D E C I S I O N T R E E

决策树

原理与python实现

汇报人： 廖如瑶

高尔夫球场经理的管理难题

经理小王想要通过天气情况预测顾客是否会来俱乐部打球

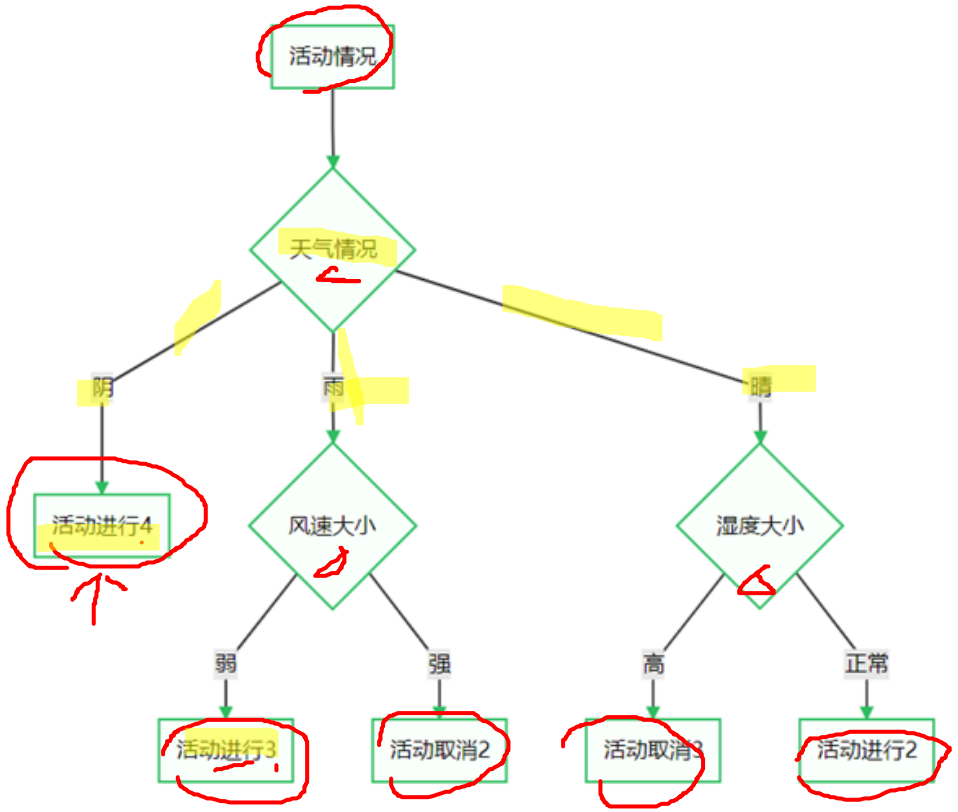
收集关于天气的四个特征数据

建立模型：输入天气特征，输出活动是否进行

天气	温度	湿度	风速	活动
晴	炎热	高	弱	取消
晴	炎热	高	强	取消
阴	炎热	高	弱	进行
雨	适中	高	弱	进行
雨	寒冷	正常	弱	进行
雨	寒冷	正常	强	取消
阴	寒冷	正常	强	进行
晴	适中	高	弱	取消
晴	寒冷	正常	弱	进行
雨	适中	正常	弱	进行
晴	适中	正常	强	进行
阴	适中	高	强	进行
阴	炎热	正常	弱	进行
雨	适中	高	强	取消

