Python编程与人工智能实践

算法篇：决策树

于泓
鲁东大学
信息与电气工程学院
2019.11.13

决策树（Decision Tree）

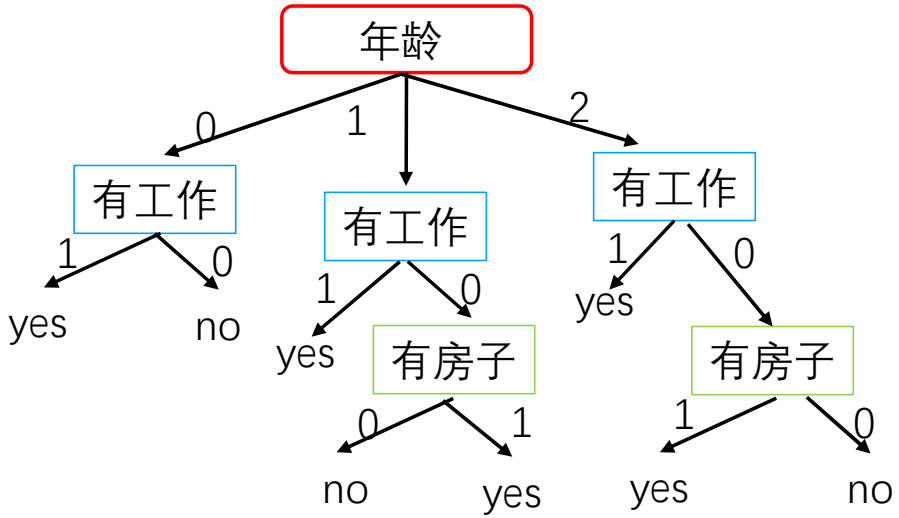
- 类似“二十个问题”的游戏，一方在脑里想某个事物，其他参与者通过提问问题，得到“是”/“否”的答案，最终猜出结果。

是否放贷?

决策树（Decision Tree）

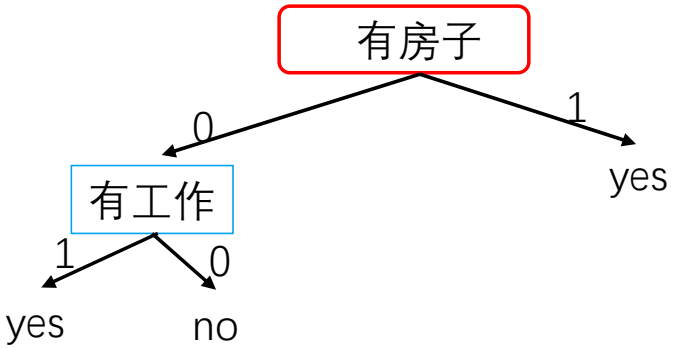
- 已知一组训练数据，用一棵树来描述训练数据的分布

年龄	有工作	有 自 己 的房子	信贷情况	是否放贷
0	0	0	0	No
0	0	0	1	No
0	1	0	1	Yes
0	1	1	0	Yes
0	0	0	0	No
1	0	0	0	No
1	0	0	1	No
1	1	1	1	Yes
1	0	1	2	Yes
1	0	1	2	Yes
2	0	1	2	Yes
2	0	1	1	Yes
2	1	0	1	Yes
2	1	0	2	Yes
2	0	0	0	No



测试数据：[0,1,0,2]　　是否放贷?

年龄	有工作	有 自 己 的房子	信贷情况	是否放贷
0	0	0	0	No
0	0	0	1	No
0	1	0	1	Yes
0	1	1	0	Yes
0	0	0	0	No
1	0	0	0	No
1	0	0	1	No
1	1	1	1	Yes
1	0	1	2	Yes
1	0	1	2	Yes
2	0	1	2	Yes
2	0	1	1	Yes
2	1	0	1	Yes
2	1	0	2	Yes
2	0	0	0	No



[年龄 工作 房子 信贷]

测试数据：[0, 1, 0, 2] 是否放贷?

如何构建树?

