预处理

1. 预处理的一些通用方法：
   - `get_params([deep])`：返回模型的参数。
     - `deep`： 如果为`True`，则可以返回模型参数的子对象。
   - `set_params(**params)`：设置模型的参数。
     - `params`：待设置的关键字参数。
   - `fit(X[, y])`：获取预处理需要的参数（如：特征的最大值、最小值等），不同的预处理方法需要的参数不同。
     - `X`：训练集样本集合。通常是一个`numpy array`，每行代表一个样本，每列代表一个特征。
     - `y`：训练样本的标签集合。它与`X`的每一行相对应。
   - `transform(X[, copy])`：执行预处理，返回处理后的样本集。
     - `X`：训练集样本集合。通常是一个`numpy array`，每行代表一个样本，每列代表一个特征。
     - `copy`：一个布尔值，指定是否拷贝数据。
   - `fit_transform(X[, y])`：获取预处理需要的参数并执行预处理，返回处理后的样本集。
     - `X`：训练集样本集合。通常是一个`numpy array`，每行代表一个样本，每列代表一个特征。
     - `y`：训练样本的标签集合。它与`X`的每一行相对应。
2. 预处理的一些通用参数：
   - `copy`： 一个布尔值，指定是否拷贝数据。

     如果为`False`则执行原地修改。此时节省空间，但修改了原始数据。

一、特征处理

1.1 二元化

1. 二元化`Binarizer`的原型为：

```
class sklearn.preprocessing.Binarizer(threshold=0.0, copy=True)
```

   - `threshold`：一个浮点数，它指定了转换阈值：低于此阈值的值转换为0，高于此阈值的值转换为 1。
   - `copy`：一个布尔值，指定是否拷贝数据。
2. 方法：
   - `fit(X[, y])`：不作任何事情，主要用于为流水线`Pipeline`提供接口。
   - `transform(X[, copy])`：将每个样本的特征二元化。
   - `fit_transform(X[, y])`：将每个样本的特征二元化。

1.2 独热码

1. 独热码`OneHotEncoder`的原型为：

```
class sklearn.preprocessing.OneHotEncoder(n_values='auto', categorical_features='all',
dtype=<class 'float'>, sparse=True, handle_unknown='error')
```

- `n_values`：字符串`'auto'`，或者一个整数，或者一个整数的数组，它指定了样本每个特征取值的上界（特征的取值为从0开始的整数）：
  - `'auto'`：自动从训练数据中推断特征值取值的上界。
  - 一个整数：指定了所有特征取值的上界。
  - 一个整数的数组：每个元素依次指定了每个特征取值的上界。
- `categorical_features`：字符串`'all'`，或者下标的数组，或者是一个`mask`，指定哪些特征需要独热码编码：
  - `'all'`：所有的特征都将独热码编码。
  - 一个下标的数组：指定下标的特征将独热码编码。
  - 一个`mask`：对应为`True`的特征将编码为独热码。

  所有的非`categorical`特征都将被安排在`categorical`特征的右边。
- `dtype`：一个类型，指定了独热码编码的数值类型，默认为`np.float`。
- `sparse`：一个布尔值，指定编码结果是否作为稀疏矩阵。
- `handle_unknown`：一个字符串，指定转换过程中遇到了未知的`categorical`特征时的异常处理策略。可以为：
  - `'error'`：抛出异常。
  - `'ignore'`：忽略。

2. 属性：
   - `active_features_`：一个索引数组，存放转换后的特征中哪些是由独热码编码而来。

     仅当`n_values='auto'`时该属性有效。
   - `feature_indices_`：一个索引数组，存放原始特征和转换后特征位置的映射关系。

     第`i`个原始特征将被映射到转换后的`[feature_indices_[i],feature_indices_[i+1])`之间的特征。
   - `n_values_`：一个计数数组，存放每个原始特征取值的种类。

