模型评估

一、数据集切分

1. 数据集切分的通用参数：
   - `random_state`：一个整数或者一个`RandomState`实例，或者`None`，指定随机数种子。
     - 如果为整数，则它指定了随机数生成器的种子。
     - 如果为`RandomState`实例，则指定了随机数生成器。
     - 如果为`None`，则使用默认的随机数生成器。
   - `X`：样本集合。通常是一个`numpy array`，每行代表一个样本，每列代表一个特征。
   - `y`：样本的标签集合。它与`X`的每一行相对应。
   - `groups`：样本的分组标记集合。它与`X`的每一行相对应，用于训练集、测试集的拆分。

1.1 train_test_split

1. `train_test_split`用于将数据集切分成训练集和测试集，其原型为：

   ```
   sklearn.model_selection.train_test_split(*arrays, **options)
   ```

   返回值：一个列表，依次给出一个或者多个数据集的划分的结果。每个数据集都划分为两部分：训练集、测试集。

   参数：
   - `*arrays`：一个或者多个数组，代表被拆分的一些数据集。
   - `test_size`：一个浮点数，整数或者`None`，指定测试集大小。
     - 浮点数：必须是0.0 到 1.0 之间的数，代表测试集占原始数据集的比例。
     - 整数：代表测试集大小。
     - `None`：如果训练集大小也指定为`None`，则`test_size`设为 0.25。
   - `train_size`：一个浮点数，整数或者`None`，指定训练集大小。
     - 浮点数：必须是0.0 到 1.0 之间的数，代表训练集占原始数据集的比例。
     - 整数：代表训练集大小。
     - `None`：如果测试集大小也指定为`None`，则`test_size`设为 0.75。
   - `random_state`：指定随机数种子。
   - `stratify`：一个数组对象或者`None`。如果它不是`None`，则原始数据会分层采样，采样的标记数组就由该参数指定。

1.2 KFold

1. `KFold`类实现了数据集的 k 折交叉切分。其原型为：

```
class sklearn.model_selection.KFold(n_splits=3, shuffle=False, random_state=None)
```

- `n_splits`：一个整数，即 k（要求该整数值大于等于 2）。
- `shuffle`：一个布尔值。如果为`True`，则在切分数据集之前先混洗数据集。
- `random_state`：指定随机数种子。

2. 方法：
   - `get_n_splits([X, y, groups])`：返回`n_splits`参数。

     参数：其参数都被忽略，用于保持接口的兼容性。
   - `split(X[, y, groups])`：切分数据集为训练集和测试集。返回测试集的样本索引、训练集的样本索引。

     参数：
     - `X`为训练数据集，形状为`(n_samples,n_features)`
     - `y`为标记信息，形状为`(n_samples,)`
     - `groups`：样本的分组标记，用于拆分。
3. `KFold`首先将`0~(n-1)`之间的整数从前到后均匀划分成`n_splits`份。每次迭代时依次挑选一份作为测试集样本的下标。
   - 如果`shuffle=True`， 则按顺序划分。
   - 如果`shuffle=False`， 则按随机划分。

1.3 StratifiedKFold

1. `StratifiedKFold`类实现了数据集的分层采样 k 折交叉切分。其原型为：

```
class sklearn.model_selection.StratifiedKFold(n_splits=3, shuffle=False,
random_state=None)
```

   参数：参考`KFold`。
2. 方法：参考`KFold`。
3. `StratifiedKFold`的用法类似于`KFold`，但是`StratifiedKFold`执行的是分层采样：保证训练集、测试集中各类别样本的比例与原始数据集中相同。

1.4 LeaveOneOut

1. `LeaveOneOut`类实现了数据集的留一法拆分(简称`LOO`)。它是个生成器，其原型为：

```
class sklearn.model_selection.LeaveOneOut(n)
```

   - `n`：一个整数，表示数据集大小。
2. `LeaveOneOut`的用法很简单。它每次迭代时，依次取`0,1,...(n-1)`作为测试集样本的下标。

```
from sklearn.model_selection import LeaveOneOut
for train_index,test_index in LeaveOneOut(len(y)):
    #train_index 保存训练集样本下标，test_index 保存测试集样本下标
```

