Sklearn

- **获取数据**

```
sklearn.datasets:
    sklearn.datasets.load_*()
    sklearn.datasets.fetch_*(data_home=)
```

数据集返回值介绍【知道】

- 返回值类型是bunch--是一个字典类型
- 返回值的属性:
  - data：特征数据数组
  - target：标签(目标)数组
  - DESCR：数据描述
  - feature_names：特征名,
  - target_names：标签(目标值)名

数据集划分和预处理

1数据集预处理

1.1最大最小缩放MinMaxScaler

```
sklearn.preprocessing.MinMaxScaler
(feature_range=0, 1, *, copy=True, clip=False
```

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

- **参数**
  - **feature_range:tuple (min, max), default=(0, 1)**
    - 所需的转换数据范围,默认0到1之间
  - **copy :bool, default=True**
    - 默认为True，复制，，False时是原地修改
  - **clip: bool, default=False**

        - 剪切，默认为false
- **属性**
    - **min_ ： ndarray of shape (n_features,)**
        - 调整的最小值
    - **scale_ ： ndarray of shape (n_features,)**
        - 相对缩放范围
    - **data_min_ ： ndarray of shape (n_features,)**
        - 数据的最小值
    - **data_max_ ： ndarray of shape (n_features,)**
        - 数据的最大值
    - **data_range_ : ndarray of shape (n_features,)**
        - 数据的范围
    - **n_samples_seen_ : int**
        - 处理的样本数
- **方法**
    - **fit（X，y= None)**
        - 拟合数据
    - **transform（X)**
        - 根据fit的拟合器转换X数据
    - **fit_transform(X,y=None, *fit_params)**
        - 拟合数据并且转换
    - **get_params`(*deep=True*)** deep ： bool, default=True
        - 获取此估计器的参数。
        - **params*dict***返回一个字典
    - **inverse_transform(*X*)**
        - 反转换
    - **partial_fit(*X*, *y=None*)**
        - 执行一次X的拟合
    - **set_params（参数)**
        - 设置此估计其的参数

1.2 标准化StandardScaler

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

z = (x - u) / s   u是均值，  s是标准差
```

- **参数**
    - **copy :bool, default=True**

        - 默认为True，复制，False时是原地修改
    - **with_mean： bool, default=True**
        - 默认使用均值
        - 对稀疏矩阵不起作用，因为会将稀疏矩阵变为密集矩阵
    - **with_std 'bool, default=True**
        - 默认使用标准差
- **属性**
    - **scale_： ndarray of shape (n_features,)**
        - 相对缩放范围
    - **mean_ : ndarray of shape (n_features,) or None**
        - 均值
    - **var_： ndarray of shape (n_features,) or None**
        - 方差
    - **n_samples_seen_ : int**
        - 处理的样本数
- **方法**
    - **fit（X，y= None)**
        - 拟合数据
    - **transform（X)**
        - 根据fit的拟合器转换X数据
    - **fit_transform(X,y=None, *fit_params)**
        - 拟合数据并且转换
    - **get_params`(*deep=True*)** deep： bool, default=True
        - 获取此估计器的参数。
        - **params*dict*返回一个字典**
    - **inverse_transform(*X*)**
        - 反转换
    - **partial_fit(*X*, *y=None*)**
        - 执行一次X的拟合
    - **set_params（参数)**
        - 设置此估计其的参数

1.3 OneHotEncoder one-hot编码

```
sklearn.preprocessing.OneHotEncoder
(*, categories='auto', drop=None, sparse=True, dtype=<class 'numpy.float64'>,
handle_unknown='error')
```

- **参数**
    - **categories： ‘auto’ or a list of array-like, default=’auto’**
        - ‘auto’ 自动确定类别
        - list 用列表自定义类别
    - **drop : {‘first’, ‘if_binary’} or a array-like of shape (n_features,), default=None**

        - 指定一种删除每个特征中特定分类的功能
        - **None** 保留所有
        - **'first'** 删除每个特征当中的第一个类，如果只有一个类别，将完全删除
        - **'if_binary'** 每个特征中如果有两个类别，删除第一个
        - array： 删除`drop[i]`是功能中应该删除的类别。`X[:, i]`
    - **sparse： bool, default=True**
        - 默认为True返回稀疏矩阵
        - Fasle时返回数组
    - **dtype： number type, default=float**
        - 输出的数字类型
    - **handle_unknown： {'error', 'ignore'}, default='error'**
        - 在转换过程中是否引发错误 或者 忽略那些异常项
        - 默认是error引发异常
        - 如果忽略则异常项的编码全为0并且反转换时候为Nan
- **属性**
    - **categories_： list of arrays**
        - 每个特征的类别
    - **drop_idx_ : array of shape (n_features,)**
        - `drop_idx_[i]`是`categories_[i]`每个功能要删除的类别的索引。
        - `drop_idx_[i] = None`如果没有要从带有index的特征中删除任何类别`i`，例如when`drop='if_binary'`并且该特征不是二进制的。
        - `drop_idx_ = None` 如果所有已转换的要素都将保留。
- **方法**
    - **fit（X，y= None）**
        - 拟合数据
    - **transform（X）**
        - 根据fit的拟合器转换X数据
    - **fit_transform(X,y=None, *fit_params)**
        - 拟合数据并且转换
    - **get_params`(*deep=True*)** deep： bool, default=True
        - 获取此估计器的参数。
        - **params*dict***返回一个字典
    - **inverse_transform(*X*)**
        - 反转换
    - **partial_fit(*X*, *y=None*)**
        - 执行一次X的拟合
    - **set_params（参数）**
        - 设置此估计其的参数

1.4多项式特征结合PolynomialFeature

```
sklearn.preprocessing.PolynomialFeatures
(degree=2, *, interaction_only=False, include_bias=True, order='C')
```

- **参数**
  - **degree ; int, default=2**
    - 多项式的次数
  - **interaction_only ； bool, default=False**
    - 如果为True，只有组合的特征多项式
  - **include_bias ： bool, default=True**
    - 是否包含偏差列，多项式的幂为0
  - **order ： {'C', 'F'}, default='C'**
    - 在密集情况下输出数组的顺序。“ F”阶的计算速度更快，但可能会减慢后续的估计量。
- **属性**
  - **powers_ ： ndarray of shape (n_output_features, n_input_features)**
    - powers_ [i，j]是第i个输出中第j个输入的指数
  - **n_input_features_ ： int**
    - 输入特征的总数
  - **n_output_features_ ： int**
    - 多项式输出特征的总数
- **方法**
  - **fit（X，y= None）**
    - 拟合数据
  - **transform（X）**
    - 根据fit的拟合器转换X数据
  - **fit_transform(X,y=None, *fit_params)**
    - 拟合数据并且转换
  - **get_params`(*deep=True*)** deep ： bool, default=True
    - 获取此估计器的参数。
    - **params*dict***返回一个字典
  - **inverse_transform(*X*)**
    - 反转换
  - **set_params（参数）**
    - 设置此估计其的参数

2. 特征提取feature_extraction

2.1 字典提取DictVectorizer

```
sklearn.feature_extraction.DictVectorizer
(*, dtype=<class 'numpy.float64'>, separator='=', sparse=True, sort=True)
```

- **参数**
  - **dtype : dtype, default=np.float64**
    - 特征值的类型，默认是浮点
  - **separator： str, default="="**
    - 分隔符，当构造One-hot编码的特征值时要使用的分割字符串。分割传入字典数据的键与值的字符串，生成的字符串会作为特征矩阵的列名。
  - **sparse： bool, default=True**
    - 可选参数,默认为True。transform是否要使用scipy产生一个sparse矩阵。DictVectorizer的内部实现是将数据直接转换成sparse矩阵，如果sparse为False， 再把sparse矩阵转换成numpy.ndarray型数组
  - **sort： bool, default=True**
    - ,可选参数,默认为True。在拟合时是否要多feature_names和vocabulary_进行排序。
- **属性**
  - vocabulary_:
    - 特征名称和特征列索引的映射字典。
  - feature_names_:
    - 一个包含所有特征名称的，长度为特征名称个数的列表。
- **方法**
  - **fit(X,y=None):**
    - 计算出转换结果中feature name与 列索引之间的对照字典vocabulary， *同时会计算出特征名称列表feature_names*。这里的参数y没有任何作用。
  - **fit_transform(X,y=None):**
    - 包含fit函数的功能，并且会将X转换成矩阵。
  - **get_feature_names():**
    - 返回feature_names_
  - **get_params(deep=True):**
    - 返回当前DictVectorizer对象的构造参数。
  - **inverse_transform(X[,dict_type]):**
    - 将矩阵还原成特征字典列表。还原出来的字典跟原数据并不是完全一样。传入的X必须是这个DictVectorizer经过transform或者fit_transform产生的X。
  - **restrict(support, indicies=False):**
    - 根据传入的support参数，对特征矩阵进行筛选。
  - **set_params(*params):**
    - 设置DictVectorizer的参数
  - **transform(X):**
    - 将X转换为numpy.ndarray或者Scipy.sparse

2.2 文本提取CountVectorizer

```
sklearn.feature_extraction.text.CountVectorizer

(*, input='content', encoding='utf-8', decode_error='strict', strip_accents=None,
lowercase=True, preprocessor=None, tokenizer=None, stop_words=None, token_pattern='(?
u)\b\w\w+\b', ngram_range=(1, 1), analyzer='word', max_df=1.0, min_df=1, max_features=None,
vocabulary=None, binary=False, dtype=<class 'numpy.int64'>)
```

- **参数**
  - **input： string {‘filename’, ‘file’, ‘content’}, default=‘content’**
    - 如果为'filename'，则作为参数传递给fit的序列应该是需要读取以获取原始内容进行分析的文件名列表
    - 如果为'file'，则序列项必须具有一个'read'方法（类似于文件的对象），该方法被调用以获取内存中的字节
    - 输入应该是可以是字符串或字节类型的项目序列
  - **encoding :string, default=‘utf-8’**
    - 解编码方法
  - **decode_error： {‘strict’, ‘ignore’, ‘replace’}, default=‘strict’**
    - 如果给出了包含不属于给定字符的字节序列进行分析的指示`encoding`。默认情况下，它是“严格”的，这意味着将引发UnicodeDecodeError。其他值是“忽略”和“替换”
  - **strip_accents**=None*：{‘ascii’, ‘unicode’, None}
    - 在预处理步骤中删除重音。
    - 'ascii'是一种快速方法，仅适用于具有直接ASCII映射的字符。
    - 'unicode'是一种稍微慢一点的方法，适用于任何字符。
    - 无（默认）不执行任何操作。
  - **Lowercase***：*boolean, True by default
    - 在标记化之前将所有字符转换为小写
  - **Preprocessor***：*callable or None (default)
    - 覆盖预处理（字符串转换）阶段，同时保留标记化和n-gram生成步骤。
  - **Tokenizer***：*callable or None (default)
    - 覆盖字符串标记化步骤，同时保留预处理和n-gram生成步骤。 仅适用于analyzer =='word'。
  - **stop_words***：*string {‘english’}, list, or None (default)
    - 如果是“英语”，则使用英语的内置停用词列表。
    - 如果列表，该列表被假定包含停用词，则所有这些将从生成的结果中删除。 仅适用于analyzer =='word'。
    - 如果为None，则不使用停用词。 max_df可以设置为[0.7,1.0]范围内的值，以根据术语的语料库文档频率自动检测和过滤停用词。
  - **token_pattern***：*string
    - 正则表达式表示什么构成“标记”，仅在analyzer =='word'时使用。 默认正则表达式选择2个或更多字母数字字符的标记（标点符号完全被忽略，并始终被视为标记分隔符）。
  - **ngram_range***：tuple (min_n, max_n)*
    - 要提取的不同n-gram的n值范围的下边界和上边界。 将使用n的所有值，使得min_n <= n <= max_n。
  - **Analyzer***：*string, {‘word’, ‘char’, ‘char_wb’} or callable
    - 该功能是否应由单词或字符n-gram组成。

        - 选项'char_wb'仅从字边界内的文本创建字符n-gram; 单词边缘的n-gram用空格填充。
        - 如果传递了一个callable，它将用于从原始未处理的输入中提取特征序列。
    - **max_df***：* float in range [0.0, 1.0] or int, default=1.0
        - 在构建词汇表时，忽略文档频率严格高于给定阈值的术语（语料库特定的停用词）。 如果是float，则参数表示文档的比例，整数绝对计数。 如果词汇表不是None，则忽略此参数。
    - **min_df***：*float in range [0.0, 1.0] or int, default=1
        - 构建词汇表时，请忽略文档频率严格低于给定阈值的术语。 该值在文献中也称为截止值。 如果是float，则参数表示文档的比例，整数绝对计数。 如果词汇表不是None，则忽略此参数。
    - **max_features***：*int or None, default=None
        - 如果不是None，则构建一个词汇表，该词汇表仅考虑语料库中按术语频率排序的最高max_features。
        - 如果词汇表不是None，则忽略此参数。
    - **Vocabulary***：*Mapping or iterable, optional
        - 其中键是术语和值的映射（例如，字典）是特征矩阵中的索引，或者是可迭代的术语。 如果没有给出，则从输入文档确定词汇表。 映射中的索引不应重复，并且不应该在0和最大索引之间存在任何差距。
    - **Binary***：*boolean, default=False
        - 如果为True，则所有非零计数都设置为1.这对于模拟二进制事件而非整数计数的离散概率模型非常有用。
    - **dtype***：*type, optional
        - fit_transform() or transform().返回的矩阵类型
- **属性**
    - **vocabulary_： dict**
        - 到特征索引的映射
    - **fixed_vocabulary_: boolean**
        - 如果用户提供了固定的映射表，则为True
    - **stop_words_： set**
        - 被忽略的词语
        - 出现在太多（`max_df`）
        - 出现太少（`min_df`）中
        - 被最大特征抛弃（`max_features`）。
- **方法**
    - **build_analyzer()**
        - 返回一个进行预处理和分词的分析器函数对象。
    - **build_preprocessor（）：**
        - 返回一个在分词之前进行预处理的可调用方法
    - **build_tokenizer（）：**
        - 返回一个将字符串分为词语序列的方法
    - **decode（）：**
        - 将输入转换为unicode字符表
        - 解码的策略参照vectorizer的参数。
    - **fit(X,y=None):**

        - 计算出转换结果中feature name与 列索引之间的对照字典vocabulary， *同时会计算出特征名称列表feature_names*。这里的参数y没有任何作用。
    - **fit_transform(X,y=None):**
        - 包含fit函数的功能，并且会将X转换成矩阵。
    - **get_feature_names():**
        - 返回feature_names_
    - **get_params(deep=True):**
        - 返回当前DictVectorizer对象的构造参数。
    - **get_stop_words()**
        - 获取被忽略的列表
    - **inverse_transform(X[,dict_type]):**
        - 将矩阵还原成特征字典列表。还原出来的字典跟原数据并不是完全一样。传入的X必须是这个DictVectorizer经过transform或者fit_transform产生的X。
    - **set_params(*params):**
        - 设置DictVectorizer的参数
    - **transform(X):**
        - 将X转换为numpy.ndarray或者Scipy.sparse

2.3 Tf-idf文本特征提取

