In [ ]:

```
# 各列分别为：cons 0 1 2 3 01 02 03 12 13 23 012 013 023 123
# 未生成的列：00 000 0012 0123等
polyres[:1]
```

2.5 自定义转换器

将任何一个已有的Python函数转化为sklearn中的转换器，用于在统一的流程中清理数据。

class sklearn.preprocessing.FunctionTransformer(

```
    func = None : 用于定义转换器的python函数
    inverse_func  = None  : 用于逆转换的函数
    validate = True : bool, 在调用函数前是否对数据做检查
    accept_sparse = False : boolean, 是否允许转换函数接受稀疏矩阵格式
    pass_y = False : bool, 转换函数是否会一并提交因变量y
    kw_args : dict, 转换函数使用的参数列表
    inv_kw_args : dict, 逆转换函数使用的参数列表
```

)

FunctionTransformer类无属性

In [ ]:

```
import numpy as np
from sklearn.preprocessing import FunctionTransformer

# np.log1p ==> log(1 + x)
transformer = FunctionTransformer(np.log1p)
X = np.array([[0, 1],
              [2, 3]])
transformer.transform(X)
```

2.6 实战练习

尝试对iris数据使用本章学到的知识进行自变量的标准化、稳健标准化、分位数转换，并生成所有的交互项和高次项。

尝试分别使用sklearn和pandas将iris中的因变量转换为哑变量组。

```
    注意：使用sklearn转化时，可能需要先将因变量转化为二维矩阵结构
```

3 特征选择与信息浓缩

3.1 基于离散程度进行筛选

移除变异度明显过低的特征。

class sklearn.feature_selection.VarianceThreshold(threshold = 0.0)

默认设定将会移除模型中的常量

VarianceThreshold类的属性:

variances_ : array, shape (n_features,), 各特征列的方差值

In [ ]:

```
from sklearn.feature_selection import VarianceThreshold

X = [[0, 2, 0, 3],
     [0, 1, 4, 3],
     [0, 1, 1, 3]]
selector = VarianceThreshold()
selector.fit(X)
selector.variances_
```

In [ ]:

```
selector.transform(X)
```

3.2 基于单变量检验进行筛选

此类方法可以更加灵活的使用统计量进行特征筛选。

3.2.1 筛选关联程度最高的特征

基于统计量进行筛选，达到将分析范围聚焦于核心特征的目的。

SelectKBest : 移除那些除了评分最高的 K 个特征之外的所有特征。
SelectPercentile : 移除除了用户指定的最高得分百分比之外的所有特征。

class sklearn.feature_selection.SelectPercentile/SelectKBest(

score_func = <function f_classif> : 用于计算评分的统计方法
    f_classif : ANOVA F-value，用于类别预测
    mutual_info_classif : 类别预测中的共同信息，非参方法，样本量要求高
    chi2 : 卡方检验

    f_regression : 回归分析中的F-value
    mutual_info_regression : 数值预测中的共同信息

percentile = 10 : 用于SelectPercentile，希望保留的特征比例
k = 10 : 用于SelectKBest，希望保留的特征数量

)

SelectPercentile/SelectKBest类的属性:

scores_ : array-like, shape=(n_features,)
pvalues_ : array-like, shape=(n_features,)

In [ ]:

```
boston.data[:2]
```

In [ ]:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression

X2 = SelectKBest(f_regression,
                 k = 2).fit_transform(boston.data, boston.target)
```

In [ ]:

```
X2[:2]
```

3.2.2 基于检验误差进行筛选

在单变量检验的基础上对特征进行筛选:

SelectFpr : 基于假阳性率（false positive rate）的大小进行筛选
SelectFdr : 基于伪发现率（false discovery rate）的大小进行筛选，伪发现率指错误拒绝H0的个数占所有被拒绝的原假设个数的比例的期望值
SelectFwe : 基于族系误差（family wise error）进行筛选，FWE即一般所说的多次检验的总体I类错误。

class sklearn.feature_selection.SelectFpr/SelectFdr/SelectFwe(

score_func = <function f_classif>
alpha = 0.05

)

sklearn.feature_selection.SelectFpr/SelectFdr/SelectFwe类的属性:

scores_ : array-like, shape=(n_features,)
pvalues_ : array-like, shape=(n_features,)

In [ ]:

```
from sklearn.feature_selection import SelectFpr
from sklearn.feature_selection import f_regression

SelFpr = SelectFpr(f_regression, alpha = 10e-10).fit(
    boston.data, boston.target)
SelFpr.pvalues_
```

In [ ]:

```
SelFpr.scores_
```

In [ ]:

```
SelFpr.transform(boston.data)[:2]
```

3.2.3 统计量用于特征筛选的通用框架

GenericUnivariateSelect允许使用可配置方法来进行单变量特征选择。

class sklearn.feature_selection.GenericUnivariateSelect(

```
    score_func = <function f_classif>
    mode = 'percentile' : 用于特征筛选的标准
        'percentile', 'k_best', 'fpr', 'fdr', 'fwe'
    param = 1e-05 : float or int, 对应上述筛选标准的阈值
    )
```

In [ ]:

```
from sklearn.feature_selection import GenericUnivariateSelect
from sklearn.feature_selection import f_regression

GenSel = GenericUnivariateSelect(f_regression,
                                 mode = 'fpr', param = 10e-10
                                ).fit(boston.data, boston.target)
GenSel.pvalues_
```

In [ ]:

```
GenSel.transform(boston.data)[:2]
```

3.3 基于建模结果进行筛选

SelectFromModel可以处理任何带有"coef"*或者*"feature__importances"属性的训练之后的评估器。 如果相关的"coef_"或者"feature__importances"属性值低于预先设置的阈值，这些特征将会被认为不重要并且移除掉。

除了指定数值上的阈值之外，还可以通过给定字符串参数来使用内置的启发式方法找到一个合适的阈值。可以使用的启发式方法有mean、median以及基于这些指标的乘法运算，如0.1*mean。

class sklearn.feature_selection.SelectFromModel(

```
    estimator : 用于评估变量重要性的估计器
        该估计器在fit后必须要有feature_importances_或者coef_属性
    threshold = None : string, float, 用于筛选变量的阈值
        float: 评分低于此阈值的变量将被剔除
        string: 'median', 'mean', '1.25*mean'等表述方式
        None : 有l1参数时为1e-5, 否则为'mean'

    prefit = False :
    norm_order = 1 :
```

)

SelectFromModel类的属性 :

```
    estimator_ :
    threshold_ :
```

In [ ]:

```
boston.data[:2]
```

In [ ]:

```
from sklearn.linear_model import LinearRegression

reg = LinearRegression()

reg.fit(boston.data, boston.target)
reg.coef_
```

In [ ]:

```
from sklearn.feature_selection import SelectFromModel

sfm = SelectFromModel(reg, threshold = 0.1)
sfm.fit(boston.data, boston.target)
```

In [ ]:

```
sfm.transform(boston.data)[:2]
```

3.4 数据降维与信息浓缩

注意：sklearn中的PCA方法默认使用协方差阵，这一点和SPSS等统计软件不同。

因此必要时应当先对数据做标化

class sklearn.decomposition.PCA(

```
n_components = None : int/float/None/string，希望保留的主成分数量
    如果为None，则所有主成分均被保留，为'mle'时自动选择最佳数量

copy = True
whiten = False : 输出的主成分是否*sqrt(n_samples)/特征根，即标准化
    该转换会损失部分方差信息，但有时候会使得后续的建模效果有所改善

svd_solver = 'auto' : {'auto', 'full', 'arpack', 'randomized'}
    auto : 根据X.shape和n_components自动选择方法
    full : 完整的SVD解法，即LAPACK
    arpack : ARPACK法，要求0 < n_components <  X的列数
    randomized : Halko等提出的随机SVD法
tol = 0.0, iterated_power = 'auto', random_state = None
```

)

sklearn.decomposition.PCA类的属性：

components_ : array, 形如(n_components, n_features), 主成分系数矩阵
explained_variance_ : array, 形如(n_components,), 各主成分解释的方差量
explained_variance_ratio_ : array, 形如(n_components,), 解释方差比例
singular_values_ : array, 形如(n_components,), 各主成分对应的奇异值
mean_ : array, 形如(n_features,), 各属性的均数, 等价于X.mean(axis=1)
n_components_ : int
noise_variance_ : float, 剩余的噪声协方差

sklearn.decomposition.PCA类的方法:

fit(X[, y])
fit_transform(X[, y])
get_covariance() : 给出模型的协方差阵
get_params([deep])
get_precision() : 给出协方差矩阵的逆矩阵 (Precision Matrix)
inverse_transform(X)
score(X[, y]) : 给出样本的平均对数似然值
score_samples(X) : 给出每个样本的对数似然值
set_params(**params)
transform(X)

In [ ]:

```
from sklearn import preprocessing

X_scaled = preprocessing.scale(iris.data)
```

In [ ]:

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
# 使用标化后的数据, 因此等价于采用相关系数阵做PCA
pca.fit(X_scaled)
```

In [ ]:

```
# 给出主成分系数矩阵
pca.components_
```

In [ ]:

```
# 各主成分的方差解释量 (特征值)
pca.explained_variance_
```

In [ ]:

```
# 换算后的各主成分方差解释比例
pca.explained_variance_ratio_
```

In [ ]:

```
Zdf = pd.DataFrame(X_scaled)
Zdf['z1'] = 0.52237162 * Zdf[0] - 0.26335492 * Zdf[1] \
            + 0.58125401 * Zdf[2] + 0.56561105 * Zdf[3]
Zdf['z2'] = 0.37231836 * Zdf[0] + 0.92555649 * Zdf[1] \
            + 0.02109478 * Zdf[2] + 0.06541577 * Zdf[3]
Zdf.describe()
```

In [ ]:

```
Zdf.head()
```

In [ ]:

```
# 各主成分相加时应当按照携带信息量的大小进行加权
Zdf['tot'] = Zdf.z1 * 1.711828 + Zdf.z2 * 0.963018
Zdf.head(10)
```

In [ ]:

```
# 计算出主成分用于后续分析
pca.transform(X_scaled)[:5]
```

3.5 实战练习

使用GenericUnivariateSelect()类，针对boston数据实现SelectKBest、SelectPercentile、SelectFpr、SelectFdr和SelectFwe类的功能。

使用PCA方法对boston数据的自变量进行降维。

4 回归类模型的训练

4.1 线性回归模型

4.1.1 模型概述

4.1.2 线性回归模型的sklean实现

class sklearn.linear_model.LinearRegression(

    fit_intercept = True : 模型是否包括常数项
        使用该选项就不需要在数据框中设定cons
    normalize = False : 是否对数据做正则化，具体为(x - mean)/L2-norm
    copy_X = True : 是否复制x矩阵
    n_jobs = 1 : 使用的例程数，为-1时使用全部CPU，大样本多因变量时有加速效果

)

注意: