中山大學
SUN YAT-SEN UNIVERSITY

实 验 报 告

人工智能实验报告–决策树

学　　院：数据科学与计算机学院
专　　业：计算机科学与技术
学生姓名：刘斯宇
学　　号：17341110

2019 年 9 月 10 日

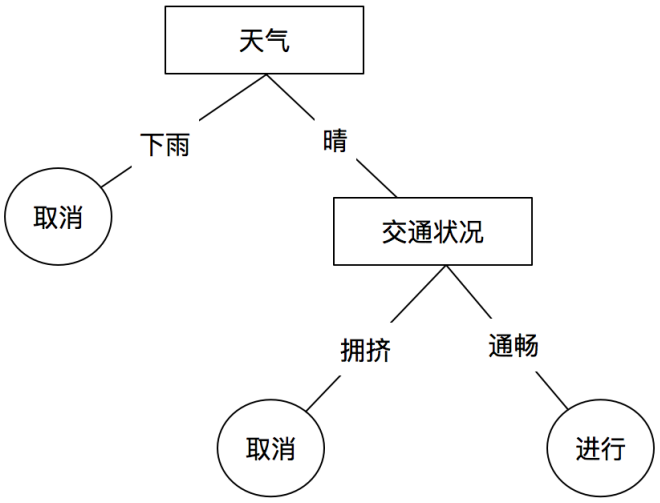
目录

1 实验原理

分类决策树模型是对实例进行分类的树形结构。由结点和有向边组成。其中内部结点表示一个属性，叶子节点表示分类的一个类，有向边是属性的分类依据。决策树的学习一般包括三个步骤：

* 特征选择
* 决策树的生成
* 决策树的剪枝

一个典型的决策树模型如下图所示：



这个图表示的含义是：根据天气情况和交通状况来决定某个活动是否进行。如果下雨，那么活动取消，如果天气是晴天，那么需要看交通状态，如果交通状况拥挤的话，活动也取消，否则举办活动。在介绍决策树的特征选择之前，我们先来介绍一些信息论里面的概念：

* 熵

  熵是表示随机变量不确定性的度量，设 X 是一个取有限个值的随机变量，其概率分布为

$$P(X = x_i) = p_i \ \ i = 1, 2, \ldots, n$$

  则随机变量 X 的熵定义为：

$$H(p) = -\sum_{i=1}^{n} p_i \log p_i$$

  熵的值越大，随机变量的不确定性越大。

* 条件熵

  条件熵 $H(Y|X)$ 表示在已知随机变量 X 的条件下随机变量 Y 的不确定性, 定义为 X 给

定条件下 Y 的条件概率分布的熵对 X 的数学期望

$$H(Y|X) = \sum_{i=1}^{n} p_i H(Y|X = x_i)$$

* 信息增益

信息增益表示得知特征 X 的信息而使得类 Y 的信息的不确定性减少的程度。特征 X 对训练数据集 D 的信息增益 $g(D, X)$，定义为集合 D 的经验熵 $H(D)$ 与特征 X 给定条件下 D 的经验条件熵 $H(D|A)$ 之差，即：

$$g(D, X) = H(D) - H(D|X)$$

一般的，我们也称熵 $H(Y)$ 与条件熵 $H(Y|X)$ 之间的差为互信息

1.1 特征选择

特征选择有 $ID3$ 算法、$C4.5$ 算法、$CART$ 算法三种不同的选择方式。

1.1.1 $ID3$ 算法

$ID3$ 算法是通过在决策树各个结点上应用信息增益准则选择特征。具体的做法是：对于每个结点计算所有可能的特征的信息增益，选择信息增益最大的特征作为结点的特征。当一个属性已经作为划分的依据，在后面就不再参与竞选了

· 信息增益的算法

1) 计算数据集 D 的经验熵 $H(D)$

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} \log_2 \frac{|C_k|}{|D|}$$

2) 计算特征 A 对数据集 D 的经验条件熵 $H(D|A)$

$$H(D|A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} H\left(D_i\right) = -\sum_{i=1}^{n} \frac{|D_i|}{D|} \sum_{k=1}^{K} \frac{|D_{ik}|}{|D_i|} \log_2 \frac{|D_{ik}|}{|D_i|}$$