——《机器学习》Tom M.Mitchell

绘制决策树解决难题



决策树：是一个树结构（可以是二叉树或非二叉树）其每个非叶节点表示一个特征属性上的测试，每个分支代表这个特征属性在某个值域上的输出，而每个叶节点存放一个类别

使用决策树进行决策的过程就是从根节点开始，测试待分类项中相应的特征属性，并按照其值选择输出分支，直到到达叶子节点，将叶子节点存放的类别作为决策结果

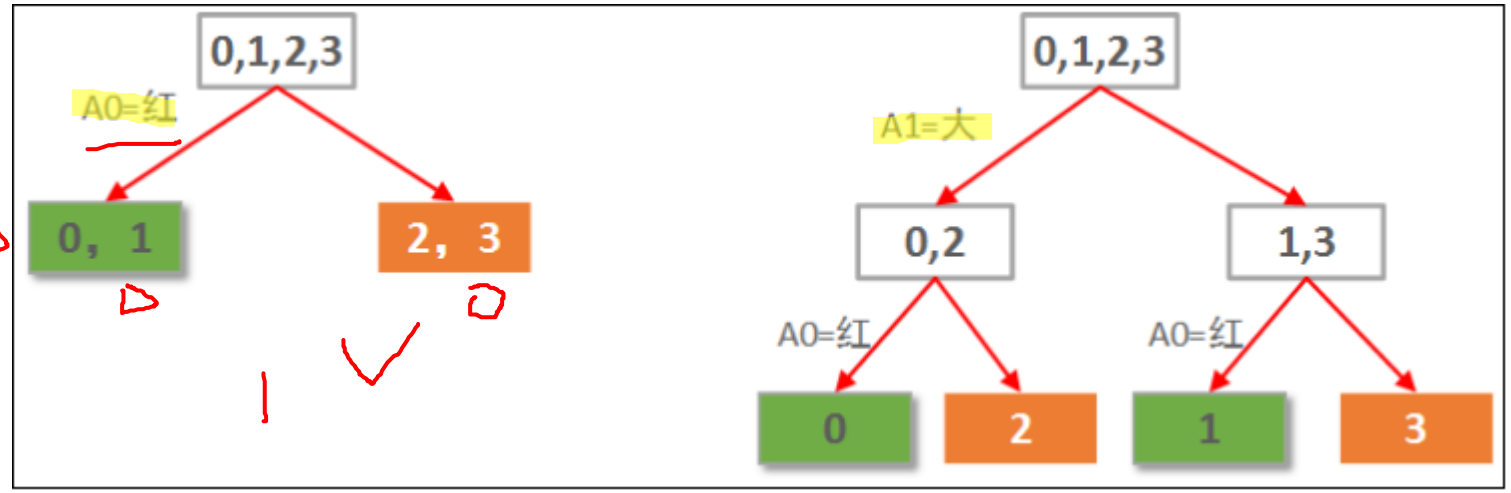
决策树算法就是利用“先验数据”构造一棵最佳的决策树，用以预测未知数据的类别

决策树的构造评价指标

样本	红	大	好苹果
0	1	1	1
1	1	0	1
2	0	1	0
3	0	0	0

思考：

- 哪棵树更好？
- 有什么评价指标？ → 信息熵



信息熵

熵：熵度量了随机事件的不确定性，越不确定的事物，它的熵就越大

$$H(X) = -\sum_{i=1}^{n} p_i log p_i$$

其中n代表X的n种不同的离散取值

pi代表X取值为i的概率；

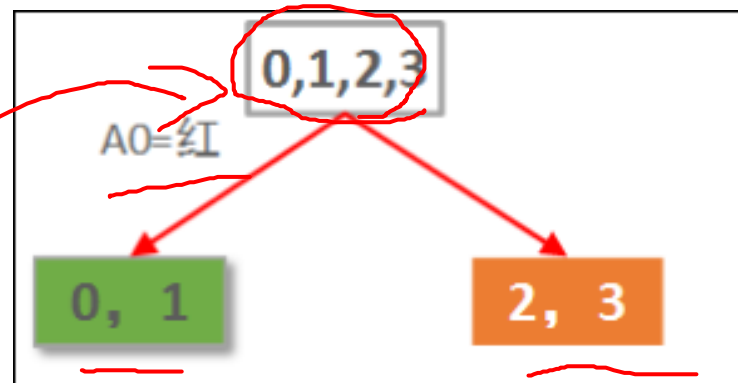
log是以2或者e为底的对数

原始样本数据的熵：

样例总数：4

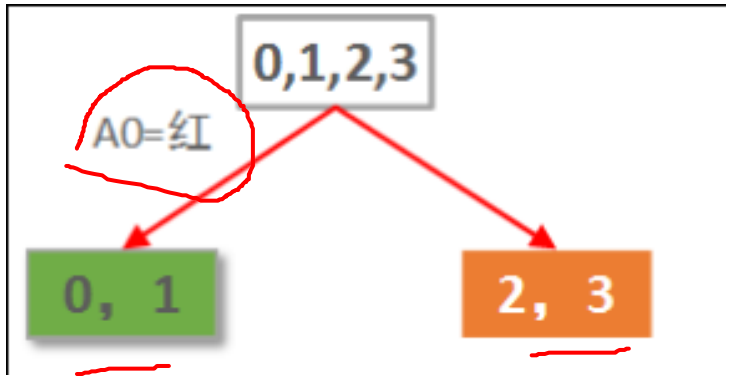
好苹果：2

坏苹果：2



熵：S= -(1/2 * log(1/2) + 1/2 * log(1/2)) = 1

计算信息熵



选择A0作为特征划分

各子节点的信息熵为：e1 = -(2/2 * log(2/2) + 0/2 * log(0/2)) = 0