1.5 cross_val_score

1. 便利函数`cross_val_score`对`estimator`执行`k`折交叉验证。其原型为：

```
sklearn.model_selection.cross_val_score(estimator, X, y=None, scoring=None, cv=None,
n_jobs=1, verbose=0, fit_params=None, pre_dispatch='2*n_jobs')
```

返回值：返回一个浮点数的数组。每个浮点数都是针对某次 k 折交叉的数据集上，`estimator`预测性能得分。

参数：

- `estimator`：指定的学习器，该学习器必须有`.fit`方法来进行训练。
- `X`：样本集合。通常是一个`numpy array`，每行代表一个样本，每列代表一个特征。
- `y`：样本的标签集合。它与`X`的每一行相对应。
- `groups`：样本的分组标记集合。它与`X`的每一行相对应，用于训练集、测试集的拆分。
- `scoring`：一个字符串，或者可调用对象，或者`None`，它指定了评分函数。

  如果为可调用对象，则参数为`estimator, X, y`，返回值为一个浮点数表示预测能力得分。

  如果为`None`，则采用`estimator`学习器的`.score`方法。

  如果为字符串，则可以为下列字符串：

  - `'accuracy'`：采用的是`metrics.accuracy_score`评分函数。
  - `'average_precision'`：采用的是`metrics.average_precision_score`评分函数。
  - `f1`系列：采用的是`metrics.f1_score`评分函数。包括：
    - `'f1'`：`f1`值作为评分。用于二分类问题。
    - `'f1_micro'`：微`f1`值作为评分。用于多分类问题。
    - `'f1_macro'`：宏`f1`值作为评分。用于多分类问题。
    - `'f1_weighted'`：加权`f1`值作为评分。
    - `'f1_samples'`：多标签`f1`值作为评分。
  - `'log_loss'`：采用的是`metrics.log_loss`评分函数。
  - `precision`系列：采用的是`metrics.precision_score`评分函数。

    具体形式类似`f1`系列。
  - `recall`系列：采用的是`metrics.recall_score`评分函数。

    具体形式类似`f1`系列。
  - `'roc_auc'`：采用的是`metrics.roc_auc_score`评分函数。
  - `'adjusted_rand_score'`：采用的是`metrics.adjusted_rand_score`评分函数。
  - `'mean_absolute_error'`：采用的是`metrics.mean_absolute_error`评分函数。

        - `'mean_squared_error''`：采用的是`metrics.mean_squared_error`评分函数。
        - `'median_absolute_error'`：采用的是`metrics.median_absolute_error`评分函数。
        - `'r2'`：采用的是`metrics.r2_score`评分函数 。
    - `cv`：一个整数、k 折交叉生成器、一个迭代器、或者`None`，指定`k`折交叉参数。
        - 如果为`None`，则使用默认的 3 折交叉生成器。
        - 如果为整数，则指定了 k 折交叉生成器的 k 值。
        - 如果为 k 折交叉生成器，则直接指定了 k 折交叉生成器。
        - 如果为迭代器，则迭代器的结果就是数据集划分的结果。
    - `fit_params`：一个字典，指定了`estimator`执行`.fit`方法时的关键字参数。
    - `n_jobs`：一个整数，指定并行性。
    - `verbose`：一个整数，用于控制输出日志。
    - `pre_dispatch`：一个整数或者字符串或者`None`，用于控制并行执行时，分发的总的任务数量。
        - 如果为`None`，则所有的`job`立即创建并派生。
        - 如果为整数，则它指定了立即派生的`job`的数量。
        - 如果为字符串，则指定了`n_jobs`的表达式。如`'2*n_jobs'`表示立即派生2倍`n_jobs`数量的`job`。
2. 之所以称`cross_val_score`为便利函数，是因为完全可以凭借现有的函数手动完成这个功能，步骤为：
    - k 折交叉划分数据集，对每次划分结果执行：
        - 在训练集上训练`estimator`。
        - 用训练好的`estimator`预测测试集，返回测试性能得分。
    - 收集所有的测试性能得分，放入一个数组并返回。

二、性能度量

1. 在`scikit-learn`中有三种方法来评估`estimator`的预测性能：
    - `estimator`的`.score`方法。
    - 通过使用`model_selection`中的模型评估工具来评估，如`model_selection.cross_val_score`等方法。
    - 通过`scikit-learn`的`metrics`模块中的函数来评估`estimator`的预测性能。这里重点讲解这些函数。
2. `metrics`模块中的性能评价函数的通用参数：
    - `y_true`：一个数组，给出了真实的标记集合。
    - `y_pred`：一个数组，给出了预测的标记集合。
    - `sample_weight`：一个浮点数，给出了样本权重。默认每个样本的权重为 1。

