西北大学

智能信息系统综合实践
实验报告

题　　目：　Adaboost

年　　级：　2020

专　　业：　软件工程

姓　　名：　朱泽昊

一、题目（原题目）

•AdaBoost算法

1. 采用AdaBoost算法对一个自选数据集进行分类实验

2. 测试不同分类器和集成策略对性能的影响

3. 讨论和图示基分类器的偏差和方差

二、解题步骤（思路+代码）

Boosting： 用于分类，回归问题。从弱学习器开始加强，通过加权来进行训练。即一种迭代算法，通过不断使用一个弱学习器弥补前一个弱学习器的“不足”的过程，来串行地构造一个较强的学习器，这个强学习器能够使目标函数值足够小。Boosting 是一种串行算法，具体代表有 AdaBoost。

Adaboost: AdaBoost 是 Adaptive Boosting（自适应增强）的缩写，它的自适应在于：被前一个基本分类器误分类的样本的权值会增大，而正确分类的样本的权值会减小，并再次用来训练下一个基本分类器。同时，在每一轮迭代中，加入一个新的弱分类器，直到达到某个预定的足够小的错误率或预先指定的最大迭代次数再确定最后的强分类器。

Adaboost 迭代算法 3 步：

(1)初始化训练数据的权值分布。如果有 N 个样本，则每一个训练样本最开始时都被赋予相同的权值：1/N。
(2)训练弱分类器。具体训练过程中，如果某个样本点已经被准确地分类，那么在构造下一个训练集中，它的权值就被降低；相反，如果某个样本点没有被准确地分类，那么它的权值就得到提高。然后，权值更新过的样本集被用于训练下一个分类器，整个训练过程如此迭代地进行下去。
(3)将各个训练得到的弱分类器组合成强分类器。各个弱分类器的训练过程结束后，加大分类误差率小的弱分类器的权重，使其在最终的分类函数中起着较大的决定作用，而降低分类误差率大的弱分类器的权重，使其在最终的分类函数中起着较小的决定作用。换言之，误差率低的弱分类器在最终分类器中占的权重较大，否则较小。

Adaboost 算法流程:

步骤 1. 首先，初始化训练数据的权值分布。每一个训练样本最开始时都被赋予相同的权值：1/N。

$$D_1 = (w_{11}, w_{12} \cdots w_{1i} \cdots, w_{1N}), \quad w_{1i} = \frac{1}{N}, \quad i = 1,2,\cdots,N$$

步骤 2. 进行多轮迭代，用 m = 1,2, ..., M 表示迭代的第多少轮

a. 使用具有权值分布 Dm 的训练数据集学习，得到基本分类器（选取让误差率最低的阈值来设计基本分类器）：

$$G_m(x): \chi \to \{-1,+1\}$$

b. 计算 Gm(x)在训练数据集上的分类误差率

$$e_m = P(G_m(x_i) \neq y_i) = \sum_{i=1}^{N} w_{mi} I(G_m(x_i) \neq y_i)$$

由上述式子可知，Gm(x)在训练数据集上的 误差率 em 就是被 Gm(x)误分类样本的权之和。

c. 计算 Gm(x)的系数，am 表示 Gm(x)在最终分类器中的重要程度（目的：得到基本分类器在最终分类器中所占的权重。注：这个公式写成 am=1/2ln((1-em)/em) 更准确，因为底数是自然对数 e，故用 In，写成 log 容易让人误以为底数是 2 或别的底数，下同）：

$$\alpha_m = \frac{1}{2} \log \frac{1 - e_m}{e_m}$$

由上述式子可知，em <= 1/2 时，am >= 0，且 am 随着 em 的减小而增大，意味着分类误差率越小的基本分类器在最终分类器中的作用越大。

d. 更新训练数据集的权值分布（目的：得到样本的新的权值分布），用于下一轮迭代

$$D_{m+1} = (w_{m+1,1}, w_{m+1,2} \cdots w_{m+1,i} \cdots, w_{m+1,N}),$$
$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), \quad i = 1,2,\cdots,N$$

使得被基本分类器 Gm(x)误分类样本的权值增大，而被正确分类样本的权值减小。就这样，通过这样的方式，AdaBoost 方法能“重点关注”或“聚焦于”那些较难分的样本上。