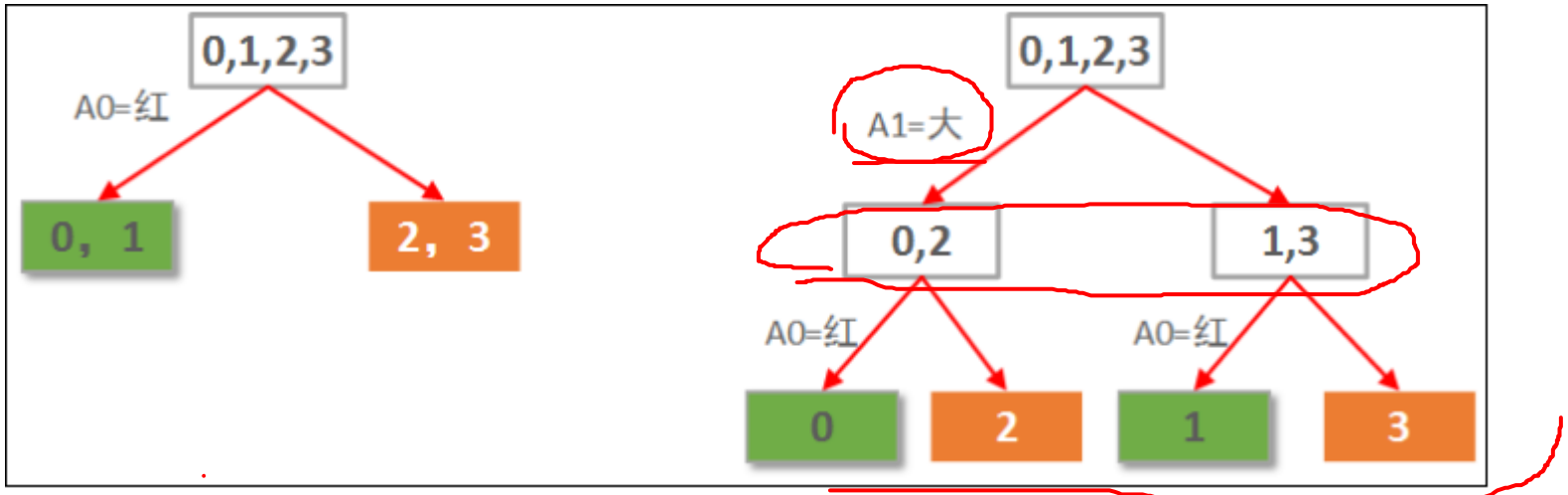
e2 = -(2/2 * log(2/2) + 0/2 * log(0/2)) = 0

因此选择A0划分后的信息熵为每个子节点的信息熵所占比重的加权和：

E = e1*2/4 + e2*2/4 = 0

选择A0做划分的信息熵增益G（S, A0）=S - E = 1 - 0 = 1

决策树的构造评价指标



选择A1作为划分特征，各子节点信息熵计算如下：

e1 = -(1/2 * log(1/2) + 1/2 * log(1/2)) = 1

e2 = -(1/2 * log(1/2) + 1/2 * log(1/2)) = 1

因此选择A1划分后的信息熵为：E = e1*2/4 + e2*2/4 = 1

选择A1做划分的信息熵增益G（S, A1）=S - E = 1 - 1 = 0

计算出按各个特征的信息熵增益，选择最大的那种进行划分即可！

ID3算法过程

- 数据是怎么分裂的?
- 如何选择分类的属性?
- 什么时候停止分裂?

停止条件：

- 所有特征使用完毕或叶节点的纯度为1
- 结点的数据样本或树的叶结点的熵小于阈值

1. 计算原始数据集的信息熵
2. 遍历属性集，计算按照各个属性分类后子集的信息熵加权和
3. 求出各个属性的信息增益，然后比较选择信息增益最大的属性
4. 循环第二步和第三步，直到达到停止条件，则输出最终决策树

代码框架：加亿点点细节...

```
#! usr/bin/env
#coding=utf-8
from math import log
import numpy as np
import operator
import pandas as pd
```

	天气	温度	湿度	风速	活动
0	晴	炎热	高	弱	取消
1	晴	炎热	高	强	取消
2	阴	炎热	高	弱	进行
3	雨	适中	高	弱	进行
4	雨	寒冷	正常	弱	进行
5	雨	寒冷	正常	强	取消
6	阴	寒冷	正常	强	进行
7	晴	适中	高	弱	取消
8	晴	寒冷	正常	弱	进行
9	雨	适中	正常	弱	进行
10	晴	适中	正常	强	进行
11	阴	适中	高	强	进行
12	阴	炎热	正常	弱	进行
13	雨	适中	高	强	取消

```
def creatDataSet(filename):
    return list(dataSet),classLabelVector
```

```
([array([2., 1., 1., 0., 0.]),
  array([2., 1., 1., 1., 0.]),
  array([0., 1., 1., 0., 1.]),
  array([1., 0., 1., 0., 1.]),
  array([1., 2., 0., 0., 1.]),
  array([1., 2., 0., 1., 0.]),
  array([0., 2., 0., 1., 1.]),
  array([2., 0., 1., 0., 0.]),
  array([2., 2., 0., 0., 1.]),
  array([1., 0., 0., 0., 1.]),
  array([2., 0., 0., 1., 1.]),
  array([0., 0., 1., 1., 1.]),
  array([0., 1., 0., 0., 1.]),
  array([1., 0., 1., 1., 0.])],
 ['天气', '温度', '湿度', '风速'])
```

```
# 计算给定数据集合的香农熵
def calcShannonEnt(dataset):
```

```
for key in labelCounts:
    prob = float(labelCounts[key]) / numEntries #使用频率pi
    shannonEnt -= prob * log(prob, 2)
```

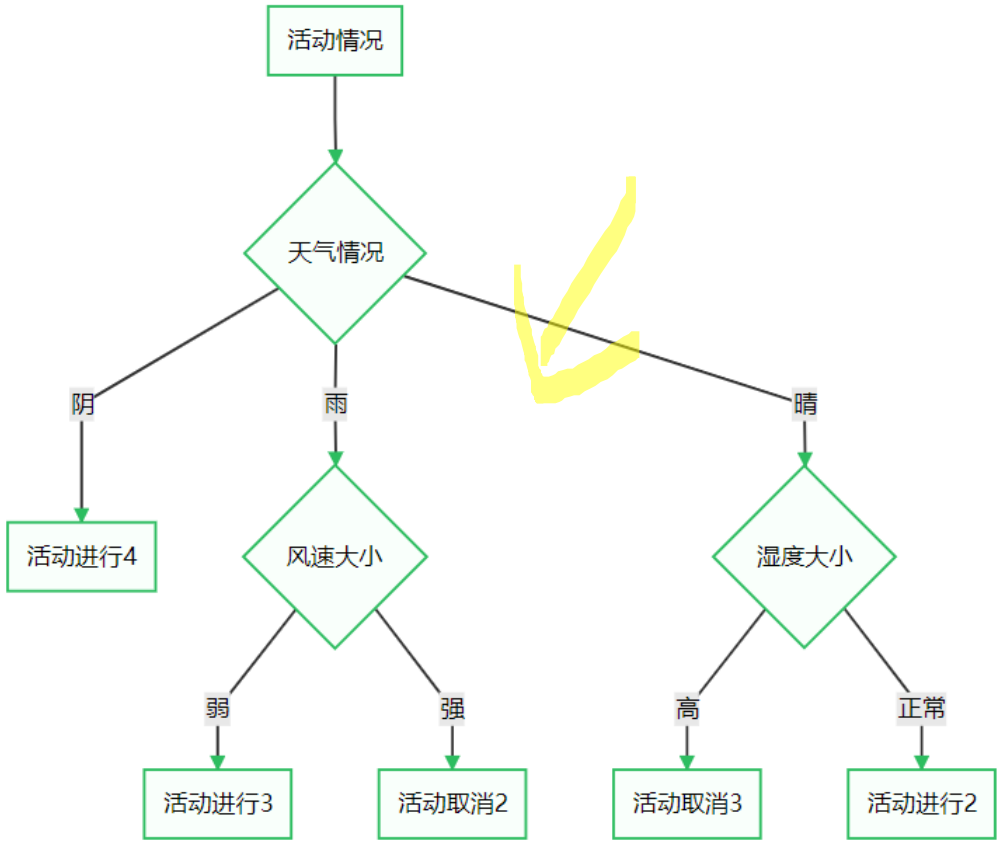
```
#按照给定特征划分数据集
def splitDataSet(dataSet, axis, value):
    return retDataSet
splitDataSet(dataSet,0, 1)
```

```
[[0.0, 1.0, 0.0, 1.0],
 [2.0, 0.0, 0.0, 1.0],
 [2.0, 0.0, 1.0, 0.0],
 [0.0, 0.0, 0.0, 1.0],
 [0.0, 1.0, 1.0, 0.0]]
```

```
#选择最好的数据集划分方式
def chooseBestFeatureToSplit(dataSet):
```