2.1 分类问题性能度量

2.1.1 accuracy_score

1. `accuracy_score`函数用于计算分类结果的准确率，其原型为：

```
sklearn.metrics.accuracy_score(y_true, y_pred, normalize=True, sample_weight=None)
```

返回值：如果`normalize`为`True`，则返回准确率；如果`normalize`为`False`，则返回正确分类的数量。

参数：

- `y_true`：真实的标记集合。
- `y_pred`：预测的标记集合。
- `normalize`：一个布尔值，指示是否需要归一化结果。
  - 如果为`True`，则返回分类正确的比例（准确率）。
  - 如果为`False`，则返回分类正确的样本数量。
- `sample_weight`：样本权重，默认每个样本的权重为 1 。

2.1.2 precision_score

1. `precision_score`函数用于计算分类结果的查准率，其原型为：

```
sklearn.metrics.precision_score(y_true, y_pred, labels=None, pos_label=1,
average='binary', sample_weight=None)
```

返回值：查准率。即预测结果为正类的那些样本中，有多少比例确实是正类。

参数：

- `y_true`：真实的标记集合。
- `y_pred`：预测的标记集合。
- `labels`：一个列表。当`average`不是`'binary'`时使用。
  - 对于多分类问题，它指示：将计算哪些类别。不在`labels`中的类别，计算`macro precision`时其成分为 0 。
  - 对于多标签问题，它指示待考察的标签的索引。
  - 除了`average=None`之外，`labels`的元素的顺序也非常重要。
  - 默认情况下，`y_true`和`y_pred`中所有的类别都将被用到。
- `pos_label`：一个字符串或者整数，指定哪个标记值属于正类。
  - 如果是多分类或者多标签问题，则该参数被忽略。
  - 如果设置`label=[pos_label]`以及`average!='binary'`则会仅仅计算该类别的`precision`。
- `average`：一个字符串或者`None`，用于指定二分类或者多类分类的`precision`如何计算。
  - `'binary'`：计算二类分类的`precision`。 此时由`pos_label`指定的类为正类，报告其`precision`。

    它要求`y_true、y_pred`的元素都是`0,1`。
  - `'micro'`：通过全局的正类和父类，计算`precision`。
  - `'macro'`：计算每个类别的`precision`，然后返回它们的均值。
  - `'weighted'`：计算每个类别的`precision`，然后返回其加权均值，权重为每个类别的样本数。
  - `'samples'`：计算每个样本的`precision`，然后返回其均值。

    该方法仅仅对于多标签分类问题有意义。

        - `None`：计算每个类别的`precision`，然后以数组的形式返回每个`precision`。
    - `sample_weight`：样本权重，默认每个样本的权重为 1

2.1.3 recall_score

1. `recall_score`函数用于计算分类结果的查全率，其原型为：

```
sklearn.metrics.recall_score(y_true, y_pred, labels=None, pos_label=1,
average='binary', sample_weight=None)
```

返回值：查全率。即真实的正类中，有多少比例被预测为正类。

参数：参考`precision_score`。

2.1.4 f1_score

1. `f1_score`函数用于计算分类结果的 F_1 值，其原型为：

```
sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1,
average='binary', sample_weight=None)
```

返回值：F_1 值。即查准率和查全率的调和均值。

参数：参考`precision_score`。

2.1.5 fbeta_score

1. `fbeta_score`函数用于计算分类结果的 F_β 值，其原型为：

```
sklearn.metrics.fbeta_score(y_true, y_pred, beta, labels=None, pos_label=1,
average='binary', sample_weight=None)
```

返回值：F_β 值。

参数：

- `beta`：β 值
- 其它参数参考`precision_score`。

2.1.6 classification_report

1. `classification_report`函数以文本方式给出了分类结果的主要预测性能指标。其原型为：

```
sklearn.metrics.classification_report(y_true, y_pred, labels=None, target_names=None,
sample_weight=None, digits=2)
```

