sklearn是做特征工程（做模型调算法）最常用也是最好用的工具没有之一，本文为相关内容的总结，分为如下几个部分：

- 什么是特征工程?
- 数据预处理
- 特征选择
- 降维

什么是特征工程

数据和特征决定机器学习的上限，模型和算法只是逼近这个上限。

特征工程本质是一项工程活动，目的是最大限度地从原始数据中提取特征以供算法和模型使用。

特征工程主要分为三部分：

1. **数据预处理** 对应的sklearn包：sklearn-Processing data (http://scikit-learn.org/stable/modules/preprocessing.html#non-linear-transformation)
2. **特征选择** 对应的sklearn包： sklearn-Feature selection (http://scikit-learn.org/stable/modules/feature_selection.html)
3. **降维** 对应的sklearn包： sklearn-Dimensionality reduction (http://scikit-learn.org/stable/modules/decomposition.html#decompositions)

本文中使用sklearn中的IRIS（鸢尾花）数据集来对特征处理功能进行说明，首先导入IRIS数据集的代码如下：

In [1]:
```
from sklearn.datasets import load_iris

# 导入IRIS数据集
iris = load_iris()

# 特征矩阵，目标向量
iris.data, iris.target
```

```
       [5.9, 3. , 4.2, 1.5],
       [6. , 2.2, 4. , 1. ],
       [6.1, 2.9, 4.7, 1.4],
       [5.6, 2.9, 3.6, 1.3],
       [6.7, 3.1, 4.4, 1.4],
       [5.6, 3. , 4.5, 1.5],
       [5.8, 2.7, 4.1, 1. ],
       [6.2, 2.2, 4.5, 1.5],
       [5.6, 2.5, 3.9, 1.1],
       [5.9, 3.2, 4.8, 1.8],
       [6.1, 2.8, 4. , 1.3],
       [6.3, 2.5, 4.9, 1.5],
       [6.1, 2.8, 4.7, 1.2],
       [6.4, 2.9, 4.3, 1.3],
       [6.6, 3. , 4.4, 1.4],
       [6.8, 2.8, 4.8, 1.4],
       [6.7, 3. , 5. , 1.7],
       [6. , 2.9, 4.5, 1.5],
       [5.7, 2.6, 3.5, 1. ],
       [5.5, 2.4, 3.8, 1.1],
```

数据预处理

通过特征提取，可以得到未经处理的特征，此时存在一些问题：

- **不属于同一量纲**：特征规格不同，无法比较。——**无量纲化**
- **信息冗余**：对某些定量特征，其包含有效信息为区间划分（例如学习成绩，若只关心“及格”或“不及格”，则需要将定量的考分转化为1和0表示）——**二值化**
- **定性特征无法直接使用**：通常使用独热编码方式将定性特征转换为定量特征（假设N种定性值，则将这一个特征扩展为N种特征，当原始特征值为第i种定性值时，第i个扩展特征赋值为1，其他扩展特征赋值为0），独热编码相比直接指定方式，不用增加调参工作，对于线性模型，使用**独热编码**后的特征可以达到非线性的效果。
- **缺失值**：填充缺失值
- **信息利用率低**：不同机器学习算法和模型对数据中信息的利用是不同的（线性模型中，使用对定性特征独热编码可以达到非线性的效果），对定量变量多项式化，，或者进行其他**数据变换**，也可以达到非线性效果。

