人工智能之机器学习

特征工程（Feature Engineering）

上海育创网络科技股份有限公司

主讲人：赵翌臣

缺省值填充

- 对于缺省的数据，在处理之前一定需要进行预处理操作，一般采用中位数、均值或者众数来进行填充，主要通过Imputer类来实现对缺省值的填充

class sklearn.preprocessing. **Imputer**(*missing_values='NaN', strategy='mean', axis=0, verbose=0, copy=True*) ¶ [source]

Notes

- When `axis=0`, columns which only contained missing values at fit are discarded upon transform.
- When `axis=1`, an exception is raised if there are rows for which it is not possible to fill in the missing values (e.g., because they only contain missing values).

Methods

`fit` (X[, y])	Fit the imputer on X.
`fit_transform` (X[, y])	Fit to data, then transform it.
`get_params` ([deep])	Get parameters for this estimator.
`set_params` (**params)	Set the parameters of this estimator.
`transform` (X)	Impute all missing values in X.

缺省值填充

```python
import numpy as np
from sklearn.preprocessing import Imputer
```

```python
X = [
    [2, 2, 4, 1],
    [np.nan, 3, 4, 4],
    [1, 1, 1, np.nan],
    [2, 2, np.nan, 3]
]
X2 = [
    [2, 6, np.nan, 1],
    [np.nan, 5, np.nan, 1],
    [4, 1, np.nan, 5],
    [np.nan, np.nan, np.nan, 1]
]
```

```python
imp1 = Imputer(missing_values='NaN', strategy='mean', axis=0)
imp2 = Imputer(missing_values='NaN', strategy='mean', axis=1)
imp1.fit(X)
imp2.fit(X)
```

```python
print imp1.transform(X2)
print "------------------------------"
print imp2.transform(X2)
```

```
[[ 2.          6.          3.          1.        ]
 [ 1.66666667  5.          3.          1.        ]
 [ 4.          1.          3.          5.        ]
 [ 1.66666667  2.          3.          1.        ]]
------------------------------
[[ 2.  6.  4.  1.]
 [ 2.  5.  4.  1.]
 [ 4.  1.  4.  5.]
 [ 2.  2.  4.  1.]]
```

```python
imp1 = Imputer(missing_values='NaN', strategy='mean', axis=0)
imp2 = Imputer(missing_values='NaN', strategy='median', axis=0)
imp3 = Imputer(missing_values='NaN', strategy='most_frequent', axis=0)
imp1.fit(X)
imp2.fit(X)
imp3.fit(X)
```

```python
print X2
print "------------------------------"
print imp1.transform(X2)
print "------------------------------"
print imp2.transform(X2)
print "------------------------------"
print imp3.transform(X2)
```

```
[[2, 6, nan, 1], [nan, 5, nan, 1], [4, 1, nan, 5], [nan, nan, nan, 1]]
------------------------------
[[ 2.          6.          3.          1.        ]
 [ 1.66666667  5.          3.          1.        ]
 [ 4.          1.          3.          5.        ]
 [ 1.66666667  2.          3.          1.        ]]
------------------------------
[[ 2.  6.  4.  1.]
 [ 2.  5.  4.  1.]
 [ 4.  1.  4.  5.]
 [ 2.  2.  4.  1.]]
------------------------------
[[ 2.  6.  4.  1.]
 [ 2.  5.  4.  1.]
 [ 4.  1.  4.  5.]
 [ 2.  2.  4.  1.]]
```

二值化

- 二值化(Binarizer)：对于定量的数据根据给定的阈值，将其进行转换，如果大于阈值，那么赋值为1；否则赋值为0

class sklearn.preprocessing. **Binarizer** (*threshold=0.0, copy=True*) ¶ [source]

```python
import numpy as np
from sklearn.preprocessing import Binarizer
```

```python
arr = np.array([
    [1.5, 1.3, 1.9],
    [0.5, 0.5, 1.6],
    [1.1, 2.1, 0.2]
])
```

```python
binarizer = Binarizer(threshold=1.0).fit(arr)
binarizer
```

```
Binarizer(copy=True, threshold=1.0)
```

```python
binarizer.transform(arr)
```

```
array([[ 1.,  1.,  1.],
       [ 0.,  0.,  1.],
       [ 1.,  1.,  0.]])
```