3) 计算信息增益

$$g(D, A) = H(D) - H(D|A)$$

4) 选择信息增益最大的特征作为决策点

1.1.2 $C4.5$ 算法

C4.5 决策树的提出完全是为了解决 ID3 决策树的一个缺点，当一个属性的可取值数目较多时，那么可能在这个属性对应的可取值下的样本只有一个或者是很少个，那么这个时候它的

信息增益是非常高的，这个时候纯度很高，ID3 决策树会认为这个属性很适合划分，但是较多取值的属性来进行划分带来的问题是它的泛化能力比较弱，不能够对新样本进行有效的预测。**由于信息增益选择分裂属性的方式会倾向于选择具有大量值的属性（即自变量）**如对于客户 ID，每个客户 ID 对应一个满意度，即按此变量划分每个划分都是纯的（即完全的划分，只有属于一个类别），客户 ID 的信息增益为最大值 1。但这种按该自变量的每个值进行分类的方式是没有任何意义的。为了克服这一弊端，有人提出了采用增益率（GainRate）来选择分裂属性。而 C4.5 决策树则不直接使用信息增益来作为划分样本的主要依据，而提出了另外一个概念，增益率

· 信息增益率的算法

1) 计算特征 A 对数据集 D 的信息增益

$$g(D,A)=H(D)-H(D|A)$$

2) 计算数据集 D 关于特征 A 的值的熵 $H_A(D)$

$$H_A(D)=-\sum_{j=1}^{v}\frac{|D_j|}{|D|}\times\log\left(\frac{|D_j|}{|D|}\right)$$

3) 计算信息增益 $g_R(D,A)$

$$g_R(D,A)=\frac{g(D,A)}{H(D)}$$

4) 选择信息增益增益率最大的特征作为决策点

1.1.3 *CART* 算法

上面介绍的 ID3、C4.5 算法都是基于信息熵来进行划分节点选取的，主要用于分类问题。而 CART 决策树全称为分类回归树（Classification And Regression Tree），分类和回归问题都可以使用。

CART 和前面两种算法不同的地方是，在每一次节点做判断时，只考虑二分类的情况，即使征能够取到多个值（比如属性颜色有红、黄、蓝三种取值，ID3 和 C4.5 直接就划分为红、黄、蓝三个子类，而 CART 只能在一次划分时划分为是不是红（黄、蓝）然后再进行判断。）

· CART 算法

1) 计算特征 A 的条件下，数据集 D 的 $GINI$ 系数

$$gini(\mathrm{D},\mathrm{A})=\sum_{j=1}^{v}p\left(A_j\right)\times gini\left(D_j|A=A_j\right)$$

$$gini\left(D_j|A=A_j\right)=\sum_{i=1}^{n}p_i\left(1-p_i\right)=1-\sum_{i=1}^{n}p_i^2$$

2) 选择 $GINI$ 系数最小的特征作为决策点

1.2 决策树的生成

1.2.1 建树的方法

* 选择特征
  通过不同的策略来选取特征作为当前分类的标准。

* 创建结点
  按上一步选出来的策略按照不同的取值创建不同的结点。

* 划分数据
  按上一步选出来的策略将数据分类，将含有该策略一样值的数据归为对应的结点。并去除该特征。

* 递归建树
  对于每一个结点，回到特征选择建树，直到到达边界条件。

1.2.2 递归结束的条件

* 1
  Dataset 中的样本属于同一类别 C，则将当前结点标记为 C 类叶结点。(也就是说该数据集的所有的 lable 都是一样的)

* 2
  Attritute 为空集，或 Dataset 中所有样本在 Attritute 中所有特征上取值相同，此时无法划分。将当前结点标记为叶结点，类别为 Dataset 中出现最多的类。

* 3
  Dataset 为空集，则将当前结点标记为叶结点，类别为父结点中出现最多的类。(也就是说在训练集中的某个特征的数值没有出现过么，但是在测试集中出现)

