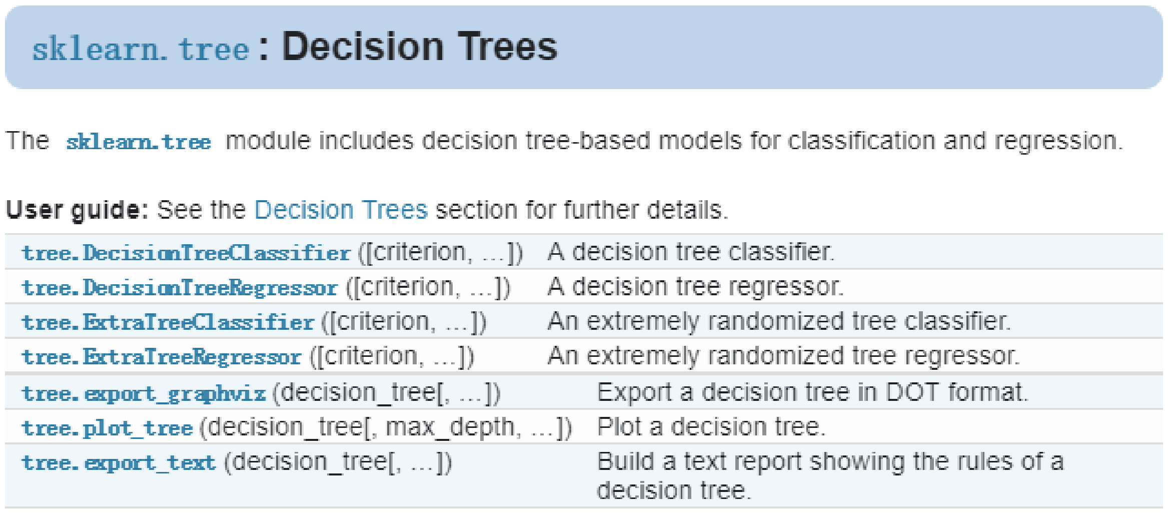
sklearn.tree模块提供了决策树模型, 用于解决分类合回归问题. 具体包括如下几个大块:



sklearn.tree: Decision Trees

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

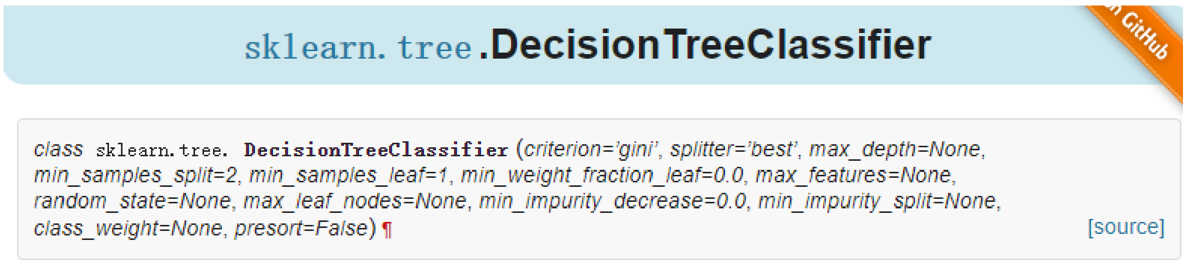
User guide: See the Decision Trees section for further details.

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 (decision_tree[, …])	Export a decision tree in DOT format.
tree.plot_tree (decision_tree[, max_depth, …])	Plot a decision tree.
tree.export_text (decision_tree[, …])	Build a text report showing the rules of a decision tree.

这里主要介绍的是决策树分类, 用到是tree.DecisionTreeClassifier, 同时为了能够使得决策树可视化, 还会用到tree.export_graphviz模块

一、tree.DecisionTreeClassifier

tree.DecisionTreeClassifie这个函数中, 共有12个参数, 见下图:



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_decrease=0.0, min_impurity_split=None,
class_weight=None, presort=False) ¶                                                    [source]
```

criterion : *string, optional (default=”gini”)*

The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.

splitter : *string, optional (default=”best”)*

The strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split.

max_depth : *int or None, optional (default=None)*

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : *int, float, optional (default=2)*

The minimum number of samples required to split an internal node:

- If int, then consider `min_samples_split` as the minimum number.
- If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Changed in version 0.18: Added float values for fractions.

min_samples_leaf : *int, float, optional (default=1)*

The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least `min_samples_leaf` training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

- If int, then consider `min_samples_leaf` as the minimum number.
- If float, then `min_samples_leaf` is a fraction and `ceil(min_samples_leaf * n_samples)` are the minimum number of samples for each node.

Changed in version 0.18: Added float values for fractions.

min_weight_fraction_leaf : *float, optional (default=0.)*

The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

max_features : *int, float, string or None, optional (default=None)*

The number of features to consider when looking for the best split:

- If int, then consider `max_features` features at each split.
- If float, then `max_features` is a fraction and `int(max_features * n_features)` features are considered at each split.
- If “auto”, then `max_features=sqrt(n_features)`.
- If “sqrt”, then `max_features=sqrt(n_features)`.
- If “log2”, then `max_features=log2(n_features)`.
- If None, then `max_features=n_features`.

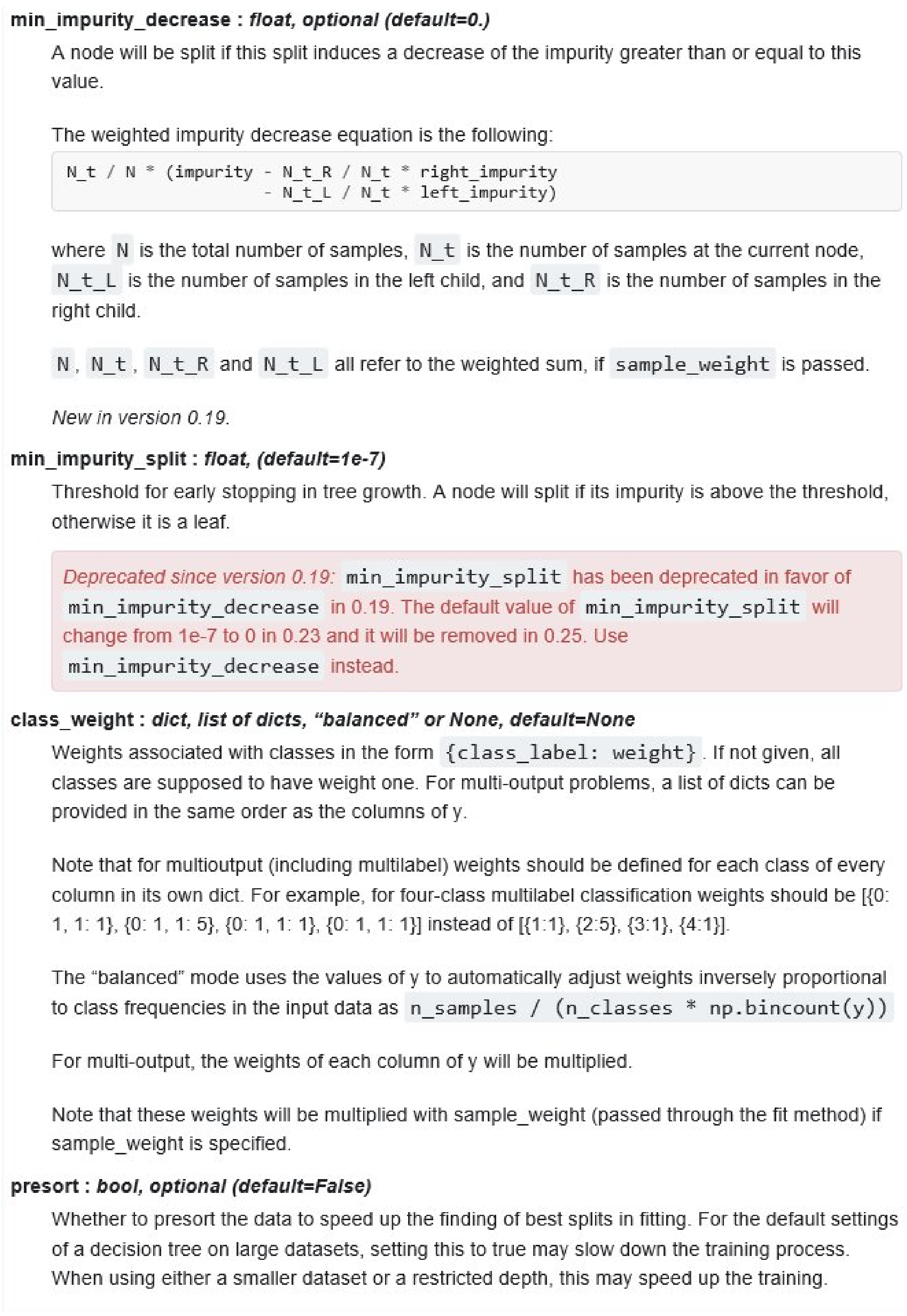
Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than `max_features` features.

random_state : *int, RandomState instance or None, optional (default=None)*

If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by `np.random`.

max_leaf_nodes : *int or None, optional (default=None)*

Grow a tree with `max_leaf_nodes` in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.



min_impurity_decrease : *float, optional (default=0.)*

A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

The weighted impurity decrease equation is the following:

```
N_t / N * (impurity - N_t_R / N_t * right_impurity
                    - N_t_L / N_t * left_impurity)
```

where `N` is the total number of samples, `N_t` is the number of samples at the current node, `N_t_L` is the number of samples in the left child, and `N_t_R` is the number of samples in the right child.

`N`, `N_t`, `N_t_R` and `N_t_L` all refer to the weighted sum, if `sample_weight` is passed.

New in version 0.19.

min_impurity_split : *float, (default=1e-7)*

Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.

Deprecated since version 0.19: `min_impurity_split` has been deprecated in favor of `min_impurity_decrease` in 0.19. The default value of `min_impurity_split` will change from 1e-7 to 0 in 0.23 and it will be removed in 0.25. Use `min_impurity_decrease` instead.

class_weight : *dict, list of dicts, “balanced” or None, default=None*

Weights associated with classes in the form `{class_label: weight}`. If not given, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.

Note that for multioutput (including multilabel) weights should be defined for each class of every column in its own dict. For example, for four-class multilabel classification weights should be [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] instead of [{1:1}, {2:5}, {3:1}, {4:1}].

The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`

For multi-output, the weights of each column of y will be multiplied.

Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified.

presort : *bool, optional (default=False)*

Whether to presort the data to speed up the finding of best splits in fitting. For the default settings of a decision tree on large datasets, setting this to true may slow down the training process. When using either a smaller dataset or a restricted depth, this may speed up the training.

每一个参数的具体含义如下:

- **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, 是一种基于统计的思想. `entropy`是香农熵, 也就是上篇文章讲过的内容, 是一种基于信息论的思想. Sklearn把`gini`设为默认参数, 应该也是做了相应的斟酌的, 精度也许更高些? **ID3算法使用的是**`entropy`**, CART算法使用的则是**`gini`.

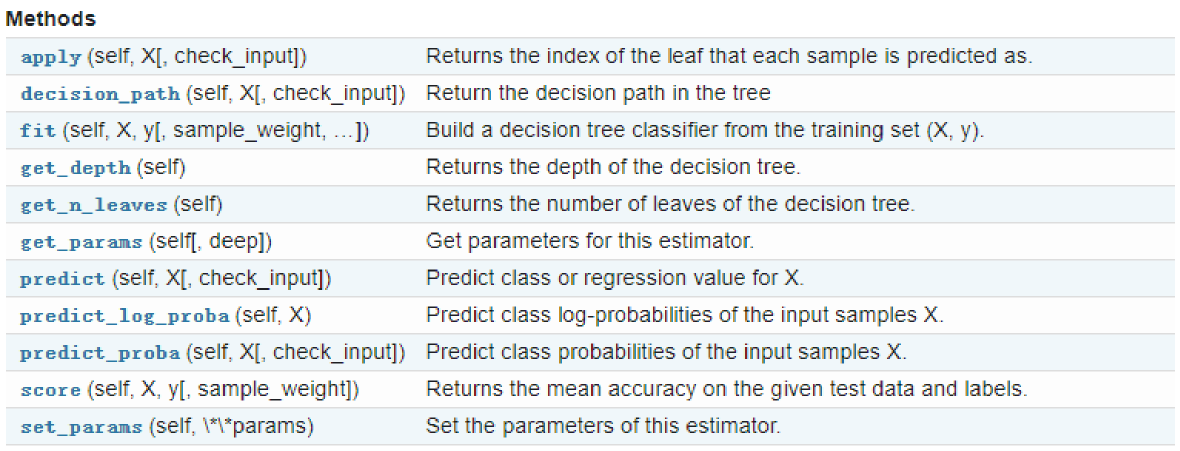
- **splitter: 特征划分点选择标准**, 可选参数, **默认是**`best`, 可以设置为`random`. 每个结点的选择策略. `best`参数是根据算法选择最佳的切分特征, 例如`gini`、`entropy`. `random`随机的在部分划分点中找局部最优的划分点. 默认的"best"适合样本量不大的时候, 而如果样本数据量非常大, 此时决策树构建推荐random.
- **max_features**: **划分时考虑的最大特征数**, 可选参数, 默认是None. 寻找最佳切分时考虑的最大特征数(n_features为总共的特征数), 有如下6种情况:
    - 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: 决策树最大深度**, 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.
- **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.
- **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： 随机数种子**, 可选参数, 默认是`None`. . 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 .
- **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. 如果样本量太大的话, 反而没有什么好处. 问题是样本量少的时候, 我速度本来就不慢. 所以这个值一般懒得理它就可以了.

除了这些参数要注意以外, 其他在调参时的**注意**点有：

- 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
- 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.
- 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.
- 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.
- 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.
- 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.

同时, sklearn.tree.DecisionTreeClassifier()提供了一些方法, 具体如下:



Methods

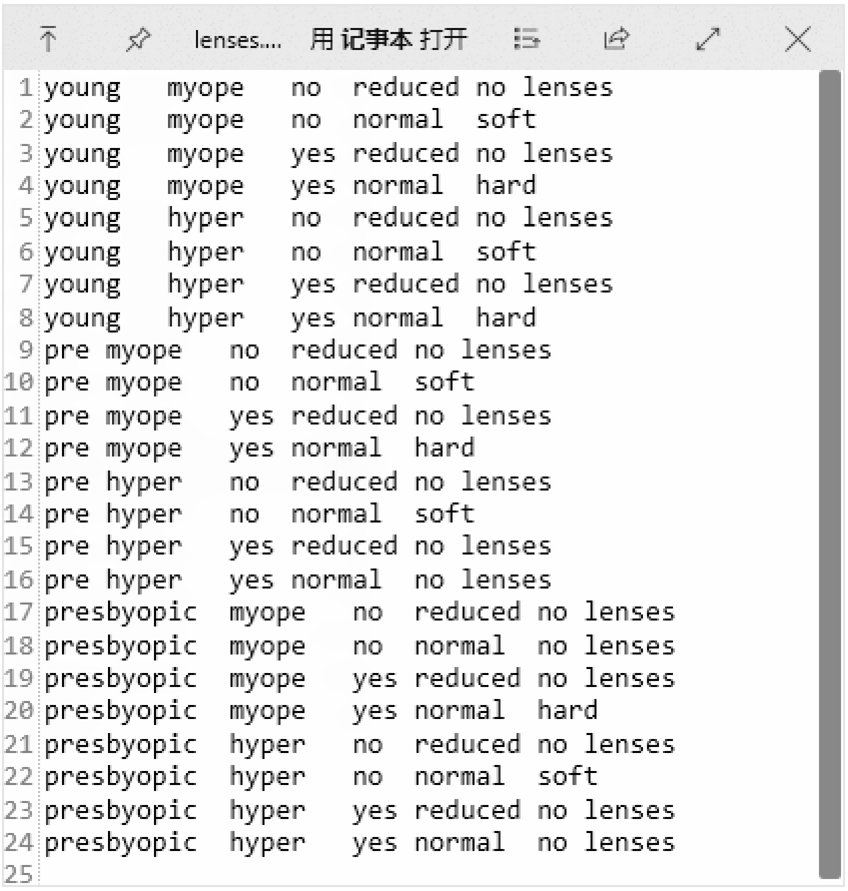
apply (self, X[, check_input])	Returns the index of the leaf that each sample is predicted as.
decision_path (self, X[, check_input])	Return the decision path in the tree
fit (self, X, y[, sample_weight, …])	Build a decision tree classifier from the training set (X, y).
get_depth (self)	Returns the depth of the decision tree.
get_n_leaves (self)	Returns the number of leaves of the decision tree.
get_params (self[, deep])	Get parameters for this estimator.
predict (self, X[, check_input])	Predict class or regression value for X.
predict_log_proba (self, X)	Predict class log-probabilities of the input samples X.
predict_proba (self, X[, check_input])	Predict class probabilities of the input samples X.
score (self, X, y[, sample_weight])	Returns the mean accuracy on the given test data and labels.
set_params (self, **params)	Set the parameters of this estimator.

二、sklearn决策树实战之挑选眼镜

在使用sklearn决策树之前, 先用前面构造的决策树代码运行一下.

之前的代码保存在trees.Hsu01.py中, 且眼镜的相关数据在lenses.txt中,.

lenses.txt中的数据大致如下:



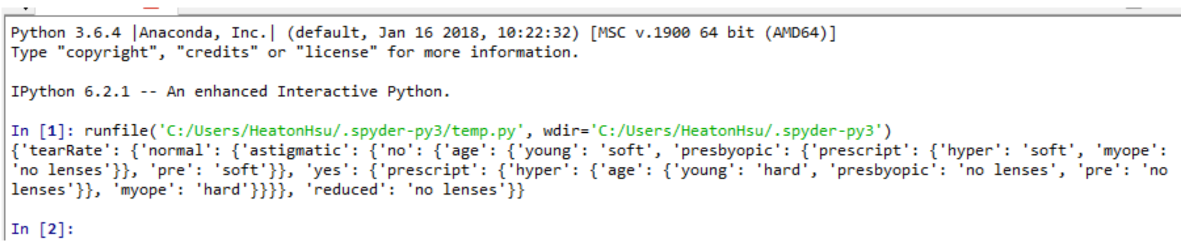
```
young	myope	no	reduced	no lenses
young	myope	no	normal	soft
young	myope	yes	reduced	no lenses
young	myope	yes	normal	hard
young	hyper	no	reduced	no lenses
young	hyper	no	normal	soft
young	hyper	yes	reduced	no lenses
young	hyper	yes	normal	hard
pre	myope	no	reduced	no lenses
pre	myope	no	normal	soft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

