In [ ]:

```
# 读入外部保存的模型文件
reg2 = joblib.load('f:/reg.pkl')
reg2.coef_
```

1.5 实战练习

加载sklearn自带的iris数据集，熟悉该数据集的各种属性，并尝试将其转换为数据框。

尝试在不参考任何帮助文档的情况下，按照sklearn中的标准API操作方式，使用BP神经网络对iris数据进行拟合，并返回各案例的预测类别、预测概率等结果。

> BP神经网络对应的类为：class sklearn.neural_network.MLPClassifier()
> 此处只为API操作演示，不进一步讨论模型拟合前的数据预处理问题

将上题中生成的模型存储为外部文件，并重新读入。

2 数据的预处理

2.1 数值变量的标准化

数据标准化可以去除均数、离散程度量纲差异太大的影响。

> 减去均数：去除均数的影响。
> 除以标准差：去除离散程度的影响。

但是标准化对离群值的影响无能为力，其结果仍然受离群值的严重影响。

2.1.1 对单个数据集进行标化

sklearn.preprocessing.scale(

> X : {array-like, sparse matrix}，需要进行变换的数据阵
> axis = 0 : 指定分别按照列(0)还是整个样本(1)计算均数、标准差并进行变换
> with_mean = True : 是否中心化数据（移除均数）
> with_std = True : 是否均一化标准差（除以标准差）
> copy = True : 是否生成副本而不是替换原数据

)

In [ ]:

```
bostondf.head()
```

In [ ]:

```
bostondf.describe()
```

In [ ]:

```
from sklearn import preprocessing

X_scaled = preprocessing.scale(bostondf)
X_scaled[:2]
```

In [ ]:

```
# 计算转换后的均数和标准差
X_scaled.mean(axis = 0), X_scaled.std(axis = 0)
```

In [ ]:

```
# 对整个矩阵统一做标准化
X_scaled1 = preprocessing.scale(bostondf, axis = 1)
X_scaled1[:2]
```

In [ ]:

```
X_scaled1.mean(axis = 0), X_scaled1.std(axis = 0)
```

In [ ]:

```
X_scaled1.mean(), X_scaled1.std()
```

In [ ]:

```
# scale函数也可以直接对单列的因变量做标准化
preprocessing.scale(boston.target)[:10]
```

2.1.2 在多个数据集上使用相同的标准化变换

使用API接口，可以将标准化变换方法设置为可调用的函数，每次直接调用即可。

class sklearn.preprocessing.StandardScaler(

```
copy = True : 是否生成副本而不是替换原对象（但这种替换不一定能成功）
with_mean = True : 该选项对稀疏矩阵无效
with_std = True :
```

)

StandardScaler类的属性：

```
scale_ : ndarray, shape (n_features,)
mean_ : array of floats with shape [n_features]
var_ : array of floats with shape [n_features]
n_samples_seen_ : int
```

StandardScaler类的方法：

```
inverse_transform(X[, copy])      将数据进行逆变换
partial_fit(X[, y])     在线计算数据特征用于后续拟合

fit(X[, y])     计算数据特征用于后续拟合
transform(X[, y, copy])      使用模型设定进行转换
fit_transform(X[, y])      计算数据特征，并且进行转换
get_params([deep])     获取模型的参数设定
set_params(**params)      设置模型参数
```

In [ ]:

```
std = preprocessing.StandardScaler()
std.fit(bostondf)
std.mean_, std.scale_
```

In [ ]:

```
std.transform(bostondf[:2])
```

2.1.3 将特征缩放至特定范围

常见的操作是将数据转换至0~1之间，也可以将每个特征的最大绝对值转换至指定数值大小。

此类方法仍然对离群值非常敏感。

此类方法的基本用途：

放大方差极小的特征，使其在特征选择时能保留在模型中。
聚类分析中常见的数据预处理方式之一。
实现特征极小方差的稳健性，或者在稀疏矩阵中保留零元素。

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

将数据缩放至指定的范围内。

class sklearn.preprocessing.MaxAbsScaler(copy = True)

将数据的最大值缩放至1。

In [ ]:

```
scaler = preprocessing.MinMaxScaler((1, 10))
scaler.fit_transform(bostondf)[:2]
```

2.1.4 数据的正则化

正则化（Normalization）/归一化/范数化是机器学习领域提出的基于向量空间模型上的一个转换，经常被使用在分类与聚类中。

sklearn.preprocessing.normalize(

```
    X, axis = 1, copy = True
    norm = 'l2' : 'l1', 'l2', or 'max', 用于正则化的具体范数
    return_norm = False : 是否返回所使用的范数
```

)

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

该类独立对待每个样本，其fit方法实际不做任何事情

In [ ]:

```
X = [[ -1., -1.,  2.]]
X_normalized = preprocessing.normalize(X, norm = 'l2', return_norm = True)

X_normalized
```

In [ ]:

```
-1/2.44948974
```

2.2 考虑异常分布的标准化方法

2.2.1 稳健标准化

将中位数和百分位数（默认使用四分位间距）分别代替均数和标准差用于数据的标准化。

更适合于已知有离群值的数据。

sklearn.preprocessing.robust_scale(

```
    X, axis = 0, with_centering = True, with_scaling = True
    quantile_range = (25.0, 75.0) : 用于计算离散程度的百分位数
    copy = True
```

)

class sklearn.preprocessing.RobustScaler(

```
    with_centering = True, with_scaling = True,
    quantile_range = (25.0, 75.0), copy = True
```

)

