ID3

算法实现及决策树可视化

徐遥 SA16214017

2017 年 3 月 30 日

目录

使用示例

Golf I

以经典的 Golf 决策树为例，展示所有功能。

```
# 利用 pd.read_csv 读取 csv 文件为 DataFrame
train_data = pd.read_csv('golf.csv')
# 初始化 ID3 算法时需要提供训练数据集以及目标属性
id3_solver = ID3(train_data, target='play')
# 进行训练
id3_solver.run()
# 输出训练的到的决策树
id3_solver.render_decision_tree('./dtree')
# 这里为了展示功能，所以直接把训练数据集当作了测试集
result = test_data['play'].values
test_data.drop('play', axis=1, inplace=True)
# 比较预测与实际结果获得正确率
predict = id3_solver.predict(test_data)
accuracy = id3_solver.score(predict, result)
```

Golf II

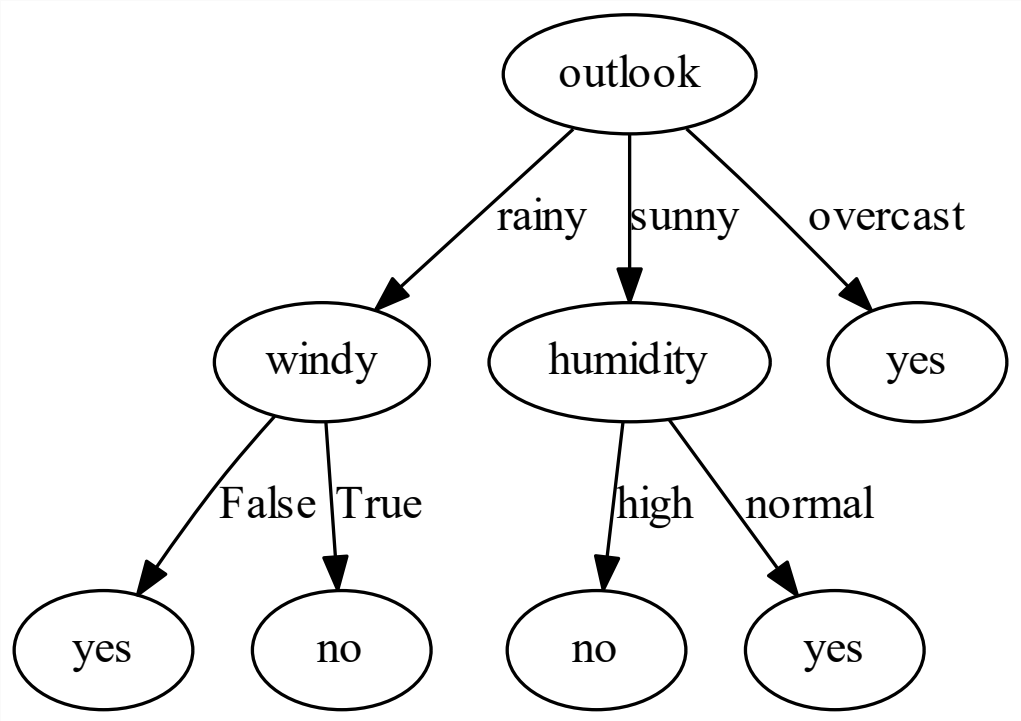


图 1: Golf 决策树

数据处理

Pandas

在构造决策树时，需要频繁地计算窗口下计算某属性的熵值。

$$\text{Entropy(A)} = \sum_{i=1}^{a} \text{Info(data[A])} \tag{1}$$

$$\text{Info} = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{2}$$

这就要求需要一种能够根据属性值快速筛选数据的数据结构。因此，我选择了Pandas (Python Data Analysis Library) 来处理数据。

- 具备按轴或显式数据对齐功能的数据结构；
- 数学运算和约简 (比如对某个轴求和) 可以根据不同元数据 (轴编号) 进行；
- 灵活的处理缺失数据；
- 具有多种文件格式的读取能力 (CSV、JSON、XLS 等等)；

DataFrame

最终使用的是二维数据结构 DataFrame(属性名即为列名，一个样本即为一行)。

```
1  import pandas as pd
2
3  golf_data = pd.read_csv('../../golf.csv')
4
5  # 筛选出 outlook 为 sunny 的数据
6  golf_data[golf_data['outlook']=='sunny']
7  # 查看 outlook 各个值所具有的数据个数
8  golf_data['outlook'].value_counts()
9  # 丢弃 outlook 列
10 golf_data.drop('outlook', axis=1, inplace=True)
```

