第 6 章 决策树

内容提要

- 决策树简介
- 在数据集中度量一致性
- 使用递归构造决策树
- 使用 Matplotlib 绘制树形图

6.1 决策树

你是否玩过二十个问题的游戏，游戏的规则很简单：参与游戏的一方在脑海里想某个事物，其他参与者向他提问题，只允许提 20 个问题，问题的答案也只能用对或错回答。问问题的人通过推断分解，逐步缩小待猜测事物的范围。决策树的工作原理与 20 个问题类似，用户输入一系列数据，然后给出游戏的答案。

我们经常使用决策树处理分类问题，近来的调查表明决策树也是最经常使用的数据挖掘算法。它之所以如此流行，一个很重要的原因就是不需要了解机器学习的知识，就能搞明白决策树是如何工作的。

如果你以前没有接触过决策树，完全不用担心，它的概念非常简单。即使不知道它也可以通过简单的图形了解其工作原理，图6.1所示的流程图就是一个决策树，长方形代表判断模块（decision block），椭圆形代表终止模块（terminating block），表示已经得出结论，可以终止运行。从判断模块引出的左右箭头称作分支（branch），它可以到达另一个判断模块或者终止模块。图6.1 构造了一个假想的邮件分类系统，它首先检测发送邮件域名地址。如果地址为myEmployer.com，则将其放在分类“无聊时需要阅读的邮件”中。如果邮件不是来自这个域名，则检查邮件内容里是否包含单词曲棍球，如果包含则将邮件归类到“需要及时处理的朋友邮件”，如果不包含则将邮件归类到“无需阅读的垃圾邮件”。

k-近邻算法可以完成很多分类任务，但是它最大的缺点就是无法给出数据的内在含义，决策树的主要优势就在于数据形式非常容易理解。

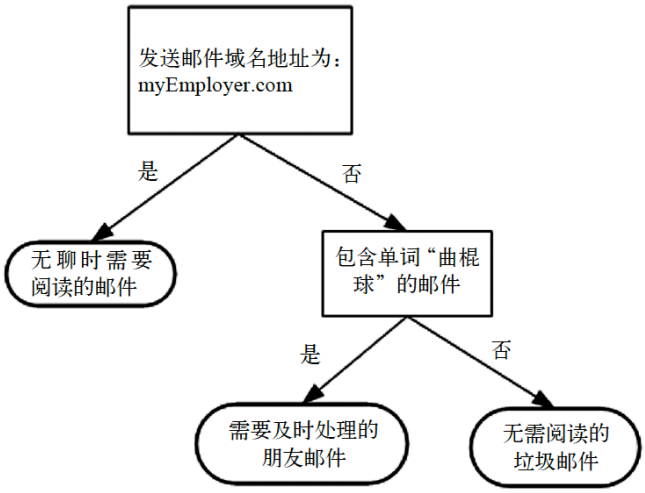

图 6.1: 流程图形式的决策树

本章构造的决策树算法能够读取数据集合，构建类似于图6.1的决策树。决策树的一个重要任务是为了数据中所蕴含的知识信息，因此决策树可以使用不熟悉的数据集合，并从中提取出一系列规则，在这些机器根据数据集创建规则时，就是机器学习的过程。专家系统中经常使用决策树，而且决策树给出结果往往可以匹敌在当前领域具有几十年工作经验的人类专家。

现在我们已经大致了解了决策树可以完成哪些任务，接下来我们将学习如何从一堆原始数据中构造决策树。首先我们讨论构造决策树的方法，以及如何编写构造树的 Python 代码；接着提出一些度量算法成功率的方法；最后使用递归建立分类器，并且使用 Matplotlib 绘制决策树图。构造完成决策树分类器之后，我们将输入一些隐形眼镜的处方数据，并由决策树分类器预测需要的镜片类型。

