实验介绍

1.实验内容

本实验学习并实现决策树算法。

2.实验目标

通过本实验掌握决策树算法的基本原理。

3.实验知识点

- 香农熵
- 信息增益

4.实验环境

- python 3.6.5

5.预备知识

- Python编程基础

准备工作

点击屏幕右上方的下载实验数据模块，选择下载decision_tree_glass.tgz到指定目录下，然后再依次选择点击上方的File->Open->Upload,上传刚才下载的数据集压缩包，再使用如下命令解压：

In [1]:
```
!tar -zxvf decision_tree_glass.tgz
```

```
decision_tree_glass/
decision_tree_glass/lenses.txt
decision_tree_glass/classifierStorage.txt
```

决策树构建---ID3算法

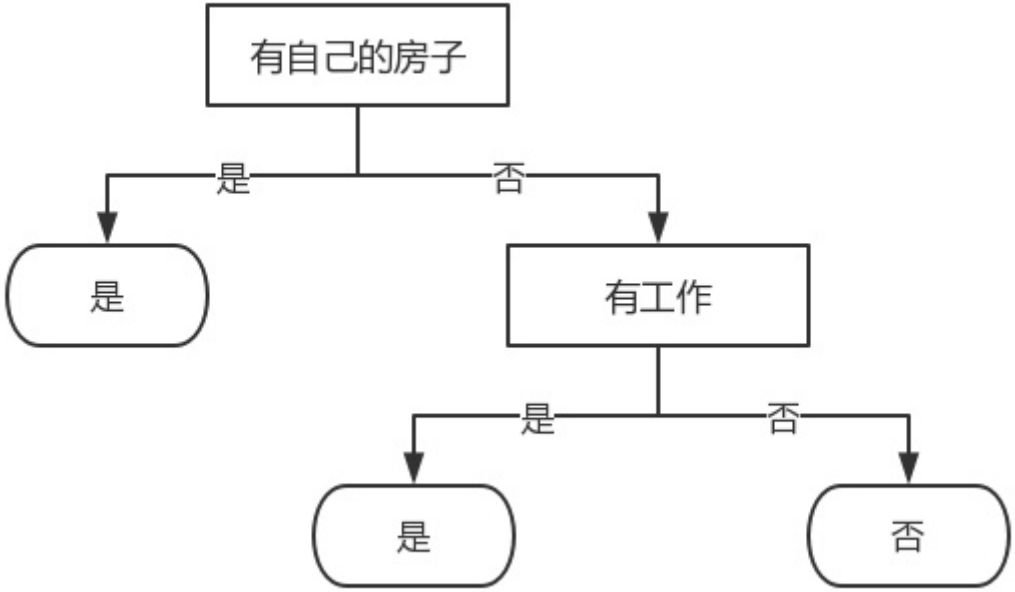
ID3算法的核心是在决策树各个结点上对应信息增益准则选择特征，递归地构建决策树。具体方法是：从根结点(root node)开始，对结点计算所有可能的特征的信息增益，选择信息增益最大的特征作为结点的特征，由该特征的不同取值建立子节点；再对子结点递归地调用以上方法，构建决策树；直到所有特征的信息增益均很小或没有特征可以选择为止。最后得到一个决策树。ID3相当于用极大似然法进行概率模型的选择。

利用决策树实验求得的结果，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：

- $g(D_2, A_1) = H(D_2) - H(D_2|A_1) = 0.251$
- $g(D_2, A_2) = H(D_2) - H(D_2|A_2) = 0.918$
- $g(D_2, A_4) = H(D_2) - H(D_2|A_4) = 0.474$

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"。

这样就生成了一个决策树，该决策树只用了两个特征(有两个内部结点)，生成的决策树如下图所示:



【练习】决策树构建---编写代码构建决策树

我们使用字典存储决策树的结构，比如上小节我们分析出来的决策树，用字典可以表示为：

{'有自己的房子': {0: {'有工作': {0: 'no', 1: 'yes'}}, 1: 'yes'}}

创建函数majorityCnt统计classList中出现此处最多的元素(类标签)，创建函数createTree用来递归构建决策树。编写代码如下：

In [1]:
```python
# -*- coding: UTF-8 -*-
import operator
from math import log


def calcShannonEnt(dataSet):
    """计算给定数据集的经验熵(香农熵)

    Args:
        dataSet: 数据集
    Returns:
        shannonEnt: 经验熵(香农熵)
    """
    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)


def createDataSet():
    """函数说明:创建测试数据集

    Returns:
        dataSet: 数据集
        labels: 特征标签
    """
    dataSet = [[0, 0, 0, 0, 'no'],  #数据集
               [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']]
    labels = ['年龄', '有工作', '有自己的房子', '信贷情况']  #特征标签
    return dataSet, labels  #返回数据集和分类属性


def splitDataSet(dataSet, axis, value):
    """
    函数说明:按照给定特征划分数据集

    Args:
        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
    """
    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  #返回划分后的数据集


def chooseBestFeatureToSplit(dataSet):
    """
    函数说明:选择最优特征

    Args:
        dataSet: 数据集
    Returns:
        bestFeature: 信息增益最大的(最优)特征的索引值
    """
    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


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

    Args:
        classList: 类标签列表
    Returns:
        sortedClassCount[0][0]: 出现此处最多的元素(类标签)
    """
    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


def createTree(dataSet, labels, featLabels):
    """
    函数说明:创建决策树

    Args:
        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
    """
    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)
```

```
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
第0个特征的增益为0.252
第1个特征的增益为0.918
第2个特征的增益为0.474
{'有自己的房子': {0: {'有工作': {0: 'no', 1: 'yes'}}, 1: 'yes'}}
```

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。

【练习】使用决策树进行分类

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结点；最后将测试数据定义为叶子结点所属的类型。在构建决策树的代码，可以看到，有个featLabels参数，它就是用来记录各个分类结点的，在用决策树做预测的时候，我们按顺序输入需要的分类结点的属性值即可。举个例子，比如用上节已经训练好的决策树做分类，那么只需要提供这个人是否有房子，是否有工作这两个信息即可，无需提供冗余的信息。

用决策树做分类的代码很简单，编写代码如下：

In [2]:
```python
# -*- coding: UTF-8 -*-

def classify(inputTree, featLabels, testVec):
    """
    函数说明:使用决策树分类

    Args:
        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
    """
    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')
```

```
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
第0个特征的增益为0.252
第1个特征的增益为0.918
第2个特征的增益为0.474
放贷
```

这里只增加了classify函数，用于决策树分类。输入测试数据[0,1]，它代表没有房子，但是有工作。

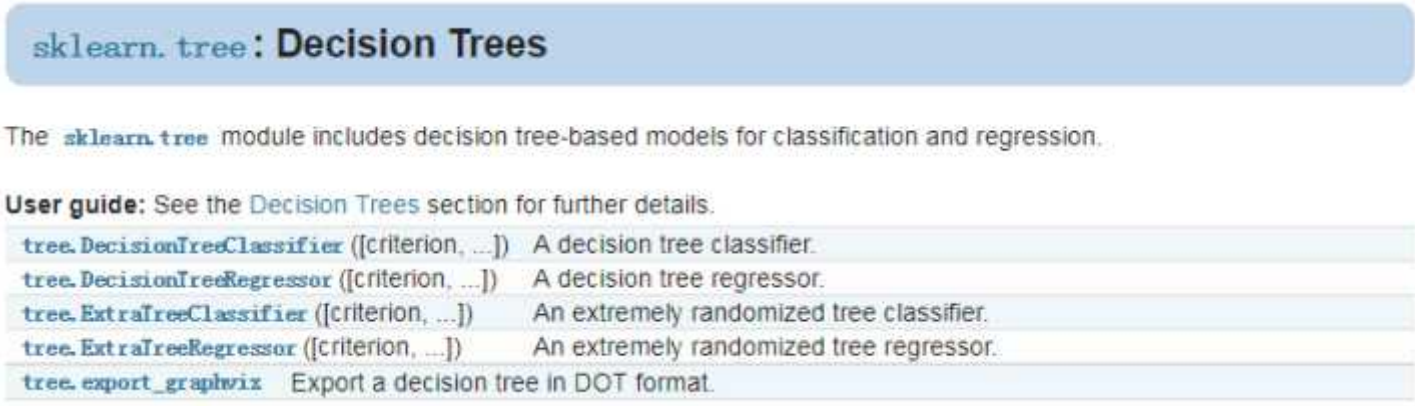
【练习】基于决策树预测隐形眼睛类型---使用Sklearn

一旦理解了决策树的工作原理，我们就可以帮助人们判断需要佩戴的镜片类型。隐形眼镜数据集是非常著名的数据集，它包含很多换着眼部状态的观察条件以及医生推荐的隐形眼镜类型。隐形眼镜类型包括硬材质(hard)、软材质(soft)以及不适合佩戴隐形眼镜(no lenses)。