每列分别是: age, prescript, astigmatic 和 tearRate

那么用前面编写好的决策树分类器进行构造, 代码如下:

```python
import trees_Hsu01
#打开文件
with open('lenses.txt') as f:
    #读取所有的行,然后分别进行去除空格和以制表符分割
    #注意split()是以某个规则分隔形成列表
    lensens = [inst.strip().split('\t') for inst in f.readlines()]

lensesLabels = ['age','prescript', 'astigmatic', 'tearRate']
lensesTree = trees_Hsu01.createTree(lensens,lensesLabels)
print(lensesTree)
```

结果如下:



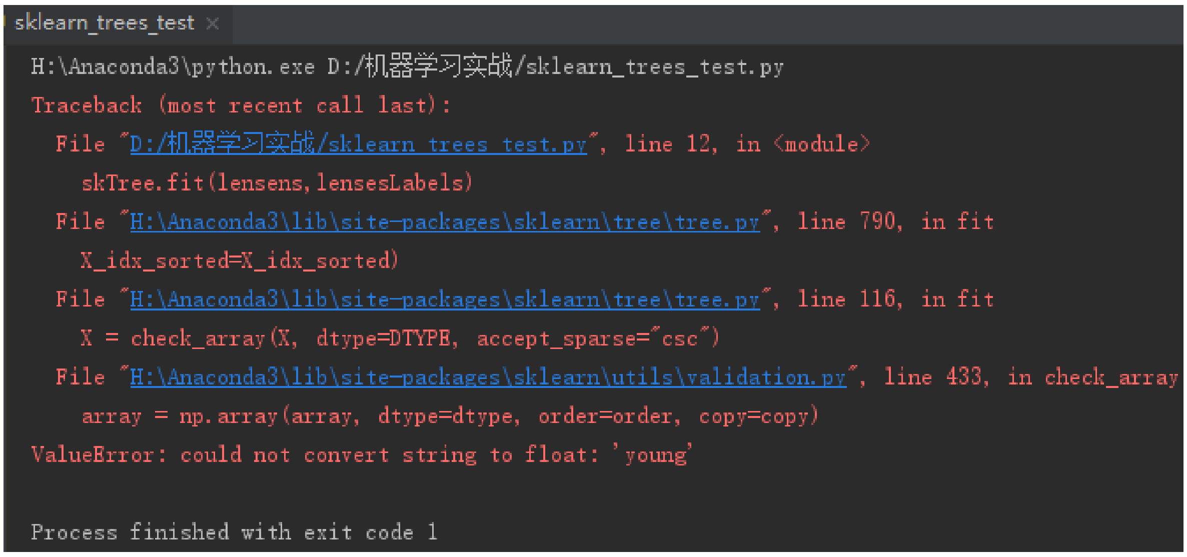
```
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/HeatonHsu/.spyder-py3/temp.py', wdir='C:/Users/HeatonHsu/.spyder-py3')
{'tearRate': {'normal': {'astigmatic': {'no': {'age': {'young': 'soft', 'presbyopic': {'prescript': {'hyper': 'soft', 'myope':
'no lenses'}}, 'pre': 'soft'}}, 'yes': {'prescript': {'hyper': {'age': {'young': 'hard', 'presbyopic': 'no lenses', 'pre': 'no
lenses'}}, 'myope': 'hard'}}}}, 'reduced': 'no lenses'}}

In [2]:
```

