In [ ]:

```
test_scores
```

In [ ]:

```
np.mean(train_scores, axis = 1)
```

In [ ]:

```
# 用散点图或者线图的方式来绘制曲线
plt.scatter(y = np.mean(train_scores, axis = 1), x = size)
plt.scatter(y = np.mean(test_scores, axis = 1), x = size)
```

In [ ]:

```
# 用散点图或者线图的方式来绘制曲线
sns.lineplot(y = np.mean(train_scores, axis = 1), x = size)
sns.lineplot(y = np.mean(test_scores, axis = 1), x = size)
```

In [ ]:

```
from sklearn.model_selection import learning_curve
from sklearn.svm import SVC

# 将原始数据打乱为随机顺序
X, y = boston.data, boston.target
indices = np.arange(y.shape[0])
np.random.shuffle(indices)
X, y = X[indices], y[indices]

size = np.linspace(0.1, 1, 10) # 以百分比的形式设定样本量

from sklearn import linear_model

reg = linear_model.LinearRegression()

train_sizes, train_scores, test_scores = learning_curve(
    reg, X, y, train_sizes = size, cv = 5)
train_scores
```

In [ ]:

```
plt.scatter(y = np.mean(train_scores, axis = 1), x = train_sizes)
plt.scatter(y = np.mean(test_scores, axis = 1), x = train_sizes)
```

9.5 实战练习

使用网格搜索功能重新拟合岭回归数据，比较两种实现方式的异同，包括分析结果、所需时间等。

当同时对多个参数进行调优时，该如何实现绘制验证曲线的功能？请思考解决方案，并用程序加以实现。

10 模型集成

10.1 投票分类器(Voting Classifier)

10.1.1 基于多个优化模型投票

class sklearn.ensemble.VotingClassifier(

```
estimators : list of (string, estimator) tuples，进行集成的分类器列表
voting = 'hard' : str，具体的投票策略
    'hard'，直接用各个模型的预测类别投票，平局时按模型顺序，靠后的有优先权
    'soft'，基于各个模型的各类别加权平均预测概率进行投票
        当各个预测器都进行过优化时，该方法较好

weights = None : shape = [n_classifiers]，各模型整合时的权重

n_jobs = 1
flatten_transform = None : 设定后续调用transform方法时输出矩阵的形状
    矩阵默认为(n_classifiers, n_samples, n_classes)
    如果voting = 'soft'且flatten_transform = True，
        矩阵为(n_samples, n_classifiers * n_classes)
```

)

VotingClassifier类的属性：

```
estimators_ : 分类器列表
classes_ : array-like, shape = [n_predictions]，类别标签
```

In [ ]:

```python
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, VotingClassifier

clf1 = LogisticRegression(random_state = 1)
clf2 = RandomForestClassifier(random_state = 1)
clf3 = GaussianNB()


eclf1 = VotingClassifier(estimators = [
    ('lr', clf1), ('rf', clf2), ('gnb', clf3)],
                         voting = 'hard').fit(iris.data, iris.target)
```

In [ ]:

```python
eclf1.estimators_
```

In [ ]:

```python
eclf1.predict(iris.data)[:10]
```

In [ ]:

```python
eclf1.predict_proba(iris.data)[:10]
```

In [ ]:

```
# 使用软投票方式进行模型集成
eclf2 = VotingClassifier(estimators = [
        ('lr', clf1), ('rf', clf2), ('gnb', clf3)],
                        voting = 'soft').fit(iris.data, iris.target)
eclf2.predict_proba(iris.data)[:10]
```

10.1.2 投票分类器与网格搜索的联合使用

在仍未完成模型参数调优时，可以将网格搜索与投票分类器联合使用，一次性的完成参数调优和模型集成工作。仍然调用sklearn.model_selection.GridSearchCV类，但是对参数设定略加调整：

```
estimator : 给出分类器列表
param_grid : 需要将参数名称设定为"模型标签__参数标签"格式，以便区分不同模型
```

