北风网
IBEIFENG.COM

人工智能之机器学习

数据清洗和特征工程

上海育创网络科技股份有限公司

主讲人：刘老师(GerryLiu)

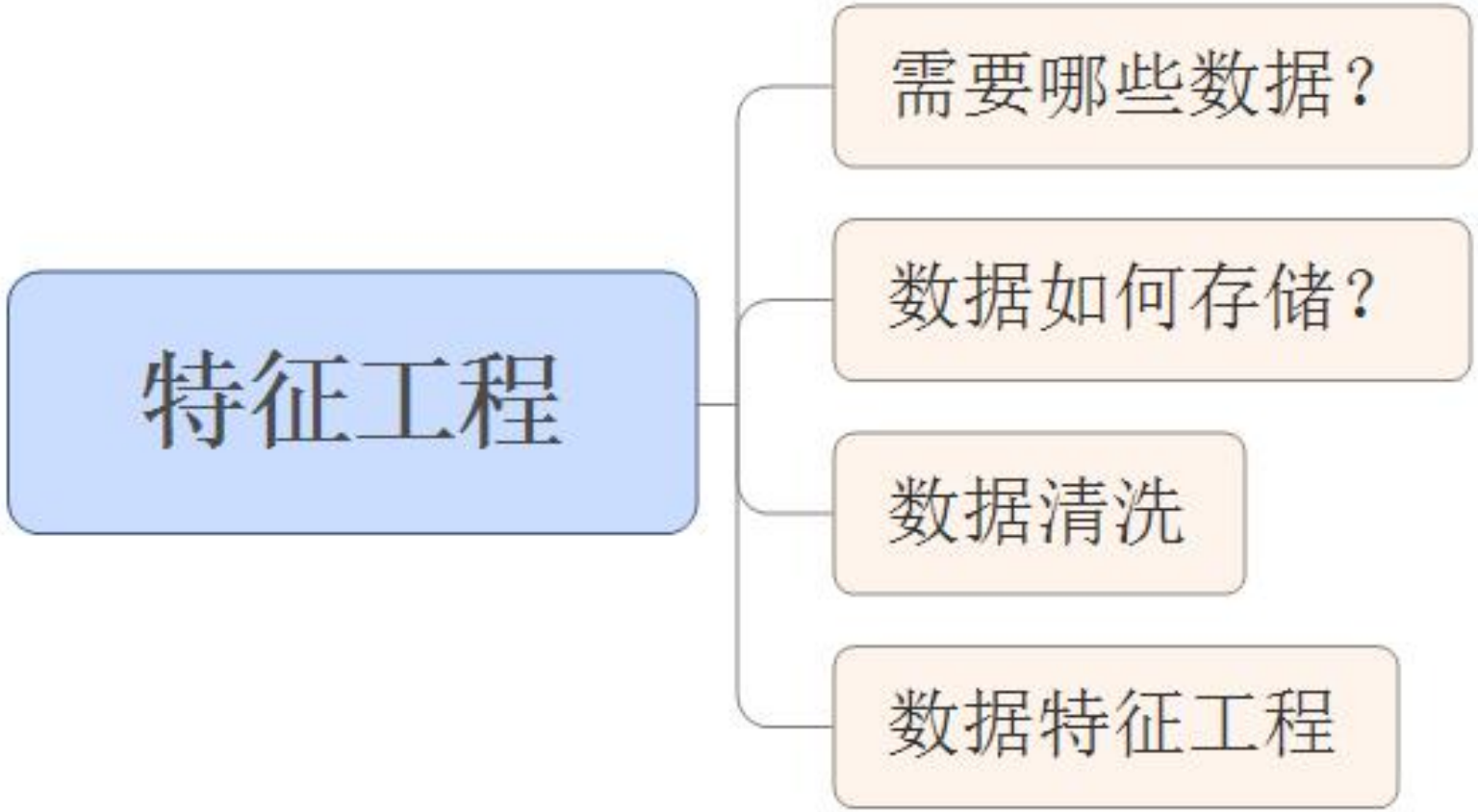
课程要求

- 课上课下“九字”真言
  - 认真听，**善摘录，勤思考**
  - **多温故，乐实践**，再发散
- 四不原则
  - 不懒散惰性，不迟到早退
  - 不请假旷课，不拖延作业

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....