返回值：一个格式化的字符串，给出了分类评估报告。

参数：

- `y_true`：真实的标记集合。

- `y_pred`：预测的标记集合。
- `labels`：一个列表，指定报告中出现哪些类别。
- `target_names`：一个列表，指定报告中类别对应的显示出来的名字。
- `digits`：用于格式化报告中的浮点数，保留几位小数。
- `sample_weight`：样本权重，默认每个样本的权重为 1

2. 分类评估报告的内容如下，其中：
   - `precision` 列：给出了查准率。它依次将类别 0 作为正类，类别 1 作为正类...
   - `recall` 列：给出了查全率。它依次将类别 0 作为正类，类别 1 作为正类...
   - `recall` 列：给出了 F_1 值。
   - `support` 列：给出了该类有多少个样本。
   - `avg / total` 行：
     - 对于 `precision,recall,recall`，给出了该列数据的算术平均。
     - 对于 `support` 列，给出了该列的算术和（其实就等于样本集总样本数量）。

```
Classification Report:
precision    recall  f1-score   support
class_0       0.62      1.00      0.77         5
class_1       1.00      0.40      0.57         5
avg / total       0.81      0.70      0.67        10
```

2.1.7 confusion_matrix

1. `confusion_matrix` 函数给出了分类结果的混淆矩阵。其原型为：

```
sklearn.metrics.confusion_matrix(y_true, y_pred, labels=None)
```

返回值：一个格式化的字符串，给出了分类结果的混淆矩阵。

参数：参考 `classification_report` 。

2. 混淆矩阵的内容如下，其中 $C_{i,j}$ 表示真实标记为 i 但是预测为 j 的样本的数量。

```
Confusion Matrix:
[[5 0]
[3 2]]
```

2.1.8 precision_recall_curve

1. `precision_recall_curve` 函数用于计算分类结果的 `P-R` 曲线。其原型为：

```
sklearn.metrics.precision_recall_curve(y_true, probas_pred, pos_label=None,
sample_weight=None)
```

返回值：一个元组，元组内的元素分别为：

- `P-R`曲线的查准率序列。该序列是递增序列，序列第 `i` 个元素是当正类概率的判定阈值为 `thresholds[i]` 时的查准率。
- `P-R`曲线的查全率序列。该序列是递减序列，序列第 `i` 个元素是当正类概率的判定阈值为 `thresholds[i]` 时的查全率。
- `P-R`曲线的阈值序列 `thresholds`。该序列是一个递增序列，给出了判定为正例时的正类概率的阈值。

参数：

- `y_true`：真实的标记集合。
- `probas_pred`：每个样本预测为正类的概率的集合。
- `pos_label`：正类的类别标记。
- `sample_weight`：样本权重，默认每个样本的权重为 1。

2.1.9 roc_curve

1. `roc_curve`函数用于计算分类结果的`ROC`曲线。其原型为：

```
sklearn.metrics.roc_curve(y_true, y_score, pos_label=None, sample_weight=None,
drop_intermediate=True)
```

返回值：一个元组，元组内的元素分别为：

- `ROC`曲线的 FPR 序列。该序列是递增序列，序列第 `i` 个元素是当正类概率的判定阈值为 `thresholds[i]` 时的假正例率。
- `ROC`曲线的 TPR 序列。该序列是递增序列，序列第 `i` 个元素是当正类概率的判定阈值为 `thresholds[i]` 时的真正例率。
- `ROC`曲线的阈值序列 `thresholds`。该序列是一个递减序列，给出了判定为正例时的正类概率的阈值。

参数：

- `y_true`：真实的标记集合。
- `y_score`：每个样本预测为正类的概率的集合。
- `pos_label`：正类的类别标记。
- `sample_weight`：样本权重，默认每个样本的权重为 1。
- `drop_intermediate`：一个布尔值。如果为 `True`，则抛弃某些不可能出现在 `ROC` 曲线上的阈值。

2.1.10 roc_auc_score

1. `roc_auc_score`函数用于计算分类结果的`ROC`曲线的面积`AUC`。其原型为：

```
sklearn.metrics.roc_auc_score(y_true, y_score, average='macro', sample_weight=None)
```

返回值：`AUC`值。

参数：参考 `roc_curve` 。

2.2 回归问题性能度量

2.2.1 mean_absolute_error

1. `mean_absolute_error`函数用于计算回归预测误差绝对值的均值(`mean absolute error:MAE`)，其原型为：

```
sklearn.metrics.mean_absolute_error(y_true, y_pred, sample_weight=None,
multioutput='uniform_average')
```

