

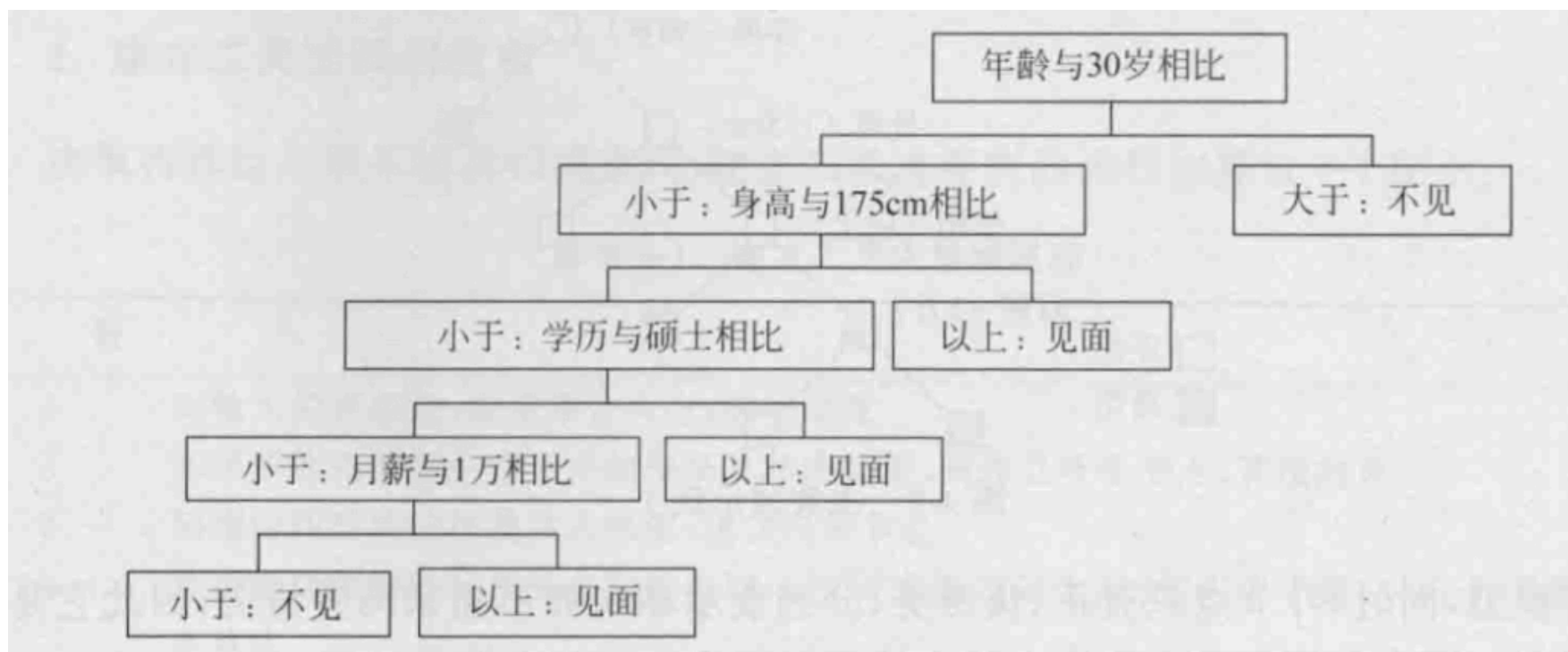
决策树

实验内容：动手实现分类决策数，并与调用sklearn的决策树进行对比

1. 决策树的基本思想

1.1 树状结构：

- 与人类思考过程非常相似，某一特征在某一指标下的表现情况，以此进入下一个分支选择。
- 由节点和分支组成，类似于树的结构。每个内部节点代表一个特征上的测试，每个分支代表测试的结果，每个叶节点代表一个类别或决策。



1.2 特征选择:

- 在每个内部节点, 算法会选择一个特征和该特征的一个阈值, 将数据集分割成两个(二叉决策树)或多个子集。
- 人在思考时对于“年龄、身高、学历、月薪”会有一个下意识的先后排序, 而决策数则需要一个定量的指标来选择需要优先考量的特征。在选择划分属性时, 并不能随意选择任意属性作为划分依据, 因为可能有些属性并不能明显划分不同的类, 可能是一些无意义的属性。希望通过划分选择使得分支节点所包含的样本纯度尽可能属于同一类别。

常用划分选择算法:

