

人工智能之机器学习

数据清洗和特征工程

主讲人：李老師

学习目标

- 了解特征工程在机器学习当中的重要性
- 特征预处理
- 特征提取
- 特征选择和特征的降维

特征工程做一个总结

- 所有一切为了**让模型效果变的更好**的数据处理方式都可以认为属于特征工程这个范畴中的一个操作；
- 至于需求做不做这个特征工程，需要我们在开发过程中不但的进行尝试。
- 常规的特征工程需要处理的内容：
 - 异常数据的处理
 - 数据不平衡处理
 - 文本处理：词袋法、TF-IDF
 - 多项式扩展、哑编码、标准化、归一化、区间缩放法、PCA、特征选择.....
 - 将均值、方差、协方差等信息作为特征属性，对特征属性进行对数转换、指数转换.....
 - 结合业务衍生出一些新的特征属性....

特征工程介绍

- 为什么需要特征工程(Feature Engineering)
 - 机器学习领域的大神Andrew Ng(吴恩达)老师说 “Coming up with features is difficult, time-consuming, requires expert knowledge. ‘Applied machine learning’ is basically feature engineering.”
 - 业界广泛流传：数据和特征决定了机器学习的上限，而模型和算法只是逼近这个上限而已。
- 什么是特征工程
 - 特征工程是使用专业背景知识和技巧处理数据，使得特征能在机器学习算法上发挥更好的作用的过程。
 - 意义：会直接影响机器学习的效果

特征预处理

- 数据清洗
- 归一化和标准化

数据清洗

- 数据清洗(data cleaning)是在机器学习过程中一个不可缺少的环节，其数据的清洗结果直接关系到模型效果以及最终的结论。在实际的工作中，数据清洗通常占开发过程的30%-50%左右的时间。

数据清洗--预处理

- 在数据预处理过程主要考虑两个方面，如下：
 - 选择数据处理工具：关系型数据库或者Python
 - 查看数据的元数据以及数据特征：一是查看元数据，包括字段解释、数据来源等一切可以描述数据的信息；另外是抽取一部分数据，通过人工查看的方式，对数据本身做一个比较直观的了解，并且初步发现一些问题，为之后的数据处理做准备。

数据清洗--格式内容错误数据清洗

- 一般情况下，数据是由用户/访客产生的，也就有很大的可能性存在格式和内容上不一致的情况，所以在进行模型构建之前需要先进行数据的格式内容清洗操作。格式内容问题主要有以下几类：
 - 时间、日期、数值、半全角等显示格式不一致：直接将数据转换为一类格式即可，该问题一般出现在多个数据源整合的情况下。
 - 内容中有不该存在的字符：最典型的就是在头部、中间、尾部的空格等问题，这种情况下，需要以半自动校验加半人工方式来找出问题，并去除不需要的字符。
 - 内容与该字段应有的内容不符：比如姓名写成了性别、身份证号写成手机号等问题。

数据清洗--逻辑错误清洗

- 主要是通过简单的逻辑推理发现数据中的问题数据，防止分析结果走偏，主要包含以下几个步骤：
 - 数据去重
 - 去除/替换不合理的值
 - 去除/重构不可靠的字段值(修改矛盾的内容)

数据清洗--去除不需要的数据

- 一般情况下，我们会尽可能多的收集数据，但是不是所有的字段数据都是可以应用到模型构建过程的，也不是说将所有的字段属性都放到构建模型中，最终模型的效果就一定会好，实际上来讲，字段属性越多，模型的构建就会越慢，所以有时候可以考虑将不要的字段进行删除操作。在进行该过程的时候，要注意备份原始数据。

数据清洗--关联性验证

- 如果数据有多个来源，那么有必要进行关联性验证，该过程常应用到多数据源合并的过程中，通过验证数据之间的关联性来选择比较正确的特征属性，比如：汽车的线下购买信息和电话客服问卷信息，两者之间可以通过姓名和手机号进行关联操作，匹配两者之间的车辆信息是否是同一辆，如果不是，那么就需要进行数据调整。

数据清洗——案例：数据缺省值填充

- python科学计算库
- `from sklearn.preprocessing import Imputer`
 - 均值填充
 - 中值填充
 - 众数填充

归一化和标准化


- 为什么我们要进行归一化/标准化？
 - 特征的单位或者大小相差较大，或者某特征的方差相比其他的特征要大出几个数量级，容易影响（支配）目标结果，使得一些算法无法学习到其它的特征
 - 我们需要用到一些方法进行无量纲化，使不同规格的数据转换到同一规格

