同濟大學
TONGJI UNIVERSITY

王睿智 ruizhiwang@tongji.edu.cn

人工智能技术与应用

第三章 监督学习

3.1 k近邻分类

3.2 回归分析

3.3 Logistic回归

3.4 支持向量机

3.5 决策树

3.6 集成学习

- 3.6.1 集成学习
- 3.6.2 Bagging
- 3.6.3 随机森林
- 3.6.4 Boosting
- 3.6.5 硬投票与软投票
- 3.6.6 处理不平衡数据集

回顾决策树

- **决策树非常容易过拟合，导致对新数据的泛化能力差**
  - 拟合局部数据，而没有对整个数据分布的大局观。
  - 换一角度，可认为模型训练的是数据的一个子集。若用不同的训练子集，训练出多个**不同的树**，聚合多个不同决策树的预测，泛化性能会提高吗?

- 通过组合多个学习器来提升泛化性能的技术称为**集成学习**（ensemble learning）。

3.6.1 集成学习 | 影响集成效果的因素

在两类问题中，假定训练了3个分类器（C_1,C_2, C_3），在三个测试样本中的表现如下图所示，其中√ 表示分类正确，×号表示分类错误，集成的结果通过多数投票机制产生。

	测试例1	测试例2	测试例3
C_1	√	√	×
C_2	×	√	√
C_3	√	×	√
集成	√	√	√

(a) 集成提升性能

3个66.6%精度的不同分类器

	测试例1	测试例2	测试例3
C_1	√	√	×
C_2	√	√	×
C_3	√	√	×
集成	√	√	×

(b) 集成不起作用

3个分类器没差别

	测试例1	测试例2	测试例3
C_1	√	×	×
C_2	×	√	×
C_3	×	×	√
集成	×	×	×

(c) 集成起负作用

3个33.3%精度的不同分类器

表明：集成个体好而不同，效果好。

3.6.1 集成学习 | 影响集成效果的因素

1. 好而不同：

- **好** ：一定的“准确性”，即个体学习器不能太坏。至少是泛化性能略高于随机猜测的**弱学习器**。
- **不同**：要有“多样性”，即个体学习器间要有差异，彼此错误不相关。当各学习器尽可能相互独立时，集成的效果最优。

2. 多样性：

- 使用不同的学习算法（模型）
- 在不同的训练集随机子集或特征子集上训练

这会增加它们犯不同类型错误的机会，从而提升集成的准确率。

可以是训练样本的随机子集，也可以是特征集合的随机子集，或两者都进行。

3.6.1 集成学习 | 集成学习分类

- 关键问题：如何生成个体学习器？怎样对它们进行集成？
- 根据个体学习器的生成方式，**集成学习大致分为两类**：

① **个体间不存在强依赖关系，可同时生成的并行化方法。**

bagging法： 多个个体学习器独立进行学习的方法 。

典型算法，如 随机森林

② **个体间存在强依赖关系，必须串行生成的序列化方法。**

boosting法： 多个个体学习器（常为弱学习器）**依次进行学习**的方法。

典型算法，如 AdaBoost、梯度boosting中的XGBoost

3.6.2 bagging | bagging的基本思想

bagging（**b**ootstrap **agg**regat**ing**） bootstrap聚合

并行

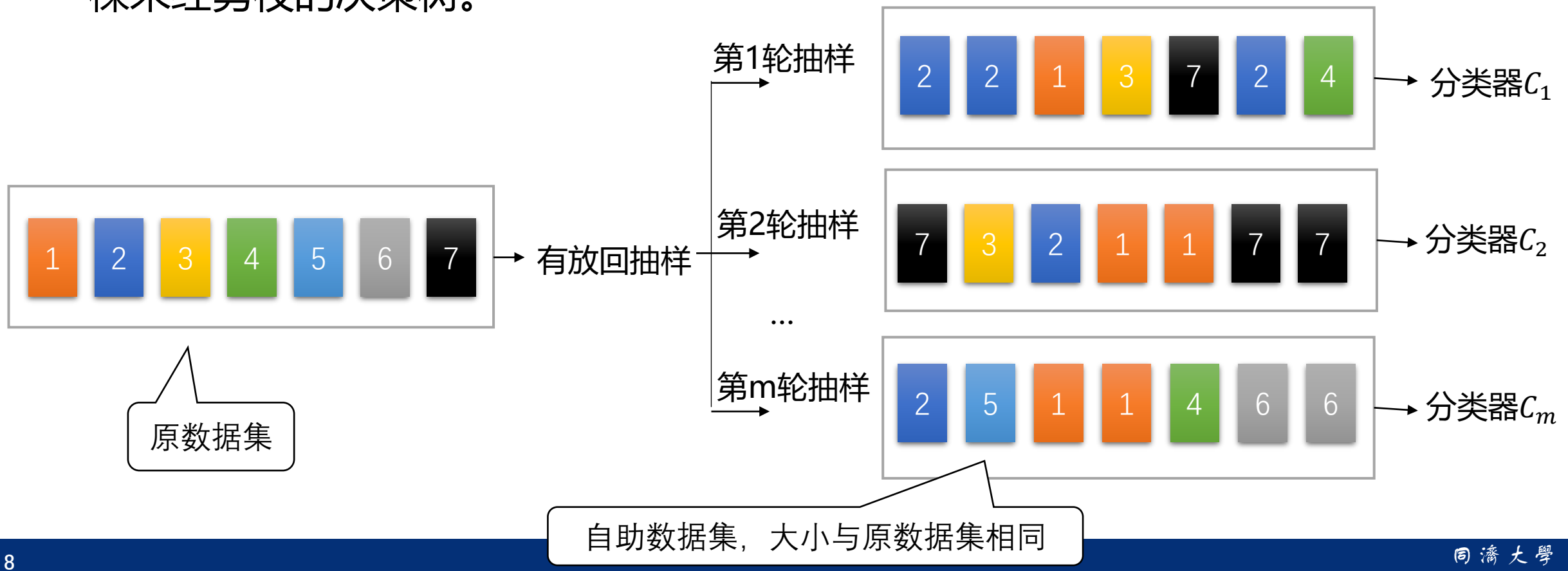
基本思想:

为获得差异尽可能大的个体学习器（**一般是同种类型**），对训练集随机采样得到不同的训练子集，分别训练出不同的个体学习器。然后，**聚合**所有个体学习器的预测，对新样本做出预测。分类问题用**投票法**聚合，回归问题用**平均法**来聚合。

随机采样时将样本放回，称为自助采样（bootstrap）。基于自助采样的聚合集成，称为bagging。若采样时样本不放回，则称为pasting。

bagging 示例

有7个不同的训练样本(索引1~7)。在每轮bagging抽样中有放回地**随机**抽取样本。然后用每个bootstrap样本集拟合一个分类器——通常是一棵未经剪枝的决策树。



3.6.2 bagging | 自助采样

问题：1000个样本的数据集，用自助法采样1000次，原数据集中约有多少样本出现在采样集中?

回答：大约有630个

假设初始数据集D中有 n 个样本，用**自助法**采样 n 次得采样集 D_i 。

D中有的样本在采样集D_i里多次出现，有的则从未出现。一个样本在 n 次采样中始终不被采到的概率为 $(1-\frac{1}{n})^n$, 且

$$\lim_{n\to\infty}(1-\frac{1}{n})^n=\frac{1}{e}\approx 0.368$$

多样性!

