机器学习与模式识别课程报告

题目：应用决策树方法判断活动举行状况

姓　　名：王 倩 妮

班　　级：交通 2015-02 班

学　　号：2015112956

任课教师：郝 莉

2018 年 3 月 23 日

一、问题描述

某活动举办与否与天气等条件有着直接关系，现需对多次活动记录进行分类，并由新的数据预测（确定）活动举办与否。

二、数据准备与预处理

2.1 数据描述

数据来自于网络中决策树案例数据，为标称型数据，有天气状况 weather、气温状况 temperature、湿度状况 humidity、风速 wind speed 四项特征，与活动是否举办的标签值：activity。

数据保存在 txt 文件中，每条数据占据 1 行，每行的不同特征间利用 tab 分割，每行的最后一个数据为标签，即活动是否举办。

本次所使用的函数对于数据的选用有如下要求：数据需满足由列表元素组成的列表，所有列表元素都需要具有相同的数据长度，且最后一列必须为类标签的特征。

2.2 数据预处理

查看数据，无缺失数据与异常数据，无需进行补全与修改操作，可以直接运用决策树方法进行分类。

三、数据可视化

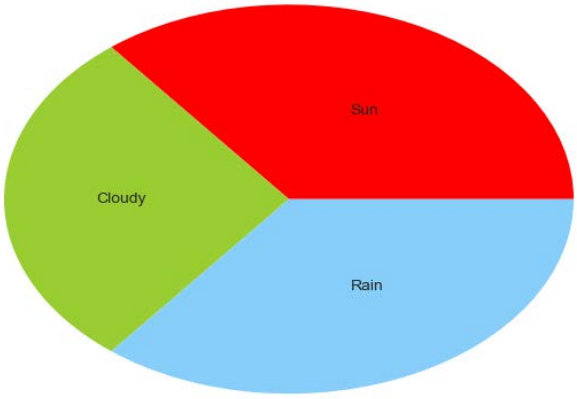


图 1 weather 天气状况

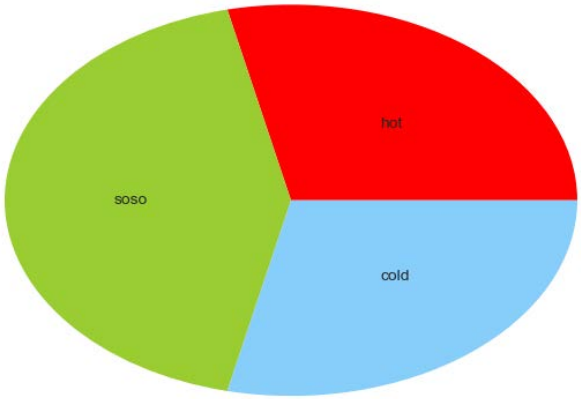


图 2 temperature 温度状况

图 3 humidity 湿度状况

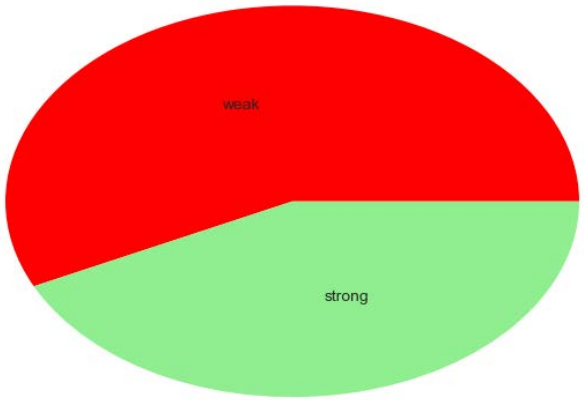


图 4 wind speed 风速状况

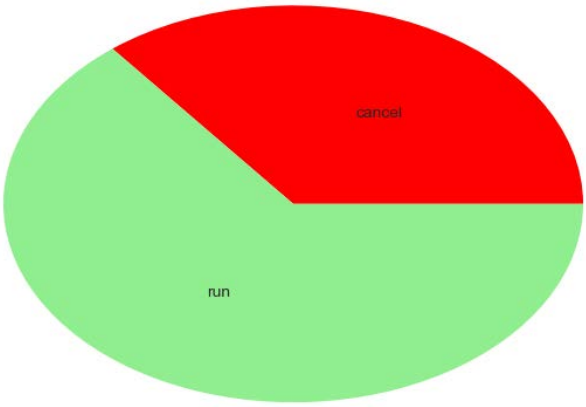


图 5 activity 活动举办情况

四、模型基本原理与算法实现

4.1 模型基本原理

本次决策树采用 ID3 算法实现，ID3 算法由 Ross Quinlan 发明，建立在“奥卡姆剃刀”的基础上：越是小型的决策树越优于大的决策树（be simple 简单理论）。ID3 算法中根据信息增益评估和选择特征，每次选择信息增益最大的特征作为判断模块建立子结点。ID3 算法可用于划分标称型数据集，没有剪枝的过程，为了去除过度数据匹配的问题，可通过裁剪合并相邻的无法产生大量信息增益的叶子节点（例如设置信息增益阀值）。使用信息增益的话其实是有一个缺点，那就是它偏向于具有大量值的属性。就是说在训练集中，某个属性所取的不同值的个数越多，那么越有可能拿它来作为分裂属性，而这样做有时候是没有意义的，另外 ID3 不能处理连续分布的数据特征，也衍生出了 CART 算法、C4.5 算法等其他决策树算法解决 ID3 算法的缺陷。

4.2 决策树算法实现

本次运用三个 py 文件存储需要实现的代码。tree.py 文件主要用于决策树的构建，treePlotter.py 文件主要用于绘制决策树分类示意图，run_new_data.py 文件主要用于导入数据、数据处理及决策树的运行。

① tree.py 文件

```
def calcShannonEnt(dataSet):
    numEntries = len(dataSet)
    labelCounts = {}
    for featVec in dataSet:
        currentLabel = featVec[-1]
        if currentLabel not in labelCounts.keys(): labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1
    shannonEnt = 0.0
    for key in labelCounts:
        prob = float(labelCounts[key])/numEntries
        shannonEnt -= prob * log(prob,2)
    return shannonEnt
```

calcShannonEnt 函数主要用于计算香农熵，决策树用于量化集合无序程度的方法主要有基尼系数法与熵法，此处我们选择与 ID3 算法相适应的熵法，计算一个数据集划分状况下的信息熵。

```
def splitDataSet(dataSet, axis, value):
    retDataSet = []
    for featVec in dataSet:
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]
            reducedFeatVec.extend(featVec[axis+1:])
            retDataSet.append(reducedFeatVec)
    return retDataSet
```

splitDataSet 主要用于按特征划分数据，通过输入待划分的数据集、划分数据集的特征、需返回的特征的值，将所有符合要求的元素抽取出来。

```
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1
    baseEntropy = calcShannonEnt(dataSet)
    bestInfoGain = 0.0; bestFeature = -1
    for i in range(numFeatures):
        featList = [example[i] for example in dataSet]
        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)
        infoGain = baseEntropy - newEntropy
        if (infoGain > bestInfoGain):
            bestInfoGain = infoGain
            bestFeature = i
    return bestFeature
```

chooseBestFeatureToSplit 函数用于通过调用 calcShannonEnt 与 splitDataSet 函数，计算每种划分方式的信息熵，通过计算出最好的信息增益，划分数据集。

```
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=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]
```

majorityCnt 函数利用排序的方法，确定每个类标签出现的频率，并返回出现次数最多的类标签名称。

```
def createTree(dataSet,labels):
    classList = [example[-1] for example in dataSet]
    if classList.count(classList[0]) == len(classList):
        return classList[0]#stop splitting when all of the classes are equal
    if len(dataSet[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 = [example[bestFeat] for example in dataSet]
    uniqueVals = set(featValues)
    for value in uniqueVals:
        subLabels = labels[:]       #copy all of labels, so trees don't mess up existing labels
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value),subLabels)
    return myTree
```