返回值：预测误差绝对值的均值。

参数：

- `y_true`：真实的标记集合。
- `y_pred`：预测的标记集合。
- `multioutput`：指定对于多输出变量的回归问题的误差类型。可以为：
  - `'raw_values'`：对每个输出变量，计算其误差 。
  - `'uniform_average'`：计算其所有输出变量的误差的平均值。
- `sample_weight`：样本权重，默认每个样本的权重为 1。

2.2.2 mean_squared_error

1. `mean_squared_error` 函数用于计算回归预测误差平方的均值(`mean square error:MSE`)，其原型为：

```
sklearn.metrics.mean_squared_error(y_true, y_pred, sample_weight=None,
multioutput='uniform_average')
```

返回值：预测误差的平方的平均值。

参数：参考 `mean_absolute_error` 。

三、验证曲线 && 学习曲线

3.1 验证曲线

1. 验证曲线给出了 `estimator` 因为某个超参数的不同取值在同一个测试集上预测性能曲线。

   它的作用是执行超参数调优。

2. `validation_curve` 用于生成验证曲线，其原型为：

```
sklearn.model_selection.validation_curve(estimator, X, y, param_name, param_range,
cv=None, scoring=None, n_jobs=1, pre_dispatch='all', verbose=0)
```

返回值：返回一个元组，其元素依次为：

- `train_scores`：学习器在训练集上的预测得分的序列（针对不同的参数值），是个二维数组。
- `test_scores`：学习器在测试集上的预测得分的序列（针对不同的参数值），是个二维数组。

> 因为对于每个固定的参数值，k 折交叉会产生多个测试集，得到多个测试得分。

参数：

- `estimator`：一个学习器对象。它必须有 `.fit` 方法用于学习，`.predict` 方法用于预测。
- `param_name`：一个字符串，指定了学习器需要变化的参数。
- `param_range`：一个序列，指定了 `param_name` 指定的参数的取值范围。

- 其它参数参考 `cross_val_score` 。

3.2 学习曲线

1. 学习曲线给出了 `estimator` 因为数据集大小的不同而导致的学习器在训练集和测试集上预测性能曲线。

   其作用是评估样本集大小的变化对学习器的性能的影响。

2. `learning_curve` 函数用于生成学习曲线，其原型为：

   ```
   sklearn.model_selection.learning_curve(estimator, X, y,
   train_sizes=array([ 0.1, 0.33, 0.55, 0.78, 1. ]), cv=None,scoring=None,
   exploit_incremental_learning=False,n_jobs=1, pre_dispatch='all', verbose=0)
   ```

   返回值：返回一个元组，其元素依次为：

   - `train_sizes_abs` ：考察数据集大小组成的序列。
   - `train_scores` ：学习器在训练集上的预测得分的序列（针对不同的考察数据集），是个二维数组。
   - `test_scores` ：学习器在测试集上的预测得分的序列（针对不同的考察数据集），是个二维数组。

   参数：

   - `train_sizes` ：一个数组，给出了训练集的大小。
     - 如果元素为整数，则表示每个训练集的绝对大小。
     - 如果元素为浮点数，则表示每个训练集的相对大小。
   - `exploit_incremental_learning` ：一个布尔值。如果 `estimator` 支持增量学习，那么设置它为 `True` 。

     此时该函数会使用增量学习来加速学习曲线的生成过程。
   - 其它参数参考 `validation_curve` 。

四、超参数优化

4.1 GridSearchCV

1. `GridSearchCV` 用于实现超参数优化，其原型为：

   ```
   class sklearn.model_selection.GridSearchCV(estimator, param_grid, scoring=None,
   fit_params=None, n_jobs=1, iid=True, refit=True, cv=None,verbose=0,
   pre_dispatch='2*n_jobs', error_score='raise',return_train_score='warn')
   ```

   - `estimator` ：一个学习器对象。它必须有 `.fit` 方法用于学习， `.predict` 方法用于预测，有 `.score` 方法用于性能评分。
   - `param_grid` ：字典或者字典的列表。每个字典都给出了学习器的一个超参数，其中：
     - 字典的键就是超参数名。
     - 字典的值是一个列表，指定了超参数对应的候选值序列。
   - `fit_params` ：一个字典，用来给学习器的 `.fit` 方法传递参数。
   - `iid` ：如果为 `True` ，则表示数据是独立同分布的。

