Python与机器学习

Python篇

Python篇

第2讲： 决策树

决策树简介

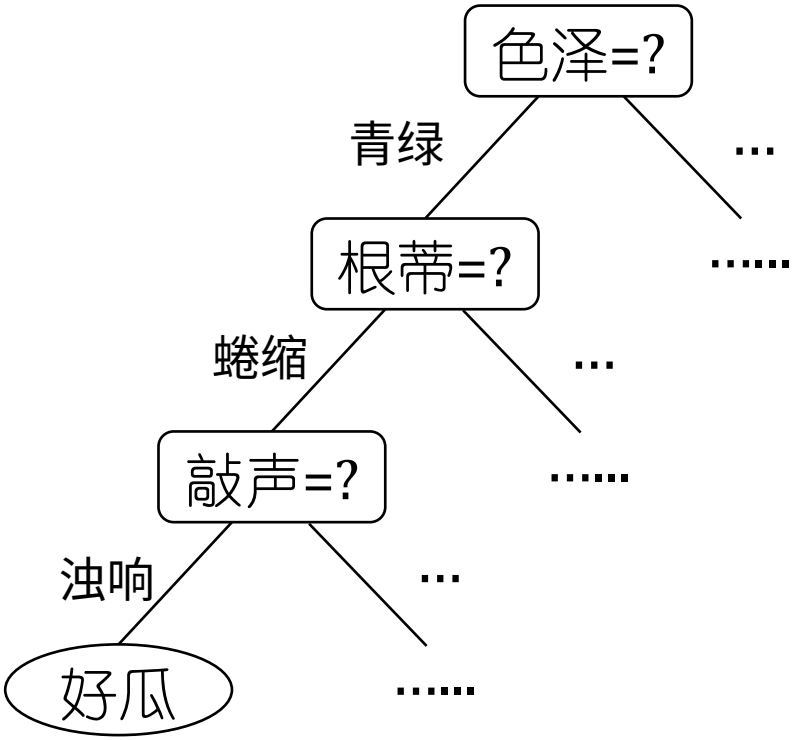
分类

- K近邻学习
  - 优点：简单易实现
  - 缺点：无法给出数据的内在含义，可解释性差

- 决策树(Decision tree)
  - 优点：学习到的模型非常容易理解，可解释性强
  - 缺点：过拟合问题

决策树基本流程

决策树基于树结构来进行预测



基本流程

- 决策过程中提出的每个判定问题都是对某个属性的“测试”
- 决策过程的最终结论对应了我们所希望的判定结果
- 每个测试的结果或是导出最终结论，或者导出进一步的判定问题，其考虑范围是在上次决策结果的限定范围之内
- 从根结点到每个叶结点的路径对应了一个判定测试序列

决策树学习的目的是为了产生一棵泛化能力强，
即处理未见示例能力强的决策树

基本流程

Algorithm 1 决策树学习基本算法

输入:

- 训练集 $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$;
- 属性集 $A = \{a_1, \ldots, a_d\}$.

过程: **函数** TreeGenerate(D, A)

```
1: 生成结点 node;
2: if D 中样本全属于同一类别 C then
3:     将 node 标记为 C 类叶结点; return
4: end if
5: if A = ∅ OR D 中样本在 A 上取值相同 then
6:     将 node 标记叶结点，其类别标记为 D 中样本数最多的类; return
7: end if
8: 从 A 中选择最优划分属性 a_*;
9: for a_* 的每一个值 a_*^v do
10:    为 node 生成每一个分枝; 令 D_v 表示 D 中在 a_* 上取值为 a_*^v 的样本子集;
11:    if D_v 为空 then
12:        将分枝结点标记为叶结点, 其类别标记为 D 中样本最多的类; return
13:    else
14:        以 TreeGenerate(D_v, A − {a_*}) 为分枝结点
15:    end if
16: end for
```

输出: 以 node 为根结点的一棵决策树

（1）当前结点包含的样本全部属于同一类别

（2）当前属性集为空，或所有样本在所有属性上取值相同

（3）当前结点包含的样本集合为空

用Python实现决策树

决策树的一般流程

- 1. 收集数据
- 2. 准备数据 (对连续型数据进行离散化)
- 3. 训练： (在训练集中构建决策树)
- 4. 验证： (在验证集中验证决策树)
- 5. 测试： (在测试集中测试决策树)