利用 createTree 函数调用以上各函数构建决策树，输入数据集与标签列表，分为三个部分，首先设定类别完全相同停止继续划分的条件，其次遍历全部特征值返回出现次数最多的类别，再次得到列表包含的所有属性值，最后将决策树构建为一个嵌套字典的样式返回。

② treePlotter.py 文件

此文件主要利用 matplotlib 模块，进行决策树的绘制，具体代码主要与画图有关，因此不再赘述。

③ run_new_data.py 文件

```
import trees
import treePlotter
fr=open("weather.txt")
data=[inst.strip().split('\t') for    inst in fr.readlines()]
data_label=['weather','temperature','humidity','wind speed','activity']
newdata_Tree = trees.createTree(data,data_label)
treePlotter.createPlot(newdata_Tree)
```

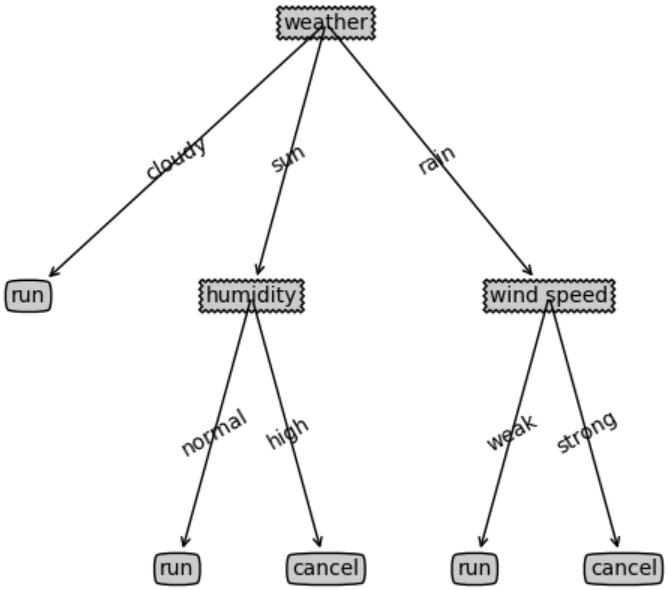
在此文件中首先导入决策树构建模块与决策树绘制模块，以备后续用途。导入数据文件，并将数据文件整合成为列表的形式，调用 createTree 函数构建决策树，并设定数据的输入。最终绘制决策树，完成分类器的构建。

五、测试方法与结果

5.1 测试方法

在测试阶段决策树构建完成，可以使用 pickle 模块进行决策树的存储与调用，当需进行活动举办与否的决策时，可以利用已经训练完成的决策树直接用于决策，或者预测活动的举办状况。

5.2 运行结果



由以上运行结果可知，活动举办与否与天气有着密切的关系，在天气多云的时候会矩形；当天气晴朗的时候取决于空气的湿度，当湿度正常时可以举办，当湿度过高时，活动取消；当天气下雨时取决于风速，当风速弱的时候，活动能够正常举办，当风速强的时候活动取消。

六、总结

决策树方法是分类算法的一种，决策树表示基于特征对实例进行分类的过程。它可以认为是 if-then 规则的集合，也可以认为是定义在特征空间与类空间上的条件概率分布。与朴素贝叶斯分类方法相比，决策树的优势在于构造过程不需要任何领域知识或参数设置，因此在实际应用中，决策树更加适用于探测式的知识发现。

在交通问题中，决策树方法也有着广泛的应用，比如在交通规划层面，进行居民出行调查时，会以问卷的形式对家庭、个人的经济状况、出行情况、居住情况、职业情况等进行调查。这些特征与居民出行方式的选择有着密切的关系，我们也可以利用决策数方法，对于居民出行调查的基础数据进行整合与学习，分析与居民出行选择密切相关的具体特征，从而更为准确地预测居民出行方式，为四阶段法的出行生成阶段、方式划分阶段提供思路。