菊安酱的机器学习第2期

菊安酱的直播间：https://live.bilibili.com/14988341

每周一晚8:00 菊安酱和你不见不散哦~(^o^)/~

更新日期：2018-11-11

作者：菊安酱

课件内容说明：

- 本文为作者原创内容，转载请注明作者和出处
- 如果想获得此课件及完整视频，可扫描下面二维码，回复"k"进群

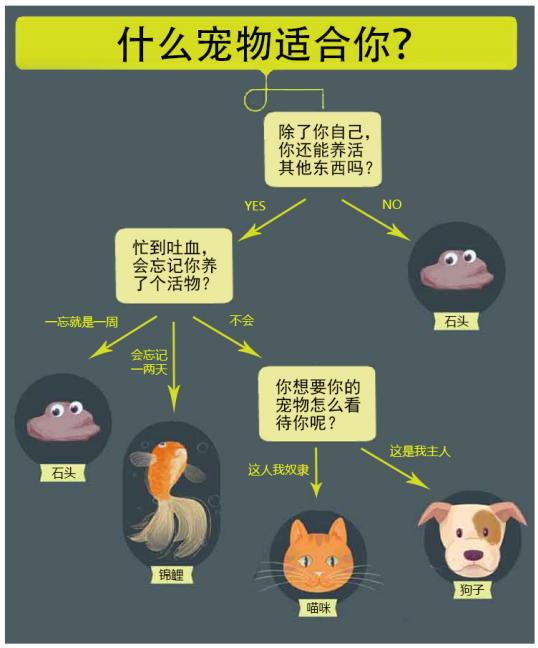
决策树

一、概述

决策树（Decision Tree）是有监督学习中的一种算法，并且是一种基本的分类与回归的方法。也就是说，决策树有两种：分类树和回归树。这里我们主要讨论分类树，后面再为大家讲解回归树。

什么是决策树

让我们从养宠物开始说起~



通过上面的例子，我们很容易理解：决策树算法的本质就是树形结构，我们可以通过一些精心设计的问题，就可以对数据进行分类了。在这里，我们需要了解三个概念：

节点	说明
根节点	没有进边，有出边
中间节点	既有进边也有出边，但进边有且仅有一条，出边也可以有很多条
叶节点	只有进边，没有出边，进边有且仅有一条。**每个叶节点都是一个类别标签**
*父节点和子节点	在两个相连的节点中，更靠近根节点的是父节点，另一个则是子节点。**两者是相对的。**

我们可以把决策树看作是一个if-then规则的集合。将决策树转换成if-then规则的过程是这样的：

- 由决策树的根节点到叶节点的每一条路径构建一条规则
- 路径上中间节点的特征对应着规则的条件，也叶节点的类标签对应着规则的结论

  决策树的路径或者其对应的if-then规则集合有一个重要的性质：互斥并且完备。也就是说，每一个实例都被**有且仅有一条**路径或者规则所覆盖。这里的覆盖是指实例的特征与路径上的特征一致，或实例满足规则的条件。

二、决策树的构建准备工作

使用决策树做分类的每一个步骤都很重要，首先我们要收集足够多的数据，如果数据收集不到位，将会导致没有足够的特征去构建错误率低的决策树。数据特征充足，但是不知道用哪些特征好，也会导致最终无法构建出分类效果好的决策树。从算法方面来看的话，决策树的构建就是我们的核心内容。

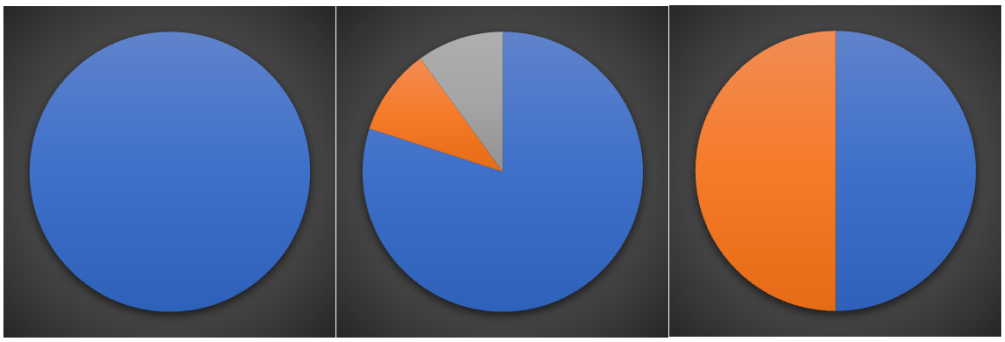
决策树如何构建呢？通常，这一过程可以概括为3个步骤：特征选择、决策树的生成和决策树的剪枝。关于剪枝部分，我会放到回归树中给大家作详细的讲解。