```
sklearn.feature_extraction.text.TfidfVectorizer
(*, input='content', encoding='utf-8', decode_error='strict', strip_accents=None,
lowercase=True, preprocessor=None, tokenizer=None, analyzer='word', stop_words=None,
token_pattern='(?u)\b\w\w+\b', ngram_range=(1, 1), max_df=1.0, min_df=1, max_features=None,
vocabulary=None, binary=False, dtype=<class 'numpy.float64'>, norm='l2', use_idf=True,
smooth_idf=True, sublinear_tf=False)
```

- **参数**
    - **input： string {‘filename’, ‘file’, ‘content’}, default=’content’**
        - 如果为'filename'，则作为参数传递给fit的序列应该是需要读取以获取原始内容进行分析的文件名列表
        - 如果为'file'，则序列项必须具有一个'read'方法（类似于文件的对象），该方法被调用以获取内存中的字节
        - 输入应该是可以是字符串或字节类型的项目序列
    - **encoding :string, default=’utf-8’**
        - 解编码方法
    - **decode_error： {‘strict’, ‘ignore’, ‘replace’}, default=’strict’**
        - 如果给出了包含不属于给定字符的字节序列进行分析的指示`encoding`。默认情况下，它是“严格”的，这意味着将引发UnicodeDecodeError。其他值是“忽略”和“替换”
    - **strip_accents**=None*：{‘ascii’, ‘unicode’, None}
        - 在预处理步骤中删除重音。

    - 'ascii'是一种快速方法，仅适用于具有直接ASCII映射的字符。
    - 'unicode'是一种稍微慢一点的方法，适用于任何字符。
    - 无（默认）不执行任何操作。
- **lowercase: bool, default=True**
    - 在标记化之前，将所有字符转换为小写。
- **preprocessor； callable, default=None**
    - 覆盖预处理（字符串转换）阶段，同时保留标记化
        - 仅适用于。`analyzer is not callable`
- **tokenizer： callable, default=None**
    - 在保留预处理和n-gram生成步骤的同时，覆盖字符串标记化步骤，仅适用于。`analyzer == 'word'`
- **analyzer： {‘word’, ‘char’, ‘char_wb’} or callable, default=‘word’**
    - 用单词还是字符来分析
- **stop_words： {‘english’}, list, default=None**
    - 如果是“英语”，则使用英语的内置停用词列表。
    - 如果列表，该列表被假定包含停用词，则所有这些将从生成的结果中删除。 仅适用于analyzer =='word'。
    - 如果为None，则不使用停用词。 max_df可以设置为[0.7,1.0]范围内的值，以根据术语的语料库文档频率自动检测和过滤停用词。
- **token_pattern： str, default=r”(?u)\b\w\w+\b”**
    - 正则表达式，
- **ngram_range： tuple (min_n, max_n), default=(1, 1)**
    - 要提取的不同n-gram的n值范围的上下边界。将使用所有min_n <= n <= max_n的n值。
- **max_df**float or int, default=1.0**
    - 构建词汇表时，请忽略文档频率严格高于给定阈值（特定于语料库的停用词）的术语。如果float在范围[0.0，1.0]中，则该参数表示文档的比例，整数绝对计数
- **min_df： float or int, default=1**
    - 构建词汇表时，请忽略文档频率严格低于给定阈值的术语。该值在文献中也称为临界值。如果float在[0.0，1.0]范围内，则该参数表示文档的比例，整数绝对计数。
- **max_features： int, default=None**
    - 如果不是None，则建立一个仅考虑整个语料库中按词频排列的最大max_features的词汇表。
- **Vocabulary***：*Mapping or iterable, optional
    - 其中键是术语和值的映射（例如，字典）是特征矩阵中的索引，或者是可迭代的术语。 如果没有给出，则从输入文档确定词汇表。 映射中的索引不应重复，并且不应该在0和最大索引之间存在任何差距。
- **Binary***：*boolean, default=False
    - 如果为True，则所有非零计数都设置为1.这对于模拟二进制事件而非整数计数的离散概率模型非常有用。
- **dtype***：*type, optional
    - fit_transform() or transform().返回的矩阵类型
- **norm: {‘l1’, ‘l2’}, default=‘l2’**
    - 每个输出行将具有单位范数，
    - ：*'l2'：矢量元素的平方和为1。当应用l2范数时，两个矢量之间的余弦相似度是它们的点积。*

        - 'l1'：矢量元素的绝对值之和为1
    - **use_idf： bool, default=True**
        - 启用反向文档频率重新加权
    - **smooth_idf: bool, default=True**
        - 通过在文档频率上增加一个来平滑idf权重，就好像看到一个额外的文档包含集合中的每个术语恰好一次。防止零除。
    - **sublinear_tf: bool, default=False**
        - 用1 + log（tf）替换tf。
- **属性**
    - **vocabulary_： dict**
        - 到特征索引的映射
    - **fixed_vocabulary_: boolean**
        - 如果用户提供了固定的映射表，则为True
    - **stop_words_ ： set**
        - 被忽略的词语
        - 出现在太多（`max_df`）
        - 出现太少（`min_df`）中
        - 被最大特征抛弃（`max_features`）。
    - **idf_**:
        - 逆文档频率（IDF）向量；仅在`use_idf`为True时定义。
- **方法**
    - **build_analyzer()**
        - 返回一个进行预处理和分词的分析器函数对象。
    - **build_preprocessor（）：**
        - 返回一个在分词之前进行预处理的可调用方法
    - **build_tokenizer（）：**
        - 返回一个将字符串分为词语序列的方法
    - **decode（）：**
        - 将输入转换为unicode字符表
        - 解码的策略参照vectorizer的参数。
    - **fit(X,y=None):**
        - 计算出转换结果中feature name与 列索引之间的对照字典vocabulary， *同时会计算出特征名称列表 feature_names*。这里的参数y没有任何作用。
    - **fit_transform(X,y=None):**
        - 包含fit函数的功能，并且会将X转换成矩阵。
    - **get_feature_names():**
        - 返回feature_names_
    - **get_params(deep=True):**
        - 返回当前DictVectorizer对象的构造参数。
    - **get_stop_words()**
        - 获取被忽略的列表
    - **inverse_transform(X[,dict_type]):**

        - 将矩阵还原成特征字典列表。还原出来的字典跟原数据并不是完全一样。传入的X必须是这个DictVectorizer经过transform或者fit_transform产生的X。
    - **set_params(*params):**
        - 设置DictVectorizer的参数
    - **transform(X):**
        - 将X转换为numpy.ndarray或者Scipy.sparse

3.特征降维

3.1单边量统计降维

计算每个特征单独于目标值的关系，将超过阈值的删去

根据百分数来确定阈值

```
sklearn.feature_selection.SelectPercentile
(score_func=<function f_classif>, *, percentile=10)[source]
```

- **参数**
    - **score_func： callable, default=f_classif**
        - 函数接受两个数组X和y，并返回一对数组（分数，pvalue）或带分数的单个数组。
        - 默认值为f_classif 仅接受分类任务
        - 回归任务设置为f_regression
    - **percentile： int, default=10**
        - 阈值，要保留特征的百分比
- **属性**
    - **scores_： array-like of shape (n_features,)**
        - 分数
    - **pvalues_： array-like of shape (n_features,)**
        - 和score_func 有关
- **方法**
    - **fit（X，y= None)**

        - 拟合数据
    - **transform（X)**
        - 根据fit的拟合器转换X数据
    - **fit_transform(X,y=None, *fit_params)**
        - 拟合数据并且转换
    - **get_params`(*deep=True*)** deep： bool, default=True
        - 获取此估计器的参数。
        - **params*dict***返回一个字典
    - **get_support()**
        - 获得选取特征的索引
    - **inverse_transform(*X*)**
        - 反转换
    - **set_params（参数)**
        - 设置此估计其的参数

按照个数选取特征

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

- 和按照百分数选取相同，
- 唯一不同是参数变为
    - **k： int or “all”, default=10**
        - 选取特征的K的数量，默认是10个

3.2 按照模型降维

模型降维即用一种模型机器学习来计算哪些特征应该被抛弃

```
sklearn.feature_selection.SelectFromModel
(estimator, *, threshold=None, prefit=False, norm_order=1, max_features=None,
importance_getter='auto')
```

- **参数**
    - **estimator： object**
        - 用来计算特征的估计器，拟合后必须有`feature_importances_`或`coef_`两个方法，一般用随机森林
    - **threshold： string or float, default=None**
        - 用于特征选择的阈值
            - 如果选择None且估计器的惩罚参数设置为l1，则阈值为1e-5
    - **prefit： bool, default=False**

        - 预设模型是否应该直接传递给构造函数。
    - **norm_order： non-zero int, inf, -inf, default=1**
        - 在估算器`threshold`的`coef_`属性为维度2的情况下，用于过滤范数
    - **max_features： int, default=None**
        - 要保留的最大特征数
    - **importance_getter : str or callable, default='auto'**
        - 如果为“自动”，则通过`coef_`属性或`feature_importances_`估计器属性使用特征重要性。
        - str和可调用情况不知道什么意思
- **属性**
    - **estimator**
        - 估计器
    - **threshold_**
        - 阈值
- **方法**
    - **fit（X，y= None）**
        - 拟合数据
    - **transform（X）**
        - 根据fit的拟合器转换X数据
    - **fit_transform(X,y=None, *fit_params)**
        - 拟合数据并且转换
    - **partial_fit(X,y)**
        - 执行一次拟合过程
    - **get_params`(*deep=True*)** deep： bool, default=True
        - 获取此估计器的参数。
        - **params*dict***返回一个字典
    - **get_support()**
        - 获得选取特征的索引
    - **inverse_transform(*X*)**
        - 反转换
    - **set_params（参数）**
        - 设置此估计其的参数

3.3 按照方差降维

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

- **参数**
    - **threshold： float, default=0**
        - 阈值，方差低于阈值的将被删除
- **属性**
    - **variances_： array, shape (n_features,)**

        - 特征的方差
- **方法**
    - **fit（X，y= None）**
        - 拟合数据
    - **transform（X）**
        - 根据fit的拟合器转换X数据
    - **fit_transform(X,y=None, *fit_params)**
        - 拟合数据并且转换
    - **get_params`(*deep=True*)** deep： bool, default=True
        - 获取此估计器的参数。
        - **params*dict*返回一个字典
    - **get_support()**
        - 获得选取特征的索引
    - **inverse_transform(*X*)**
        - 反转换

3.4 主成分分析PCA

```
 sklearn.decomposition.PCA
    (n_components=None, *, copy=True, whiten=False, svd_solver='auto', tol=0.0,
iterated_power='auto', random_state=None)
```

- **参数**
    - **n_components： int, float or ‘mle’, default=None**
        - 要保留的特征数
        - 如果为设置，则默认保留所有n_components=min(样本数，特征数)
        - 最常用的做法是直接指定降维到的维度数目，此时n_components是一个大于等于1的整数
        - 我们也可以指定主成分的方差和所占的最小比例阈值，让PCA类自己去根据样本特征方差来决定降维到的维度数，此时n_components是一个（0，1]之间的数
        - 我们还可以将参数设置为"mle", 此时PCA类会用MLE算法根据特征的方差分布情况自己去选择一定数量的主成分特征来降维。，
    - **copy： bool, default=True**
        - 表示是否在运行算法时，将原始训练数据复制一份
    - **whiten： bool, default=False**
        - 判断是否进行白化。所谓白化，就是对降维后的数据的每个特征进行归一化，让方差都为1.对于PCA降维本身来说，一般不需要白化。如果你PCA降维后有后续的数据处理动作，可以考虑白化。默认值是False，即不进行白化
    - **svd_solver： {‘auto’, ‘full’, ‘arpack’, ‘randomized’}, default=‘auto’**
        - 指定奇异值分解SVD的方法
        - 默认是auto，即PCA类会自己去在三种算法里面去权衡，选择一个合适的SVD算法来降维。

        - randomized一般适用于数据量大，数据维度多同时主成分数目比例又较低的PCA降维，它使用了一些加快SVD的随机算法
        - full则是传统意义上的SVD，使用了scipy库对应的实现。
        - arpack直接使用了scipy库的sparse SVD实现
    - **tol： float, default=0.0**
        - svd_solver =='arpack'计算的奇异值的公差，float> = 0，可选（默认.0）
    - **iterated_power： int or 'auto', default='auto'**
        - （默认为'auto'）,svd_solver =='randomized'计算出的幂方法的迭代次数
    - **random_state：int, RandomState instance or None, default=None**
        - 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
- **属性**
    - **components_ ：**
        - 特征空间中的主轴，表示数据中最大方差的方向。组件按排序`explained_variance_`
    - **explained_variance_:**
        - 它代表降维后的各主成分的方差值。方差值越大，则说明越是重要的主成分
    - **explained_variance_ratio_:**
        - 它代表降维后的各主成分的方差值占总方差值的比例，这个比例越大，则越是重要的主成分
    - **singular_values_:**
        - 每个特征的奇异值，奇异值等于`n_components`低维空间中变量的2范数
    - **mean_:**
        - 每个特征的均值
    - **n_components_:**
        - 即是上面输入的参数值
    - **n_features_:**
        - 训练数据中的特征数量
    - **n_samples_:**
        - 训练数据中的样本数
    - **noise_variance_:**
        - 等于X协方差矩阵的（min（n_features，n_samples）-n_components）个最小特征值的平均值
- **方法**
    - fit（X，y= None）**
        - 拟合数据
    - **transform（X)**
        - 根据fit的拟合器转换X数据
    - **fit_transform(X,y=None, *fit_params)**
        - 拟合数据并且转换
    - **get_covariance（)**
        - 用生成模型计算数据协方差。
    - **get_params(*deep=True*)** deep： bool, default=True
        - 获取此估计器的参数。

        - **params*dict***返回一个字典
    - **get_precision（）**
        - 用生成模型计算数据精度矩阵。
    - **score（X [，y]）**
        - 返回所有样本的平均对数似然率。
    - **score_samples（X）**
        - 返回每个样本的对数似然。
    - **inverse_transform(*X*)**
        - 反转换

4. 数据集划分

4.1 正常划分

```
sklearn.model_selection.train_test_split
(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None
```

- **参数**
    - **arrays :sequence of indexables with same length / shape[0]**
        - 输入的数据集
    - **test_size： float or int, default=None**
        - 要划分的测试集的大小
        - 浮点数在0到1之间表示百分比
        - 整数表示样本数量
    - **random_state : int, RandomState instance or None, default=None**
        - 划分之前的随机种子，默认是None每次划分不同
        - 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。
    - **shuffle : bool, default=True**
        - 划分之前是否对数据进行洗牌
    - **stratify : array-like, default=None**

        - 划分后的标签占比
        - 默认为None随机划分，每个数据集的分类占比不同
        - True时，训练集和测试集的分类占比相同
- **返回**
    - **splitting : list, length=2 * len(arrays)**
        - x_train, x_test, y_train, y_test

4.2 留一法

```
sklearn.model_selection.LeaveOneOut
```

- **无参数，无属性**
- **方法**
    - **get_n_splits(*X*, *y=None*, *groups=None*)**
        - 返回交叉验证器中的拆分迭代次数
    - **split（X，y，groups=None)**
        - 拆分，返回x_train, x_test
        - 是可迭代的对象

4.3 交叉验证法

```
sklearn.model_selection.StratifiedKFold
   (n_splits=5, *, shuffle=False, random_state=None)
```

- **参数**
    - **n_splits： int, default=5**
        - 划分的份数
    - **shuffle： bool, default=False**
        - 每次划分是否洗牌
        - ①若为Falses时，其效果等同于random_state等于整数，每次划分的结果相同
        - ②若为True时，每次划分的结果都不一样，表示经过洗牌，随机取样的
    - **random_state； int, RandomState instance or None, default=None**
        - 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。
- **方法**
    - **get_n_splits(*X*, *y=None*, *groups=None*)**
        - 返回交叉验证器中的拆分迭代次数
    - **split（X，y，groups=None)**
        - 拆分，返回x_train, x_test
        - 是可迭代的对象

sklearn交叉验证和网格搜索

1.网格搜索和交叉验证