知识要点



需要哪些数据?

- 在进行机器学习之前，收集数据的过程中，我们主要按照以下规则找出我们所需要的数据：
  - 1. 业务的实现需要哪些数据?
    - 基于对业务规则的理解，尽可能多的找出对因变量有影响的所有自变量数据。
  - 2. 数据可用性评估
    - 在获取数据的过程中，首先需要考虑的是这个数据获取的成本；
    - 获取得到的数据，在使用之前，需要考虑一下这个数据是否覆盖了所有情况以及这个数据的可信度情况。

需要哪些数据?

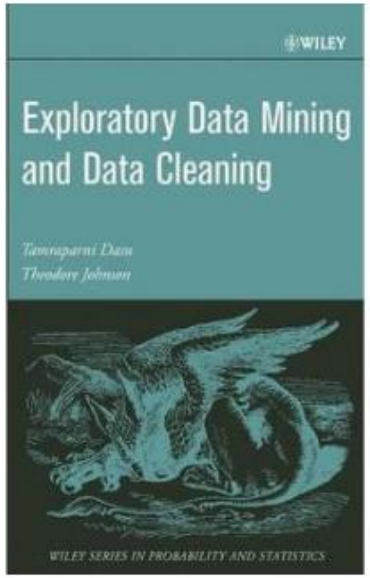
- 一般公司内部做机器学习的数据源:
  - 用户行为日志数据：记录的用户在系统上所有操作所留下来的日志行为数据
  - 业务数据：商品/物品的信息、用户/会员的信息……
  - 第三方数据：爬虫数据、购买的数据、合作方的数据....

数据如何存储?

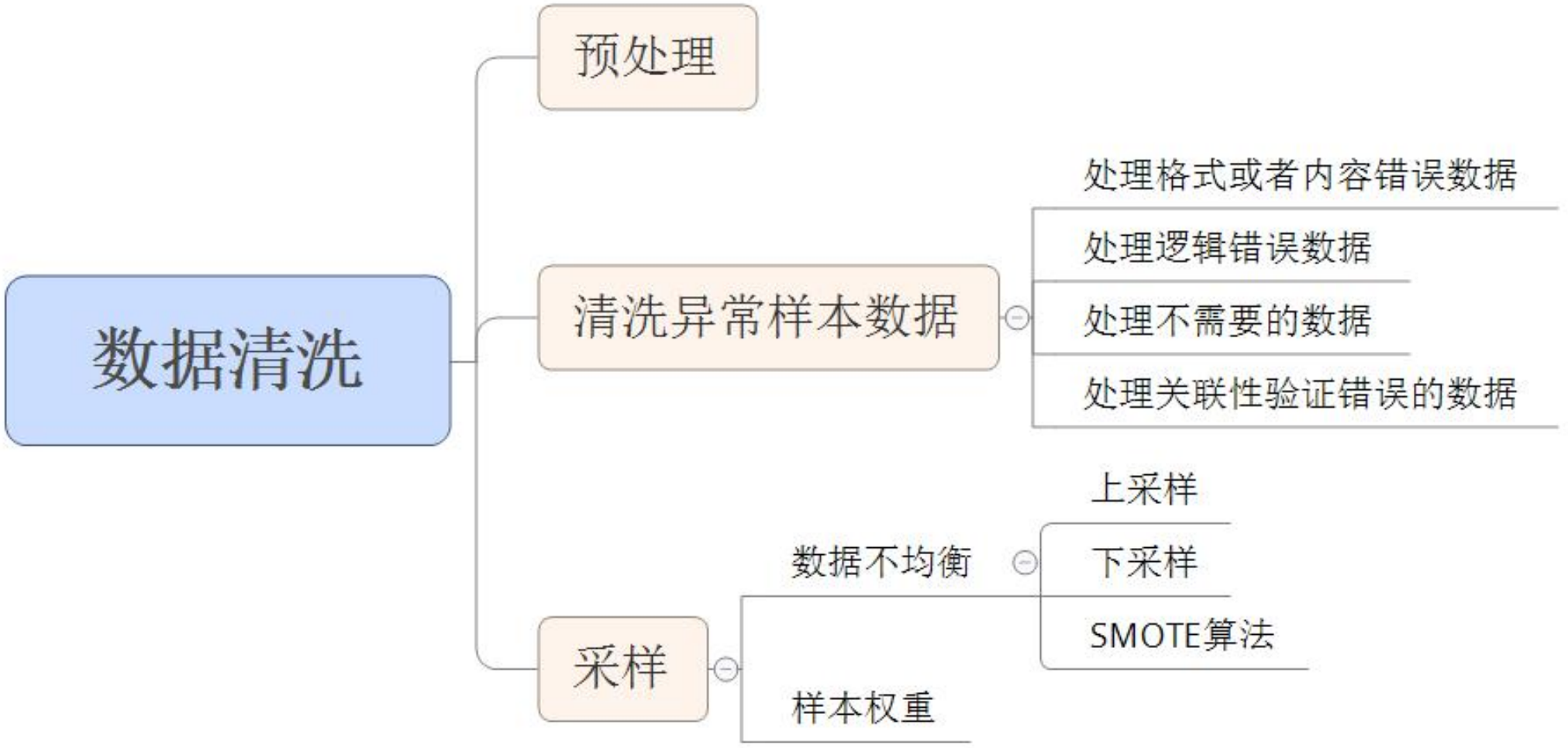
- 一般情况下，用于后期模型创建的数据都是存在在本地磁盘、关系型数据库或者一些相关的分布式数据存储平台的。
  - 本地磁盘
  - MySQL
  - Oracle
  - HBase
  - HDFS
  - Hive