因此，**初始数据集D中约有63.2%的样本出现在采样集D_i中。**

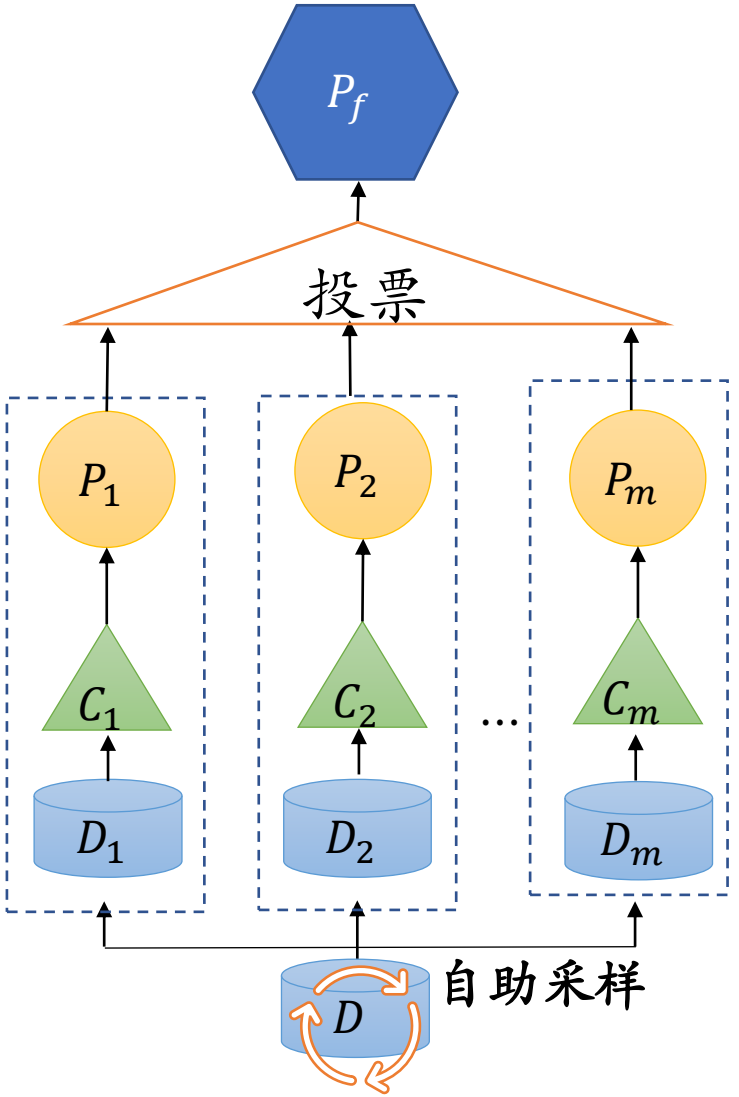
3.6.2 bagging | 基本流程

bagging的基本流程:

首先，对原训练集用bootstrap，获得m个自助数据集。

然后，用每个自助数据集D_i独立地训练个体学习器C_i。

最后，将新数据输入每个个体学习器，做出预测。最终结果由多数投票（分类）或取平均（回归）决定。



bagging在训练集的不同随机样本上并行训练多个学习器

说明：预先确定待训练的个体学习器数目，如 m 个

优点：所有个体学习器的训练和预测可以并行!

3.6.2 bagging | 特点

bagging特点：

- 个体学习器是同类型的
- 个体学习器不存在强依赖关系
- 并行化生成
- 自助采样法
- 算法时间复杂度低

  训练一个bagging集成，与直接使用个体学习器的复杂度同阶。
- 从偏差-方差分解的角度看，bagging主要关注**降低方差**。

  它在未剪枝的决策树、神经网络等易受样本扰动的学习器上效果更明显。

3.6.2 bagging | sklearn中的bagging

Sklearn.ensemble提供了**BaggingClassifier** 元估计器，可进行bagging集成分类。也提供了BaggingRegressor用于回归。

```
BaggingClassifier(estimator=None, n_estimators=10,
                  max_samples=1.0,max_features=1.0,
                  n_jobs=None,random_state=None,…)
```

参数:

- *estimator*：个体学习器类型。默认None，此时个体学习器为**决策树**。默认未剪枝
- *n_estimators*：个人学习器数目，默认10个。
- *max_samples*：从X中抽取出多少样本用于训练每个个体分类器。若int，则是抽取的样本数目；若float，则抽取“max_samples * X.shape[0]”个样本。
- *max_features*：从X中抽取出多少特征用于训练每个个体分类器。
- *n_jobs*: 用多少CPU内核进行训练和预测。默认为None,即1。-1表示使用所有可用内核。

例3.19 红酒分类

（1）下载并读取UCI红酒数据 https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data

（2）创建一个单棵决策树（不剪枝）；再创建一个包含500个决策树分类器的bagging集成分类器，每次随机自助采样80%训练样本，用所有CPU内核。

（3）在红酒数据上分别训练集成分类器、单棵决策树，比较两模型分类性能。

```
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Class label                   178 non-null    int64
 1   Alcohol                       178 non-null    float64
 2   Malic acid                    178 non-null    float64
 3   Ash                           178 non-null    float64
 4   Alcalinity of ash             178 non-null    float64
 5   Magesium                      178 non-null    int64
 6   Total phenols                 178 non-null    float64
 7   Flavanoids                    178 non-null    float64
 8   Nonflavanoid phenols          178 non-null    float64
 9   Proanthocyanins               178 non-null    float64
 10  Color intensity               178 non-null    float64
 11  Hue                           178 non-null    float64
 12  OD280/OD315 of diluted wines  178 non-null    float64
 13  Proline                       178 non-null    int64
```

例3.19 红酒分类 | 代码

(1) 下载并读取UCI红酒数据

```
import pandas as pd

df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data', header = None)
df_wine.columns = ['Class label','Alcohol',
                   'Malic acid','Ash','Alcalinity of ash',
                  'Magesium','Total phenols','Flavanoids',
                  'Nonflavanoid phenols','Proanthocyanins',
                  'Color intensity','Hue',
                  'OD280/OD315 of diluted wines','Proline']
```

```
# 拆分数据
from sklearn.model_selection import train_test_split
X = df_wine.drop('Class label',axis=1).values
y = df_wine['Class label'].values
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,
                                                 stratify=y,
                                                 random_state = 1)
```

（2）分别创建单棵决策树、bagging集成分类器

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier

tree = DecisionTreeClassifier(criterion = 'entropy',
                              random_state = 1,
                              max_depth = None)
bag_clf = BaggingClassifier(estimator=tree,
                            n_estimators = 500,
                            max_samples = 0.8,
                            n_jobs  = -1)
```

（3）训练并评估两个模型

```
from sklearn.metrics import accuracy_score
bag_clf.fit(X_train,y_train)
tree.fit(X_train,y_train)
y_tree_pred = tree.predict(X_test)
y_bag_pred = bag_clf.predict(X_test)
print('决策树训练准确率',tree.score(X_train,y_train))
print('决策树测试准确率',accuracy_score(y_test,y_tree_pred))
print('bagging训练准确率',bag_clf.score(X_train,y_train))
print('bagging测试准确率',accuracy_score(y_test,y_bag_pred))
```

```
决策树训练准确率 1.0
决策树测试准确率 0.8888888888888888
bagging训练准确率 1.0
bagging测试准确率 0.9722222222222222
```

3.6.3 随机森林

随机森林 (Random Forest, 简称**RF**) 是个体学习器都为**决策树**的特殊bagging算法，其特殊之处在于拟合单棵树时也使用了特征集合的随机子集。

其算法有四步：

1. 使用**自助法**从训练数据集抽取n个样本。
2. 用上述抽取的样本训练一棵决策树。在每个节点上：
   a. 不放回地**随机选取d个特征**。Scikit-learn中$d = \sqrt{m}$，m是特征总数
   b. 根据目标函数的要求，例如最大信息增益，选择最佳特征分裂节点。
3. 重复步骤1和步骤2 k次。
4. 给定一样本，收集每棵树对该样本的预测标签，**投票**决定最终预测标签。

不需要所有特征来确定一个节点上的最佳划分特征。

随机选d个特征

随机抽n个样本

3.6.3 随机森林| 随机森林特点

随机森林特点：

- 随机采样样本和随机采样特征，得到多个不相关的单个决策树。
- 简单、容易实现、计算开销小（RF的训练效率优于bagging）。
- 在很多现实任务中展现出强大的性能，被誉为“代表集成学习技术水平的方法”。
- RF的收敛性与bagging相似。而随着个体学习器数目的增加，随机森林通常会收敛更低的泛化误差。

3.6.3 随机森林 | Scikit-learn中的随机森林分类器

sklearn.ensemble中提供了RandomForestClassifier估计器，是随机森林分类器。也提供了RandomForestRegressor用于回归。

默认未剪枝

