机器学习上机报告

林远钊 *1700010672*

2019 年 4 月 12 日

目录

1 支持向量机方法 Review

基本介绍 决策树（Decision Tree）是一种基本的分类与回归方法，本文主要讨论分类决策树。决策树模型呈树形结构，在分类问题中，表示基于特征对数据进行分类的过程。它可以认为是 if-then 规则的集合。每个内部节点表示在属性上的一个测试，每个分支代表一个测试输出，每个叶节点代表一种类别。

划分选择 包括，信息熵、信息增益率和基尼系数，其分别对应三种主要的算法。主要的算法包括：ID3：信息增益 $Gain(D, A)$，C4.5：信息增益率 $Gain\_Ratio$，CART：基尼系数 $Gini\_index$

过拟合与剪枝 决策树容易出现过拟合状况，因为它们可以通过精确的描述每个训练样本的特征而构建出复杂的决策树，从而忽略了一般性的真实关联关系；可以通过减枝来消除过拟合；决策树剪枝的基本策略有“预剪枝”和“后剪枝”，预剪枝是在决策树生成的过程中，对每个结点在划分前先进行估计，若当前节点的划分不能带来决策树泛化性能提升，则停止划分并将当前节点标记为叶结点；后剪枝则是从训练集中先生成一颗完整的决策树，然后自底向上地对非叶结点进行考察，若将该结点对应的子树替换成叶结点能带来决策树泛化性能提升，则将该子树替换成叶结点。

2 基本实现思路

考虑递归地实现决策树。在决策树的基本算法中，有三种基本情况：

1. 当前节点包含的样本属于同一类别，无需划分
2. 当前属性集为空集，或是所有样本在所有属性上取值相同，无法划分
3. 当前结点包含的样本集合为空，不能划分

其对应的处理方法当为

i 直接将结点标记为他们所属的类别

ii 将其结点类别设定为所含样本最多的类别

iii 将其类别设定为父节点所含样本最多的类别

具体的伪代码在西瓜书 p74 即有给出，此处不再赘述。

3 python 实现

算法的思路基本遵从书中。以下首先对处理问题过程中的一些假设做一些说明，再叙述具体 python 实现的思路。

3.1 基本声明

在处理书中所给表格 4.2 和 4.3 中的数据时，我们将数据处理为.csv 文件形式，第一行为数据的属性集合，以下每行对应一个数据在这些属性上的取值。

3.2 实现思路

首先基于基本声明中的处理方式，需要先将所给数据转化成易于处理的形式。这里我们利用 numpy 和 pandas 包中的方法进行简单的转化，相应定义一个信息处理函数 loadData(filename)

```
"""默认csv文件第一行是列属性，以下每行是训练数据
并且数据label属性代指其所属的类"""
frame = pd.read_csv(filename)#读为DataFrame格式
label = frame['label']#获取标记
attrset = list(frame.columns)
attrset.remove('label')
f = frame.T#转置，以便获得Series对象，转化成dict类型
dataset = DataSet()
for i in range(frame.shape[0]):#遍历所有数据
        d = dict(f[i])
        del d['label']
        t = T_data(d, label[i])
        dataset.add(t)
return dataset, attrset #返回数据集(Dataset)和属性集(list)
```

考虑在决策树机器学习问题中的要素，发现对象包括数据，数据集合和决策树，而欲构建决策树，建立树的结点的类也是必须的。以下给出几个简单的类及其主要方法

```
#类：学习数据
class T_data:
        def __init__(self,data,label):
                """data must be a dictionary 特征 -> 数据"""

#类：数据集
class DataSet:
        def __init__(self):
                self.datas = []
```

```
10                  self.labels = dict()
11          def gini_index(self, a):
12          """calculate the gini_index using character a"""
13          def info_gain(self,a):
14          """calculater the information gain of character a"""
```

之后给出较为复杂的两个类，为了更好地实现决策树，需要结点类包含属性：

i 该结点包含的数据集 (dataset) 和未被作为分类依据的属性集 (attrset)

ii 该结点是否为叶节点 (leaf)，如果是，该结点中的数据当被视为生命类 (label)

iii 该结点的子结点为哪些，为了方便使用，这一属性被定义为，属性到结点的一个字典

iv 该结点处划分结点的依据是什么 (basis)

```
1  class Node:
2          def __init__
3          def set_as_leaf(self, label)
4                  self.leaf = True
5                  self.label = label
6          def recovery(self):
7                  self.leaf = False
8                  self.label = None
```

最后便是最重要的决策树类型，其中涉及到一系列的参数，包括：

1. 输入的数据集 (dataset) 和属性集 (attrset)
2. 决策树的划分判断标准 (criterion)，支持 entropy 和 gini（即信息增益和基尼系数）两种参数，默认值为 entropy（信息增益）
3. 是否进行剪枝 (pruning)，支持三种参数'un' 'pre' 'post'，分别代表不剪枝，预剪枝和后剪枝，默认值为'un'
4. 若进行剪枝操作，需提供验证集数据 (testData)

```
1  class DesicionTreeClassifier:
2          def __init__(self, dataset, attrset, criterion="entropy"):
3          """criterion代表分类准则，有entropy和gini两个选项"""
4                  self.critertion = critertion
5                  self.root = Node(dataset, attrset, name='task')
6          def generateTree(self, pruning='un',testData=None):...
```

预计的调用方式是（模仿机器学习 python 包 sklearn）：

```
dataset, attrset = loadData("4.2_data.csv")
testdataset, testattrset = loadData('4.2_test.csv')
clf = DesicionTreeClassifier(dataset, attrset)
clf.generateTree(pruning='un')
clf.export_tree('4.2_un.dot')
```