下面就来用sklearn决策树模块来运行

(一) 直接用sklearn出现的问题

如果不对原数据进行处理而直接用tree.DecisionTreeClassifer(), 如

```
from sklearn import tree
#打开文件
with open('lenses.txt') as f:
    #读取所有的行,然后分别进行去除空格和以制表符分割
    #注意split()是以某个规则分隔形成列表
    lensens = [inst.strip().split('\t') for inst in f.readlines()]

lensesLabels = ['age','prescript', 'astigmatic', 'tearRate']
skTree = tree.DecisionTreeClassifier()
#其实, 这里fit的第二个参数不能用lensesLabels, 应该是每个样本的类别,如soft,hard,no lense
等
#暂且输入参数
skTree.fit(lensens,lensesLabels)
```


```
H:\Anaconda3\python.exe D:/机器学习实战/sklearn_trees_test.py
Traceback (most recent call last):
  File "D:/机器学习实战/sklearn_trees_test.py", line 12, in <module>
    skTree.fit(lensens,lensesLabels)
  File "H:\Anaconda3\lib\site-packages\sklearn\tree\tree.py", line 790, in fit
    X_idx_sorted=X_idx_sorted)
  File "H:\Anaconda3\lib\site-packages\sklearn\tree\tree.py", line 116, in fit
    X = check_array(X, dtype=DTYPE, accept_sparse="csc")
  File "H:\Anaconda3\lib\site-packages\sklearn\utils\validation.py", line 433, in check_array
    array = np.array(array, dtype=dtype, order=order, copy=copy)
ValueError: could not convert string to float: 'young'

Process finished with exit code 1
```

直接报错了, 因为fit()函数不能直接接收`string`类型的数据. 因此, 在使用fit()函数之前, 需要**对数据进行编码.**

(二) 数据的编码

对数据进行编码, 主要使用两种方法:

1. LabelEncoder: 将字符串转换为增量值 , 适用于字符以及混合类型编码
2. OneHotEncoder: 使用 One-of-K算法将字符串转换为整数, 适用于非负整数类型编码

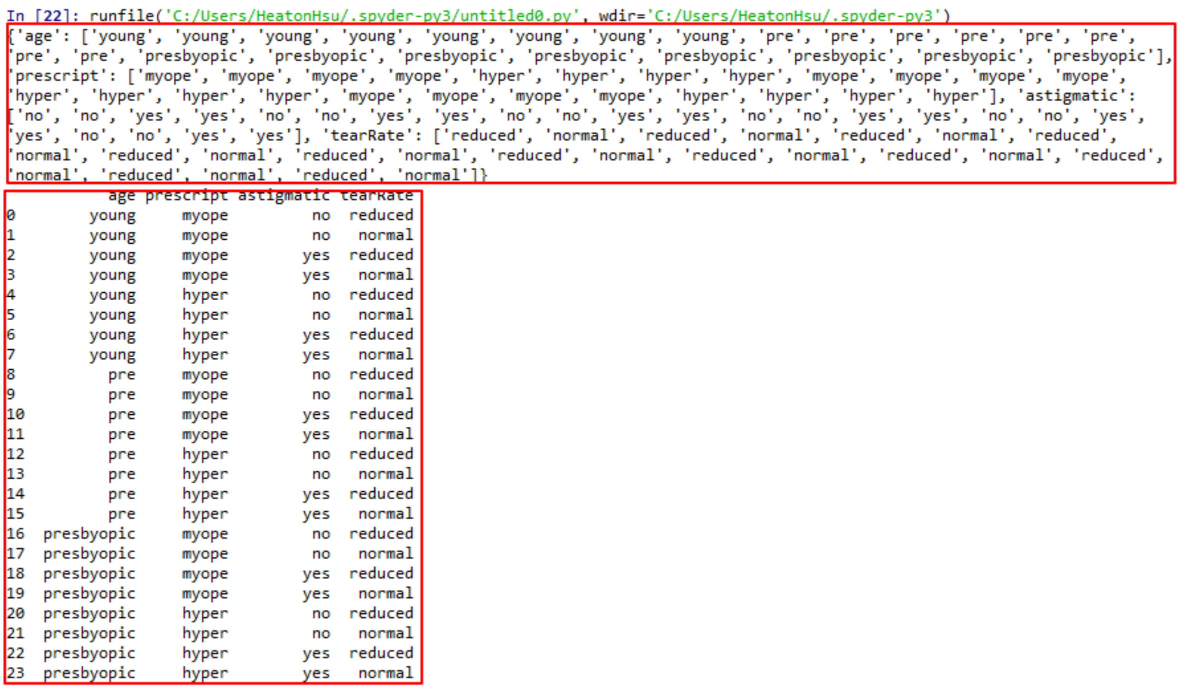
这里先将原始数据转化为字典, 再把字典转换称pd数据, 最后利用LabelEncoder转换称增量值,

具体如下:

```
from sklearn.preprocessing import LabelEncoder
from sklearn import tree
import pandas as pd
import numpy as np
```

```python
if __name__ == '__main__':
    #打开文件
    with open('lenses.txt') as f:
        #读取所有的行,然后分别进行去除空格和以制表符分割
        lenses = [inst.strip().split('\t') for inst in f.readlines()]
    lenses_target = []
    #获取每一行(每一个样本)的类别
    for each in lenses:
        lenses_target.append(each[-1])

    #构建特征标签,按照原始数据的顺序
    lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']
    #初始化特征列表和特征字典
    lenses_list = []
    lenses_dict = {}
    #遍历lenses列表,再遍历lensesLabels
    for each_label in lensesLabels:
        for each in lenses:
            #这段代码就是指每一个each_label(比如age)对应的元素都从lenses提取出来
            lenses_list.append(each[lensesLabels.index(each_label)])
        #然后处理完第一个eachlabel后把循环来的所有对应的元素组成的列表一起组成字典
        lenses_dict[each_label] = lenses_list
        #清除lenses_list,使得lenses_list重新为空,重新接收
        lenses_list = []

    #转化成pandas中的DataFrame格式
    lenses_pd = pd.DataFrame(lenses_dict)
    print(lenses_dict)
    print(lenses_pd)
```

看下结果:


```
In [22]: runfile('C:/Users/HeatonHsu/.spyder-py3/untitled0.py', wdir='C:/Users/HeatonHsu/.spyder-py3')
{'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', 'myope', 'myope', 'myope', 'hyper', 'hyper', 'hyper', 'hyper', 'myope', 'myope', 'myope', 'myope',
'hyper', 'hyper', 'hyper', 'hyper', 'myope', 'myope', 'myope', 'myope', 'hyper', 'hyper', 'hyper', 'hyper'], 'astigmatic':
['no', 'no', 'yes', 'yes', 'no', 'no', 'yes', 'yes', 'no', '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
```

