8.集成学习算法（一）

# 1 实验目的

1.理解sklearn.ensemble模块中投票集成法、装袋集成法、随机森林算法的实现原理；

2.了解如何使用sklearn.ensemble模块中各个常用算法完成机器学习任务。

# 2 实验要求

本次实验后，要求同学们能：

1.理解集成算法的基本原理概念；

1.掌握投票法、袋装法、随机森林算法的实现原理；

2.学会使用Scikit-learn实现常用的集成算法；

3.了解sklearn.ensemble模块中各常用算法的不同特点，了解各个超参数的含义。

# 3 实验内容

## 3.1 集成学习算法简介

集成算法不是一种独立的机器学习算法，它是将多种互不相关的较弱学习器，通过适当的方式“集成”在一起的算法，往往能得到比单个弱学习器更好的结果。就好比中国俗语常说的“三个臭皮匠顶个诸葛亮”。

集成算法主要是通过一下几种结构实现集成，让弱学习器们相互配合：

（1）并联方式，所有学习器并行地进行训练、预测，在训练与预测期间互不干扰。最后通过取平均值法或者少数服从多数的投票法，确定最终的预测结果，例如有放回的装袋算法Bagging，以及无放回的装袋算法Pasting。

一般在弱学习器势均力敌、不分主次时可以选择并联结构。由于是并联结构，可以在不同CPU内核，甚至不同服务器上并行地训练集成学习器、并行地预测结果，充分利用计算资源，在资源充足的条件下极大地提升计算效率，这也是Bagging算法十分流行的原因。

（2）串联方式，例如Boosting算法。所有学习器串行地进行训练，前一个学习器的预测结果会影响后一个学习器，从而使后面的学习器得到更好的训练结果，例如基于前一个学习器的错误来提升性能的自适应提升法AdaBoost，以及对前一个学习器的误差进行拟合的梯度提升法GradientBoost。

一般在弱学习器之间已经有了明确的分工，知道谁负责主要工作，谁负责辅助工作，使用串联结构可以用较少的开销得到很好的效果。

（3）堆叠方式，学习器之间既有并联又有串联。第一层的弱学习器并行地进行预测，然后第二层的学习器把对第一层学习器的预测结果作为训练数据进行训练，如果还有第三层、第四层……也是同理，例如Stacking算法。

scikit-learn库中sklearn.ensemble模块提供了不同的集成算法模型，下面介绍Voting、Bagging、RandomForest算法。其他集成算法将在下一章节进行介绍。

## 3.2 Voting投票法

### 3.2.1原理

投票集成算法最简单的集成算法，将基于几个不同机器学习算法的弱学习器并行运行，在**同一个数据集上**训练测试，将它们的结果简单地进行求均值（回归）或投票（分类）作为最终结果。投票法集成出的强学习器往往比集成中最好的弱学习器效果更好。

投票法集成分类器时可以分为硬投票和软投票。硬投票是根据每个分类器对样本实际预测出的类别进行投票，而软投票是根据每一个分类器计算出的样本属于每一个类别的概率来投票，将某样本属于某类别的概率进行平均。一般情况下，软投票法的效果更好，但是需要被集成的弱学习器都有预测概率的能力。

需要注意的是，理论上用于集成的相互独立的弱学习器越多，集成出的强学习器效果就越好。哪怕单个弱学习器正确率仅有51%，理论上1000个这样的弱学习器集成出的强学习器也能达到75%左右的正确率。

然而，实际应用时无法保证弱学习器之间相互独立，因为它们是用同一个训练集进行训练的。可能会因为这个训练集自身的特性（比如噪音、异常值等），导致多个学习器犯很多相同的错误，同时投给了错误结果，最终正确率降低。

所以需要尽可能地让弱学习器相关性降低，投票法一般会**选用不同的机器学习算法**作为弱学习器。

### 3.2.2 实验

（1）导入依赖并处理数据

本实验将三个弱学习器通过投票法集成为一个强学习器，在双月型二分类数据集上进行分类实验。投票集成算法也可以解决回归问题，原理和大致步骤一致。

用Scikit-learn.datasets中的make\_moons()方法得到带有噪音的两个交错半月（双月型）数据，每个半月代表一类，可以用这个数据集来展示二分类算法实验。