数据集一共有24组数据，数据的Labels依次是age、prescript、astigmatic、tearRate、class，也就是第一列是年龄，第二列是症状，第三列是是否散光，第四列是眼泪数量，第五列是最终的分类标签。数据如下图所示：

```
young   myope   no  reduced no lenses
young   myope   no  normal  soft
young   myope   yes reduced no lenses
young   myope   yes normal  hard
young   hyper   no  reduced no lenses
young   hyper   no  normal  soft
young   hyper   yes reduced no lenses
young   hyper   yes normal  hard
pre myope   no  reduced no lenses
pre myope   no  normal  soft
pre myope   yes reduced no lenses
pre myope   yes normal  hard
pre hyper   no  reduced no lenses
pre hyper   no  normal  soft
pre hyper   yes reduced no lenses
pre hyper   yes normal  no lenses
presbyopic  myope   no  reduced no lenses
presbyopic  myope   no  normal  no lenses
presbyopic  myope   yes reduced no lenses
presbyopic  myope   yes normal  hard
presbyopic  hyper   no  reduced no lenses
presbyopic  hyper   no  normal  soft
presbyopic  hyper   yes reduced no lenses
presbyopic  hyper   yes normal  no lenses
```

接下来我们来说如何使用Sklearn构建决策树。sklearn.tree模块提供了决策树模型，用于解决分类问题和回归问题。方法如下图所示：



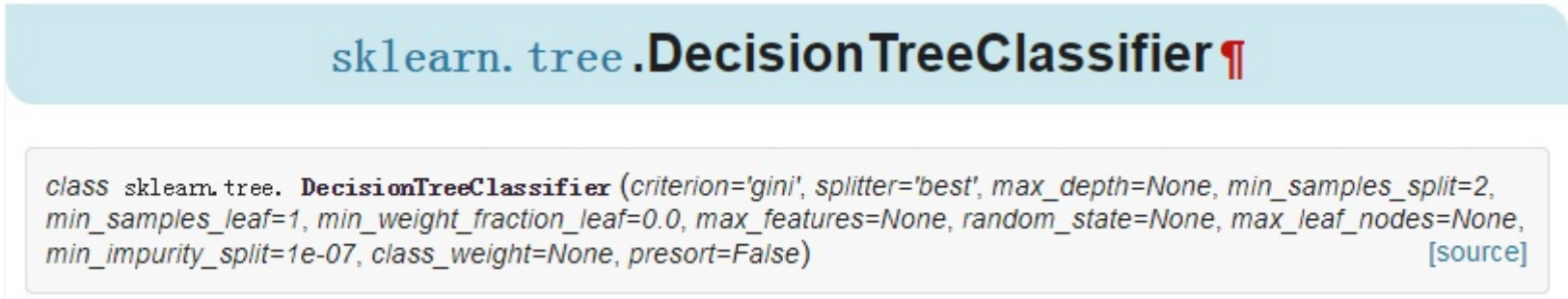
sklearn.tree: Decision Trees

The sklearn.tree module includes decision tree-based models for classification and regression.

User guide: See the Decision Trees section for further details.

tree.DecisionTreeClassifier([criterion, ...])	A decision tree classifier.
tree.DecisionTreeRegressor([criterion, ...])	A decision tree regressor.
tree.ExtraTreeClassifier([criterion, ...])	An extremely randomized tree classifier.
tree.ExtraTreeRegressor([criterion, ...])	An extremely randomized tree regressor.
tree.export_graphviz	Export a decision tree in DOT format.

我们使用DecisionTreeClassifie构建决策树，这个函数共有12个参数：



sklearn.tree.DecisionTreeClassifier¶

class sklearn.tree. **DecisionTreeClassifier** (*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_split=1e-07, class_weight=None, presort=False*) [source]