归一化

- 通过对原始数据进行变换把数据映射到(默认为[0,1])之间

$$X' = \frac{x - \min}{\max - \min} \quad X'' = X' * (mx - mi) + mi$$

特征1	特征2	特征3	特征4
90	2	10	40
60	4	15	45
75	3	13	46

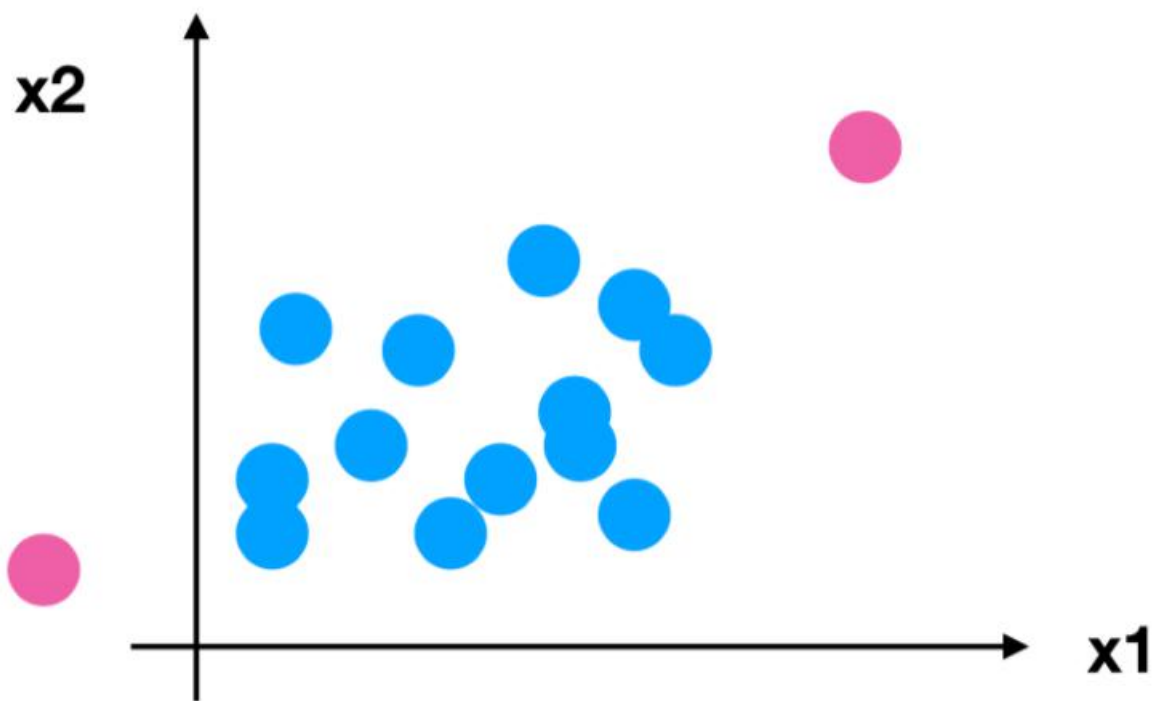


特征1	特征2	特征3	特征4
$\frac{90 - 60}{90 - 60}$	$\frac{2 - 2}{4 - 2}$	$\frac{10 - 10}{15 - 10}$	$\frac{40 - 40}{46 - 40}$
$\frac{60 - 60}{90 - 60}$	$\frac{4 - 2}{4 - 2}$	$\frac{15 - 10}{15 - 10}$	$\frac{45 - 40}{46 - 40}$
$\frac{75 - 60}{90 - 60}$	$\frac{3 - 2}{4 - 2}$	$\frac{13 - 10}{15 - 10}$	$\frac{46 - 40}{46 - 40}$