In [ ]:

```
from sklearn.model_selection import GridSearchCV

clf1 = LogisticRegression(random_state = 1)
clf2 = RandomForestClassifier(random_state = 1)
clf3 = GaussianNB()

eclf = VotingClassifier(estimators=[('lr', clf1),
                                    ('rf', clf2),
                                    ('gnb', clf3)], voting = 'soft')

# 注意下面命令中对参数名称的命名方式
params = {'lr__C': [1.0, 10.0],
          'rf__n_estimators': [20, 200]}

# 在网格搜索中直接调用了投票分类器
grid = GridSearchCV(estimator = eclf, param_grid = params, cv = 5)
grid = grid.fit(iris.data, iris.target)
```

In [ ]:

```
pd.DataFrame(grid.cv_results_)
```

In [ ]:

```
grid.best_estimator_
```

In [ ]:

```
grid.best_params_
```

In [ ]:

```
grid.predict_proba(iris.data)[:5]
```

10.2 Bagging方法

10.2.1 分类模型的Bagging方法

class sklearn.ensemble.BaggingClassifier(

```
base_estimator = None : 准备拟合的基分类器，为None时设定为decision tree
n_estimators = 10 : 需要集成的基分类器数量
max_samples = 1.0 : 每次抽取的样本数量（int）/样本比例（float）
max_features = 1.0 : 每次抽取的特征数量（int）/特征比例（float）
bootstrap = True : 抽取样本时是否为可放回的bootstrap抽样
bootstrap_features = False : 抽取特征时是否为可放回的bootstrap抽样
oob_score = False : 是否使用OOB样本做误差估计
warm_start = False, n_jobs = 1, random_state = None, verbose = 0
```

)

BaggingClassifier类的属性：

```
base_estimator_ : 使用的基估计器（分类器）
estimators_ : list of estimators，所拟合的基分类器的集成列表
estimators_samples_ : list of arrays，每个基分类器拟合时使用的样本
estimators_features_ : list of arrays，每个基分类器拟合时使用的特征
classes_ : array of shape = [n_classes]，类别标签
n_classes_ : int or list，类别数
oob_score_ : float，使用OOB样本获得的训练集评分
oob_decision_function_ : array of shape = [n_samples, n_classes]
    使用OOB样本获得的预测结果
```

In [ ]:

```
from sklearn import datasets

iris = datasets.load_iris()
```

In [ ]:

```
from sklearn.ensemble import BaggingClassifier

bagging = BaggingClassifier(n_estimators = 100,
                            max_samples = 0.5, max_features = 0.5)
bagging.fit(iris.data, iris.target)
```

In [ ]:

```
bagging.estimators_samples_[0][:20]
```

In [ ]:

```
bagging.estimators_features_[:5]
```

In [ ]:

```
bagging.estimators_[:2]
```

In [ ]:

```
# 每一个基分类器都可以被单独使用
tree1 = bagging.estimators_[0].fit(iris.data, iris.target)
tree1.predict(iris.data)[:10]
```

In [ ]:

```
# 未指定oob_score = True时无此参数的估计值
bagging.oob_score_
```

In [ ]:

```
bagging = BaggingClassifier(n_estimators = 100, max_samples = 0.5,
                            oob_score = True)
bagging.fit(iris.data, iris.target)
bagging.oob_score_
```

In [ ]:

```
bagging.oob_decision_function_[:10]
```

In [ ]:

```
# 将集成模型用于预测
bagging.predict(iris.data)[:10]
```

10.2.2 回归模型的Bagging方法

class sklearn.ensemble.BaggingRegressor()

```
参数设置与BaggingClassifier完全相同
```

)

BaggingRegressor类的属性:

```
estimators_ : list of estimators, 所拟合的基分类器的集成列表
estimators_samples_ : list of arrays, 每个基分类器拟合时使用的样本
estimators_features_ : list of arrays, 每个基分类器拟合时使用的特征
oob_score_ : float, 使用OOB样本获得的训练集评分
oob_prediction_ : array of shape = [n_samples]
```