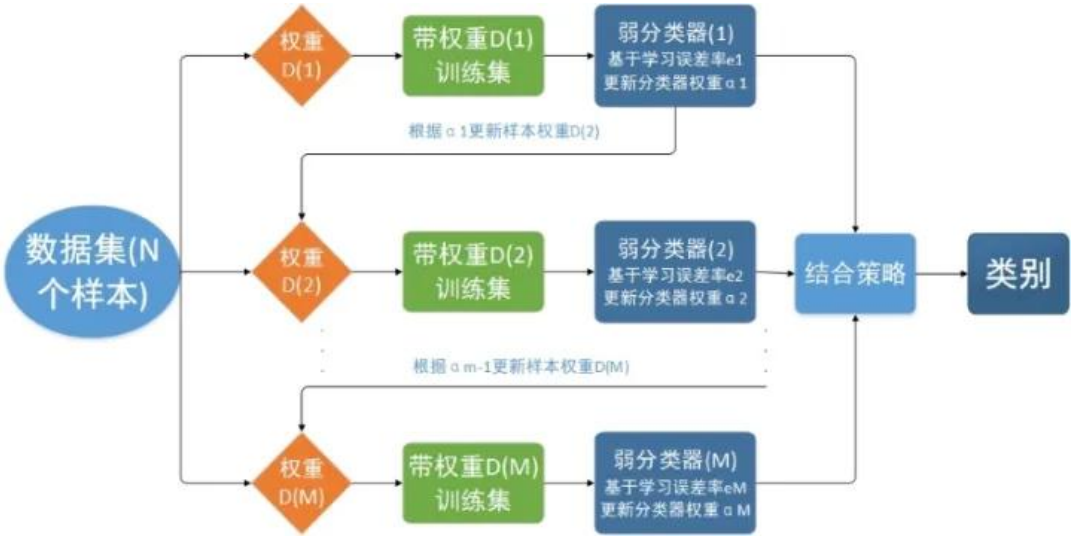
步骤 3. 组合各个弱分类器

$$f(x)=\sum_{m=1}^{M}\alpha_m G_m(x)$$

从而得到最终分类器，如下:

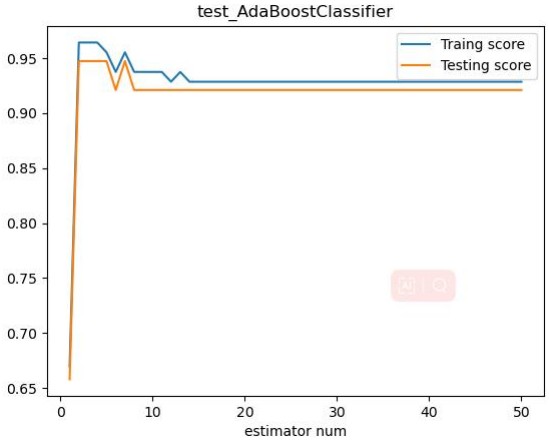
$$G(x)=sign(f(x))=sign\left(\sum_{m=1}^{M}\alpha_m G_m(x)\right)$$

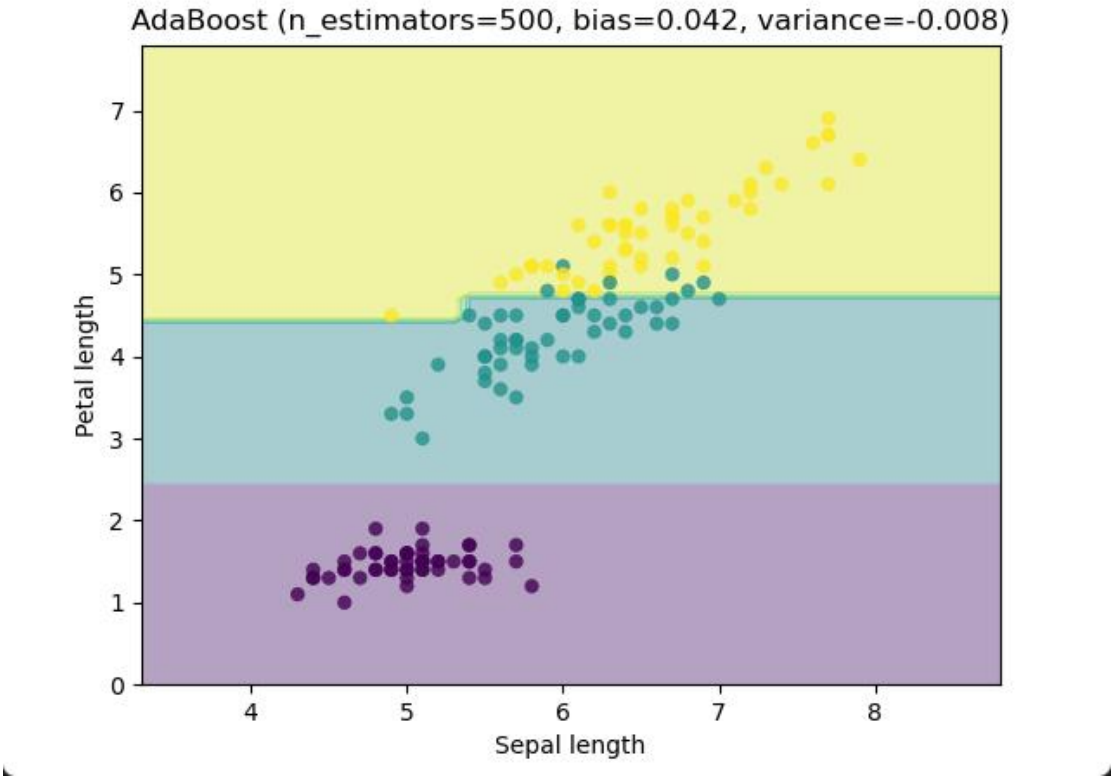
图解 adaboost:



1）本题选择 iris 数据集,参考以往实验方法随机划分测试集训练集

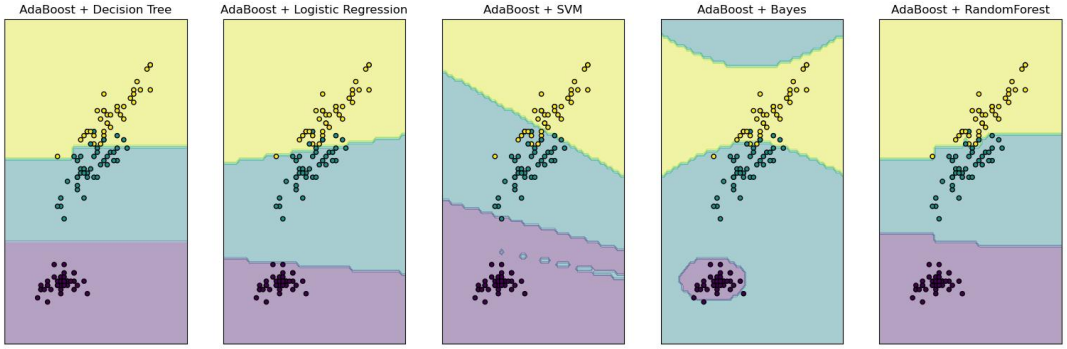
2）利用 sklearn 中提供的 AdaBoostClassifier，首先选用默认分类器 DecisionTreeClassifier。algorithm 默认为 SAMME.R.n_estimators 默认为 50。learning_rate 设置为 0.1。然后将这 50 个分类器或者说迭代次数从一开始,看分类器的增加对 Adaboost 算法精确度有多少提升，再继续使用这个分类器，迭代 500 轮，查看使用图示查看分类器性能，并计算出方差和偏差。

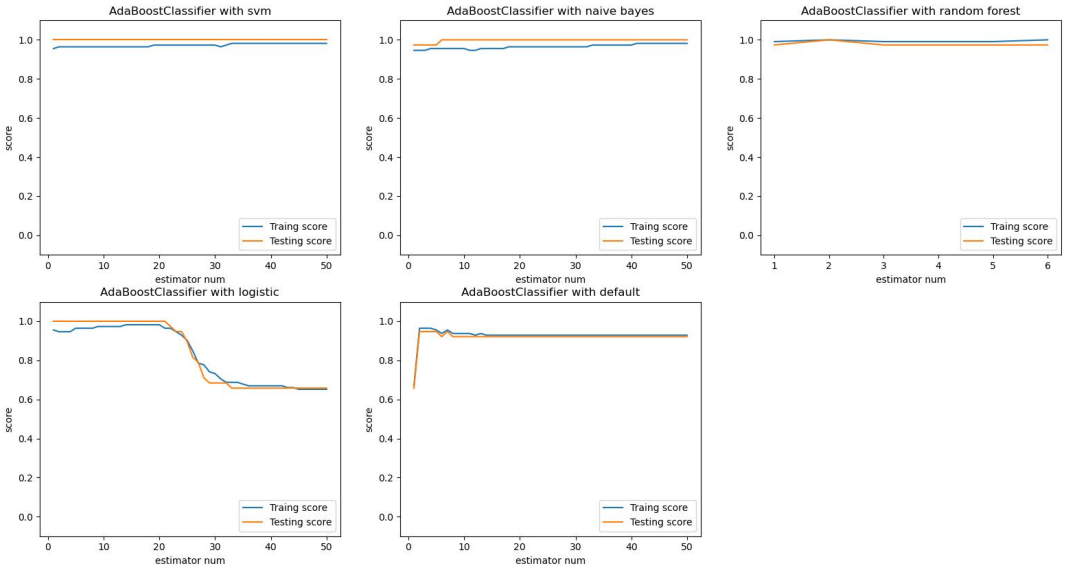





```
  warnings.warn(
Bias: 0.042
Variance: -0.008
```

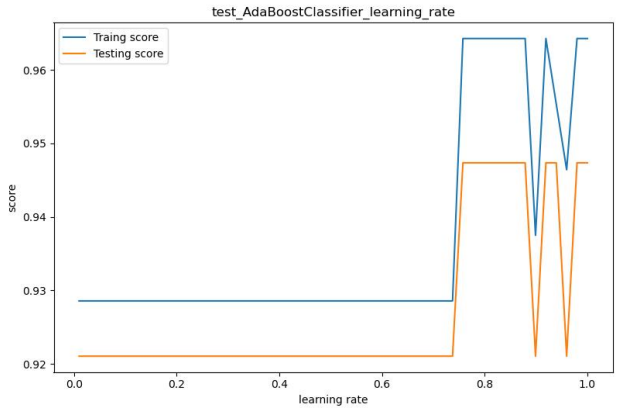
3）然后选用 svm,朴素贝叶斯,随机森林,逻辑回归分类器,进行训练,并且像上面一样画出图示



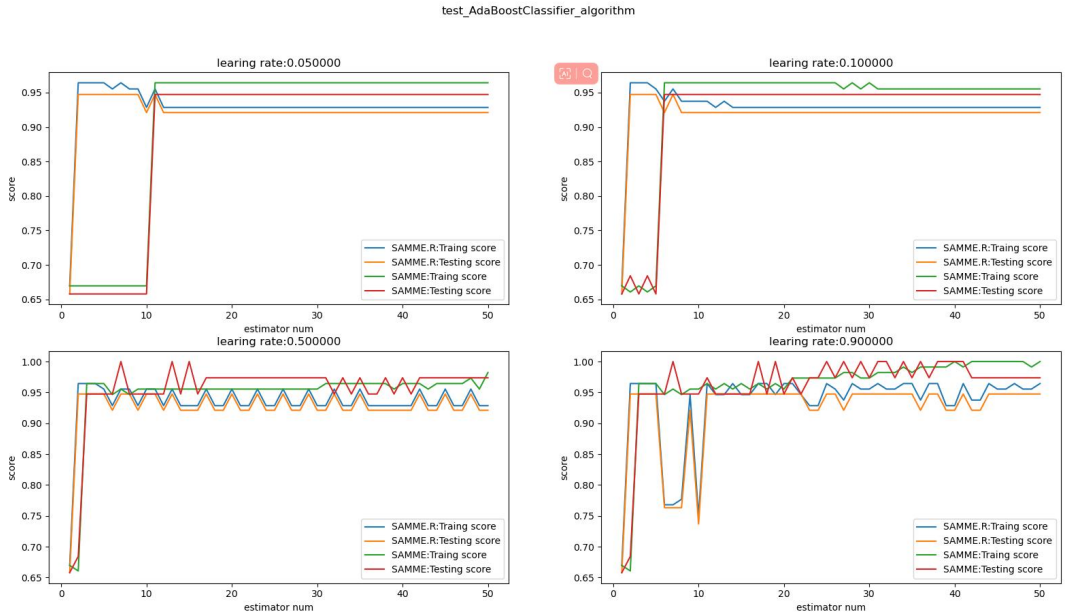



```
Tree bias: 0.042
Tree variance: -0.008
Logistic Regression bias: 0.058
Logistic Regression variance: 0.008
SVM bias: 0.117
SVM variance: -0.083
Bayes bias: 0.083
Bayes variance: -0.050
RandomForest bias: 0.008
RandomForest variance: -0.008
```

4）然后选用默认分类器，使用 500 轮迭代画图示 AdaBoostClassifier 的预测性能随学习率的影响



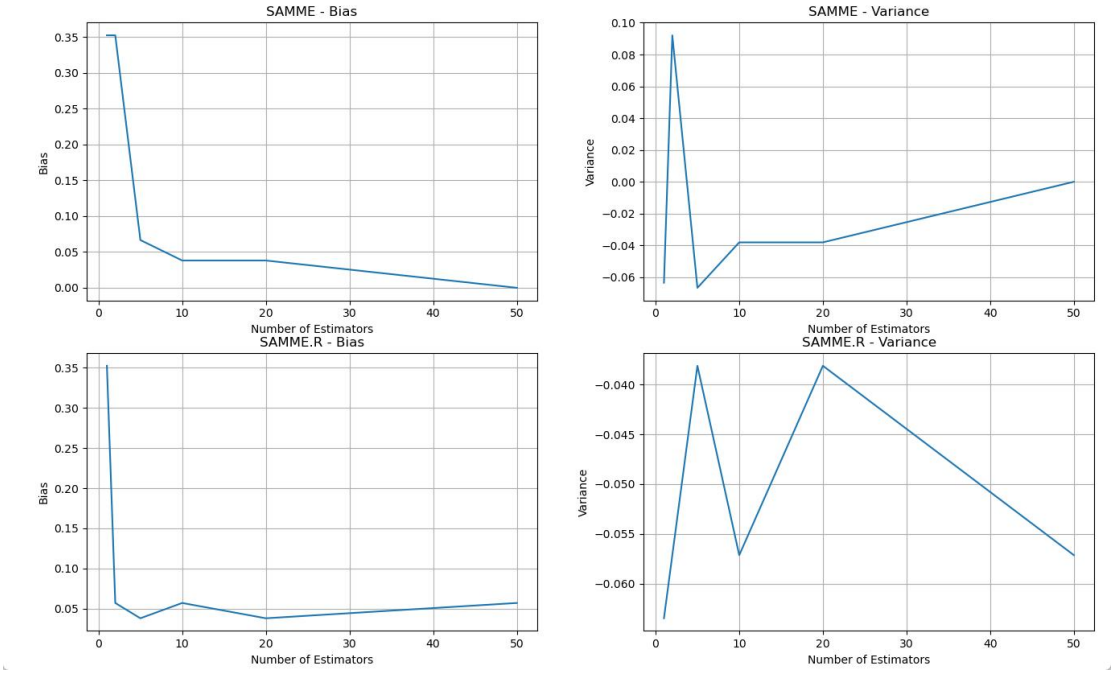
5）然后选用默认分类器，使用 500 轮迭代画图示 AdaBoostClassifier 的预测性能随随学习率和 algorithm 参数的影响,随机选择 4 个学习率,将 algorithm 设置为 SAMME 和 SAMME.R



6）使用不同的集成策略，使用不同的迭代次数，画出不同的集成策略下不同迭代次数的方差和偏差

SAMME（Stagewise Additive Modeling using a Multiclass Exponential loss function）：与指数损失加权类似，但用于多类分类问题。在每轮迭代中，分类器将被训练为区分正确类别和错误类别。每个分类器的输出将使用多类指数损失加权来更新样本权重。

SAMME.R（SAMME with Real valued predictions）：与 SAMME 类似，但分类器的输出被视为连续值，而不是离散类别。这允许使用回归算法来训练弱分类器，而不仅仅是分类算法。在每轮迭代中，样本权重将根据实际输出值和真实标签之间的误差进行更新。



在机器学习中，偏差和方差是两个重要的概念，用来描述模型在拟合数据时的表现。偏差描述了模型的拟合能力，即模型在训练集上的平均误差；而方差描述了模型的泛化能力，即模

型在不同的训练集上的预测结果的波动程度。
在集成学习中，我们使用多个基学习器来组成一个更强的模型，以提高模型的泛化能力。不同的集成方法具有不同的优缺点，因此需要根据具体情况来选择合适的集成方法。
偏差是指模型在训练集上表现不佳，也就是说，模型无法很好地拟合训练数据。这种情况通常发生在模型过于简单，无法很好地适应训练数据的复杂性时。例如，如果我们使用线性模型来拟合非线性数据，就会出现偏差较大的情况。
方差是指模型在测试集上表现不佳，也就是说，模型对于新数据的泛化能力较差。这种情况通常发生在模型过于复杂，过度拟合了训练数据的细节，而忽略了整体趋势时。例如，在决策树模型中，如果树的深度过大，就会出现过拟合的情况，使得模型在测试集上的表现不佳。
在选择集成方法时，需要根据具体情况选择合适的方法来平衡偏差和方差之间的关系，并提高模型的预测性能。同时，还需要注意控制模型的复杂度，避免过拟合和欠拟合的情况发生

代码：

（1）

```
def test_AdaBoostClassifier(*data):
    '''
    测试 AdaBoostClassifier 的用法，绘制 AdaBoostClassifier 的预测性能随基础分类器数量的影响

    :param data: 可变参数。它是一个元组，这里要求其元素依次为：训练样本集、测试样本集、训练样本的标记、测试样本的标记
    :return: None
    '''
    X_train,X_test,y_train,y_test=da (function) learning_rate: Any
    clf=ensemble.AdaBoostClassifier(learning_rate=0.1)
    clf.fit(X_train,y_train)
    ## 绘图
    fig=plt.figure()
    ax=fig.add_subplot(1,1,1)
    estimators_num=len(clf.estimators_)
    X=range(1,estimators_num+1)
    ax.plot(list(X),list(clf.staged_score(X_train,y_train)),label="Traing score")
    ax.plot(list(X),list(clf.staged_score(X_test,y_test)),label="Testing score")
    ax.set_xlabel("estimator num")
    ax.set_ylabel("score")
    ax.legend(loc="best")
    ax.set_title("test_AdaBoostClassifier")
    plt.show()
```

```python
#接下来，我们将使用AdaBoost分类器：
n_estimators = 500
tree = DecisionTreeClassifier(max_depth=1)
adaboost = AdaBoostClassifier(base_estimator=tree, n_estimators=n_estimators, learning_rate=1.0, random_state=42)
adaboost.fit(X_train, y_train)

#计算偏差和方差
y_pred_train = adaboost.predict(X_train)
train_error = 1 - accuracy_score(y_train, y_pred_train)

y_pred_test = adaboost.predict(X_test)
test_error = 1 - accuracy_score(y_test, y_pred_test)

bias = train_error
variance = test_error - train_error
print(f"Bias: {bias:.3f}")
print(f"Variance: {variance:.3f}")

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))

Z = adaboost.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.4)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, alpha=0.8, edgecolors='none')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, alpha=0.8, edgecolors='none')
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xlabel('Sepal length')
plt.ylabel('Petal length')
plt.title(f"AdaBoost (n_estimators={n_estimators}, bias={bias:.3f}, variance={variance:.3f})")
plt.show()
```

（2）

```python
def get_models():
    """Generate a library of base learners."""
    nb = GaussianNB()
    svc = SVC(C=100, probability=True)
    lr = LogisticRegression(C=100, random_state=SEED)
    rf = ensemble.RandomForestClassifier(n_estimators=10, max_features=3, random_state=SEED)
    nn = MLPClassifier((150, 3), early_stopping=False, random_state=SEED)
    models = {'svm': svc,
              'naive bayes': nb,
              'random forest': rf,
              'logistic': lr,
              'default':None
              }

    return models

def test_AdaBoostClassifier_base_classifier(*data):
    '''
    测试  AdaBoostClassifier 的预测性能随基础分类器数量和基础分类器的类型的影响

    :param data: 可变参数。它是一个元组，这里要求其元素依次为：训练样本集、测试样本集、训练样本的标记、测试样本的标记
    :return:  None
    '''

    X_train,X_test,y_train,y_test=data
    fig=plt.figure()
    models=get_models()
    for i, (name, model) in enumerate(models.items()):
        if name=='default':
            clf=ensemble.AdaBoostClassifier(
        else:
            clf=ensemble.AdaBoostClassifier(learning_rate=0.1,base_estimator=model)

        ax=fig.add_subplot(2,3,i+1)
        clf.fit(X_train,y_train)
        ## 绘图
        estimators_num=len(clf.estimators_)
        X=range(1,estimators_num+1)
        ax.plot(list(X),list(clf.staged_score(X_train,y_train)),label="Traing score")
        ax.plot(list(X),list(clf.staged_score(X_test,y_test)),label="Testing score")
        ax.set_xlabel("estimator num")
        ax.set_ylabel("score")
        ax.legend(loc="lower right")
        ax.set_ylim(-0.1,1.1)
        ax.set_title("AdaBoostClassifier with %s"%name)
    plt.show()
```