区间缩放法

- 区间缩放法：是指按照数据的方差特性对数据进行缩放操作，将数据缩放到给定区间上，常用的计算方式如下:

$$X\_std = \frac{X - X.\min}{X.\max - X.\min} \qquad X\_scaled = X\_std * (\max - \min) + \min$$

class sklearn.preprocessing. **MinMaxScaler** (*feature_range=(0, 1), copy=True*) [source]

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

```
X = np.array([
    [1, -1, 2, 3],
    [2, 0, 0, 3],
    [0, 1, -1, 3]
], dtype=np.float64)
```

```
scaler = MinMaxScaler(feature_range=(1,5))
scaler.fit(X)
```

```
MinMaxScaler(copy=True, feature_range=(1, 5))
```

```
print scaler.data_max_
print scaler.data_min_
print scaler.data_range_
```

```
[ 2.  1.  2.  3.]
[ 0. -1. -1.  3.]
[ 2.  2.  3.  0.]
```

```
print scaler.transform(X)
```

```
[[ 3.          1.          5.          1.        ]
 [ 5.          3.          2.33333333  1.        ]
 [ 1.          5.          1.          1.        ]]
```

规范化

- 规范化：将矩阵的行均转换为“单位向量”，l2规则转换公式如下：

$$x' = \frac{x}{\sqrt{\sum_{j=1}^{m} x(j)^2}}$$

class sklearn.preprocessing. Normalizer (*norm='l2'*, *copy=True*) [source]

```python
import numpy as np
from sklearn.preprocessing import Normalizer
```

```python
X = np.array([
    [1, -1, 2],
    [2, 0, 0],
    [0, 1, -1]
], dtype=np.float64)
```

```python
normalizer1 = Normalizer(norm='l1')
normalizer2 = Normalizer(norm='l2')
normalizer1.fit(X)
normalizer2.fit(X)
```

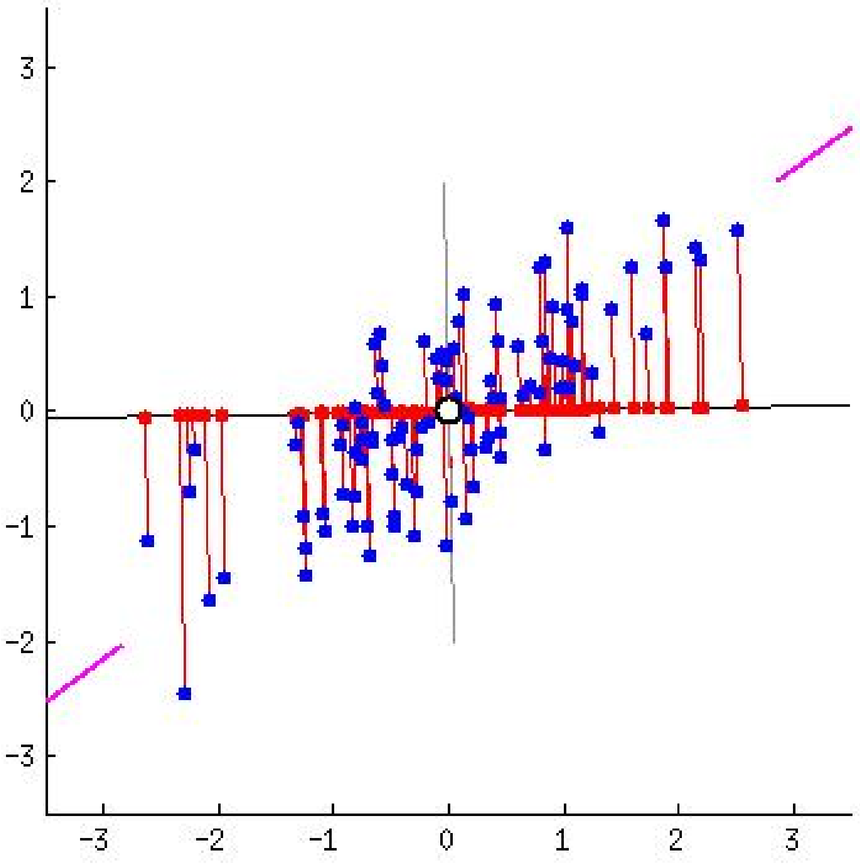
```
Normalizer(copy=True, norm='l2')
```

```python
print normalizer1.transform(X)
print "——————————————————————"
print normalizer2.transform(X)
```