6.1.1 决策树的构造

在构造决策树时，我们需要解决的第一个问题就是，当前数据集上哪个特征在划分数据分类时起决定性作用。为了找到决定性的特征，划分出最好的结果，我们必须评估每个特征。完成测试之后，原始数据集就被划分为几个数据子集。这些数据子集会分布在第一个决策点的所有分支上。如果某个分支下的数据属于同一类型，则当前无需阅读的垃圾邮件已经正确地划分数据分类，无需进一步对数据集进行分割。如果数据子集内的数据不属于同一类型，则需要重复划分数据子集的过程。如何划分数据子集的算法和划分原始数据集的方法相同，直到所有具有相同类型的数据均在一个数据子集内。

创建分支的伪代码函数 createBranch() 如下所示：

```
检测数据集中的每个子项是否属于同一分类:
    If so return 类标签;
    Else
        寻找划分数据集的最好特征
        划分数据集
        创建分支节点
            for 每个划分的子集
                调用函数createBranch并增加返回结果到分支节点中
        return 分支节点
```

上面的伪代码 createBranch 是一个递归函数，在倒数第二行直接调用了它自己。后面我们将把上面的伪代码转换为 Python 代码，这里我们需要进一步了解算法是如何划分数据集的。

我们将使用 ID3 算法划分数据集，该算法处理如何划分数据集，何时停止划分数据集（进一步的信息可以参见Wiki 。每次划分数据集时我们只选取一个特征属性，如果训练集中存在 20 个特征，第一次我们选择哪个特征作为划分的参考属性呢?

表 3-1 的数据包含 5 个海洋动物，特征包括：不浮出水面是否可以生存，以及是否有脚蹼。我们可以将这些动物分成两类：鱼类和非鱼类。现在我们想要决定依据第一个特征还是第二个特征划分数据。在回答这个问题之前，我们必须采用量化的方法判断如何划分数据。

表 6.1: 海洋生物数据

Index	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

6.1.2 信息增益

划分数据集的大原则是：将无序的数据变得更加有序。我们可以使用多种方法划分数据集，但是每种方法都有各自的优缺点。组织杂乱无章数据的一种方法就是使用信息论度量信息，信息论是量化处理信息的分支科学。我们可以在划分数据之前或之后使用信息论量化度量信息的内容。

在划分数据集之前之后信息发生的变化称为信息增益，知道如何计算信息增益，我们就可以计算每个特征值划分数据集获得的信息增益，获得信息增益最高的特征就是最好的选择。

在可以评测哪种数据划分方式是最好的数据划分之前，我们必须学习如何计算信息增益。集合信息的度量方式称为香农熵或者简称为熵。

熵定义为信息的期望值，在明晰这个概念之前，我们必须知道信息的定义。如果待分类的事务可能划分在多个分类之中，则符号 x_i 的信息定义为

$$l(x_i) = -\log_2 p(x_i) \tag{6.1}$$

其中 $p(x_i)$ 是选择该分类的概率。

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，通过下面的公式得到：

$$H = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{6.2}$$

其中 n 是分类的数目。

计算给定数据集的香农熵:

```
"""
Parameters:
    dataSet - 数据集
Returns:
    shannonEnt - 经验熵(香农熵)
"""
# 函数说明:计算给定数据集的经验熵(香农熵)
def calcShannonEnt(dataSet):
    numEntires = len(dataSet)                        #返回数据集的行数
    labelCounts = {}                                 #保存每个标签(Label)出现次数的字典
    for featVec in dataSet:                          #对每组特征向量进行统计
        currentLabel = featVec[-1]                   #提取标签(Label)信息
        if currentLabel not in labelCounts.keys(): #如果标签(Label)没有放入统计次数的字典,添加进去
```

```
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1          #Label计数
    shannonEnt = 0.0                            #经验熵(香农熵)
    for key in labelCounts:                     #计算香农熵
        prob = float(labelCounts[key]) / numEntires #选择该标签(Label)的概率
        shannonEnt -= prob * log(prob, 2)       #利用公式计算
    return shannonEnt                           #返回经验熵(香农熵)
```

```
"""
Parameters:
    无
Returns:
    dataSet - 数据集
    labels - 分类属性
"""
# 函数说明:创建测试数据集
def createDataSet():
    dataSet = [[1, 1, 'yes'],#数据集
            [1, 1, 'yes'],
            [1, 0, 'no'],
            [0, 1, 'no'],
            [0, 1, 'no']]
    labels = ['no surfacing', 'flippers']#分类属性
    return dataSet, labels#返回数据集和分类属性
```

```
dataSet, features = createDataSet()
print(dataSet)
print(calcShannonEnt(dataSet))
```

