs = labs[index_train]


test_datas = datas[index_test,:]
test_labs = labs[index_test]

stopping_sz = 1

decision_tree_classifier = tree( train_datas, train_labs,stopping_sz )
decision_tree_classifier.fit()
ret_tree = decision_tree_classifier.print_tree()
print(ret_tree)
```

Splitting 90 samples into 32 and 58 samples by 2 th =2.45
Splitting 58 samples into 27 and 31 samples by 3 th =1.65
Splitting 27 samples into 26 and 1 samples by 0 th =7.10

生成的树结构

{'dim2 th=2.45': {'dim 2<2.45': 'Iris-setosa', 'dim 2>2.45': {'dim3 th=1.65': {'dim 3<1.65': {'dim0 th=7.10': {'dim 0<7.10': 'Iris-versicolor', 'dim 0>7.10': 'Iris-virginica'}}, 'dim 3>1.65': 'Iris-virginica'}}}}

```
{'dim2 th=2.45’:
            {'dim 2<2.45’:  'Iris-setosa’,
            ‘ dim 2>2.45’:
                        {'dim3 th=1.65’:
                                    {'dim 3<1.65’:
                                                {'dim0 th=7.10’:
                                                      {'dim 0<7.10': 'Iris-versicolor’,
                                                       'dim 0>7.10': 'Iris-virginica'}},
                         'dim 3>1.65': 'Iris-virginica'}}}}
```

测试部分

```python
n_right =0
for i in range(test_datas.shape[0]):
    prediction = decision_tree_classifier.predict(test_datas[i])

    if prediction == test_labs[i]:
        n_right = n_right+1


    print(prediction,test_labs[i])

print("acc = %.2f%%"%(n_right*100/len(test_labs)))
```

```
acc = 96.67%
```