问题描述

- 问题：判断某种海洋动物是否是鱼类
  - 类标签：鱼类、非鱼类
  - 属性：
    - 不浮出水面是否可以生存
    - 是否有脚蹼

	不浮出水面是否可以生存	是否有脚蹼	属于鱼类
1	是	是	是
2	是	是	是
3	是	否	否
4	否	是	否
5	否	是	否

创建数据

- createDataSet()

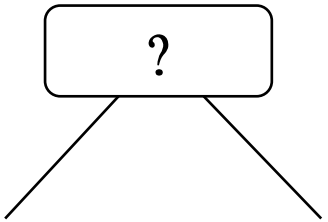
```
def createDataSet():
    dataSet = [[1, 1, 'yes'],
               [1, 1, 'yes'],
               [1, 0, 'no'],
               [0, 1, 'no'],
               [0, 1, 'no']]
    attributes = ['no surfacing','flippers']
    return dataSet, attributes
```

- dataSet:整个数据集
- attributes:属性

```
>>> import trees
>>> myDat, attributes = trees.createDataSet()
>>> myDat
[[1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0, 1, 'no']]
>>> attributes
['no surfacing', 'flippers']
>>>
```

第1步：计算信息增益

- 每一次判断，我们应该选择哪个属性作为参考属性？



- 计算每个属性的信息增益：
  - 划分数据集之前和之后信息发生的变化
  - 获得信息增益最高的属性就是最好的选择

第1.1步：计算信息熵

- “信息熵”是度量样本集合纯度最常用的一种指标，假定当前样本集合D中第k类样本所占的比例为$p_k\,(K=1,2,...,|\mathcal{Y}|)$，则$D$的信息熵定义为

$$\mathrm{Ent}(D)=-\sum_{k=1}^{|\mathcal{Y}|}p_k\log_2 p_k$$

$\mathrm{Ent}(D)$的值越小，则D的纯度越高

- 计算信息熵时约定：若 $p=0$，则$p\log_2 p=0$
- $\mathrm{Ent}(D)$的最小值为0，最大值为$\log_2|\mathcal{Y}|$

第1.1步：计算信息熵

```
def calcEnt(dataSet):
    numEntries = len(dataSet)    # 计算数据中的实例总数
    labelCounts = {}        # 创建字典，保存各类标签的数量
   for featVec in dataSet: # 计算各个类的数量
        currentLabel = featVec[-1]
        if currentLabel not in labelCounts.keys():
            labelCounts[currentLabel] = 0
            labelCounts[currentLabel] += 1
        else:
            labelCounts[currentLabel] += 1
    ent = 0.0
    for key in labelCounts:
        prob = float(labelCounts[key])/numEntries
        ent -= prob * log(prob,2)
    return ent
```

```
>>> trees.calcEnt(myDat)
0.9709505944546686
```

第1.2步：计算信息增益

- 离散属性a有V个可能的取值$\{a^1, a^2, ..., a^V\}$，用a来进行划分，则会产生V个分支结点，其中第v个分支结点包含了D中所有在属性a上取值为a^v的样本，记为D^v。则可计算出用属性a对样本集D进行划分所获得的“信息增益”：

$$\mathrm{Gain}(D,a) = \mathrm{Ent}(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} \mathrm{Ent}(D^v)$$

为分支结点权重，样本数越多的分支结点的影响越大

- 一般而言，信息增益越大，则意味着使用属性a来进行划分所获得的“纯度提升”越大
- ID3决策树学习算法[Quinlan, 1986]以信息增益为准则来选择划分属性

第1.2步：计算信息增益

- 1. 按照给定属性划分数据集(即给定属性a，及其取值V，获得D^V)

```
def splitDataSet(dataSet, attr, value):
    retDataSet = []
    for featVec in dataSet:
        if featVec[attr] == value:
            reducedFeatVec = featVec[:attr]  # 获取attr前面的属性
            reducedFeatVec.extend(featVec[attr+1:])#获取attr后面的属性
            retDataSet.append(reducedFeatVec) #保存
    return retDataSet
```

- dataSet：数据集
- attr：待划分属性
- value：待划分属性的取值

- 返回attr==value的数据(同时删除attr)

```
>>> trees.splitDataSet(myDat,0,1)
[[1, 'yes'], [1, 'yes'], [0, 'no']]
>>> trees.splitDataSet(myDat,0,0)
[[1, 'no'], [1, 'no']]
```

