9.集成学习算法

# 1 实验目的

1.理解sklearn.ensemble模块中提升法和堆叠法的实现原理；

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.掌握提升法和堆叠法两类机器学习集成算法的实现原理；

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 AdaBoost自适应提升法

### 3.1.1 原理

与上一章中并联方式集成的投票法和袋装法不同，提升法是串行地循环地训练弱学习器，每一次都对上一个学习器做出一些改正，最后得到一个最好的强学习器。目前最流行的提升法主要有AdaBoost自适应提升法和GradientBoost梯度提升法。

AdaBoost自适应提升法是让新学习器对上一个学习器进行纠正，提高上一个学习器拟合错误的部分的权值，从而使学习器越来越关注最难以解决、常犯错的那一部分问题，最终实现减少犯错并提升准确率的目标。

具体步骤为：①先用训练集训练一个弱学习器（如决策树），用它先进行一次预测。②根据第一个所学习器的分类结果，增加被它错误分类的样本的权重。③用这个新的权重再对第二个学习器进行训练。④不断更新权重，不断训练新的学习器，循环地提升。

### 3.1.2 自适应提升算法实验

（1）导入依赖并加载数据

创建一个带扰动的波动函数作为实验数据。

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import AdaBoostRegressor

from sklearn.ensemble import BaggingRegressor

# 创建数据集

rs = np.random.RandomState(1)

X\_train = np.linspace(0, 6, 97)[:, np.newaxis]

Y\_train = np.sin(X\_train).ravel() + np.sin(3 \* X\_train).ravel() + rs.randn(X\_train.shape[0]) \* 0.1

X\_test = np.linspace(0, 5, 17)[:, np.newaxis]

Y\_test = np.sin(X\_test).ravel() + np.sin(3 \* X\_test).ravel() + rs.randn(X\_test.shape[0]) \* 0.1

（2）创建学习器实例

用sklearn.ensemble模块的AdaBoostRegressor类可以创建一个AdaBoost集成算法强学习器实例，其构造器的重要参数及含义如下：

**base\_estimator：设定用于集成的弱学习器，默认为最大深度为3的决策树。**

**n\_estimators：设定用于集成的弱学习器个数。默认为50个，但是如果算法已经在训练集上实现完全拟合，则会提前停止。**

**learning\_rate：设定学习率，即，每个弱学习器的贡献率，太低容易欠拟合，太高容易过拟合。与参数n\_estimators相互影响。**

**loss：每次迭代提升时，会用到损失函数来更新样本权重。默认值为“linear”，也可以选择“square”。**

**random\_state：用于固定算法的随机性。**

**下面用300个最大深度为4的决策树回归器集成出一个AdaBoost强学习器，并创建一个用300个最大深度为4的决策树回归器集成出的随机森林强学习器，和一个最大深度为4的决策树回归器，进行对比。代码如下：**

regr\_1 = DecisionTreeRegressor(max\_depth=4)

regr\_2 = AdaBoostRegressor(DecisionTreeRegressor(max\_depth=4),

n\_estimators=300, random\_state=rs)

regr\_3 = BaggingRegressor(DecisionTreeRegressor(max\_depth=4),

n\_estimators=300, random\_state=rs)

（3）训练并评分

在训练集上分别对三个模型进行训练，查看三个模型在测试集上的评分。

# 训练学习器

regr\_1.fit(X\_train, Y\_train)

regr\_2.fit(X\_train, Y\_train)

regr\_3.fit(X\_train, Y\_train)

# 学习器评分

print("Decision Tree(n=1) Test Score\n", regr\_1.score(X\_test, Y\_test))

print("AdaBoost(n\_estimators=300) Test Score\n", regr\_2.score(X\_test, Y\_test))

print("Bagging(n\_estimators=300) Test Score\n", regr\_3.score(X\_test, Y\_test))

（4）可视化

将各个算法对训练集的拟合情况进行可视化，代码如下：

# 预测

y\_1 = regr\_1.predict(X\_train)

y\_2 = regr\_2.predict(X\_train)

y\_3 = regr\_3.predict(X\_train)