In [ ]:

```
from sklearn.ensemble import BaggingRegressor
from sklearn.linear_model import LinearRegression

bagging = BaggingRegressor(LinearRegression(), n_estimators = 100,
                           max_samples = 0.2, oob_score = True)
bagging.fit(boston.data, boston.target)
bagging.oob_score_
```

In [ ]:

```
# 给出每个案例的OOB预测结果
bagging.oob_prediction_[:10]
```

In [ ]:

```
# 将集成模型用于预测
bagging.predict(boston.data[:10])
```

10.3 随机森林

class sklearn.ensemble.RandomForestClassifier/RandomForestRegressor(

```
n_estimators = 10 : 森林中树的数量
criterion = 'gini'/'mse' : 树生长时使用的指标
    分类树为'gini'或'entropy'，回归树为'mse'或'mae'

max_features = 'auto' : int/float/string/None，搜索分支时考虑的特征数
    'auto'/'sqrt', max_features = sqrt(n_features)
    'log2', max_features = log2(n_features)
    None, max_features = n_features

max_depth = None : 树生长的最大高度
min_samples_split = 2 : 节点允许进一步分枝时的最低样本数
min_samples_leaf = 1 : 叶节点的最低样本量
min_weight_fraction_leaf = 0.0 : 有权重时叶节点的最低权重分值
max_leaf_nodes = None : 最高叶节点数量
min_impurity_decrease = 0.0 : 分枝时需要的最低信息量下降量

bootstrap = True : 是否使用可放回的bootstap抽样
oob_score = False : 是否使用OOB方式计算模型准确率

n_jobs = 1, random_state = None
verbose = 0, warm_start = False, class_weight = None
```

)

RandomForestClassifier/RandomForestRegressor类共有的属性：

```
estimators_ : 森林中所有基模型的列表
feature_importances_ : array of shape = [n_features]，各特征重要性
n_features_ : int
n_outputs_ : int，因变量数量
oob_score_ : float，使用OOB方式得到的训练集评分
```

RandomForestClassifier独有的属性：

```
classes_ : array of shape = [n_classes] or a list of such arrays
n_classes_ : int or list
oob_decision_function_ : array of shape = [n_samples, n_classes]
    使用OOB方式计算出的各案例的类别预测概率
```

RandomForestRegressor类独有的属性：

```
oob_prediction_ : array of shape = [n_samples]
```

RandomForestClassifier/RandomForestRegressor类独有的方法：

```
apply(X) : 对X拟合模型，并返回所属的节点索引
decision_path(X) : 返回对应案例的决策路径
```

10.3.1 随机森林分类实例

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(500, oob_score = True)
rfc.fit(iris.data, iris.target)
```

In [ ]:

```
print(len(rfc.estimators_))
rfc.estimators_[0]
```

In [ ]:

```
rfc.estimators_[0].predict(iris.data[:5])
```

In [ ]:

```
rfc.estimators_[0].feature_importances_
```

In [ ]:

```
rfc.feature_importances_
```

In [ ]:

```
rfc.oob_score_
```

In [ ]:

```
rfc.oob_decision_function_[:5]
```

In [ ]:

```
rfc.predict_proba(iris.data[:5])
```

10.3.2 随机森林回归实例

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor

rfr = RandomForestRegressor(500, oob_score = True)
rfr.fit(boston.data, boston.target)
```

In [ ]:

```
rfr.feature_importances_
```

In [ ]:

```
rfr.oob_score_
```

In [ ]:

```
rfr.oob_prediction_[:5]
```

In [ ]:

```
rfr.predict(boston.data[:5])
```

10.4 AdaBoost

class sklearn.ensemble.AdaBoostClassifier(

```
base_estimator = DecisionTreeClassifier : 使用的基估计器
n_estimators = 50 : integer, 迭代次数
learning_rate = 1. : float, 学习率，用于减少每个分类器的贡献
algorithm = 'SAMME.R' : {'SAMME', 'SAMME.R'}, 具体算法
    'SAMME.R' : real boosting algorithm, 一般比SAMME更快拟合
    'SAMME' : SAMME discrete boosting algorithm
random_state = None : int, 随机种子
```