In [ ]:

```
rscaler = preprocessing.RobustScaler()
rs = rscaler.fit_transform(bostondf)
rs[:2]
```

In [ ]:

```
np.median(rs, axis = 0), rs.mean(axis = 0), rs.std(axis = 0)
```

2.2.2 分位数转换

当数据分布严重异常时，可以考虑对原始数据做秩变换，用秩次/百分位点代替原变量值进行后续分析。

通过损失部分信息的办法解决分布异常问题。
sklearn文档中将该方法称为非线性转换。

class sklearn.preprocessing.QuantileTransformer(

```
n_quantiles = 1000 : int, 希望计算的分位数总数
output_distribution = 'uniform' : str, 转换后指标服从的分布
    {'uniform', 'normal'}
ignore_implicit_zeros = False, subsample = 100000
random_state = None : 随机种子的数值
copy = True
```

)

In [ ]:

```
QT = preprocessing.QuantileTransformer()
qtres = QT.fit_transform(bostondf)
pd.DataFrame(qtres)[0].hist()
```

2.3 分类变量的预处理

2.3.1 二值化

特征二值化：当原始数据类似于阳性事件计数时，可以将数值特征用阈值过滤得到逻辑值

例：本月迟到次数 ==> 本月是否迟到

sklearn.preprocessing.binarize(

```
X, copy = True
threshold = 0.0 : 设定一个非负的阈值，小于等于阈值的赋值0，大于阈值则赋值1
```

)

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

该类独立对待每个样本，其fit方法实际不做任何事情

In [ ]:

```
preprocessing.binarize(bostondf, threshold = 2.5)[:2]
```

2.3.2 分类特征的重编码（哑变量化）

sklearn中的模型默认只能使用连续变量，无法识别分类特征。

class sklearn.preprocessing.OneHotEncoder(

```
    n_values = 'auto' : int or array of ints, 每个特征的取值数
        'auto' : 通过训练数据来自动确定
        int : 指定类别数，具体类别的取值必须在range(n_values)范围内
        array : n_values[i]表示在X[:, i]中的类别数
    categorical_features = 'all' : 'all' or array of indices or mask
        将哪些列作为分类进行处理，其余列将直接在结果中输出
    dtype = np.float : number type, 输出的数据类型
    sparse = True : 是否返回稀疏矩阵而不是普通矩阵
    handle_unknown = 'error' : str, 'error' or 'ignore', 有未知类别时如何处理
```

)

In [ ]:

```
enc = preprocessing.OneHotEncoder()
enc.fit([[0, 0, 3],
         [1, 1, 0],
         [0, 2, 1],
         [1, 0, 2]])
enc.transform([[0, 1, 3]]).toarray()
```

2.3.3 用pandas完成分类特征的哑变量化

pandas不仅可以完成分类变量的哑变量化，还可以很容易地完成字符串变量的数值化转换，一这点是sklearn中提供的功能所很难做到的。

pd.get_dummies(

```
    data : 希望转换的数据框/变量列
    prefix = None : 哑变量名称前缀
    prefix_sep = '_': 前缀和序号之间的连接字符，设定有prefix或列名时生效
    dummy_na = False : 是否为NaNs专门设定一个哑变量列
    columns = None : 希望转换的原始列名，如果不设定，则转换所有符合条件的列
    drop_first = False : 是否返回k-1个哑变量，而不是k个哑变量
```

) # 返回值为数据框

In [ ]:

```
bostondf.RAD.value_counts()
```

In [ ]:

```
bostondf.RAD.astype('int').astype('str').value_counts()
```

In [ ]:

```
pd.get_dummies(bostondf.RAD.astype('int').astype('str'),
               prefix = 'RAD').head()
```

2.4 处理缺失值与交互项

2.4.1 缺失值的填充

sklearn中的绝大部分模型都不能自动处理缺失值（树模型除外），必须进行预处理。

方法一：删除缺失值所在的整行或整列，代价是信息丢失
方法二：使用所在行/列中的平均值、中位数或者众数来填充缺失值，代价是引入偏差

class sklearn.preprocessing.Imputer(

```
missing_values = 'NaN' : integer or 'NaN', 缺失值在数据中的表示方法
strategy = 'mean' : 填充方法, 'mean', 'median', 'most_frequent'
axis = 0 : integer, 按照列(0)还是行(1)进行填充
verbose = 0 : integer, 控制填充的冗余度
copy = True :
```

)

Imputer类的属性：

```
statistics_ : array of shape (n_features,), 按列填充时各属性的填充值
```

In [ ]:

```
from sklearn.preprocessing import Imputer

imp = Imputer()
imp.fit([[1, 2],
         [np.nan, 3],
         [7, 6]])
imp.transform([[np.nan, 2],
               [6, np.nan]])
```

In [ ]:

```
imp.statistics_
```

2.4.2 生成多项式特征（交互项）

一般而言，建模时往往可以找到确实对模型改善有贡献的二次交互项，但三次以上的交互项则很少需要考虑。

三阶及以上的交互作用很难得到专业知识的支持。
不仅对模型改善贡献很小，还有可能会造成过拟合。

class sklearn.preprocessing.PolynomialFeatures(

```
degree = 2 : 希望计算的多项式的阶数
interaction_only  = False : 是否只计算交互项，而不包括原变量的任何高次项
include_bias = True : 是否加入一个偏移量/常数项
```

)

In [ ]:

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 3, interaction_only = True)
polyres = poly.fit_transform(bostondf.iloc[:,[0,1,2,3]])
```

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)