# 拟合可视化

plt.figure(num=1, figsize=(12, 8))

plt.scatter(X\_train, Y\_train, c="black", s=8, label="training samples")

plt.plot(X\_train, y\_1, c="g", label="DecisionTree n=1", linewidth=4, alpha=0.8)

plt.plot(X\_train, y\_2, c="r", label="AdaBoost n\_estimators=300", linewidth=4, alpha=0.8)

plt.plot(X\_train, y\_3, c="b", label="Bagging n\_estimators=300", linewidth=4, alpha=0.8)

plt.xlabel("data")

plt.ylabel("target")

plt.title("Boosted Decision Tree Regression")

plt.legend()

plt.show()

### 3.1.3 实验结果

（1）模型评分

可以看到，用AdaBoost法对300个深度为4的决策树进行集成，得到的强学习器在测试集上的评分相比单个决策树学习器有显著的提升。

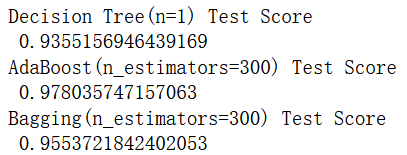


图1 各模型评分

（2）拟合图像

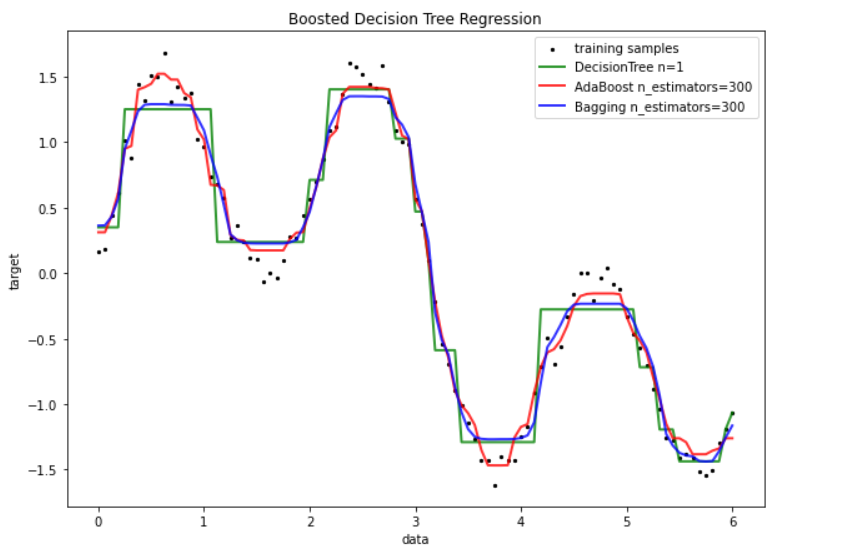


图2 不同模型训练集上拟合情况

## 3.2 GradientBoosting梯度提升法

### 3.2.1 原理

另一种流行的提升集成算法就是梯度提升法GradientBoosting。与AdaBoost类似，梯度提升法也是不断在集成中添加新的学习器，每个新学习器都会对前一个学习器做出改进。不同之处在于，梯度提升法中新学习器是对前一个学习器拟合时的误差进行拟合。

具体步骤为：①先用训练集训练一个弱学习器（如决策树），用它先进行一次预测。②计算出第一个弱学习器预测据结果与实际值的偏差，用这个偏差值来训练第二个学习器，所以第一、二弱学习器集成后的预测结果为第一个学习器的预测值（预测训练数据）加上第二个学习器的预测值（预测第一个学习器预测结果与实际数据的偏差）。③不断增加新的弱学习器，每个新的弱学习器都对上一个学习器的偏差进行拟合，而整体集成出的强学习器的预测值为全部弱学习器预测值之和。

下面用决策树作为弱学习器，集成一个梯度提升法强学习器，这样用决策树集成得到的强学习器又称为梯度提升树（梯度提升回归树BGRT或梯度提升分类树BGCT）。结合实验对梯度提升法进行进一步介绍。

### 3.2.2 梯度提升集成算法实验

（1）数据预处理

还是使用3.1实验中的数据进行演示，代码如下：

from sklearn.ensemble import GradientBoostingRegressor

import matplotlib.pyplot as plt

import numpy as np

# 创建数据集

rs = np.random.RandomState(1)

X\_train = np.linspace(0, 6, 97)[:, np.newaxis]

Y\_train = np.sin(X\_train).ravel() + np.sin(3 \* X\_train).ravel() + rs.randn(X\_train.shape[0]) \* 0.1

X\_test = np.linspace(0, 5, 17)[:, np.newaxis]

Y\_test = np.sin(X\_test).ravel() + np.sin(3 \* X\_test).ravel() + rs.randn(X\_test.shape[0]) \* 0.1

（2）创建梯度提升树模型并测试

使用sklearn.ensemble模块的GradientBoostingClassifier类可以创建一个梯度提升树模型分类学习器。它的构造器与RandomForestClassifier的一样，不仅包含梯度提升集成算法的参数，还包含了决策树算法的全部参数，可以更方便地对作为弱学习器的决策树进行详细设定。重要参数及含义如下（决策树相关参数不再赘述）：

**n\_estimators：**设定弱学习器的数量，太小容易欠拟合，太大容易过拟合，默认值为100。

**learning\_rate：**设定每个弱学习器的贡献率，也称为学习步长，常常与参数n\_estimators一起考虑。

**subsample：**设定用于拟合单个弱学习器的样本比例，默认为1代表用全部训练样本来训练，如果小于1，就变成了随机梯度提升算法，可以增加模型的泛化能力。

**loss：**设定梯度提升算法中要拟合的损失函数。用于分类任务时可以选择对数似然损失函数“deviance”和指数损失函数“exponential”，推荐使用默认值“deviance”。如果使用“exponential”，则变成了AdaBoost算法。用于回归任务时可以选择均方差“ls”, 绝对损失“lad”, Huber损失“huber”和分位数损失“quantile”。噪音不多时，使用默认的“ls”效果最好。

**n\_iter\_no\_change、validation\_fraction、tol：三个参数共同用于设定算法的提前停止。** n\_iter\_no\_change默认为None，表示禁用提前停止功能。当n\_iter\_no\_change为任意数字时，会提前从训练集中预留**validation\_fraction（0~1之间的浮点数）比例的样本作为验证集，每次训练新弱学习器之后都会在验证集上验证，如果准确度的提升小于tol，则不再训练新的弱学习器，提前停止梯度提升算法。**

**下面创建一个由3棵最大深度为3的决策树由梯度提升法集成出的梯度提升回归树，训练模型并验证，代码如下：**

gbrt = GradientBoostingRegressor(max\_depth=3, n\_estimators=3, learning\_rate=1)

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

# 训练并预测

y\_pred = gbrt.predict(X\_train)

y\_pred\_test = gbrt.predict(X\_test)

# 可视化训练集拟合

plt.figure(figsize=(9, 3))

plt.subplot(1, 2, 1)

plt.title("Fit Train Data")

plt.scatter(X\_train, Y\_train, c="k", s=3, label="train data")

plt.plot(X\_train, y\_pred, c="r", label="h3(x)", linewidth=3)

plt.xlabel("data")

plt.ylabel("target")

plt.legend()

# 可视化测试集预测

plt.subplot(1, 2, 2)

plt.title("Predict Test Data")

plt.scatter(X\_test, Y\_test, c="k", s=3, label="train data")

plt.plot(X\_test, y\_pred\_test, c="r", label="h3(x)", linewidth=3)

plt.xlabel("data")

plt.ylabel("target")

plt.legend()

plt.show()

# 查看算法评分

print("Score on Train: ", gbrt.score(X\_train, Y\_train))

print("Score on Test: ", gbrt.score(X\_test, Y\_test))

### 3.2.3 分步骤演示

下面创建3个最大深度为3的决策树，自己编写代码用梯度提升法将它们进行集成，绘制出每一个弱学习器的拟合图像，方便同学们加深对梯度提升法的理解。

from sklearn.tree import DecisionTreeRegressor

tree1 = DecisionTreeRegressor(max\_depth=3)

tree2 = DecisionTreeRegressor(max\_depth=3)

tree3 = DecisionTreeRegressor(max\_depth=3)

## S1：第1个弱学习器拟合原数据

y1 = Y\_train

tree1.fit(X\_train, y1)

y\_pred1 = tree1.predict(X\_train)

plt.figure(num=1, figsize=(9, 9))

plt.subplots\_adjust(wspace=0.2,hspace=0.4)

# 绘制第1个弱学习器的拟合图像

plt.subplot(3, 2, 1)

plt.scatter(X\_train, y1, c="k", s=3, label="train data")

plt.plot(X\_train, y\_pred1, c="r", label="h1(x)", linewidth=2)

plt.xlabel("data")

plt.ylabel("target")

plt.legend()

# 绘制第1个弱学习器集成出的强学习器对原数据的拟合图像

plt.subplot(3, 2, 2)

plt.scatter(X\_train, Y\_train, c="k", s=3, label="train data")

plt.plot(X\_train, y\_pred1, c="r", label="H(x)=h1(x)", linewidth=2)

plt.xlabel("data")

plt.ylabel("target-1")

plt.legend()

## S2：第2个学习器拟合第一个学习器与原数据的偏差

y2 = y1 - y\_pred1

tree2.fit(X\_train, y2)

y\_pred2 = tree2.predict(X\_train)

# 绘制第2个学习器的拟合图像

plt.subplot(3, 2, 3)

plt.scatter(X\_train, y2, c="k", s=3, label="train data")

plt.plot(X\_train, y\_pred2, c="r", label="h2(x)", linewidth=2)

plt.xlabel("data")

plt.ylabel("loss1=target-predict1")

plt.legend()

# 绘制前2个弱学习器集成出的强学习器对原数据的拟合图像

plt.subplot(3, 2, 4)

plt.scatter(X\_train, Y\_train, c="k", s=3, label="train data")

plt.plot(X\_train, y\_pred1 + y\_pred2, c="r", label="H(x)=h1(2)+h3(x)", linewidth=2)

plt.xlabel("data")

plt.ylabel("target-2")

plt.legend()

## S3：第3个弱学习器拟合第2个所学习器预测结果与第1个学习器预测偏差的偏差

y3 = y2 - y\_pred2

tree3.fit(X\_train, y3)

y\_pred3 = tree3.predict(X\_train)

# 绘制第3个学习器的拟合图像

plt.subplot(3, 2, 5)

plt.scatter(X\_train, y3, c="k", s=3, label="train data")

plt.plot(X\_train, y\_pred3, c="r", label="h3(x)", linewidth=2)

plt.xlabel("data")

plt.ylabel("loss2=loss1-predict2")

plt.legend()

# 绘制前3个弱学习器集成出的强学习器对原数据的拟合图像

plt.subplot(3, 2, 6)

plt.scatter(X\_train, Y\_train, c="k", s=3, label="train data")

plt.plot(X\_train, y\_pred1 + y\_pred2 + y\_pred3, c="r", label="H(x)=h1(x)+h2(x)+h3(x)", linewidth=2)

plt.xlabel("data")

plt.ylabel("target-3")

plt.legend()

plt.show()

3.2.4 实验结果

（1）梯度上升集成算法拟合结果

用sklearn.ensemble中自带的BGRT集成学习器训练一个由三个最大深度为3的决策树集合而成的梯度提升法强学习器。该强学习器的拟合、预测情况，以及其在训练集、测试集上的评分如下：

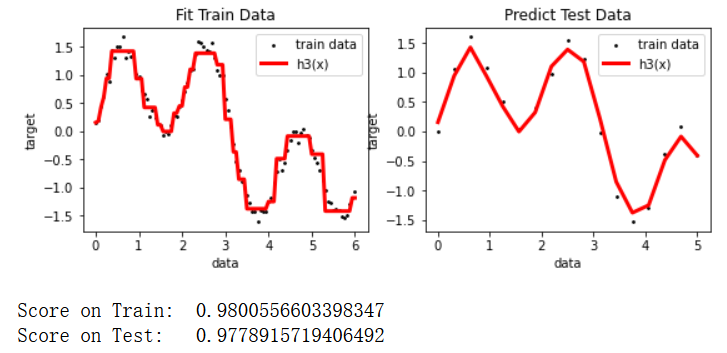


图3 梯度上升集成算法实验结果结果

（2）梯度上升集成算法分步演示

可以看到，每一个弱学习器都是对上一个学习器的预测误差进行拟合，最后总体的拟合效果越来越接近原数据。

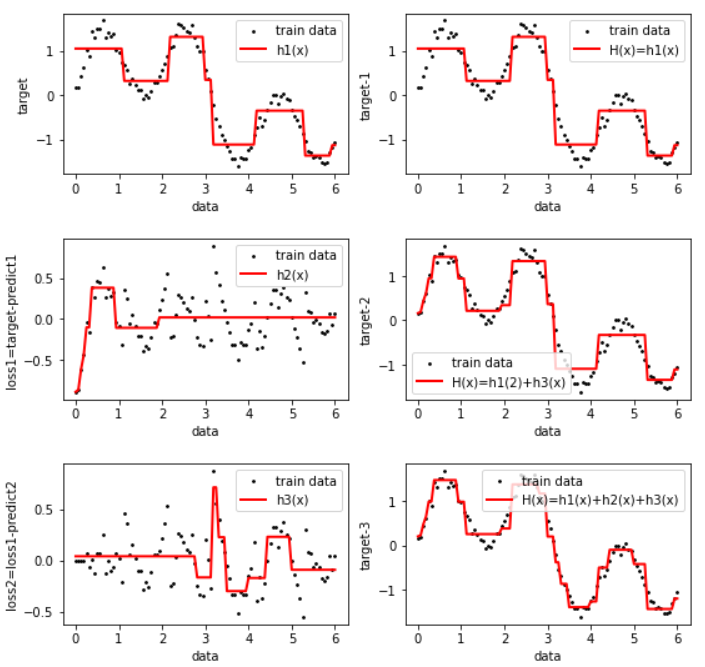


图4 梯度上升集成算法分步演示

## 3.3 Stacking堆叠法

### 3.3.1 原理

最后要介绍的一种集成方法就是堆叠法，它的主要思想其实很简单，相比前面介绍的并联集成算法是用一个函数来聚合所有弱学习器的结果，比如投票、求平均值，堆叠法训练了一个模型来聚合所有弱学习器的预测。

以两层的堆叠集成法为例，底层的学习器（称为基础学习器）分别预测出不同的值，然后在第二层的最终学习器（称为元学习器）把第一层得到的预测值作为输入，进行最终的预测。

堆叠法的难点主要在于如何得到用于训练元学习器的元数据，如果每个基础学习器就简单地使用全部训练数据做训练，并将全部训练数据上的预测结果作为元训练数据，训练和预测都使用了同样的数据，会导致模型泛化能力很弱。常使用的方法是折外预测法或留存集法，下面对折外预测法进行简单介绍。

以二层堆叠法集成为例，类似k折交叉验证（下一章会具体讲解），先将训练数据划分为k份，每一份数据称为一“折”，每次用第i折之外的k-1折数据训练各个基础学习器（i=1，2，……，k），并在第i折上用训练出的基础学习器进行预测，第j个基础学习器的预测结果作为元训练数据的第j个特征（保存在元训练数据的第j列）。