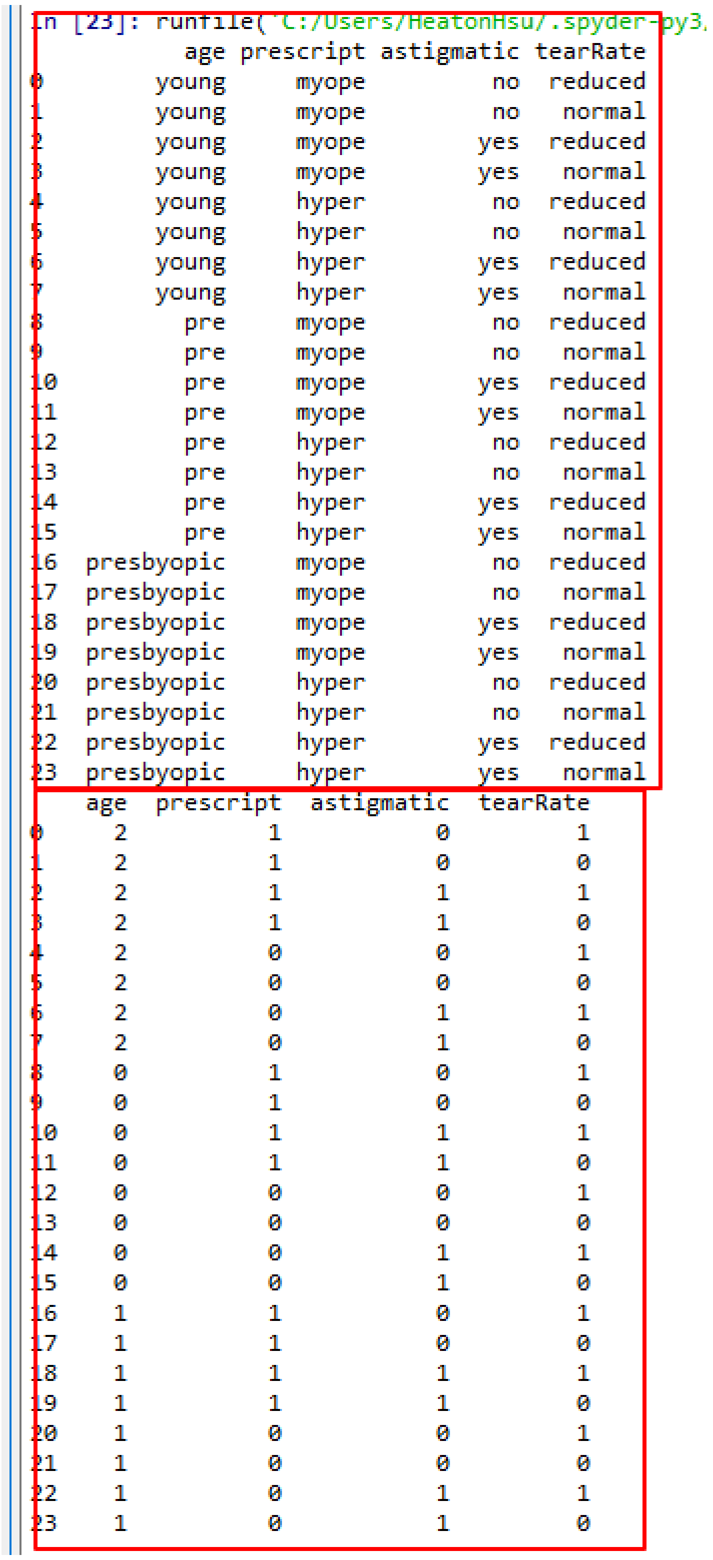
第一个就是lenses_dict, 第二个就是lenses_pd

接着就可以利用 LabelEncoder将lenses_pd转换称增量的形式,

具体如下:

```python
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.externals.six import StringIO
from sklearn import tree
import pandas as pd
import numpy as np

if __name__ == '__main__':
    #打开文件
    with open('lenses.txt') as f:
        #读取所有的行,然后分别进行去除空格和以制表符分割
        lenses = [inst.strip().split('\t') for inst in f.readlines()]
    lenses_target = []
    for each in lenses:
        lenses_target.append(each[-1])

    #构建特征标签
    lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']
    #初始化特征列表和特征字典
    lenses_list = []
    lenses_dict = {}
    #遍历lenses列表,再遍历lensesLabels
    for each_label in lensesLabels:
        for each in lenses:
            #这段代码就是指每一个each_label对应的元素都从lenses提取出来
            lenses_list.append(each[lensesLabels.index(each_label)])
        #然后处理完第一个eachlabel后把循环来的所有对应的元素组成的列表一起组成字典
        lenses_dict[each_label] = lenses_list
        #清除lenses_list,使得lenses_list重新为空,重新接收
        lenses_list = []


    #转化成pandas中的DataFrame格式
    lenses_pd = pd.DataFrame(lenses_dict)
    #print(lenses_dict)
    print(lenses_pd)
    #sklearn中的LabelEncoder,可以将字符串转化为增量数字
    le = LabelEncoder()
    #以列标签进行序列化
    for col in lenses_pd.columns:
        #方法fit_transform可以将fit和transform结合在一起处理
        lenses_pd[col] = le.fit_transform(lenses_pd[col])
    print(lenses_pd)
```

结果如下:



```
In [23]: runfile('C:/Users/HeatonHsu/.spyder-py3,
         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
```

第一个就是转化前的lenses_pd, 后一个就是利用LabelEncoder转化后的.

这里注意一下fit_transform, 它是把fit和transform结合在一起了.

先说下,**LabelEncoder的用法**:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit([1,5,67,100])
le.transform([1,1,100,67,5])

out:  array([0, 0, 3, 2, 1], dtype=int64)
```

也就是先拟合(即fit), 再转化(transform), 转化的根据就是来自于fit中给的数字的顺序.如果把fit里的数据改写一下, 结果就变成了:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit([12,15,1,5,67,100])
le.transform([1,1,100,67,5])

out:  array([0, 0, 5, 4, 1], dtype=int64)
```

而fit_transform是直接根据给的数字, 直接排序转化:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit_transform([1,1,100,67,5])

out:  array([0, 0, 3, 2, 1], dtype=int64)
```

可以体会其中的差别.

那么如果刚才的转化眼镜数据的代码改为先fit,再transform, 是什么样的, 如下:

```
"""
#转化成pandas中的DataFrame格式
lenses_pd = pd.DataFrame(lenses_dict)
#print(lenses_dict)
print(lenses_pd)
#sklearn中的LabelEncoder,可以将字符串转化为增量数字
le = LabelEncoder()
#以列标签进行序列化
for col in lenses_pd.columns:
    #方法fit_transform可以将fit和transform结合在一起处理
    lenses_pd[col] = le.fit_transform(lenses_pd[col])
print(lenses_pd)
"""


#转化成pandas中的DataFrame格式
lenses_pd = pd.DataFrame(lenses_dict)
#print(lenses_dict)
print(lenses_pd)
#sklearn中的LabelEncoder,可以将字符串转化为增量数字
le = LabelEncoder()
#以列标签进行序列化
for col in lenses_pd.columns:
    le.fit(list(set(lenses_pd[col])))
    lenses_pd[col] = le.transform(lenses_pd[col])
print(lenses_pd)
```

可以得到相同的结果.

(三) 使用tree.DecisionTreeClassifier

到这里, 我们就可以使用tree.DecisionTreeClassifier了

完整的代码如下:

```python
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.externals.six import StringIO
from sklearn import tree
import pandas as pd
import numpy as np

if __name__ == '__main__':
    #打开文件
    with open('lenses.txt') as f:
        #读取所有的行,然后分别进行去除空格和以制表符分割
        lenses = [inst.strip().split('\t') for inst in f.readlines()]
    lenses_target = []
    for each in lenses:
        lenses_target.append(each[-1])