1.特征选择

特征选择就是决定用哪个特征来划分特征空间，其目的在于选取对训练数据具有分类能力的特征。这样可以提高决策树学习的效率。如果利用一个特征进行分类的结果与随机分类的结果没有很大的差别，则称这个特征是没有分类能力的，经验上扔掉这些特征对决策树学习的精度影响不会很大。

那如何来选择最优的特征来划分呢？一般而言，随着划分过程不断进行，我们希望决策树的分支节点所包含的样本尽可能属于同一类别，也就是节点的**纯度**（purity）越来越高。

下面三个图表示的是纯度越来越低的过程，最后一个表示的是纯度最低的状态。



在实际使用中，我们衡量的常常是不纯度。度量不纯度的指标有很多种，比如：熵、增益率、基尼值数。

这里我们使用的是熵，也叫作**香农熵**，这个名字来源于信息论之父 克劳德·香农。

1.1 香农熵及计算函数

熵定义为信息的期望值。在信息论与概率统计中，熵是表示随机变量不确定性的度量。

假定当前样本集合D中一共有n类样本，第i类样本为x_i，那么x_i的信息定义为：

$$l(x_i) = -log_2 p(x_i)$$

其中$p(x_i)$是选择该分类的概率。

通过上式，我们可以得到所有类别的信息。为了计算熵，我们需要计算所有类别所有可能值包含的信息期望值(数学期望)，通过下面的公式得到：

$$Ent(D) = -\sum_{i=1}^{n} p(x_i) log_2 p(x_i)$$

$Ent(D)$的值越小，则D的不纯度就越低。

香农熵的python代码如下：

```
"""
函数功能：计算香农熵
参数说明：
    dataSet：原始数据集
返回：
    ent:香农熵的值
"""
def calEnt(dataSet):
    n = dataSet.shape[0]                                #数据集总行数
    iset = dataSet.iloc[:,-1].value_counts()            #标签的所有类别
    p = iset/n                                          #每一类标签所占比
    ent = (-p*np.log2(p)).sum()                         #计算信息熵
    return ent
```

以书上的海洋生物数据为例，我们来构建数据集，并计算其香农熵

No.	no surfacing	flippers	fish
1	1	1	yes
2	1	1	yes
3	1	0	no
4	0	1	no
5	0	1	no

表1 海洋生物数据

```
#创建数据集
import numpy as np
import pandas as pd

def createDataSet():
    row_data = {'no surfacing':[1,1,1,0,0],
                'flippers':[1,1,0,1,1],
                'fish':['yes','yes','no','no','no']}
    dataSet = pd.DataFrame(row_data)
    return dataSet
```

```
dataSet = createDataSet()
dataSet
```

```
calEnt(dataSet)
```

熵越高，信息的不纯度就越高。也就是混合的数据就越多。

1.2 信息增益

信息增益（Information Gain）的计算公式其实就是父节点的信息熵与其下所有子节点总信息熵之差。但这里要注意的是，此时计算子节点的总信息熵不能简单求和，而要求在求和汇总之前进行修正。

假设离散属性a有V个可能的取值$\{a^1,a^2,.....,a^V\}$,若使用a对样本数据集D进行划分，则会产生V个分支节点，其中第v个分支节点包含了D中所有在属性a上取值为a^v的样本，记为D^v.我们可根据信息熵的计算公式计算出D^v的信息熵，再考虑到不同的分支节点所包含的样本数不同，给分支节点赋予权重$|D^v|/|D|$,这就是所谓的的修正。

所以信息增益的计算公式为

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

那我们手动计算一下，海洋生物数据集中第0列的信息增益：

$$\begin{aligned}Gain('nosurfacing')&=Ent(D)-[\frac{3}{5}Ent(D_1)+\frac{2}{5}Ent(D_2)]\\&=calEnt(dataSet)-[\frac{3}{5}(-\frac{2}{3}log_2\frac{2}{3}-\frac{1}{3}log_2\frac{1}{3})+\frac{2}{5}(-\frac{2}{2}log_2\frac{2}{2})]\\&=0.97-0.55\\&=0.42\end{aligned}$$

```
a=(3/5)*(-(2/3)*np.log2(2/3)-(1/3)*np.log2(1/3))
calEnt(dataSet)-a
```