```
sklearn.model_selection.GridSearchCV
(estimator, param_grid, *, scoring=None, n_jobs=None, refit=True, cv=None, verbose=0,
pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False
```

- **参数**
  - **estimator： estimator object.**
    - 估计器对象，算法对象
  - **param_grid： dict or list of dictionaries**
    - 用字典类传入估计器算法的参数
  - **scoring： str, callable, list, tuple or dict, default=None**
    - 评估估计器的策略，参考https://scikit-learn.org/stable/modules/model_evaluation.html#scoring
    - 可以是单一指标，入准确率,F1,查全率等等
    - 如果是多个指标用列表，元组，字典来指定
  - **n_jobs： int, default=None**
    - 并行计算，-1表示使用所有cpu
  - **pre_dispatch： int, or str, default=n_jobs**
    - 和n_jobs配合使用设置并行计算时cpu的调度
    - None，立即开始所有的计算
    - 整数int，给出确切的并行计算数
    - 字符串str，如'2 * n_jobs'
  - **cv： int, cross-validation generator or an iterable, default=None**
    - 交叉验证的策略
    - None：使用默认的5倍交叉
    - 整数int 用于指定的折叠次数
    - 一个可迭代的生成器函数（train，test）拆分
  - **refit： bool, str, or callable, default=True**
    - 默认为True,程序将会以交叉验证训练集得到的最佳参数，重新对所有可用的训练集与开发集进行，作为最终用于性能评估的最佳模型参数。即在搜索参数结束后，用最佳参数结果再次fit一遍全部数据集。
    - 对于多指标，需要用str指定一个计分器
  - **verbose： int**
    - 日志冗长度，int：冗长度，0：不输出训练过程，1：偶尔输出，>1：对每个子模型都输出。
  - **error_score： ‘raise’ or numeric, default=np.nan**

        - 如果设置为“ raise”，则会引发错误。
        - 如果给出数值，则引发FitFailedWarning。此参数不会影响重新安装步骤，这将始终引发错误。
    - **return_train_score： bool, default=False**
        - 如果“False”，cv_results_属性将不包括训练分数
        - 计算培训分数用于了解不同的参数设置如何影响过拟合/欠拟合权衡。但是，在训练集上计算分数可能在计算上非常昂贵，并且并非严格要求选择产生最佳泛化性能的参数。
- **属性**
    - **cv_results_： numpy（掩码）ndarrays字典**
        - 可以导入dataframe
        - 里面包含了所有的估计器计算信息
    - **best_estimator_： estimator**
        - 最好的估计器
    - **best_score_ ： float**
        - 最好的评分
        - 多指标评估只有在`refit`指定时才存在。如果`refit`为函数，则此属性不可用。
    - **best_params_： dict**
        - 最好的估计器的参数
        - 对于多指标评估，只有在`refit`指定时才存在。
    - **best_index_ ： int**
        - `cv_results_`对应于最佳候选参数设置的（数组的）索引
    - **scorer_： 函数或字典**
        - 对保留的数据使用计分器功能
    - **n_splits_ ; int**
        - 交叉验证拆分的数量
    - **refit_time_ ： float**
        - 用最好的参数重新拟合数据集的秒数
        - `refit` is not False.时候才能用
    - **multimetric_ ： bool**
        - 是否用多指标计分
- **方法**
    - **decision_function（X)**
        - 在最佳估计器上调用预测样本的置信度概率
    - **fit(*X*, *y*, *sample_weight=None*)**
        - 训练
    - **get_params(deep=True)**
        - deep： bool 默认为True
        - 返回字典，估计器的各个设置参数
    - **predict（X)**
        - 用最佳的估计其预测X，得到预测值
    - **predict_log_proba（*X*)**
        - 用最佳估计器，返回一个数组，数组的元素一次是 X 预测为各个类别的概率的对数值。
    - **predict_proba（X）：**

        - 用最佳估计器，返回一个数组，数组元素一次是 X 预测为各个类别的概率的概率值。
    - **score(X,y,sample_weight)：**
        - 用最佳估计器，返回（X，y）上的预测准确率（accuracy）。
    - **sparsify()**：
        - 将系数矩阵转换为稀疏格式。
    - **set_params()**
        - 该估计其的设置
    - **score_samples（X)**
        - 使用找到的最佳参数在估算器上调用score_samples。
    - **transform（X)**
        - 使用找到的最佳参数在估算器上调用transform。
        - 仅在基础估算器支持`transform`和`refit=True`.时 可用
    - **inverse_transform（X)**
        - 使用找到的最佳参数在估算器上调用inverse_transform。
        - 仅在基础估算器支持inverse_transform和`refit=True`.时 可用

2.随机搜索

```
sklearn.model_selection.RandomizedSearchCV
(estimator, param_distributions, *, n_iter=10, scoring=None, n_jobs=None, refit=True,
cv=None, verbose=0, pre_dispatch='2*n_jobs', random_state=None, error_score=nan,
return_train_score=False)
```

- **参数**
    - **estimator： estimator object.**
        - 估计器对象，算法对象
    - **param_distributions： dict or list of dicts**
        - 以参数名称（`str`）作为键以及要尝试的分布或参数列表的字典。
        - 如果给出列表，则对其进行统一采样。如果给出了字典列表，则首先对一个字典进行统一采样，然后如上所述使用该字典对参数进行采样
    - **n_iter： int, default=10**
        - 采样的参数设置数。n_iter权衡了运行时间和解决方案质量。
    - **scoring： str, callable, list, tuple or dict, default=None**
        - 评估估计器的策略，参考https://scikit-learn.org/stable/modules/model_evaluation.html#scoring
        - 可以是单一指标，入准确率,F1,查全率等等
        - 如果是多个指标用列表，元组，字典来指定
    - **n_jobs： int, default=None**
        - 并行计算，-1表示使用所有cpu
    - **pre_dispatch： int, or str, default=n_jobs**
        - 和n_jobs配合使用设置并行计算时cpu的调度
        - None，立即开始所有的计算
        - 整数int，给出确切的并行计算数
        - 字符串str，如'2 * n_jobs'

- **cv： int, cross-validation generator or an iterable, default=None**
  - 交叉验证的策略
  - None：使用默认的5倍交叉
  - 整数int 用于指定的折叠次数
  - 一个可迭代的生成器函数（train，test）拆分
- **refit： bool, str, or callable, default=True**
  - 默认为True,程序将会以交叉验证训练集得到的最佳参数，重新对所有可用的训练集与开发集进行，作为最终用于性能评估的最佳模型参数。即在搜索参数结束后，用最佳参数结果再次fit一遍全部数据集。
  - 对于多指标，需要用str指定一个计分器
- **verbose： int**
  - 日志冗长度，int：冗长度，0：不输出训练过程，1：偶尔输出，>1：对每个子模型都输出。
- **error_score： 'raise' or numeric, default=np.nan**
  - 如果设置为" raise"，则会引发错误。
  - 如果给出数值，则引发FitFailedWarning。此参数不会影响重新安装步骤，这将始终引发错误。
- **return_train_score： bool, default=False**
  - 如果"False"，cv_results_属性将不包括训练分数
  - 计算培训分数用于了解不同的参数设置如何影响过拟合/欠拟合权衡。但是，在训练集上计算分数可能在计算上非常昂贵，并且并非严格要求选择产生最佳泛化性能的参数。
- **random_state :int, RandomState instance, default=None**
  - 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。

- **属性**
  - **属性**
    - **cv_results_： numpy（掩码）ndarrays字典**
      - 可以导入dataframe
      - 里面包含了所有的估计器计算信息
    - **best_estimator_： estimator**
      - 最好的估计器
    - **best_score_： float**
      - 最好的评分
      - 多指标评估只有在`refit`指定时才存在。如果`refit`为函数，则此属性不可用。
    - **best_params_： dict**
      - 最好的估计器的参数
      - 对于多指标评估，只有在`refit`指定时才存在。
    - **best_index_： int**
      - `cv_results_`对应于最佳候选参数设置的（数组的）索引
    - **scorer_： 函数或字典**
      - 对保留的数据使用计分器功能
    - **n_splits_ ; int**
      - 交叉验证拆分的数量
    - **refit_time_： float**

        - 用最好的参数重新拟合数据集的秒数
        - `refit` is not False.时候才能用
    - **multimetric_ ： bool**
        - 是否用多指标计分
- **方法**
    - **decision_function（X)**
        - 在最佳估计器上调用预测样本的置信度概率
    - **fit(*X, y, sample_weight=None*)**
        - 训练
    - **get_params(deep=True)**
        - deep ： bool 默认为True
        - 返回字典，估计器的各个设置参数
    - **predict（X)**
        - 用最佳的估计其预测X，得到预测值
    - **predict_log_proba（*X*)**
        - 用最佳估计器，返回一个数组，数组的元素一次是 X 预测为各个类别的概率的对数值。
    - **predict_proba（X）：**
        - 用最佳估计器，返回一个数组，数组元素一次是 X 预测为各个类别的概率的概率值。
    - **score(X,y,sample_weight)：**
        - 用最佳估计器，返回（X，y）上的预测准确率（accuracy）。
    - **set_params()**
        - 该估计其的设置
    - **score_samples（X)**
        - 使用找到的最佳参数在估算器上调用score_samples。
    - **transform（X)**
        - 使用找到的最佳参数在估算器上调用transform。
        - 仅在基础估算器支持`transform`和`refit=True`.时 可用
    - **inverse_transform（X)**
        - 使用找到的最佳参数在估算器上调用inverse_transform。
        - 仅在基础估算器支持inverse_transform和`refit=True`.时 可用

sklearn算法

1.KNN

- ```
  sklearn.neighbors.KNeighborsClassifier
  (n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
  metric='minkowski', metric_params=None, n_jobs=None, kwargs)
  ```

- **参数**
    - **n_neighbors**：int,可选（默认= 5），k_neighbors查询默认使用的邻居数
  ```

- **weights**：预测的权函数，概率值。
  - ‘uniform’：同一的权重，即每个邻域中的所有点都是平均加权的。
  - ‘distance’ ： 这种情况下，距离越近权重越大，反之，距离越远其权重越小。
  - [callable]（可调用）： 用户定义的函数，它接受一个距离数组，并返回一个包含权重的相同形状的数组
- **algorithm**： 用于计算最近邻居的算法,。有{‘auto’, ‘ball_tree’, ‘kd_tree’, ‘brute’}
  - ‘auto’ ： 根据样本数据自动刷选合适的算法。
  - ‘ball_tree’：构建“球树”算法模型。
  - ‘kd_tree’ ： “kd树”算法。
  - ‘brute’ ： 使用蛮力搜索，即或相当于Knn算法，需遍历所有样本数据与目标数据的距离，进而按升序排序从而选取最近的K个值，采用投票得出结果。
- **leaf_size**：默认30
  - 叶的大小，针对算法为球树或KD树而言。这个设置会影响构造和查询的速度，以及存储树所需的内存。最优值取决于问题的性质。
- **metric**：
  - 用于树的距离度量。默认度量是Minkowski，p=2等价于标准的欧几里德度量。有关可用度量的列表，可以查阅距离度量类的文档。如果度量是“预先计算的”，则假定X是距离矩阵，在拟合期间必须是平方。
- **p**：
  - Minkowski度量参数的参数来自sklearn.emeics.pairwise.pairwise距离。当p=1时，这等价于使用曼哈顿距离(L1)，欧几里得距离(L2)等价于p=2时，对于任意的p，则使用Minkowski_距离(L_P)。 **具体的其他距离可以参阅文档**
- **metric_params**：
  - 度量函数的附加关键字参数，设置应为dict（字典）形式。
- **n_jobs**：
  - 要为邻居搜索的并行作业的数量。`None`指1，除非在 joblib.parallel_backend背景。`-1`意味着使用所有处理器，若要了解相关的知识应该具体查找一下。

- **属性**
  - **classes_**分类器已知的类标签
  - **effective_metric_**

    使用的距离度量。与`metric`参数或其同义词相同，例如，如果`metric`参数设置为“ minkowski”且`p`参数设置为2，则为“ euclidean” 。
  - **effective_metric_params_***字典*度量功能的其他关键字参数。对于大多数指标，`metric_params`参数将与参数相同，但`p`如果`effective_metric_`属性设置为“ minkowski” ，则也可能包含 参数值。
  - **n_samples_fit_** *int*拟合数据中的样本数。
  - **outputs_2d**：输出bool值在拟合期间当形状为（n_samples，）或（n_samples，1）时为False，否则为True。
- **方法**：
  - **fit(X，y)**以X为训练数据，y为目标值拟合模型，返回分类器
  - **get_params`(*deep=True*)**获取此估计器的参数。

        - **deep*bool, default=True***
        - **params*dict***返回一个字典
    - **kneighbors(*X=None*, *n_neighbors=None*, *return_distance=True*)**
        - **X** 查询X这个点的最近的距离和样本
        - **n_neighbors*int, default=None*** k值，默认是分类的k值
        - **return_distance*bool, default=True*** 是否返回距离
        - 返回的是两个ndarray。第一个是距离，第二个是最近的样本索引
    - **kneighbors_graph(*X=None*, *n_neighbors=None*, *mode='connectivity'*)**
        - 计算最邻域的图
        - **X** 查询X这个点的最近的距离和样本
        - **n_neighbors： int, default=None*** k值，默认是分类的k值
        - **model：{'connectivity'，'distance'}，默认='connectivity'**返回矩阵的类型：'connectivity'将返回具有1和0的连通性矩阵，在'distance'中，边为点之间的欧几里得距离。
    - **predict（X）**预测 数据，返回预测后的标签
    - **predict_proba（*X*）** 测试数据X的返回概率估计
    - **score（*X*，*y*，*sample_weight =无*）**返回给定X和y的平均准确度
    - **set_params（*参数*）**设置此估计其的参数
- **备注**
    - 关于最近邻居算法，如果发现两个邻居，neighbor `k+1` 和 `k`，具有相同的距离，但标签不同，则结果将取决于训练数据的顺序。

- **距离公式metric**

identifier	class name	args	distance function
“euclidean”	EuclideanDistance		`sqrt(sum((x - y)^2))`
“manhattan”	ManhattanDistance		`sum(|x - y|)`
“chebyshev”	ChebyshevDistance		`max(|x - y|)`
“minkowski”	MinkowskiDistance	p	`sum(|x - y|^p)^(1/p)`
“wminkowski”	WMinkowskiDistance	p, w	`sum(|w * (x - y)|^p)^(1/p)`
“seuclidean”	SEuclideanDistance	V	`sqrt(sum((x - y)^2 / V))`
“mahalanobis”	MahalanobisDistance	V or VI	`sqrt((x - y)' V^-1 (x - y))`

- **KNeighborsRegressor** k近邻回归算法，同上

```
from sklearn.neighbors import KNeighborsClassifier
x = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]
estimator = KNeighborsClassifier(n_neighbors=1)
# 使用fit方法进行训练
estimator.fit(x, y)

estimator.predict([[1]])
```

2.线性回归linearmodel

2.1 线性回归LinearRegression

- **线性回归**

```
sklearn.linear_model.LinearRegression(*, fit_intercept=True, normalize=False, copy_X=True, n_jobs=None, positive=False)
```

- **参数**
  - **fit_intercept: bool, default=True***
    - 默认为True,是否计算该模型的截距。如果使用中心化的数据，可以考虑设置为False,不考虑截距。注意这里是考虑，一般还是要考虑截距
  - **normalize: bool, default=False***
    - 默认为false. 当fit_intercept设置为false的时候，这个参数会被自动忽略。如果为True,回归器会标准化输入参数：减去平均值，并且除以相应的二范数。当然啦，在这里还是建议将标准化的工作放在训练模型之前。通过设置**sklearn.preprocessing.StandardScaler**来实现，而在此处设置为false
  - **copy_X: bool, default=True***
    - 默认为True, X会被拷贝，False时可能会被更改
  - **n_jobs: int, default=None***
    - 默认为1. 当-1时默认使用全部CPUs
  - **positive: bool, default=False***
    - 默认为False，True时候，强制 系数为正数，新版本添加的
- **属性**
  - **coef_： array of shape (n_features,) or (n_targets, n_features)**
    - 线性回归问题的估计系数。如果在拟合过程中传递了多个目标（y 2D），则这是一个2D形状的数组（n_targets，n_features），而如果仅传递了一个目标，则这是长度n_features的一维数组。
  - **rank_： int**
    - 矩阵X的秩
  - **singular_： array of shape (min(X, y),)**

        - X的奇异值
    - **intercept_： float or array of shape (n_targets,)**
        - 截距，如果fit_intercept=False则为0.0
- **方法**
    - **fit(X,y,sample_weight=None):**
        - 训练器
    - **get_params(deep=True)**
        - deep： bool 默认为True
        - 返回字典，估计器的各个设置参数
    - **predict（X)**
        - 用估计其预测X，得到预测值
    - **score（X，y， sample_weight）：**
        - 返回预测的评价：这里用的是R^2 = (1-u/v)，u是真实值和预测值的均方误差，v是真实值和预测值平均值的均方误差，越接近1.0越好
    - **set_params()**
        - 该估计其的设置
- **备注**