信息与熵

- 信息

$$l(x_i) = -\log_2 P(x_i)$$

- 平均信息（熵）

$$\mathrm{H} = -\sum_{i=1}^{N} P(x_i)\log_2 P(x_i)$$

假设有2个事件 A、B

$\mathrm{P(A)}=\frac{1}{2}$

$P(B)=\frac{1}{2}$

$H = -\frac{1}{2}\log_2(\frac{1}{2}) - \frac{1}{2}\log_2(\frac{1}{2}) = 1$

$\mathrm{P(A)}=\frac{1}{4}$

$P(B)=\frac{3}{4}$

$H = -\frac{1}{4}\log_2(\frac{1}{4}) - \frac{3}{4}\log_2(\frac{3}{4}) = 0.81$

$\mathrm{P(A)}=\frac{1}{8}$

$P(B)=\frac{7}{8}$

$H = -\frac{1}{8}\log_2(\frac{1}{8}) - \frac{7}{8}\log_2(\frac{7}{8}) = 0.54$

$\mathrm{P(A)}=0$

$P(B)=1$

$H = -0\log_2(0) - 1\log_2(1) = 0$

熵越大，事件的发生越无序
熵越小，事件的发生越确定

树的建立/划分规则

- 选取一种能够令熵最大程度变小的划分方案

年龄	有工作	有自己的房子	信贷情况	是否放贷
0	0	0	0	No
0	0	0	1	No
0	1	0	1	Yes
0	1	1	0	Yes
0	0	0	0	No
1	0	0	0	No
1	0	0	1	No
1	1	1	1	Yes
1	0	1	2	Yes
1	0	1	2	Yes
2	0	1	2	Yes
2	0	1	1	Yes
2	1	0	1	Yes
2	1	0	2	Yes
2	0	0	0	No

按年龄划分

age=0

$$P(\text{yes})=\frac{2}{5}$$

$$P(\text{no})=\frac{3}{5}$$

$$H=-\frac{2}{5}\log_2(\frac{2}{5})-\frac{3}{5}\log_2(\frac{3}{5})=0.97$$

age=1

$$P(\text{yes})=\frac{3}{5}$$

$$P(\text{no})=\frac{2}{5}$$

$$H=-\frac{3}{5}\log_2(\frac{3}{5})-\frac{2}{5}\log_2(\frac{2}{5})=0.97$$

age=2

$$P(\text{yes})=\frac{4}{5}$$

$$P(\text{no})=\frac{1}{5}$$

$$H=-\frac{4}{5}\log_2(\frac{4}{5})-\frac{1}{5}\log_2(\frac{1}{5})=0.59$$

平均熵

$$H=\frac{1}{3}*0.97+\frac{1}{3}*0.97+\frac{1}{3}*0.59=0.84$$

年龄	有工作	有 自 己 的房子	信贷情况	是否放贷
0	0	0	0	No
0	0	0	1	No
0	1	0	1	Yes
0	1	1	0	Yes
0	0	0	0	No
1	0	0	0	No
1	0	0	1	No
1	1	1	1	Yes
1	0	1	2	Yes
1	0	1	2	Yes
2	0	1	2	Yes
2	0	1	1	Yes
2	1	0	1	Yes
2	1	0	2	Yes
2	0	0	0	No

按有房子划分

有房子=1

P(yes)=1

$P(\text{no}) = 0$

$H = 0$

有房子=0

$P(\text{yes})=\frac{3}{9}=\frac{1}{3}$

$P(\text{no}) = \frac{6}{9}=\frac{2}{3}$

$H = -\frac{1}{3}\log_2(\frac{1}{3}) - \frac{2}{3}\log_2(\frac{2}{3}) = 0.91$

平均熵

$H = \frac{6}{15}*0+\frac{9}{15}*0.91=0.55$

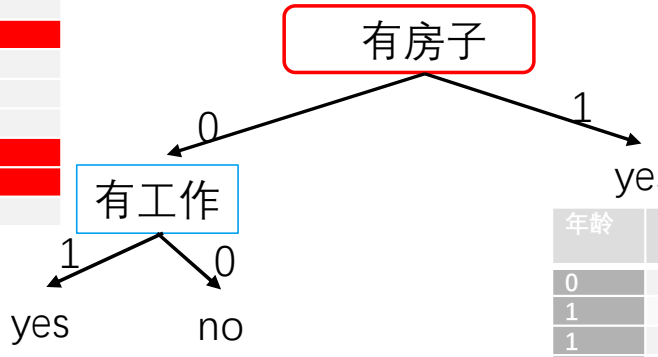
（1） 找到可以令平均熵最小的特征维度对数据集进行分割

（2） 对分割后的数据集再找寻可以使平均熵最小的特征维度，再对数据集进行分割

（3） 重复上面步骤直到 用完所有特征、或者子集中目标标签全部相同

（4） 如果所有特征都用完，最终的子集中，目标标签仍不一致，则使用最多标签作为最终输出

年龄	有工作	信贷情况	是否放贷
0	0	0	No
0	0	1	No
0	1	1	Yes
0	0	0	No
1	0	0	No
1	0	1	No
2	1	1	Yes
2	1	2	Yes
2	0	0	No