	不浮出水面是否可以生存	是否有脚蹼	属于鱼类
1	是	是	是
2	是	是	是
3	是	否	否
4	否	是	否
5	否	是	否

第1.2步：计算信息增益

$$\mathrm{Gain}(D,a)=\mathrm{Ent}(D)-\sum_{v=1}^{V}\frac{|D^v|}{|D|}\mathrm{Ent}(D^v)$$

- 2.计算信息熵、及对所有属性划分数据集，然后找到最好的属性

```
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1      # 属性数量
    baseEntropy = calcEnt(dataSet)         # 计算整个数据集的信息熵 Ent(D)
    bestInfoGain = 0.0; bestFeature = -1
    for i in range(numFeatures):           # 循环执行全部属性
        featList = [example[i] for example in dataSet]    # 对每一列数据(属性)创建一个列表
        uniqueVals = set(featList)         # 获得该属性所有可能取值, set('abbba')={'a', 'b'}
        newEntropy = 0.0
        for value in uniqueVals:
            subDataSet = splitDataSet(dataSet, i, value)     # 划分数据集，对i==value划分
            prob = len(subDataSet)/float(len(dataSet))
            newEntropy += prob * calcEnt(subDataSet)
        infoGain = baseEntropy - newEntropy
        if (infoGain > bestInfoGain):
            bestInfoGain = infoGain
            bestFeature = i
    return bestFeature                     # 返回最好的属性
```

```
>>> trees.chooseBestFeatureToSplit(myDat)
0
```

第1.2步：计算信息增益

- 通过以上两步(第1.1，第1.2步)，我们解决了决策树生成的第一个问题，如何决定应该首先判断哪个属性

- 这个结果好吗?

```
>>> trees.chooseBestFeatureToSplit(myDat)
0
```

  - 重新看数据集
  - 判断第0个属性和第1个属性，哪个更好

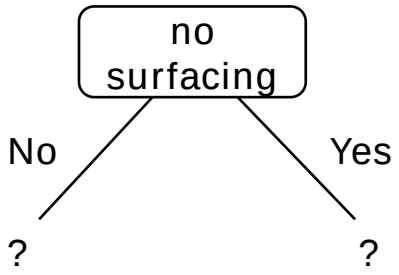
	不浮出水面是否可以生存	是否有脚蹼	属于鱼类
1	是	是	是
2	是	是	是
3	是	否	否
4	否	是	否
5	否	是	否

第2步：如何构建决策树

第2步：如何构建决策树

- 每个节点中属性的确定，都是一次信息增益的重新计算(第1步)，因此，我们想到的解决办法是 --- 递归

- 回到“判断某种海洋动物是否是鱼类”问题



	不浮出水面是否可以生存	是否有脚蹼	属于鱼类
1	是	是	是
2	是	是	是
3	是	否	否
4	否	是	否
5	否	是	否

第2步：如何构建决策树

- 递归结束的条件：程序遍历完所有属性，或者每个分支下的所有实例都具有相同的分类

- 那么，如果程序遍历完所有属性，依然无法100%判断，就应该用投票法决定叶节点(Majority voting)

```
def majorityCnt(classList):
    classCount={}
    for vote in classList:
        if vote not in classCount.keys():
            classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.items(), key=lambda
itemLabel:itemLabel[1], reverse=True)
    return sortedClassCount[0][0]
```

第2步：如何构建决策树

```
def createTree(dataSet,attributes):
    classList = [example[-1] for example in dataSet]  #为最后列(label)创建列表
    if classList.count(classList[0]) == len(classList):  # 如果所有数据类都相同
        return classList[0]
    if len(dataSet[0]) == 1:        # 如果当前数据集中，这是最后一个待处理的属性
        return majorityCnt(classList)   # 采用投票法决定叶节点
    bestFeat = chooseBestFeatureToSplit(dataSet)  # 判断哪个属性是最好的
    bestFeatLabel = attributes[bestFeat]
    myTree = {bestFeatLabel:{}}           # 创建一个字典，保存决策树
    del(attributes[bestFeat])       # 删除已处理的属性
    featValues = [example[bestFeat] for example in dataSet] # 为属性创建列表
    uniqueVals = set(featValues)     # 获得该属性的所有可能取值(确定树节点的分支)
    for value in uniqueVals:
        subAttributes = attributes[:]       # 复制所有属性(除已删除的bestFeat)
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet,
bestFeat, value),subAttributes)  # 递归创建树的节点(对每一个分支)
    return myTree
```

```
>>> myTree = trees.createTree(myDat, attributes)
>>> myTree
{'no surfacing': {0: 'no', 1: {'flippers': {0: 'no', 1: 'yes'}}}}
```