从实现的角度来看，这只是包装为预测对象的普通最小二乘（scipy.linalg.lstsq）或非负最小二乘（scipy.optimize.nnls）。

2.2 罗森回归Lasso

- **罗森回归（L1正则化）**
- ```
  sklearn.linear_model.Lasso
  (alpha=1.0, *, fit_intercept=True, normalize=False, precompute=False, copy_X=True,
  max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None,
  selection='cyclic')
  ```
- **参数**
    - **alpha: float, default=1.0**
        - L1正则化系数，浮点数，默认是1.0，alpha=0时候相当于没有正则化项，但是不建议这么做，因为这个API专门是为了L1正则化的
    - **fit_intercept: bool, default=True***
        - 默认为True,是否计算该模型的截距。如果使用中心化的数据，可以考虑设置为False,不考虑截距。注意这里是考虑，一般还是要考虑截距
    - **normalize: bool, default=False***
        - 默认为false. 当fit_intercept设置为false的时候，这个参数会被自动忽略。如果为True,回归器会标准化输入参数：减去平均值，并且除以相应的二范数。当然啦，在这里还是建议将标准化的工作放在训练模型之前。通过设置**sklearn.preprocessing.StandardScaler**来实现，而在此处设置为false
    - **precompute: ‘auto’, bool or array-like of shape (n_features, n_features), default=False**
  ```

        - 是否使用预先计算的Gram矩阵来加快计算速度。如果设置`'auto'`让我们决定。语法矩阵也可以作为参数传递。对于稀疏输入，此选项始终`True`是保留稀疏性。
    - **copy_X: bool, default=True***
        - 默认为True, X会被拷贝，False时可能会被更改
    - **max_iter: int, default=1000**
        - 最大迭代次数，默认1000
    - **tol :float, default=1e-4**
        - 意思应该是收敛的最小偏差。但是解释的和VIF以及容忍度有关，暂时不知道到底有关系没有。
        - **vif方差膨胀因子**，1/(1-R^2) 衡量特征X之间有没有线性关系（通常我们不希望有线性关系），越接近于1则表示没有共线性
        - **容忍度** “VIF的倒数
    - **warm_start: bool, default=False**
        - 默认为False，如果为True，那么使用前一次训练结果继续训练，否则从头开始训练。
    - **positive: bool, default=False***
        - 默认为False，True时候，强制 系数为正数，新版本添加的
    - **random_state :int, RandomState instance, default=None**
        - 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。
    - **selection: {‘cyclic’, ‘random’}, default=‘cyclic’**
        - 一个字符串，可以选择‘cyclic’或者‘random’。它指定了当每轮迭代的时候，选择权重向量的哪个分量来更新。
            - ‘ramdom’：更新的时候，随机选择权重向量的一个分量来更新。
            - ‘cyclic’：更新的时候，从前向后一次选择权重向量的一个分量来更新
- **属性**
    - **coef_ ： array of shape (n_features,) or (n_targets, n_features)**
        - 线性回归问题的估计系数，损失函数cost中的w参数
    - **dual_gap_ ： float or ndarray of shape (n_targets,)**
        - 对于给定的L1正则化参数alpha，返回优化到最后的迭代完成后，最后一次的迭代的对偶间隙值
        - dual gap 是一个衡量迭代收敛的指标，太tm深奥了
    - **sparse_coef_ : sparse matrix of shape (n_features, 1) or (n_targets, n_features)**
        - 对于coef_ 的稀疏表示
    - **intercept_： float or array of shape (n_targets,)**
        - 截距，如果fit_intercept=False则为0.0
    - **n_iter_ ： int or list of int**
        - 运行到指定公差的迭代次数
- **方法**
    - **fit(X,y,sample_weight=None):**
        - 训练器
    - **get_params(deep=True)**
        - deep： bool 默认为True

        - 返回字典，估计器的各个设置参数
    - **predict（X)**
        - 用估计其预测X，得到预测值
    - **score（X，y， sample_weight）：**
        - 返回预测的评价：这里用的是R^2 = (1-u/v)，u是真实值和预测值的均方误差，v是真实值和预测值平均值的均方误差，越接近1.0越好
    - **set_params()**
        - 该估计其的设置

2.3 岭回归Ridge

```
sklearn.linear_model.Ridge
(alpha=1.0, *, fit_intercept=True, normalize=False, copy_X=True, max_iter=None, tol=0.001,
solver='auto', random_state=None)[source]
```

- **参数**
    - **alpha: float, default=1.0**
        - **L2正则化系数**
    - **fit_intercept: bool, default=True***
        - 默认为True,是否计算该模型的截距。如果使用中心化的数据，可以考虑设置为False,不考虑截距。注意这里是考虑，一般还是要考虑截距
    - **normalize: bool, default=False***
        - 默认为false. 当fit_intercept设置为false的时候，这个参数会被自动忽略。如果为True,回归器会标准化输入参数：减去平均值，并且除以相应的二范数。当然啦，在这里还是建议将标准化的工作放在训练模型之前。通过设置**sklearn.preprocessing.StandardScaler**来实现，而在此处设置为fals
    - **copy_X: bool, default=True**
        - 默认为True, X会被拷贝，False时可能会被更改
    - **max_iter: int, default=1000**
        - 最大迭代次数，默认1000
    - **tol :float, default=1e-3**
        - **迭代的精度**
    - **solver：{'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga'}, default='auto'**
        - **求解最优化问题的算法**
        - **'auto'**根据数据类型自动选择求解器。
        - **'svd'**使用X的奇异值分解来计算Ridge系数。对于奇异矩阵比'cholesky'更稳定。
        - **'cholesky'**使用标准的scipy.linalg.solve函数来获得闭合形式的解。
        - **'sparse_cg'**使用在scipy.sparse.linalg.cg中找到的共轭梯度求解器。作为迭代算法，这个求解器比大规模数据（设置tol和max_iter的可能性）的"cholesky"更合适。
        - **'lsqr'**使用专用的正则化最小二乘常数scipy.sparse.linalg.lsqr。它是最快的，但可能不是在旧的scipy版本可用。它还使用迭代过程。
        - **'sag'**使用随机平均梯度下降。它也使用迭代过程，并且当n_samples和n_feature都很大时，通常比其他求解器更快。注意，"sag"快速收敛仅在具有近似相同尺度的特征上被保证。您可以使用sklearn.preprocessing的缩放器预处理数据。

        - 最后四个求解器支持密集和稀疏数据。但是，当fit_intercept为True时，只有'sag'支持稀疏输入。
    - **random_state :int, RandomState instance, default=None**
        - `solver` == ‘sag’ or ‘saga’时用于洗牌数据
        - 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。
- **属性**
    - **coef_： array of shape (n_features,) or (n_targets, n_features)**
        - 线性回归问题的估计系数，损失函数cost中的w参数
    - **intercept_： float or array of shape (n_targets,)**
        - 截距，如果fit_intercept=False则为0.0
    - **n_iter_： int or list of int**
        - 运行到指定公差的迭代次数
- **方法**
    - **fit(X,y,sample_weight=None):**
        - 训练器
    - **get_params(deep=True)**
        - deep： bool 默认为True
        - 返回字典，估计器的各个设置参数
    - **predict（X)**
        - 用估计其预测X，得到预测值
    - **score（X，y， sample_weight）：**
        - 返回预测的评价：这里用的是R^2 = (1-u/v)，u是真实值和预测值的均方误差，v是真实值和预测值平均值的均方误差，越接近1.0越好
    - **set_params()**
        - 该估计其的设置

3.逻辑回归LogisticRegression

```
sklearn.linear_model.LogisticRegression
(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1,
class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto',
verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
```

- **参数**
    - **penalty： {‘l1’, ‘l2’, ‘elasticnet’, ‘none’}, default=‘l2’**
        - 字符串‘l1’或‘l2’或‘elasticnet‘或者‘none‘
        - **‘l1’** L1正则化 使用SAGA求解器的惩罚为l1不能用其他
        - **‘l2’** L2正则化，'newton-cg'，'sag'和'lbfgs'只支持l2正则化
        - **’elasticnet‘** L1和L2联合正则化

        - ElasticNet在我们发现用Lasso回归太过(太多特征被稀疏为0),而岭回归也正则化的不够(回归系数衰减太慢)的时候
        - 仅“ saga”求解器支持“ elasticnet”
    - **'none'** 无正则化
- **dual：bool, default=False**
    - Dual只适用于正则化相为l2的'liblinear'的情况，通常样本数大于特征数的情况下，默认为False。
- **tol :float, default=1e-4**
    - 意思应该是收敛的最小偏差。但是解释的和VIF以及容忍度有关，暂时不知道到底有关系没有。
- **C： float, default=1.0**
    - C为正则化系数λ的倒数，必须为正数，默认为1。和SVM中的C一样，值越小，代表正则化越强
- **fit_intercept: bool, default=True***
    - 默认为True,是否计算该模型的截距。如果使用中心化的数据，可以考虑设置为False,不考虑截距。注意这里是考虑，一般还是要考虑截距
- **intercept_scaling： float, default=1**
    - 仅在正则化项为'liblinear'，且fit_intercept设置为True时有用。
    - 变为[x，self.intercept_scaling]，即，将常量值等于intercept_scaling的“合成”特征附加到实例矢量。截距变为。intercept_scaling * synthetic_feature_weight
- **class_weight :dict or 'balanced', default=None**
    - 与形式的类有关的权重。如果未给出，则所有类均应具有权重。`{class_label: weight}`
    - “balanced”模式使用y的值来自动调整与输入数据中的类频率成反比的权重。`n_samples / (n_classes * np.bincount(y))`
    - 请注意，如果指定了sample_weight，则这些权重将与sample_weight（通过fit方法传递）相乘。
- **random_state : int, RandomState instance, default=None**
    - 在`solver`=='sag'，'saga'或'liblinear'时，用于洗牌数据的随机数
    - 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。
- **solver : {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}, default='lbfgs'**
    - solver参数决定了我们对逻辑回归损失函数的优化方法，有四种算法可以选择。
    - a) **liblinear**：使用了开源的liblinear库实现，内部使用了坐标轴下降法来迭代优化损失函数。
    - b) **lbfgs**：拟牛顿法的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。
    - c) **newton-cg**：也是牛顿法家族的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。
    - d) **sag**：即随机平均梯度下降，是梯度下降法的变种，和普通梯度下降法的区别是每次迭代仅仅用一部分的样本来计算梯度，适合于样本数据多的时候。
        - 对于小型数据集，**“ liblinear”**是一个不错的选择，而对于大型数据集，**“ sag”**和“ **saga”**则更快。
        - 对于多类问题，只有**'newton-cg'，'sag'，'saga'**和**'lbfgs'**处理多项式损失。**“ liblinear”**仅限于“一站式”计划。
        - **'newton-cg'，'lbfgs'，'sag'**和**'saga'**处理L2或没有正则项
        - **'liblinear'和'saga'**也可以处理L1
        - **“ saga”**还支持**“ elasticnet”**惩罚
        - **'liblinear'**不支持设置`penalty='none'`

请注意，只有在比例大致相同的要素上才能确保“ sag”和“ saga”快速收敛。您可以使用sklearn.preprocessing中的缩放器对数据进行预处理。

- **max_iter : int, default=100**
  - 最大迭代次数
- **multi_class :{'auto', 'ovr', 'multinomial'}, default='auto'**
  - 默认是ovr（one-vs-rest(OvR)，**还有一种**Mvm（many-vs-many）**，如果是二元逻辑回归，ovr和multinomial并没有任何区别，区别主要在多元逻辑回归上。
  - **ovr**
    - 不论是几元回归，都当成二元回归来处理
    - OvR相对简单，但分类效果相对略差
    - 如果选择了ovr，则4种损失函数的优化方法liblinear，newton-cg,lbfgs和sag都可以选择
  - **MVM**
    - mvm从从多个类中每次选两个类进行二元回归。如果总共有T类，需要T(T-1)/2次分类。
    - MvM分类相对精确，但是分类速度没有OvR快。
    - 如果选择了multinomial,则只能选择newton-cg, lbfgs和sag了。
- **verbose : int, default=0**
  - 日志冗长度int：
    - 冗长度；0：不输出训练过程；
    - 1：偶尔输出；
- **warm_start : bool, default=False**
  - 是否热启动，如果是，则下一次训练是以追加树的形式进行（重新使用上一次的调用作为初始化）。布尔型，默认False。
- **n_jobs : int, default=None**
  - 并行数，int：个数；-1：跟CPU核数一致；1:默认值。
- **l1_ratio : float, default=None**
  - 表征L1和L2的联合惩罚系数
  - 只在`penalty='elasticnet'`.时使用，范围0到1之间
  - 等于1时候相当于l1正则化
  - 等于0时候相当于l2正则化

- **属性**
  - **classes_ : ndarray of shape (n_classes,)**
    - 分类器已知的类别标签
  - **coef_ : ndarray of shape (1, n_features) or (n_classes, n_features)**
    - `coef_`当给定问题为二分类时，其形状为（1，n_features）
    - 特别地，当时`multi_class='multinomial'`，`coef_`对应于结果1（真），并且`-coef_`对应于结果0（假）。
  - **intercept_ : ndarray of shape (1,) or (n_classes,)**
    - 截距
    - 如果`fit_intercept`设置为False，则截距设置为零。
    - `intercept_`当给定问题二分类时，其形状为（1，）。特别地，当时`multi_class='multinomial'`，`intercept_`对应于结果1（真），并且`-intercept_`对应于结果

0（假）。

- **n_iter_ : ndarray of shape (n_classes,) or (1,)**
  - 所有类的实际迭代数，对于liblinear求解器，仅给出所有类的最大迭代次数。

- **方法**
  - **decision_function（X)**
    - 预测样本的置信度概率
  - **densify（)**
    - 将系数矩阵转换为密集阵列格式。
  - **fit(*X*, *y*, *sample_weight=None*)**
    - 训练分类器模型
  - **get_params(deep=True)**
    - deep： bool 默认为True
    - 返回字典，估计器的各个设置参数
  - **predict（X)**
    - 用估计其预测X，得到预测值
  - **predict_log_proba（*X*)**
    - 返回一个数组，数组的元素一次是 X 预测为各个类别的概率的对数值。
  - **predict_proba（X)：**
    - 返回一个数组，数组元素一次是 X 预测为各个类别的概率的概率值。
  - **score(X,y,sample_weight)：**
    - 返回（X，y）上的预测准确率（accuracy）。
  - **sparsify()**：
    - 将系数矩阵转换为稀疏格式。
  - **set_params()**
    - 该估计其的设置
- **备注**
  - 底层的C实现在拟合模型时使用随机数生成器选择特征。因此，对于相同的输入数据具有略微不同的结果并不罕见。如果发生这种情况，请尝试使用较小的tol参数。

    在某些情况下，预测输出可能与独立liblinear的输出不匹配。请参见 叙述文档

4.随机梯度下降SGD

4.1 随机下降回归SGDRegressor

```
sklearn.linear_model.SGDRegressor
(loss='squared_loss', *, penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
max_iter=1000, tol=0.001, shuffle=True, verbose=0, epsilon=0.1, random_state=None,
learning_rate='invscaling', eta0=0.01, power_t=0.25, early_stopping=False,
validation_fraction=0.1, n_iter_no_change=5, warm_start=False, average=False)
```

- **参数**
  - **loss： str, default='squared_loss'**
    - 损失函数，字符串 默认是最小二乘损失
    - 'huber'： 平滑的最小二乘，在小于参数ε时为最小二乘，当Huber损失在[0-δ,0+δ]之间时，等价为MSE，而在[-∞,δ]和[δ,+∞]时为MAE。
    - " epsilon_insensitive" 忽略小于 ε的误差。使用的是SVR的损失函数。
    - "squared_epsilon_insensitive" 超过ε容差后变为平方损耗
    - https://scikit-learn.org/stable/modules/sgd.html#sgd-mathematical-formulation 参考公式
  - **penalty : {'l2', 'l1', 'elasticnet'}, default='l2'**
    - 默认值为" l2"，这是线性SVM模型的标准正则化器。" l1"和" elasticnet"可能会给模型带来稀疏性（功能选择），而" l2"是无法实现的。
      - **" l2"** L2正则化
      - **'l1'** L1正则化
      - **'elasticnet'** L1和L2的联合
  - **alpha： float, default=0.0001**
    - 正则化系数，越大正则化惩罚力度越大
  - **l1_ratio： float, default=0.15**
    - **'elasticnet'的参数** 其中0 <= l1_ratio <=1。l1_ratio = 0对应于L2惩罚，l1_ratio = 1到L1。仅在`penalty`为" elasticnet"时使用。
  - **fit_intercept： bool, default=True**
    - 默认为True,是否计算该模型的截距。如果使用中心化的数据，可以考虑设置为False,不考虑截距。注意这里是考虑，一般还是要考虑截距
  - **max_iter： int, default=1000**
    - 最大迭代次数
  - **tol： float, default=1e-3**
    - 迭代收敛的精度
  - **shuffle： bool, default=True**
    - 在每一轮迭代之后是或否重新洗牌
  - **verbose : int, default=0**
    - 整数，可选的，控制调试信息的详尽程度
  - **epsilon : float, default=0.1**
    - 仅当`loss`是" huber"，" epsilon_insensitive"或" squared_epsilon_insensitive"时
    - 调整这些loss函数中的ε阈值
  - **random_state : int, RandomState instance, default=None**
    - 当`shuffle`设置为`True`时候洗牌的参数
    - 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。

- **learning_rate： string, default='invscaling'**
  - 字符串，学习率的侧罗
  - 'constant'：eta=eta0 ，常数，学习率不变
  - 'optimal'：eta = 1.0 / (alpha * (t + t0))默认值， 优化的学习率，会随着梯度调整
  - 'invscaling'：eta = eta0 / pow(t, power_t)
  - 'adaptive' 自适应
- **eta0 : double, default=0.01**
  - 浮点数，参与learning_rate计算，默认值为0
- **power_t : double, default=0.25**
  - 参与learning_rate计算，默认值为0.5
- **early_stopping : bool, default=False**
  - 当验证分数没有提高时是否使用提前停止来终止训练
  - 如果设置为True，它将自动预留一部分训练数据作为验证，并在该`score`方法返回的验证份数无法提高时终止训练
- **validation_fraction : float, default=0.1**
  - 预留的训练数据比例作为早期停止的验证集。必须介于0和1之间。仅在`early_stopping`为True时使用。
- **n_iter_no_change : int, default=5**
  - 训练器训练到无提升的情况后再迭代的次数
- **warm_start : bool, default=False**
  - 设置为True时，使用之前的拟合得到的解继续拟合
- **average : bool or int, default=False**
  - 布尔值，整数，可选的。True时，计算平均SGD权重并存储于coef_属性中。设置为大于1的整数时，拟合使用过的样本数达到average时，开始计算平均权重

- **属性**
  - **coef_**：数组，shape=(1, n_features)二元分类；(n_classes, n_features)多元分类
    - 训练器的参数w
  - **intercept_**：数组，决策函数中常量b。shape=(1,)二元分类；(n_classes,)多元分类
    - 截距
  - **average_intercept_ ： ndarray of shape (1,)**
    - 平均截距项。仅在时可用`average=True`。
  - **n_iter：整数**，
    - 训练结束时，实际的迭代次数。对于多元分类来说，该值为所有二元拟合过程中迭代次数最大的
  - **t_ ： int**
    - 权重的更新次数
- **方法**
  - **densify（）**
    - 将系数矩阵转换为密集阵列格式。
  - **fit(*X*, *y*, *sample_weight=None*)**
    - 训练分类器模型
  - **get_params(deep=True)**
    - deep： bool 默认为True

        - 返回字典，估计器的各个设置参数
    - **predict（X)**
        - 用估计其预测X，得到预测值
    - **partial_fit（X，y，*sample_weight =无)**
        - 执行一次迭代得到的结果，在意外停止或者最后一次调用计算cost等情况下使用
    - **score(X,y,sample_weight)：**
        - 返回（X，y）上的准确率 R^2
    - **sparsify()**：
        - 将系数矩阵转换为稀疏格式。
    - **set_params()**
        - 该估计其的设置

4.2 随机下降分类SGDClassifier