1.3 决策树的剪枝

1.3.1 预剪枝

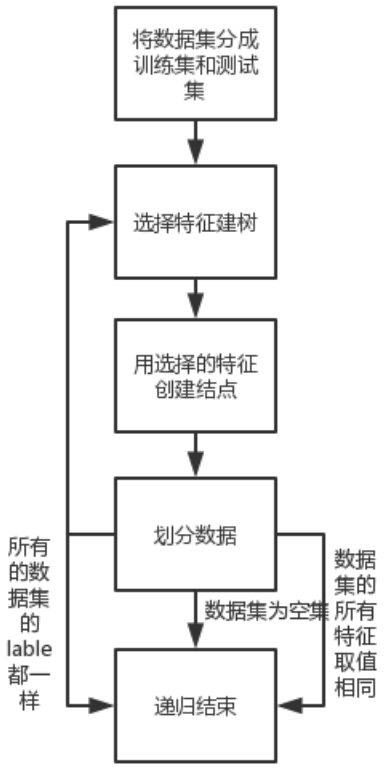
预剪枝：在构造决策树的同时进行剪枝。所有决策树的构建方法，都是在无法进一步降低熵的情况下才会停止创建分支的过程，为了避免过拟合，可以设定一个阈值，熵减小的数量小于这个阈值，即使还可以继续降低熵，也停止继续创建分支。但是这种方法实际中的效果并不好。

1.3.2 后剪枝

后剪枝是在决策树生长完成之后，对树进行剪枝，得到简化版的决策树。剪枝的过程是对拥有同样父节点的一组节点进行检查，判断如果将其合并，熵的增加量是否小于某一阈值。如

果确实小，则这一组节点可以合并一个节点，其中包含了所有可能的结果。后剪枝是目前最普遍的做法。后剪枝的剪枝过程是删除一些子树，然后用其叶子节点代替，这个叶子节点所标识的类别通过大多数原则 (majority class criterion) 确定。所谓大多数原则，是指剪枝过程中, 将一些子树删除而用叶节点代替, 这个叶节点所标识的类别用这棵子树中大多数训练样本所属的类别来标识, 所标识的类称为 majority class 。

2 流程图



3 关键代码截图

3.1 计算信息增益函数函数

方法：便利整个数据集，用上面的公式找出某个特征的信息增益。

```
1    def cal_information_gain(data_set,attribute_index):
2      """
```

```
3
4           data_set->
5
6       """
7       HD=0#
8       HDA=0#
9
10      label={}
11
12      num=len(data_set)
13
14      attribute_map={}
15      for data in data_set:
16          temp=data[-1]
17          if temp not in label.keys():
18              label[temp]=0
19          label[temp]+=1
20          attribute=data[attribute_index]
21          if attribute not in attribute_map.keys():
22              attribute_map[attribute]=0
23          attribute_map[attribute]+=1
24
25      #print(label)
26      for temp in label.items():
27          HD+=-temp[1]/num*math.log2(temp[1]/num)
28
29      for temp in attribute_map.items():
30          temp_map={}
31          length=0
32          for data in data_set:
33              if data[attribute_index]==temp[0]:
34                  length+=1
35                  if data[-1] not in temp_map.keys():
36                      temp_map[data[-1]]=0
37                  temp_map[data[-1]]+=1
38          #print(temp_map,length)
39          for f in temp_map.items():
40              HDA+=temp[1]/num*(-f[1]/length*math.log2(f[1]/length))
41      #print(HDA)
```

```
    return HD-HDA
```

3.2 计算信息增益率函数函数

方法：先计算信息增益，再计算熵。

```
def  cal_infor_gian_ratio(data_set,attribute_index):
    information_gain=cal_information_gain(data_set,attribute_index)

    splitinfo=0

    attribute_map={}
    for data in data_set:
        attribute=data[attribute_index]
        if attribute not in attribute_map.keys():
            attribute_map[attribute]=1
        else:
            attribute_map[attribute]+=1

    num=len(dataset)
    for attribute in attribute_map.keys():
        splitinfo+=-(attribute_map[attribute]/num)*math.log2(attribute_map[attribut
    return information_gain/splitinfo
```