真正生成决策树的过程是在调用 DesicionTreeClassifier 的 generateTree 方法时完成的，这段代码的实现思路和书中伪代码几乎是相同的。

```
def gen_iter(node, pru, tD=testData):
        if_pruning = pru
        dataset = node.dataset
        attrset = node.attrset
        if if_pruning =='pre':#若需预剪枝，记录初始验证率
                node.set_as_leaf(label=self.best_label(node))
                ori_rate = self.clarrify_rate(tD)
                node.recovery()

        if len(dataset.labels) == 1: #样本全属一类，情形1
                label = list(dataset.labels.keys())[0]
                node.set_as_leaf(label)
                return
        elif not attrset or same_in_A(dataset, attrset):
        #对应之前叙述的情形2
                label = self.best_label(node)
                node.set_as_leaf(label)
                return
        attr, value, Dv = self.best_split(node)
        node.set_basis(attr)
        new_attrset = attrset.copy()
        new_attrset.remove(attr)
        if if_pruning == "un" or if_pruning == "post":
                for av in Dv:
                        ds = Dv[av]
                        av_node = Node(ds, new_attrset, name=av)
                        node.set_child(av_node, av)
                        gen_iter(av_node, pru=if_pruning,tD=testData)
                        #此处进行迭代
        elif if_pruning == 'pre':
                ...#(省略相似代码)
```

```
32                  new_rate = self.clarrify_rate(tD)
33                  if ori_rate > new_rate:#泛化能力下降，剪枝
34                          node.set_as_leaf(self.best_label(node))
35                          node.child = dict()
36                          return
37                  else:#泛化能力上升，分叉
38                          for av in node.child:
39                                  av_node = node.child[av]
40                                  av_node.recovery()
41                          gen_iter(av_node, pru = if_pruning, tD=testData)
42
43          if (pruning == 'pre' or pruning == 'post') and testData is None:
44                  raise TestDataError
```

如果是后剪枝，需在决策树完成后进行深度优先的剪枝

```
1  gen_iter(self.root,pru=pruning,tD=testData)
2  if pruning == 'post':
3          ori_rate = self.clarrify_rate(testData)
4          print(ori_rate)
5                  dfs = []
6          def postorder(node):
7                  if node.leaf:#叶节点不进入dfs序列
8                          return
9                  nonlocal dfs
10                 for x in node.child:
11                         ch_node = node.child[x]
12                         postorder(ch_node)
13                         dfs.append(node)
14         postorder(self.root)
15         for post_node in dfs:
16                 post_node.set_as_leaf(self.best_label(post_node))
17                 new_rate = self.clarrify_rate(testData)
18                 if new_rate > ori_rate:
19                         post_node.child = dict()
20                         ori_rate = new_rate
21                 else:
22                         post_node.recovery()
```

3.3 输出形式

在 class DesicionTreeClassifier 类中，还定义了 export_tree 函数（此也为模仿 python 机器学习包 sklearn 的函数），将决策树按照.dot 文件格式输出，输出形式能够直接被 Graphiz 在 cmd 命令行中处理。

（ps：但是 Graphviz 包似乎无法很好地处理中文的情况，输出总是乱码，因而在此无法附上树图，只能附上 dot 文件，从文件中能比较容易地验证决策树的正确性）

4 运行结果

为了一定程度上先验证决策树的正确性，在代码完成后先对书中已给出的示例用相同的数据和策略进行划分，验证包括：以增益率为标注的，对表 4.2 中数据的无剪枝、预剪枝和后剪枝的验证，都和书中给出的结论完全相同。（我的运行结果在压缩文件中的 4.2_un.dot 和 4.2_pre.dot 和 4.2_post.dot 可以看到。）

在验证了算法的正确性后，更改判定标准，给出了 4.2_gini.dot 文件中所示的决策树分类结果。并且对表 4.3 中的数据根据信息熵为划分准则的决策树给出了结果，记录在 4.3_gini.dot 文件中。

5 多变量问题

西瓜书中的 4.10 要求自己实现或下载一个多变量划分的决策树算法，观察算法在表 4.3 的学习结果。在此首先感谢孟响同学在此为我提供了完整的算法，我得以直接拿来主义，利用他的成果观察算法的学习效果。在此特别说明。

对数据处理 在这个学习算法中，将每个属性对应的取值都统一化为数字，因此方便比较容易地对所有数据进行同一的处理。数据的形式是一个二位的 numpy.ndarray 数组。附件图 (try.png)

```
x_sha = x.shape
tra = int(x_sha[0]*(1-test_ratio))
labe = [10, 1, 14, 13, 11, 15, 6, 4, 12, 5, 8, 9, 16, 3, 7, 0, 2]
x_train = x[labe[0:tra],:]
x_test = x[labe[tra:],:]
y_train = y[labe[0:tra]]
y_test = y[labe[tra:]]
root = multi_decision_tree(x_train, y_train, tot, yset,whemul)
test_all = x_sha[0]-tra
success = 0
for i in range(tra,x_sha[0]):
    print(x[labe[i],:],y[labe[i]])
    if root.find_n(x[labe[i],:]) == y[labe[i]]:
        success += 1
print('purity is {}'.format(success/test_all))
```

```
[0.   0.   1.   0.   0.   0.   0.608 0.318] 1
[1.   1.   0.   0.   1.   0.   0.437 0.211] 1
[0.   0.   0.   0.   0.   0.   0.697 0.46 ] 1
[1.   0.   0.   0.   0.   0.   0.634 0.264] 1
purity is 1.0
```

图 1: Test

中给出了学习完毕后，对学习算法的验证（以纯度 (impurity) 的度量为依据）。

压缩文件中包含了所有的图片和源码。