总结一下就是，这样做可以确保用于预测的数据对于各个基础学习器来说是“干净”的（基础学习器训练时从未见过这些数据）。最终得到的元训练数据与原训练数据有相同的样本数大小、相同的数据标签，但是元训练数据的各个特征为各个基础学习器的预测结果。

### 3.3.2 堆叠法算法实验

（1）导入依赖并处理数据

使用Scikit-learn自带的糖尿病数据集进行实验。该数据集一共有442个样本数据，每个样本有10个特征维度分别代表糖尿病病人的十个不同生理指标。而样本标签为该病人一年之后患病程度的定量指标。

from sklearn.datasets import load\_diabetes

from sklearn.tree import DecisionTreeRegressor

from sklearn.linear\_model import LinearRegression, Ridge

from sklearn.neighbors import KNeighborsRegressor

from sklearn.svm import SVR

from sklearn.ensemble import StackingRegressor

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

from sklearn.preprocessing import MinMaxScaler

from sklearn.preprocessing import StandardScaler

diabetes = load\_diabetes()

data = diabetes.data[:440]

target = diabetes.target[:440]

# 划分数据集

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

data,

target,

train\_size=380,

random\_state=1

)

实验中需要用到标准化和归一化，将它们集成到一个函数中，方便之后调用，函数的定义如下：

def Data\_preprocess(X\_train, X\_test):

MMS = MinMaxScaler()

MMS.fit(X\_train)

X1 = MMS.transform(X\_train)

X2 = MMS.transform(X\_test)

STD = StandardScaler()

STD.fit(X1)

X1 = STD.transform(X1)

X2 = STD.transform(X2)

return X1, X2

# 将数据标准归一化

X\_train, X\_test = Data\_preprocess(X\_train, X\_test)

（2）创建强弱学习器

使用sklearn.ensemble模块的StackingRegressor类可以创建一个两层的堆叠集成法回归器，其构造器的重要参数及含义如下：

**estimators：用一个列表指定基础学习器，列表的每个元素是一个基础学习器名称加基础学习器实例组成的元组。**

**final\_estimator：指定最终的元学习器。**

**cv：指定用折外预测法训练基础学习器时的拆分折数。默认为None，表示使用5折交叉。也可以传入一个其他整数。**

**n\_jobs：表示算法运行时并行训练的学习器个数，默认为1，可以指定为其他整数。取-1时代表所有学习器并行训练。**

下面创建一个k值为12的KNN回归学习器，一个最大深度为3的决策树学习器，一个线性内核的支持向量机回归学习器。将它们作为基础学习器，用一个线性回归学习器作为元学习器，创建一个堆叠法集成强学习器。

# 创建基础学习器

base\_estimators = []

knn = KNeighborsRegressor(n\_neighbors=12)

base\_estimators.append(("knn", knn))

tree = DecisionTreeRegressor(max\_depth=3 , random\_state=2)

base\_estimators.append(("tree", tree))

svr = SVR(kernel="linear")

base\_estimators.append(("svr", svr))

# 创建堆叠法集成学习器

model = StackingRegressor(estimators=base\_estimators,

final\_estimator=LinearRegression(),

cv=5)

（3）训练并评分

用训练数据对堆叠出的强学习器进行训练，并输出强学习器训练集、测试集上的评分。

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

print("Stacking Ensemble Train Score: ", model.score(X\_train, Y\_train))

print("Stacking Ensemble Test Score: ", model.score(X\_test, Y\_test))

### 3.3.3 分步骤演示

（1）导入依赖并处理数据

数据处理与上面实验一致，不再赘述。

from sklearn.datasets import load\_diabetes

from sklearn.tree import DecisionTreeRegressor

from sklearn.linear\_model import LinearRegression, Ridge

from sklearn.neighbors import KNeighborsRegressor

from sklearn.svm import SVR

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

from sklearn.model\_selection import KFold

from sklearn.preprocessing import MinMaxScaler

from sklearn.preprocessing import StandardScaler

diabetes = load\_diabetes()

data = diabetes.data[:440]

target = diabetes.target[:440]

# 划分数据集

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

data,

target,

train\_size=380,

random\_state=1

)

def Data\_preprocess(X\_train, X\_test):

MMS = MinMaxScaler()

MMS.fit(X\_train)

X1 = MMS.transform(X\_train)

X2 = MMS.transform(X\_test)

STD = StandardScaler()

STD.fit(X1)

X1 = STD.transform(X1)

X2 = STD.transform(X2)

return X1, X2

（2）创建模型

先创建第一层的基础学习器.