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。



数据清洗过程


数据清洗
预处理
清洗异常样本数据
处理格式或者内容错误数据
处理逻辑错误数据
处理不需要的数据
处理关联性验证错误的数据
采样
数据不均衡
上采样
下采样
SMOTE算法
样本权重

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。

数据不平衡

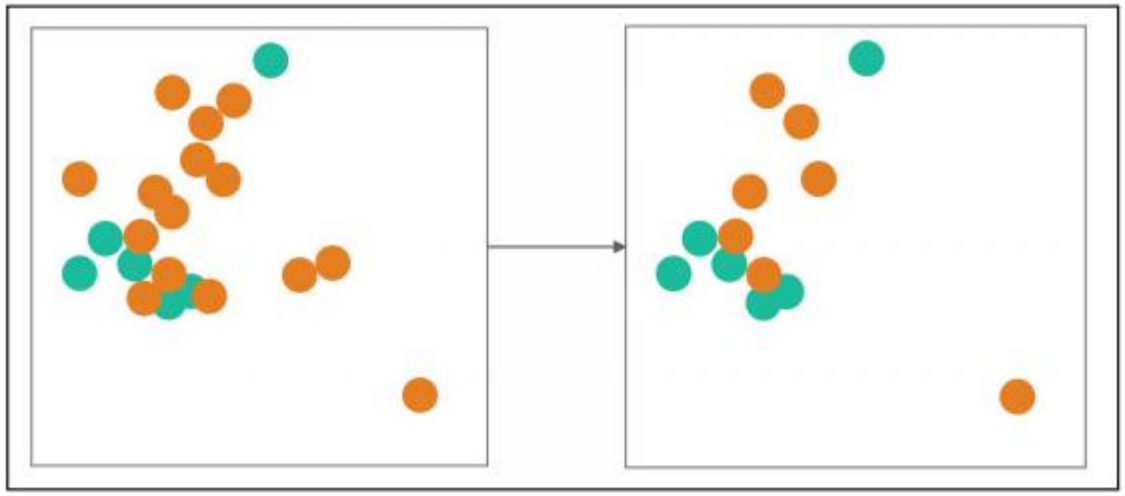
- 在实际应用中，数据往往分布得非常不均匀，也就是会出现“长尾现象”，即绝大多数的数据在一个范围/属于一个类别，而在另外一个范围或者另外一个类别中，只有很少的一部分数据。那么这个时候直接使用机器学习可能效果会不太少，所以这个时候需要我们进行一系列的转换操作。