```
RandomForestClassifier(n_estimators=100, max_depth=None,
                       max_features='sqrt', max_samples=None,
                       min_samples_leaf=1, n_jobs=None,
                       random_state=None, …)
```

参数:

- *n_estimators* ： 森林中决策树的个数。int，默认值100。通常树越多性能越好，但计算成本也会增加。
- max_depth：树深度。默认None，表示不限制树的生长。
- *max_features*：在每个节点上随机采样的特征数量。默认为sqrt(n_features)。
- *max_samples*：若bootstrap为*True*,每个学习器训练时自助采样集的大小。默认None，表示采样集大小为X.shape[0]。
- *min_samples_leaf*：叶节点上允许的最小样本数。

属性： feature_importances_ 基于不纯度的特征重要性。

例3.19 红酒分类（续1）

（4）创建一个随机森林分类器，集成500棵树，令随机自助采样比为0.8，利用所有CPU内核来训练。评估其分类性能。

```
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(n_estimators=500,
           max_samples = 0.8, n_jobs  = -1, random_state=1)

rf_clf.fit(X_train,y_train)
y_rf_pred = rf_clf.predict(X_test)
print('RF训练准确率',rf_clf.score(X_train,y_train))
print('RF测试准确率',accuracy_score(y_test,y_rf_pred))
```

```
RF训练准确率 1.0
RF测试准确率 1.0
```

3.6.4 boosting

提升法（**boosting**）是一种**依次**训练多个学习器的集成学习技术，每个学习器都试图纠正前序学习器中的错误。

- 与bagging一样，boosting可用于任何监督学习算法。但在**使用弱学习器作为个体学习器时，boosting最有效**。**弱学习器**是指比随机猜测略好的预测模型，如**决策树桩**（是只有一个划分的决策树）。
- 根据纠正前序模型错误方式的不同，boosting分两类：
  - AdaBoost（Adaptive Boosting的简称）
  - 梯度boosting（Gradient Boosting）
- 从偏差-方差分解的角度看，boosting主要关注降低偏差。

3.6.4 boosting | AdaBoost的工作原理

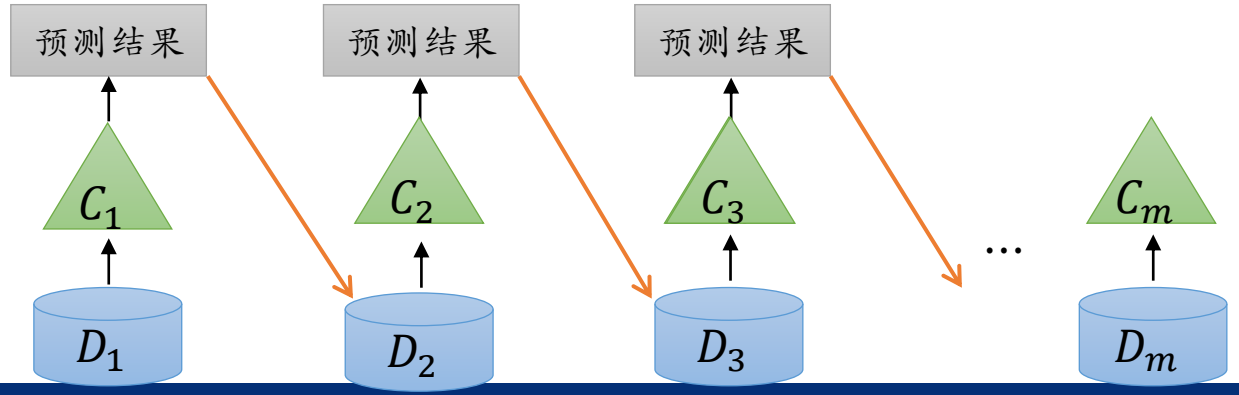
AdaBoost思想：新学习器对其前序学习器的纠正是通过更多地关注前序学习器欠拟合的训练实例来实现的。

1. 先从原始训练集D_1训练出一个基学习器C_1，并使用该基学习器对训练集进行预测。

2. 增加预测错误的训练实例的相对权重，使得先前基学习器做错的训练样本在后续受到更多关注。

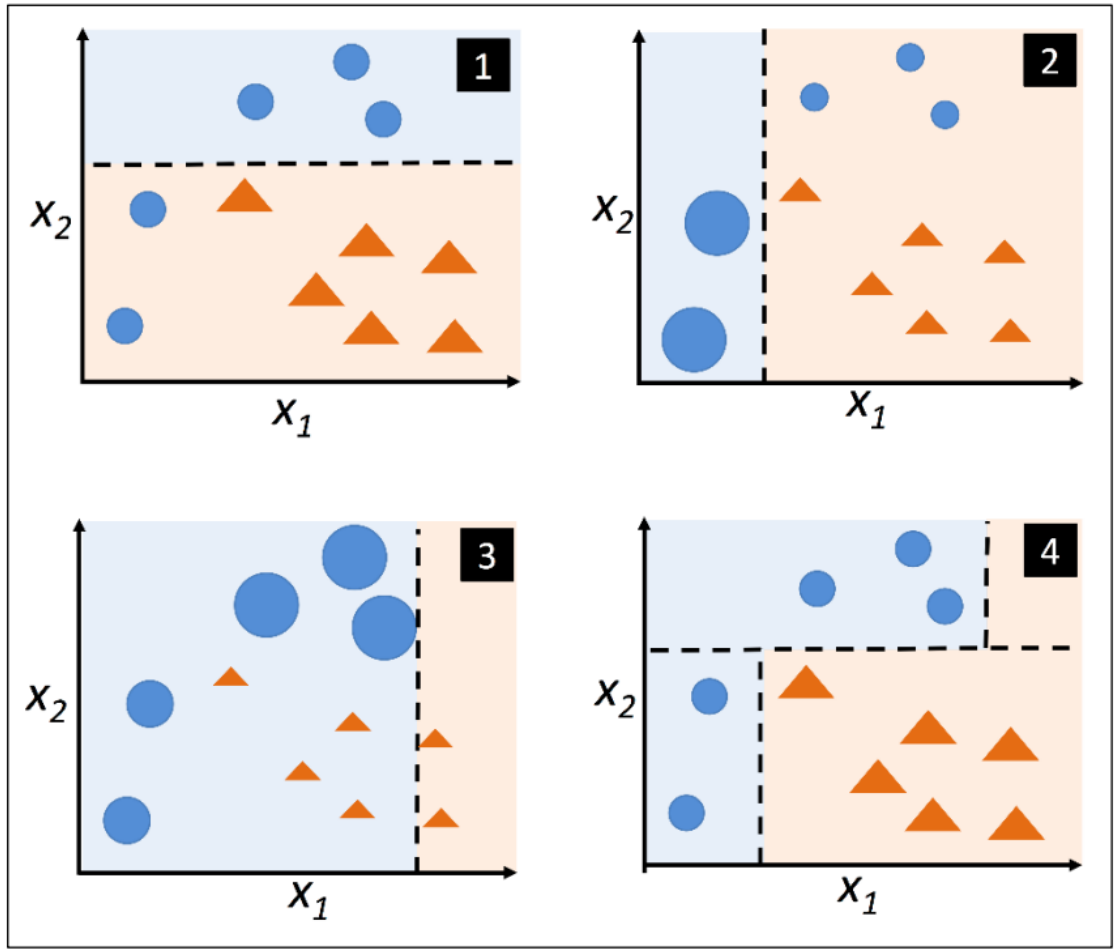
3. 基于调整后的样本分布D_2（同D_1但调整了权重）来训练下一个基学习器C_2；

如此重复进行，直到基学习器数目达到事先指定的值m，最终将**这m个基学习器进行加权组合**。



缺点：无法并行

AdaBoost 示例



一个两类数据集

1. 从图1开始，所有训练样本具有相同的权重。用该训练集训练**决策树桩**(决策边界为虚线)，来对两类样本分类。有两个圆形样本被分错。
2. 在图2中，对分错的两个圆形样本分配更大的权重，并降低正确分类样本的权重。下一个决策树桩将更关注具有较大权重的训练样本，即难分的训练样本。图2训得的弱分类器（图2中虚线）把三个圆形样本分错。
3. 随后这三个样本被赋予较大的权重，如图3，再训一个决策树桩（图3中虚线）。
4. 假设这里的AdaBoost仅含三轮的boosting。最后，通过加权多数投票机制，将三个在不同权重的训练子集上训得的弱学习器组合起来。如图4所示。

3.6.4 boosting | Scikit-learn中的AdaBoost分类器

sklearn.ensemble中提供了多种用于分类和回归任务的boosting元估计器，

包括：AdaBoostClassifier、AdaBoostRegressor、GradientBoostingClassifier、GradientBoostingRegressor。

