数据科学 9 机器学习： 决策树2

决策树构建

ID3算法

ID3算法的核心是在决策树各个结点上对应信息增益准则选择特征，递归地构建决策树。

从根结点(root node)开始，对结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结点的特征，由该特征的不同取值建立子节点；再对子结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。

依然使用上节贷款数据演示

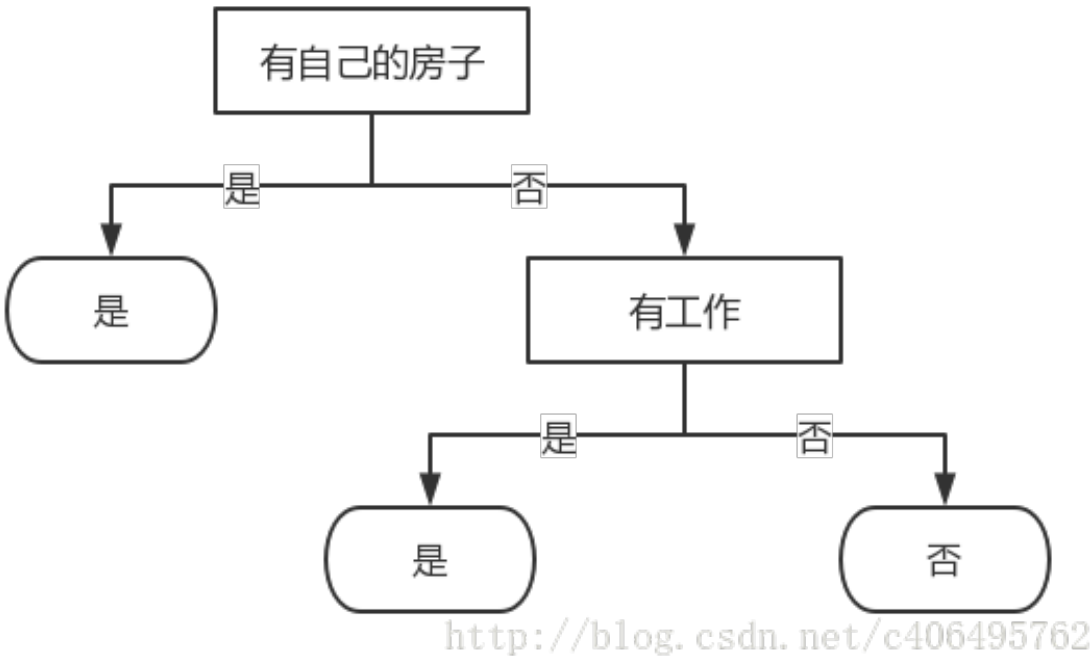
由于特征A3(有自己的房子)的信息增益值最大，所以选择特征A3作为根结点的特征。它将训练集D划分为两个子集D1(A3取值为"是")和D2(A3取值为"否")。由于D1只有同一类的样本点，所以它成为一个叶结点，结点的类标记为"是"。

对D2则需要从特征A1(年龄)，A2(有工作)和A4(信贷情况)中选择新的特征，计算各个特征的信息增益：

- g(D2,A1) = H(D2) - H(D2 | A1) = 0.251
- g(D2,A2) = H(D2) - H(D2 | A2) = 0.918
- g(D2,A3) = H(D2) - H(D2 | A3) = 0.474

根据计算，选择信息增益最大的特征A2(有工作)作为结点的特征。由于A2有两个可能取值，从这一结点引出两个子结点：一个对应"是"(有工作)的子结点，包含3个样本，它们属于同一类，所以这是一个叶结点，类标记为"是"；另一个是对应"否"(无工作)的子结点，包含6个样本，它们也属于同一类，所以这也是一个叶结点，类标记为"否"。

生成了一个决策树，该决策树只用了两个特征(有两个内部结点)



代码构建决策树

```
# 构建出来的结果
{'有自己的房子': {0: {'有工作': {0: 'no', 1: 'yes'}}, 1: 'yes'}}
```

```
"""
函数说明:统计classList中出现此处最多的元素(类标签)

Parameters:
    classList - 类标签列表
Returns:
    sortedClassCount[0][0] - 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
```

```
"""
函数说明:创建决策树

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
"""
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

if __name__ == '__main__':
    dataSet, labels = createDataSet()
    featLabels = []
    myTree = createTree(dataSet, labels, featLabels)
    print(myTree)
```

递归创建决策树时，递归有两个终止条件：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；第二个停止条件是使用完了所有特征，仍然不能将数据划分仅包含唯一类别的分组，即决策树构建失败，特征不够用。此时说明数据纬度不够，由于第二个停止条件无法简单地返回唯一的类标签，这里挑选出现数量最多的类别作为返回值。

使用决策树执行分类

```
"""
函数说明:使用决策树分类

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
"""
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
```

```
if __name__ == '__main__':
    dataSet, labels = createDataSet()
    featLabels = []
    myTree = createTree(dataSet, labels, featLabels)
    testVec = [0,1]                                                    #测试数据
    result = classify(myTree, featLabels, testVec)
    if result == 'yes':
        print('放贷')
    if result == 'no':
        print('不放贷')
```

决策树的存储

为了解决这个问题，需要使用Python模块pickle序列化对象。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。

```
import pickle

"""
函数说明:存储决策树

Parameters:
    inputTree - 已经生成的决策树
    filename - 决策树的存储文件名
Returns:
    无
"""
def storeTree(inputTree, filename):
    with open(filename, 'wb') as fw:
        pickle.dump(inputTree, fw)

if __name__ == '__main__':
    myTree = {'有自己的房子': {0: {'有工作': {0: 'no', 1: 'yes'}}, 1: 'yes'}}
    storeTree(myTree, 'classifierStorage.txt')
```

```
import pickle


"""
函数说明:读取决策树


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

if __name__ == '__main__':
    myTree = grabTree('classifierStorage.txt')
    print(myTree)
```