代码框架：加亿点点细节...

```
def createTree(dataSet, labels):
    for value in uniqueVals:
        subLabels = labels[:]
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), subLabels)
    return myTree
```



Result：嵌套字典

{'天气'：{0.0: 1.0，1.0：
{'风速'：{0.0: 1.0，
1.0: 0.0}}，
2.0：{'湿度'：{0.0:1.0，
1.0: 0.0}}}

决策树的三种算法

ID3

- 信息增益
- 在相同条件下，取值比较多的特征比取值少的特征信息增益大

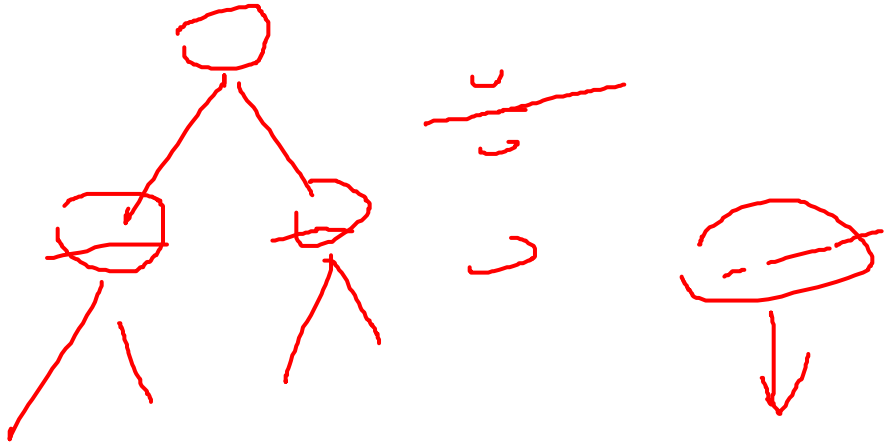
C4.5

- 信息增益率
- 只能处理分类问题

在信息增益的基础上增加了一个惩罚参数
抵消这个特征的类别太多带来的影响

CART(classification and regression tree)

- gini系数、平方误差
- 生成二叉树，可处理回归问题



Sklearn-tree

sklearn中决策树的类都在“tree”这个模块之下，这个模块总共包含五个类

Tree.DecisionTreeClassifier	分类树
Tree.DecisionTreeRegressor	回归树
Tree.export_graphviz	将生成的决策树导出为DOT格式，可视化

```
DecisionTreeClassifier(criterion="gini",
                       splitter="best",
                       max_depth=None,
                       min_samples_split=2,
                       min_samples_leaf=1,
                       min_weight_fraction_leaf=0.,
                       max_features=None,
                       random_state=None,
                       max_leaf_nodes=None,
                       min_impurity_decrease=0.,
                       min_impurity_split=None,
                       class_weight=None,
                       presort=False)
```

七个参数：

- Criterion
- 两个随机性相关的参数（random_state，splitter）
- 四个剪枝参数（max_depth，，min_sample_leaf，max_feature，min_impurity_decrease）

四个接口：fit，score，apply，predict

Sklearn-tree

输入的特征矩阵必须至少是一个二维矩阵
如果数据的确只有一个特征，那必须用reshape(-1,1)来给矩阵增维；
如果数据只有一个特征和一个样本，使用reshape(1,-1)来给你的数据增维

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

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

```
filename = 'data中文.csv'
df = pd.DataFrame(pd.read_csv(filename))
le = LabelEncoder()
for col in df.columns:
    df[col] = le.fit_transform(df[col])
print(df)
attr_data=df[['天气','温度','湿度','风速']]
result_mat=df['活动']
attr_names=['天气','温度','湿度','风速']
```

构造决策树

```
clf = tree.DecisionTreeClassifier(criterion='entropy')
clf.fit(attr_data, result_mat)
print(clf)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Sklearn决策树实例

```
from sklearn.tree import export_text
r = export_text(clf, feature_names=attr_names)
print(r)
```

```
|--- 湿度 <= 0.50
|   |--- 天气 <= 1.50
|   |   |--- class: 1
|   |--- 天气 > 1.50
|   |   |--- 风速 <= 0.50
|   |   |   |--- class: 1
|   |   |--- 风速 > 0.50
|   |   |   |--- class: 0
|--- 湿度 > 0.50
|   |--- 天气 <= 0.50
|   |   |--- class: 0
|   |--- 天气 > 0.50
|   |   |--- 天气 <= 1.50
|   |   |   |--- class: 1
|   |   |--- 天气 > 1.50
|   |   |   |--- 风速 <= 0.50
|   |   |   |   |--- class: 1
|   |   |   |--- 风速 > 0.50
|   |   |   |   |--- class: 0
```

```
clf = tree.DecisionTreeClassifier(max_depth=2)
clf.fit(attr_data[:10],result_mat[:10])
print(clf)
```

```
from sklearn.tree import export_text
r = export_text(clf, feature_names=attr_names)
print(r)
```

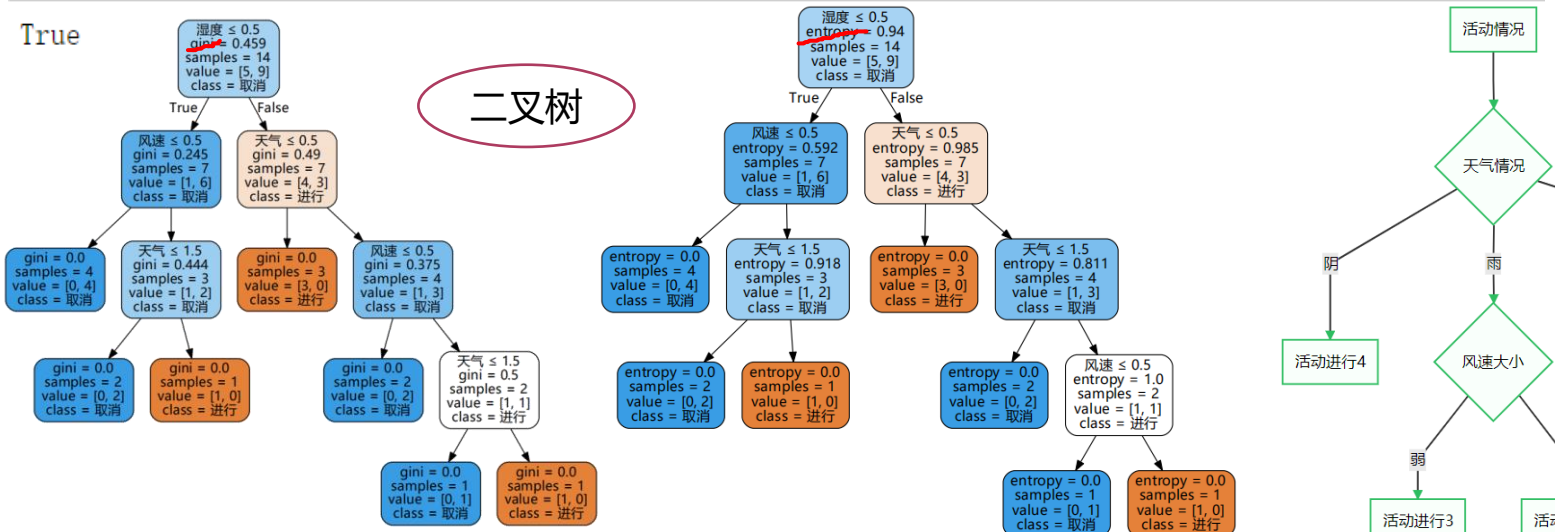
```
|--- 天气 <= 0.50
|   |--- 湿度 <= 0.50
|   |   |--- class: 1
|   |--- 湿度 > 0.50
|   |   |--- class: 0
|--- 天气 > 0.50
|   |--- 风速 <= 0.50
|   |   |--- class: 1
|   |--- 风速 > 0.50
|   |   |--- class: 0
```

Sklearn决策树实例

```
import pydotplus

dot_data = StringIO()
target_name=['进行','取消']
tree.export_graphviz(clf,feature_names=attr_names,
                     class_names=target_name,filled=True,rounded=True,
                     special_characters=True,out_file=dot_data)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue().replace('helvetica','"Microsoft YaHei"'))
graph.write_pdf('golf_tree_entropy.pdf')
```

graphviz需要额外下载可执行文件，并配置环境变量



解决可视化中文乱码的问题：
https://blog.csdn.net/qq_39386012/article/details/83857609#commentBox

模型评价

Confusion Matrix		真实值	
		P	N
预测值	P'	TP	FP
	N'	FN	TN

	公式	意义
准确率 ACC	$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$	分类模型所有判断正确的结果占总观测值的比重
精确率 PPV	$\text{Precision} = \frac{TP}{TP + FP}$	在模型预测是Positive的所有结果中，模型预测对的比重
灵敏度 TPR	$\text{Sensitivity} = \text{Recall} = \frac{TP}{TP + FN}$	在真实值是Positive的所有结果中，模型预测对的比重
特异度 TNR	$\text{Specificity} = \frac{TN}{TN + FP}$	在真实值是Negative的所有结果中，模型预测对的比重

模型评价

$$\text{F1 Score} = \frac{2PR}{P+R}$$

P代表Precision，R代表Recall

F1-Score的取值范围从0到1的，1代表模型的输出最好，
0代表模型的输出结果最差。

混淆矩阵		真实值		
		猫	狗	猪
预测值	猫	10	1	2
	狗	3	15	4
	猪	5	6	20

混淆矩阵		真实值	
		猫	不是猫
预测值	猫	10	3
	不是猫	8	45

Sklearn决策树实例