熵越高，则混合的数据也越多，我们可以在数据集中添加更多的分类，观察熵是如何变化的。

6.1.3 划分数据集

分类算法除了需要测量信息熵，还需要划分数据集，度量划分数据集的熵，以便判断当前是否正确地划分了数据集。我们将对每个特征划分数据集的结果计算一次信息熵，然后判断按照哪个特征划分数据集是最好的划分方式。想象一个分布在二维空间的数据散点图，需要在数据之间划条线，将它们分成两部分，我们应该按照 x 轴还是 y 轴划线呢？答案就是本节讲述的内容。

按照给定特征划分数据集

```
"""
Parameters:
    dataSet - 待划分的数据集
    axis - 划分数据集的特征
    value - 需要返回的特征的值
Returns:
    无
```

```
"""
# 函数说明:按照给定特征划分数据集
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
```

代码使用了三个输入参数：待划分的数据集、划分数据集的特征、需要返回的特征的值。需要注意的是，Python 语言不用考虑内存分配问题。Python 语言在函数中传递的是列表的引用，在函数内部对列表对象的修改，将会影响该列表对象的整个生存周期。为了消除这个不良影响，我们需要在函数的开始声明一个新列表对象。因为该函数代码在同一数据集上被调用多次，为了不修改原始数据集，创建一个新的列表对象。数据集这个列表中的各个元素也是列表，我们要遍历数据集中的每个元素，一旦发现符合要求的值，则将其添加到新创建的列表中。在 if 语句中，程序将符合特征的数据抽取出来。后面讲述得更简单，这里我们可以这样理解这段代码：当我们按照某个特征划分数据集时，就需要将所有符合要求的元素抽取出来。代码中使用了 Python 语言列表类型自带的 extend() 和 append() 方法。这两个方法功能类似，但是在处理多个列表时，这两个方法的处理结果是完全不同的。

选择最好的数据集划分方式:

```
"""
Parameters:
    dataSet - 数据集
Returns:
    bestFeature - 信息增益最大的(最优)特征的索引值
"""
# 函数说明:选择最优特征
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
```

该函数实现选取特征，划分数据集，计算得出最好的划分数据集的特征。在函数中调用的数据需要满足一定的要求：第一个要求是，数据必须是一种由列表元素组成的列表，而且所有的列表元素都要具有相同的数据长度；第二个要求是，数据的最后一列或者每个实例的最后一个元素是当前实例的类别标签。数据集一旦满足上述要求，我们就可以在函数的第一行判定当前数据集包含多少特征属性。我们无需限定 list 中的数据类型，它们既可以是数字也可以是字符串，并不影响实际计算。

在开始划分数据集之前，程序的第 3 行代码计算了整个数据集的原始香农熵，我们保存最初的无序度量值，用于与划分完之后的数据集计算的熵值进行比较。第 1 个 for 循环遍历数据集中的所有特征。使用列表推导（List Comprehension）来创建新的列表，将数据集中所有第 i 个特征值或者所有可能存在的值写入这个新 list 中。然后使用 Python 语言原生的集合（set）数据类型。集合数据类型与列表类型相似，不同之处仅在于集合类型中的每个值互不相同。从列表中创建集合是 Python 语言得到列表中唯一元素值的最快方法。

遍历当前特征中的所有唯一属性值，对每个特征划分一次数据集，然后计算数据集的新熵值，并对所有唯一特征值得到的熵求和。信息增益是熵的减少或者是数据无序度的减少，大家肯定对于将熵用于度量数据无序度的减少更容易理解。最后，比较所有特征中的信息增益，返回最好特征划分的索引值。

现在我们可以测试上面代码的实际输出结果：

```
>>> dataSet, features = createDataSet()
>>> chooseBestFeatureToSplit(dataSet)
0
```