```
sklearn.linear_model.SGDClassifier
(loss='hinge', *, penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
max_iter=1000, tol=0.001, shuffle=True, verbose=0, epsilon=0.1, n_jobs=None,
random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, early_stopping=False,
validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False,
average=False)
```

- **参数**
    - **loss： str, default='hinge'**
        - 'hinge'：默认的loss函数，这是SVM的函数
        - 'log'：对数损失函数，表示逻辑回归模型
        - 'modified_huber'：'hing'和'log'损失函数的结合，表现两者的优点
        - 'squared_hinge'：平方合页损失函数，表示线性SVM模型
        - 'perceptron'：感知机损失函数
        - 回归的损失函数，也可以用于分类，详情见SGD线性回归: 'squared_loss', 'huber', 'epsilon_insensitive', or 'squared_epsilon_insensitive'.
    - **penalty : {'l2', 'l1', 'elasticnet'}, default='l2'**
        - 默认值为" l2"，这是线性SVM模型的标准正则化器。" l1"和" elasticnet"可能会给模型带来稀疏性（功能选择），而" l2"是无法实现的。
            - **" l2"** L2正则化
            - **'l1'** L1正则化
            - **'elasticnet'** L1和L2的联合
    - **alpha： float, default=0.0001**
        - 正则化系数，越大正则化惩罚力度越大
    - **l1_ratio： float, default=0.15**
        - **'elasticnet'的参数** 其中0 <= l1_ratio <=1。l1_ratio = 0对应于L2惩罚，l1_ratio = 1到L1。仅在`penalty`为" elasticnet"时使用。
    - **fit_intercept： bool, default=True**

    - 默认为True,是否计算该模型的截距。如果使用中心化的数据，可以考虑设置为False,不考虑截距。注意这里是考虑，一般还是要考虑截距
- **max_iter： int, default=1000**
    - 最大迭代次数
- **tol： float, default=1e-3**
    - 迭代收敛的精度
- **shuffle： bool, default=True**
    - 在每一轮迭代之后是或否重新洗牌
- **verbose : int, default=0**
    - 整数，可选的，控制调试信息的详尽程度
- **epsilon : float, default=0.1**
- **n_jobs： int, default=None**
    - 并行计算的设置，-1 表示使用所有的cpu处理器
- **random_state : int, RandomState instance, default=None**
    - 当`shuffle`设置为`True`时候洗牌的参数
    - 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。
- **learning_rate： string, default='invscaling'**
    - 字符串，学习率的侧罗
    - 'constant'：eta=eta0 ，常数，学习率不变
    - 'optimal'：eta = 1.0 / (alpha * (t + t0))默认值， 优化的学习率，会随着梯度调整
    - 'invscaling'：eta = eta0 / pow(t, power_t)
    - 'adaptive' 自适应
- **eta0 : double, default=0.01**
    - 浮点数，参与learning_rate计算，默认值为0
- **power_t : double, default=0.25**
    - 参与learning_rate计算，默认值为0.5
- **early_stopping : bool, default=False**
    - 当验证分数没有提高时是否使用提前停止来终止训练
    - 如果设置为True，它将自动预留一部分训练数据作为验证，并在该`score`方法返回的验证份数无法提高时终止训练
- **validation_fraction : float, default=0.1**
    - 预留的训练数据比例作为早期停止的验证集。必须介于0和1之间。仅在`early_stopping`为True时使用。
- **n_iter_no_change : int, default=5**
    - 训练器训练到无提升的情况后再迭代的次数
- **warm_start : bool, default=False**
    - 设置为True时，使用之前的拟合得到的解继续拟合
- **average : bool or int, default=False**
    - 布尔值，整数，可选的。True时，计算平均SGD权重并存储于coef_属性中。设置为大于1的整数时，拟合使用过的样本数达到average时，开始计算平均权重
- **class_weight :dict or 'balanced', default=None**

        - 与形式的类有关的权重。如果未给出，则所有类均应具有权重。`{class_label: weight}`
        - “balanced”模式使用y的值来自动调整与输入数据中的类频率成反比的权重。`n_samples / (n_classes * np.bincount(y))`
        - 请注意，如果指定了sample_weight，则这些权重将与sample_weight（通过fit方法传递）相乘
- **属性**
    - **coef_**：数组，shape=(1, n_features)二元分类；(n_classes, n_features)多元分类
        - 训练器的参数w
    - **intercept_**：数组，决策函数中常量b。shape=(1,)二元分类；(n_classes,)多元分类
        - 截距
    - **n_iter：整数**，
        - 训练结束时，实际的迭代次数。对于多元分类来说，该值为所有二元拟合过程中迭代次数最大的
    - **t_ : int**
        - 权重的更新次数
    - **loss_function_**：使用的损失函数
    - **classes_*array of shape (n_classes,)***
- **方法**
    - **decision_function(X)**：
        - 对样本预测置信度得分
        - 二分类时候， 概率>0则为分类为1
    - **densify（）**
        - 将系数矩阵转换为密集阵列格式。
    - **fit(*X*, *y*, *sample_weight=None*)**
        - 训练分类器模型
    - **get_params(deep=True)**
        - deep： bool 默认为True
        - 返回字典，估计器的各个设置参数
    - **predict（X)**
        - 用估计其预测X，得到预测值
    - **partial_fit（X，y，*sample_weight =无)**
        - 执行一次迭代得到的结果，在意外停止或者最后一次调用计算cost等情况下使用
    - **score(X,y,sample_weight)：**
        - 返回（X，y）上的的平均准确度
    - **sparsify()**：
        - 将系数矩阵转换为稀疏格式。
    - **set_params()**
        - 该估计其的设置

5.支持向量机SVM

sklearn的SVM分为 SVC支持向量分类和SVR支持向量回归，同时还有简单的线性分类和线性回归

5.1 linearSVC 线性SVC分类

```
sklearn.svm.LinearSVC
(penalty='l2', loss='squared_hinge', *, dual=True, tol=0.0001, C=1.0, multi_class='ovr',
fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None,
max_iter=1000)
```

- **参数**
  - **penalty** : string, 'l1' or 'l2' (default='l2')

    默认值为" l2"，这是线性SVM模型的标准正则化器。" l1"和" elasticnet"可能会给模型带来稀疏性（功能选择），而" l2"是无法实现的。

    - **" l2"** L2正则化
    - **'l1'** L1正则化
    - **'elasticnet'** L1和L2的联合
  - **loss** : string, 'hinge' or 'squared_hinge' (default='squared_hinge')

    指定损失函数。

    - "hinge"是标准的SVM损失（例如由SVC类使用）
    - "squared_hinge"是hinge损失的平方。
  - **dual** : bool, (default=True)

    选择算法以解决双优化或原始优化问题。 当n_samples> n_features时，首选dual = False。
  - **tol** : float, optional (default=1e-4)

    迭代精度停止阈值
  - **C** : float, optional (default=1.0)

    常规项的参数，正则化参数的倒数。正则化的强度与C成反比。必须严格为正
  - **multi_class** : string, 'ovr' or 'crammer_singer' (default='ovr')

    如果y包含两个以上的类，则确定多类策略。

    - "ovr"训练n_classes one-vs-rest分类器，
    - 而"crammer_singer"优化所有类的联合目标。 虽然crammer_singer在理论上是有趣的，因为它是一致的，但它在实践中很少使用，因为它很少能够提高准确性并且计算成本更高。 如果选择"crammer_singer"，则将忽略选项loss，penalty和dual。
  - **fit_intercept** : boolean, optional (default=True)

    是否计算此模型的截距。 如果设置为false，则不会在计算中使用截距（即，预期数据已经居中）。
  - **intercept_scaling： float, default=1**
    - 仅在正则化项为'liblinear'，且fit_intercept设置为True时有用。
    - 变为[x，self.intercept_scaling]，即，将常量值等于intercept_scaling的"合成"特征附加到实例矢量。截距变为。intercept_scaling * synthetic_feature_weight
  - **class_weight :dict or 'balanced', default=None**
    - 与形式的类有关的权重。如果未给出，则所有类均应具有权重。`{class_label: weight}`
    - "balanced"模式使用y的值来自动调整与输入数据中的类频率成反比的权重。`n_samples / (n_classes * np.bincount(y))`
    - 请注意，如果指定了sample_weight，则这些权重将与sample_weight（通过fit方法传递）相乘。
  - **verbose** : int, (default=0)

启用详细输出。 请注意，此设置利用liblinear中的每进程运行时设置，如果启用，可能无法在多线程上下文中正常工作。

- **random_state** : int, RandomState instance or None, optional (default=None)

  控制伪随机数生成，以对双坐标下降（如果`dual=True`）进行数据混排。当`dual=False`的基础实现`LinearSVC`不是随机的并且`random_state`对结果没有影响时

  - 如果是int，则random_state是随机数生成器使用的种子;
  - 如果是RandomState实例，则random_state是随机数生成器;
  - 如果为None，则随机数生成器是np.random使用的RandomState实例。

- **max_iter** : int, (default=1000)

  要运行的最大迭代次数。

- **属性**
  - **coef**_*形状为（1，n_features）的ndarray如果n_classes == 2 else（n_classes，n_features）*
    - 训练器的系数。仅在线性内核的情况下可用。
  - **intercept**_ *ndarray形状（1），如果n_classes == 2别的（n_classes，）*
    - 截距
  - **classes**_*形状的ndarray（n_classes，）*
    - 分类的的类标签。
  - **n_iter**_ *int*
    - 所有类的最大迭代次数。
- **方法**
  - **decision_function(X)**:
    - 对样本预测置信度得分
    - 二分类时候， 概率>0则为分类为1
  - **densify（）**
    - 将系数矩阵转换为密集阵列格式。
  - **fit(*X, y, sample_weight=None*)**
    - 训练分类器模型
  - **get_params(deep=True)**
    - deep： bool 默认为True
    - 返回字典，估计器的各个设置参数
  - **predict（X)**
    - 用估计其预测X，得到预测值
  - **score(X,y,sample_weight):**
    - 返回（X，y）上的的平均准确度
  - **sparsify()**:
    - 将系数矩阵转换为稀疏格式。
  - **set_params()**
    - 该估计其的设置
- **备注**
  - 底层的C实现在拟合模型时使用随机数生成器选择特征。因此，对于相同的输入数据具有略有不同的结果并不罕见。如果发生这种情况，请尝试使用较小的`tol`参数。

基本的实现liblinear使用了稀疏的内部表示形式来表示将导致内存复制的数据。

在某些情况下，预测输出可能与独立liblinear的输出不匹配。请参见 叙述文档

5.2 SVC支持向量分类（可以做非线性）

```
sklearn.svm.SVC
(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True,
probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-
1, decision_function_shape='ovr', break_ties=False, random_state=None)
```

- **参数**
  - **C**：***float, default=1.0***默认值是1.0
    - 常规项系数，正则系数的倒数。
    - C越大，相当于惩罚松弛变量，希望松弛变量接近0，即对误分类的惩罚增大，趋向于对训练集全分对的情况，这样对训练集测试时准确率很高，但泛化能力弱。
    - C值小，对误分类的惩罚减小，允许容错，将他们当成噪声点，泛化能力较强。
  - **kernel**：***{‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’}, default=‘rbf’***
    - 核函数，默认是rbf，
    - ‘linear’:线性核函数
    - ‘poly’：多项式核函数
    - ‘rbf’：径像核函数/高斯核
    - ‘sigmod’:sigmod核函数
    - ‘precomputed’:核矩阵，precomputed表示自己提前计算好核函数矩阵，这时候算法内部就不再用核函数去计算核矩阵，而是直接用你给的核矩阵
  - **degree**：***int, default=3***
    - 多项式poly函数的维度，默认是3，选择其他核函数时会被忽略。
  - **gamma**：***{‘scale’, ‘auto’} or float, default=‘scale’***
    - ‘rbf’,‘poly’ 和‘sigmoid’的核函数参数。
      - auto’，则会选择1/n_features
      - 'scale'：默认值，1 / (n_features * X.var())作为参数
  - **coef0**：***float, default=0.0***
    - 核函数的常数项。对于‘poly’和 ‘sigmoid’有用。
  - **probability**：***bool, default=False***
    - 是否采用概率估计.默认为False 决定是否启用概率估计。需要在训练fit()模型时加上这个参数，之后才能用相关的方法：predict_proba和predict_log_proba
  - **shrinking**：***bool, default=True***
    - 是否采用shrinking heuristic方法，默认为true
  - **tol**：***float, default=1e-3***
    - 停止训练的误差值大小，默认为1e-3
  - **cache_size**：***float, default=200***
    - 核函数cache缓存大小，默认为200
  - **class_weight :dict or ‘balanced’, default=None**

        - 与形式的类有关的权重。如果未给出，则所有类均应具有权重。`{class_label: weight}`
        - “balanced”模式使用y的值来自动调整与输入数据中的类频率成反比的权重。`n_samples / (n_classes * np.bincount(y))`
        - 请注意，如果指定了sample_weight，则这些权重将与sample_weight（通过fit方法传递）相乘。
    - **verbose** : int, (default=0)

      启用详细输出。 请注意，此设置利用liblinear中的每进程运行时设置，如果启用，可能无法在多线程上下文中正常工作。
    - **max_iter** ：
        - 最大迭代次数。-1为无限制。
    - **decision_function_shape** ： ‘ovo’, ‘ovr’ or None, default=None3
        - ’ovr’决策函数one-vs-rest，shape (n_samples, n_classes)
        - one-vs-one (‘ovo’)决策函数(n_samples, n_classes *(n_classes - 1) / 2). ，始终将OVO用作多类别策略，如果为二分类则忽视该设置
    - **random_state** : int, RandomState instance or None, optional (default=None)

      控制伪随机数生成，当`probability`为False时被忽略，为True时才被弃用
        - 如果是int，则random_state是随机数生成器使用的种子;
        - 如果是RandomState实例，则random_state是随机数生成器;
        - 如果为None，则随机数生成器是np.random使用的RandomState实例。
    - **break_ties： bool，默认= False**
        - 如果为真，`decision_function_shape='ovr'`以及类> 2， predict将根据置信值打破僵局decision_function 否则，将返回绑定类中的第一类。请注意，与简单预测相比，打破平局的计算成本较高。

- **属性**
    - **class_weight_： ndarray of shape (n_classes,)**
        - 每个类的参数C，根据class_weight确定
    - **classes_ : ndarray of shape (n_classes,)**
        - 分类的标签
    - **coef_ : ndarray of shape (n_classes * (n_classes - 1) / 2, n_features)**
        - 训练器的系数，只在线性内核的时候有效
    - **dual_coef_ : ndarray of shape (n_classes -1, n_SV)**
        - 决策函数中支持向量的对偶系数
    - **fit_status_： int**
        - 如果正确拟合，则为0，否则为1
    - **intercept_： ndarray of shape (n_classes * (n_classes - 1) / 2,)**
        - 决策函数中的截距
    - **support_ : ndarray of shape (n_SV)**
        - 支持向量的索引
    - **support_vectors_ : ndarray of shape (n_SV, n_features)**
        - 各类的所有支持向量
    - **n_support_ : ndarray of shape (n_classes,), dtype=int32**

        - 各类有多少个支持向量
    - **probA_ ： ndarray of shape (n_classes * (n_classes - 1) / 2)**
    - **probB_ ： ndarray of shape (n_classes * (n_classes - 1) / 2)**
        - `probability=True`，才能输出，False时为空数组
        - `1 / (1 + exp(decision_value * probA_ + probB_))`
    - **shape_fit_ ： tuple of int of shape (n_dimensions_of_X,)**
        - 训练向量X的维度
- **方法**
    - **decision_function(X)**：
        - 对样本预测置信度得分
        - 二分类时候， 概率>0则为分类为1
    - **fit(*X, y, sample_weight=None*)**
        - 训练分类器模型
    - **get_params(deep=True)**
        - deep： bool 默认为True
        - 返回字典，估计器的各个设置参数
    - **predict（X)**
        - 用估计其预测X，得到预测值
    - **score(X,y,sample_weight)：**
        - 返回（X，y）上的的平均准确度
    - **set_params()**
        - 该估计其的设置

5.3 SVR支持向量回归

