文本分类

杨善超

2016 年 4 月 20 日

[1]姓名: 杨善超
[2]学号: 13020510013
[3]邮箱: yangshanchaoysc@gmail.com

目　录

1 爬取数据

2 中文分词

ID3 决策树有两个缺点 [1]：

(1) 用信息增益选择属性时偏向于选择分枝比较多的属性值，即取值多的属性

(2) 不能处理连贯属性

而如表 2 所示, 题目所给数据的两个属性**湿度**和**温度**是连续值, 所以要对其进行离散化处理。此处采用论文中 C4.5 算法处理连续属性的方法 [1]。

> Attributes can be either numeric or nominal and this determines the format of the test outcomes.For a numeric attribute A they are $\{A \leq h, A > h\}$ where the threshold h is found by sorting S on the values of A and choosing the split between successive values that maximizes the criterion above.

经编程计算后得到，温度最佳分割点为 **83**，湿度最佳分割点为 **80**，并将高于阈值的属性值标记为“高”，不高于阈值的属性值标记为“正常”。

3 实验结果

3.1 代码实现

本次程序代码中 ID3 算法的核心部分基于《机器学习实战》[2]。另外进行的额外工作为：

(1) 使用 Pandas 库替代 Numpy 库进行数据分析

(2) 编写处理连续属性值的函数

(3) 重新编写决策树的分类函数

(4) 编写程序读入文件等程序

(5) 更改原书 2.x 代码，使之在 3.5 版本上可以工作 (绘图程序可以自动绘制决策树)

3.2 运行结果

3.3 结果分析

正如图??所示, 人们在使用决策树进行决策时, 只需要使用**天气**、**湿度**、**风况**三个属性即可完成最终决策，不需要使用温度属性。究其原因，现实生活中除非剧烈高温和低温，否则温度对人们是否室外活动的影响很小，而天气当然是最重要的因素，其次风况对于室外活动则影响很大，比如羽毛球等运动。湿度的范围倘若不适合运动，则对人的身体会产生较大影响。

输入新样本属性值, 进行决策后结果如表1所示：

表 1: 不同属性下决策结果

天气	温度	湿度	风况	运动
晴	寒冷	正常	有	适合
晴	寒冷	高	无	不适合

4 参考文献

[1] Top 10 algorithms in data mining. *Knowledge and Information Systems*[M]. 2008(1), Volume 14, 3–4.

[2] Peter Harringtoo 著, 李悦等译. 机器学习实战 [M]. 人民邮电出版社. 2015(1): 32–52.

5 附录

5.1 程序代码

```python
# 主运行函数
from math import log
import pandas as pd
import treePlotter as tP


# 计算湿度和温度的最佳分割点，数据规约后返回 dataSet
def getBestSplit():
    dataSet = pd.read_csv('weatherData.csv', encoding='gbk')
    labels = list(dataSet.columns[:-1])
    # 按温度进行升序排序
    sortedData = dataSet.sort_values(by=dataSet.columns[1])
    # 计算温度最佳分割点
    # 只对属性值发生改变的地方才需要切开
    # 记录信息增益以及最优分割点
    infGain = 0
    bestSplit = -1
    # 初始样本的经验熵
    HD = calcShannonEnt(dataSet)
    for pos, item in enumerate(sortedData.values[:-2, :]):
        prob = (pos + 1) / len(sortedData)
        newEnt = prob * calcShannonEnt(sortedData.iloc[:pos+1, :]) + (1 - prob) * calcShannonEnt(sortedData.iloc[pos+1:, :])
        newEnt = HD - newEnt
        if newEnt > infGain:
            infGain = newEnt
            bestSplit = item[1]
    temperatureSplit = bestSplit
    print('温度最佳分割点为 %d' % temperatureSplit)
    # 计算湿度最佳分割点
     # 按湿度进行升序排序
    sortedData = dataSet.sort_values(by=dataSet.columns[2])
    # 计算湿度最佳分割点
    # 只对属性值发生改变的地方才需要切开
```

```
    # 记录最优分割点以及信息增益
    infGain = 0
    bestSplit = -1
    # 初始样本的经验熵
    # HD = calcShannonEnt(dataSet)
    for pos, item in enumerate(sortedData.values[:-2, :]):
        # if item[-1] != sortedData.iloc[pos + 1, -1]:
        prob = (pos + 1) / len(sortedData)
        newEnt = prob * calcShannonEnt(sortedData.iloc[:pos+1, :]) + (1 - prob) * calcShannonEnt(sortedData.iloc[pos+1:, :])
        newEnt = HD - newEnt
        if newEnt > infGain:
            infGain = newEnt
            bestSplit = item[2]
    humiditySplit = bestSplit
    print('湿度最佳分割点为 %d' % humiditySplit)
    # 将高于阈值的属性值标记为“高”，不高于阈值的属性值标记为“正常”
    highPos = dataSet.iloc[:, 1] > temperatureSplit
    lowPos = dataSet.iloc[:, 1] <= temperatureSplit
    dataSet.ix[highPos, 1] = '高'
    dataSet.ix[lowPos, 1] = '正常'
    highPos = dataSet.iloc[:, 2] > humiditySplit
    lowPos = dataSet.iloc[:, 2] <= humiditySplit
    dataSet.ix[highPos, 2] = '高'
    dataSet.ix[lowPos, 2] = '正常'
    return dataSet, labels

# 计算给定的熵DataSetShannnon
def calcShannonEnt(dataSet):
    # 计算类别的分布
    numEntries = len(dataSet)
    labelEntries = dataSet.iloc[:, -1]
    # 计算频度
    labelCounts = labelEntries.value_counts()
    # 计算熵
    shannonEnt = 0
    for count in labelCounts:
        prob = count/numEntries
```

```
        # log base 2
        shannonEnt -= prob * log(prob, 2)
    return shannonEnt

# 对取出dataSet 第轴上值为的子数据aixsvalue
def splitDataSet(dataSet, axis, value):
    retDataSet = dataSet.loc[dataSet.iloc[:, axis] == value]
    del retDataSet[retDataSet.columns[axis]]
    return retDataSet

# 根据信息增益选择最好的划分属性
def chooseBestFeatureToSplit(dataSet):
    # the last column is used for the labels
    numFeatures = len(dataSet.iloc[0]) - 1
    baseEntropy = calcShannonEnt(dataSet)
    bestInfoGain = 0.0
    bestFeature = -1
    for i in range(numFeatures):
        # create a list of all the examples of this feature
        featList = dataSet.iloc[:, i]
        # get a set of unique values
        uniqueVals = set(featList)
        newEntropy = 0.0
        for value in uniqueVals:
            subDataSet = splitDataSet(dataSet, i, value)
            prob = len(subDataSet)/float(len(dataSet))
            newEntropy += prob * calcShannonEnt(subDataSet)
        # calculate the info gain; ie reduction in entropy
        infoGain = baseEntropy - newEntropy
        # compare this to the best gain so far
        if infoGain > bestInfoGain:
            # if better than current best, set to best
            bestInfoGain = infoGain
            bestFeature = i
    # returns an integer
    return bestFeature

# 多数表决
```

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

# 递归创建决策树
def createTree(dataSet,labels):
    classList = list(dataSet.iloc[:, -1])
    if classList.count(classList[0]) == len(classList):
        # stop splitting when all of the classes are equal
        return classList[0]
    if len(dataSet.iloc[0]) == 1:
        # stop splitting when there are no more features in dataSet
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)
    bestFeatLabel = labels[bestFeat]
    myTree = {bestFeatLabel: {}}
    del(labels[bestFeat])
    featValues = list(dataSet.iloc[:, bestFeat])
    uniqueVals = set(featValues)
    for value in uniqueVals:
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), labels[:])

    return myTree

# given a problem instance, 决策是否适合运动
def decision(tree, instance):
    outcome = tree
    while type(outcome) == dict:
        # 得到决策树当前需要决策的属性
        key = list(outcome.keys())[0]
        # 得到实际的属性值instance
```

```python
        pos = labels.index(key)
        keyVal = instance[pos]
        # get 结果
        outcome = outcome[key][keyVal]
    return outcome

if __name__ == '__main__':
    # 读入数据
    dataSet, labels = getBestSplit()
    # dataSet, labels = createDataSet()

    # 计算出决策树
    tree = createTree(dataSet, labels[:])

    # test two instances
    instance = ['晴', '寒冷', '正常', '有']
    decide = decision(tree, instance)
    print(', '.join(instance), '->', decide)
    instance = ['晴', '寒冷', '高', '无']
    decide = decision(tree, instance)
    print(', '.join(instance), '->', decide)