用matplotlib绘制出图像。用train\_test\_split()方法划分数据集。代码如下：

import matplotlib.pyplot as plt

from sklearn.datasets import make\_moons

from sklearn.model\_selection import train\_test\_split

# 弱学习器

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KneighborsClassifier

# 投票集成法

from sklearn.ensemble import VotingClassifier

# 构造一个用于演示的交错半圆数据集

moons = make\_moons(n\_samples=(60, 40), noise=0.3, random\_state=5)

data = moons[0]

target = moons[1]

# 双月数据集可视化

plt.scatter(data[:, 0], data[:, 1], c=target, s=30, cmap="cool")

plt.xlabel('X')

plt.ylabel('Y')

plt.title("moons noise=0.3")

plt.show()

# 划分训练集测试集

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

data,

target,

test\_size=0.2,

random\_state=1

)

（2）创建学习器实例

首先，先创建四个弱学习器，分别是一个逻辑回归分类器、一个支持向量机分类器、一个决策树分类器、一个KNN分类器。将KNN分类器的K设置为10。

将SVC的probability参数设置为True，这是为了让SVC模型具备计算样本属于每一类概率的能力，从而实现“软投票”。

代码如下：

log\_clf = LogisticRegression()

svc\_clf = SVC(probability=True)

tree\_clf = DecisionTreeClassifier()

knn\_clf = KNeighborsClassifier(n\_neighbors=10)

之后，用投票法将它们集成为强学习器。

用sklearn.ensemble模块的VotingClassifier类可以创建用投票法集成的强学习器。其构造器的重要参数及含义如下：

**estimators：用于传入集成的弱学习器列表，列表的元素为（str, estimator）的形式，str为字符串类型的学习器名，estimator为创建好的学习器实例。**

**voting：选择投票方法，可以选择“hard”硬投票，根据每一个分类器的对样本实际分类结果来投票预测；如果用于集成的弱学习器都具有predict\_proba()方法，也可以选择“soft”软投票，根据所有单个学习器计算出的每个样本属于各个类别概率的均值来预测。**

**weights：用于设置弱学习器权重。可以给效果好的弱学习器更高权重，给效果不好的学习器较低权重，从而提示集成出的强学习器的学习能力。默认为统一权重。**

**verbose：设置是否输出集成学习器学习的详细过程（各个弱学习器的训练完成时间）。**

实验使用“软投票”法将之前的四个弱学习器进行集成，每个弱学习器权值一致，代码如下：

vote\_clf = VotingClassifier(

estimators=[("log\_clf", log\_clf), ("svc\_clf", svc\_clf), ("tree\_clf", tree\_clf), ("knn\_clf", knn\_clf)],

voting="soft",

)

（3）训练并测试

训练各个学习器，打印出各个弱学习器与强学习器在测试集上的得分。

for clf in (log\_clf, svc\_clf, tree\_clf, knn\_clf, vote\_clf):

clf.fit(X\_train, Y\_train)

print(clf.\_\_class\_\_.\_\_name\_\_, clf.score(X\_test, Y\_test))

可以对集成出的投票强学习器调用transform(X)方法，查看每一个弱学习器的预测结果。如果选择的是“软投票”，会返回每个分类器计算出X中各个样本属于每一类的概率；如果选择“硬投票”，返回每个分类器对X中各个样本的分类结果。代码如下：

print(np.round(vote\_clf.transform(X\_test), 4))

### 3.2.3 实验结果

（1）双月型数据集可视化

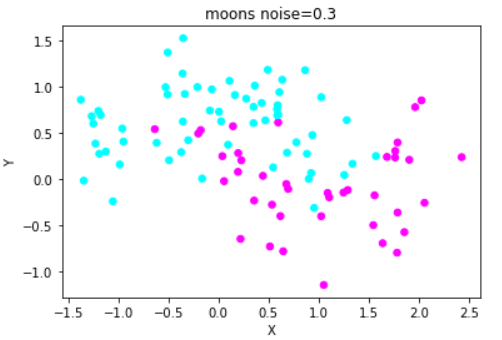


图1 实验数据集可视化

（2）各个学习器的评分

可以看到，投票法集成所得的强学习器（VotingClassifier）在测试集上有最高的评分。

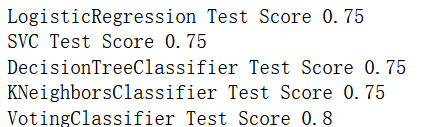


图2 各个学习器评分

（3）查看各个弱学习器预测结果

因为是二分类问题，transform(X)返回的矩阵中，第i行第2k-1列上的数据代表第k个弱学习器认为第i个样本属于第1个类别的概率；第i行第2k列上的数据代表第k个弱学习器认为第i个样本属于第2个类别的概率