熵的计算 I

有了 DataFrame 的帮助计算窗口下各个属性的熵值就变得非常简单。

```
1  class ID3(object):
2      def _entropy(self, data, attribute):
3          """ 计算某个 attribute 的熵, 私有
4          """
5          value_freq = data[attribute].value_counts()
6          data_entropy = 0.0
7          N = len(data)
8          #  Entropy(A) = ∑_{i=1}^{S} Info(data[A])
9          for value, freq in value_freq.items():
10             p = freq / N
11             data_entropy += p * self._info(data,
   ↪  attribute, value)
12         return data_entropy
```

熵的计算 II

```
13
14      def _info(self, data, attribute, attribute_value):
15          data = data[data[attribute] == attribute_value]
16          target_value_freq =
    ↪   data[self.target].value_counts()
17          data_info = 0.0
18          N = len(data)
19          # Info = ∑_{i=1}^{c} −p_i log_2 p_i
20          for freq in target_value_freq.values:
21              p = freq / N
22              data_info -= p * math.log(p, 2)
23          return data_info
```

决策树的构造与使用

Node

首先定义了 Node 类，每个节点拥有自己的属性以及子节点。

```
class Node(object):
    def __init__(self, _id):
        self.id = _id
        self.attribute = None
        self.branches = {}
```

其中，branches 为一个 dict，key 为当前节点属性所对应的值。如果节点的 branches 为空，则其 attribute 即为分类。

构造与预测

理想的建立决策树的步骤为：

1. 寻找当前窗口下熵值最小的属性，将该属性设为节点，并按其属性值进行数据分割；
2. 对这些子数据集重复执行上步骤，直到数据集为同一类别。

理想状态下，使用决策树预测数据的分类就更为简单了，只需要根据数据的属性值一步步从根节点走到叶子节点即可。

遇到的问题

在数据量较大，属性也较多时，再遵循理想的策略就会遇到问题。

构造

有可能遇到所有属性都已被用来划分数据了，但剩下的数据仍然不是同一种类。按照理想的步骤，没有办法继续了。

预测

在沿着决策树预测时，可能会遇到无路可走的情况，即当前节点没有测试数据相对应属性值的子节点。

为了解决问题，我采用了简单粗暴的策略：那个多选哪个！

最终实现 I

```
class ID3(object):
    def _make_decision_tree(self, data, node):
        # 当数据都为同一类数据时，直接返回
        if len(data[self.target].value_counts()) == 1:
            node.attribute =
 ↪  data[self.target].value_counts().index[0]
            return
        # 如果除了 target 外，已经没有其他 attribute 了，那也返回
        if len(data.columns) == 1:
            node.attribute =
 ↪  data[self.target].value_counts().argmax()
            return

        # 寻找熵最小的属性
        min_entropy = math.inf
        for attribute in data.columns:
            if attribute == self.target:
```

最终实现 II

```
16                  continue
17              temp_entropy = self._entropy(data, attribute)
18              if temp_entropy < min_entropy:
19                  min_entropy = temp_entropy
20                  node.attribute = attribute
21
22          # 建立子节点
23          for value in data[node.attribute].value_counts().index:
24              branch_data = data[data[node.attribute] == value]
25              branch_data = branch_data.drop(node.attribute,
    ↪  axis=1)
26              branch_node = self._new_node()
27              node.add_branch_node(value, branch_node)
28              self._make_decision_tree(branch_data, branch_node)
29
30      def run(self):
31          self.root_node = self._new_node()
32          self._make_decision_tree(self.data, self.root_node)
```

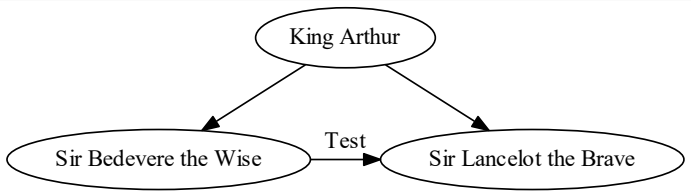
决策树的可视化

Graphviz

决策树的可视化利用了贝尔实验室开发的 Graphviz 工具包。用户可以使用 DOT 语言来描述图形，然后利用该工具进行图形的布局与绘制，省去手动调整元素的大小与局部的繁琐过程。对于决策树而言，我们只需要了解如何往有向图 (digraph) 添加节点与边。

```
// The Round Table
digraph {
        A [label="King Arthur"]
        B [label="Sir Bedevere the Wise"]
        L [label="Sir Lancelot the Brave"]
                A -> B
                A -> L
                B -> L [label=Test
 ↪    constraint=false]
}
```

(a) DOT 文件



(b) 渲染后

图 2: Graphviz 使用示例

Graphviz 的 Python 接口

graphviz 提供了创建 DOT 文件的 Pyhton 接口。实际上，图 2 就是其官方示例。

```python
# The round Table
from graphviz import Digraph
# Create a graph object
dot = Digraph(comment='The Round Table')
# Add nodes and edges
dot.node('A', 'King Arthur')
dot.node('B', 'Sir Bedevere the Wise')
dot.node('L', 'Sir Lancelot the Brave')
dot.edges(['AB', 'AL'])
dot.edge('B', 'L', constraint='false', label='Test')
# Save and render the source code
dot.render('../figures/graphviz_demo')
```

自动生成决策树的 DOT 文件 I

有了上述工具，决策树的可视化就变得非常简单了: 只需要递归地把所有节点和边加入有向图即可。

```
1  class ID3(object):
2      class Node(object):
3          @property
4          def node_name(self):
5              # 可视化时，每个 node，必须要有独一无二的 name
6              return ''.join([self.attribute,
   ↪  str(self.id)])
7
8          def add_to_graph(self, graph):
9              graph.node(self.node_name, self.__str__())
10             for edge_name, branch_node in
   ↪  self.branches.items():
```

自动生成决策树的 DOT 文件 II

```
11                  branch_node.add_to_graph(graph)
12                  graph.edge(self.node_name,
    ↪ branch_node.node_name, label=str(edge_name))
13
14      def render_decision_tree(self, filename):
15          if not self.root_node:
16              raise ValueError('Tree not decided!')
17          from graphviz import Digraph
18          dot_graph = Digraph(comment="Decision Tree")
19          self.root_node.add_to_graph(dot_graph)
20          dot_graph.render(filename)
```

生成树展示 I

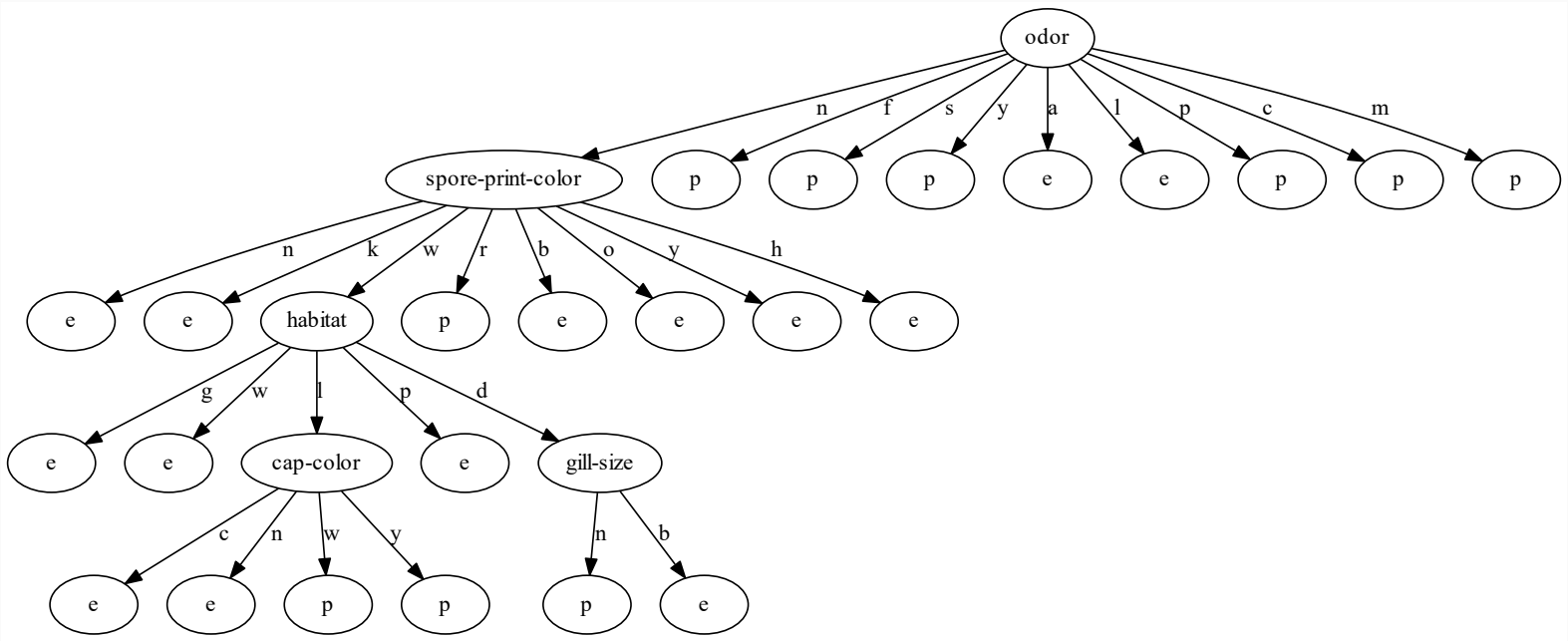


图 3: 蘑菇毒性分类决策树