```
AdaBoostClassifier(estimator=None, n_estimators=50,
                   learning_rate=1.0, random_state=None)
```

参数：

- estimator：基学习器。默认值的基学习器是用max_depth=1初始的决策树分类器。
- n_estimator：终止提升的最大学习器个数。int，默认50个。
- learning_rate ：学习率，用于调整每个分类器的贡献。float,取值[0.0,inf],默认值1。

基学习器也可以是其他类型

```
# 创建一个基分类器是决策树的AdaBoost分类器clf，提升到有50个分类器为止
from sklearn.ensemble import AdaBoostClassifier

clf=AdaBoostClassifier(n_estimators=50,random_state=1)
```

例 3.19 红酒分类（续2）

（5）创建一个AdaBoost分类器，集成500棵树，令其基分类器为决策树桩。评估其分类性能。

```
from sklearn.ensemble import AdaBoostClassifier
ada_clf=AdaBoostClassifier(n_estimators=500,random_state=42)

ada_clf.fit(X_train,y_train)
y_ada_pred = ada_clf.predict(X_test)
print('AdaBoost树桩训练准确率',ada_clf.score(X_train,y_train))
print('AdaBoost树桩测试准确率',accuracy_score(y_test,y_ada_pred))
```

```
AdaBoost树桩训练准确率 0.978873239436619
AdaBoost树桩测试准确率 0.9166666666666666
```

3.6.4 boosting | 梯度提升法

梯度提升（Gradient Boosting）法

- 依次训练**弱学习器**来组合成一个**强学习器**，是一个序贯过程。
- 与AdaBoost非常相似，只是在纠正前序模型的错误方面不同。**后续模型**不是根据样本的分类准确性对样本进行权重调整，而是**将损失函数梯度下降方向作为优化目标**。
- 梯度boosting法中，树常比**决策树桩**深，最大深度通常为3~6。
- 梯度boosting既可用于分类，也可用于回归。
- 最著名的XGBoost。常用的还有LightGBM和CatBoost。

3.6.4 boosting | sklearn中的梯度提升回归树

sklearn.ensemble中的**GradientBoostingRegressor**是用于回归的GB元估计器。它允许优化任意可微损失函数。GradientBoostingClassifier是用于分类的GB元估计器。

```
GradientBoostingRegressor(loss='squared_error',
    learning_rate=0.1, n_estimators=100, max_depth=3,
    n_iter_no_change=None, random_state=None,…)
```

参数:

- loss：*{'squared_error', 'absolute_error', 'huber', 'quantile'},*默认为*'squared_error'*, 待优化的损失函数。
- learning_rate：*float, default=0.1* 学习率，调整每棵树的贡献。
- n_estimators：*int, default=100* 基回归器个数。较大通常会带来更好的性能。
- max_depth：*int, default=3* 单个回归器的最大深度。它限制了树中节点数。调整此参数以获得最佳性能。
- n_iter_no_change: *int, default=None* 用于决定当验证分数没有提高时是否使用提前停止来终止训练。默认None 禁用提前停止。若设为一整数，当验证分数在之前n_iter_no_change次迭代中都没提高时终止训练。

属性: n_estimators_：*int,*通过提前停止选择的个体学习器数量(若n_iter_no_change 被指定). 否则为n_estimators.

例3.20 气温预测

（1）读取温度数据weather.csv，对分类变量Description进行编码。取其中Temperature_c为目标，其余为特征数据。

注意：个体学习器为决策树时，序数编码效果好于one-hot编码。

单一特征序数编码可用LabelEncoder也可用OrdinalEncoder.

（2）创建一个梯度提升回归器，包含1000个决策树，最大树深为4，学习率为0.1，连续10次没有提升就终止。

在温度数据上训练该回归器，输出其训练得分和测试得分。显示树数量。

例3.20 气温预测 | 代码

（1）读取温度数据

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
df = pd.read_csv('weather.csv')
df['Description']=LabelEncoder().fit_transform(df['Description'])
```

```
# 随机打乱数据
from sklearn.utils import shuffle
df_shuffled = shuffle(df, random_state=42)
X = df_shuffled.drop('Temperature_c', axis=1) # 获取X
y = df_shuffled['Temperature_c'] # 获取y

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                   test_size=0.33, random_state=42)
```

例3.20 气温预测 | 代码

(2) 训练一个梯度提升回归模型

```
from sklearn.ensemble import GradientBoostingRegressor
GBTR = GradientBoostingRegressor(max_depth=4,
                                 learning_rate=0.1,
                                 n_estimators=1000,
                                 n_iter_no_change=10,
                                 random_state=42)
GBTR.fit(X_train, y_train)
```

```
print(f"梯度提升树在训练集得分{GBTR.score(X_train,y_train):0.4f}")
print(f"梯度提升树在测试集得分{GBTR.score(X_test,y_test):0.4f}")
```

```
梯度提升树在训练集得分0.9026
梯度提升树在测试集得分0.8949
```

```
1  GBTR.n_estimators_  # 通过提前停止选择的树数量
```

```
69
```

3.6.4 boosting | XGBoost

- **梯度boosting是一个序贯过程，训练慢。**
- **XGBoost**（Extreme Gradient Boosting）
  - 是一个优化的分布式**梯度提升库**。高效、灵活、可移植。
  - XGBoost 提供了一种**并行树提升**，可以快速准确地解决许多数据科学问题。相同的代码在主要的分布式环境（Hadoop、SGE、MPI）上运行，可以解决超过数十亿个实例的问题。
  - 在完成多任务时性能表现良好，往往优于大多数其他监督学习算法。
  - 可以处理缺失值。从1.5版本开始，XGBoost的Python包支持特征是分类变量。
  - 有适用于Scikit-learn包的接口。https://xgboost.readthedocs.io/en/stable/python/index.html
  - 安装：pip install xgboost

3.6.4 boosting | XGBoost

适用于Scikit-learn的API主要有：

- xgboost.XGBRegressor 元估计器，用于回归。
- xgboost.XGBClassifier 元估计器，用于分类。

https://xgboost.readthedocs.io/en/stable/python/index.html

3.6.4 boosting | XGBoost

xgboost.XGBRegressor 元估计器，用于回归

```
XGBRegressor(n_estimators=100, gamma=0, learning_rate=0.3,
             max_depth=6, random_state=None,…)
```

参数	• *n_estimators：int,* 梯度提升树的数目，也是提升的轮次。默认100。 • *max_depth：int,*基学习器的最大深度。 默认6。 • *gamma:* 节点用来改善损失函数（MSE）的最小划分量。gamma越大，算法越保守，可防止过拟合。 • *learning_rate:*学习率，取值0~1，通常介于0.01~0.1。防止过拟合。较小的值通常更好，但模型训练需要更长时间。默认0.3。 • *booster：* 指定基学习器，默认为'gbtree'，可选'gblinear' (线性回归)，'dart' (带有dropout 的提升树) • *subsample:*每个决策树随机抽样的比例，设置为1即可使用训练集中的所有样本。默认为1。
方法	• <xgb模型名>.fit(X,y,***eval_set=None***) 、 <xgb模型名>.predict(X) 、<xgb模型名>.score(X,y) • <xgb模型名>.save_model(文件名): 将模型保存为json文件。 • <xgb模型名>.load_model(文件名): 从文件载入模型。
属性	• feature_importances_：数组,形状 *[n_features]*。返回特征对决策的相对重要性。

例3.20 气温预测（续）

(3) 重新划分数据集，留出20%做测试，剩余80%中的20%做验证。

(4) 创建一个XGBOOST回归器，包含1000个决策树，最大树深为6，学习率为0.01，gamma为10。

训练并验证该回归器，观察验证损失。输出该模型的训练得分和测试得分。

(5) 用横条图显示各个属性的相对重要度。

(6) 以”weather_xgb.json”保存模型，以备后用。

加载模型，取测试集前4个样本，用该模型预测。

例3.20 气温预测（续） | 代码

（3）重新划分数据集，20%留作测试，剩余80%中的20%做验证

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                   test_size=0.2, random_state=42)
n = int(X_train.shape[0]*0.2) # 设置验证集大小
X_val,y_val = X_train[:n],y_train[:n]
X_train,y_train = X_train[n:],y_train[n:]
```

（4）训练并验证XGBRegressor模型，输出测试集得分

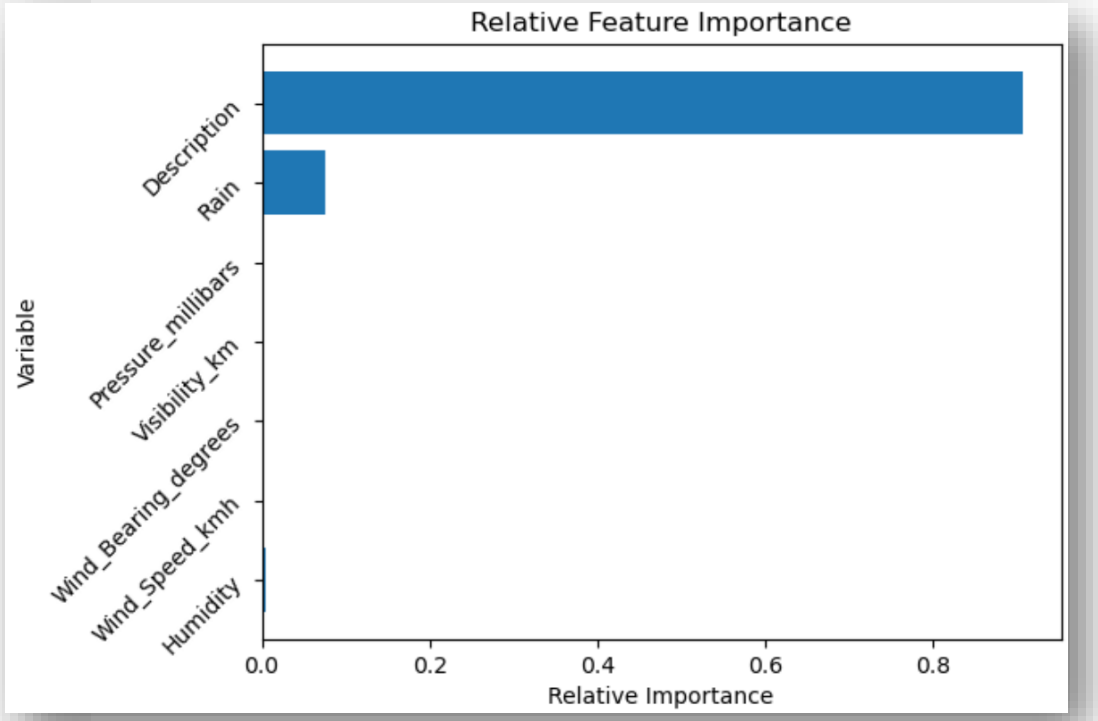
```
import xgboost as xgb

xgbr = xgb.XGBRegressor(n_estimators=1000,
                        learning_rate=0.01,
                        gamma=10,
                        max_depth=6,
                        random_state=42)
xgbr.fit(X_train, y_train,eval_set=[(X_val, y_val)])
```

```
print(f"xgboost在训练集得分{xgbr.score(X_train,y_train):0.4f}")
print(f"xgboost在测试集得分{xgbr.score(X_test,y_test):0.4f}")
```

```
xgboost在训练集得分0.9291
xgboost在测试集得分0.8986
```

(5) 画出属性相对重要度

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

n_features=X_train.shape[1]  # 特征数目
plt.barh(range(n_features),
         xgbr.feature_importances_,
         align='center')
feature_names = X.columns
plt.yticks(np.arange(n_features),
           feature_names,
           rotation=45)
plt.title('Relative Feature Importance')
plt.xlabel('Relative Importance')
plt.ylabel('Variable')
plt.show()
```



(6) 保存模型到文件中

```
xgbr.save_model('weather_xgb.json')
```

加载被保存的模型，来预测数据

```
loaded_model = xgb.XGBRegressor()
loaded_model.load_model('weather_xgb.json')
y_pred = loaded_model.predict(X_test[:4])
print('预测结果:',y_pred)
```

```
预测结果: [11.760563   6.656265  -2.8080163 12.033995 ]
```

3.6.4 boosting | XGBoost

xgboost.XGBClassifier 元估计器，用于分类

```
XGBClassifier(max_depth=6,learning_rate=0.3, gamma=0,
              n_estimators=100,use_label_encoder=False,…)
```

参数	• *n_estimators：int,* 梯度提升树的数目，也是提升的轮次。默认100。 • *max_depth：int,*基学习器的最大深度。 默认6。根据数据集规模，较大的值可能得到较好的效果。 • *gamma:* 节点用来改善损失函数（MSE）的最小划分量。gamma越大，算法越保守，可防止过拟合。 • *booster：* 指定基学习器，默认为'gbtree'，可选'gblinear' (线性回归)，'dart' (带有dropout 的提升树) • *learning_rate:*学习率，取值0~1，通常介于0.01~0.1。防止过拟合。默认0.3。 • *subsample:*每个决策树随机抽样的比例，设置为1即可使用训练集中的所有样本。默认为1。 • *use_label_encoder:* bool, 将弃用，建议设为False。 • *scale_pos_weight* 当类高度不平衡时，很有用。可设为：负实例的和/正实例的和。默认为1 • *eval_metric:* 在验证集上观察模型的性能指标。默认指标是根据目标函数选择（RMSE用于回归，Logloss用于分类）
方法	• <xgb模型名>.fit(X,y,***eval_set=None***) 、 <xgb模型名>.predict(X) 、<xgb模型名>.score(X,y) • <xgb模型名>.save_model(文件名): 将模型保存为json文件。 • <xgb模型名>.load_model(文件名): 从文件载入模型。
属性	• feature_importances_：数组,形状 *[n_features]*。返回特征对决策的相对重要性。

例3.21 蘑菇类别预测

蘑菇数据集

包含8124个样本，分两类(有毒p、无毒e)，有22个特征，
为菌盖颜色、菌盖形状、菌盖表面形状、气味、菌褶等。

下载网址： https://archive.ics.uci.edu/ml/datasets/Mushroom

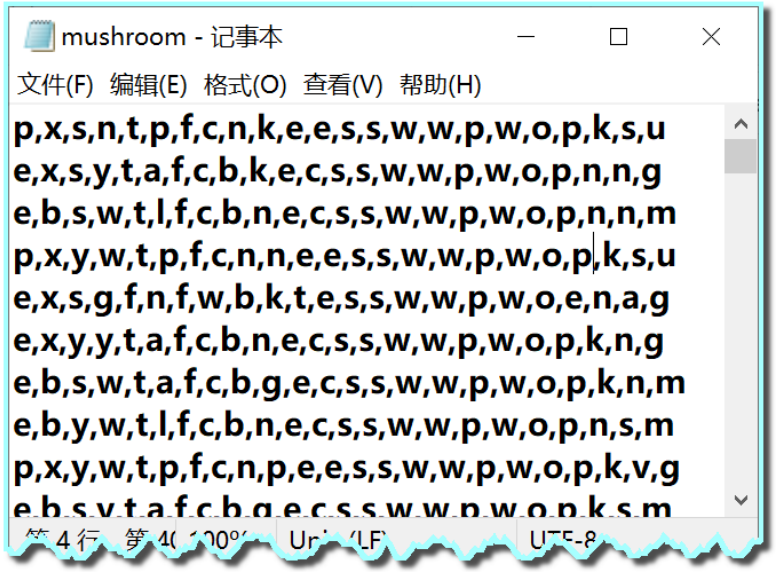
比较RF、Adaboost、xgboost在蘑菇数据集上的表现

（1）用数据集的10%训练并测试RF、Adaboost、xgboost 3个集成模型，分别输出RF和XGBoost的分类性能报告，并画出混淆矩阵。

（2）画出Adaboost的验证曲线。

非数值的、取值为类型的分类变量，怎么处理?

