决策树自主实践与SKlearn调用

TA: 吕松霖

E-mail: lvsl@lamda.nju.edu.cn

实验内容

1. 从0构建决策树对简单数据分类
2. 可视化构建出的决策树
3. 调用SKlearn包中的决策树直接处理数据

实验目标

1. 从代码层面了解决策树的构建与计算
2. 学会直接调用现有决策树函数解决问题

实验内容

1 导入实验中的相关包

```
"""
pickle包可以将决策树保存下来，方便下次直接调用
"""
from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt
from math import log
import operator
import pickle
```

2 构建简单的分类数据集

```
1
```

```
"""
函数说明：创建测试数据集

Parameters:
    None

Returns:
    dataSet - 数据集
    labels - 分类属性

    用字典存储决策树结构：
    {'有自己的房子':{0:{'有工作':{0:'no', 1:'yes'}}, 1:'yes'}}
    年龄：0代表青年，1代表中年，2代表老年
    有工作：0代表否，1代表是
    有自己的房子：0代表否，1代表是
    信贷情况：0代表一般，1代表好，2代表非常好
    类别（是否给贷款）：no代表否，yes代表是
"""
def createDataSet():
    # 数据集
    dataSet = [[0, 0, 0, 0, 'no'],
               [0, 0, 0, 1, 'no'],
               [0, 1, 0, 1, 'yes'],
               [0, 1, 1, 0, 'yes'],
               [0, 0, 0, 0, 'no'],
               [1, 0, 0, 0, 'no'],
               [1, 0, 0, 1, 'no'],
               [1, 1, 1, 1, 'yes'],
               [1, 0, 1, 2, 'yes'],
               [1, 0, 1, 2, 'yes'],
               [2, 0, 1, 2, 'yes'],
               [2, 0, 1, 1, 'yes'],
               [2, 1, 0, 1, 'yes'],
               [2, 1, 0, 2, 'yes'],
               [2, 0, 0, 0, 'no']]
    # 分类属性
```

```
    labels = ['年龄', '有工作', '有自己的房子', '信贷情况']
    # 返回数据集和分类属性
    return dataSet, labels
```

3 计算香农熵函数

```
"""
函数说明：计算给定数据集的经验熵（香农熵）
        Ent（D） = -SUM（kp*Log2（kp））

Parameters:
    dataSet - 数据集

Returns:
    shannonEnt - 经验熵（香农熵）
"""
def calcShannonEnt(dataSet):
    # 返回数据集的行数
    numEntires = len(dataSet)
    # 保存每个标签（Label）出现次数的“字典”
    labelCounts = {}
    # 对每组特征向量进行统计
    for featVec in dataSet:
        # 提取标签（Label）信息
        currentLabel = featVec[-1]
        # 如果标签（Label）没有放入统计次数的字典，添加进去
        if currentLabel not in labelCounts.keys():
            # 创建一个新的键值对，键为currentLabel值为0
            labelCounts[currentLabel] = 0
        # Label计数
        labelCounts[currentLabel] += 1
    # 经验熵（香农熵）
    shannonEnt = 0.0
    # 计算香农熵
    for key in labelCounts:
```

```
        # 选择该标签（Label）的概率
        prob = float(labelCounts[key]) / numEntires
        # 利用公式计算
        shannonEnt -= prob*log(prob, 2)
    # 返回经验熵（香农熵）
    return shannonEnt

"""
函数说明：选择最优特征
        Gain(D,g) = Ent(D) - SUM(|Dv|/|D|)*Ent(Dv)

Parameters:
    dataSet - 数据集

Returns:
    bestFeature - 信息增益最大的（最优）特征的索引值
"""
def chooseBestFeatureToSplit(dataSet):
    # 特征数量
    numFeatures = len(dataSet[0]) - 1
    # 计算数据集的香农熵
    baseEntropy = calcShannonEnt(dataSet)
    # 信息增益
    bestInfoGain = 0.0
    # 最优特征的索引值
    bestFeature = -1
    # 遍历所有特征
    for i in range(numFeatures):
        # 获取dataSet的第i个所有特征存在featList这个列表中（列表生成式）
        featList = [example[i] for example in dataSet]
        # 创建set集合{}，元素不可重复，重复的元素均被删掉
        # 从列表中创建集合是python语言得到列表中唯一元素值得最快方法
        uniqueVals = set(featList)
        # 经验条件熵
        newEntropy = 0.0
        # 计算信息增益
```

```
        for value in uniqueVals:
            # subDataSet划分后的子集
            subDataSet = splitDataSet(dataSet, i, value)
            # 计算子集的概率
            prob = len(subDataSet) / float(len(dataSet))
            # 根据公式计算经验条件熵
            newEntropy += prob * calcShannonEnt(subDataSet)
        # 信息增益
        infoGain = baseEntropy - newEntropy
        # 打印每个特征的信息增益
        print("第%d个特征的增益为%.3f" % (i, infoGain))
        # 计算信息增益
        if(infoGain > bestInfoGain):
            # 更新信息增益，找到最大的信息增益
            bestInfoGain = infoGain
            # 记录信息增益最大的特征的索引值
            bestFeature = i
    # 返回信息增益最大的特征的索引值
    return bestFeature