     一般为训练数据中该特征取值的最大值加1，这是因为默认每个特征取值从零开始。
3. 方法：
   - `fit(X[, y])`：训练编码器。
   - `transform(X)`：执行独热码编码。
   - `fit_transform(X[, y])`：训练编码器，然后执行独热码编码。

1.3 标准化

1.3.1 MinMaxScaler

1. `MinMaxScaler`实现了`min-max`标准化，其原型为：

```
class sklearn.preprocessing.MinMaxScaler(feature_range=(0, 1), copy=True)
```

- `feature_range`：一个元组`(min,max)`，指定了执行变换之后特征的取值范围。
- `copy`：一个布尔值，指定是否拷贝数据。

2. 属性：
   - `min_`：一个数组，给出了每个特征的原始最小值的调整值。

     设特征 j 的原始最小值为 $j_{\min}$ ，原始最大值为 $j_{\max}$ 。则特征 j 的原始最小值的调整值为： $\frac{j_{\min}}{j_{\max}-j_{\min}}$ 。
   - `scale_`：一个数组，给出了每个特征的缩放倍数 。
   - `data_min_`：一个数组，给出了每个特征的原始最小值 。
   - `data_max_`：一个数组，给出了每个特征的原始最大值。
   - `data_range_`：一个数组，给出了每个特征的原始的范围（最大值减最小值）。
3. 方法：
   - `fit(X[, y])` ：计算每个特征的最小值和最大值，从而为后续的转换做准备。
   - `transform(X)` ：执行特征的标准化。
   - `fit_transform(X[, y])` ：计算每个特征的最大小值和最大值，然后执行特征的标准化。
   - `inverse_transform(X)`：逆标准化，还原成原始数据。
   - `partial_fit(X[, y])` ：学习部分数据，计算每个特征的最小值和最大值，从而为后续的转换做准备。

     它支持批量学习，这样对于内存更友好。即训练数据并不是一次性学习，而是分批学习。

1.3.2 MaxAbsScaler

1. `MaxAbsScaler` 实现了`max-abs` 标准化，其原型为：

```
class sklearn.preprocessing.MaxAbsScaler(copy=True)
```

   - `copy`：一个布尔值，指定是否拷贝数据。

2. 属性：
   - `scale_`：一个数组，给出了每个特征的缩放倍数的倒数。
   - `max_abs_`：一个数组，给出了每个特征的绝对值的最大值。
   - `n_samples_seen_`：一个整数，给出了当前已经处理的样本的数量（用于分批训练）。
3. 方法：参考`MinMaxScaler` 。

1.3.3 StandardScaler

1. `StandardScaler`实现了 `z-score`标准化，其原型为：

```
class sklearn.preprocessing.StandardScaler(copy=True, with_mean=True, with_std=True)
```

   - `copy`：一个布尔值，指定是否拷贝数据。
   - `with_mean`：一个布尔值，指定是否中心化。
     - 如果为`True`，则缩放之前先将每个特征中心化（即特征值减去该特征的均值）。
     - 如果元素数据是稀疏矩阵的形式，则不能指定`with_mean=True` 。

- `with_std`：一个布尔值，指定是否方差归一化。

  如果为`True`，则缩放每个特征到单位方差。

2. 属性：
   - `scale_`：一个数组，给出了每个特征的缩放倍数的倒数。
   - `mean_`：一个数组，给出了原始数据每个特征的均值。
   - `var_`：一个数组，给出了原始数据每个特征的方差。
   - `n_samples_seen_`：一个整数，给出了当前已经处理的样本的数量（用于分批训练）。
3. 方法：参考`MinMaxScaler`。

1.4 正则化

1. `Normalizer`实现了数据正则化，其原型为：

```
class sklearn.preprocessing.Normalizer(norm='l2', copy=True)
```

   - `norm`：一个字符串，指定正则化方法。可以为：
     - `'l1'`：采用 L_1 范数正则化。
     - `'l2'`：采用 L_2 范数正则化。
     - `'max'`：采用 L_∞ 范数正则化。
   - `copy`：一个布尔值，指定是否拷贝数据。