参数说明如下：

criterion：特征选择标准，可选参数，默认是gini，可以设置为entropy。gini是基尼不纯度，是将来自集合的某种结果随机应用于某一数据项的预期误差率。entropy是香农熵。

splitter：特征划分点选择标准，可选参数，默认是best，可以设置为random。每个结点的选择策略。best参数是根据算法选择最佳的切分特征，例如gini、entropy。random随机的在部分划分点中找局部最优的划分点。默认的"best"适合样本量不大的时候，而如果样本数据量非常大，此时决策树构建推荐"random"。

max_features：划分时考虑的最大特征数，可选参数，默认是None。寻找最佳切分时考虑的最大特征数(n_features为总共的特征数)，有如下6种情况：
如果max_features是整型的数，则考虑max_features个特征；
如果max_features是浮点型的数，则考虑int(max_features * n_features)个特征；
如果max_features设为auto，那么max_features = sqrt(n_features)；
如果max_features设为sqrt，那么max_featrues = sqrt(n_features)，跟auto一样；
如果max_features设为log2，那么max_features = log2(n_features)；
如果max_features设为None，那么max_features = n_features，也就是所有特征都用。 一般来说，如果样本特征数不多，比如小于50，我们用默认的"None"就可以了，如果特征数非常多，我们可以灵活使用刚才描述的其他取值来控制划分时考虑的最大特征数，以控制决策树的生成时间。

max_depth：决策树最大深，可选参数，默认是None。这个参数是这是树的层数的。层数的概念就是，比如在贷款的例子中，决策树的层数是2层。如果这个参数设置为None，那么决策树在建立子树的时候不会限制子树的深度。一般来说，数据少或者特征少的时候可以不管这个值。或者如果设置了min_samples_slipt参数，那么直到少于min_smaples_split个样本为止。如果模型样本量多，特征也多的情况下，推荐限制这个最大深度，具体的取值取决于数据的分布。常用的可以取值10-100之间。

min_samples_split：内部节点再划分所需最小样本数，可选参数，默认是2。这个值限制了子树继续划分的条件。如果min_samples_split为整数，那么在切分内部结点的时候，min_samples_split作为最小的样本数，也就是说，如果样本已经少于min_samples_split个样本，则停止继续切分。如果min_samples_split为浮

点数，那么min_samples_split就是一个百分比，ceil(min_samples_split * n_samples)，数是向上取整的。如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值。

min_samples_leaf：叶子节点最少样本数，可选参数，默认是1。这个值限制了叶子节点最少的样本数，如果某叶子节点数目小于样本数，则会和兄弟节点一起被剪枝。叶结点需要最少的样本数，也就是最后到叶结点，需要多少个样本才能算一个叶结点。如果设置为1，哪怕这个类别只有1个样本，决策树也会构建出来。如果min_samples_leaf是整数，那么min_samples_leaf作为最小的样本数。如果是浮点数，那么min_samples_leaf就是一个百分比，同上，celi(min_samples_leaf * n_samples)，数是向上取整的。如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值

class_weight：类别权重，可选参数，默认是None，也可以字典、字典列表、balanced。指定样本各类别的的权重，主要是为了防止训练集某些类别的样本过多，导致训练的决策树过于偏向这些类别。类别的权重可以通过{class_label：weight}这样的格式给出，这里可以自己指定各个样本的权重，或者用balanced，如果使用balanced，则算法会自己计算权重，样本量少的类别所对应的样本权重会高。当然，如果你的样本类别分布没有明显的偏倚，则可以不管这个参数，选择默认的None。

random_state：可选参数，默认是None。随机数种子。如果是证书，那么random_state会作为随机数生成器的随机数种子。随机数种子，如果没有设置随机数，随机出来的数与当前系统时间有关，每个时刻都是不同的。如果设置了随机数种子，那么相同随机数种子，不同时刻产生的随机数也是相同的。如果是RandomState instance，那么random_state是随机数生成器。如果为None，则随机数生成器使用np.random。

min_impurity_split：节点划分最小不纯度,可选参数，默认是1e-7。这是个阈值，这个值限制了决策树的增长，如果某节点的不纯度(基尼系数，信息增益，均方差，绝对差)小于这个阈值，则该节点不再生成子节点。即为叶子节点 。

min_weight_fraction_leaf：叶子节点最小的样本权重和，可选参数，默认是0。这个值限制了叶子节点所有样本权重和的最小值，如果小于这个值，则会和兄弟节点一起被剪枝。一般来说，如果我们有较多样本有缺失值，或者分类树样本的分布类别偏差很大，就会引入样本权重，这时我们就要注意这个值了。

max_leaf_nodes：最大叶子节点数，可选参数，默认是None。通过限制最大叶子节点数，可以防止过拟合。如果加了限制，算法会建立在最大叶子节点数内最优的决策树。如果特征不多，可以不考虑这个值，但是如果特征分成多的话，可以加以限制，具体的值可以通过交叉验证得到。

presort：数据是否预排序，可选参数，默认为False，这个值是布尔值，默认是False不排序。一般来说，如果样本量少或者限制了一个深度很小的决策树，设置为true可以让划分点选择更加快，决策树建立的更加快。