```
pre = clf.predict(attr_data[11:])
print(pre)
import sklearn.metrics as metrics
print(metrics.classification_report(result_mat[11:],pre))
```

```
[1 1 0]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         2

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3
```

```
score=decision_tree.score(X_test,y_test)
score#返回的是平均准确度，即accuracy
```

```
0.9777777777777777
```

课堂练习

导入鸢尾花数据集
划分训练集和测试集
绘制决策树模型（CART）
评估模型好坏

```
|--- petal width (cm) <= 0.80
|    |--- class: 0
|--- petal width (cm) >  0.80
|    |--- petal width (cm) <= 1.75
|    |    |--- class: 1
|    |--- petal width (cm) >  1.75
|    |    |--- class: 2
```

【模型预测值】

```
[2 1 0 1 2 1 2 2 2 1 1 0 1 0 0 1 0 1 0 1 2 0 0 0 1 0 2 2 0 0 0 0 0 1 1 2 0
0 1 0 2 1 1 0 0]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	20
1	0.80	1.00	0.89	12
2	1.00	0.77	0.87	13
accuracy			0.93	45
macro avg	0.93	0.92	0.92	45
weighted avg	0.95	0.93	0.93	45

过拟合的处理方法——剪枝

过拟合：在训练集上表现很好，在测试集上却表现糟糕

预先剪枝指在决策树生长过程中，使用一定条件加以限制，使得产生完全拟合的决策树之前就停止生长。

- 限制树的最大深度，超过设定深度的树枝全部剪掉
- 限制一个节点在分枝后的每个子节点都必须包含的训练样本数量，少于一定数量停止分支
- 限制分支时考虑的特征个数

后剪枝是在决策树生长完成之后，按照自底向上的方式修剪决策树

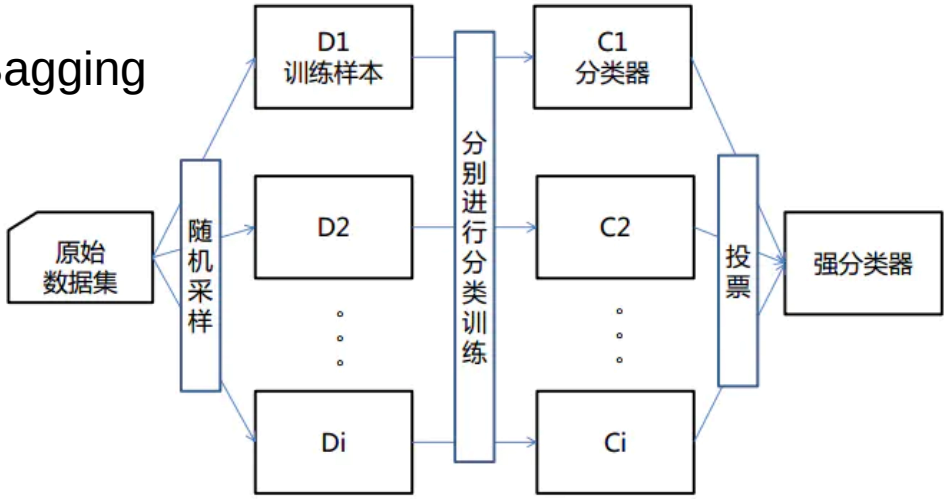
- 用新的叶子节点替换子树，该节点的预测类由子树数据集中的多数类决定
- 用子树中最常使用的分支代替子树

集成算法-随机森林、GBDT

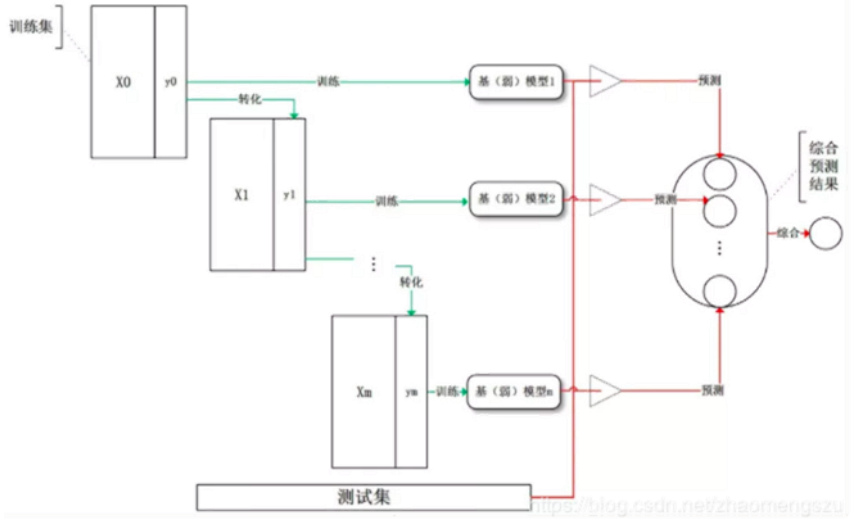
常见的集成学习框架有三种：Bagging、Boosting 和 Stacking

Bagging	对训练集进行有放回抽样得到子训练集，对不同子训练集进行训练得到很多基学习器，最后综合所有基学习器的预测值得到最终的预测结果。Bagging 常用的综合方法是投票法，票数最多的类别为预测类别	**随机森林（**Random Forest**）** • 随机选择样本（放回抽样）； • **随机选择特征；** • 构建决策树； • 随机森林投票（平均）
Boosting	Boosting 训练过程为阶梯状，基模型的训练是有顺序的，每个基模型都会在前一个基模型学习的基础上进行学习，最终综合所有基模型的预测值产生最终的预测结果，用的比较多的综合方式为加权法	GBDT（Gradient Boosting Decision Tree）通过多轮迭代,每轮迭代产生一个弱分类器（CART TREE），每个分类器在上一轮分类器的**梯度（残差值）**基础上进行训练

Bagging



Boosting



随机森林

```
#随机森林
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(random_state=0)#这里使用了默认的参数设置
iris_rf=rf.fit(X_train, y_train)#进行模型的训练
pre_rf = iris_rf.predict(X_test)
print('【随机森林预测值】\n',pre_rf)
score=iris_rf.score(X_test,y_test)
print(score)
```

```
【随机森林预测值】
 [2 2 1 2 0 2 1 1 0 1 2 0 0 2 0 0 0 0 0 2 0 1 1 2 2 1 1 2 1 1 2 1 2 2 2 1 1
 1 1 1 1 1 1 1 0]
0.9555555555555556
```

```
iris_rf=rf.fit(X_train, y_train)#进行模型的训练
score=iris_rf.score(X_test,y_test)
score
```

```
0.9555555555555556
```

随机森林

```
#随机森林
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()#这里使用了默认的参数设置
iris_rf=rf.fit(X_train, y_train)#进行模型的训练
pre_rf = iris_rf.predict(X_test)
print('【随机森林预测值】\n',pre_rf)
score=iris_rf.score(X_test,y_test)
print(score)
```

```
【随机森林预测值】
 [2 2 1 2 0 2 1 1 0 2 2 0 0 2 0 0 0 0 0 2 0 1 1 2 2 1 1 2 1 1 2 1 2 2 2 1 1
 1 1 1 1 1 1 1 0]
0.9777777777777777
```

```
iris_rf=rf.fit(X_train, y_train)#进行模型的训练
score=iris_rf.score(X_test,y_test)
score
```

```
1.0
```

GBDT

```
from sklearn import ensemble
clf = ensemble.GradientBoostingClassifier()
gbdt_iris = clf.fit(X_train, y_train)  # Training model
pre_GBDT = gbdt_iris.predict(X_test)
print('【GBDT预测值】\n',pre_GBDT)
score=gbdt_iris.score(X_test,y_test)
print(score)
```

```
【GBDT预测值】
[2 2 1 2 0 2 1 1 0 2 2 0 0 2 0 0 0 0 0 2 0 1 2 2 2 1 1 2 1 1 2 1 2 2 2 1 1
 1 1 1 1 1 1 1 0]
0.9555555555555556
```

模型评价

模型优势：

- 非常直观，易于实现和理解
- 数据的准备工作比较简单，因为其可以处理多种类型数据（连续性和离散型），对于部分数据的缺失和错误不敏感。
- 计算复杂度相对较低，能够在较短时间内对大量数据做出非常好的结果

什么时候用比较好？

- 实例是由“属性-值”对表示的；
- 目标函数具有离散的输出值；
- 该问题是非线性问题

什么时候表现差？

- 决策树匹配的数据过多时；
- 分类的类别过于复杂；
- 数据的属性之间具有非常强的关联。


Thanks