    # 绘制决策树
    tP.createPlot(tree)
```

```python
# 绘图程序
import matplotlib.pyplot as plt
import matplotlib as mpl
# 指定默认字体
mpl.rcParams['font.sans-serif'] = ['SimHei']
# 自定义决策节点、叶子节点以及箭头的绘制属性（颜色什么的)
decisionNode = dict(boxstyle="sawtooth", fc="c", edgecolor='b')
leafNode = dict(boxstyle="round", facecolor="0.8", edgecolor='r')
arrow_args = dict(arrowstyle="<-", facecolor='b')


def getNumLeafs(myTree):
    numLeafs = 0
```

```
    firstStr = list(myTree.keys())[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        # test to see if the nodes are dictonaires, if not they are leaf nodes
        if type(secondDict[key]) == dict:
            numLeafs += getNumLeafs(secondDict[key])
        else:
            numLeafs += 1
    return numLeafs

def getTreeDepth(myTree):
    maxDepth = 0
    firstStr = list(myTree.keys())[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        if type(secondDict[key]) == dict:
            # test to see if the nodes are dictonaires, if not they are leaf nodes
            thisDepth = 1 + getTreeDepth(secondDict[key])
        else:
            thisDepth = 1
        if thisDepth > maxDepth:
            maxDepth = thisDepth
    return maxDepth

def plotNode(nodeTxt, centerPt, parentPt, nodeType):
    createPlot.ax1.annotate(nodeTxt, xy=parentPt, xycoords='axes fraction',
                            xytext=centerPt, textcoords='axes fraction', va="center", ha="center", bbox=nodeType,
                            arrowprops=arrow_args)

def plotMidText(cntrPt, parentPt, txtString):
    xMid = (parentPt[0]-cntrPt[0])/2.0 + cntrPt[0]
    yMid = (parentPt[1]-cntrPt[1])/2.0 + cntrPt[1]
    createPlot.ax1.text(xMid, yMid, txtString, va="center", ha="center", rotation=30)
```

```


# if the first key tells you what feat was split on
def plotTree(myTree, parentPt, nodeTxt):
    # this determines the x width of this tree
    numLeafs = getNumLeafs(myTree)
    # the text label for this node should be this
    firstStr = list(myTree.keys())[0]
    cntrPt = (plotTree.xOff + (1.0 + float(numLeafs))/2.0/plotTree.totalW, plotTree.yOff)
    plotMidText(cntrPt, parentPt, nodeTxt)
    plotNode(firstStr, cntrPt, parentPt, decisionNode)
    secondDict = myTree[firstStr]
    plotTree.yOff = plotTree.yOff - 1.0/plotTree.totalD
    for key in secondDict.keys():
        # test to see if the nodes are dictonaires, if not they are leaf nodes
        if type(secondDict[key]) == dict:
            # recursion
            plotTree(secondDict[key], cntrPt, str(key))
        else:
            # it's a leaf node print the leaf node
            plotTree.xOff = plotTree.xOff + 1.0/plotTree.totalW
            plotNode(secondDict[key], (plotTree.xOff, plotTree.yOff), cntrPt, leafNode)
            plotMidText((plotTree.xOff, plotTree.yOff), cntrPt, str(key))
    plotTree.yOff = plotTree.yOff + 1.0/plotTree.totalD


# if you do get a dictonary you know it's a tree, and the first element will be another dict
def createPlot(inTree):
    fig = plt.figure(1, facecolor='white')
    fig.clf()
    axprops = dict(xticks=[], yticks=[])
    # no ticks
    createPlot.ax1 = plt.subplot(111, frameon=False, **axprops)

```

```
    plotTree.totalW = float(getNumLeafs(inTree))
    plotTree.totalD = float(getTreeDepth(inTree))
    plotTree.xOff = -0.5/plotTree.totalW
    plotTree.yOff = 1.0
    plotTree(inTree, (0.5, 1.0), '')
    plt.show()
```

5.2 原始数据

表 2: 天气对户外活动影响

天气	温度	湿度	风况	运动
晴	85.0	85.0	无	不适合
晴	85.0	90.0	有	不适合
多云	83.0	78.0	无	适合
有雨	70.0	96.0	无	适合
有雨	68.0	80.0	无	适合
有雨	65.0	70.0	有	不适合
多云	64.0	65.0	有	适合
晴	72.0	95.0	无	不适合
晴	69.0	70.0	无	适合
有雨	75.0	80.0	无	适合
晴	75.0	70.0	有	适合
多云	72.0	90.0	有	适合
多云	81.0	75.0	无	适合
有雨	71.0	80.0	有	不适合