代码运行结果告诉我们，第 0 个特征是最好的用于划分数据集的特征。结果是否正确呢？这个结果又有什么实际意义呢？数据集中的数据来源于表 3-1，让我们回头再看一下表 1-1 或者变量 myDat 中的数据。如果我们按照第一个特征属性划分数据，也就是说第一个特征是 1 的放在一个组，第一个特征是 0 的放在另一个组，数据一致性如何？按照上述的方法划分数据集，第一个特征为 1 的海洋生物分组将有两个属于鱼类，一个属于非鱼类；另一个分组则全部属于非鱼类。如果按照第二个特征分组，结果又是怎么样呢？第一个海洋动物分组将有两个属于鱼类，两个属于非鱼类；另一个分组则只有一个非鱼类。第一种划分很好地处理了相关数据。

接下来我们将信息熵，划分数据集放在一起，构建决策树。

6.1.4 递归构建决策树

决策树工作原理如下：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，数据将被向下传递到树分支的下一个节点，在这个节点上，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。

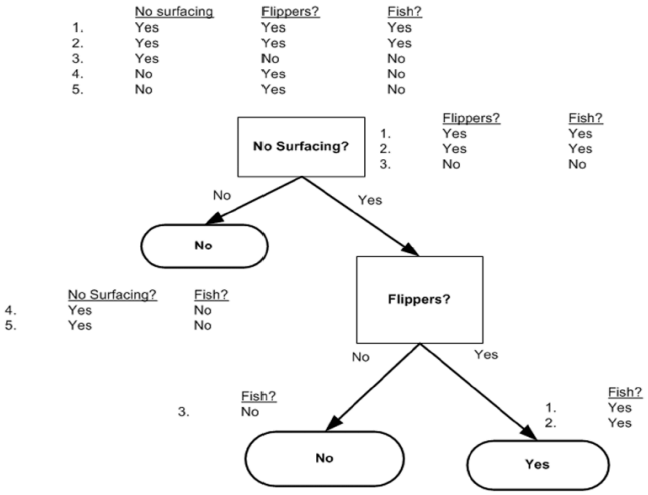

图 6.2: 划分数据集时的数据路径

递归结束的条件是：程序遍历完所有划分数据集的属性，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的分类，则得到一个叶子节点或者终止块。任何到达叶子节点的数据必然属于叶子节点的分类

第一个结束条件使得算法可以终止，我们甚至可以设置算法可以划分的最大分组数目。后续章节还会介绍其他决策树算法，如 C4.5 和 CART，这些算法在运行时并不总是在每次划分分组时都会消耗特征。由于特征数目并不是在每次划分数据分组时都减少，因此这些算法在实际使用时可能引起一定的问题。目前我们并不需要考虑这个问题，只需要在算法开始运行前计算列的数目，查看算法是否使用了所有属性即可。如果数据集已经处理了所有属性，但是类标签依然不是唯一的，此时我们需要决定如何定义该叶子节点，在这种情况下，我们通常会采用多数表决的方法决定该叶子节点的分类。

```
import operator

"""
Parameters:
    classList - 类标签列表
Returns:
    sortedClassCount[0][0] - 出现此处最多的元素(类标签)
"""
# 函数说明:统计classList中出现此处最多的元素(类标签)
def majorityCnt(classList):
    classCount = {}
    for vote in classList:#统计classList中每个元素出现的次数
        if vote not in classCount.keys():classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.items(), key = operator.itemgetter(1), reverse = True)#根据字
        典的值降序排序
    return sortedClassCount[0][0]#返回classList中出现次数最多的元素
```

该函数使用分类名称的列表，然后创建键值为 classList 中唯一值的数据字典，字典对象存储了 classList 中每个类标签出现的频率，最后利用 operator 操作键值排序字典，并返回出现