- `refit`：一个布尔值。如果为`True`，则在参数优化之后使用整个数据集来重新训练该最优的`estimator`。
- `error_score`：一个数值或者字符串`'raise'`，指定当`estimator`训练发生异常时，如何处理：
  - 如果为`'raise'`，则抛出异常。
  - 如果为数值，则将该数值作为本轮`estimator`的预测得分。
- `return_train_score`： 一个布尔值，指示是否返回训练集的预测得分。

  如果为`'warn'`，则等价于`True`并抛出一个警告。
- 其它参数参考`cross_val_score`。

2. 属性：
   - `cv_results_`：一个数组的字典。可以直接用于生成`pandas DataFrame`。其中键为超参数名，值为超参数的数组。

     另外额外多了一些键：
     - `mean_fit_time`、`mean_score_time`、`std_fit_time`、`std_score_time`：给出了训练时间、评估时间的均值和方差，单位为秒。
     - `xx_score`：给出了各种评估得分。
   - `best_estimator_`：一个学习器对象，代表了根据候选参数组合筛选出来的最佳的学习器。

     如果`refit=False`，则该属性不可用。
   - `best_score_`：最佳学习器的性能评分。
   - `best_params_`：最佳参数组合。
   - `best_index_`：`cv_results_`中，第几组参数对应着最佳参数组合。
   - `scorer_`：评分函数。
   - `n_splits_`：交叉验证的`k`值。
3. 方法：
   - `fit(X[, y,groups])`：执行参数优化。
   - `predict(X)`：使用学到的最佳学习器来预测数据。
   - `predict_log_proba(X)`：使用学到的最佳学习器来预测数据为各类别的概率的对数值。
   - `predict_proba(X)`：使用学到的最佳学习器来预测数据为各类别的概率。
   - `score(X[, y])`：通过给定的数据集来判断学到的最佳学习器的预测性能。
   - `transform(X)`：对最佳学习器执行`transform`。
   - `inverse_transform(X)`：对最佳学习器执行逆`transform`。
   - `decision_function(X)`：对最佳学习器调用决策函数。
4. `GridSearchCV`实现了`estimator`的`.fit`、`.score`方法。这些方法内部会调用`estimator`的对应的方法。

   在调用`GridSearchCV.fit`方法时，首先会将训练集进行 k 折交叉，然后在每次划分的集合上进行多轮的训练和验证（每一轮都采用一种参数组合），然后调用最佳学习器的`.fit`方法。

4.2 RandomizedSearchCV

1. `GridSearchCV`采用的是暴力寻找的方法来寻找最优参数。当待优化的参数是离散的取值的时候，`GridSearchCV`能够顺利找出最优的参数。但是当待优化的参数是连续取值的时候，暴力寻找就有心无力。

`GridSearchCV`的做法是从这些连续值中挑选几个值作为代表，从而在这些代表中挑选出最佳的参数。

2. `RandomizedSearchCV`采用随机搜索所有的候选参数对的方法来寻找最优的参数组合。其原型为：

```
class sklearn.model_selection.RandomizedSearchCV(estimator, param_distributions,
n_iter=10, scoring=None, fit_params=None, n_jobs=1, iid=True, refit=True,
cv=None, verbose=0, pre_dispatch='2*n_jobs', random_state=None,
error_score='raise',return_train_score='warn')
```

   - `param_distributions`：字典或者字典的列表。每个字典都给出了学习器的一个参数，其中：
     - 字典的键就是参数名。
     - 字典的值是一个分布类，分布类必须提供`.rvs`方法。

       通常你可以使用`scipy.stats`模块中提供的分布类，比如`scipy.expon`(指数分布)、`scipy.gamma`(gamma分布)、`scipy.uniform`(均匀分布)、`randint`等等。
     - 字典的值也可以是一个数值序列，此时就在该序列中均匀采样。
   - `n_iter`：一个整数，指定每个参数采样的数量。通常该值越大，参数优化的效果越好。但是参数越大，运行时间也更长。
   - 其它参数参考`GridSearchCV`。

3. 属性：参考`GridSearchCV`。

4. 方法：参考`GridSearchCV`。