使用sklearn中的preprocessing库进行数据预处理

无量纲化

将不同量纲的数据转换到同一规格

标准化（Z-score standardization）—— 对列向量处理

将服从正态分布的特征值转换成标准正态分布，标准化需要计算特征的均值和标准差，公式如下：$x' = \frac{x-\overline{X}}{S}$

使用preprocessing库的StandardScaler类对数据进行标准化

In [2]:
```
from sklearn.preprocessing import StandardScaler

# 标准化，返回值为标准化后的数据
StandardScaler().fit_transform(iris.data)
```

```
       -1.31544430e+00],
       [-9.00681170e-01,  5.58610819e-01, -1.16971425e+00,
        -9.20547742e-01],
       [-1.26418478e+00,  7.88807586e-01, -1.05603939e+00,
        -1.31544430e+00],
       [-1.02184904e+00, -1.31979479e-01, -1.22655167e+00,
        -1.31544430e+00],
       [-1.02184904e+00,  7.88807586e-01, -1.22655167e+00,
        -1.05217993e+00],
       [-7.79513300e-01,  1.01900435e+00, -1.28338910e+00,
        -1.31544430e+00],
       [-7.79513300e-01,  7.88807586e-01, -1.34022653e+00,
        -1.31544430e+00],
       [-1.38535265e+00,  3.28414053e-01, -1.22655167e+00,
        -1.31544430e+00],
       [-1.26418478e+00,  9.82172869e-02, -1.22655167e+00,
        -1.31544430e+00],
       [-5.37177559e-01,  7.88807586e-01, -1.28338910e+00,
        -1.05217993e+00],
```

区间缩放——对列向量处理

区间缩放有很多思路，常见为利用两个最值进行缩放，公式如下：$x' = \frac{x-Min}{Max-Min}$

使用preprocessing库的MinMaxScaler类对数据进行区间缩放

In [3]:
```
from sklearn.preprocessing import MinMaxScaler

# 区间缩放，返回值为缩放到[0, 1]区间的数据
MinMaxScaler().fit_transform(iris.data)
```

```
[0.80111111, 0.33333333, 0.86440678, 0.75       ],
[1.        , 0.75      , 0.91525424, 0.79166667],
[0.58333333, 0.33333333, 0.77966102, 0.875      ],
[0.55555556, 0.33333333, 0.69491525, 0.58333333],
[0.5       , 0.25      , 0.77966102, 0.54166667],
[0.94444444, 0.41666667, 0.86440678, 0.91666667],
[0.55555556, 0.58333333, 0.77966102, 0.95833333],
[0.58333333, 0.45833333, 0.76271186, 0.70833333],
[0.47222222, 0.41666667, 0.6440678 , 0.70833333],
[0.72222222, 0.45833333, 0.74576271, 0.83333333],
[0.66666667, 0.45833333, 0.77966102, 0.95833333],
[0.72222222, 0.45833333, 0.69491525, 0.91666667],
[0.41666667, 0.29166667, 0.69491525, 0.75      ],
[0.69444444, 0.5       , 0.83050847, 0.91666667],
[0.66666667, 0.54166667, 0.79661017, 1.        ],
[0.66666667, 0.41666667, 0.71186441, 0.91666667],
[0.55555556, 0.20833333, 0.6779661 , 0.75      ],
[0.61111111, 0.41666667, 0.71186441, 0.79166667],
[0.52777778, 0.58333333, 0.74576271, 0.91666667],
[0.44444444, 0.41666667, 0.69491525, 0.70833333]])
```

何时用标准化，何时用区间缩放

- 后续分类、聚类算法中，需要使用距离来度量相似性的时候、或者使用PCA、LDA这种需要用到协方差分析进行降维的时候，同时数据分布可以近似为正态分布——**标准化**
- 不涉及距离度量，协方差计算，数据不符合正态分布的时候，可以使用**区间缩放法，或其他归一化。**（比如图像处理中，将RGB图像转换为灰度图像后将其值限定在[0, 255]的范围）

归一化 —— 对行向量处理

目的在于样本向量在点乘运算或其他核函数计算相似性时，拥有统一的标准，即都转化为“单位向量”。

公式如下：$x' = \frac{x}{\sqrt{\sum_j^m x[j]^2}}$

使用preprocessing库的Normalizer类对数据进行归一化：

In [4]:
```
from sklearn.preprocessing import Normalizer

# 归一化，返回值为归一化后的数据
Normalizer().fit_transform(iris.data)
```

```
[0.72766159, 0.27553141, 0.59982915, 0.18685205],
[0.71578999, 0.34430405, 0.5798805 , 0.18121266],
[0.69417747, 0.30370264, 0.60740528, 0.2386235 ],
[0.72366005, 0.32162669, 0.58582004, 0.17230001],
[0.69385414, 0.29574111, 0.63698085, 0.15924521],
[0.73154399, 0.28501714, 0.57953485, 0.21851314],
[0.67017484, 0.36168166, 0.59571097, 0.2553047 ],
[0.69804799, 0.338117  , 0.59988499, 0.196326  ],
[0.71066905, 0.35533453, 0.56853524, 0.21320072],
[0.72415258, 0.32534391, 0.56672811, 0.22039426],
[0.69997037, 0.32386689, 0.58504986, 0.25073566],
[0.73337886, 0.32948905, 0.54206264, 0.24445962],
[0.69052512, 0.32145135, 0.60718588, 0.22620651],
[0.69193502, 0.32561648, 0.60035539, 0.23403685],
[0.68914871, 0.33943145, 0.58629069, 0.25714504],
[0.72155725, 0.32308533, 0.56001458, 0.24769876],
[0.72965359, 0.28954508, 0.57909015, 0.22005426],
[0.71653899, 0.3307103 , 0.57323119, 0.22047353],
[0.67467072, 0.36998072, 0.58761643, 0.25028107],
[0.69025916, 0.35097923, 0.5966647 , 0.21058754]])
```