次数最多的分类名称。

创建树的函数代码:

```
"""
Parameters:
    dataSet - 训练数据集
    labels - 分类属性标签
    featLabels - 存储选择的最优特征标签
Returns:
    myTree - 决策树
"""
# 函数说明:创建决策树
def createTree(dataSet, labels, featLabels):
    classList = [example[-1] for example in dataSet] #取分类标签(是否放贷:yes or no)
    if classList.count(classList[0]) == len(classList): #如果类别完全相同则停止继续划分
        return classList[0]
    if len(dataSet[0]) == 1:                          #遍历完所有特征时返回出现次数最多的类标签
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)      #选择最优特征
    bestFeatLabel = labels[bestFeat]                  #最优特征的标签
    featLabels.append(bestFeatLabel)
    myTree = {bestFeatLabel:{}}                       #根据最优特征的标签生成树
    del(labels[bestFeat])                             #删除已经使用特征标签
    featValues = [example[bestFeat] for example in dataSet]#得到训练集中所有最优特征的属性值
    uniqueVals = set(featValues)                      #去掉重复的属性值
    for value in uniqueVals:                          #遍历特征，创建决策树。
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), labels,
            featLabels)
    return myTree
```

代码使用两个输入参数：数据集和标签列表。标签列表包含了数据集中所有特征的标签，算法本身并不需要这个变量，但是为了给出数据明确的含义，我们将它作为一个输入参数提供。此外，前面提到的对数据集的要求这里依然需要满足。上述代码首先创建了名为 classList 的列表变量，其中包含了数据集的所有类标签。递归函数的第一个停止条件是所有的类标签完全相同，则直接返回该类标签。递归函数的第二个停止条件是使用完了所有特征，仍然不能将数据集划分成仅包含唯一类别的分组。由于第二个条件无法简单地返回唯一的类标签，这里使用选择最好的数据集划分方式的函数挑选出现次数最多的类别作为返回值。

下一步程序开始创建树，这里使用 Python 语言的字典类型存储树的信息，当然也可以声明特殊的数据类型存储树，但是这里完全没有必要。字典变量 myTree 存储了树的所有信息，这对于其后绘制树形图非常重要。当前数据集选取的最好特征存储在变量 bestFeat 中，得到列表包含的所有属性值。

最后代码遍历当前选择特征包含的所有属性值，在每个数据集划分上递归调用函数 createTree()，得到的返回值将被插入到字典变量 myTree 中，因此函数终止执行时，字典中将会嵌套很多代表叶子节点信息的字典数据。在解释这个嵌套数据之前，我们先看一下循环的第一行 subLabels = labels[:]，这行代码复制了类标签，并将其存储在新列表变量 subLabels 中。之所

以这样做，是因为在 Python 语言中函数参数是列表类型时，参数是按照引用方式传递的。为了保证每次调用函数 createTree() 时不改变原始列表的内容，使用新变量 subLabels 代替原始列表。

现在我们可以测试上面代码的实际输出结果:

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

变量 myTree 包含了很多代表树结构信息的嵌套字典，从左边开始，第一个关键字 no surfacing 是第一个划分数据集的特征名称，该关键字的值也是另一个数据字典。第二个关键字是 no surfacing 特征划分的数据集，这些关键字的值是 no surfacing 节点的子节点。这些值可能是类标签，也可能是另一个数据字典。如果值是类标签，则该子节点是叶子节点；如果值是另一个数据字典，则子节点是一个判断节点，这种格式结构不断重复就构成了整棵树。本节的例子中，这棵树包含了 3 个叶子节点以及 2 个判断节点。

6.1.5 测试算法：使用决策树执行分类

依靠训练数据构造了决策树之后，我们可以将它用于实际数据的分类。在执行数据分类时，需要决策树以及用于构造树的标签向量。然后，程序比较测试数据与决策树上的数值，递归执行该过程直到进入叶子节点；最后将测试数据定义为叶子节点所属的类型。

为了验证算法的实际效果:

```
"""
Parameters:
    inputTree - 已经生成的决策树
    featLabels - 存储选择的最优特征标签
    testVec - 测试数据列表，顺序对应最优特征标签
Returns:
    classLabel - 分类结果
"""
# 函数说明:使用决策树分类
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
```

程序定义的函数也是一个递归函数，在存储带有特征的数据会面临一个问题：程序无法确定特征在数据集中的位置，例如前面例子的第一个用于划分数据集的特征是 no surfacing 属性，但是在实际数据集中该属性存储在哪个位置？是第一个属性还是第二个属性？特征标签列表将帮助程序处理这个问题。使用 index 方法查找当前列表中第一个匹配 firstStr 变量的元