注：里面是第一步，还需要第二步乘以(1-0)+0

归一化

- 思考：如果数据中异常点较多，会有什么影响？



注意最大值最小值是变化的，另外，最大值与最小值非常容易受异常点影响，所以这种方法鲁棒性较差，只适合传统精确小数据场景。

标准化

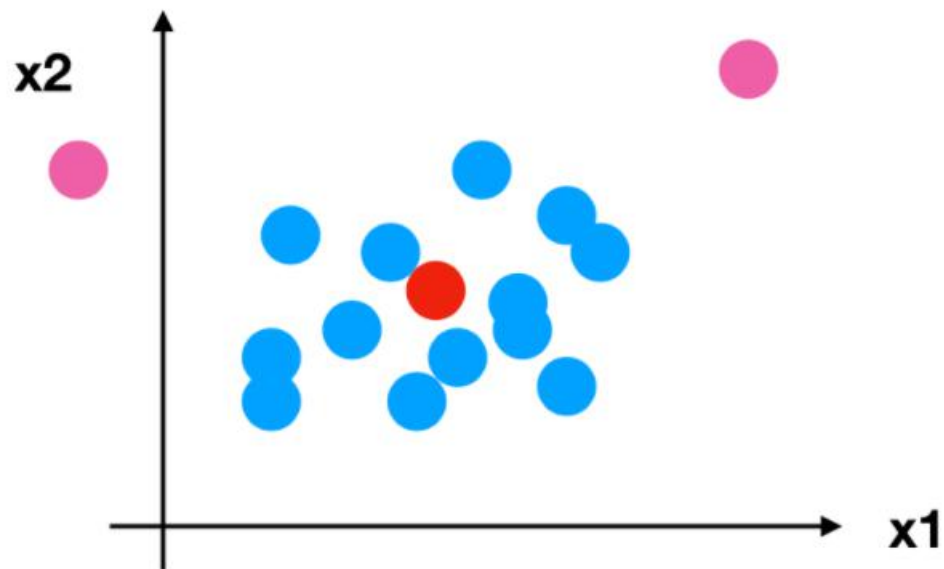
- 通过对原始数据进行变换把数据变换到均值为0,标准差为1范围内

$$X' = \frac{x - \text{mean}}{\sigma}$$

思考：能解决归一化的问题吗？

标准化

- 回到刚才异常点的地方，我们再来看看标准化



对于归一化来说：如果出现异常点，影响了最大值和最小值，那么结果显然会发生改变

对于标准化来说：如果出现异常点，由于具有一定数据量，少量的异常点对于平均值的影响并不大，从而方差改变较小。

- 在已有样本足够多的情况下比较稳定，适合现代嘈杂大数据场景。

特征提取

- 将任意数据（如文本或图像）转换为可用于机器学习的数字特征
 - 字典特征提取(特征离散化)
 - 文本特征提取
 - 图像特征提取【一般图片本身就是个数组数据】

字典特征提取(特征离散化)

- 对字典数据进行特征值化

```
In [1]: from sklearn.feature_extraction import DictVectorizer
```

```
In [2]: data = [{'city': '北京', 'temperature': 100}, {'city': '上海', 'temperature': 60}, {'city': '深圳', 'temperature': 30}]
```

```
In [3]: # 1. 实例化一个转换器类
transfer = DictVectorizer(sparse=False)
# 2. 调用fit_transform
data = transfer.fit_transform(data)
print("返回的结果:\n", data)
# 打印特征名字
print("特征名字: \n", transfer.get_feature_names())
```

返回的结果:

```
[[ 0.  1.  0. 100.]
 [ 1.  0.  0.  60.]
 [ 0.  0.  1.  30.]]
```

特征名字:

```
['city=上海', 'city=北京', 'city=深圳', 'temperature']
```

One-Hot

```
In [4]: enc = OneHotEncoder()
a=np.array([[0, 0, 3], [2, 1, 0], [0, 2, 1], [1, 0, 2], [1,1,1]])
enc.fit(a)
print(a)
print("编码结果",enc.n_values_)
print("原来的特征(进行哑编码的)在新的特征矩阵中的位置映射关系:")
print(enc.feature_indices_)
enc.transform([[1, 2, 2],[2, 2, 2]]).toarray()
```

```
[[0 0 3]
 [2 1 0]
 [0 2 1]
 [1 0 2]
 [1 1 1]]
```

编码结果 [3 3 4]

原来的特征(进行哑编码的)在新的特征矩阵中的位置映射关系:

```
[ 0  3  6 10]
```

```
Out[4]: array([[ 0.,  1.,  0.,  0.,  0.,  1.,  0.,  0.,  1.,  0.],
               [ 0.,  0.,  1.,  0.,  0.,  1.,  0.,  0.,  1.,  0.]])
```

```
[7]: import pandas as pd

a = pd.DataFrame([
    ['a', 1, 2],
    ['b', 1, 1],
    ['a', 2, 1],
    ['c', 1, 2],
    ['c', 1, 2]
], columns=['c1', 'c2', 'c3'])
a = pd.get_dummies(a)
a
```

Out[7]:

	c2	c3	c1_a	c1_b	c1_c
0	1	2	1	0	0
1	1	1	0	1	0
2	2	1	1	0	0
3	1	2	0	0	1
4	1	2	0	0	1

文本特征提取

- 中文jieba分词处理
 - `pip3 install jieba`
- 词袋法
- TF-IDF

特征降维

- 降维是指在某些限定条件下，降低随机变量(特征)个数，得到一组“不相关”主变量的过程
 - 正是因为在进行训练的时候，我们都是使用特征进行学习。如果特征本身存在问题或者特征之间相关性较强，对于算法学习预测会影响较大
- 降维的两种方式
 - 特征选择
 - 主成分分析（可以理解一种特征提取的方式）

什么是特征选择

- 数据中包含冗余或无关变量（或称特征、属性、指标等），旨在从原有特征中找出主要特征。



特征？

- 1、羽毛颜色
- 2、眼睛宽度
- 3、眼睛长度
- 4、爪子长度
- 5、体格大小

特征选择

- 特征选择的方法主要有以下三种：
 - Filter: 过滤法, 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, 从而选择特征; 常用方法包括方差选择法、相关系数法、卡方检验、互信息法等。
 - Wrapper: 包装法, 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; 常用方法主要是递归特征消除法。
 - Embedded: 嵌入法, 先使用某些机器学习的算法和模型进行训练, 得到各个特征的权重系数, 根据系数从大到小选择特征; 常用方法主要是基于惩罚项的特征选择法。