```
sklearn.svm.SVR
(*, kernel='rbf', degree=3, gamma='scale', coef0=0.0, tol=0.001, C=1.0, epsilon=0.1,
shrinking=True, cache_size=200, verbose=False, max_iter=- 1)
```

- **参数**
    - **kernel**： ***{‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’}, default=‘rbf’***
        - 核函数，默认是rbf，
        - ‘linear’:线性核函数
        - ‘poly’：多项式核函数
        - ‘rbf’：径像核函数/高斯核
        - ‘sigmod’:sigmod核函数
        - ‘precomputed’:核矩阵，precomputed表示自己提前计算好核函数矩阵，这时候算法内部就不再用核函数去计算核矩阵，而是直接用你给的核矩阵
    - **degree**： ***int, default=3***
        - 多项式poly函数的维度，默认是3，选择其他核函数时会被忽略。
    - **gamma**： ***{‘scale’, ‘auto’} or float, default=‘scale’***
        - ‘rbf’,‘poly’ 和‘sigmoid’的核函数参数。
            - auto’，则会选择1/n_features

        - 'scale'：默认值，1 / (n_features * X.var())作为参数
    - **coef0** ： ***float, default=0.0***
        - 核函数的常数项。对于‘poly’和 ‘sigmoid’有用。
    - **C**：***float, default=1.0***默认值是1.0
        - 常规项系数，正则系数的倒数。
        - C越大，相当于惩罚松弛变量，希望松弛变量接近0，即对误分类的惩罚增大，趋向于对训练集全分对的情况，这样对训练集测试时准确率很高，但泛化能力弱。
        - C值小，对误分类的惩罚减小，允许容错，将他们当成噪声点，泛化能力较强。
    - **epsilon： float, default=0.1**
        - 松弛系数，svr回归的间隔带（在间隔带内不计算损失）
    - **shrinking** ： ***bool, default=True***
        - 是否采用shrinking heuristic方法，默认为true
    - **tol** ： ***float, default=1e-3***
        - 停止训练的误差值大小，默认为1e-3
    - **verbose** : int, (default=0)

      启用详细输出。 请注意，此设置利用liblinear中的每进程运行时设置，如果启用，可能无法在多线程上下文中正常工作。
    - **max_iter** ：
        - 最大迭代次数。-1为无限制。
    - **cache_size** ： ***float, default=200***
        - 核函数cache缓存大小，默认为200

- **属性**
    - **class_weight_： ndarray of shape (n_classes,)**
        - 每个类的参数C，根据class_weight确定
    - **coef_ : ndarray of shape (n_classes * (n_classes - 1) / 2, n_features)**
        - 训练器的系数，只在线性内核的时候有效
    - **dual_coef_ : ndarray of shape (n_classes -1, n_SV)**
        - 决策函数中支持向量的对偶系数
    - **fit_status_： int**
        - 如果正确拟合，则为0，否则为1
    - **intercept_： ndarray of shape (n_classes * (n_classes - 1) / 2,)**
        - 决策函数中的截距
    - **n_support_ : ndarray of shape (n_classes,), dtype=int32**
        - 各类有多少个支持向量
    - **shape_fit_： tuple of int of shape (n_dimensions_of_X,)**
        - 训练向量X的维度
    - **support_vectors_： ndarray of shape (n_SV, n_features)**
        - 支持向量
- **方法**
    - **fit(*X*, *y*, *sample_weight=None*)**

        - 训练分类器模型
    - **get_params(deep=True)**
        - deep： bool 默认为True
        - 返回字典，估计器的各个设置参数
    - **predict（X)**
        - 用估计其预测X，得到预测值
    - **score(X,y,sample_weight):**
        - 返回（X，y）的准确度R^2
    - **set_params()**
        - 该估计其的设置

6.聚类cluster

6.1KMeans

```
sklearn.cluster.KMeans
(n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001,
precompute_distances='deprecated', verbose=0, random_state=None, copy_x=True,
n_jobs='deprecated', algorithm='auto')
```

- **参数**
    - **n_clusters： int, default=8**
        - 簇的数量，默认是8
    - **init： {‘k-means++’, ‘random’}, callable or array-like of shape (n_clusters, n_features), default=’k-means++’**
        - 初始化簇的方法
        - ‘k-means++’：能够加快收敛速度，选择的质心极可能的分散
        - ‘random’： 从初始质心随机选择进行观察
        - 如果传递了数组，则其形状应为（n_clusters，n_features），并给出初始中心
        - 如果传递了callable，则应使用参数X，n_clusters和随机状态并返回初始化。
    - **n_init： int, default=10**
        - 用不同的质心初始化值运行算法的次数，最终解是在inertia意义下选出的最优结果。
    - **max_iter*int, default=300**
        - 最大迭代次数，默认300
    - **precompute_distances*{‘auto’, True, False}, default=’auto’**
        - 预计算距离（更快，但占用更多内存）
            - 'auto'：如果n_samples * n_clusters> 1200万，则不预先计算距离。使用双精度，这相当于每个作业大约100MB的开销。
            - True始终预先计算距离。
            - False：永远不要预先计算距离。
            - 这个参数*0.23版*开始不推荐使用*：* 'precompute_distances'在0.22版中不再推荐使用，并将在1.0版中删除（重新命名0.25版）。没有作用
    - **verbose： int, default=0**

        - 默认为0,冗余模式
    - **random_state： int, RandomState instance or None, default=None**
        - 如果是int，则random_state是随机数生成器使用的种子;
        - 如果是RandomState实例，则random_state是随机数生成器;
        - 如果为None，则随机数生成器是np.random使用的RandomState实例。
    - **copy_x： bool, default=True**
        - True:对输入训练数据x所做的任何操作都是对x.copy()进行的，不改变x的原值
        - False:对x的操作会同步到x上，原地修改
    - **n_jobsint, default=None**
        - 并行运行的个数。-1:使用所有CPU.
    - **algorithm*{"auto", "full", "elkan"}, default="auto"**
        - auto:自动选择，数据稀疏选择，保持向后兼容性会选择" elkan"
        - full：采用经典的EM算法模式，数据稠密
        - elkan:通过使用三角不等式从而更有效，但不支持稀疏数据
- **属性**
    - **cluster_centers_ndarray of shape (n_clusters, n_features)**
        - 簇质心的坐标
    - **labels_： ndarray of shape (n_samples,)**
        - 样本的标签
    - **inertia_ : float**
        - 样本到其最近的聚类中心的平方距离
    - **n_iter_ : int**
        - 运行的迭代次数
- **方法**
    - **fit(*X*, *y*, *sample_weight=None*)**
        - 训练分类器模型
    - **fit_predict(X,y)**
        - 训练模拟器并预测每个样本的聚类
    - **fit_transform(X,y)**
        - 拟合并且把X映射到聚类空间，返回的是距离
    - **transform(X)**
        - 把X转换到聚类空间，返回的是距离
    - **get_params(deep=True)**
        - deep： bool 默认为True
        - 返回字典，估计器的各个设置参数
    - **predict（X)**
        - 用估计其预测X，得到预测值
    - **score(X,y,sample_weight):**
        - 返回（X，y）上的的平均准确度
    - **set_params()**
        - 该估计其的设置
- **备注**

使用Lloyd或Elkan算法可以解决k-均值问题。

平均复杂度由O（kn T）给出，其中n是样本数，T是迭代数。

最坏情况下的复杂度由O（n ^（k + 2 / p））给出，其中n = n_samples，p = n_features。

实际上，k-means算法非常快（是可用的最快的聚类算法之一），但它属于局部最小值。这就是为什么多次重新启动它会很有用。

如果算法在完全收敛之前停止（由于`tol`或`max_iter`），`labels_`并且`cluster_centers_`会不一致，即，`cluster_centers_`它将不是每个聚类中点的均值。同样，估算器将`labels_`在最后一次迭代后重新分配，以`labels_`与`predict`训练集保持一致。

7.决策树

```
sklearn.tree.DecisionTreeClassifier
(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None,
max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None,
ccp_alpha=0.0
```

- **参数**
  - **criterion: {“gini”, “entropy”}, default=”gini”**
    - 算法，用来分支的计算方法
    - ”gini” 默认是基尼系数，表示在这个属性下能选出不同类的概率
    - “entropy” 信息熵，混乱程度
  - **splitter ; {“best”, “random”}, default=”best”**
    - 特征划分点选择不熬转
    - “best” 寻找最优划分
    - “random” 随机划分
  - **max_depth： int, default=None**
    - 决策树的最大深度
  - **min_samples_split ; int or float, default=2**
    - 节点再划分时候所需的最小样本数
    - 默认是2，如果传入int则整数作为最小样本数
    - 如果为float，`min_samples_split`则为分数意思为百分比，用该份数乘以样本数的结果作为最小样本数
  - **min_samples_leaf： int or float, default=1**
    - 在叶节点处需要的最小样本数
    - 如果为int，则认为`min_samples_leaf`是最小值。
    - 如果为float，`min_samples_leaf`则为分数， 是每个节点的最小样本数。`ceil(min_samples_leaf * n_samples)`
  - **min_weight_fraction_leaf：： float, default=0.0**
    - 在所有叶节点处（所有输入样本）的权重总和中的最小加权分数。如果未提供sample_weight，则样本的权重相等。
  - **max_features： int, float or {“auto”, “sqrt”, “log2”}, default=None**

        - 划分时考虑的最大特征数
        - 默认是"None"，表示分时考虑所有的特征数
        - 如果是整数，代表考虑的特征绝对数
        - 如果是浮点数，代表考虑特征百分比，即考虑（百分比xN）取整后的特征数。其中N为样本总特征数。
        - "log2"意味着划分时最多考虑log2N个特征
        - sqrt"或者"auto"意味着划分时最多考虑根号下N个特征。
    - **random_state： int, RandomState instance or None, default=None**
        - 算法将在每个拆分中随机选择，然后找到它们之间的最佳拆分。但是最佳发现的拆分可能会因不同的运行而有所不同
        - 如果是int，则random_state是随机数生成器使用的种子;
        - 如果是RandomState实例，则random_state是随机数生成器;
        - 如果为None，则随机数生成器是np.random使用的RandomState实例。
    - **max_leaf_nodes： int, default=None**
        - 通过限制最大叶子节点数，可以防止过拟合，默认是"None”，即不限制最大的叶子节点数。
    - **min_impurity_decrease： float, default=0.0**
        - 如果节点分裂会导致混乱的减少大于或等于该值，则该节点将被分裂
    - **min_impurity_split： float, default=0**
        - 节点划分最小不纯度，这个值限制了决策树的增长，如果某节点的不纯度(基尼系数，信息增益，均方差，绝对差)小于这个阈值，则该节点不再生成子节点
    - **class_weight*dict, list of dict or “balanced”, default=None**
        - 与形式的类有关的权重。如果为None，则所有类的权重都应为1。对于多输出问题，可以按与y列相同的顺序提供字典列表
            - 注意，应当为onehot编码的编号1处都定义权重，比如四分类 [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] 而不是 [{1:1}, {2:5}, {3:1}, {4:1}].
        - 如果使用“balanced”，则算法会自己计算权重，样本量少的类别所对应的样本权重会高。默认是None
        - 如果指定了sample_weight，则这些权重将与sample_weight（通过fit方法传递）相乘
    - **ccp_alpha ; non-negative float, default=0.0**
        - 最小剪枝系数，默认为0
        - 该参数用来限制树过拟合的剪枝参数,，模型将会选择小于输入值最大α ,ccp_alpha=0时，决策树不剪枝；ccp_alpha越大，越多的节点被剪枝。Scikit learn 0.22版本新增参数。
- **属性**
    - **classes_**： 形状（n_classes）的ndarray或ndarray的列表**
        - 类标签
    - **feature_importances_： ndarray of shape (n_features,)**
        - 特征重要性。该值越高，该特征越重要。
        - 特征的重要性为该特征导致的评价准则的（标准化的）总减少量。它也被称为基尼的重要性
    - **max_features_： int**
        - 最大迭代次数
    - **n_classes_： int or list of int**
        - 类的数量
    - **n_features_： int**

        - 执行fit的时候，特征的数量。
    - **tree_： Tree instance**
        - 底层的Tree对象。
- **方法**
    - **apply（X)**
        - 返回每个样本被预测的叶子的索引
    - **cost_complexity_pruning_path（X，y，sample_weight=None)**
        - 计算修剪路径
    - **decision_path（X，check_input=True)**
        - 返回决策路径，是个稀疏矩阵
    - **fit(*X, y, sample_weight=None*)**
        - 训练分类器模型
    - **fit_predict(X,y)**
        - 训练模拟器并预测每个样本的聚类
    - **get_depth()**
        - 返回决策树的深度
    - **get_n_leaves()**
        - 返回决策树的叶子数
    - **get_params(deep=True)**
        - deep： bool 默认为True
        - 返回字典，估计器的各个设置参数
    - **predict（X)**
        - 用估计其预测X，得到预测值
    - **predict_log_proba（X)**
        - 预测X的类对数概率
    - **predict_proba（X)**
        - 预测X的概率
    - **score(X,y,sample_weight)：**
        - 返回（X，y）上的的平均准确度
    - **set_params()**
        - 该估计其的设置
- **备注**

控制树（例如`max_depth`，`min_samples_leaf`等）大小的参数的默认值会导致树完全生长和未修剪，这在某些数据集上可能非常大。为了减少内存消耗，应通过设置这些参数值来控制树的复杂性和大小。

- class sklearn.tree.DecisionTreeRegressor**回归决策树，参数和分类决策树相似，是用决策树的方法来做回归问题拟合曲线。
- **可视化**

```
# 先安装python-graphviz  用下面的方法会生成一个图文件和一个pdf
# 可视化
clf = tree.DecisionTreeClassifier(criterion='entropy')

# 生成一个pdf
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("iris")
```

8.集成学习ensemble

8.1 adaboost 分类和回归AdaBoostClassifier