除了这些参数要注意以外，其他在调参时的注意点有：

1）当样本数量少但是样本特征非常多的时候，决策树很容易过拟合，一般来说，样本数比特征数多一些会比较容易建立健壮的模型。

2）如果样本数量少但是样本特征非常多，在拟合决策树模型前，推荐先做维度规约，比如主成分分析（PCA），特征选择（Losso）或者独立成分分析（ICA）。这样特征的维度会大大减小。再来拟合决策树模型效果会好。

3）推荐多用决策树的可视化，同时先限制决策树的深度，这样可以先观察下生成的决策树里数据的初步拟合情况，然后再决定是否要增加深度。

4）在训练模型时，注意观察样本的类别情况（主要指分类树），如果类别分布非常不均匀，就要考虑用class_weight来限制模型过于偏向样本多的类别。

5）决策树的数组使用的是numpy的float32类型，如果训练数据不是这样的格式，算法会先做copy再运行。

6）如果输入的样本矩阵是稀疏的，推荐在拟合前调用csc_matrix稀疏化，在预测前调用csr_matrix稀疏化。

sklearn.tree.DecisionTreeClassifier()提供了一些方法供我们使用，如下图所示：

Methods

apply (X[, check_input])	Returns the index of the leaf that each sample is predicted as.
decision_path (X[, check_input])	Return the decision path in the tree
fit (X, y[, sample_weight, check_input, ...])	Build a decision tree classifier from the training set (X, y).
fit_transform (X[, y])	Fit to data, then transform it.
get_params ([deep])	Get parameters for this estimator.
predict (X[, check_input])	Predict class or regression value for X.
predict_log_proba (X)	Predict class log-probabilities of the input samples X.
predict_proba (X[, check_input])	Predict class probabilities of the input samples X.
score (X, y[, sample_weight])	Returns the mean accuracy on the given test data and labels.
set_params (**params)	Set the parameters of this estimator.
transform (*args, **kwargs)	DEPRECATED: Support to use estimators as feature selectors will be removed in version 0.19.

【练习】基于决策树预测隐形眼睛类型---编写代码

In [3]:
```
# -*- coding: UTF-8 -*-
from sklearn import tree

if __name__ == '__main__':
    fr = open('decision_tree_glass/lenses.txt')
    lenses = [inst.strip().split('\t') for inst in fr.readlines()]
    print(lenses)
    lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']
    clf = tree.DecisionTreeClassifier()
    lenses = clf.fit(lenses, lensesLabels)
```

```
[['young', 'myope', 'no', 'reduced', 'no lenses'], ['young', 'myope', 'no', 'normal', 'soft'], ['young', 'myope', 'yes', 'reduced', 'no lense
s'], ['young', 'myope', 'yes', 'normal', 'hard'], ['young', 'hyper', 'no', 'reduced', 'no lenses'], ['young', 'hyper', 'no', 'normal', 'sof
t'], ['young', 'hyper', 'yes', 'reduced', 'no lenses'], ['young', 'hyper', 'yes', 'normal', 'hard'], ['pre', 'myope', 'no', 'reduced', 'no len
ses'], ['pre', 'myope', 'no', 'normal', 'soft'], ['pre', 'myope', 'yes', 'reduced', 'no lenses'], ['pre', 'myope', 'yes', 'normal', 'hard'],
['pre', 'hyper', 'no', 'reduced', 'no lenses'], ['pre', 'hyper', 'no', 'normal', 'soft'], ['pre', 'hyper', 'yes', 'reduced', 'no lenses'], ['p
re', 'hyper', 'yes', 'normal', 'no lenses'], ['presbyopic', 'myope', 'no', 'reduced', 'no lenses'], ['presbyopic', 'myope', 'no', 'normal', 'n
o lenses'], ['presbyopic', 'myope', 'yes', 'reduced', 'no lenses'], ['presbyopic', 'myope', 'yes', 'normal', 'hard'], ['presbyopic', 'hyper',
'no', 'reduced', 'no lenses'], ['presbyopic', 'hyper', 'no', 'normal', 'soft'], ['presbyopic', 'hyper', 'yes', 'reduced', 'no lenses'], ['pres
byopic', 'hyper', 'yes', 'normal', 'no lenses']]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [3], line 10
      8 lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']
      9 clf = tree.DecisionTreeClassifier()
---> 10 lenses = clf.fit(lenses, lensesLabels)

File D:\Program Files\Miniconda\envs\pytorch\lib\site-packages\sklearn\tree\_classes.py:969, in DecisionTreeClassifier.fit(self, X, y, sample_
weight, check_input)
    939 def fit(self, X, y, sample_weight=None, check_input=True):
    940     """Build a decision tree classifier from the training set (X, y).
    941
    942     Parameters
   (...)
    966         Fitted estimator.
    967     """
--> 969     super().fit(
    970         X,
    971         y,
    972         sample_weight=sample_weight,
    973         check_input=check_input,
    974     )
    975     return self
```

```
File D:\Program Files\Miniconda\envs\pytorch\lib\site-packages\sklearn\tree\_classes.py:172, in BaseDecisionTree.fit(self, X, y, sample_weight, check_input)
    170 check_X_params = dict(dtype=DTYPE, accept_sparse="csc")
    171 check_y_params = dict(ensure_2d=False, dtype=None)
--> 172 X, y = self._validate_data(
    173     X, y, validate_separately=(check_X_params, check_y_params)
    174 )
    175 if issparse(X):
    176     X.sort_indices()

File D:\Program Files\Miniconda\envs\pytorch\lib\site-packages\sklearn\base.py:591, in BaseEstimator._validate_data(self, X, y, reset, validate_separately, **check_params)
    589 if "estimator" not in check_X_params:
    590     check_X_params = {**default_check_params, **check_X_params}
--> 591 X = check_array(X, input_name="X", **check_X_params)
    592 if "estimator" not in check_y_params:
    593     check_y_params = {**default_check_params, **check_y_params}

File D:\Program Files\Miniconda\envs\pytorch\lib\site-packages\sklearn\utils\validation.py:856, in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator, input_name)
    854         array = array.astype(dtype, casting="unsafe", copy=False)
    855     else:
--> 856         array = np.asarray(array, order=order, dtype=dtype)
    857 except ComplexWarning as complex_warning:
    858     raise ValueError(
    859         "Complex data not supported\n{}\n".format(array)
    860     ) from complex_warning

ValueError: could not convert string to float: 'young'
```