```

4 必要的工具函数

```
"""
函数说明：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
    values - 需要返回的特征的值

Returns:
    None
"""
def splitDataSet(dataSet, axis, value):
```

```
    # 创建返回的数据集列表
    retDataSet = []
    # 遍历数据集的每一行
    for featVec in dataSet:
        if featVec[axis] == value:
            # 去掉axis特征
            reducedFeatVec = featVec[:axis]
            # 将符合条件的添加到返回的数据集
            # extend() 函数用于在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）。
            reducedFeatVec.extend(featVec[axis+1:])
            # 列表中嵌套列表
            retDataSet.append(reducedFeatVec)
    # 返回划分后的数据集
    return retDataSet

"""
函数说明：统计classList中出现次数最多的元素（类标签）
        服务于递归第两个终止条件

Parameters:
    classList - 类标签列表

Returns:
    sortedClassCount[0][0] - 出现次数最多的元素（类标签）
"""
def majorityCnt(classList):
    classCount = {}
    # 统计classList中每个元素出现的次数
    for vote in classList:
        if vote not in classCount.keys():
            classCount[vote] = 0
        classCount[vote] += 1
    # 根据字典的值降序排序
    # operator.itemgetter(1)获取对象的第1列的值
```

```
47      sortedClassCount = sorted(classCount.items(), key =
    operator.itemgetter(1), reverse = True)
48      # 返回classList中出现次数最多的元素
49      return sortedClassCount[0][0]
```

5 决策树相关函数

```
1  """
2  函数说明：创建决策树（ID3算法）
3          递归有两个终止条件：1、所有的类标签完全相同，直接返回类标签
4                              2、用完所有标签但是得不到唯一类别的分组，即特征不够
   用，挑选出现数量最多的类别作为返回
5
6  Parameters:
7      dataSet - 训练数据集
8      labels - 分类属性标签
9      featLabels - 存储选择的最优特征标签
10
11 Returns:
12     myTree - 决策树
13 """
14 def createTree(dataSet, labels, featLabels):
15     # 取分类标签（是否放贷：yes or no）
16     classList = [example[-1] for example in dataSet]
17     # 如果类别完全相同则停止继续划分
18     if classList.count(classList[0]) == len(classList):
19         return classList[0]
20     # 遍历完所有特征时返回出现次数最多的类标签
21     if len(dataSet[0]) == 1:
22         return majorityCnt(classList)
23     # 选择最优特征
24     bestFeat = chooseBestFeatureToSplit(dataSet)
25     # 最优特征的标签
26     bestFeatLabel = labels[bestFeat]
27     featLabels.append(bestFeatLabel)
```

```
    # 根据最优特征的标签生成树
    myTree = {bestFeatLabel:{}}
    # 删除已经使用的特征标签
    del(labels[bestFeat])
    # 得到训练集中所有最优解特征的属性值
    featValues = [example[bestFeat] for example in dataSet]
    # 去掉重复的属性值
    uniqueVals = set(featValues)
    # 遍历特征，创建决策树
    for value in uniqueVals:
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), labels, featLabels)
    return myTree

"""
函数说明：获取决策树叶子结点的数目

Parameters:
    myTree - 决策树

Returns:
    numLeafs - 决策树的叶子结点的数目
"""
def getNumLeafs(myTree):
    # 初始化叶子
    numLeafs = 0
    # python3中myTree.keys()返回的是dict_keys,不是list,所以不能用
    # myTree.keys()[0]的方法获取结点属性，可以使用list(myTree.keys())[0]
    # next() 返回迭代器的下一个项目 next(iterator[, default])
    firstStr = next(iter(myTree))
    # 获取下一组字典
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        # 测试该结点是否为字典，如果不是字典，代表此节点为叶子结点
```

```
61          if type(secondDict[key]).__name__ == 'dict':
62              numLeafs += getNumLeafs(secondDict[key])
63          else:
64              numLeafs += 1
65      return numLeafs
66  """
67  函数说明：获取决策树的层数
68
69  Parameters:
70      myTree - 决策树
71
72  Returns:
73      maxDepth - 决策树的层数
74  """
75  def getTreeDepth(myTree):
76      # 初始化决策树深度
77      maxDepth = 0
78      # python3中myTree.keys()返回的是dict_keys,不是list,所以不能用
79      # myTree.keys()[0]的方法获取结点属性，可以使用list(myTree.keys())
    [0]
80      # next() 返回迭代器的下一个项目 next(iterator[, default])
81      firstStr = next(iter(myTree))
82      # 获取下一个字典
83      secondDict = myTree[firstStr]
84      for key in secondDict.keys():
85          # 测试该结点是否为字典，如果不是字典，代表此节点为叶子结点
86          if type(secondDict[key]).__name__ == 'dict':
87              thisDepth = 1 + getTreeDepth(secondDict[key])
88          else:
89              thisDepth = 1
90          # 更新最深层数
91          if thisDepth > maxDepth:
92              maxDepth = thisDepth
93      # 返回决策树的层数
94      return maxDepth
95
```

```
"""
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):
    # 获取决策树结点
    firstStr = next(iter(inputTree))
    # 下一个字典
    secondDict = inputTree[firstStr]
    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
```