```
sklearn.ensemble.AdaBoostClassifier
(base_estimator=None, *, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R',
random_state=None
```

- **参数**
  - **base_estimator： object, default=None**
    - 集成学习基本估计器，如果为None则会自动选择决策树来估计
  - **n_estimators： int, default=50**
    - 集成学习估计器的数量
  - **learning_rate： float, default=1.**
    - 学习率，缩小了每个估计器的贡献
  - **algorithm： {'SAMME', 'SAMME.R'}, default='SAMME.R'**
    - 如果为" SAMME.R"，则使用SAMME.R实际增强算法`base_estimator`必须支持类概率的计算，收敛更快
    - 如果为" SAMME"，则使用SAMME离散提升算法。
  - **random_state： int, RandomState instance or None, default=None**
    - 控制`base_estimator`在每次增强迭代中每个给定的随机种子
    - 如果是int，则random_state是随机数生成器使用的种子;
    - 如果是RandomState实例，则random_state是随机数生成器;
    - 如果为None，则随机数生成器是np.random使用的RandomState实例。
- **属性**
  - **base_estimator_： estimator**
    - 最好的估计器
  - **estimators_: list of classifiers**
    - 估计器的集合
  - **classes_： ndarray of shape (n_classes,)**
    - 类的标签
  - **n_classes_： int***
    - 类数量
  - **estimator_weights_ : ndarray of floats***

        - 每个估计器的权重
    - **estimator_errors_ : ndarray of floats**
        - 每个估计器的分类误差
    - **feature_importances_ :ndarray of shape (n_features,)**
        - 特征的重要性，请参阅文档，不明所以
- **方法**
    - **decision_function（X)**
        - 在最佳估计器上调用预测样本的置信度概率
    - **fit(*X*, *y*, *sample_weight=None*)**
        - 训练
    - **get_params(deep=True)**
        - deep： bool 默认为True
        - 返回字典，估计器的各个设置参数
    - **predict（X)**
        - 用最佳的估计其预测X，得到预测值
    - **predict_log_proba（*X*)**
        - 用最佳估计器，返回一个数组，数组的元素一次是 X 预测为各个类别的概率的对数值。
    - **predict_proba（X)：**
        - 用最佳估计器，返回一个数组，数组元素一次是 X 预测为各个类别的概率的概率值。
    - **score(X,y,sample_weight)：**
        - 用最佳估计器，返回（X，y）上的预测准确率（accuracy）。
    - **set_params()**
        - 该估计其的设置
    - **staged_decision_function（X)**
        - 每次迭代的X的置信区间概率
    - **staged_predict（X)**
        - 返回每个基分类器的预测数据集X的结果。
    - **staged_predict_proba(X)**:
        - 返回每个基分类器的预测数据集X的概率结果
    - **staged_score(X, Y)**
        - 返回每个基分类器的预测准确率。

adaboost回归

```
sklearn.ensemble.AdaBoostRegressor(base_estimator=None, *, n_estimators=50,
learning_rate=1.0, loss='linear', random_state=None)
```

- 和分类的参数相似只不过算法为loss函数，方法略有改变
- **loss** *{'linear'，'square'，'exponential'}，默认=“ linear”*

每次增强迭代后更新权重时使用的损失函数。

8.2 bagging 分类和回归BaggingClassifier

bagging分类

```
sklearn.ensemble.BaggingClassifier
(base_estimator=None, n_estimators=10, *, max_samples=1.0, max_features=1.0, bootstrap=True,
bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None,
verbose=0)
```

- **参数**
  - **base_estimator： object, default=None**
    - 集成学习基本估计器，如果为None则会自动选择决策树来估计
  - **n_estimators： int, default=50**
    - 集成学习估计器的数量
  - **max_samples： int or float, default=1.0**
    - 从从X抽取以训练每个基本估计量的样本数
      - 如果为int，则抽取`max_samples`样本。
      - 如果漂浮，则抽取样品。`max_samples * X.shape[0]`
  - **max_features： int or float, default=1.0**
    - 决定从x_train抽取去训练基估计器的特征数量。
    - 如果为int，则绘制`max_features`特征。
    - 如果是浮点的，则为比例。`max_features * X.shape[1]`
  - **bootstrap： bool, default=True**
    - 决定样本子集的抽样方式（有放回和不放回）
  - **bootstrap_features： bool, default=False**
    - 决定特征子集的抽样方式（有放回和不放回）
  - **oob_score： bool, default=False**
    - 决定是否使用包外估计（out of bag estimate）泛化误差
  - **warm_start： bool, default=False**
    - 是否热启动，为True时会从接着上一次计算开始继续计算
  - **n_jobs： int, default=None**
    - 并行计算的cpu数量，-1表示用所有的cpu
  - **verbose ; int, default=0**
    - 在拟合和预测时控制详细程度。
  - **random_state： int, RandomState instance or None, default=None**
    - 控制`base_estimator`在每次增强迭代中每个给定的随机种子
    - 如果是int，则random_state是随机数生成器使用的种子;
    - 如果是RandomState实例，则random_state是随机数生成器;
    - 如果为None，则随机数生成器是np.random使用的RandomState实例。
- **属性**

- **base_estimator_： estimator**
  - 基估计器的种类
- **n_features_ : int**
  - 拟合的特征数量
- **estimators_** : list of estimators
  - 基本估计器的列表
- **estimators_samples_** : list of arrays
  - 每个基本估计量的抽取样本的子集。
- **estimators_features_** : list of arrays
  - 每个基本估计量的特征子集。
- **classes_： ndarray of shape (n_classes,)**
  - 类的标签
- **n_classes_ : int**
  - 类数量
- **oob_score_** : float，
  - 使用包外估计这个训练数据集的得分。
- **oob_prediction_** : array of shape = [n_samples]。
  - 在训练集上用out-of-bag估计计算的预测。 如果n_estimator很小，则可能在抽样过程中数据点不会被忽略。 在这种情况下，oob_prediction_可能包含NaN。

- **方法**
  - decision_function（X）**
    - 在最佳估计器上调用预测样本的置信度概率
  - **fit(*X*, *y*, *sample_weight=None*)**
    - 训练
  - **get_params(deep=True)**
    - deep： bool 默认为True
    - 返回字典，估计器的各个设置参数
  - **predict（X)**
    - 用最佳的估计其预测X，得到预测值
  - **predict_log_proba（*X*)**
    - 用最佳估计器，返回一个数组，数组的元素一次是 X 预测为各个类别的概率的对数值。
  - **predict_proba（X）：**
    - 用最佳估计器，返回一个数组，数组元素一次是 X 预测为各个类别的概率的概率值。
  - **score(X,y,sample_weight)：**
    - 用最佳估计器，返回（X，y）上的预测准确率（accuracy）。
  - **set_params()**
    - 该估计其的设置

bagging回归

```
sklearn.ensemble.BaggingRegressor
(base_estimator=None, n_estimators=10, *, max_samples=1.0, max_features=1.0, bootstrap=True,
bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None,
verbose=0)
```

- 参数属性方法和分类差不多

8.3 随机森林RandomForest

随机森林分类器

```
sklearn.ensemble.RandomForestClassifier
(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False,
n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None,
ccp_alpha=0.0, max_samples=None)
```

- **参数**
  - **n_estimators： int, default=100**
    - 森林里的树木数量120,200,300,500,800,1200
      - 在利用最大投票数或平均值来预测之前，你想要建立子树的数量。
  - **criterion: {“gini”, “entropy”}, default=”gini”**
    - 算法，用来分支的计算方法
    - ”gini” 默认是基尼系数，表示在这个属性下能选出不同类的概率
    - “entropy” 信息熵，混乱程度
  - **max_depth： int, default=None**
    - 决策树的最大深度
  - **min_samples_split ; int or float, default=2**
    - 节点再划分时候所需的最小样本数
    - 默认是2，如果传入int则整数作为最小样本数
    - 如果为float，`min_samples_split`则为分数意思为百分比，用该份数乘以样本数的结果作为最小样本数
  - **min_samples_leaf： int or float, default=1**
    - 在叶节点处需要的最小样本数
    - 如果为int，则认为`min_samples_leaf`是最小值。
    - 如果为float，`min_samples_leaf`则为分数， 是每个节点的最小样本数。`ceil(min_samples_leaf * n_samples)`
  - **min_weight_fraction_leaf：： float, default=0.0**
    - 在所有叶节点处（所有输入样本）的权重总和中的最小加权分数。如果未提供sample_weight，则样本的权重相等。
  - **max_features： int, float or {“auto”, “sqrt”, “log2”}, default=None**
    - 划分时考虑的最大特征数
    - 默认是"None"，表示分时考虑所有的特征数

    - 如果是整数，代表考虑的特征绝对数
    - 如果是浮点数，代表考虑特征百分比，即考虑（百分比xN）取整后的特征数。其中N为样本总特征数。
    - "log2"意味着划分时最多考虑log2N个特征
    - sqrt"或者"auto"意味着划分时最多考虑根号下N个特征。
- **max_leaf_nodes： int, default=None**
    - 通过限制最大叶子节点数，可以防止过拟合，默认是"None”，即不限制最大的叶子节点数。
- **min_impurity_decrease： float, default=0.0**
    - 如果节点分裂会导致混乱的减少大于或等于该值，则该节点将被分裂
- **min_impurity_split： float, default=0**
    - 节点划分最小不纯度，这个值限制了决策树的增长，如果某节点的不纯度(基尼系数，信息增益，均方差，绝对差)小于这个阈值，则该节点不再生成子节点
- **bootstrap： bool, default=True**
    - 是否在构建树时使用放回抽样
- **oob_score： bool, default=False**
    - 是否使用包外估计来泛化精度
- **n_jobs： int, default=None**
    - 并行计算cpu使用数，-1表示使用所有cpu
- **random_state： int, RandomState instance or None, default=None**
    - 如果是int，则random_state是随机数生成器使用的种子;
    - 如果是RandomState实例，则random_state是随机数生成器;
    - 如果为None，则随机数生成器是np.random使用的RandomState实例。
- **verbose int, default=0**
    - 详细程度
- **warm_start： bool, default=False**
    - 是否热启动，为True时会从接着上一次计算开始继续计算
- **class_weight： {“balanced”, “balanced_subsample”}, dict or list of dicts, default=None**
    - 与形式的类有关的权重。如果为None，则所有类的权重都应为1。对于多输出问题，可以按与y列相同的顺序提供字典列表
        - 注意，应当为onehot编码的编号1处都定义权重，比如四分类 [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] 而不是 [{1:1}, {2:5}, {3:1}, {4:1}].
    - 如果使用“balanced”，则算法会自己计算权重，样本量少的类别所对应的样本权重会高。模式使用y值自动调整与输入数据中的类频率成反比的权重，如下所示： `n_samples / (n_classes * np.bincount(y))`
    - “ balanced_subsample”模式与“ balanced”相同，不同之处在于，权重是根据每个树生长的引导程序样本计算的。
    - 如果指定了sample_weight，则这些权重将与sample_weight（通过fit方法传递）相乘
- **ccp_alpha ; non-negative float, default=0.0**
    - 最小剪枝系数，默认为0
    - 该参数用来限制树过拟合的剪枝参数，，模型将会选择小于输入值最大α ,ccp_alpha=0时，决策树不剪枝；ccp_alpha越大，越多的节点被剪枝。Scikit learn 0.22版本新增参数。
- **max_samples： int or float, default=None**
    - 如果bootstrap为True，则从X抽取以训练每个基本估计量的样本数。

    - 如果为“无”（默认），则绘制`X.shape[0]`样本。
    - 如果为int，则抽取`max_samples`样本。
    - 如果漂浮，则抽取样品。因此， 应在间隔内。`max_samples * X.shape[0]``max_samples``(0, 1)`
- **属性**
  - **base_estimator_： estimator**
    - 基估计器的种类
  - **n_features_ : int**
    - 拟合的特征数量
  - **n_outputs_ : int**
    - 输出的数量
  - **estimators_** : list of estimators
    - 基本估计器的列表
  - **classes_： ndarray of shape (n_classes,)**
    - 类的标签
  - **n_classes_ : int**
    - 类数量
  - **oob_score_** : float，
    - 使用包外估计这个训练数据集的得分。
  - **oob_prediction_** : array of shape = [n_samples]。
    - 在训练集上用out-of-bag估计计算的预测。 如果n_estimator很小，则可能在抽样过程中数据点不会被忽略。 在这种情况下，oob_prediction_可能包含NaN。
- **方法**
  - **apply（X)**
    - 返回每个样本被预测的叶子的索引
  - **decision_path（X，check_input=True)**
    - 返回决策路径，是个稀疏矩阵
  - **fit(*X, y, sample_weight=None*)**
    - 训练分类器模型
  - **get_params(deep=True)**
    - deep： bool 默认为True
    - 返回字典，估计器的各个设置参数
  - **predict（X)**
    - 用估计其预测X，得到预测值
  - **predict_log_proba（X)**
    - 预测X的类对数概率
  - **predict_proba（X)**
    - 预测X的概率
  - **score(X,y,sample_weight)：**
    - 返回（X，y）上的的平均准确度
  - **set_params()**
    - 该估计其的设置

随机森林回归

```
sklearn.ensemble.RandomForestRegressor
(n_estimators=100, *, criterion='mse', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False,
n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0,
max_samples=None)
```

- 和分类类似，只不过算法不用基尼系数
- **criterion*{"mse", "mae"}, default="mse"***
    - 衡量分割质量的功能。支持的标准是均方误差的" mse"（等于特征选择标准的方差减少）和均值绝对误差的" mae"。

8.4 GBDT 梯度提升决策树GradientBoostingClassifier

分类