3.3 计算 GINI 指数函数

方法：先计算出每种属性的频率，再根据 GINI 指数的公式计算 GINI 指数

```
def cal_GINI(data_set,attribute_index):
    attribute_list=[line[attribute_index] for line in data_set]
    attribute_set=set(attribute_list)
    num=len(data_set)

    gini=0

    for attribute in attribute_set:
        attribute_number=0

        key_map={}
        for line in data_set:
            if line[attribute_index]==attribute:
```

```
                attribute_number+=1
                if line[-1] not in key_map.keys():
                    key_map[line[-1]]=1
                else:
                    key_map[line[-1]]+=1
        temp_pos=1
        for k in key_map.keys():
            temp_pos-=np.power(key_map[k]/attribute_number,2)
        gini+=attribute_number/num*temp_pos

    return gini
```

3.4 建树方法

方法：递归式的建树。通过不同的特征选择函数递归建树。

```
         def create_tree(dataset,attribute):
#    print(attribute,len(dataset[0]))
    #                      D          A
    labellist=[line[-1] for line in dataset]
    #D                    C
    if labellist.count(labellist[0])==len(labellist):
        return labellist[0]
    #A      ,  D            A
    if len(dataset[0])==1:
        return get_themost(labellist)

    num=len(dataset[0])
    index=7
    temp=1000000

#    for i in range(num-1):
#        data=cal_information_gain(dataset,i)
#        data=cal_infor_gian_ratio(dataset,i)
#        if temp<data:
#            index=i
#            temp=data
#

    for i in range(num-1):
```

```
25          data=cal_GINI(dataset,i)
26  #        data=cal_infor_gian_ratio(dataset,i)
27          if temp>data:
28              index=i
29              temp=data
30
31      bestattribute=attribute[index]
32
33      Tree={bestattribute:{}}
34
35      attribute_values=[line[index] for line in dataset]
36
37      attribute_values=set(attribute_values)
38                  #D
39      Tree[bestattribute]['majority']=get_themost(labellist)
40      #print(bestattribute,attribute_values)
41      del(attribute[index])
42      for value in attribute_values:
43          newattribute=attribute[:]
44          Tree[bestattribute][value]=create_tree(renew_dataset(dataset,index,value),n
45      return Tree
```

3.5 带有剪枝的建树方法

方法：后剪枝，从后往前剪枝，因为是递归的建树，所以最后建的那些树先剪枝，所以是后剪枝。

```
1           def create_tree_new(dataset,attribute,testset):
2  #    print(attribute,len(dataset[0]))
3      #                        D          A
4      labellist=[line[-1] for line in dataset]
5      #D                     C
6      if labellist.count(labellist[0])==len(labellist):
7          return labellist[0]
8      #A       ,  D             A
9      if len(dataset[0])==1:
10         return get_themost(labellist)
11
12     num=len(dataset[0])
13     index=7
```

```
    temp=1000000

#    for i in range(num-1):
#        data=cal_information_gain(dataset,i)
#        data=cal_infor_gian_ratio(dataset,i)
#        if temp<data:
#            index=i
#            temp=data
#

    for i in range(num-1):
        data=cal_GINI(dataset,i)
#        data=cal_infor_gian_ratio(dataset,i)
        if temp>data:
            index=i
            temp=data

    bestattribute=attribute[index]

    Tree={bestattribute:{}}

    attribute_values=[line[index] for line in dataset]

    attribute_values=set(attribute_values)
    Tree[bestattribute]['majority']=get_themost(labellist)
    #print(bestattribute,attribute_values)
    tempattribute=attribute[:]
    del(attribute[index])
    for value in attribute_values:
        newattribute=attribute[:]
        Tree[bestattribute][value]=create_tree_new(renew_dataset(dataset,index,valu
    acc=0
    test_list=[line[-1] for line in testset]
    if(len(test_list)==0 or len(testset[0])<=1):
        return Tree

    for line in testset:
        if new_test(Tree,line,tempattribute)==line[-1]:
            acc+=1
```

```
53
54      acc2=test_list.count(get_themost(test_list))
55      #print(acc2)
56      if acc<acc2:
57          for value in attribute_values:
58              Tree[bestattribute][value]=get_themost(labellist)
59      return Tree
```