2. 方法：
   - `fit(X[, y])`：不作任何事情，主要用于为流水线`Pipeline`提供接口。
   - `transform(X[, y, copy])`：将每一个样本正则化为范数等于单位1。
   - `fit_transform(X[, y])`：将每一个样本正则化为范数等于单位1。

二、特征选择

2.1 过滤式特征选取

2.1.1 VarianceThreshold

1. `VarianceThreshold`用于剔除方差很小的特征，其原型为：

```
class sklearn.feature_selection.VarianceThreshold(threshold=0.0)
```

   - `threshold`：一个浮点数，指定方差的阈值。低于此阈值的特征将被剔除。

2. 属性：
   - `variances_`：一个数组，元素分别是各特征的方差。
3. 方法：
   - `fit(X[, y])`：从样本数据中学习每个特征的方差。
   - `transform(X)`：执行特征选择，即删除低于指定阈值的特征。
   - `fit_transform(X[, y])`：从样本数据中学习每个特征的方差，然后执行特征选择。

- `get_support([indices])`：返回保留的特征。
  - 如果`indices=True`，则返回被选出的特征的索引。
  - 如果`indices=False`，则返回一个布尔值组成的数组，该数组指示哪些特征被选择。
- `inverse_transform(X)`：根据被选出来的特征还原原始数据（特征选取的逆操作），但是对于被删除的特征的值全部用 0 代替。

2.1.2 SelectKBest

1. `SelectKBest` 用于保留统计得分最高的 k 个特征，其原型为：

```
class sklearn.feature_selection.SelectKBest(score_func=<function f_classif>, k=10)
```

   - `score_func`：一个函数，用于给出统计指标。

     该函数的参数为 `(X,y)` ，返回值为`(scores,pvalues)` 。
     - `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。
     - `scores` ：样本的得分集合。它与`X` 的每一行相对应。
     - `pvalues`：样本得分的`p` 值。它与`X` 的每一行相对应。
   - `k`：一个整数或者字符串`'all'`，指定要保留最佳的几个特征。

     如果为`'all'`，则保留所有的特征。
2. `sklearn`提供的常用的统计指标函数为：
   - `sklearn.feature_selection.f_regression`：基于线性回归分析来计算统计指标，适用于回归问题。
   - `sklearn.feature_selection.chi2`：计算卡方统计量，适用于分类问题。
   - `sklearn.feature_selection.f_classif`：根据方差分析`Analysis of variance: ANOVA`的原理，依靠`F-分布`为机率分布的依据，利用平方和与自由度所计算的组间与组内均方估计出`F`值。适用于分类问题 。
3. 属性：
   - `scores_`：一个数组，给出了所有特征的得分。
   - `pvalues_`：一个数组，给出了所有特征得分的`p-values` 。
4. 方法：参考`VarianceThreshold` 。

2.1.3 SelectPercentile

1. `SelectPercentile` 用于保留统计得分最高的 $k\%$ 比例的特征，其原型为：

```
class sklearn.feature_selection.SelectPercentile(score_func=<function f_classif>,
percentile=10)
```

   - `score_func`：一个函数，用于给出统计指标。参考`SelectKBest` 。
   - `percentile`：一个整数，指定要保留最佳的百分之几的特征，如`10`表示保留最佳的百分之十的特征
2. 属性：参考`SelectKBest` 。
3. 方法：参考`VarianceThreshold` 。

2.2 包裹式特征选取

2.2.1 RFE

1. `RFE` 类用于实现包裹式特征选取，其原型为：

```
class sklearn.feature_selection.RFE(estimator,
n_features_to_select=None,step=1,verbose=0)
```

   - `estimator`：一个学习器，它必须提供一个 `.fit` 方法和一个 `.coef_` 特征。其中 `.coef_` 特征中存放的是学习到的各特征的权重系数。

     通常使用 `SVM` 和广义线性模型作为 `estimator` 参数。
   - `n_features_to_select`：一个整数或者 `None`，指定要选出几个特征。

     如果为 `None`，则默认选取一半的特征。
   - `step`：一个整数或者浮点数，指定每次迭代要剔除权重最小的几个特征。
     - 如果大于等于1，则作为整数，指定每次迭代要剔除特征的数量。
     - 如果在 `0.0~1.0` 之间，则指定每次迭代要剔除特征的比例。
   - `verbose`：一个整数，控制输出日志。
2. `RFE` 要求学习器能够学习特征的权重（如线性模型），其原理为：
   - 首先学习器在初始的特征集合上训练。
   - 然后学习器学得每个特征的权重，剔除当前权重一批特征，构成新的训练集。
   - 再将学习器在新的训练集上训练，直到剩下的特征的数量满足条件。
3. 属性：
   - `n_features_`：一个整数，给出了被选出的特征的数量。
   - `support_`：一个数组，给出了特征是否被选择的 `mask` 。
   - `ranking_`：特征权重排名。原始第 `i` 个特征的排名为 `raning_[i]` 。
   - `estimator_`： 外部提供的学习器 。
4. 方法：
   - `fit(X,y)`：训练 `RFE` 模型
   - `transform(X)`：执行特征选择。
   - `fit_transform(X,y)`：从样本数据中学习 `RFE` 模型，然后执行特征选择。
   - `get_support([indices])`：返回保留的特征。
     - 如果 `indices=True`，则返回被选出的特征的索引。
     - 如果 `indices=False`，则返回一个布尔值组成的数组，该数组指示哪些特征被选择。
   - `inverse_transform(X)`：根据被选出来的特征还原原始数据（特征选取的逆操作），但是对于被删除的特征值全部用 0 代替。
   - `predict(X)/predict_log_proba(X) /predict_proba(X)`：将 `X` 进行特征选择之后，在使用内部的 `estimator` 来预测。
   - `score(X, y)` ：将 `X` 进行特征选择之后，训练内部 `estimator` 并对内部的 `estimator` 进行评分。

2.2.2 RFECV

1. `RFECV` 是 `RFE` 的一个变体，它执行一个交叉验证来寻找最优的剩余特征数量，因此不需要指定保留多少个特征。

2. `RFECV` 的原型为：

```
class sklearn.feature_selection.RFECV(estimator, step=1, cv=None, scoring=None,verbose=0)
```

   - `cv`：一个整数，或者交叉验证生成器或者一个可迭代对象，它决定了交叉验证策略。
     - 如果为`None`，则使用默认的`3`折交叉验证。
     - 如果为整数 k，则使用 k 折交叉验证。
     - 如果为交叉验证生成器，则直接使用该对象。
     - 如果为可迭代对象，则使用该可迭代对象迭代生成`训练-测试`集合。
   - 其它参数参考`RFE`。

3. 属性：

   - `grid_scores_`：一个数组，给出了交叉验证的预测性能得分。其元素为每个特征子集上执行交叉验证后的预测得分。
   - 其它属性参考`RFE`。

4. 方法：参考`RFE`。

2.3 嵌入式特征选择

1. `SelectFromModel`用于实现嵌入式特征选取，其原型为：

```
class sklearn.feature_selection.SelectFromModel(estimator, threshold=None,
prefit=False)
```

   - `estimator`：一个学习器，它可以是未训练的(`prefit=False`)，或者是已经训练好的(`prefit=True`)。

     `estimator`必须有`coef_`或者`feature_importances_`属性，给出每个特征的重要性。当某个特征的重要性低于某个阈值时，该特征将被移除。
   - `threshold`：一个字符串或者浮点数或者`None`，指定特征重要性的一个阈值。低于此阈值的特征将被剔除。
     - 如果为浮点数，则指定阈值的绝对大小。
     - 如果为字符串，可以是：
       - `'mean'`：阈值为特征重要性的均值。
       - `'median'`：阈值为特征重要性的中值。
       - 如果是`'1.5*mean'`，则表示阈值为 1.5 倍的特征重要性的均值。
     - 如果为`None`：
       - 如果`estimator`有一个`penalty`参数，且该参数设置为`'l1'`，则阈值默认为`1e-5`。
       - 其他情况下，阈值默认为`'mean'`。
   - `prefit`：一个布尔值，指定`estimator`是否已经训练好了。

     如果`prefit=False`，则`estimator`是未训练的。

2. 属性：

   - `threshold_`：一个浮点数，存储了用于特征选取重要性的阈值。

3. 方法：
    - `fit(X,y)`：训练`SelectFromModel`模型。
    - `transform(X)`：执行特征选择。
    - `fit_transform(X,y)`：从样本数据中学习`SelectFromModel`模型，然后执行特征选择。
    - `get_support([indices])`：返回保留的特征。
        - 如果`indices=True`，则返回被选出的特征的索引。
        - 如果`indices=False`，则返回一个布尔值组成的数组，该数组指示哪些特征被选择。
    - `inverse_transform(X)`：根据被选出来的特征还原原始数据（特征选取的逆操作），但是对于被删除的特征值全部用 0 代替。
    - `partial_fit(X[, y])`：通过部分数据来学习`SelectFromModel`模型。

      它支持批量学习，这样对于内存更友好。即训练数据并不是一次性学习，而是分批学习。

三、字典学习

3.1 DictionaryLearning

1. `DictionaryLearning`用于字典学习，其原型为：

```
class sklearn.decomposition.DictionaryLearning(n_components=None, alpha=1,
max_iter=1000, tol=1e-08, fit_algorithm='lars', transform_algorithm='omp',
transform_n_nonzero_coefs=None, transform_alpha=None, n_jobs=1,
code_init=None, dict_init=None, verbose=False, split_sign=False, random_state=None)
```

    - `n_components`：一个整数，指定了字典大小 k 。
    - `alpha`：一个浮点数，指定了L_1 正则化项的系数 λ ，它控制了稀疏性。
    - `max_iter`：一个整数，指定了最大迭代次数。
    - `tol`：一个浮点数，指定了收敛阈值。
    - `fit_algorithm`：一个字符串，指定了求解算法。可以为：
        - `'lars'`：使用`least angle regression`算法来求解。
        - `'cd'`：使用`coordinate descent`算法来求解。
    - `transform_algorithm`：一个字符串，指定了数据转换的方法。可以为：
        - `'lasso_lars'`：使用`Lars`算法来求解。
        - `'lasso_cd'`：使用`coordinate descent`算法来求解。
        - `'lars'`：使用`least angle regression`算法来求解。
        - `'omp'`：使用正交匹配的方法来求解。
        - `'threshold'`：通过字典转换后的坐标中，小于`transform_alpha`的特征的值都设成零。
    - `transform_n_nonzero_coefs`：一个整数，指定解中每一列中非零元素的个数，默认为`0.1*n_features`。

      只用于`lars`算法和`omp`算法（`omp`算法中，可能被`transform_alpha`参数覆盖）。
    - `transform_alpha`：一个浮点数，默认为 1.0 。

        - 如果算法为`lasso_lars`或者`lasso_cd`，则该参数指定了`L1`正则化项的系数。
        - 如果算法为`threshold`，则该参数指定了特征为零的阈值。
        - 如果算法为`omp`，则该参数指定了重构误差的阈值，此时它覆盖了`transform_n_nonzero_coefs`参数。
    - `split_sign`：一个布尔值，指定是否拆分系数特征向量为其正向值和负向值的拼接。
    - `n_jobs`：一个整数，指定并行性。
    - `code_init`：一个数组，指定了初始编码，它用于字典学习算法的热启动。
    - `dict_init`：一个数组，指定了初始字典，它用于字典学习算法的热启动。
    - `verbose`：一个整数，控制输出日志。
    - `random_state`：一个整数或者一个`RandomState`实例，或者`None`，指定随机数种子。
2. 属性：
    - `components_`：一个数组，存放学到的字典。
    - `error_`：一个数组，存放每一轮迭代的误差。
    - `n_iter_`：一个整数，存放迭代的次数。
3. 方法：
    - `fit(X,y)`：学习字典。
    - `transform(X)`：根据学到的字典进行编码。
    - `fit_transform(X,y)`：学习字典并执行字典编码。