在个体学习器为决策树的情况下，用标签编码



第一步：数据预处理

读入数据文件(csv)后，进行预处理数据。

1）对分类变量分别做编码转换

用sklearn.preprocessing的**OrdinalEncoder**。

2) 拆分成特征和标签。

载入数据

```
mushrooms=pd.read_csv('mushroom.data',header=None)
mushrooms.columns=['class','cap-shape','cap-surface','cap-
color','ruises','odor','gill-attachment','gill-spacing','gill-size','gill-
color','stalk-shape','stalk-root','stalk-surface-above-ring','stalk-surface-
below-ring','stalk-color-above-ring','stalk-color-below-ring','veil-
type','veil-color','ring-number','ring-type','spore-print-
color','population','habitat']
```

数据预处理

```
#序数编码
from sklearn.preprocessing import OrdinalEncoder

enc=OrdinalEncoder()
mushrooms_ = enc.fit_transform(mushrooms)
```

OrdinalEncoder()	用介于0和n_class-1之间的值对数据进行序数编码

```
# 拆分成特征和标签
X=mushrooms_[:,1:]
y=mushrooms_[:,0]
```

```
# 划分数据集，取出10%用于训练和验证模型
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,
                        test_size=0.9,random_state=42)
```

第二步：RF模型训练与评估

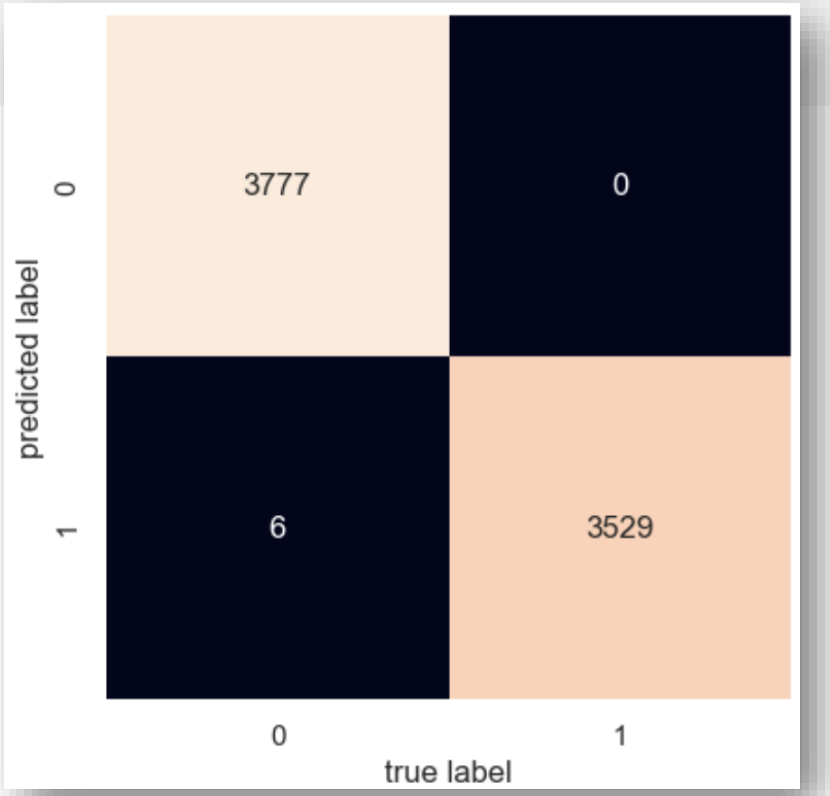
```
from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(n_estimators=10,random_state=1)
RF.fit(X_train,y_train)
```

```
from sklearn import metrics
y_pred=RF.predict(X_test)
print('RF模型在测试集上的分类性能：')
print(metrics.classification_report(y_test,y_pred))
```

RF模型在测试集上的分类性能：

	precision	recall	f1-score	support
0	1.00	1.00	1.00	3777
1	1.00	1.00	1.00	3535
accuracy			1.00	7312
macro avg	1.00	1.00	1.00	7312
weighted avg	1.00	1.00	1.00	7312

```
# 画出RF混淆矩阵
import matplotlib.pyplot as plt
import seaborn as sns;sns.set()
mat = metrics.confusion_matrix(y_test,y_pred)
sns.heatmap(mat,square=True,annot=True,fmt='d',cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label')
```



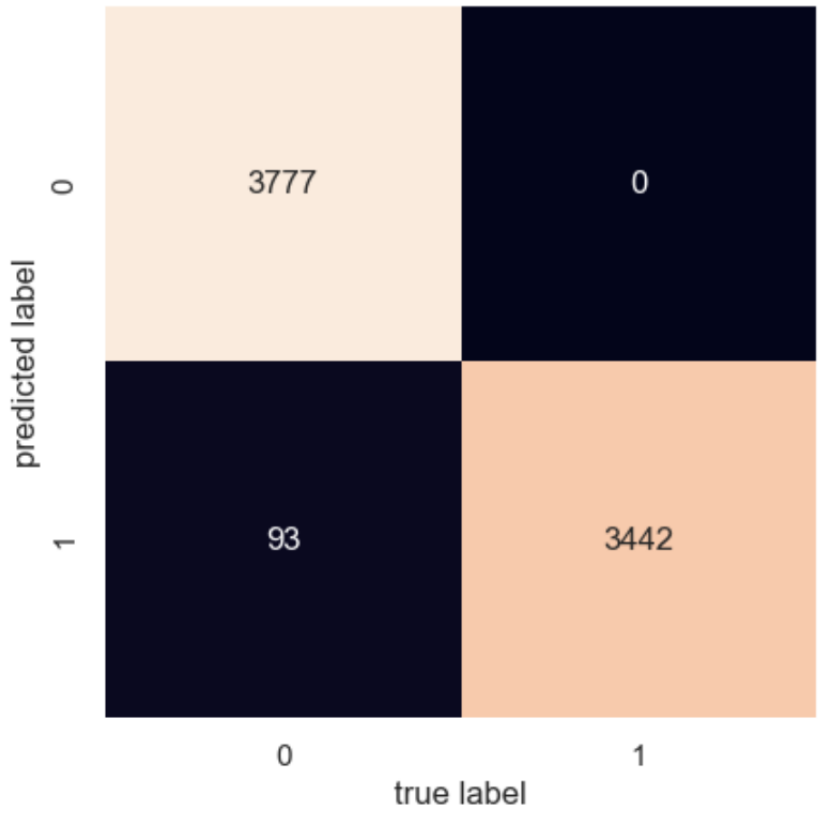
第三步：XGBoost模型训练与评估

```
import xgboost as xgb
xgbc = xgb.XGBClassifier(n_estimators=1000,
                         learning_rate=0.01,
                         use_label_encoder = False,
                         gamma=10,
                         max_depth=4,
                         random_state=42)
xgbc.fit(X_train, y_train)
ypred = xgbc.predict(X_test)
print('XGBoost的分类性能报告\n',metrics.classification_report(y_test,ypred))
```

XGBoost的分类性能报告

	precision	recall	f1-score	support
0	0.98	1.00	0.99	3777
1	1.00	0.97	0.99	3535
accuracy			0.99	7312
macro avg	0.99	0.99	0.99	7312
weighted avg	0.99	0.99	0.99	7312