对定量特征二值化——对列向量处理

定性与定量

- 定性：他很胖，她很瘦
- 定量：他100kg，她50kg
- 定性都有相关描述词，定量的描述用数字进行量化

定量二值化的核心在于设定一个阈值，大于阈值为1，小于阈值为0，公式如下：

$$x' = \begin{cases} 1, & x > threshold \\ 0, & x \leq threshold \end{cases}$$

使用preprocessing库的Binarizer类对数据进行二值化：

In [5]:
```
from sklearn.preprocessing import Binarizer

# 归一化，返回值为归一化后的数据
Binarizer().fit_transform(iris.data)
```

```
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

对定性特征独热编码——对列向量处理

有些特征用文字分类表达，或将这些类转化为数字，但数字与数字之间没有大小关系，纯粹的分类标记，此时需要独热编码对其进行编码。

IRIS数据集的特征都是定量特征，使用其目标值进行独热编码。

使用preprocessing库的OneHotEncoder类对数据进行独热编码：

In [6]:
```
from sklearn.preprocessing import OneHotEncoder

# 独热编码，对IRIS数据集的目标值，返回值为独热编码后的数据
OneHotEncoder().fit_transform(iris.target.reshape((-1,1)))
```

Out[6]:
```
<150x3 sparse matrix of type '<class 'numpy.float64'>'
        with 150 stored elements in Compressed Sparse Row format>
```

缺失值计算——对列向量处理

由于IRIS数据集没有缺失值，因此对数据集增加一个样本，4个特征值均赋值为Nan。

使用preprocessing库的SimpleImputer类对数据进行缺失值计算：

In [7]:
```
from numpy import vstack, array, nan
from sklearn.impute import SimpleImputer

# 缺失值计算，返回值为计算缺失值后的数据
# 参数missing_value为缺失值的表示形式，默认为NaN
# 参数strategy为缺失值填充方式，默认为mean（均值）
SimpleImputer().fit_transform(vstack((array([nan, nan, nan, nan]), iris.data)))
```

```
       [7.4       , 2.8       , 6.1       , 1.9       ],
       [7.9       , 3.8       , 6.4       , 2.        ],
       [6.4       , 2.8       , 5.6       , 2.2       ],
       [6.3       , 2.8       , 5.1       , 1.5       ],
       [6.1       , 2.6       , 5.6       , 1.4       ],
       [7.7       , 3.        , 6.1       , 2.3       ],
       [6.3       , 3.4       , 5.6       , 2.4       ],
       [6.4       , 3.1       , 5.5       , 1.8       ],
       [6.        , 3.        , 4.8       , 1.8       ],
       [6.9       , 3.1       , 5.4       , 2.1       ],
       [6.7       , 3.1       , 5.6       , 2.4       ],
       [6.9       , 3.1       , 5.1       , 2.3       ],
       [5.8       , 2.7       , 5.1       , 1.9       ],
       [6.8       , 3.2       , 5.9       , 2.3       ],
       [6.7       , 3.3       , 5.7       , 2.5       ],
       [6.7       , 3.        , 5.2       , 2.3       ],
       [6.3       , 2.5       , 5.        , 1.9       ],
       [6.5       , 3.        , 5.2       , 2.        ],
       [6.2       , 3.4       , 5.4       , 2.3       ],
       [5.9       , 3.        , 5.1       , 1.8       ]])
```

数据变换

多项式变换——对行向量处理

常见的数据变换有基于多项式的，基于指数函数、基于对数函数的。

4个特征，度为2的多项式转换公式如下：

$$(x'_1, x'_2, x'_3, x'_4, x'_5, x'_6, x'_7, x'_8, x'_9, x'_{10}, x'_{11}, x'_{12}, x'_{13}, x'_{14}, x'_{15})$$
$$= (1, x_1, x_2, x_3, x_4, x_1^2, x_1x_2, x_1x_3, x_1x_4, x_2^2, x_2x_3, x_2x_4, x_3^2, x_3x_4, x_4^2)$$

使用preprocessing库的PolynormialFeatures类对数据进行多项式转换：

In [8]:
```
from sklearn.preprocessing import PolynomialFeatures