    #构建特征标签
    lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']
    #初始化特征列表和特征字典
    lenses_list = []
    lenses_dict = {}
    #遍历lenses列表,再遍历lensesLabels
    for each_label in lensesLabels:
        for each in lenses:
            #这段代码就是指每一个each_label对应的元素都从lenses提取出来
            lenses_list.append(each[lensesLabels.index(each_label)])
        #然后处理完第一个eachlabel后把循环来的所有对应的元素组成的列表一起组成字典
        lenses_dict[each_label] = lenses_list
        #清除lenses_list,使得lenses_list重新为空,重新接收
        lenses_list = []
    #print(lenses_dict)
    #转化成pandas中的DataFrame格式
    lenses_pd = pd.DataFrame(lenses_dict)
    #print(lenses_pd)
    #sklearn中的LabelEncoder,可以将字符串转化为增量数字
    le = LabelEncoder()
    #以列标签进行序列化
    for col in lenses_pd.columns:
        #方法fit_transform可以将fit和transform结合在一起处理
        lenses_pd[col] = le.fit_transform(lenses_pd[col])
    print(lenses_pd)


    clf = tree.DecisionTreeClassifier(max_depth=4)
    clf.fit(lenses_pd.values.tolist(),lenses_target)
    a = np.array([0,1,1,0])
    result = clf.predict(a.reshape(1,-1))
    print(result)
```

输入了一个a, 预测其最后的结果.最后显示为: hard

注意: a.reshape(1,-1)的含义是一个样本, 如果是一个特征的话应该写成: a.reshape(-1,1)

三、决策树可视化

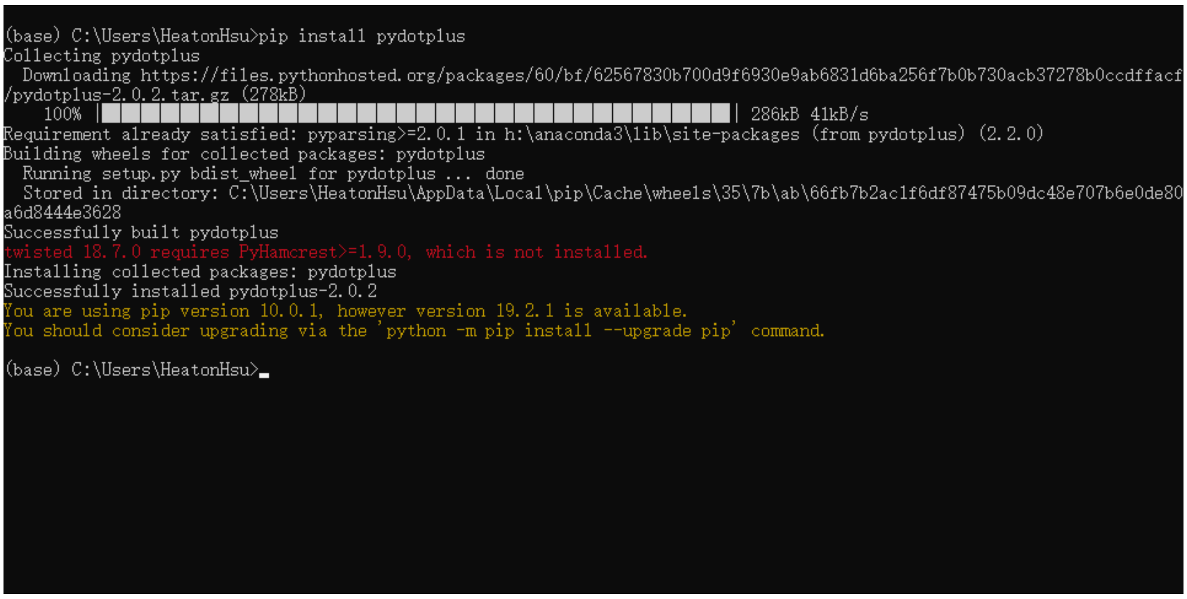
Graphviz的是AT&T Labs Research开发的图形绘制工具, 他可以很方便的用来绘制结构化的图形网络, 支持多种格式输出, 生成图片的质量和速度都不错. 它的输入是一个用dot语言编写的绘图脚本, 通过对输入脚本的解析, 分析出其中的点, 边以及子图, 然后根据属性进行绘制.

而Sklearn生成的决策树就是dot格式的, 因此我们可以直接利用Graphviz将决策树可视化.

要想正常使用Graphviz需要安装pydotplus和graphviz.

(一) 安装pydotplus

直接pip安装即可



```
(base) C:\Users\HeatonHsu>pip install pydotplus
Collecting pydotplus
  Downloading https://files.pythonhosted.org/packages/60/bf/62567830b700d9f6930e9ab6831d6ba256f7b0b730acb37278b0ccdffacf
/pydotplus-2.0.2.tar.gz (278kB)
    100% |████████████████████████████████| 286kB 41kB/s
Requirement already satisfied: pyparsing>=2.0.1 in h:\anaconda3\lib\site-packages (from pydotplus) (2.2.0)
Building wheels for collected packages: pydotplus
  Running setup.py bdist_wheel for pydotplus ... done
  Stored in directory: C:\Users\HeatonHsu\AppData\Local\pip\Cache\wheels\35\7b\ab\66fb7b2ac1f6df87475b09dc48e707b6e0de80
a6d8444e3628
Successfully built pydotplus
twisted 18.7.0 requires PyHamcrest>=1.9.0, which is not installed.
Installing collected packages: pydotplus
Successfully installed pydotplus-2.0.2
You are using pip version 10.0.1, however version 19.2.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