年龄	信贷情况	是否放贷
0	1	Yes
2	1	Yes
2	2	Yes

年龄	信贷情况	是否放贷
0	0	No
0	1	No
0	0	No
1	0	No
1	1	No
2	0	No

年龄	有工作	信贷情况	是否放贷
0	1	0	Yes
1	1	1	Yes
1	0	2	Yes
1	0	2	Yes
2	0	2	Yes
2	0	1	Yes

定义数据集，最后一个维度是标签

```
# 创建数据集
def createDataSet():
    dataSet = [[0, 0, 0, 0, 'no'],                        #数据集
               [0, 0, 0, 1, 'no'],
               [0, 1, 0, 1, 'yes'],
               [0, 1, 1, 0, 'yes'],
               [0, 0, 0, 0, 'no'],
               [1, 0, 0, 0, 'no'],
               [1, 0, 0, 1, 'no'],
               [1, 1, 1, 1, 'yes'],
               [1, 0, 1, 2, 'yes'],
               [1, 0, 1, 2, 'yes'],
               [2, 0, 1, 2, 'yes'],
               [2, 0, 1, 1, 'yes'],
               [2, 1, 0, 1, 'yes'],
               [2, 1, 0, 2, 'yes'],
               [2, 0, 0, 0, 'no']]
    labels = ['年龄', '有工作', '有自己的房子', '信贷情况']        #特征标签
    return dataSet, labels                                #返回数据集和分类属性
```

计算一个数据集的熵

```
#计算经验熵(香农熵)
def calcShannonEnt(dataSet):
    #返回数据集的行数
    numEntires = len(dataSet)

    #收集所有目标标签 （最后一个维度）
    labels= [featVec[-1] for featVec in dataSet]

    # 去重、获取标签种类
    keys = set(labels)

    shannonEnt = 0.0
    for key in keys:
        # 计算每种标签出现的次数
        prob = float(labels.count(key)) / numEntires
        shannonEnt -= prob * log(prob, 2)
    return shannonEnt
```

```
# 数据集分割
# 将第axis维 等于 value 的数据集提取出来
def splitDataSet(dataSet, axis, value):
    retDataSet = []                                     #创建返回的数据集列表
    for featVec in dataSet:                            #遍历数据集
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]             #去掉axis特征
            reducedFeatVec.extend(featVec[axis+1:])    #将符合条件的添加到返回的数据集
            retDataSet.append(reducedFeatVec)
    return retDataSet                                   #返回划分后的数据集
```

数据集的分割

```
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1                   #特征数量
    baseEntropy = calcShannonEnt(dataSet)               #计算数据集的香农熵
    bestInfoGain = 0.0                                  #信息增益
    bestFeature = -1                                    #最优特征的索引值
    for i in range(numFeatures):                        #遍历所有特征
        #获取dataSet的第i个所有特征
        featList = [example[i] for example in dataSet]
        uniqueVals = set(featList)                      #创建set集合{},元素不可重复
        newEntropy = 0.0                                #经验条件熵
        for value in uniqueVals:                        #计算信息增益
            subDataSet = splitDataSet(dataSet, i, value)        #subDataSet划分后的子集
            prob = len(subDataSet) / float(len(dataSet))        #计算子集的概率
            newEntropy += prob * calcShannonEnt(subDataSet)     #根据公式计算经验条件熵
        infoGain = baseEntropy - newEntropy             #信息增益
        # print("第%d个特征的增益为%.3f" % (i, infoGain))       #打印每个特征的信息增益
        if (infoGain > bestInfoGain):                   #计算信息增益
            bestInfoGain = infoGain                     #更新信息增益，找到最大的信息增益
            bestFeature = i                             #记录信息增益最大的特征的索引值
    return bestFeature                                  #返回信息增益最大的特征的索引值
```