# 多项式转换
# 参数degree为度，默认值为2
PolynomialFeatures().fit_transform(iris.data)
```

Out[8]:
```
array([[ 1.  ,  5.1 ,  3.5 , ...,  1.96,  0.28,  0.04],
       [ 1.  ,  4.9 ,  3.  , ...,  1.96,  0.28,  0.04],
       [ 1.  ,  4.7 ,  3.2 , ...,  1.69,  0.26,  0.04],
       ...,
       [ 1.  ,  6.5 ,  3.  , ..., 27.04, 10.4 ,  4.  ],
       [ 1.  ,  6.2 ,  3.4 , ..., 29.16, 12.42,  5.29],
       [ 1.  ,  5.9 ,  3.  , ..., 26.01,  9.18,  3.24]])
```

自定义变换

基于单变元函数的数据变换可以使用一个统一的方式完成。

使用preprocessing库的FunctionTransformer对数据进行对数函数转换：

In [9]:
```
from numpy import log1p
from sklearn.preprocessing import FunctionTransformer

# 自定义转换函数为对数函数的数据变换
# 第一个参数是单变元函数
FunctionTransformer(log1p).fit_transform(iris.data)
```

```
array([[2.00148   , 1.43508453, 1.84054963, 1.19392247],
       [2.01490302, 1.38629436, 1.87180218, 1.02961942],
       [2.16332303, 1.56861592, 2.04122033, 1.16315081],
       [2.16332303, 1.28093385, 2.06686276, 1.19392247],
       [1.94591015, 1.16315081, 1.79175947, 0.91629073],
       [2.06686276, 1.43508453, 1.90210753, 1.19392247],
       [1.88706965, 1.33500107, 1.77495235, 1.09861229],
       [2.16332303, 1.33500107, 2.04122033, 1.09861229],
       [1.98787435, 1.30833282, 1.77495235, 1.02961942],
       [2.04122033, 1.45861502, 1.90210753, 1.13140211],
       [2.10413415, 1.43508453, 1.94591015, 1.02961942],
       [1.97408103, 1.33500107, 1.75785792, 1.02961942],
       [1.96009478, 1.38629436, 1.77495235, 1.02961942],
       [2.00148   , 1.33500107, 1.88706965, 1.13140211],
       [2.10413415, 1.38629436, 1.91692261, 0.95551145],
       [2.12823171, 1.33500107, 1.96009478, 1.06471074],
       [2.18605128, 1.56861592, 2.00148   , 1.09861229],
       [2.00148   , 1.33500107, 1.88706965, 1.16315081],
       [1.98787435, 1.33500107, 1.80828877, 0.91629073],
       [1.96009478, 1.28093385, 1.88706965, 0.87546874],
```

总结

- StandardScaler：**无量纲化**，标准化，基于特征矩阵的列，将特征值转换至服从标准正态分布
- MinMaxScaler：**无量纲化**，区间缩放，基于最大最小值，将特征值转换到[0,1]区间
- Normalizer：**归一化**，基于特征矩阵的行，将样本向量转换为“单位向量”
- Binarizer：**二值化**，基于给定阈值，将定量特征按阈值划分
- OneHotEncoder：**独热编码**，将定性数据编码为定量数据
- SimpleImputer：**缺失值计算**，计算缺失值，缺失值可填充为均值等
- PolynomialFeatures：**多项式数据转换**
- FunctionTransformer：**自定义单元数据时转换**，使用单变元的函数来转换数据

特征选择

数据预处理完成后，需要选择有意义的特征输入机器学习的算法和模型进行训练。

通常从两个方面考虑来选择特征：

- 特征是否发散：如果一个特征不发散（例如方差趋近于0），则样本在这个特征上基本没有差异，因此该特征对样本的区别并没有什么用。
- 特征与目标相关性：优先选择目标相关性高的特征。

特征选择形式可以将特征选择方法分为三种：

- 过滤法（Filter）：不考虑后续学习器，按照发散性或者相关性对各个特征进行评分，设定阈值或者待选择阈值的个数，选择特征。
- 包装法（Wrapper）：考虑后续学习器，根据目标函数（通常是预测效果评分），每次选择若干特征，或者排除若干特征。
- 嵌入法（Embedded）：Filter与Wrapper方法结合，根据机器学习的算法和模型进行训练，得到各个特征的权值系数，根据系数从大到小选择特征。

使用sklearn中的feature_selection库进行特征选择

过滤法（Filter）

方差选择法

计算各个特征的方差，根据阈值，选择方差大于阈值的特征。

使用feature_selection库的VarianceThreshold类来选择特征：

In [10]:
```
from sklearn.feature_selection import VarianceThreshold