用同样的方法，我们可以把第1列的信息增益也算出来，结果为0.17

2. 数据集最佳切分函数

划分数据集的最大准则是选择**最大信息增益**，也就是信息下降最快的方向。

```
"""
函数功能：根据信息增益选择出最佳数据集切分的列
参数说明：
    dataSet：原始数据集
返回：
    axis:数据集最佳切分列的索引
"""

#选择最优的列进行切分
def bestSplit(dataSet):
    baseEnt = calEnt(dataSet)                                   #计算原始熵
    bestGain = 0                                                #初始化信息增益
    axis = -1                                                   #初始化最佳切分列，标签列
    for i in range(dataSet.shape[1]-1):                         #对特征的每一列进行循环
        levels= dataSet.iloc[:,i].value_counts().index          #提取出当前列的所有取值
        ents = 0                                                #初始化子节点的信息熵
        for j in levels:                                        #对当前列的每一个取值进行循环
            childSet = dataSet[dataSet.iloc[:,i]==j]            #某一个子节点的dataframe
            ent = calEnt(childSet)                              #计算某一个子节点的信息熵
            ents += (childSet.shape[0]/dataSet.shape[0])*ent #计算当前列的信息熵
        #print(f'第{i}列的信息熵为{ents}')
```

```
        infoGain = baseEnt-ents                                    #计算当前列的信息增益
        #print(f'第{i}列的信息增益为{infoGain}')
        if (infoGain > bestGain):
            bestGain = infoGain                                    #选择最大信息增益
            axis = i                                               #最大信息增益所在列的索引
    return axis
```

通过上面手动计算，我们知道：

第0列的信息增益为0.42，第1列的信息增益为0.17，0.42>0.17，所以我们应该选择第0列进行切分数据集。

接下来，我们来验证我们构造的数据集最佳切分函数返回的结果与手动计算的结果是否一致。

```
bestSplit(dataSet)        #返回的结果为0，即选择第0列来切分数据集
```

3. 按照给定列切分数据集

通过最佳切分函数返回最佳切分列的索引，我们就可以根据这个索引，构建一个按照给定列切分数据集的函数

```
"""
函数功能：按照给定的列划分数据集
参数说明：
    dataSet：原始数据集
    axis：指定的列索引
    value：指定的属性值
返回：
    redataSet：按照指定列索引和属性值切分后的数据集
"""

def mySplit(dataSet,axis,value):
    col = dataSet.columns[axis]
    redataSet = dataSet.loc[dataSet[col]==value,:].drop(col,axis=1)
    return redataSet
```

验证函数，以axis=0，value=1为例

```
mySplit(dataSet,0,1)
```

三、递归构建决策树

目前我们已经学习了从数据集构造决策树算法所需要的子功能模块，其工作原理如下：得到原始数据集，然后基于最好的属性值划分数据集，由于特征值可能多于两个，因此可能存在大于两个分支的数据集划分。第一次划分之后，数据集被向下传递到树的分支的下一个结点。在这个结点上，我们可以再次划分数据。因此我们可以采用递归的原则处理数据集。

决策树生成算法递归地产生决策树，直到不能继续下去未为止。这样产生的树往往对训练数据的分类很准确，但对未知的测试数据的分类却没有那么准确，即出现过拟合现象。过拟合的原因在于学习时过多地考虑如何提高对训练数据的正确分类，从而构建出过于复杂的决策树。解决这个问题的办法是考虑决策树的复杂度，对已生成的决策树进行简化，也就是常说的剪枝处理。剪枝处理的具体讲解我会放在回归树里面。

1. ID3算法

构建决策树的算法有很多，比如ID3、C4.5和CART，基于《机器学习实战》这本书，我们选择ID3算法。

ID3算法的核心是在决策树各个节点上对应信息增益准则选择特征，递归地构建决策树。具体方法是：从根节点开始，对节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节点的特征，由该特征的不同取值建立子节点；再对子节点递归地调用以上方法，构建决策树；直到所有特征的信息增益均很小或没有特征可以选择为止。最后得到一个决策树。

递归结束的条件是：程序遍历完所有的特征列，或者每个分支下的所有实例都具有相同的分类。如果所有实例具有相同分类，则得到一个叶节点。任何到达叶节点的数据必然属于叶节点的分类，即叶节点里面必须是标签。