(function) learning_rate: Any

```python
# 使用决策树作为基分类器
tree = DecisionTreeClassifier(max_depth=1)
adaboost_tree = AdaBoostClassifier(estimator=tree, n_estimators=n_estimators, learning_rate=1.0, random_state=42)
adaboost_tree.fit(X_train, y_train)
y_pred_tree_train = adaboost_tree.predict(X_train)
train_error_tree = 1 - accuracy_score(y_train, y_pred_tree_train)
y_pred_tree_test = adaboost_tree.predict(X_test)
test_error_tree = 1 - accuracy_score(y_test, y_pred_tree_test)

# 使用逻辑回归作为基分类器
logreg = LogisticRegression(random_state=42)
adaboost_logreg = AdaBoostClassifier(estimator=logreg, n_estimators=n_estimators, learning_rate=1.0, random_state=42)
adaboost_logreg.fit(X_train, y_train)
y_pred_logreg_train = adaboost_logreg.predict(X_train)
train_error_logreg = 1 - accuracy_score(y_train, y_pred_logreg_train)
y_pred_logreg_test = adaboost_logreg.predict(X_test)
test_error_logreg = 1 - accuracy_score(y_test, y_pred_logreg_test)

# 使用支持向量机作为基分类器
svm = SVC(kernel='linear', random_state=42)
adaboost_svm = AdaBoostClassifier(estimator=svm, n_estimators=n_estimators, learning_rate=1.0, random_state=42,algorithm='SAMME')
adaboost_svm.fit(X_train, y_train)
y_pred_svm_train = adaboost_svm.predict(X_train)
train_error_svm = 1 - accuracy_score(y_train, y_pred_svm_train)
y_pred_svm_test = adaboost_svm.predict(X_test)
test_error_svm = 1 - accuracy_score(y_test, y_pred_svm_test)

# 使用朴素贝叶斯作为基分类器
nb = GaussianNB()
adaboost_bayes=AdaBoostClassifier(estimator=nb,n_estimators=n_estimators,learning_rate=0.1,random_state=42)
adaboost_bayes.fit(X_train, y_train)
y_pred_bayes_train = adaboost_bayes.predict(X_train)
train_error_bayes = 1 - accuracy_score(y_train, y_pred_bayes_train)
y_pred_bayes_test = adaboost_bayes.predict(X_test)
test_error_bayes = 1 - accuracy_score(y_test, y_pred_bayes_test)

# 使用随机森林作为基分类器
rf = RandomForestClassifier(n_estimators=10, max_features=3, random_state=42)
adaboost_rf=AdaBoostClassifier(estimator=rf,n_estimators=n_estimators,learning_rate=0.1,random_state=42)
adaboost_rf.fit(X_train, y_train)
y_pred_rf_train = adaboost_rf.predict(X_train)
train_error_rf = 1 - accuracy_score(y_train, y_pred_rf_train)
y_pred_rf_test = adaboost_rf.predict(X_test)
test_error_rf = 1 - accuracy_score(y_test, y_pred_rf_test)
```

```python
# 输出偏差和方差
print(f"Tree bias: {train_error_tree:.3f}")
print(f"Tree variance: {(test_error_tree - train_error_tree):.3f}")
print(f"Logistic Regression bias: {train_error_logreg:.3f}")
print(f"Logistic Regression variance: {(test_error_logreg - train_error_logreg):.3f}")
print(f"SVM bias: {train_error_svm:.3f}")
print(f"SVM variance: {(test_error_svm - train_error_svm):.3f}")
print(f"Bayes bias: {train_error_bayes:.3f}")
print(f"Bayes variance: {(test_error_bayes - train_error_bayes):.3f}")
print(f"RandomForest bias: {train_error_rf:.3f}")
print(f"RandomForest variance: {(test_error_rf - train_error_rf):.3f}")
```

```python
# 绘制决策边界和样本点
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))
fig, axs = plt.subplots(nrows=1, ncols=5, figsize=(12, 6))

# 决策树
Z_tree = adaboost_tree.predict(np.c_[xx.ravel(), yy.ravel()])
Z_tree = Z_tree.reshape(xx.shape)
axs[0].contourf(xx, yy, Z_tree, alpha=0.4)
axs[0].scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=20, edgecolor='k')
axs[0].set_title('AdaBoost + Decision Tree')

# 逻辑回归
Z_logreg = adaboost_logreg.predict(np.c_[xx.ravel(), yy.ravel()])
Z_logreg = Z_logreg.reshape(xx.shape)
axs[1].contourf(xx, yy, Z_logreg, alpha=0.4)
axs[1].scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=20, edgecolor='k')
axs[1].set_title('AdaBoost + Logistic Regression')

# 支持向量机
Z_svm = adaboost_svm.predict(np.c_[xx.ravel(), yy.ravel()])
Z_svm = Z_svm.reshape(xx.shape)
axs[2].contourf(xx, yy, Z_svm, alpha=0.4)
axs[2].scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=20, edgecolor='k')
axs[2].set_title('AdaBoost + SVM')

# Bayes
Z_bayes = adaboost_bayes.predict(np.c_[xx.ravel(), yy.ravel()])
Z_bayes = Z_bayes.reshape(xx.shape)
axs[3].contourf(xx, yy, Z_bayes, alpha=0.4)
axs[3].scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=20, edgecolor='k')
axs[3].set_title('AdaBoost + Bayes')

# 随机森林
Z_rf = adaboost_rf.predict(np.c_[xx.ravel(), yy.ravel()])
Z_rf = Z_rf.reshape(xx.shape)
axs[4].contourf(xx, yy, Z_rf, alpha=0.4)
axs[4].scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=20, edgecolor='k')
axs[4].set_title('AdaBoost + RandomForest')
```

```python
for ax in axs:
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
```