4 实验结果

这个实验我采用的是 7 折交叉验证

4.1 用信息增益来选择特征的实验结果

```
In [5]: runfile('/Users/liusiyu/Downloads/大三上/人工智能/Lab2 决策树/lab2_dataset/1.py'
剪枝前:accuracy 0.968812030075188
剪枝后:accuracy 0.9727017543859648
```

4.2 用信息增益率来选择特征的实验结果

```
In [4]: runfile('/Users/liusiyu/Downloads/大三上/人工智能/Lab2 决策树/lab2_dataset/1.py'
剪枝前:accuracy 0.9665263157894737
剪枝后:accuracy 0.9763508771929824
```

4.3 用基尼指数来选择特征的实验结果

```
In [7]: runfile('/Users/liusiyu/Downloads/大三上/人工智能/Lab2 决策树/lab2_dataset/1.py'
剪枝前:accuracy 0.9682406015037595
剪枝后:accuracy 0.9763508771929824
```

4.4 实验结果分析

从上面的图可以看出来，决策树在剪枝前和剪枝后其实准确率是差不多的，剪枝后准确率提升的幅度不算大，可能与这个题目本身有关吧，因为 label 只有两种情况，但是这个结果还是挺令人鼓舞的。

5 实验创新

5.1 剪枝

5.2 实验结果分析

这个实验我主要是做到了剪枝的创新。其实我之前是不懂怎么剪枝的，看到 PPT 上面写的是需要后序遍历，但是我又是用字典来实现的树结构，所以一时间没想出来怎么能够做到后序遍历，但是仔细想想，后剪枝不就是从叶子节点往上面遍历么，这跟递归的后一段方向不是

一样的么，想到了这个，我就恍然大悟，突然知道了该怎么实现这个剪枝，就是在递归的后面检查是否需要剪枝，也就是说在分支之前考虑验证集的正确率是否能提高，如果不能提高就干脆不剪枝了。

6 思考题

6.1 决策树有哪些避免过拟合的方法?

原因 1：样本问题

* 样本里的噪音数据干扰过大，大到模型过分记住了噪音特征，反而忽略了真实的输入输出间的关系；
* 样本抽取错误，包括（但不限于）样本数量太少，抽样方法错误，抽样时没有足够正确考虑业务场景或业务特点，等等导致抽出的样本数据不能有效足够代表业务逻辑或业务场景；
* 建模时使用了样本中太多无关的输入变量。

原因 2：构建决策树的方法问题

* 在决策树模型搭建中，我们使用的算法对于决策树的生长没有合理的限制和修剪的话，决策树的自由生长有可能每片叶子里只包含单纯的事件数据或非事件数据，可以想象，这种决策树当然可以完美匹配（拟合）训练数据，但是一旦应用到新的业务真实数据时，效果是一塌糊涂。

解决方法一

* 针对原因 1 的解决方法：合理、有效地抽样，用相对能够反映业务逻辑的训练集去产生决策树；

解决方法二

* 针对原因 2 的解决方法（主要）：剪枝：提前停止树的增长或者对已经生成的树按照一定的规则进行后剪枝。剪枝的方法剪枝是一个简化过拟合决策树的过程。有两种常用的剪枝方法：(1) 先剪枝（prepruning）：通过提前停止树的构建而对树“剪枝”，一旦停止，节点就成为树叶。该树叶可以持有子集元组中最频繁的类；(2) 后剪枝（postpruning）：它首先构造完整的决策树，允许树过度拟合训练数据，然后对那些置信度不够的结点子树用叶子结点来代替，该叶子的类标号用该结点子树中最频繁的类标记。后剪枝的剪枝过程是删除一些子树，然后用其叶子节点代替，这个叶子节点所标识的类别通过大多数原则 (majority class criterion) 确定。

6.2 C4.5 相比于 ID3 的优点是什么，C4.5 又可能有什么缺点?

6.2.1 优点

C4.5 决策树的提出完全是为了解决 ID3 决策树的一个缺点，当一个属性的可取值数目较多时，那么可能在这个属性对应的可取值下的样本只有一个或者是很少个，那么这个时候它的