6 利用pickle包储存及读取决策树

```
"""
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:
    None
"""
def storeTree(inputTree, filename):
    with open(filename, 'wb') as fw:
        pickle.dump(inputTree, fw)


"""
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)

```

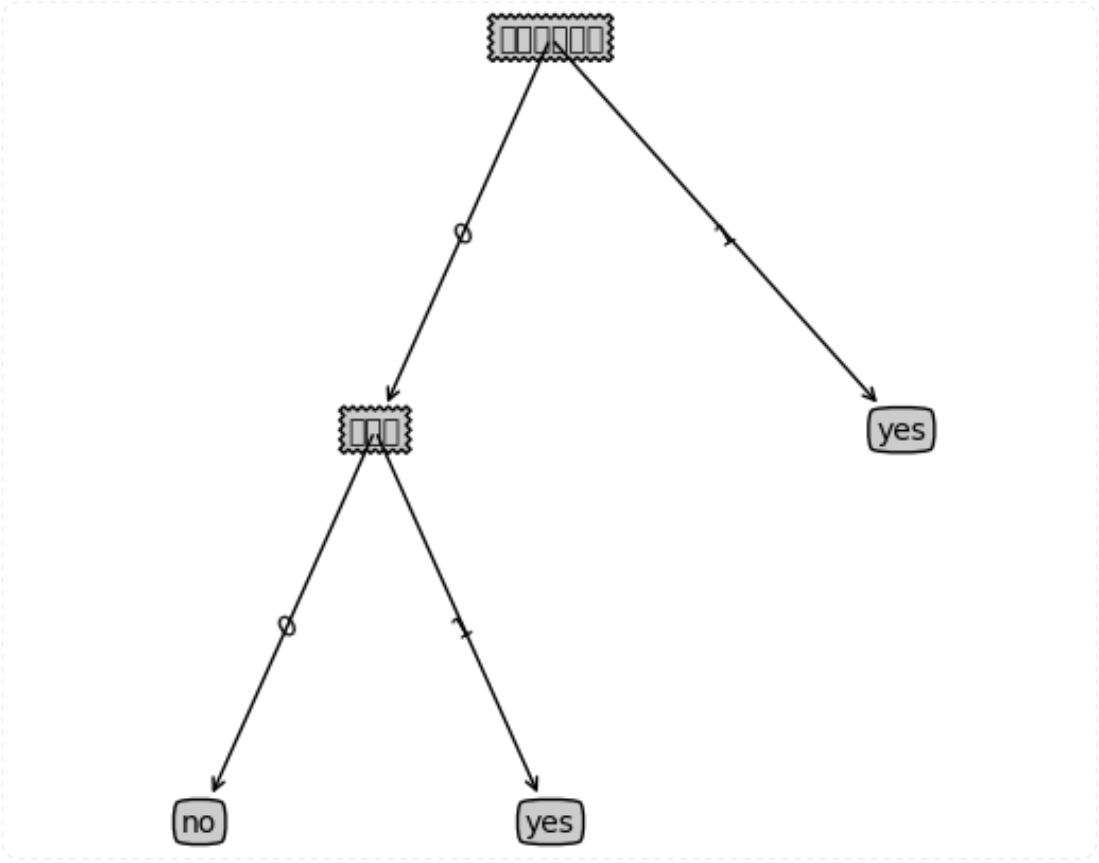
7 主函数

```

dataSet, features = createDataSet()
featLabels = []
myTree = createTree(dataSet, features, featLabels)
# storeTree(myTree, 'classifierStorage.txt')
# myTree = grabTree('classifierStorage.txt')
# print(myTree)
# 测试数据
testVec = [0, 1, 1, 1]
result = classify(myTree, featLabels, testVec)
if result == 'yes':
```

```
    print('放贷')
if result == 'no':
    print('不放贷')
print(myTree)
createPlot(myTree)
# print(dataSet)
# print(calcShannonEnt(dataSet))
print("最优特征索引值:" + str(chooseBestFeatureToSplit(dataSet)))
```