# 方差选择法，返回值为特征选择后的数据
# 参数threshold为方差的阈值
VarianceThreshold(threshold=3).fit_transform(iris.data)
```

Out[10]:
```
array([[1.4],
       [1.4],
       [1.3],
       [1.5],
       [1.4],
       [1.7],
       [1.4],
       [1.5],
       [1.4],
       [1.5],
       [1.5],
       [1.6],
       [1.4],
       [1.1],
       [1.2],
       [1.5],
       [1.3],
       [1.4],
       [1.7],
       [1.5]
```

卡方检验

检验特征对标签的相关性，选择其中K个与标签最相关的特征。

使用feature_selection库的SelectKBest类结合卡方检验来选择特征：

In [11]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

# 选择K个最好的特征，返回选择特征后的数据
SelectKBest(chi2, k=2).fit_transform(iris.data, iris.target)
```

```
       [1.5, 0.2],
       [1.3, 0.3],
       [1.3, 0.3],
       [1.3, 0.2],
       [1.6, 0.6],
       [1.9, 0.4],
       [1.4, 0.3],
       [1.6, 0.2],
       [1.4, 0.2],
       [1.5, 0.2],
       [1.4, 0.2],
       [4.7, 1.4],
       [4.5, 1.5],
       [4.9, 1.5],
       [4. , 1.3],
       [4.6, 1.5],
       [4.5, 1.3],
       [4.7, 1.6],
       [3.3, 1. ],
       [4.6, 1.3],
```

包装法（Wrapper）

递归特征消除法

通过基模型进行多轮训练，每轮训练后消除若干权值系数特征，再基于新的特征集进行下轮训练。

使用feature_selection库的RFE类来选择特征：

In [12]:
```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

# 递归特征消除法，返回特征选择后的数据
# 参数estimator为基模型
# 参数n_features_to_select为选择的特征个数
RFE(estimator=LogisticRegression(), n_features_to_select=2).fit_transform(iris.data, iris.target)
```

```
       [4.9, 1.5],
       [4. , 1.3],
       [4.6, 1.5],
       [4.5, 1.3],
       [4.7, 1.6],
       [3.3, 1. ],
       [4.6, 1.3],
       [3.9, 1.4],
       [3.5, 1. ],
       [4.2, 1.5],
       [4. , 1. ],
       [4.7, 1.4],
       [3.6, 1.3],
       [4.4, 1.4],
       [4.5, 1.5],
       [4.1, 1. ],
       [4.5, 1.5],
       [3.9, 1.1],
       [4.8, 1.8],
       [4.  , 1.2],
```

嵌入法（Embedded）

基于惩罚项的特征选择法

筛选出特征的同时进行降维。

使用feature_selection库的SelectFromModel类结合L2惩罚项的逻辑回归模型来选择特征：

In [13]:
```
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression

# 带L1惩罚项的逻辑回归作为基模型的特征选择
SelectFromModel(LogisticRegression(penalty="12", C=0.1)).fit_transform(iris.data, iris.target)
```

```
       [6.1, 1.9],
       [6.4, 2. ],
       [5.6, 2.2],
       [5.1, 1.5],
       [5.6, 1.4],
       [6.1, 2.3],
       [5.6, 2.4],
       [5.5, 1.8],
       [4.8, 1.8],
       [5.4, 2.1],
       [5.6, 2.4],
       [5.1, 2.3],
       [5.1, 1.9],
       [5.9, 2.3],
       [5.7, 2.5],
       [5.2, 2.3],
       [5. , 1.9],
       [5.2, 2. ],
       [5.4, 2.3],
       [5.1, 1.8]])
```