ID3

Iterative Dichotome 迭代二分器, 以**信息增益**为准则来选择划分属性。信息增益的计算公式为:

$$Gain(X, A) = Ent(X) - \sum_{v \in A} \frac{|X_v|}{|X|} Ent(X_v)$$

其中 $Ent(\cdot)$ 为**信息熵**, 计算公式为:

$$Ent(X) = - \sum_{i=1}^k p_i \log_2 p_i$$

信息熵是度量样本集合纯度最常用的一种指标。由信息熵的定义可知, 熵值越大越混乱, 因此信息熵越小, X越趋于稳定, 即X的纯度越高。

对于属性A, 信息熵越小, 这个属性的纯度越大, 这个属性对整体划分的作用越大, 即信息增益越大, 使用属性A来划分所得到的纯度提升越高。

ID3的缺点: 容易偏向于取值较多的特征。当特征的取值较多时, 根据此特征划分更容易得到纯度更高的样本子集, 因此划分后的信息熵更低, 即不确定性更低, 因此信息增益更大。

C4.5

C4.5算法是ID3算法的改进版本, 以**信息增益率**为准则来选择划分属性, 计算公式为:

$$Gain_Ratio(X, A) = \frac{Gain(X, A)}{Split_Info(A)}$$

其中

$$Split_Info(A) = - \sum_{v \in A} \frac{|X_v|}{|X|} \log_2 \frac{|X_v|}{|X|}$$

Split_Info(A)为属性A的固有值，属性A的可取值数目越多，Split_Info(A)的取值越大。因此基于信息增益率的算法对可取值数目较少的属性有所偏好。