# 基础学习器

base\_estimators = []

knn = KNeighborsRegressor(n\_neighbors=12)

base\_estimators.append(knn)

tree = DecisionTreeRegressor(max\_depth=3 , random\_state=2)

base\_estimators.append(tree)

svr = SVR(kernel="linear")

base\_estimators.append(svr)

再创建一个元学习器，用于对基础学习器的预测结果进行拟合。

# 元学习器

meta\_estimator = LinearRegression()

（3）得到元学习器的训练数据和测试数据。

用sklearn.model\_selection模块的Kfold类创建k折预处理器，将训练数据划分为5折，用第i折之外的训练数据进行训练，用第i折数据进行预测，第j个基础学习器的预测结果放在第j列，得到元训练数据。

# 初始化数据

X\_meta\_train = np.ones((len(base\_estimators), len(X\_train)))

Y\_meta\_train = Y\_train

KF = KFold(n\_splits=5)

# 用每一折形成的数据切片为每个训练基础学习器学习模型

for train\_indices, pred\_indices in KF.split(X\_train):

train, to\_pred = Data\_preprocess(X\_train[train\_indices], X\_train[pred\_indices])

for i in range(len(base\_estimators)):

model = base\_estimators[i]

model.fit(train, Y\_train[train\_indices])

# 第i个元学习器的预测结果是元数据的第i列

X\_meta\_train[i][pred\_indices] = model.predict(to\_pred)

X\_meta\_train = X\_meta\_train.T # 转置

得到元学习器的测试集，顺便计算出基础学习器的评分。

X\_meta\_test = np.ones((len(base\_estimators), len(X\_test)))

# 初始化数据

Y\_meta\_test = Y\_test

X\_train, X\_test = Data\_preprocess(X\_train, X\_test)

scores = []

for i in range(len(base\_estimators)):

model = base\_estimators[i]

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

X\_meta\_test[i] = model.predict(X\_test)

scores.append(model.score(X\_test, Y\_test))

X\_meta\_test = X\_meta\_test.T

（5）训练元学习器，打印出各个学习器的评分

# 训练元学习器

meta\_estimator.fit(X\_meta\_train, Y\_meta\_train)

meta\_score = meta\_estimator.score(X\_meta\_test, Y\_meta\_test)

# 打印评分

for i in range(len(base\_estimators)):

print(base\_estimators[i].\_\_class\_\_.\_\_name\_\_ + " Test Score:")

print(scores[i])

print("Stacking Ensemble Test Score:\n", meta\_score)

### 3.3.4 实验结果

（1）用堆叠集成算法集成出的强学习器预测结果



图5 堆叠集成法实验结果

（2）手动堆叠集成得到的强学习器演示

各个学习器的评分如下：

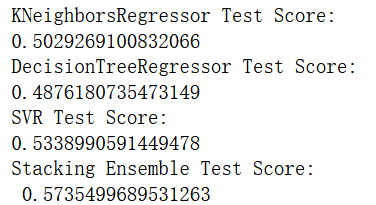


图6 各个学习器评分

可以看到，对于糖尿病数据集上的回归问题，K-邻近、决策树算法评分只有0.5左右，支持向量机回归算法也仅仅只有约0.53，但是运用堆叠法将这三个基础学习器简单地堆叠在一起，便得到了0.57的评分。

如果选用更合适的堆叠法模型，给基础学习器、元学习器分配更加合理的参数，增加更多基础学习器，模型的评分还可以进一步提高。