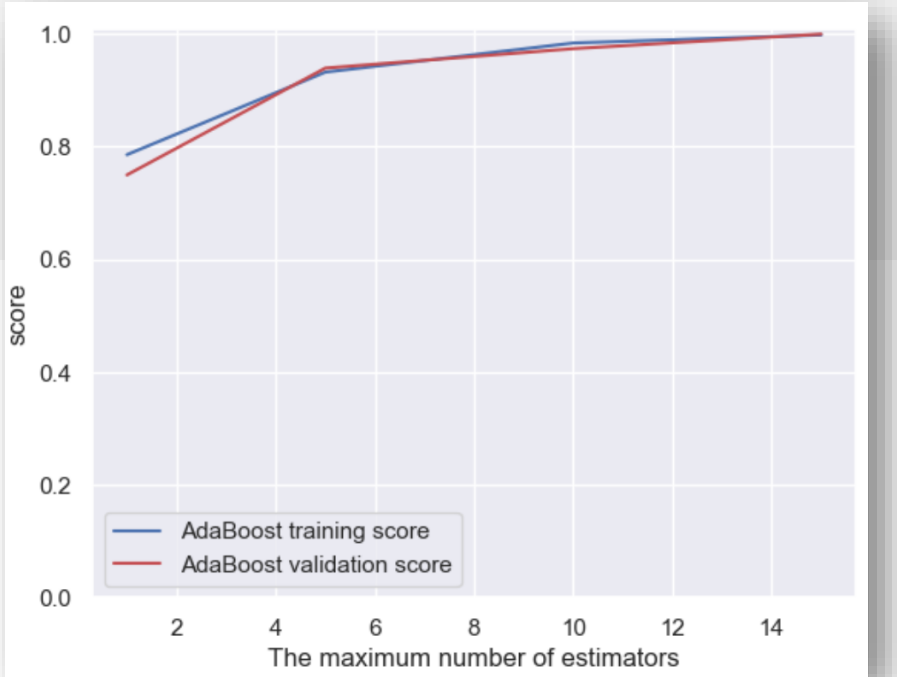
```
# 画出XGBoost混淆矩阵
mat = metrics.confusion_matrix(y_test,ypred)
sns.heatmap(mat,square=True,annot=True,fmt='d',cbar=False)
plt.xlabel('true label')
plt.ylabel('predicted label')
```



第四步：AdaBoost的模型验证

```
from sklearn.model_selection import validation_curve
from sklearn.ensemble import AdaBoostClassifier
import numpy as np

degree = [1,5,10,15] # 基分类器个数
train_score,val_score = validation_curve(AdaBoostClassifier(),X_train,y_train,
                                         param_name='n_estimators',param_range=degree,cv=7)
plt.plot(degree,np.median(train_score,1),'b',label='AdaBoost training score')
plt.plot(degree,np.median(val_score,1),'r',label='AdaBoost validation score')
plt.legend(loc='best')
plt.ylim(0,1.01)
plt.xlabel('The maximum number of estimators')
plt.ylabel('score')
```

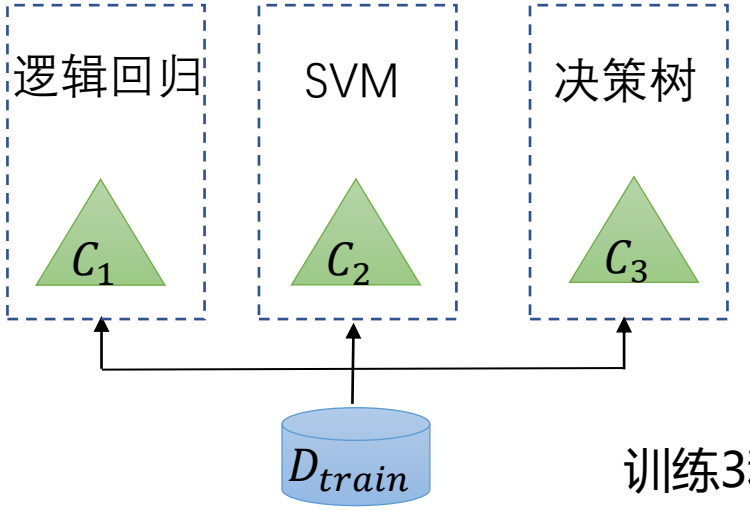
validation_curve(estimator, X, y, param_name,
param_range, groups=None, cv=None,…)
返回train_score 和 test_score



3.6.5 硬投票与软投票

思考：如果想为一个任务训练一个集成分类器，其中各个体分类器是不同质的，如逻辑回归分类器、SVM分类器、决策树分类器。如何集成呢?

最简单方法是聚合每个分类器的预测，将得票最多的结果作为预测类别。这种大多数投票分类器被称为**硬投票分类器**。若各分类器都能估算出类别的概率，可计算各类别概率之和，取和最大时的类别，则称为**软投票分类器**。



训练3种不同质的个体分类器

3.6.5 硬投票与软投票 | sklearn中的投票分类器

sklearn.ensemble提供了**VotingClassifier类**，实现了“软/硬投票分类器”。

```
VotingClassifier(estimators, voting='hard', …)
```

参数：

- *estimators*：元组(*str*，*estimator*)的列表。调用该VotingClassifier的fit方法，将拟合那些原始估计器的克隆（存储在类属性self.estimators_中）
- *voting*：*{'hard', 'soft'}, default='hard'*。 若“hard”，则用各estimator预测的类标签进行多数投票。若“soft”，根据各estimator预测概率之和的 argmax, 作为集成预测的类标签。

硬投票

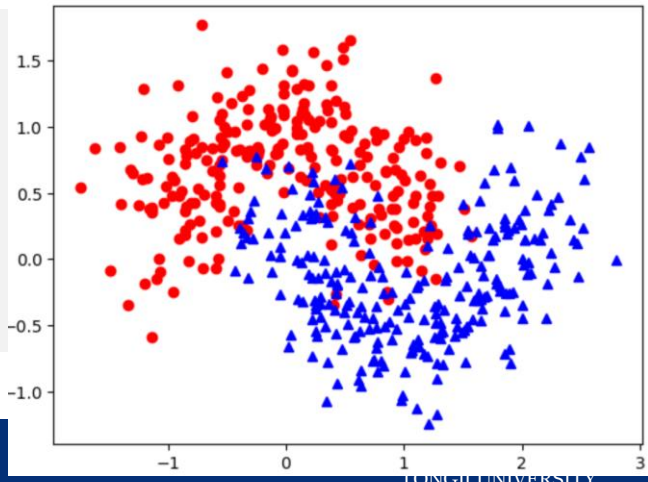
软投票

例3.22 投票分类器

为two moons数据集创建一个集成硬投票分类器和一个软投票分类器， 个体分类器由1个逻辑回归分类器，1个SVM分类器，1个决策树分类器构成。比较集成前后、以及硬投票和软投票分类器的性能。

```
from sklearn.datasets import make_moons
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
```

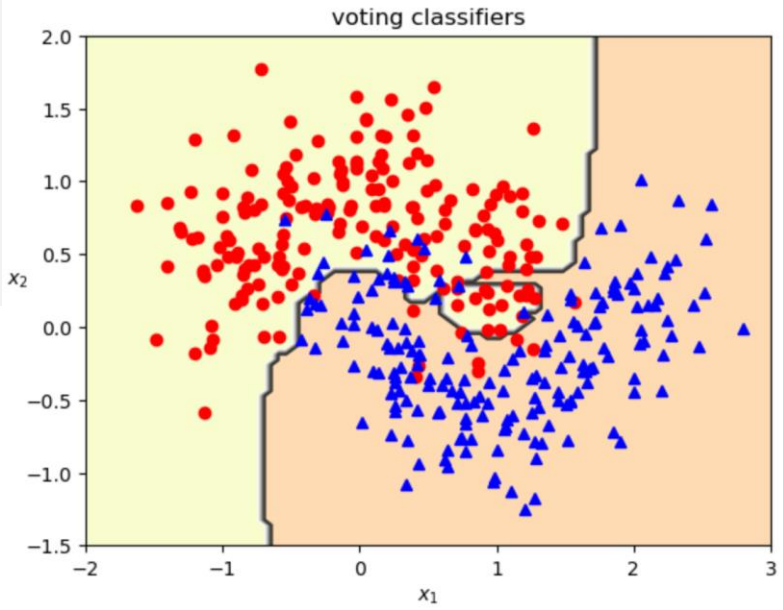
```
colors = ["r", "b"]
markers = ("o", "^")
for idx in (0, 1):
    plt.plot(X[:, 0][y == idx], X[:, 1][y == idx],
             color=colors[idx], marker=markers[idx], linestyle="none")
```



```
# 1.硬投票
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

voting_clf = VotingClassifier(estimators=[('lr', LogisticRegression(random_state=42)),
                                          ('tree', DecisionTreeClassifier(random_state=42)),
                                          ('svc', SVC(random_state=42))])
voting_clf.fit(X_train, y_train)
```

```
# 分别输出个体分类器的分类准确率
for name, clf in voting_clf.named_estimators_.items():
    print(name, "=", clf.score(X_test, y_test))
# 输出硬投票分类器的准确率
print('voting_clf =',voting_clf.score(X_test, y_test))
```

```
lr = 0.864
tree = 0.856
svc = 0.896
voting_clf = 0.904
```



```
# 2. 软投票
voting_clf.voting = "soft"
voting_clf.named_estimators[“svc”].probability = True  # 默认False，此时SVC不启用概率估计
voting_clf.fit(X_train, y_train)
voting_clf.score(X_test, y_test)
```

```
0.912
```