在一个数据集中
找到使熵减少最大的
维度

```
# 创建决策树
def createTree(dataSet,labels, lab_sel):
    #取分类标签(是否放贷:yes or no)
    classList = [example[-1] for example in dataSet]

    #如果类别完全相同则停止继续划分
    if classList.count(classList[0]) == len(classList):
        return classList[0]

    #遍历完所有特征时返回出现次数最多的类标签
    if len(dataSet[0]) == 1 or len(labels) == 0:
        return majorityCnt(classList)

    #获取最优特征的维度
    bestFeat = chooseBestFeatureToSplit(dataSet)

    #得到最优特征的标签
    bestFeatLabel = labels[bestFeat]
    lab_sel.append(labels[bestFeat])

    #根据最优特征的标签生成树
    myTree = {bestFeatLabel:{}}

    #删除已经使用特征标签
    del(labels[bestFeat])

    #得到训练集中所有最优特征维度的所有属性值
    featValues = [example[bestFeat] for example in dataSet]
    uniqueVals = set(featValues)                                    #去掉重复的属性值
    for value in uniqueVals:                                        #遍历特征，创建决策树。
        subLabels = labels[:]
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), subLabels,lab_sel)

    return myTree
```

```
#返回classList中出现次数最多的元素
def majorityCnt(classList):
    classCount = {}
    keys = set(classLabel)
    for key in keys:
        classCount[key] = classList.count(key)

    #根据字典的值降序排序
    sortedClassCount = sorted(classCount.items(),
                              key = operator.itemgetter(1),
                              reverse = True)
    return sortedClassCount[0][0]
```

```python
# 进行分类
def classify(inputTree, featLabels, testVec):
    firstStr = next(iter(inputTree))        #获取决策树结点
    secondDict = inputTree[firstStr]        #下一个字典
    featIndex = featLabels.index(firstStr)
    for key in secondDict.keys():
        if testVec[featIndex] == key:
            if type(secondDict[key]).__name__ == 'dict':
                classLabel = classify(secondDict[key], featLabels, testVec)
            else: classLabel = secondDict[key]
    return classLabel
```

```python
if __name__ == '__main__':
    # 获取数据集
    dataSet, labels = createDataSet()

    lab_sel = []
    myTree = createTree(dataSet, labels,lab_sel)
    print(myTree)
    print(lab_sel)
    # 测试
    testVec = [0,1,1,2]
    result = classify(myTree,lab_sel,testVec)
    print(result)
```

```
yuhong@admin2:/home/sdo/machinelearning/dtree$ python DecisionTree.py
{'有自己的房子': {0: {'有工作': {0: 'no', 1: 'yes'}}, 1: 'yes'}}
['有自己的房子', '有工作']
yes
```

隐形眼镜数据集

['年龄','近视/远视','是否散光','是否眼干']

训练

train-lenses.txt | DecisionTree.py | train-len

1	青年	近视	否	干涩	不配镜
2	青年	近视	否	正常	软镜片
3	青年	近视	是	干涩	不配镜
4	青年	远视	否	干涩	不配镜
5	青年	远视	是	干涩	不配镜
6	中年	近视	否	干涩	不配镜
7	中年	近视	否	正常	软镜片
8	中年	近视	是	干涩	不配镜
9	中年	近视	是	正常	硬镜片
10	中年	远视	否	干涩	不配镜
11	中年	远视	否	正常	软镜片
12	中年	远视	是	干涩	不配镜
13	中年	远视	是	正常	不配镜
14	老年	近视	否	正常	不配镜
15	老年	近视	是	干涩	不配镜
16	老年	近视	是	正常	硬镜片
17	老年	远视	否	干涩	不配镜
18	老年	远视	否	正常	软镜片
19	老年	远视	是	干涩	不配镜
20	老年	远视	是	正常	不配镜
21	青年	远视	是	正常	硬镜片

测试

test-lenses.txt | train-lenses.txt | DecisionT

1	青年	远视	否	正常	软镜片
2	老年	近视	否	干涩	不配镜
3	青年	近视	是	正常	硬镜片

```
with open("train-lenses.txt",'r') as f:
    lines = f.read().splitlines()

dataSet = [line.split('\t') for line in lines]
labels = ['年龄','近视/远视','是否散光','是否眼干']

lab_copy = labels[:]
lab_sel = []
myTree = createTree(dataSet, labels,lab_sel)
print(myTree)
print(lab_sel)

# 测试
with open("test-lenses.txt",'r') as f:
    lines = f.read().splitlines()

for line in lines:
    data = line.split('\t')
    lab_true = data[-1]
    test_vec = data[:-1]
    result = classify(myTree,lab_copy,test_vec)

    print("输入特征：")
    print(test_vec)
    print("预测结果 %s  医生推荐 %s"%(result,lab_true))
```

```
yuhong@admin2:/home/sdo/machinelearning/dtree$ python DecisionTree.py
{'是否眼干': {'正常': {'是否散光': {'是': {'近视/远视': {'近视': '硬镜片', '远视': {'年龄': {'中年': '
不配镜', '青年': '硬镜片', '老年': '不配镜'}}}}, '否': {'年龄': {'中年': '软镜片', '青年': '软镜片', '
老年': {'近视/远视': {'近视': '不配镜', '远视': '软镜片'}}}}}}}, '干涩': '不配镜'}}
['是否眼干', '是否散光', '近视/远视', '年龄', '年龄', '近视/远视']
输入特征：
['青年', '远视', '否', '正常']
预测结果 软镜片  医生推荐 软镜片
输入特征：
['老年', '近视', '否', '干涩']
预测结果 不配镜  医生推荐 不配镜
输入特征：
['青年', '近视', '是', '正常']
预测结果 硬镜片  医生推荐 硬镜片
```