```
sklearn.ensemble.GradientBoostingClassifier
(*, loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0,
criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0,
min_impurity_split=None, init=None, random_state=None, max_features=None, verbose=0,
max_leaf_nodes=None, warm_start=False, validation_fraction=0.1, n_iter_no_change=None,
tol=0.0001, ccp_alpha=0.0)
```

- **参数**
    - **loss： {'deviance', 'exponential'}, default='deviance'**
        - 待优化的损失函数
        - 'deviance'是指对概率输出的偏差，等同于逻辑回归，GBDT使用的是和逻辑回归一眼搞得交叉熵或者叫对数几率当做损失
        - 'exponential' 指数提升
    - **learning_rate： float, default=0.1**
        - 学习率，梯度提升的复读
        - GBM设定了初始的权重值之后，每一次树分类都会更新这个值，而learning_ rate控制着每次更新的幅度。
        - 一般来说这个值不应该设的比较大，因为较小的learning rate使得模型对不同的树更加稳健，就能更好地综合它们的结果。
    - **n_estimators： int, default=100**
        - 集成学习估计器的数量
    - **subsample： float, default=1.0**
        - 训练每个决定树所用到的子样本占总样本的比例，而对于子样本的选择是随机的。
        - 用稍小于1的值能够使模型更稳健，因为这样减少了方差。
        - 一般来说用~0.8就行了，更好的结果可以用调参获得。

- **criterion： {‘friedman_mse’, ‘mse’, ‘mae’}, default=’friedman_mse’**
  - 分割的函数算法
  - ‘friedman_mse’,默认的均方误差，Friedman提供的一种均方误差计算
  - ‘mse’ 均方误差
  - ‘mae’ 均绝对误差
- **min_samples_split ; int or float, default=2**
  - 节点再划分时候所需的最小样本数
  - 默认是2，如果传入int则整数作为最小样本数
  - 如果为float，`min_samples_split`则为分数意思为百分比，用该份数乘以样本数的结果作为最小样本数
- **min_samples_leaf： int or float, default=1**
  - 在叶节点处需要的最小样本数
  - 如果为int，则认为`min_samples_leaf`是最小值。
  - 如果为float，`min_samples_leaf`则为分数， 是每个节点的最小样本数。`ceil(min_samples_leaf * n_samples)`
- **min_weight_fraction_leaf：： float, default=0.0**
  - 在所有叶节点处（所有输入样本）的权重总和中的最小加权分数。如果未提供sample_weight，则样本的权重相等。
- **max_depth： int, default=3**
  - 决策树的最大深度，默认是3
- **min_impurity_decrease： float, default=0.0**
  - 如果节点分裂会导致混乱的减少大于或等于该值，则该节点将被分裂
- **min_impurity_split： float, default=0**
  - 节点划分最小不纯度，这个值限制了决策树的增长，如果某节点的不纯度(基尼系数，信息增益，均方差，绝对差)小于这个阈值，则该节点不再生成子节点
- **init： estimator or ‘zero’, default=None**
  - 一个估计器对象，用于计算初始预测。`init`必须提供fit和predict_proba如果为“零”，则初始原始预测设置为零。默认情况下，使用`DummyEstimator`虚拟估计器预测类优先级。
- **random_state： int，RandomState实例或无，默认=无**
  - 如果是int，则random_state是随机数生成器使用的种子;
  - 如果是RandomState实例，则random_state是随机数生成器;
  - 如果为None，则随机数生成器是np.random使用的RandomState实例。
- **max_features： int, float or {“auto”, “sqrt”, “log2”}, default=None**
  - 划分时考虑的最大特征数
  - 默认是"None"，表示分时考虑所有的特征数
  - 如果是整数，代表考虑的特征绝对数
  - 如果是浮点数，代表考虑特征百分比，即考虑（百分比xN）取整后的特征数。其中N为样本总特征数。
  - "log2"意味着划分时最多考虑log2N个特征
  - sqrt"或者"auto"意味着划分时最多考虑根号下N个特征。
- **verbose int, default=0**
  - 1时候会打印详细信息，
- **max_leaf_nodes： int, default=None**
  - 通过限制最大叶子节点数，可以防止过拟合，默认是"None”，即不限制最大的叶子节点数。

    - **warm_start： bool, default=False**
        - 是否热启动，为True时会从接着上一次计算开始继续计算
    - **validation_fraction： float, default=0.1**
        - 预留的训练数据比例作为过早停止。必须在0到1之间。仅当`n_iter_no_change`设置为整数时使用。
    - **n_iter_no_change： int, default=None**
        - `n_iter_no_change`用于确定当验证分数没有改善时是否将使用提早停止来终止训练。默认情况下，将其设置为None以禁用提前停止。如果设置为数字，则它将保留`validation_fraction`训练数据的大小作为验证，并在以前的所有`n_iter_no_change`迭代次数中验证得分均未提高时终止训练。
    - **tol： float，default= 1e-4**
        - 如果损失不能改善，允许提前停止。 `n_iter_no_change如果设置为数字，训练就会停止。
    - **ccp_alpha ; non-negative float, default=0.0**
        - 最小剪枝系数，默认为0
        - 该参数用来限制树过拟合的剪枝参数,，模型将会选择小于输入值最大α ,ccp_alpha=0时，决策树不剪枝；ccp_alpha越大，越多的节点被剪枝。Scikit learn 0.22版本新增参数。
- **属性**
    - n_estimators_ ： int**
        - 提前停止时估计器的数量
    - **feature_importances_ :ndarray of shape (n_features,)**
        - 特征的重要性，类似特征用来分类的权重
    - **oob_improvement_ ： ndarray of shape (n_estimators,)**
        - loss (= deviance) 包外估计的损失值，subsample < 1.0
    - **train_score_： ndarray of shape (n_estimators,)**
        - 迭代中的损失
    - **loss_ ： LossFunction**
        - 具体`LossFunction`对象。
    - **init_ ： estimator**
        - 提供初始值的对象
    - **estimators_** : list of estimators
        - 基本估计器的列表
    - **classes_： ndarray of shape (n_classes,)**
        - 类的标签
    - **n_classes_ ： int**
        - 类数量
    - **n_features_ ： int**
        - 拟合的特征数量
    - **max_features_： int**
        - max_features_的推断
- **方法**
    - **apply（X)**
        - 返回每个样本被预测的叶子的索引
    - **decision_function（X)**

        - 在最佳估计器上调用预测样本的置信度概率
    - **fit(*X*, *y*, *sample_weight=None*)**
        - 训练分类器模型
    - **get_params(deep=True)**
        - deep： bool 默认为True
        - 返回字典，估计器的各个设置参数
    - **predict（X)**
        - 用估计其预测X，得到预测值
    - **predict_log_proba（X)**
        - 预测X的类对数概率
    - **predict_proba（X)**
        - 预测X的概率
    - **score(X,y,sample_weight)：**
        - 返回（X，y）上的的平均准确度
    - **set_params()**
        - 该估计其的设置
    - **staged_decision_function（X)**
        - 每次迭代的X的置信区间概率
    - **staged_predict（X)**
        - 返回每个基分类器的预测数据集X的结果。
    - **staged_predict_proba(X)**:
        - 返回每个基分类器的预测数据集X的概率结果

9.朴素贝叶斯naive_bayes

在scikit-learn中，提供了3中朴素贝叶斯分类算法：**GaussianNB(高斯朴素贝叶斯)、MultinomialNB(多项式朴素贝叶斯)、BernoulliNB(伯努利朴素贝叶斯)**

简单介绍：

高斯朴素贝叶斯：适用于连续型数值，比如身高在160cm以下为一类，160-170cm为一个类，则划分不够细腻。

多项式朴素贝叶斯：常用于文本分类，特征是单词，值是单词出现的次数。

伯努利朴素贝叶斯：所用特征为全局特征，只是它计算的不是单词的数量，而是出现则为1，否则为0。也就是特征等权重。

9.1 高斯贝叶斯GaussianNB

```
sklearn.naive_bayes.GaussianNB
(*, priors=None, var_smoothing=1e-09)
```

- **参数**
  - **priors： array-like of shape (n_classes,)**
    - 获取各个类标记对应的先验概率
  - **var_smoothing： float, default=1e-9**
    - 方差平滑，所有特征的最大方差部分，已添加到方差中以提高计算稳定性。
- **属性**
  - **class_count_： ndarray of shape (n_classes,)**
    - 每个类在训练时候的样本数
  - **class_prior_： ndarray of shape (n_classes,)**
    - 每个类的概率。
  - **classes_ ; ndarray of shape (n_classes,)**
    - 类标签
  - **epsilon_： float**
    - 方差绝对值和
  - **sigma_ ; ndarray of shape (n_classes, n_features)**
    - 每个类别每个特诊的方差
  - **theta_： ndarray of shape (n_classes, n_features)**
    - 每个类特征的平均值
- **方法**
  - **fit(*X, y, sample_weight=None*)**
    - 训练分类器模型
  - **get_params(deep=True)**
    - deep： bool 默认为True
    - 返回字典，估计器的各个设置参数
  - **predict（X)**
    - 用估计其预测X，得到预测值
  - **partial_fit（X，y，*sample_weight =无)**
    - 执行一次迭代得到的结果，在意外停止或者最后一次调用计算cost等情况下使用
  - **predict_log_proba（X)**
    - 预测X的类对数概率
  - **predict_proba（X)**
    - 预测X的概率
  - **score(X,y,sample_weight)：**

        - 返回（X，y）上的准确率
    - **set_params()**
        - 该估计其的设置

9.2 多项式贝叶斯MultinomialNB

```
sklearn.naive_bayes.MultinomialNB
(*, alpha=1.0, fit_prior=True, class_prior=None)
```

- **参数**
    - **alpha： float, default=1.0**
        - 拉普拉斯系数，是用来防止有的分类出现次数为0的异常情况
    - **fit_prior： bool, default=True**
        - 表示是否学习先验概率，参数为False表示所有类标记具有相同的先验概率
    - **class_prior： array-like of shape (n_classes,), default=None**
        - 类的先验概率
- **属性**
    - **class_log_prior_**：各类标记的平滑先验概率对数值，其取值会受fit_prior和class_prior参数的影响
        - 若指定了class_prior参数，不管fit_prior为True或False，class_log_prior_取值是class_prior转换成log后的结果
        - 若fit_prior参数为False，class_prior=None，则各类标记的先验概率相同等于类标记总个数N分之一
        - 若fit_prior参数为True，class_prior=None，则各类标记的先验概率相同等于各类标记个数除以各类标记个数之和
    - **classes_**
        - 类标签
    - **intercept__**：
        - 将多项式朴素贝叶斯解释的class_log_prior_映射为线性模型，其值和class_log_propr相同
    - **feature_log_prob_**：指定类的各特征概率(条件概率)对数值，返回形状为(n_classes, n_features)数组
    - **coef_:**
        - 将多项式朴素贝叶斯解释feature_log_prob_映射成线性模型，其值和feature_log_prob相同
    - **class_count_**：
        - 训练样本中各类别对应的样本数，按类的顺序排序输出
    - **feature_count_**：
        - 各类别各个特征出现的次数，返回形状为(n_classes, n_features)数组
    - **n_features_**：
        - 每个样本的特征
- **方法**
    - **fit(*X*, *y*, *sample_weight=None*)**
        - 训练分类器模型
    - **get_params(deep=True)**
        - deep： bool 默认为True
        - 返回字典，估计器的各个设置参数

- **predict（X)**
  - 用估计其预测X，得到预测值
- **partial_fit（X，y，*sample_weight =无)**
  - 执行一次迭代得到的结果，在意外停止或者最后一次调用计算cost等情况下使用
- **predict_log_proba（X)**
  - 预测X的类对数概率
- **predict_proba（X)**
  - 预测X的概率
- **score(X,y,sample_weight):**
  - 返回（X，y）上的准确率
- **set_params()**
  - 该估计其的设置

9.3伯努利贝叶斯BernoulliNB

```
sklearn.naive_bayes.BernoulliNB
(*, alpha=1.0, binarize=0.0, fit_prior=True, class_prior=None)
```

- **参数**
  - **alpha： float, default=1.0**
    - 拉普拉斯系数，是用来防止有的分类出现次数为0的异常情况
  - **fit_prior： bool, default=True**
    - 表示是否学习先验概率，参数为False表示所有类标记具有相同的先验概率
  - **class_prior： array-like of shape (n_classes,), default=None**
    - 类的先验概率
  - **binarize： float or None, default=0.0**
    - 将数据特征二值化的阈值
- **属性**
  - **class_log_prior_**：各类标记的平滑先验概率对数值，其取值会受fit_prior和class_prior参数的影响
    - 若指定了class_prior参数，不管fit_prior为True或False，class_log_prior_取值是class_prior转换成log后的结果
    - 若fit_prior参数为False，class_prior=None，则各类标记的先验概率相同等于类标记总个数N分之一
    - 若fit_prior参数为True，class_prior=None，则各类标记的先验概率相同等于各类标记个数除以各类标记个数之和
  - **classes_**
    - 类标签
  - **intercept__**：
    - 将多项式朴素贝叶斯解释的class_log_prior_映射为线性模型，其值和class_log_propr相同
  - **feature_log_prob_**：指定类的各特征概率(条件概率)对数值，返回形状为(n_classes, n_features)数组
  - **coef_:**
    - 将多项式朴素贝叶斯解释feature_log_prob_映射成线性模型，其值和feature_log_prob相同
  - **class_count_**：

        - 训练样本中各类别对应的样本数，按类的顺序排序输出
    - **feature_count_**：
        - 各类别各个特征出现的次数，返回形状为(n_classes, n_features)数组
    - **n_features_**：
        - 每个样本的特征
- **方法**
    - **fit(*X, y, sample_weight=None*)**
        - 训练分类器模型
    - **get_params(deep=True)**
        - deep： bool 默认为True
        - 返回字典，估计器的各个设置参数
    - **predict（X)**
        - 用估计其预测X，得到预测值
    - **partial_fit（X，y，*sample_weight =无)**
        - 执行一次迭代得到的结果，在意外停止或者最后一次调用计算cost等情况下使用
    - **predict_log_proba（X)**
        - 预测X的类对数概率
    - **predict_proba（X)**
        - 预测X的概率
    - **score(X,y,sample_weight)：**
        - 返回（X，y）上的准确率
    - **set_params()**
        - 该估计其的设置

10.异常检测

10.1 高斯分布检测

```
sklearn.covariance.EllipticEnvelope
(*, store_precision=True, assume_centered=False, support_fraction=None, contamination=0.1,
random_state=None)
```

- **参数**
    - **store_precision : bool, default=True**
        - 指定是否存储估计的精度
    - **assume_centered: bool, default=False**
        - 如果为True，则将计算鲁棒性位置和协方差估计的支持，并从中重新计算协方差估计，而无需将数据居中。在处理均值显着等于零但不完全为零的数据时很有用。
        - 如果为False，则可以使用FastMCD算法直接计算鲁棒位置和协方差，而无需其他处理。
    - **support_fraction； float, default=None**

        - 支持MCD原始估算的点数比例。如果为None，则将在算法中使用support_fraction的最小值：。范围是（0，1）。`[n_sample + n_features + 1] / 2`
    - **contamination： float, default=0.1**
        - 数据集的污染量，即数据集中异常值的比例。范围是（0，0.5）。
    - **random_state： int, RandomState instance or None, default=None**
        - 随机数种子
- **属性**
    - **location_： ndarray of shape (n_features,)**
        - 估计的中心
    - **covariance_： ndarray of shape (n_features, n_features)**
        - 估计的协方差矩阵
    - **precision_： ndarray of shape (n_features, n_features)**
        - 估计的伪逆矩阵
    - **support_： ndarray of shape (n_samples,)**
        - 用于计算位置和形状的可靠估计。
    - **offset_： float**
        - 偏移量用于根据原始分数定义决策函数。我们有以下关系：。偏移量取决于污染参数，其定义方式是在训练中获得预期的异常值数量（决策函数<0的样本）。`decision_function = score_samples - offset_`
    - **raw_location_： ndarray of shape (n_features,)**
        - 校正和重新加权之前的原始鲁棒估计位置。
    - **raw_covariance_： ndarray of shape (n_features, n_features)**
        - 校正和重新加权之前的原始鲁棒估计协方差。
    - **raw_support_： ndarray of shape (n_samples,)**
        - 校正和重新加权之前已用于计算位置和形状的原始鲁棒估计的观测值的掩码。
    - **dist_**： ndarray of shape (n_samples,)**
        - 训练集观测值的马氏距离。
- **方法**
    - **correct_coariance(数据)**
        - 对原始的最小协方差行列式估计值进行校正。
    - **decision_function(X)**
        - 决策函数
    - **error_norm（comp_cov [，范数，缩放比例，平方])**
        - 计算两个协方差估计量之间的均方误差。
    - **fit(*X*, *y*, *sample_weight=None*)**
        - 训练分类器模型
    - **get_params(deep=True)**
        - deep： bool 默认为True
        - 返回字典，估计器的各个设置参数
    - **get_precision()**
        - 两个随机变量的协方差矩阵(covariance matrix)的逆矩阵存在，那这个逆矩阵就被称为precison matrix.

- **predict（X)**
  - 用估计其预测X，得到预测值
- **mahalanobis(X)**
  - 计算给定观测值的马氏距离的平方。
- **score(X,y,sample_weight)：**
  - 返回（X，y）上的准确率
- **set_params()**
  - 该估计其的设置

10.2 one vs SVM

```
sklearn.svm.OneClassSVM(*, kernel='rbf', degree=3, gamma='scale', coef0=0.0, tol=0.001,
nu=0.5, shrinking=True, cache_size=200, verbose=False, max_iter=- 1)
```

- **参数**
  - **kernel*{‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’}, default=’rbf’**
    - 指定算法中要使用的内核类型，默认是高斯内核
    - 线性，多项式，高斯，sigmoid，预先计算5个可以选择
  - **degree*int, default=3**
    - 多项式核函数的度
  - **gamma*{‘scale’, ‘auto’} or float, default=’scale’***
    - “ rbf”，“ poly”和“ Sigmoid”的内核系数。
  - **coef0*float, default=0.0**
    - “ poly”和“ sigmoid”内核函数的参数
  - **tol*float, default=1e-3***
    - 停止标准的公差。
  - **nu*float, default=0.5**
    - 训练误差分数的上限，支持向量分数的下限。应该在间隔（0，1]中。默认情况下，将采用0.5。
  - **shrinking*bool, default=True**
    - 是否使用缩小的启发式方法。
  - **cache_size*float, default=200**
    - 指定内核缓存的大小（以MB为单位）。
  - **verbose*bool, default=False**
    - 是否输出详细信息
  - **max_iter*int, default=-1**
    - 对求解器内的迭代进行硬性限制，或者为-1（无限制）
- **属性**
  - **class_weight_**
    - 每个类的权重
  - **coef_**
    - 系数

- **dual_coef_**
  - 决策函数中支持向量的系数。
- **fit_status_**
  - 是都正常运行
- **intercept_**
  - 决策函数中的常数。
- **n_support_**
  - 每个类的支持向量数量
- **offset_**
  - 偏移量用于根据原始分数定义决策函数。我们具有以下关系：Decision_function = score_samples- `offset_`。偏移量与之相反，`intercept_`并且为与其他异常值检测算法保持一致而提供。
- **shape_fit_**
  - 训练向量的数组维数`X`
- **support_**
  - 支持向量的指标。
- **support_vectors_**
  - 支持向量。

- **方法**
  - **decision_function(X)**
    - 决策函数
  - **fit(*X*, *y*, *sample_weight=None*)**
    - 训练分类器模型
  - **get_params(deep=True)**
    - deep： bool 默认为True
    - 返回字典，估计器的各个设置参数
  - **predict（X)**
    - 用估计其预测X，得到预测值
  - **set_params()**
    - 该估计其的设置
  - **set_samples(X)**
    - 样本的原始评分功能