)

class sklearn.ensemble.AdaBoostRegressor(

```
base_estimator = DecisionTreeRegressor, n_estimators = 50
learning_rate = 1.0, random_state = None
loss = 'linear' {'linear', 'square', 'exponential'}
```

)

AdaBoostClassifier/AdaBoostRegressor类的属性：

```
classes_ : array of shape = [n_classes], 类标签
n_classes_ : int, 类别数

estimators_ : list of classifiers
estimator_weights_ : array of floats, 每个分类器的权重
estimator_errors_ : array of floats, 每个分类器的误差
feature_importances_ : array of shape = [n_features], 各属性的重要性
```

AdaBoostClassifier/AdaBoostRegressor类独有的方法：

```
staged_predict_proba(X) : AdaBoostClassifier类的方法，分步的预测概率

staged_predict(X) : 分步的预测值
staged_score(X, y[, sample_weight]) : 分步的评分
```

In [ ]:

```
from sklearn.ensemble import AdaBoostClassifier

adac = AdaBoostClassifier()
adac.fit(iris.data, iris.target)
```

In [ ]:

```
adac.estimators_[0]
```

In [ ]:

```
# 列出基估计器的大小
adac.estimators_[0].tree_.node_count
```

In [ ]:

```
adac.estimator_errors_
```

In [ ]:

```
adac.feature_importances_
```

In [ ]:

```
# 注意staged系列函数生成的是generator
for item in adac.staged_predict_proba(iris.data[:5]):
    print(item)
```

In [ ]:

```
adac.predict_proba(iris.data[:5])
```

10.5 梯度提升树（GBDT）

class sklearn.ensemble.GradientBoostingClassifier/GradientBoostingRegressor(

```
    loss = 'deviance'/'ls' : 损失函数的设定
        分类：{'deviance', 'exponential'}
        回归：{'ls', 'lad', 'huber', 'quantile'}
    subsample = 1.0 : 用于训练每个基分类器的样本比例，为1.0时显然无OOB输出
        降低该比例可以减少方差，但同时增大偏差

    criterion = 'friedman_mse' : 分枝质量的评价指标
        'friedman_mse', 'mse', 'mae' ，一般认为friedman_mse的效果最好

    learning_rate = 0.1, n_estimators = 100,
    min_samples_split = 2, min_samples_leaf = 1
    min_weight_fraction_leaf = 0.0, max_depth = 3
    min_impurity_decrease = .0, min_impurity_split = None, init = None
    random_state = None, max_features = None, verbose = 0
    max_leaf_nodes = None, warm_start = False

    presort = 'auto' : Bool，是否对数据做预排序以加速拟合
```

)

GradientBoostingClassifier/GradientBoostingRegressor类的属性：

```
    feature_importances_ : array, shape = [n_features]
    oob_improvement_ : array, shape = [n_estimators]
    train_score_ : array, shape = [n_estimators], deviance值的迭代记录
    loss_ : LossFunction
    init : BaseEstimator
    estimators_ : ndarray of DecisionTreeRegressor
```

GradientBoostingClassifier/GradientBoostingRegressor类独有的方法：

```
    staged_predict_proba(X) : 分步的预测概率

    staged_predict(X) : 分步的预测值
    staged_decision_function : 分步的决策函数
```

In [ ]:

```
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier()
gbc.fit(iris.data, iris.target)
```

In [ ]:

```
# 注意基模型的数量 = 类别数量
gbc.estimators_[0]
```

In [ ]:

```
gbc.feature_importances_
```

In [ ]:

```
gbc.train_score_[:10]
```

In [ ]:

```
plt.plot(gbc.train_score_)
```

In [ ]:

```
gbc.predict_proba(iris.data[:5])
```

In [ ]:

```
gbc.decision_function(iris.data[:5])
```

In [ ]:

```
for item in gbc.staged_predict_proba(iris.data[:5]):
    print(item)
```

10.6 实战练习

有无可能对同一个数据使用随机森林、Adaboost、GBDT这三种模型进行投票分类？为什么？

自行设计一个Stacking方式的模型集成方法，并用手边合适的数据加以尝试。