我们可以看到程序报错了，这是为什么？因为在fit()函数不能接收string类型的数据，通过打印的信息可以看到，数据都是string类型的。在使用fit()函数之前，我们需要对数据集进行编码，这里可以使用两种方法：

LabelEncoder：将字符串转换为增量值

OneHotEncoder：使用One-of-K算法将字符串转换为整数

为了对string类型的数据序列化，需要先生成pandas数据，这样方便我们的序列化工作。这里使用的方法是：原始数据->字典->pandas数据，编写代码如下：

In [4]:
```
# -*- coding: UTF-8 -*-
import pandas as pd

if __name__ == '__main__':
    with open('decision_tree_glass/lenses.txt', 'r') as fr:  #加载文件
        lenses = [inst.strip().split('\t') for inst in fr.readlines()]  #处理文件
    lenses_target = []  #提取每组数据的类别，保存在列表里
    for each in lenses:
        lenses_target.append(each[-1])
    lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']  #特征标签
    lenses_list = []  #保存lenses数据的临时列表
    lenses_dict = {}  #保存lenses数据的字典，用于生成pandas
    for each_label in lensesLabels:  #提取信息，生成字典
        for each in lenses:
            lenses_list.append(each[lensesLabels.index(each_label)])
        lenses_dict[each_label] = lenses_list
        lenses_list = []
    print(lenses_dict)  #打印字典信息
    lenses_pd = pd.DataFrame(lenses_dict)  #生成pandas.DataFrame
    print(lenses_pd)
```

```
{'age': ['young', 'young', 'young', 'young', 'young', 'young', 'young', 'young', 'pre', 'pre', 'pre', 'pre', 'pre', 'pre', 'pre', 'pre',
'presbyopic', 'presbyopic', 'presbyopic', 'presbyopic', 'presbyopic', 'presbyopic', 'presbyopic', 'presbyopic'], 'prescript': ['myope', 'm
yope', 'myope', 'myope', 'hyper', 'hyper', 'hyper', 'hyper', 'myope', 'myope', 'myope', 'myope', 'hyper', 'hyper', 'hyper', 'hyper', 'myop
e', 'myope', 'myope', 'myope', 'hyper', 'hyper', 'hyper', 'hyper'], 'astigmatic': ['no', 'no', 'yes', 'yes', 'no', 'no', 'yes', 'yes', 'n
o', 'no', 'yes', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes', 'yes'], 'tearRate': ['reduced', 'normal',
'reduced', 'normal', 'reduced', 'normal', 'reduced', 'normal', 'reduced', 'normal', 'reduced', 'normal', 'reduced', 'normal', 'reduced',
'normal', 'reduced', 'normal', 'reduced', 'normal', 'reduced', 'normal', 'reduced', 'normal']}
          age prescript astigmatic tearRate
0       young     myope         no  reduced
1       young     myope         no   normal
2       young     myope        yes  reduced
3       young     myope        yes   normal
4       young     hyper         no  reduced
5       young     hyper         no   normal
6       young     hyper        yes  reduced
7       young     hyper        yes   normal
8         pre     myope         no  reduced
9         pre     myope         no   normal
10        pre     myope        yes  reduced
11        pre     myope        yes   normal
12        pre     hyper         no  reduced
13        pre     hyper         no   normal
14        pre     hyper        yes  reduced
15        pre     hyper        yes   normal
```

```
16  presbyopic     myope         no  reduced
17  presbyopic     myope         no   normal
18  presbyopic     myope        yes  reduced
19  presbyopic     myope        yes   normal
20  presbyopic     hyper         no  reduced
21  presbyopic     hyper         no   normal
22  presbyopic     hyper        yes  reduced
23  presbyopic     hyper        yes   normal
```