特征选择-方差选择法

- 方差选择法：先计算各个特征属性的方差值，然后根据阈值，获取方差大于阈值的特征。

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

[\[source\]](#)

```
import numpy as np
from sklearn.feature_selection import VarianceThreshold
```

```
X = np.array([
    [0, 2, 0, 3],
    [0, 1, 4, 3],
    [0.1, 1, 1, 3]
], dtype=np.float32)
Y = np.array([1, 2, 1, 2])
```

```
variance = VarianceThreshold(threshold=0.1)
print(variance)
variance.fit(X)
print('-----')
print(variance.transform(X))
```

```
VarianceThreshold(threshold=0.1)
```

```
-----
[[2. 0.]
 [1. 4.]
 [1. 1.]]
```

特征选择-相关系数法

- 相关系数法：先计算各个特征属性对于目标值的相关系数以及阈值K，然后获取K个相关系数最大的特征属性。(备注：根据目标属性y的类别选择不同的方式)

```
class sklearn.feature_selection. SelectKBest (score_func=<function f_classif>, k=10) ¶ [source]
```

```
import numpy as np
from sklearn.feature_selection import VarianceThreshold, SelectKBest
from sklearn.feature_selection import f_regression
```

```
X = np.array([
    [0, 2, 0, 3],
    [0, 1, 4, 3],
    [0.1, 1, 1, 3]
])
Y = np.array([1, 2, 1])
```

```
: skl = SelectKBest(f_regression, k=2)
skl.fit(X, Y)
print(skl)
print('-----')
print(skl.scores_)
print('-----')
print(skl.transform(X))
```

```
SelectKBest(k=2, score_func=<function f_regression at 0x0000002A8FCF7D90>)
-----
[ 0.33333333  0.33333333 16.33333333          nan]
-----
[[2. 0.]
 [1. 4.]
 [1. 1.]]
```

特征选择-卡方检验

- 卡方检验：检查定性自变量对定性因变量的相关性。 $\chi^2 = \sum \frac{(A-E)^2}{E}$

```
class sklearn.feature_selection. SelectKBest (score_func=<function f_classif>, k=10) ¶ [source]
```

```
import numpy as np
from sklearn.feature_selection import VarianceThreshold, SelectKBest
from sklearn.feature_selection import f_regression
from sklearn.feature_selection import chi2
```

```
X = np.array([
    [0, 2, 0, 3],
    [0, 1, 4, 3],
    [0.1, 1, 1, 3]
])
Y = np.array([1, 2, 1])
```

```
sk2 = SelectKBest(chi2, k=2)
sk2.fit(X, Y)
print(sk2)
print(sk2.scores_)
print(sk2.transform(X))
```

```
SelectKBest(k=2, score_func=<function chi2 at 0x0000002A8FCF7D08>)
[0.05  0.125 4.9   0.   ]
[[2.  0.]
 [1.  4.]
 [1.  1.]]
```

特征选择-递归特征消除法

- 递归特征消除法：使用一个基模型来进行多轮训练，每轮训练后，消除若干权值系数的特征，再基于新的特征集进行下一轮训练。

```
class sklearn.feature_selection. RFE (estimator, n_features_to_select=None, step=1,
estimator_params=None, verbose=0) ¶
```

[\[source\]](#)

```
from sklearn.feature_selection import VarianceThreshold, SelectKBest
from sklearn.feature_selection import f_regression
from sklearn.feature_selection import chi2
from sklearn.feature_selection import RFE
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression
```

```
X = np.array([
    [0, 2, 0, 3],
    [0, 1, 4, 3],
    [0.1, 1, 1, 3]
], dtype=np.float32)
Y = np.array([1, 2, 1])
```

Wrapper-递归特征消除法

```
4]: # 基于特征消去法做的特征选择
estimator = LogisticRegression()
selector = RFE(estimator, 2, step=1)
selector = selector.fit(X, Y)
print(selector.support_)
print(selector.n_features_)
print(selector.ranking_)
print(selector.transform(X))

[False False  True  True]
2
[3 2 1 1]
[[ 0.  3.]
 [ 4.  3.]
 [ 1.  3.]]
```

特征选择 嵌入法-基于惩罚项的特征选择法

- 使用基于惩罚项的基模型，进行特征选择操作。

```
class sklearn.feature_selection. SelectFromModel (estimator, threshold=None, pfit=False) ¶  
[source]
```

```
import numpy as np  
from sklearn.feature_selection import VarianceThreshold, SelectKBest  
from sklearn.feature_selection import f_regression  
from sklearn.feature_selection import chi2  
from sklearn.feature_selection import RFE  
from sklearn.feature_selection import SelectFromModel  
from sklearn.linear_model import LogisticRegression  
from sklearn.svm import SVR
```

```
X2 = np.array([  
    [ 5.1,  3.5,  1.4,  0.2],  
    [ 4.9,  3. ,  1.4,  0.2],  
    [-6.2,  0.4,  5.4,  2.3],  
    [-5.9,  0. ,  5.1,  1.8]  
, dtype=np.float64)  
Y2 = np.array([0, 0, 2, 2])  
estimator = LogisticRegression(penalty='l1', C=0.1)  
sfm = SelectFromModel(estimator)  
sfm.fit(X2, Y2)  
print(sfm.transform(X2))
```

```
[[ 5.1]  
 [ 4.9]  
 [-6.2]  
 [-5.9]]
```

特征降维

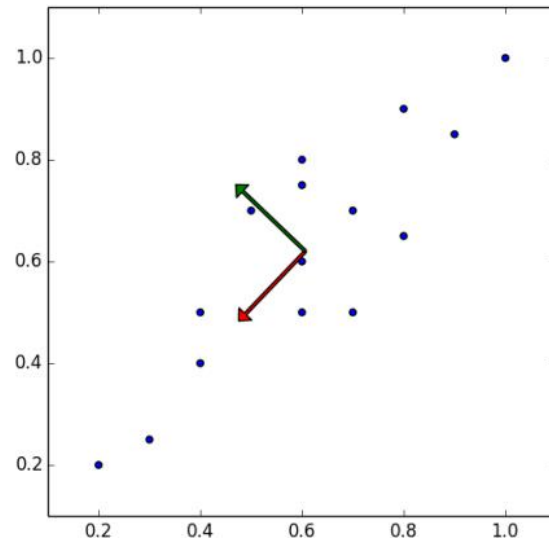
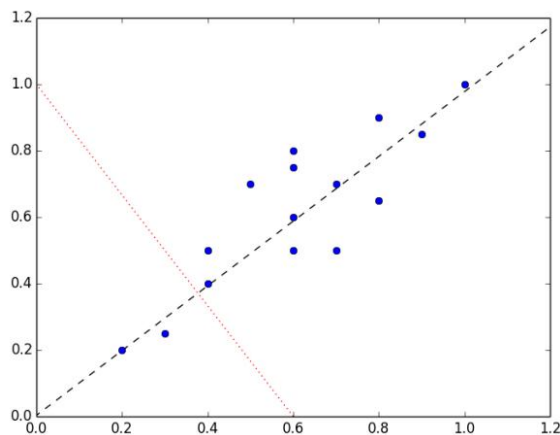
- 当特征选择完成后，可以直接可以进行训练模型了，但是可能由于特征矩阵过大，导致计算量比较大，训练时间长的问题，因此降低特征矩阵维度也是必不可少的。常见的降维方法除了基于L1的惩罚模型外，还有主成分分析法(PCA)和线性判别分析法(LDA)，这两种方法的本质都是将原始数据映射到维度更低的样本空间中；但是采用的方式不同，**PCA是为了让映射后的样本具有更大的发散性，PCA是无监督的学习算法，LDA是为了让映射后的样本有最好的分类性能，LDA是有监督学习算法。**

PCA原理

- PCA(Principal Component Analysis)是常用的线性降维方法，是一种无监督的降维算法。算法目标是通过某种线性投影，将高维的数据映射到低维的空间中表示，并且**期望在所投影的维度上数据的方差最大（最大方差理论）**，以此使用较少的数据维度，同时保留较多的原数据点的特性。
- 通俗来讲的话，如果将所有点映射到一起，那么维度一定降低下去了，但是同时也会将几乎所有的信息(包括点点之间的距离等)都丢失了，而如果映射之后的数据具有比较大的方差，那么可以认为数据点则会比较分散，这样的话，就可以保留更多的信息。从而我们可以看到PCA是一种丢失原始数据信息最少的无监督线性降维方式。

PCA原理

- 在PCA降维中，数据从原来的坐标系转换为新的坐标系，新坐标系的选择由数据本身的特性决定。第一个坐标轴选择原始数据中方差最大的方向，从统计角度来讲，这个方向是最重要的方向；第二个坐标轴选择和第一个坐标轴垂直或者正交的方向；第三个坐标轴选择和第一个、第二个坐标轴都垂直或者正交的方向；该过程一直重复，直到新坐标系的维度和原始坐标系维度数目一致的时候结束计算。而这些方向所表示的数据特征就被称为“主成分”。



特征选取/降维-PCA

- 主成分分析(PCA): 将高维的特征向量合并称为低维度的特征属性, 是一种无监督的降维方法。

```
class sklearn.decomposition. PCA (n_components=None, copy=True, whiten=False)
```

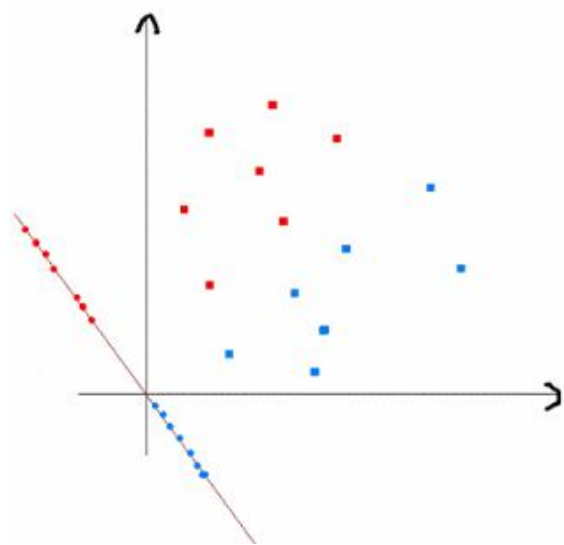
[\[source\]](#)

```
from sklearn.decomposition import PCA
X2 = np.array([
    [ 5.1,  3.5,  1.4,  0.2, 1, 23],
    [ 4.9,  3. ,  1.4,  0.2, 2, 3, 2, 1],
    [-6.2,  0.4,  5.4,  2.3, 2, 23],
    [-5.9,  0. ,  5.1,  1.8, 2, 3]
], dtype=np.float64)
pca = PCA(n_components=5)
pca.fit(X2)
print(pca.mean_)
print(pca.components_)
print(pca.transform(X2))
```

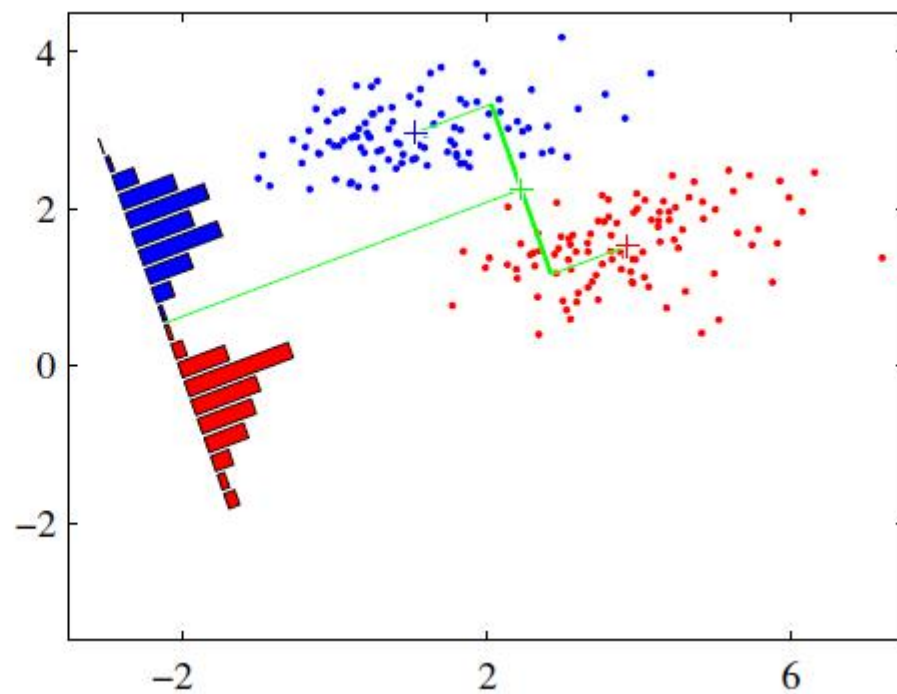
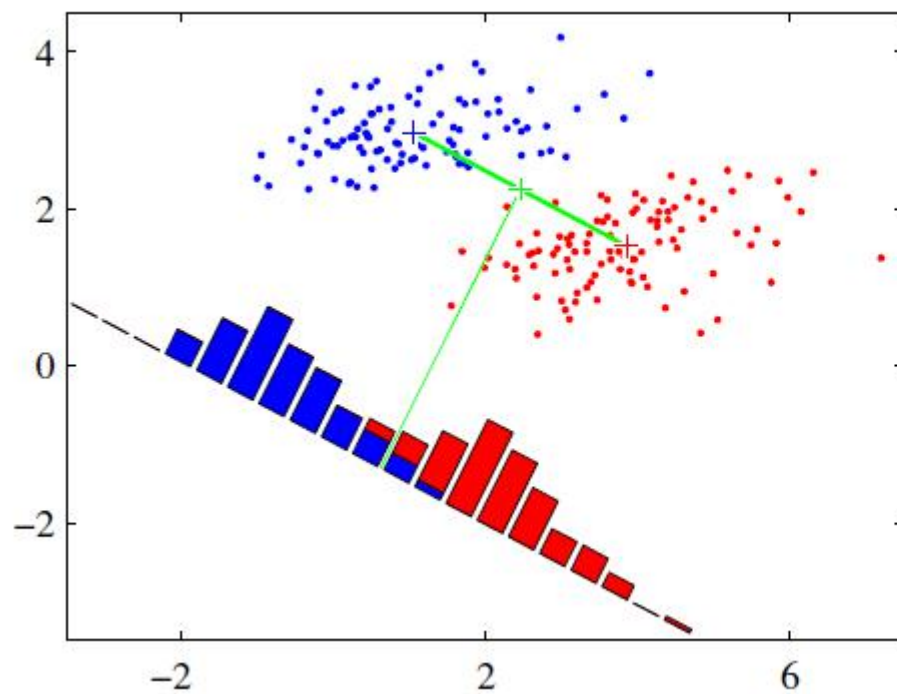
```
[-0.525  1.725  3.325  1.125  1.825 12.775]
[[ 0.02038178 -0.01698103 -0.01350052 -0.0149724  0.03184796 -0.99893718]
 [ 0.9024592  0.25030511 -0.31422084 -0.15092666 -0.03185873  0.01965141]
 [-0.08872116 -0.06952185 -0.06858116 -0.3074396 -0.94204108 -0.02512755]
 [-0.22421522  0.94883808  0.17610905 -0.13278879 -0.01781372 -0.02166191]]
[[-1.01160631e+01  6.49232600e+00  3.14197238e-01 -4.71844785e-16]
 [ 1.08075405e+01  5.73455069e+00 -3.32785235e-01  1.16573418e-15]
 [-1.03473322e+01 -6.08709685e+00 -3.29724759e-01 -1.24900090e-15]
 [ 9.65585479e+00 -6.13977984e+00  3.48312756e-01  2.22044605e-16]]
```

LDA原理

- LDA的全称是Linear Discriminant Analysis（线性判别分析），是一种有监督学习算法。
- LDA的原理是，将带上标签的数据（点），通过投影的方法，投影到维度更低的空间中，使得投影后的点，会形成按类别区分，一簇一簇的情况，相同类别的点，将会在投影后的空间中更接近。用一句话概括就是：“投影后类内方差最小，类间方差最大”



LDA原理



LDA

LDA降维

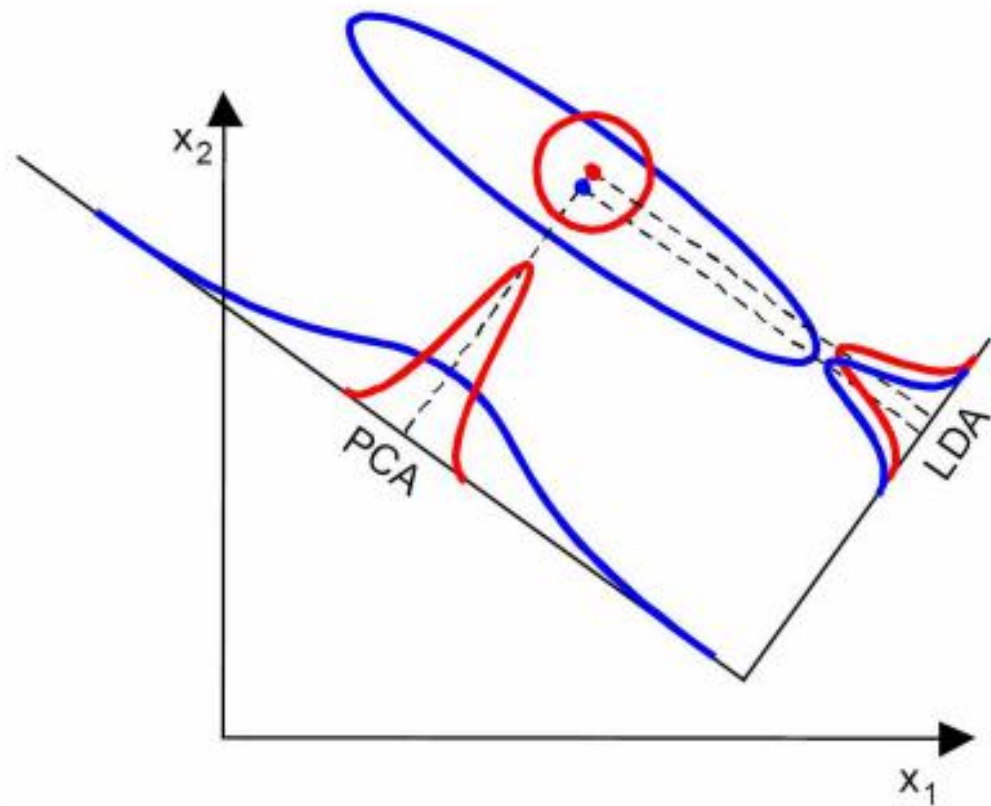
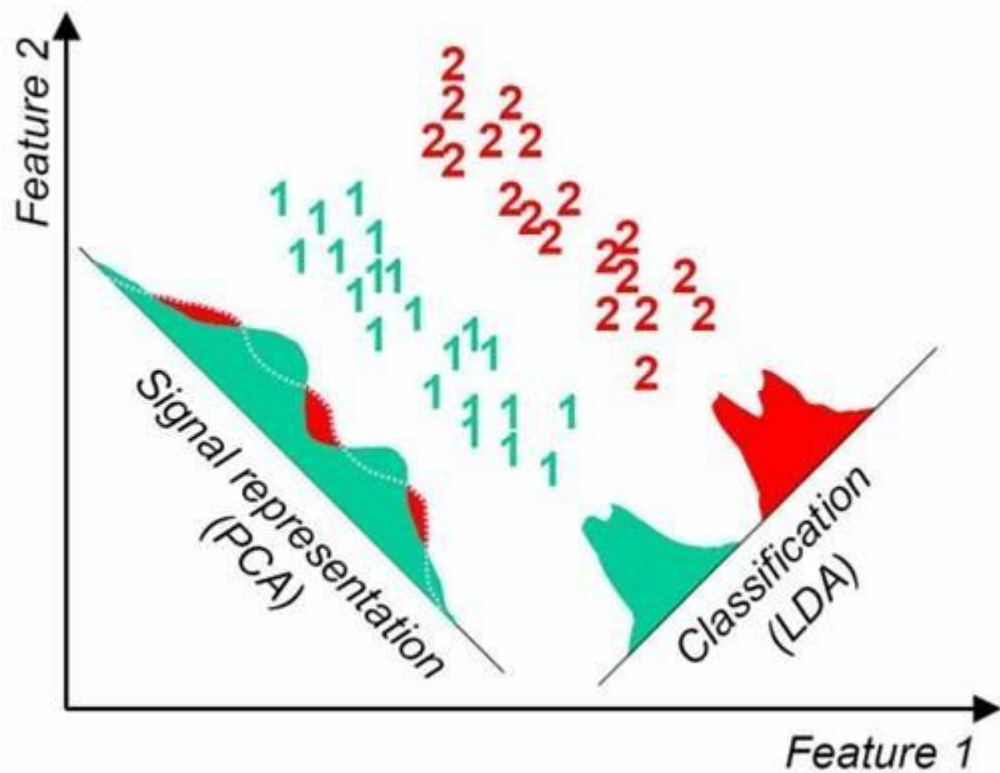
```
: import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
X = np.array([
    [-1, -1, 3, 1],
    [-2, -1, 2, 4],
    [-3, -2, 4, 5],
    [1, 1, 5, 4],
    [2, 1, 6, -5],
    [3, 2, 1, 5]])
y = np.array([1, 1, 2, 2, 0, 1])
# n_components: 给定降低到多少维度, 要求给定的这个值和y的取值数量有关, 不能超过n_class-1
clf = LinearDiscriminantAnalysis(n_components=2)
clf.fit(X, y)