CART

CART算法使**基尼指数**来选择划分，计算公式为：

$$Gini(D) = 1 - \sum_{i=1}^k p_i^2$$

基尼指数越大，X越混乱，即纯度越低。对于某一属性A，其基尼指数公式为：

$$Gini_Index(D, A) = \sum_{v \in A} \frac{|D_v|}{|D|} Gini(D)$$

1.3 决策树的生成过程与停止生成条件

递归生成：选择特征和阈值后，算法递归地对每个子集应用相同的过程，直到满足停止条件

停止生成条件：

1. 当前结点的所有样本全属于同一类，无需划分
2. 所有属性都已经划分，或是所有样本在所有属性上取值相同，无法划分
3. 当前结点包含的样本集合为空，不能划分

1.4 决策树的训练与测试：

- 训练时，利用样本数据，以样本集合纯度作为划分指标建立决策树模型

- 测试时，对新的样本数据，利用决策模型进行分类预测

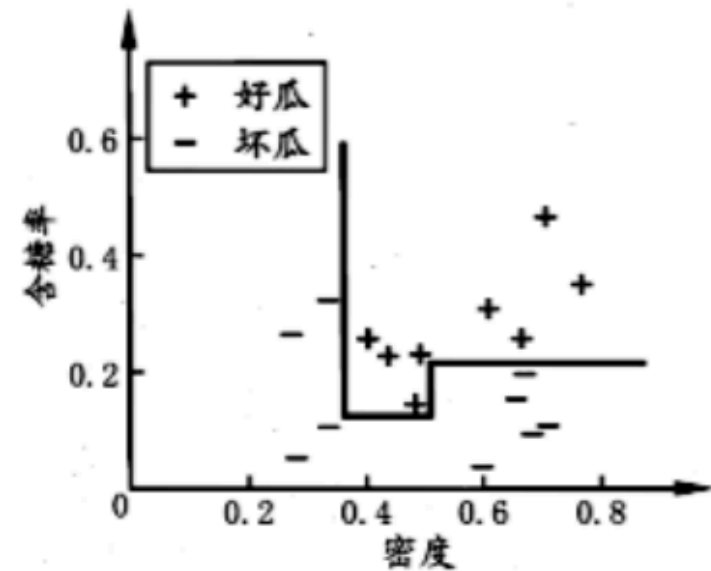
1.5 过拟合对策

决策树算法在学习的过程中为了尽可能多的分类训练样本，不停地对结点进行划分，因此这会导致整棵树的分支过多，也就导致了过拟合。常用对策为**剪枝处理**

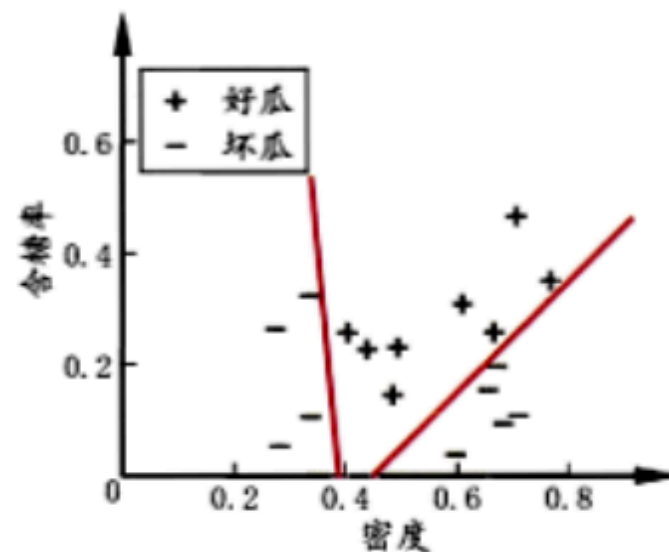
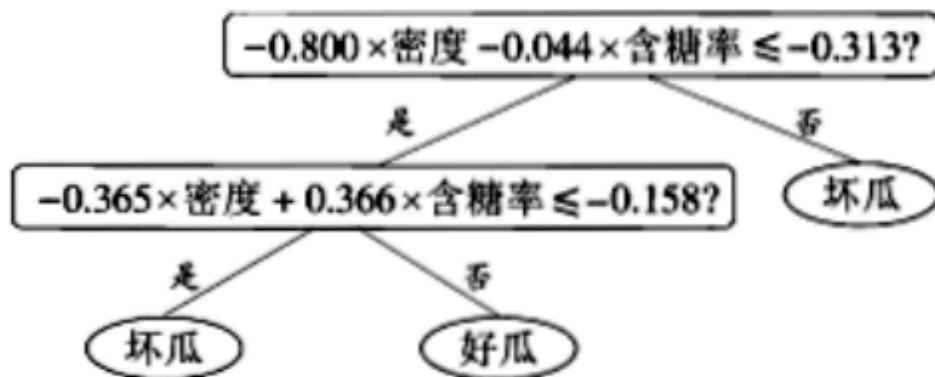
- 预剪枝：在构造决策树的过程中，先对每个结点在划分前进行估计，若果当前结点的划分不能带来决策树模型泛化性能的提升，则不对当前结点进行划分并且将当前结点标记为叶结点。这种方法优点在于能缓解过拟合，还能减少训练的时间和开销，但是可能会导致欠拟合。
- 后剪枝：先把整颗决策树构造完毕，然后自下而上的对非叶结点进行考察，若将该结点对应的子树换为叶结点能够带来泛化性能的提升，则把该子树替换为叶结点。比起预剪枝，后剪枝就不容易导致欠拟合，但是要在决策树生成后自下而上遍历结点，增加了时间和开销

1.6 多变量决策树

单变量决策在每个非叶结点仅考虑一个划分属性，分类决策边界轴平行



当学习任务的真实分类边界比较复杂时，需要多段划分才能获得较好的近似。多变量决策树的每个非叶结点不只考虑一个属性，而是建立一个线性分类器



2. 决策树的sklearn调用


```
In [1]: # 导入必要的库
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
```

```
In [2]: # 加载数据集
data = fetch_openml(name='adult', version=2, as_frame=True)
X = data.data
y = data.target
```

```
In [3]: # 原始数据集
X.head()
```

Out[3]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	unemployed
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	U
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	U
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	U
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	U
4	18	NaN	103497	Some-college	10	Never-married	NaN	Own-child	White	Female	0	0	30	U



```
In [4]: # 原始标签集
y.head()
```

```
Out[4]: 0    <=50K
1    <=50K
2     >50K
3     >50K
4    <=50K
Name: class, dtype: category
Categories (2, object): ['<=50K', '>50K']
```

原始的数据中含有非数值数据，需要进行数据预处理，统计每一个数据的取值然后进行编码（非常麻烦）

sklearn提供**LabelEncoder**能将分类特征转换为数值型

```
In [ ]: # 数据预处理：对分类变量进行编码
label_encoders = {}
for column in X.select_dtypes(include=['category']).columns:
    label_encoders[column] = LabelEncoder()
    X[column] = label_encoders[column].fit_transform(X[column])

# 数据预处理：对标签进行编码
label_encoders_y = LabelEncoder()
y = label_encoders_y.fit_transform(y)
```

```
In [6]: # 查看数据集的前几行
X.head()
```

```
Out[6]:
```

	age	workclass	fnlwgt	education	education- num	marital- status	occupation	relationship	race	sex	capital- gain	capital- loss	hours- per- week	native- country
0	25	3	226802	1	7	4	6	3	2	1	0	0	40	38
1	38	3	89814	11	9	2	4	0	4	1	0	0	50	38
2	28	1	336951	7	12	2	10	0	4	1	0	0	40	38
3	44	3	160323	15	10	2	6	0	2	1	7688	0	40	38
4	18	8	103497	15	10	4	14	3	4	0	0	0	30	38

```
In [7]: # 检查标签集
print(y)
```

```
[0 0 1 ... 0 0 1]
```

```
In [8]: # 划分数据集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
In [9]: # 创建决策树模型
clf = DecisionTreeClassifier(max_depth=10, random_state=42) # max_depth控制树的深度，便于后续可视化
```

```
In [10]: # 训练模型
clf.fit(X_train, y_train)
```

```
Out[10]: ▼ DecisionTreeClassifier ⓘ ⓘ
DecisionTreeClassifier(max_depth=10, random_state=42)
```