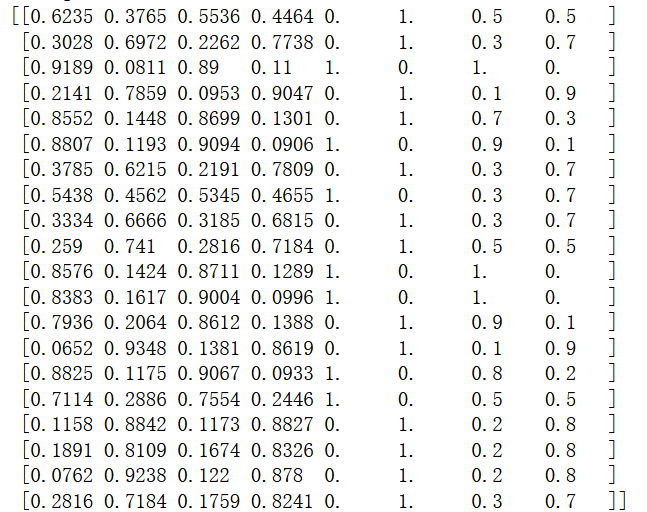


图3 弱学习器概率预测结果

## 3.3装袋法

### 3.3.1 原理

前面的投票法是在**用一个数据集上使用不同的训练算法**来得到不同的弱学习器（尽可能不相关的弱学习器）进行集成。

而另一种得到不同弱学习器的方式就是“装袋法”，它是**使用同一种机器学习算法构建弱学习器，在训练集的不同随机子集上进行训练**，尽量减少弱学习器间的相关性。最终也是通过求平均或者投票得到最终预测结果。

在选取训练集的不同随机子集时，又可以分为有放回抽样的Bagging算法，和无放回抽样的Pasting算法。

简单来说，Bagging算法允许一个训练实例被多次采样，被多个弱学习器用于训练，这样得到的弱学习器不完全独立（可能用到了相同的训练数据犯相同错误），但是因为是有放回抽样，可以生成任意数量的弱学习器；而Pasting算法一个训练实例被用于某弱学习器的训练后，就不再被别的弱学习器使用，这样得到的弱学习器更加独立（每个弱学习器用到的训练数据完全不同），但是可训练的弱学习器数量受到训练集大小的限制。

另外，大名鼎鼎的随机森林算法RandomForest，其实就是将决策树作为弱学习器进行集成的Bagging算法，它是最流行的机器学习算法之一。

### 3.3.2 实验

（1）导入依赖并处理数据

还是用双月型数据集进行演示实验。代码如下：

import matplotlib.pyplot as plt

from sklearn.datasets import make\_moons

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import BaggingClassifier

moons = make\_moons(n\_samples=(120, 80), noise=0.3, random\_state=5)

data = moons[0]

target = moons[1]

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

data,

target,

test\_size=0.2,

random\_state=1

)

（2）创建学习器实例

使用sklearn.ensemble模块的BaggingClassifier类可以创建Bagging强学习器实例，其构造器的重要参数及含义如下：

**base\_estimator：**指定用于集成的弱学习器。如果不进行指定，默认为None，会自动使用决策树作为基学习器。

**n\_estimators：设置用于集成的弱学习器个数。**

**bootstrap：选择True表示对训练样本有放回抽样，即Bagging集成算法；选择False表示对训练数据的样本无放回抽样，即Pasting集成算法。**

**max\_samples：设置抽取的用于训练弱学习器的样本数量。可以为int类型表示个数或float类型表示比例。**

**oob\_score：设置是否启用包外评估。bootstrap为True，即有放回抽样时，有些样本会被多次抽到，有些样本可能一直都没被抽到。启用包外评估就是用那些没被某个预测器抽到的样本来对预测器进行评估。启用后，可以用oob\_score\_直接查看包外评分值。**