2. 编写代码构建决策树

```
"""
函数功能：基于最大信息增益切分数据集，递归构建决策树
参数说明：
    dataSet：原始数据集（最后一列是标签）
返回：
    myTree：字典形式的树
"""
def createTree(dataSet):
    featlist = list(dataSet.columns)                          #提取出数据集所有的列
    classlist = dataSet.iloc[:,-1].value_counts()             #获取最后一列类标签
    #判断最多标签数目是否等于数据集行数，或者数据集是否只有一列
    if classlist[0]==dataSet.shape[0] or dataSet.shape[1] == 1:
        return classlist.index[0]                             #如果是，返回类标签
    axis = bestSplit(dataSet)                                 #确定出当前最佳切分列的索引
    bestfeat = featlist[axis]                                 #获取该索引对应的特征
    myTree = {bestfeat:{}}                                    #采用字典嵌套的方式存储树信息
    del featlist[axis]                                        #删除当前特征
    valuelist = set(dataSet.iloc[:,axis])                     #提取最佳切分列所有属性值
    for value in valuelist:                                   #对每一个属性值递归建树
        myTree[bestfeat][value] = createTree(mySplit(dataSet,axis,value))
    return myTree
```

查看函数运行结果

```
myTree = createTree(dataSet)
myTree
```

四、决策树的存储

构造决策树是很耗时的任务，即使处理很小的数据集，也要花费几秒的时间，如果数据集很大，将会耗费很多计算时间。因此为了节省时间，建好树之后立马将其保存，后续使用直接调用即可。

我这边使用的是numpy里面的save()函数，它可以直接把字典形式的数据保存为.npy文件，调用的时候直接使用load()函数即可。

```
#树的存储
np.save('myTree.npy',myTree)

#树的读取
read_myTree = np.load('myTree.npy').item()
read_myTree
```

五、使用决策树执行分类

```
"""
函数功能：对一个测试实例进行分类
参数说明：
    inputTree：已经生成的决策树
    labels：存储选择的最优特征标签
    testVec：测试数据列表，顺序对应原数据集
返回：
    classLabel：分类结果
"""
def classify(inputTree,labels, testVec):
    firstStr = next(iter(inputTree))                    #获取决策树第一个节点
    secondDict = inputTree[firstStr]                    #下一个字典
    featIndex = labels.index(firstStr)                  #第一个节点所在列的索引
    for key in secondDict.keys():
        if testVec[featIndex] == key:
            if type(secondDict[key]) == dict :
                classLabel = classify(secondDict[key], labels, testVec)
            else:
                classLabel = secondDict[key]
    return classLabel
```

```
"""
函数功能：对测试集进行预测，并返回预测后的结果
参数说明：
    train：训练集
    test：测试集
返回：
    test：预测好分类的测试集
"""
def acc_classify(train,test):
    inputTree = createTree(train)                       #根据测试集生成一棵树
    labels = list(train.columns)                        #数据集所有的列名称
    result = []
    for i in range(test.shape[0]):                      #对测试集中每一条数据进行循环
        testVec = test.iloc[i,:-1]                      #测试集中的一个实例
        classLabel = classify(inputTree,labels,testVec) #预测该实例的分类
        result.append(classLabel)                       #将分类结果追加到result列表中
    test['predict']=result                              #将预测结果追加到测试集最后一列
    acc = (test.iloc[:,-1]==test.iloc[:,-2]).mean()     #计算准确率
    print(f'模型预测准确率为{acc}')
```

```
    return test
```

测试函数

```
train = dataSet
test = dataSet.iloc[:3,:]
acc_classify(train,test)
```

使用SKlearn中graphviz包实现决策树的绘制

```
#导入相应的包
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
import graphviz

#特征
Xtrain = dataSet.iloc[:,:-1]
#标签
Ytrain = dataSet.iloc[:,-1]
labels = Ytrain.unique().tolist()
Ytrain = Ytrain.apply(lambda x: labels.index(x))  #将本文转换为数字

#绘制树模型
clf = DecisionTreeClassifier()
clf = clf.fit(Xtrain, Ytrain)
tree.export_graphviz(clf)
dot_data = tree.export_graphviz(clf, out_file=None)
graphviz.Source(dot_data)

#给图形增加标签和颜色
dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=['no surfacing', 'flippers'],
                                class_names=['fish', 'not fish'],
                                filled=True, rounded=True,
                                special_characters=True)
graphviz.Source(dot_data)

#利用render方法生成图形
graph = graphviz.Source(dot_data)
graph.render("fish")
```

这样我们最终要的树模型就画出来啦，当然也可以手动编写函数来实现绘制树的这个过程，接下来我们一起来看看如何手动实现建树过程吧。

- 菊安酱的直播间：https://live.bilibili.com/14988341
- 下周一（2018/11/19）将讲解朴素贝叶斯算法，欢迎各位进入菊安酱的直播间观看直播
- 如有问题，可以给我留言哦~