素。然后代码递归遍历整棵树，比较 testVec 变量中的值与树节点的值，如果到达叶子节点，则返回当前节点的分类标签。

```
dataSet, labels = createDataSet()
featLabels = []
myTree = createTree(dataSet, labels, featLabels)
testVec = [0,1]#测试数据
result = classify(myTree, featLabels, testVec)
```

6.1.6 使用算法：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，如前面的样本数据，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。然而用创建好的决策树解决分类问题，则可以很快完成。因此，为了节省计算时间，最好能够在每次执行分类时调用已经构造好的决策树。为了解决这个问题，需要使用 Python 模块 pickle 序列化对象，参见程序。序列化对象可以在磁盘上保存对象，并在需要的时候读取出来。任何对象都可以执行序列化操作，字典对象也不例外。

使用 pickle 模块存储决策树:

```
def storeTree(inputTree, filename):
    with open(filename, 'wb') as fw:
        pickle.dump(inputTree, fw)


import pickle

"""
Parameters:
    filename - 决策树的存储文件名
Returns:
    pickle.load(fr) - 决策树字典
"""
# 函数说明:读取决策树
def grabTree(filename):
    fr = open(filename, 'rb')
    return pickle.load(fr)

myTree = grabTree('classifierStorage.txt')
```

6.2 Sklearn 实现决策树

我们使用决策树来创建一个能屏蔽网页横幅广告的软件，这个软件将预测一个网页中每一张图片是否是一个广告还是文章内容。

数据集从学在浙大上下载

```
import pandas as pd
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.pipeline import Pipeline
from sklearn.grid_search import GridSearchCV

df = pd.read_csv('./ad.data', header=None)

explanatory_variable_columns = set(df.columns.values)
explanatory_variable_columns.remove(len(df.columns.values)-1)
response_variable_column = df[len(df.columns.values)-1] # The last column describes the classes

y = [1 if e == 'ad.' else 0 for e in response_variable_column]
X = df[list(explanatory_variable_columns)].copy()
X.replace(to_replace=' *?', value=-1, regex=True, inplace=True)
X_train, X_test, y_train, y_test = train_test_split(X, y)

pipeline = Pipeline([
    ('clf', DecisionTreeClassifier(criterion='entropy'))
])
parameters = {
    'clf__max_depth': (150, 155, 160),
    'clf__min_samples_split': (2, 3),
    'clf__min_samples_leaf': (1, 2, 3)
}

grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1, verbose=1, scoring='f1')
grid_search.fit(X_train, y_train)

best_parameters = grid_search.best_estimator_.get_params()
print('Best score: %0.3f' % grid_search.best_score_)
print('Best parameters set:')
for param_name in sorted(parameters.keys()):
    print('t%s: %r' % (param_name, best_parameters[param_name]))

predictions = grid_search.predict(X_test)
print(classification_report(y_test, predictions))


Best score: 0.887
Best parameters set:
tclf__max_depth: 150
tclf__min_samples_leaf: 1
tclf__min_samples_split: 3
            precision   recall f1-score  support

         0       0.98     0.99     0.98      717
         1       0.92     0.83     0.87      103
```

```
avg / total       0.97      0.97      0.97       820
```

第 6 章 练习

1. 使用决策树预测糖尿病从学在浙大下载数据集：

About Dataset

Context

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Content

The datasets consists of **several** medical predictor variables and **one** target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

```
# 导入数据
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
pima = pd.read_csv("pima-indians-diabetes.csv", header=None, names=col_names)

pima.head()

# 选择预测所需的特征
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # 特征
y = pima.label # 类别标签

# 将数据分为训练和测试数据


# 创建决策树分类器

clf =
# 训练模型


# 使用训练好的模型做预测

y_pred =
```

```
# 模型的准确性
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

可视化训练好的决策树模型

```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```

```
# 创建新的决策树，限定树的最大深度，减少过拟合
clf =

# 训练模型
clf =

# 预测
y_pred = clf.predict(X_test)

# 模型的性能
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

可视化训练好的决策树模型

```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
```