（3）

```python
def test_AdaBoostClassifier_learning_rate(*data):
    '''
    测试  AdaBoostClassifier 的预测性能随学习率的影响

    :param data: 可变参数。它是一个元组，这里要求其元素依次为：训练样本集、测试样本集、训练样本的标记、测试样本的标记
    :return: None
    '''
    X_train,X_test,y_train,y_test=data
    learning_rates=np.linspace(0.01,1)
    fig=plt.figure()
    ax=fig.add_subplot(1,1,1)
    traing_scores=[]
    testing_scores=[]
    for learning_rate in learning_rates:
        clf=ensemble.AdaBoostClassifier(learning_rate=learning_rate,n_estimators=500)
        clf.fit(X_train,y_train)
        traing_scores.append(clf.score(X_train,y_train))
        testing_scores.append(clf.score(X_test,y_test))
    ax.plot(learning_rates,traing_scores,label="Traing score")
    ax.plot(learning_rates,testing_scores,label="Testing score")
    ax.set_xlabel("learning rate")
    ax.set_ylabel("score")
    ax.legend(loc="best")
    ax.set_title("test_AdaBoostClassifier_learning_rate")
    plt.show()
```

（4）

```python
def test_AdaBoostClassifier_algorithm(*data):
    '''
    测试  AdaBoostClassifier 的预测性能随学习率和 algorithm 参数的影响

    :param data: 可变参数。它是一个元组，这里要求其元素依次为：训练样本集、测试样本集、训练样本的标记、测试样本的标记
    :return:  None
    '''
    X_train,X_test,y_train,y_test=data
    algorithms=['SAMME.R','SAMME']
    fig=plt.figure()
    learning_rates=[0.05,0.1,0.5,0.9]
    for i,learning_rate in enumerate(learning_rates):
        ax=fig.add_subplot(2,2,i+1)
        for i ,algorithm in enumerate(algorithms):
            clf=ensemble.AdaBoostClassifier(learning_rate=learning_rate,
                algorithm=algorithm)
            clf.fit(X_train,y_train)
            ## 绘图
            estimators_num=len(clf.estimators_)
            X=range(1,estimators_num+1)
            ax.plot(list(X),list(clf.staged_score(X_train,y_train)),
                label="%s:Traing score"%algorithms[i])
            ax.plot(list(X),list(clf.staged_score(X_test,y_test)),
                label="%s:Testing score"%algorithms[i])
        ax.set_xlabel("estimator num")
        ax.set_ylabel("score")
        ax.legend(loc="lower right")
        ax.set_title("learing rate:%f"%learning_rate)
    fig.suptitle("test_AdaBoostClassifier_algorithm")
    plt.show()
```

(variable) learning_rates: list[float]

（5）

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3, random_state=42)
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

estimators = [1, 2, 5, 10, 20, 50]
modes = ['SAMME', 'SAMME.R']
results = []

for mode in modes:
    for n_estimators in estimators:
        clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=n_estimators, algorithm=mode)
        clf.fit(X_train, y_train)
        train_error = 1 - clf.score(X_train, y_train)
        test_error = 1 - clf.score(X_test, y_test)
        bias = train_error
        variance = test_error - bias
        results.append((n_estimators, mode, bias, variance))
import matplotlib.pyplot as plt

fig, axs = plt.subplots(len(modes), 2, figsize=(10,10))

for i, mode in enumerate(modes):
    for j, metric in enumerate(['Bias', 'Variance']):
        values = [r[j+2] for r in results if r[1]==mode]
        axs[i,j].plot(estimators, values)
        axs[i,j].set_title(mode + ' - ' + metric)
        axs[i,j].set_xlabel('Number of Estimators')
        axs[i,j].set_ylabel(metric)
        axs[i,j].grid(True)

plt.tight_layout()
plt.show()
```

三、总结（心得体会）

Adaboost 是一种很强的集成学习算法，可以在不增加模型复杂度的情况下提高模型的准确率。

Adaboost 需要多个弱分类器来组成一个强分类器，因此需要选择合适的弱分类器，以保证整个模型的准确率。

Adaboost 需要进行多轮迭代训练，每轮迭代都会增加弱分类器的数量，因此需要控制迭代次数，以防止过拟合。

在使用 Adaboost 时，需要注意样本的不平衡性问题，如果样本分布不均匀，需要对样本进行重新采样或调整样本权重，以保证模型对少数类别的分类能力。

在选择 Adaboost 的集成策略时，需要根据具体问题选择合适的策略，以保证模型的性能。