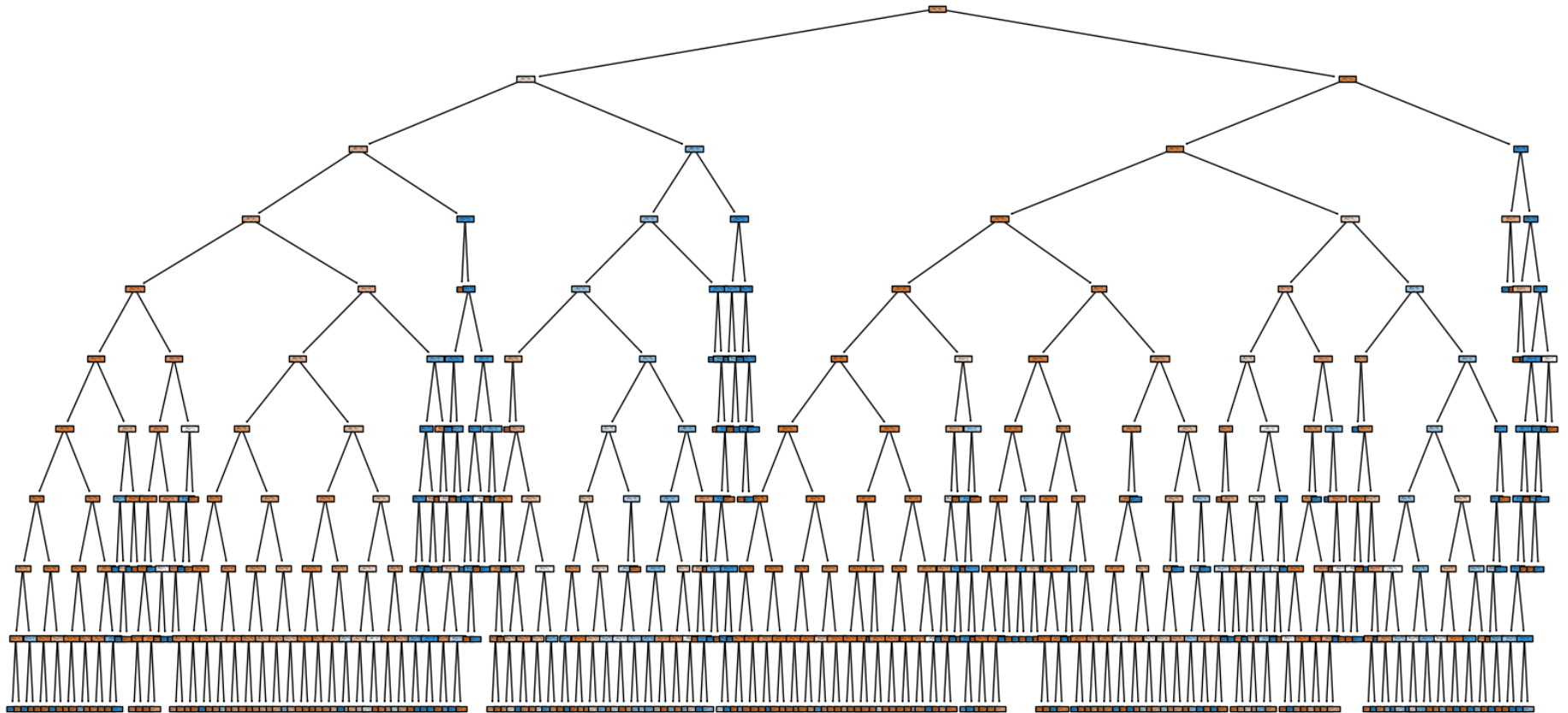
```
In [11]: # 预测测试集
y_pred = clf.predict(X_test)
```

```
In [12]: # 评估模型
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

Accuracy: 0.86

```
In [13]: # 结果可视化
plt.figure(figsize=(20,10))
plot_tree(clf, filled=True, feature_names=data.feature_names) # 受max_depth控制，深度为10，若不控制max_depth，绘制时间需要更长
plt.title('Decision Tree for Adult Dataset')
plt.show()
```


Decision Tree for Adult Dataset



3. 动手实践

- 请自己实现一个决策树分类器，你可能需要考虑：单变量or多变量、划分算法、连续值的处理、决策树的递归生成、决策树的生成终止条件、分类器的训练函数、分类器的预测函数.....
- 请调用自己实现的决策树分类器，对**adult**数据进行二分类，检验分类准确率（注意该数据的一般分类准确率为70%~90%），你可能需要考虑：分类特征转换、缺失值处理、冗余特征剔除.....
- 注意保存本次实验的代码