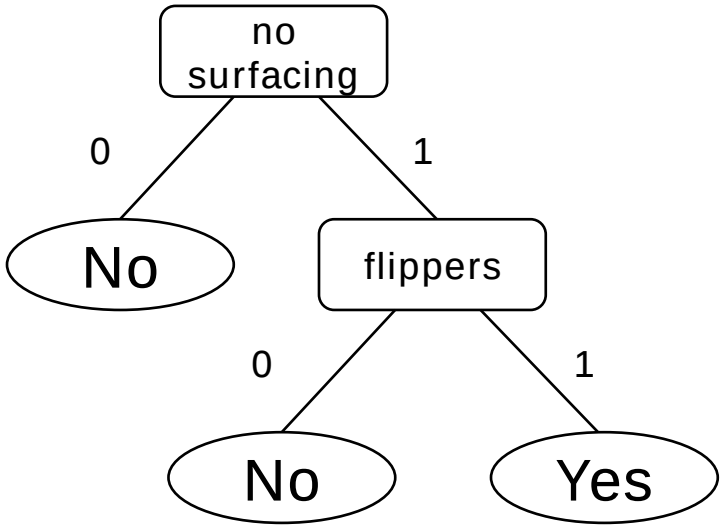
测试决策树

测试决策树

- 给定测试数据testVec，遍历决策树进行判断

```
def classify(inputTree,featAttributes,testVec):
    firstStr = list(inputTree.keys())[0]    #获取当前决策树根节点属性
    secondDict = inputTree[firstStr]        #获取该节点下的剩余决策树
    featIndex = featAttributes.index(firstStr) #根结点属性所处位置
    key = testVec[featIndex]                #获取测试数据中该属性的值
    valueOfFeat = secondDict[key]     #找到该属性值的分支，并返回
    if isinstance(valueOfFeat, dict): #如果该分支是字典，需要继续遍历
        classLabel = classify(valueOfFeat, featAttributes, testVec)
    else: classLabel = valueOfFeat     #否则，返回叶节点值
    return classLabel
```

测试决策树



	不浮出水面是否可以生存	是否有脚蹼	属于鱼类
1	是	是	是
2	是	是	是
3	是	否	否
4	否	是	否
5	否	是	否

```
>>> trees.classify(myTree, attributes, [1,0])
'no'
>>> trees.classify(myTree, attributes, [0,1])
'no'
>>> trees.classify(myTree, attributes, [1,1])
'yes'
```

决策树的存储

- 构造决策树很耗时
- 决策树是一种离线学习算法
- 将决策树用来解决分类问题，可事先对训练数据集（可大规模）进行学习，获得决策树
- 在需要对未知测试数据进行预测时，直接调用该决策树即可
- 因此，为了节省时间，对决策树进行存储

pickle模块

- Python的pickle模块
  - 通过pickle模块，可将程序中运行的对象信息保存到文件中，并永久存储
  - pickle.dump(obj, file)
    - 将对象obj存储到文件file中（其中，file需要有写权限）

```
def storeTree(inputTree,filename):
    import pickle
    fw = open(filename,'wb')
    pickle.dump(inputTree,fw)
    fw.close()
```

```
>>>
>>> trees.storeTree(myTree, 'fishClassifier.txt')
```

- pickle.load(file)
  - 读取文件file中的内容（其中，file需要有读权限）

```
def grabTree(filename):
    import pickle
    fr = open(filename, 'rb')
    return pickle.load(fr)
```

```
>>> trees.grabTree('fishClassifier.txt')
{'no surfacing': {0: 'no', 1: {'flippers': {0: 'no', 1: 'yes'}}}}
```

大作业

- 用决策树解决约会网站的推荐问题

  - 1. 文本读取训练数据
  - 2. 连续数据的离散化
  - 3. 剪枝操作？