(base) C:\Users\HeatonHsu>
```

同时, 如果是pycharm或者anoconda, 还可以在软件里安装, 方法不赘述.

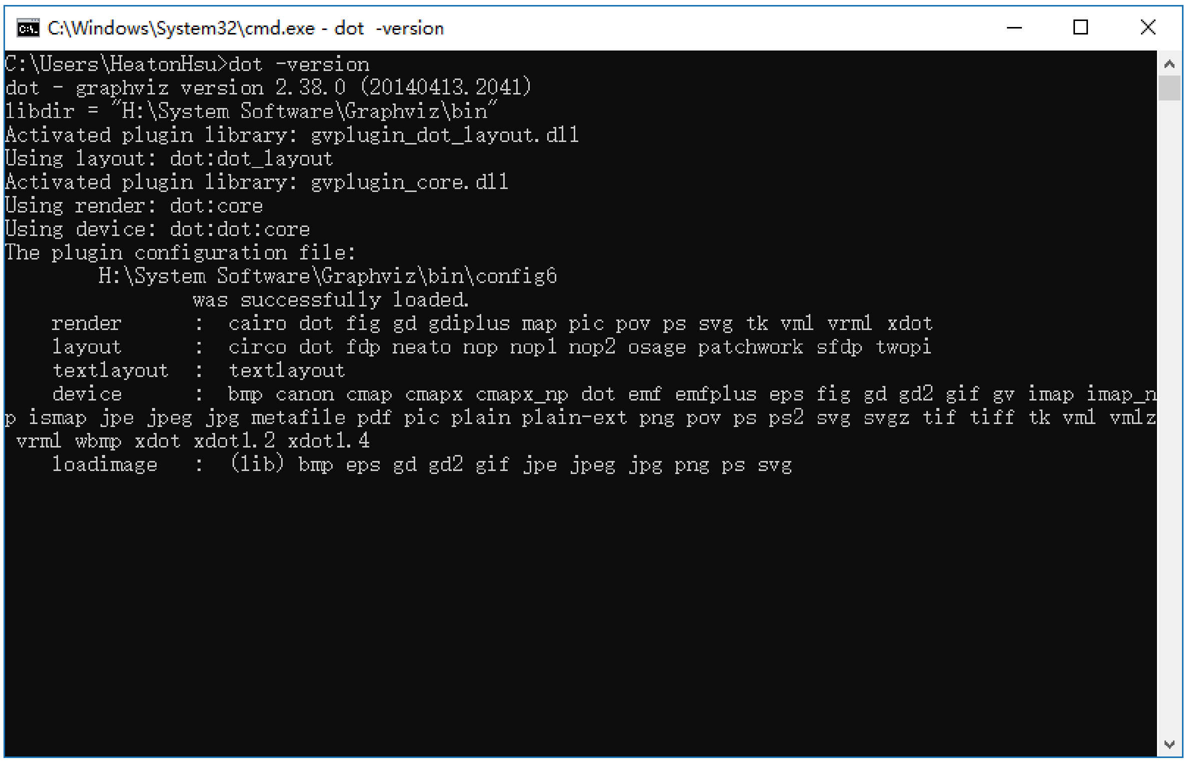
(二) 安装Graphviz

下载地址在这里: https://graphviz.gitlab.io/_pages/Download/Download_windows.html

安装完成后, 把安装地址添加到环境变量里, 比如我的安装路径在`H:\System Software\Graphviz`, 那么就把:`H:\System Software\Graphviz\bin`添加的环境变量.

步骤: 右击我的电脑-高级系统设置-环境变量, 选择用户变量里的`Path`, 点击编辑-新建, 把路径`H:\System Software\Graphviz\bin`添加进去,确定即可.

查看是否安装完成, 且已经在环境变量中了, 可以在cmd中输入: `dot -version`, 如果出现下面的输出, 说明一切正常.

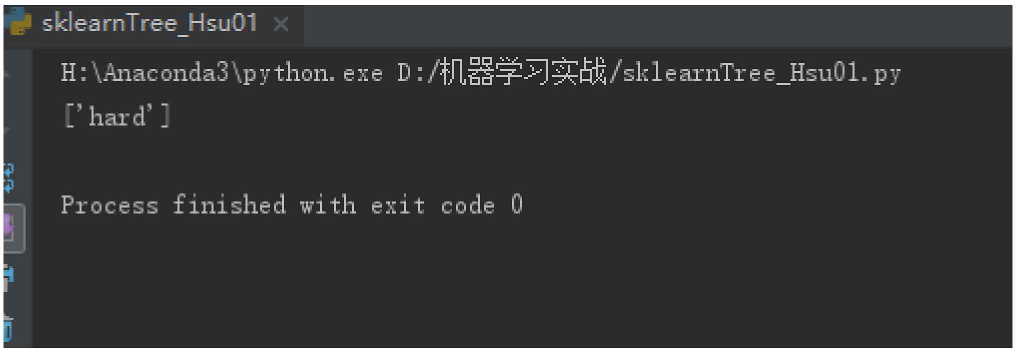


```
C:\Users\HeatonHsu>dot -version
dot - graphviz version 2.38.0 (20140413.2041)
libdir = "H:\System Software\Graphviz\bin"
Activated plugin library: gvplugin_dot_layout.dll
Using layout: dot:dot_layout
Activated plugin library: gvplugin_core.dll
Using render: dot:core
Using device: dot:dot:core
The plugin configuration file:
        H:\System Software\Graphviz\bin\config6
                was successfully loaded.
    render      :  cairo dot fig gd gdiplus map pic pov ps svg tk vml vrml xdot
    layout      :  circo dot fdp neato nop nop1 nop2 osage patchwork sfdp twopi
    textlayout  :  textlayout
    device      :  bmp canon cmap cmapx cmapx_np dot emf emfplus eps fig gd gd2 gif gv imap imap_n
p ismap jpe jpeg jpg metafile pdf pic plain plain-ext png pov ps ps2 svg svgz tif tiff tk vml vmlz
 vrml wbmp xdot xdot1.2 xdot1.4
    loadimage   :  (lib) bmp eps gd gd2 gif jpe jpeg jpg png ps svg
```

(三) 决策树可视化

```python
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.externals.six import StringIO
from sklearn import tree
import pandas as pd
import numpy as np
import pydotplus

if __name__ == '__main__':
    #打开文件
    with open('lenses.txt') as f:
        #读取所有的行,然后分别进行去除空格和以制表符分割
        lenses = [inst.strip().split('\t') for inst in f.readlines()]
    lenses_target = []
    for each in lenses:
        lenses_target.append(each[-1])

    #构建特征标签
    lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']
    #初始化特征列表和特征字典
    lenses_list = []
    lenses_dict = {}
    #遍历lenses列表,再遍历lensesLabels
    for each_label in lensesLabels:
        for each in lenses:
            #这段代码就是指每一个each_label对应的元素都从lenses提取出来
            lenses_list.append(each[lensesLabels.index(each_label)])
        #然后处理完第一个eachlabel后把循环来的所有对应的元素组成的列表一起组成字典
        lenses_dict[each_label] = lenses_list
        #清除lenses_list,使得lenses_list重新为空,重新接收
        lenses_list = []
    #print(lenses_dict)
    #转化成pandas中的DataFrame格式
    lenses_pd = pd.DataFrame(lenses_dict)
    #print(lenses_pd)
    #sklearn中的LabelEncoder,可以将字符串转化为增量数字
    le = LabelEncoder()
    #以列标签进行序列化
    for col in lenses_pd.columns:
        #方法fit_transform可以将fit和transform结合在一起处理
        lenses_pd[col] = le.fit_transform(lenses_pd[col])
    #print(lenses_pd)
    """上面的代码也可以这样写:

    #转化成pandas中的DataFrame格式
    lenses_pd = pd.DataFrame(lenses_dict)
    #print(lenses_dict)
    print(lenses_pd)
    #sklearn中的LabelEncoder,可以将字符串转化为增量数字
    le = LabelEncoder()
    #以列标签进行序列化
    for col in lenses_pd.columns:
        le.fit(list(set(lenses_pd[col])))
        lenses_pd[col] = le.transform(lenses_pd[col])
    print(lenses_pd)
    """
    #引入决策树训练方法,其中最大深度为4
    clf = tree.DecisionTreeClassifier(max_depth=4)
    #进行数据拟合
    clf.fit(lenses_pd.values.tolist(),lenses_target)

    #对a进行预测
    a = np.array([0,1,1,0])
    result = clf.predict(a.reshape(1,-1))
    print(result)