通常，软投票法的表现比硬投票法更好。因为它给予那些高度自信的投票更高的权重。

3.6.6 处理不平衡数据集

处理不平衡数据集是机器学习领域中的一个常见问题，存在多种优化策略：

1. **重采样技术**：
   - **过采样**：增加少数类样本的数量，可使用**SMOTE**（Synthetic Minority Over-sampling Technique） 等技术合成新的少数类样本。
   - **欠采样**：减少多数类样本的数量，可随机删除或使用如Tomek Links、 ENN（Edited Nearest Neighbours）等策略选择性地删除样本。
2. **算法层面**：
   - **代价敏感学习**：在训练过程中为不同类别的样本赋予不同的误分类代价，使模型更加关注少数类样本。
   - **集成学习**：使用如随机森林、梯度提升树等集成方法，通过多个基学习器的组合来提高少数类的识别率。
3. **数据生成**：使用生成对抗网络（GAN）等技术合成与少数类分布相似的合成数据。

。。。

GA-SMOTE 算法

- **GA-SMOTE**（Genetic Algorithm Synthetic Minority Over-sampling Technique）是一种结合了遗传算法（Genetic Algorithm, **GA**）和**SMOTE** 的改进算法，用于处理不平衡数据集中的分类问题。通过遗传算法的选择、交叉和变异操作，来优化SMOTE算法生成的合成样本。
- 在**GA-SMOTE**算法中，首先使用选择操作来有区别地选择少数类样本。然后，通过交叉和变异操作来控制合成样本的质量。这种结合使得GA-SMOTE能够在合成样本的整体效果上表现得更好，有效提高分类算法在不平衡数据集上的分类性能。

GA-SMOTE 算法步骤

GA-SMOTE算法的主要步骤：

1. **选择**：从少数类样本中选择个体，以有区别的方式进行，确保多样性。
2. **交叉**：通过交叉操作生成新的样本，这些样本是原有样本的组合。
3. **变异**：对生成的新样本进行变异操作，以引入新的遗传信息，增加样本的多样性。

1/4

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.neighbors import NearestNeighbors
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# 生成不平衡数据集
X, y = make_classification(n_classes=2, class_sep=2, weights=[0.1, 0.9],
                           n_informative=3, n_redundant=1, flip_y=0,
                           n_features=20, n_clusters_per_class=1,
                           n_samples=1000, random_state=42)

# 数据集划分为训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

2/4

```
# 简化版的GA-SMOTE算法框架
class GASMOTE:
    def __init__(self, X_minority, k=5, N=2, pop_size=20, generations=10, cross_prob=0.8, mutation_prob=0.2):
        self.X_minority = X_minority  # 少数类样本
        self.k = k    # k近邻数量
        self.N = N  # 采样倍率
        self.pop_size = pop_size  # 种群大小
        self.generations = generations  # 遗传代数
        self.cross_prob = cross_prob  # 交叉概率
        self.mutation_prob = mutation_prob  # 变异概率
        self.synthetic_samples = []  # 存储生成的合成样本

    def fit(self):
        # 初始化种群（这里简化为随机选择少数类样本作为初始种群）
        # 遗传算法流程
         return np.array(self.synthetic_samples)  # 返回生成的合成样本
```

详见下页

3/4

fit()详解

```
def fit(self):
    population = [self.X_minority[np.random.randint(0, len(self.X_minority))] for _ in range(self.pop_size)] # 初始化种群
    for gen in range(self.generations):
        selected_parents = [population[np.random.randint(0, self.pop_size)] for _ in range(self.pop_size)] # 选择操作
        offspring = []
        for i in range(0, self.pop_size, 2):                                    # 交叉操作
            parent1, parent2 = selected_parents[i], selected_parents[i+1]
            if np.random.rand() < self.cross_prob:
                crossover_point = np.random.randint(1, len(parent1)-1)
                child1 = np.concatenate((parent1[:crossover_point], parent2[crossover_point:]))
                child2 = np.concatenate((parent2[:crossover_point], parent1[crossover_point:]))
                offspring.extend([child1, child2])
            else:
                offspring.extend([parent1, parent2])
        for i in range(len(offspring)):                                         # 变异操作
            if np.random.rand() < self.mutation_prob:
                mutation_point = np.random.randint(0, len(offspring[i]))
                offspring[i][mutation_point] = np.random.rand()
    nn = NearestNeighbors(n_neighbors=self.k)
    nn.fit(offspring)
    for sample in offspring:
        distances, indices = nn.kneighbors([sample], return_distance=True)
        nearest_neighbors = offspring[indices[0][0]]
        for _ in range(self.N):
            nn_idx = np.random.randint(0, self.k)
            dif = nearest_neighbors[nn_idx] - sample
            gap = np.random.rand()
            synthetic_sample = sample + gap * dif
            self.synthetic_samples.append(synthetic_sample)
    population = offspring    # 更新种群
return np.array(self.synthetic_samples)   # 返回生成的合成样本
```

 # 遗传算法流程

 # 使用SMOTE算法生成合成样本

4/4

```
 # 使用GA-SMOTE算法生成合成样本
gasmote = GASMOTE(X_train[y_train == 0], k=5, N=2, pop_size=20, generations=10)
synthetic_samples = gasmote.fit()

# 将生成的合成样本添加到训练集中
X_train_balanced = np.vstack((X_train, synthetic_samples))
y_train_balanced = np.hstack((y_train, [0] * len(synthetic_samples)))  # 假设生成的合成样本都属于少数类

# 训练分类器并评估性能
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train_balanced, y_train_balanced)
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

小结

集成学习：

- 集成学习是一种可应用于任何监督学习算法的技术。
- 当每个个体学习略优于随机猜测时，通常集成效果好于个体。
- bagging　利用自助采样训练模型，投票（或平均）聚合
- 随机森林　针对决策树的bagging实现
- boosting　依次训练弱学习器来组合成一个强学习器。
- XGBoost　高效的梯度boosting

作业八　8.1 预测泰坦尼克号幸存者

titanic数据共有两个文件：

- train.csv是训练集，样本类别（Survived）已标注；
- test.csv是测试集，无标注信息，是需要用你创建的模型来预测的数据。

Train.csv是892行(含表头)、12列的数据表。特征如下：

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
乘客ID	1表示幸存,0表示遇难	舱位等级	乘客姓名	乘客性别	乘客年龄	兄弟姐妹同在船上的数量	同船的父辈人数	乘客票号	乘客的体热指标	乘客所在的船舱号	乘客登船的港口

1. 数据预处理

   1) 提取Survived列的数据作为目标向量

   2) 丢弃无用的特征数据{‘Name’，‘Ticket’，‘Cabin’，‘PassengerId’}

   3) 对数据进行转换

2. 用titanic 数据集train.csv，创建一棵预剪枝决策树（设置max_depth为4），评估该树，可视化这棵树。用该树预测test.csv中的幸存者。

3. 创建随机森林分类器，令n_estimators验证范围为[1,5,10,20,30,40,50,60,70,80,90,100]，计算训练得分和验证得分，并画出集成分类器的验证曲线。保存模型到文件中。

作业八　　8.2 预测电信客户流失

根据电信客户信息（telco-churn.csv），建立一个模型，预测哪些客户会流失。

该数据来自IBM，包含多个字段：费用、使用期限、流量信息以及一个表明客户是否会流失的变量（churn）等。

(1) 读入数据，删去不必要的变量customerID

(2) 将TotalCharges由字符串转换为数值。提示：

data['TotalCharges'] = pd.to_numeric(data['TotalCharges'],errors='coerce’)

说明：若errors设为“coerce”，无效的解析将设置为NaN。

(3) 将所有分类变量进行有序编码。

(4) 随机打乱数据。拆分出X，y。8：2 划分数据集为训练集和测试集。

(5) 训练一个XGBoost模型，令其有1000棵树,学习率为0.01,max_depth=4，输出测试集的分类性能报告。保存模型为json文件。