print(clf.transform(X))

[[-3.2688434 -0.38911349]
 [-1.25507558 -1.78088569]
 [ 5.26064254 -0.49688862]
 [ 6.34385833  1.16134391]
 [-4.05800618  3.58297801]
 [-3.02257571 -2.07743411]]
```

PCA和LDA

- 相同点：
 - 两者均可以对数据完成降维操作
 - 两者在降维时候均使用矩阵分解的思想
 - 两者都假设数据符合高斯分布
- 不同点：
 - LDA是监督降维算法，PCA是无监督降维算法
 - LDA降维最多降到类别数目 $k-1$ 的维数，而PCA没有限制
 - LDA除了降维外，还可以应用于分类
 - LDA选择的是分类性能最好的投影，而PCA选择样本点投影具有最大方差的方向

PCA和LDA



特征选择/降维

- 在实际的机器学习项目中，特征选择/降维是必须进行的，因为在数据中存在以下几个方面的问题：
 - 数据的多重共线性：特征属性之间存在着相互关联关系。多重共线性会导致解的空间不稳定，从而导致模型的泛化能力弱；
 - 高维空间样本具有稀疏性，导致模型比较难找到数据特征；
 - 过多的变量会妨碍模型查找规律；
 - 仅仅考虑单个变量对于目标属性的影响可能忽略变量之间的潜在关系。
- 通过降维的目的是：
 - 减少特征属性的个数
 - 确保特征属性之间是相互独立的