**random\_state：用于固定算法的随机性。**

**verbose：设置是否打印学习过程中的详细信息。**

**max\_features、bootstrap\_features：与bootstrap、max\_samples类似，但是这两个参数是设置对特征进行抽样，每个弱学习器将用输入样本特征的随机子集来训练，称为“随机子空间法”。**

**创建一个由500棵决策树集成的有放回Bagging强学习器，一个由20棵决策树集成的无放回Pasting法强学习器。再创建一个单独的决策树实例用于对比，代码如下：**

bag\_clf = BaggingClassifier(

DecisionTreeClassifier(max\_depth=5), n\_estimators=500,

max\_samples=20, bootstrap=True, n\_jobs=-1, oob\_score=True

)

past\_clf = BaggingClassifier(

DecisionTreeClassifier(max\_depth=5), n\_estimators=20,

max\_samples=20, bootstrap=False, n\_jobs=-1

)

tree\_clf = DecisionTreeClassifier(max\_depth=5)

（3）训练模型并评分

for clf in (tree\_clf, bag\_clf, past\_clf):

clf.fit(X\_train, Y\_train)

print(clf)

print("train score:", clf.score(X\_train, Y\_train))

print("test score:", clf.score(X\_test, Y\_test))

print("\n有放回抽样集成的包外评估：", bag\_clf.oob\_score\_) # 包外评估

### 3.3.3实验结果

（1）算法评分

可以看出，Bagging集成出的强学习器的预测效果比单个弱学习器更好。

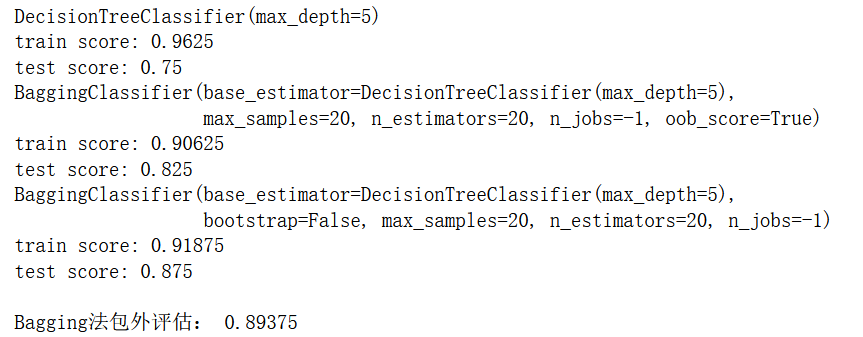


图4各个学习器评分

## 3.4 随机森林算法

（1） 随机森林实验

上面用决策树作为弱分类器，通过有放回抽样来训练集成出的强学习器其实就是随机森林。

Scikit-learn中提供了RandomForestClassifier类和RandomForestRegressor类直接实现随机森林，用这两个类实现随机森林更为方便，它们在构造时，不仅可以接收Bagging的所有参数，还可以接收DecisionTree的所有参数，更方便对作为弱分类器的决策树模型的各个超参数进行详细设定。

以下代码用RandomForestClassifier类创建了一个与前面Bagging集成算法等效的随机森林算法，在鸢尾花数据集上进行分类实验：

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

# 加载数据

iris = load\_iris()

data = iris.data

target = iris.target

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

data,

target,

test\_size=0.2,

random\_state=1

)

# 训练随机森林模型并评分

rnd\_clf = RandomForestClassifier(max\_depth=5, n\_estimators=20,max\_samples=20,

bootstrap=True, n\_jobs=-1, oob\_score=True)

rnd\_clf.fit(X\_train, Y\_train)

print("RandomForestClassifier Score: ", rnd\_clf.score(X\_test, Y\_test))

模型在测试集上的评分如下：



图5 随机森林算法评分

Scikit-learn还提供了一种更加随机的极端随机森林算法（ExtraTreesClassifier、ExtraTreesRegressor），是用极端随机数集成而来，它随机地找寻局部最佳特征，并且对每个特征使用随机阈值而不是最佳阈值，训练效率比普通随机森林快得多，常常能有更强的泛化能力。它有与随机森林一样的API用法，不再赘述。

（2） 查看特征重要性

随机森林算法和极端随机森林算法有一个优势在于，它们创建出的模型可以通过.feature\_importances\_查看样本数据每个特征的重要性。

print("Feature Importance: ", rnd\_clf.feature\_importances\_)

结果如下：



图6 由随机森林算法得到的鸢尾花特征重要性

如果是特征很多的数据集，那么特征重要性这一指标能很方便地提示我们重点观察某些特征，得到更优的模型。