基于树模型的特征选择法

树模型中GBDT可作为基模型进行特征选择。

使用feature_selection库的SelectFromModel类结合GBDT模型来选择特征：

In [14]:
```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import GradientBoostingClassifier

# GBDT作为基模型的特征选择
SelectFromModel(GradientBoostingClassifier()).fit_transform(iris.data, iris.target)
```

Out[14]:
```
array([[1.4, 0.2],
       [1.4, 0.2],
       [1.3, 0.2],
       [1.5, 0.2],
       [1.4, 0.2],
       [1.7, 0.4],
       [1.4, 0.3],
       [1.5, 0.2],
       [1.4, 0.2],
       [1.5, 0.1],
       [1.5, 0.2],
       [1.6, 0.2],
       [1.4, 0.1],
       [1.1, 0.1],
       [1.2, 0.2],
       [1.5, 0.4],
       [1.3, 0.4],
       [1.4, 0.3],
       [1.7, 0.3],
```

总结

- VarianceThreshold：**过滤法（Filter）**，方差选择法

- SelectKBest：**过滤法（Filter）**，可选关联系数、卡方检验、最大信息系数作为得分计算的方法
- RFE：**包装法（Wrapper）**，递归训练基模型，将权值系数较小的特征从特征集合中消除
- SelectFromModel：**嵌入法（Embedded）**，训练基模型，选择权值系数较高的特征

降维

特征选择完成后训练模型，但特征矩阵过大，计算量大训练时间长，需要降低特征矩阵维度。

降维方法：

- 基于L1惩罚项的模型
- 主成分分析法（PCA）
- 线性判别分析（LDA）

PCA与LDA有很多相似，其本质是将原始样本映射到维度更低的样本空间，但二者映射目标不一样：

- PCA：让映射后的样本具有最大的发散性（无监督）
- LDA：让映射后的样本有最好的分类性能（有监督）

主成分分析法（PCA）

使用decomposition库的PCA类选择特征：

In [15]:
```
from sklearn.decomposition import PCA

# 主成分分析法，返回降维后的数据
# 参数n_components为主成分数目
PCA(n_components=2).fit_transform(iris.data)
```

```
       [ 2.84167278,  0.37526917],
       [ 3.23067366,  1.37416509],
       [ 2.15943764, -0.21727758],
       [ 1.44416124, -0.14341341],
       [ 1.78129481, -0.49990168],
       [ 3.07649993,  0.68808568],
       [ 2.14424331,  0.1400642 ],
       [ 1.90509815,  0.04930053],
       [ 1.16932634, -0.16499026],
       [ 2.10761114,  0.37228787],
       [ 2.31415471,  0.18365128],
       [ 1.9222678 ,  0.40920347],
       [ 1.41523588, -0.57491635],
       [ 2.56301338,  0.2778626 ],
       [ 2.41874618,  0.3047982 ],
       [ 1.94410979,  0.1875323 ],
       [ 1.52716661, -0.37531698],
       [ 1.76434572,  0.07885885],
       [ 1.90094161,  0.11662796],
       [ 1.39018886, -0.28266094]])
```

线性判别分析法（LDA）

使用discriminant_analysis库的LDA类选择特征：

In [16]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

# 线性判别分析法，返回降维后的数据
# 参数n_components为降维后的维数
LDA(n_components=2).fit_transform(iris.data, iris.target)
```

```
[ 6.36055159e+00,  1.01516362e+00],
[ 7.34305989e+00, -9.47319209e-01],
[ 8.39738652e+00,  6.47363392e-01],
[ 7.21929685e+00, -1.09646389e-01],
[ 7.32679599e+00, -1.07298943e+00],
[ 7.57247066e+00, -8.05464137e-01],
[ 9.84984300e+00,  1.58593698e+00],
[ 9.15823890e+00,  2.73759647e+00],
[ 8.58243141e+00,  1.83448945e+00],
[ 7.78075375e+00,  5.84339407e-01],
[ 8.07835876e+00,  9.68580703e-01],
[ 8.02097451e+00,  1.14050366e+00],
[ 7.49680227e+00, -1.88377220e-01],
[ 7.58648117e+00,  1.20797032e+00],
[ 8.68104293e+00,  8.77590154e-01],
[ 6.25140358e+00,  4.39696367e-01],
[ 6.55893336e+00, -3.89222752e-01],
[ 6.77138315e+00, -9.70634453e-01],
[ 6.82308032e+00,  4.63011612e-01],
[ 7.92461638e+00,  2.09638715e-01],
```

总结

- decomposition：PCA，主成分分析法
- discriminant_analysis：LDA，线性判别分析法

所有代码实现

In [17]:
```
# encoding=utf-8
'''
用sklearn做特征工程，分为三部分：
1.数据预处理
2.特征选择
3.降维
'''