从运行结果可以看出，顺利生成pandas数据。
接下来，将数据序列化，编写代码如下：

In [5]:
```
# !pip install pydotplus
# -*- coding: UTF-8 -*-
import pandas as pd
from sklearn.preprocessing import LabelEncoder

if __name__ == '__main__':
    with open('decision_tree_glass/lenses.txt', 'r') as fr:  #加载文件
        lenses = [inst.strip().split('\t') for inst in fr.readlines()]  #处理文件
    lenses_target = []  #提取每组数据的类别，保存在列表里
    for each in lenses:
        lenses_target.append(each[-1])
    lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']  #特征标签
    lenses_list = []  #保存lenses数据的临时列表
    lenses_dict = {}  #保存lenses数据的字典，用于生成pandas
    for each_label in lensesLabels:  #提取信息，生成字典
        for each in lenses:
            lenses_list.append(each[lensesLabels.index(each_label)])
        lenses_dict[each_label] = lenses_list
        lenses_list = []
    # print(lenses_dict)                                                    #打印字典信息
    lenses_pd = pd.DataFrame(lenses_dict)  #生成pandas.DataFrame
    print(lenses_pd)  #打印pandas.DataFrame
    le = LabelEncoder()  #创建LabelEncoder()对象，用于序列化
    for col in lenses_pd.columns:  #为每一列序列化
        lenses_pd[col] = le.fit_transform(lenses_pd[col])
    print(lenses_pd)
```

```
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple (https://pypi.tuna.tsinghua.edu.cn/simple)
Collecting pydotplus
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/60/bf/62567830b700d9f6930e9ab6831d6ba256f7b0b730acb37278b0ccdffacf/pydotplus-2.0.2.tar.gz (https://pypi.tuna.tsinghua.edu.cn/packages/60/bf/62567830b700d9f6930e9ab6831d6ba256f7b0b730acb37278b0ccdffacf/pydotplus-2.0.2.tar.gz) (278 kB)
     ---------------------------------------- 278.7/278.7 kB 612.9 kB/s eta 0:00:00
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: pyparsing>=2.0.1 in d:\program files\miniconda\envs\pytorch\lib\site-packages (from pydotplus) (3.0.9)
Building wheels for collected packages: pydotplus
  Building wheel for pydotplus (setup.py): started
  Building wheel for pydotplus (setup.py): finished with status 'done'
  Created wheel for pydotplus: filename=pydotplus-2.0.2-py3-none-any.whl size=24554 sha256=d4d52459bb230c21f472127a65a9eb12a104ecff005d585d0dcbc5edb9fd98ce
  Stored in directory: c:\users\admin\appdata\local\pip\cache\wheels\84\9b\a3\9a985102a9390a960eeb4047a1e048a9e8527b5fdf4b9e02b4
Successfully built pydotplus
Installing collected packages: pydotplus
Successfully installed pydotplus-2.0.2
```

```
          age prescript astigmatic tearRate
0       young     myope         no  reduced
1       young     myope         no   normal
2       young     myope        yes  reduced
3       young     myope        yes   normal
4       young     hyper         no  reduced
5       young     hyper         no   normal
6       young     hyper        yes  reduced
7       young     hyper        yes   normal
8         pre     myope         no  reduced
9         pre     myope         no   normal
10        pre     myope        yes  reduced
11        pre     myope        yes   normal
12        pre     hyper         no  reduced
13        pre     hyper         no   normal
14        pre     hyper        yes  reduced
15        pre     hyper        yes   normal
16  presbyopic     myope         no  reduced
17  presbyopic     myope         no   normal
18  presbyopic     myope        yes  reduced
19  presbyopic     myope        yes   normal
20  presbyopic     hyper         no  reduced
21  presbyopic     hyper         no   normal
22  presbyopic     hyper        yes  reduced
23  presbyopic     hyper        yes   normal
    age  prescript  astigmatic  tearRate
0     2          1           0         1
1     2          1           0         0
2     2          1           1         1
3     2          1           1         0
4     2          0           0         1
5     2          0           0         0
6     2          0           1         1
7     2          0           1         0
8     0          1           0         1
9     0          1           0         0
10    0          1           1         1
11    0          1           1         0
12    0          0           0         1
13    0          0           0         0
14    0          0           1         1
15    0          0           1         0
16    1          1           0         1
17    1          1           0         0
18    1          1           1         1
19    1          1           1         0
```

```
20      1               0               0               1
21      1               0               0               0
22      1               0               1               1
23      1               0               1               0
```

从打印结果可以看到，我们已经将数据顺利序列化，接下来。我们就可以fit()数据，构建决策树了。

【练习】基于决策树预测隐形眼睛类型---预测

确定好决策树之后，我们就可以做预测了。可以根据自己的眼睛情况和年龄等特征，看一看自己适合何种材质的隐形眼镜。使用如下代码就可以看到预测结果：
print(clf.predict([[1,1,1,0]]))
完整代码如下：

In [12]:
```
# -*- coding: UTF-8 -*-
from sklearn.preprocessing import LabelEncoder
from six import StringIO
from sklearn import tree
import pandas as pd
import pydotplus

if __name__ == '__main__':
    with open('decision_tree_glass/lenses.txt', 'r') as fr:  #加载文件
        lenses = [inst.strip().split('\t') for inst in fr.readlines()]  #处理文件
    lenses_target = []  #提取每组数据的类别，保存在列表里
    for each in lenses:
        lenses_target.append(each[-1])
    print(lenses_target)

    lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']  #特征标签
    lenses_list = []  #保存lenses数据的临时列表
    lenses_dict = {}  #保存lenses数据的字典，用于生成pandas
    for each_label in lensesLabels:  #提取信息，生成字典
        for each in lenses:
            lenses_list.append(each[lensesLabels.index(each_label)])
        lenses_dict[each_label] = lenses_list
        lenses_list = []
    # print(lenses_dict)                                                    #打印字典信息
    lenses_pd = pd.DataFrame(lenses_dict)  #生成pandas.DataFrame
    # print(lenses_pd)                                                      #打印pandas.DataFrame
    le = LabelEncoder()  #创建LabelEncoder()对象，用于序列化
    for col in lenses_pd.columns:  #序列化
        lenses_pd[col] = le.fit_transform(lenses_pd[col])
    # print(lenses_pd)                                                      #打印编码信息
    clf = tree.DecisionTreeClassifier(max_depth=4)  #创建DecisionTreeClassifier()类
    clf = clf.fit(lenses_pd.values.tolist(), lenses_target)  #使用数据，构建决策树
    dot_data = StringIO()
    tree.export_graphviz(clf, out_file=dot_data,  #绘制决策树
                         feature_names=lenses_pd.keys(),
                         class_names=clf.classes_,
                         filled=True, rounded=True,
                         special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    print(clf.predict([[1, 1, 1, 0]]))  #保存绘制好的决策树，以PDF的形式存储。
```

```
['no lenses', 'soft', 'no lenses', 'hard', 'no lenses', 'soft', 'no lenses', 'hard', 'no lenses', 'soft', 'no lenses', 'hard', 'no lenses', 's
oft', 'no lenses', 'no lenses', 'no lenses', 'no lenses', 'no lenses', 'hard', 'no lenses', 'soft', 'no lenses', 'no lenses']
['hard']
```

实验总结

通过本实验掌握决策树算法的构建分类工作并实现基于决策树的隐形眼镜预测。