输出如下：

```
第0个特征的增益为0.083
第1个特征的增益为0.324
第2个特征的增益为0.420
第3个特征的增益为0.363
放贷
{'有自己的房子': {0: {'有工作': {0: 'no', 1: 'yes'}}, 1: 'yes'}}

第0个特征的增益为0.083
第1个特征的增益为0.324
第2个特征的增益为0.420
第3个特征的增益为0.363
最优特征索引值:2
```



8 额外内容——SKlearn直接调用

安装SKlearn包指令：

```
pip install scikit-learn
```

数据及处理代码：

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn import tree

dataSet = [[0, 0, 0, 0, 'no'],
           [0, 0, 0, 1, 'no'],
           [0, 1, 0, 1, 'yes'],
           [0, 1, 1, 0, 'yes'],
           [0, 0, 0, 0, 'no'],
           [1, 0, 0, 0, 'no'],
           [1, 0, 0, 1, 'no'],
           [1, 1, 1, 1, 'yes'],
           [1, 0, 1, 2, 'yes'],
```

```
            [1, 0, 1, 2, 'yes'],
            [2, 0, 1, 2, 'yes'],
            [2, 0, 1, 1, 'yes'],
            [2, 1, 0, 1, 'yes'],
            [2, 1, 0, 2, 'yes'],
            [2, 0, 0, 0, 'no']]

# 提取特征和标签
X = [row[:-1] for row in dataSet]  # 特征（去掉最后一列）
y = [row[-1] for row in dataSet]   # 目标标签（最后一列）

# 用 LabelEncoder 把 'no' 和 'yes' 转换为数值
le = LabelEncoder()
y = le.fit_transform(y)  # 'no' -> 0, 'yes' -> 1

clf = tree.DecisionTreeClassifier(max_depth = 4)                #创建
DecisionTreeClassifier()类
clf = clf.fit(X, y)          #使用数据，构建决策树

print(clf.predict([[1,1,1,0]]))      #预测
# 结果：1

```

DecisionTreeClassifier 是 scikit-learn 中的决策树分类器，其构造函数参数如下：

1. 基本参数

参数	说明	默认值
criterion	**选择用于划分的标准**，可选 "gini"（基尼系数）、"entropy"（信息增益）、"log_loss"（对数损失，用于概率估计）	"gini"
splitter	**选择节点划分策略**，"best" 表示选择最优划分，"random" 表示随机选择一个划分	"best"
max_depth	**树的最大深度**，None 表示不限制深度（树会生长到所有叶子纯度为止）	None

2. 控制树的形状

参数	说明	默认值
min_samples_split	**内部节点再划分所需的最小样本数**（整数：至少多少个样本，浮点数：按比例）	2
min_samples_leaf	**叶子节点最少包含的样本数**（防止过拟合）	1
min_weight_fraction_leaf	**叶子节点最小权重比例**（适用于带权重数据，避免小样本极端分割）	0
max_features	**用于寻找最佳划分的特征数量**，可选整数（指定数量）、浮点数（比例）、"auto"（全部）、"sqrt"（平方根个）、"log2"（对数个）	None
max_leaf_nodes	**最大叶子节点数**（None 表示不限制）	None
min_impurity_decrease	**分裂所需的最小不纯度减少量**（防止过拟合）	0

3. 其他参数

参数	说明	默认值
random_state	**随机数种子**，用于保证结果可复现	None
class_weight	**类别权重**，可选 None（均等权重）、"balanced"（自动调整）、字典 {class_label: weight}	None
ccp_alpha	**剪枝参数（复杂度惩罚）**，值越大剪枝越多	0

问题

1. 能否可视化库函数中的决策树?
2. 能否从网上搜索表格数据并用决策树完成分类或回归任务?
3. 自行尝试其他信息增益算法