    #输出决策树图像
    #创建绘图对象
    dot_data = StringIO()
    #输出图像数据,具体参数见tree.export_graphviz(
    tree.export_graphviz(clf,out_file=dot_data,
                         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())
    graph.write_pdf('树结构.pdf')
```

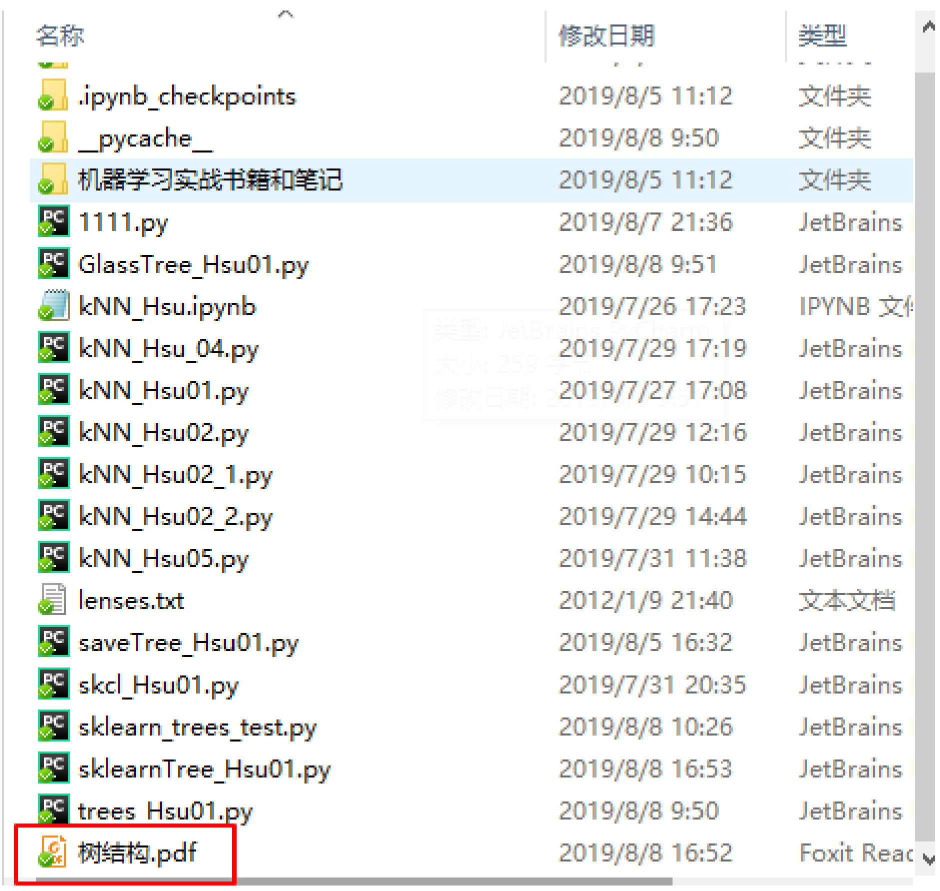
运行后, 结果如下:



sklearnTree_Hsu01

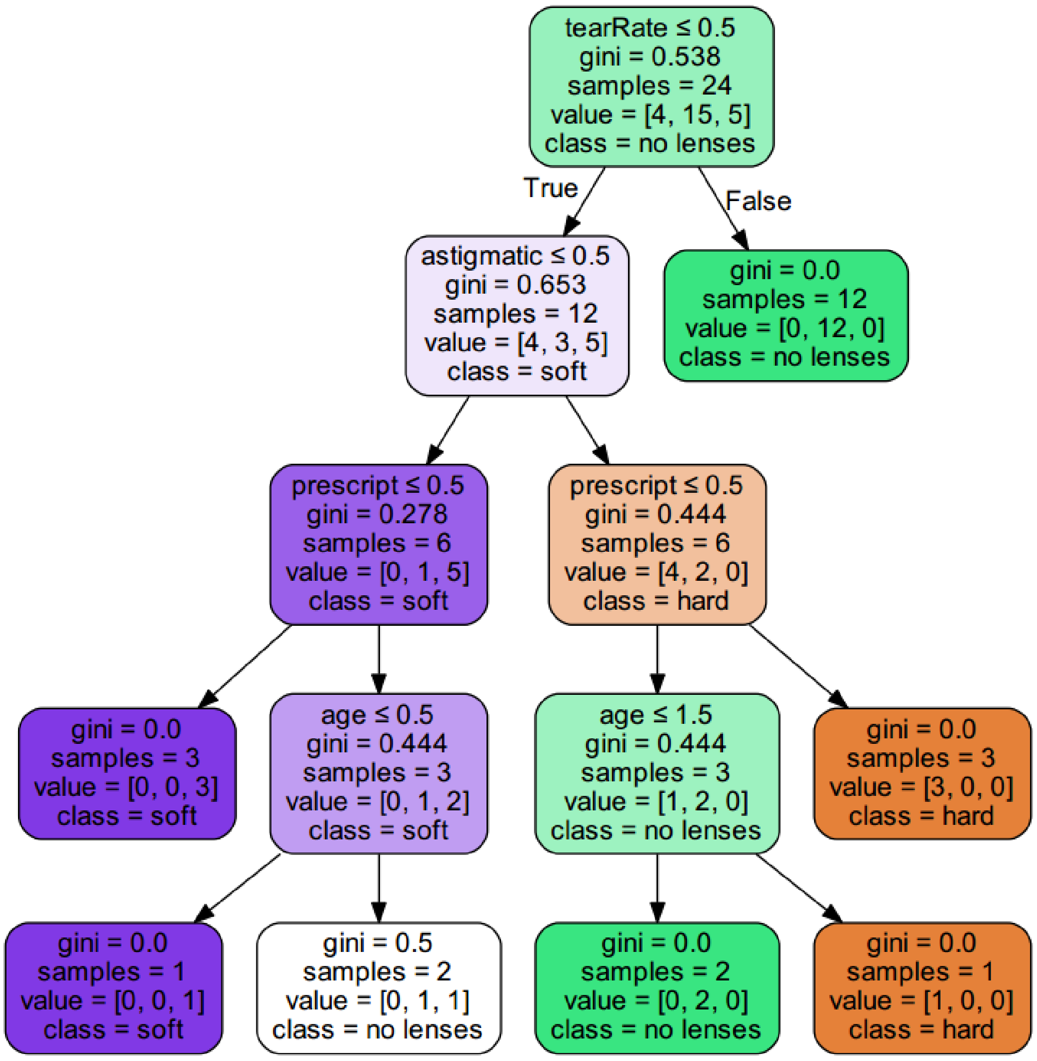
```
H:\Anaconda3\python.exe D:/机器学习实战/sklearnTree_Hsu01.py
['hard']

Process finished with exit code 0
```

生成的pdf文件在我的工作目录中, 如下:



名称	修改日期	类型
.ipynb_checkpoints	2019/8/5 11:12	文件夹
__pycache__	2019/8/8 9:50	文件夹
机器学习实战书籍和笔记	2019/8/5 11:12	文件夹
1111.py	2019/8/7 21:36	JetBrains
GlassTree_Hsu01.py	2019/8/8 9:51	JetBrains
kNN_Hsu.ipynb	2019/7/26 17:23	IPYNB 文
kNN_Hsu_04.py	2019/7/29 17:19	JetBrains
kNN_Hsu01.py	2019/7/27 17:08	JetBrains
kNN_Hsu02.py	2019/7/29 12:16	JetBrains
kNN_Hsu02_1.py	2019/7/29 10:15	JetBrains
kNN_Hsu02_2.py	2019/7/29 14:44	JetBrains
kNN_Hsu05.py	2019/7/31 11:38	JetBrains
lenses.txt	2012/1/9 21:40	文本文档
saveTree_Hsu01.py	2019/8/5 16:32	JetBrains
skcl_Hsu01.py	2019/7/31 20:35	JetBrains
sklearn_trees_test.py	2019/8/8 10:26	JetBrains
sklearnTree_Hsu01.py	2019/8/8 16:53	JetBrains
trees_Hsu01.py	2019/8/8 9:50	JetBrains
树结构.pdf	2019/8/8 16:52	Foxit Reac

打开文件后, 树结构如下:



下一篇就到了朴素贝叶斯了, 看看朴素贝叶斯的实际应用.