3.2 MiniBatchDictionaryLearning

1. `MiniBatchDictionaryLearning`也是字典学习，它主要用于大规模数据。它每次训练一批样本，然后连续多次训练。
2. `MiniBatchDictionaryLearning`的原型为：

```
class sklearn.decomposition.MiniBatchDictionaryLearning(n_components=None, alpha=1,
n_iter=1000, fit_algorithm='lars', n_jobs=1, batch_size=3, shuffle=True,
dict_init=None, transform_algorithm='omp', transform_n_nonzero_coefs=None,
transform_alpha=None, verbose=False, split_sign=False, random_state=None)
```

    - `n_iter`：一个整数，指定了总的执行迭代的数量。
    - `batch_size`：一个整数，指定了每次训练时的样本数量。
    - `shuffle`：一个布尔值，指定在训练每一批样本之前，是否对该批次样本进行混洗。
    - 其它参数参考`DictionaryLearning`。
3. 属性：
    - `components_`：一个数组，存放学到的字典。
    - `inner_stats_`：数组的元组，存放算法的中间状态。
    - `n_iter_`：一个整数，存放迭代的次数。
4. 方法：
    - `fit(X,y)`：学习字典。
    - `transform(X)`：根据学到的字典进行编码。
    - `fit_transform(X,y)`：学习字典并执行字典编码。

- `partial_fit(X[, y, iter_offset])`：只训练一个批次的样本。

四、PipeLine

1. `scikit-learn` 中的流水线的流程通常为：
   - 通过一组特征处理 `estimator` 来对特征进行处理（如标准化、正则化）。
   - 通过一组特征提取 `estimator` 来提取特征。
   - 通过一个模型预测 `estimator` 来学习模型，并执行预测。

   除了最后一个 `estimator` 之外，前面的所有的 `estimator` 必须提供 `transform` 方法。该方法用于执行数据变换（如归一化、正则化、以及特征提取等）。
2. `Pipeline` 将多个 `estimator` 组成流水线，其原型为：

```
class sklearn.pipeline.Pipeline(steps)
```

   - `steps`：一个列表，列表的元素为 `(name,transform)` 元组。其中：
     - `name` 是 `estimator` 的名字，用于输出和日志
     - `transform` 是 `estimator` 。之所以叫 `transform` 是因为这个 `estimator` （除了最后一个）必须提供 `transform` 方法。
3. 属性：
   - `named_steps`：一个字典。键就是 `steps` 中各元组的 `name` 元素，字典的值就是 `steps` 中各元组的 `transform` 元素。
4. 方法：
   - `fit(X[, y])`：启动流水线，依次对各个 `estimator` （除了最后一个）执行 `.fit` 方法和 `.transform` 方法转换数据；对最后一个 `estimator` 执行 `.fit` 方法训练学习器。
   - `transform(X)`：启动流水线，依次对各个 `estimator` （包括最后一个）执行 `.fit` 方法和 `.transform` 方法转换数据。
   - `fit_transform(X[, y])`：启动流水线，依次对各个 `estimator` （除了最后一个）执行 `.fit` 方法和 `.transform` 方法转换数据；对最后一个 `estimator` 执行 `.fit_transform` 方法转换数据。
   - `inverse_transform(X)`：将转换后的数据逆转换成原始数据。

     要求每个 `estimator` 都实现了 `.inverse_transform` 方法。
   - `predict(X)/predict_log_proba(X) /predict_proba(X)`：将 `X` 进行数据转换后，用最后一个学习器来预测。
   - `score(X, y)` ：将 `X` 进行数据转换后，训练最后一个 `estimator` ，并对最后一个 `estimator` 评分。