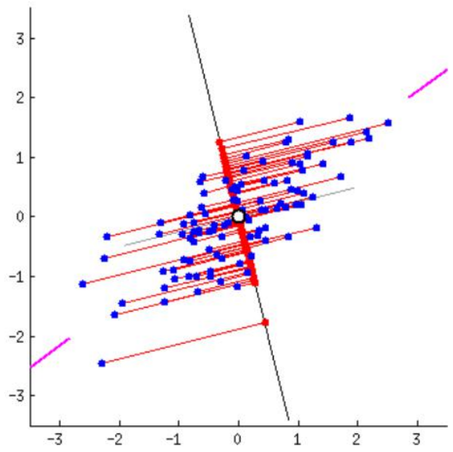
```
[[ 0.25 -0.25  0.5 ]
 [ 1.    0.    0.  ]
 [ 0.    0.5  -0.5 ]]
——————————————————————
[[ 0.40824829 -0.40824829  0.81649658]
 [ 1.          0.          0.        ]
 [ 0.          0.70710678 -0.70710678]]
```

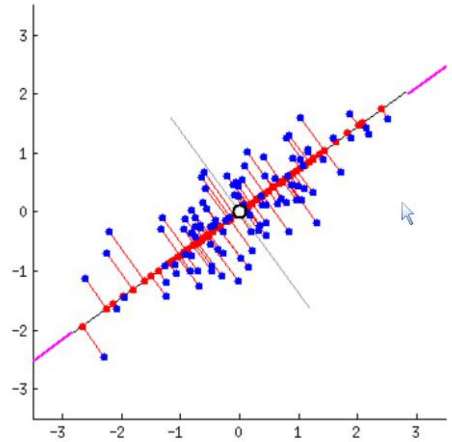
Principal Component Analysis直观解释

Principal Component Analysis直观解释



PCA主成分分析





- 介绍
  - PCA是一种无监督学习的降维技术
- 思想
  - 投影后样本越分散，保留的信息越多
- 做法
  - 将所有的样本点向直线w投影
  - 目标函数：让 投影后样本的方差 极大

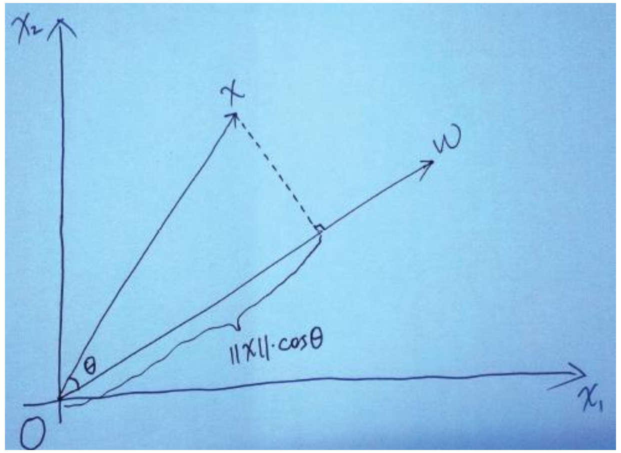
PCA主成分分析

- 样本点x_i在直线w上的投影为$w^T x_i$，这是一个实数。
- 我们的目的是让投影后的点的方差越大越好

$$\max_w \sum_i [w^T(x_i - \bar{x})]^2$$

- 若样本进行了中心化，即$\bar{x} = \vec{0}$，那么目标函数变为

$$\max_w \sum_i (w^T x_i)^2$$



- 若投影到d条线上，则希望方差和最大。先研究一个样本：

$$W = \begin{bmatrix} w_1^{(1)} & w_2^{(1)} \\ w_1^{(2)} & w_2^{(2)} \end{bmatrix}$$

$$W^T = \begin{bmatrix} w_1^{(1)} & w_1^{(2)} \\ w_2^{(1)} & w_2^{(2)} \end{bmatrix}$$

$$\underset{d\times n}{W^T}\ \underset{n\times 1}{x_i}\ \underset{1\times n}{x_i^T}\ \underset{n\times d}{W} = \begin{bmatrix} \# & \# \\ \# & \# \end{bmatrix}\begin{bmatrix} \# \\ \# \end{bmatrix}\begin{bmatrix} \# & \# \end{bmatrix}\begin{bmatrix} \# & \# \\ \# & \# \end{bmatrix} = \begin{bmatrix} \# & \# \\ \# & \# \end{bmatrix}_{d\times d}$$

$$\underset{d\times n}{W^T}\ \underset{n\times 1}{x_i} = \begin{bmatrix} \text{线1上的投影} \\ \text{线2上的投影} \end{bmatrix},\quad \underset{1\times n}{x_i^T}\ \underset{n\times d}{W} = \begin{bmatrix} \text{线1上的投影} & \text{线2上的投影} \end{bmatrix}$$