信息增益是非常高的，这个时候纯度很高，ID3 决策树会认为这个属性很适合划分，但是较多取值的属性来进行划分带来的问题是它的泛化能力比较弱，不能够对新样本进行有效的预测。**由于信息增益选择分裂属性的方式会倾向于选择具有大量值的属性（即自变量）**如对于客户 ID，每个客户 ID 对应一个满意度，即按此变量划分每个划分都是纯的（即完全的划分，只有属于一个类别），客户 ID 的信息增益为最大值 1。但这种按该自变量的每个值进行分类的方式是没有任何意义的。

C4.5 算法有如下优点：产生的分类规则易于理解，准确率较高。ID3 选用信息增益进行特征选择，第一个缺点是，由于信息增益的计算公式，ID3 会偏向选择子类别多的特征，第二个缺点是，ID3 无法处理连续型变量 ID3 有这两个缺点就决定了他选择的主观性偏强，并且也存在较大的局限性。于是 C4.5 就出来解决 ID3 的缺点。首先为了解决第一个缺点，即偏向选择子类别多的特征，C4.5 采用了信息增益率进行特征选择，弱化了主观性地偏向子类别多的特征选择。为了解决第二个缺点，C4.5 用以下算法处理连续值（和离散化连续值很像）把连续值变量进行排序成（a1,a2,⋯an) 再从 (a1,a2) 区间里取 A1 作为分界来分裂数据，算信息增益率，从（a2,a3）区间里取 A2 作为分界来分裂数据，算信息增益率，这样可以得到 n-1 个信息增益率，然后选最大的。

6.2.2 缺点

C4.5 的缺点：第一个缺点是 C4.5 时间耗费大，看连续值处理那块可以看出第二个缺点是 C4.5 没解决回归问题第三个缺点是在构造树的过程中，需要对数据集进行多次的顺序扫描和排序，因而导致算法的低效。第四个缺点是 C4.5 只适合于能够驻留于内存的数据集，当训练集大得无法在内存容纳时程序无法运行。

6.3 如何用决策树来进行特征选择（判断特征的重要性)?

6.3.1 ID3 算法

1) 计算数据集 D 的经验熵 $H(D)$

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} \log_2 \frac{|C_k|}{|D|}$$

2) 计算特征 A 对数据集 D 的经验条件熵 $H(D|A)$

$$H(D|A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} H\left(D_i\right) = -\sum_{i=1}^{n} \frac{|D_i|}{D|} \sum_{k=1}^{K} \frac{|D_{ik}|}{|D_i|} \log_2 \frac{|D_{ik}|}{|D_i|}$$

3) 计算信息增益

$$g(D, A) = H(D) - H(D|A)$$

4) 选择信息增益最大的特征作为决策点

6.3.2 C4.5 算法

1) 计算特征 A 对数据集 D 的信息增益

$$g(D,A)=H(D)-H(D|A)$$

2) 计算数据集 D 关于特征 A 的值的熵 $H_A(D)$

$$H_A(D)=-\sum_{j=1}^{v}\frac{|D_j|}{|D|}\times\log\left(\frac{|D_j|}{|D|}\right)$$

3) 计算信息增益 $g_R(D,A)$

$$g_R(D,A)=\frac{g(D,A)}{H(D)}$$

4) 选择信息增益增益率最大的特征作为决策点

6.3.3 CART 算法

1) 计算特征 A 的条件下，数据集 D 的 $GINI$ 系数

$$gini(\mathrm{D},\mathrm{A})=\sum_{j=1}^{v}p\left(A_j\right)\times gini\left(D_j|A=A_j\right)$$

$$gini\left(D_j|A=A_j\right)=\sum_{i=1}^{n}p_i\left(1-p_i\right)=1-\sum_{i=1}^{n}p_i^2$$

2) 选择 $GINI$ 系数最小的特征作为决策点

7 实验收获

这次试验还是做的比较快的，也是比较顺利的，决策树还是要比 KNN 容易实现多了，可能是因为 KNN 是第一个实验，Python 已经忘了很多的缘故吧，感觉这次写决策树还是挺快乐的，特别是实现出了剪枝的功能后，心情大爽，这个实验让我对熵的理解有了更深的认识，而且了解了各种特征选择算法的优缺点。