import pandas as pd
import numpy as np
from numpy import vstack, array, nan
from sklearn.datasets import load_iris

from sklearn import preprocessing
from sklearn import feature_selection
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

if __name__ == '__main__':

    # 导入IRIS数据集
    iris = load_iris()
    features = iris.data
    labels = iris.target

    '''
    1.数据预处理
    '''

    # 1.1 无量纲化：将不同规格的数据转换到同一规格
    # 1.1.1 标准化：将服从正态分布的特征值转换成标准正态分布（对列向量处理）
    # print(np.mean(features, axis=0))
    # print(np.std(features, axis=0))
    features_new = preprocessing.StandardScaler().fit_transform(features)
    # print(np.mean(features_new, axis=0))
    # print(np.std(features_new, axis=0))
    # 1.1.2 区间缩放：将特征值缩放到[0, 1]区间的数据（对列向量处理）
    features_new = preprocessing.MinMaxScaler().fit_transform(features)
    # 1.1.3 归一化：将行向量转化为“单位向量”（对每个样本处理）
    features_new = preprocessing.Normalizer().fit_transform(features)

    # 1.2 对定量特征二值化:设定一个阈值，大于阈值的赋值为1，小于等于阈值的赋值为0
    features_new = preprocessing.Binarizer(threshold=3).fit_transform(features)

    # 1.3 对定性（分类）特征编码(也可用pandas.get_dummies函数)
    enc = preprocessing.OneHotEncoder()
    enc.fit([[0, 0, 3],
             [1, 1, 0],
             [0, 2, 1],
             [1, 0, 2]])
    # print(enc.transform([[0, 1, 3]]))
    # print(enc.transform([[0, 1, 3]]).toarray())

    # 1.4 缺失值计算(也可用pandas.fillna函数)
    features_new = SimpleImputer().fit_transform(vstack((array([nan, nan, nan, nan]), features)))

    # 1.5 数据变换
    # 1.5.1 基于多项式变换（对行变量处理）
    features_new = preprocessing.PolynomialFeatures().fit_transform(features)
    # 1.5.2 基于自定义函数变换，以log函数为例
    features_new = preprocessing.FunctionTransformer(np.log1p).fit_transform(features)
```

```
'''
2.特征选择
'''
# 2.1 Filter
# 2.1.1 方差选择法，选择方差大于阈值的特征
features_new = feature_selection.VarianceThreshold(threshold=0.3).fit_transform(features)
# 2.1.2 卡方检验,选择K个与标签最相关的特征
features_new = feature_selection.SelectKBest(feature_selection.chi2, k=3).fit_transform(featur

# 2.2 Wrapper
# 2.2.1 递归特征消除法，这里选择逻辑回归作为基模型，n_features_to_select为选择的特征个数
features_new = feature_selection.RFE(estimator=LogisticRegression(), n_features_to_select=2).f

# 2.3 Embedded
# 2.3.1 基于惩罚项的特征选择法,这里选择带L1惩罚项的逻辑回归作为基模型
features_new = feature_selection.SelectFromModel(LogisticRegression(penalty="l2", C=0.1)).fit_
# 2.3.2 基于树模型的特征选择法,这里选择GBDT模型作为基模型
features_new = feature_selection.SelectFromModel(GradientBoostingClassifier()).fit_transform(f

'''
3.降维
'''
# 3.1 主成分分析法（PCA）,参数n_components为降维后的维数
features_new = PCA(n_components=2).fit_transform(features)

# 3.2 线性判别分析法（LDA）,参数n_components为降维后的维数
features_new = LDA(n_components=2).fit_transform(features, labels)
```

```
C:\Users\Huris\anaconda3\envs\sEMG\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stabl
e/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scik
it-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```