PCA主成分分析

- 扩展到所有样本

$$\sum_i \underset{d\times n}{W^T} \underset{n\times 1}{x_i} \underset{1\times n}{x_i^T} \underset{n\times d}{W} = \begin{bmatrix} \# & \# \\ \# & \# \end{bmatrix}_{d\times d} = \underset{d\times n}{W^T} \underset{n\times N}{X} \underset{N\times n}{X^T} \underset{n\times d}{W}$$

- 矩阵的迹就是方差和，对其最大化

$$\max_w \quad tr(W^T XX^T W)$$

$$s.t. \quad W^T W = I$$

$$W^T \underset{n\times d}{W} = \begin{bmatrix} w_1^{(1)} & w_1^{(2)} \\ w_2^{(1)} & w_2^{(2)} \end{bmatrix} \begin{bmatrix} w_1^{(1)} & w_2^{(1)} \\ w_1^{(2)} & w_2^{(2)} \end{bmatrix} = \begin{bmatrix} w_1^{(1)} * w_1^{(1)} + w_1^{(2)} * w_1^{(2)} & w_1^{(1)} * w_2^{(1)} + w_1^{(2)} * w_2^{(2)} \\ w_2^{(1)} * w_1^{(1)} + w_2^{(2)} * w_1^{(2)} & w_2^{(1)} * w_2^{(1)} + w_2^{(2)} * w_2^{(2)} \end{bmatrix}$$

PCA主成分分析

- 利用拉格朗日函数，求导令=0，得到

$$XX^TW = \lambda W_{n\times d}$$

- 因此W就是XX^T的特征向量组成的矩阵，而λ为XX^T的若干特征值组成的矩阵，特征值在主对角线上，其余位置为0。
- 特征值是来描述对应特征向量方向上包含多少信息量的，值越大信息量（方差）越大。因此对特征值排序取前d'个特征值对应的特征向量构成$W=(w_1,w_2,\ldots,w_{d'})$
- 对于原始数据集，我们只需要用$z=W^Tx$，就可以把原始数据集降维到d'维

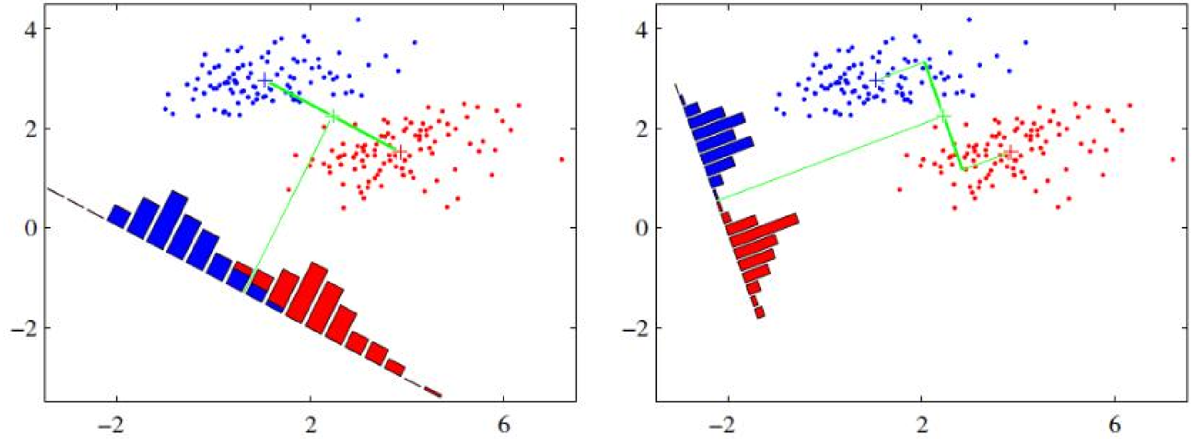
特征选取/降维-PCA

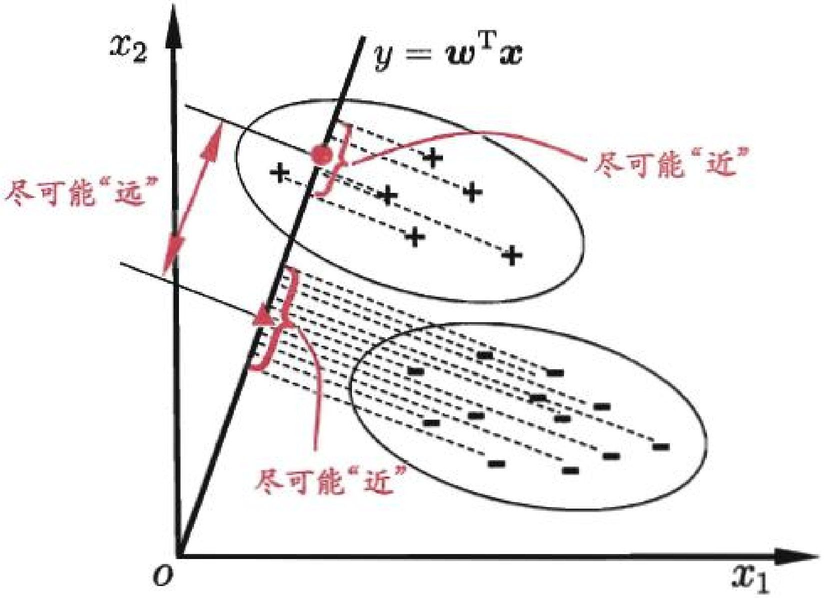
```python
import pandas as pd
from sklearn import datasets
from sklearn.decomposition import PCA

iris = datasets.load_iris()
data = pd.DataFrame(iris.data, columns=iris.feature_names)
data['class'] = iris.target
X = data[data.columns.drop('class')]
Y = data['class']

model = PCA(n_components=2)
model.fit(X, Y)
model.transform(X)
```

```
array([[-2.68420713,  0.32660731],
       [-2.71539062, -0.16955685],
       [-2.88981954, -0.13734561],
       [-2.7464372 , -0.31112432],
       [-2.72859298,  0.33392456],
       [-2.27989736,  0.74778271]
```

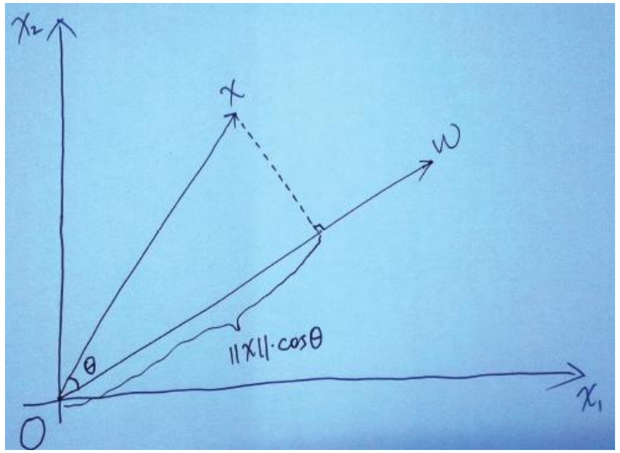
Linear Discriminant Analysis直观理解

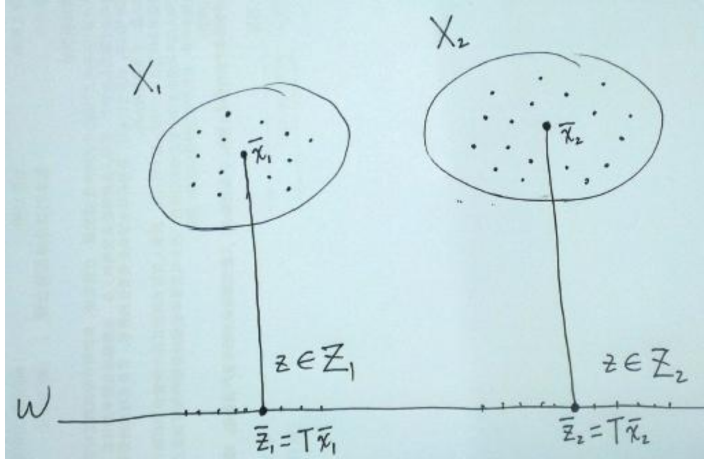
- 介绍
  - LDA是一种监督学习的降维技术
- 思想
  - 投影后类内方差最小，类间方差最大
- 做法
  - 将每个类别的中心向直线w投影，获取投影点
  - 计算直线上，每个类别下样本的方差
  - 目标函数：让 均值的投影点间的距离 / 各类别组内方差的和 极大





LDA数学原理





- 线性变换T为如下形式：

$$y = T(x) = w \cdot x,\quad w \in R^d \text{为待定向量}$$

- 再引入集合

$$Z = \{z_1, z_2, \cdots, z_N\},\quad Z_1 = \{z_j \mid x_j \in X_1\},\quad Z_2 = \{z_j \mid x_j \in X_2\}$$

其中$z_i = w \cdot x,\quad i = 1,2,\cdots,N$

- 计算中心点

记$\bar{x}_1, \bar{x}_2$分别表示样本数据集合X_1, X_2的中心点，即

$$\bar{x}_i = \frac{1}{N_i}\sum_{x \in X_i} x,\quad i = 1,2 \quad \Rightarrow \quad \bar{z}_i = T(\bar{x}_i),\quad i = 1,2$$

LDA数学原理

- 1.$\bar{z}_1$与$\bar{z}_2$离得越远越好
- 2.Z_i中的元素越集中在 $\bar{z}_i$ 附近越好

- 针对第1条可以通过类间离散度(Between-class scatter)量化

$$J_B = \left|\bar{z}_1 - \bar{z}_2\right|^2$$

- 针对第2条可以通过类内离散度(Within-class scatter)量化

$$J_W = s_1^2 + s_2^2,\quad \text{其中} s_i^2 = \sum_{z \in Z_i} (z - \bar{z}_i)^2,\quad i = 1,2$$

- 目标函数

$$J(w) = \frac{J_B}{J_W}$$

LDA数学原理

北风网 IBEIFENG.COM

- 改写目标函数

$$
\begin{aligned}
J_B &= |\bar{z}_1 - \bar{z}_2|^2 \\
&= (\mathbf{w}^T\bar{\mathbf{x}}_1 - \mathbf{w}^T\bar{\mathbf{x}}_2)^2 \\
&= (\mathbf{w}^T(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2))^2 \\
&= \mathbf{w}^T(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)\mathbf{w}^T(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2) \\
&= \mathbf{w}^T(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)^T\mathbf{w}
\end{aligned}
$$

$$S_B = (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)^T,$$

$$J_B = \mathbf{w}^T S_B \mathbf{w}$$

S_B是类间离散度矩阵

$$J_W = s_1^2 + s_2^2$$

$$s_i^2 = \sum_{\mathbf{x}\in X_i} \mathbf{w}^T\left((\mathbf{x} - \bar{\mathbf{x}}_i)(\mathbf{x} - \bar{\mathbf{x}}_i)^T\right)\mathbf{w}$$

$$= \mathbf{w}^T\left(\sum_{\mathbf{x}\in X_i}(\mathbf{x} - \bar{\mathbf{x}}_i)(\mathbf{x} - \bar{\mathbf{x}}_i)^T\right)\mathbf{w}$$

$$S_i = \sum_{\mathbf{x}\in X_i}(\mathbf{x} - \bar{\mathbf{x}}_i)(\mathbf{x} - \bar{\mathbf{x}}_i)^T$$

$$
\begin{aligned}
J_W &= s_1^2 + s_2^2 = \mathbf{w}^T S_1 \mathbf{w} + \mathbf{w}^T S_2 \mathbf{w} \\
&= \mathbf{w}^T(S_1 + S_2)\mathbf{w} = \mathbf{w}^T S_W \mathbf{w}
\end{aligned}
$$

S_W是类间离散度矩阵

LDA

```python
import pandas as pd
from sklearn import datasets
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

iris = datasets.load_iris()
data = pd.DataFrame(iris.data, columns=iris.feature_names)
data['class'] = iris.target
X = data[data.columns.drop('class')]
Y = data['class']

lda = LinearDiscriminantAnalysis(n_components=2)
lda.fit(X, Y)
lda.transform(X)
```

```
array([[-8.0849532 ,  0.32845422],
       [-7.1471629 , -0.75547326],
       [-7.51137789, -0.23807832],
       [-6.83767561, -0.64288476],
       [-8.15781367,  0.54063935],
       [-7.72363087,  1.48232345],
```

特征工程方法汇总

- 异常值检测
- 特征缩放
- 特征扩展
- 离散特征处理
- 缺失值处理
- 类别不平衡处理
- TF-IDF
- Word2vec
- 降维：主成分分析PCA、线性判别分析LDA