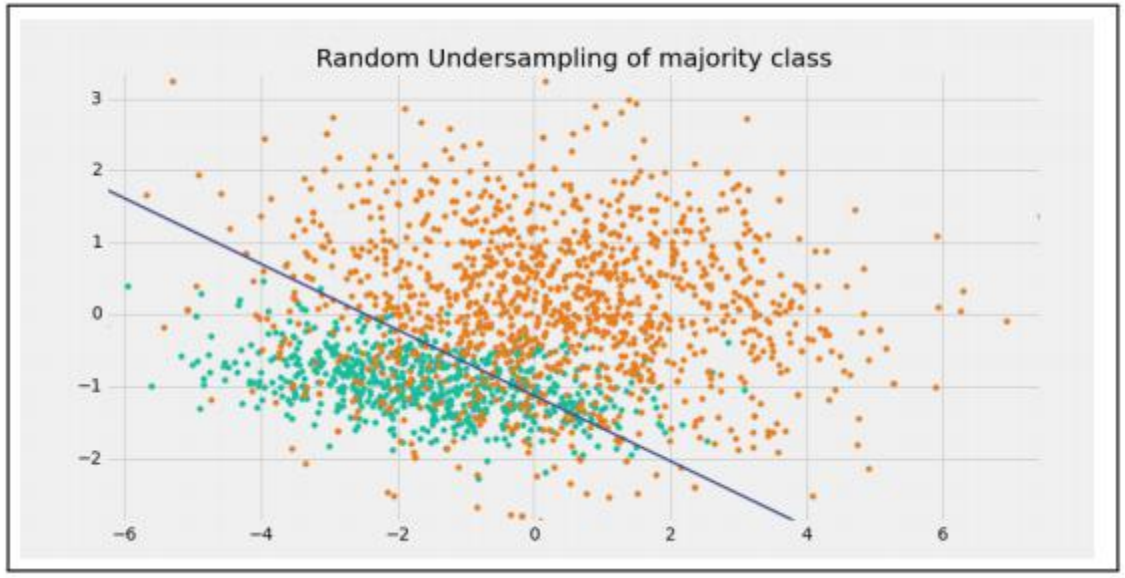
数据不平衡解决方案一

- 设置损失函数的权重，使得少数类别数据判断错误的损失大于多数类别数据判断错误的损失，即当我们的少数类别数据预测错误的时候，会产生一个比较大的损失值，从而导致模型参数往让少数类别数据预测准确的方向偏。可以通过scikit-learn中的class_weight参数来设置权重。

数据不平衡解决方案二

- 下采样/欠采样(under sampling)：从多数类中随机抽取样本从而减少多数类别样本数据，使数据达到平衡的方式。



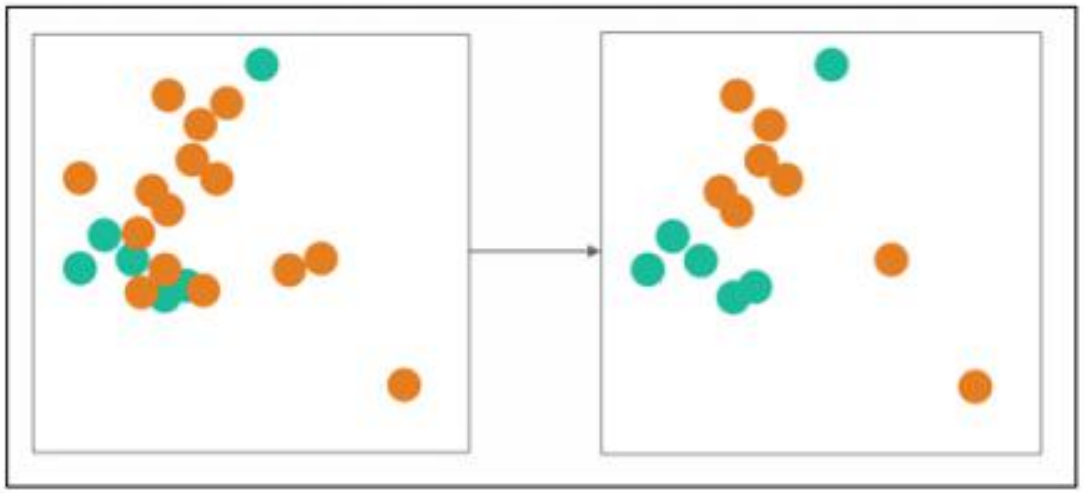


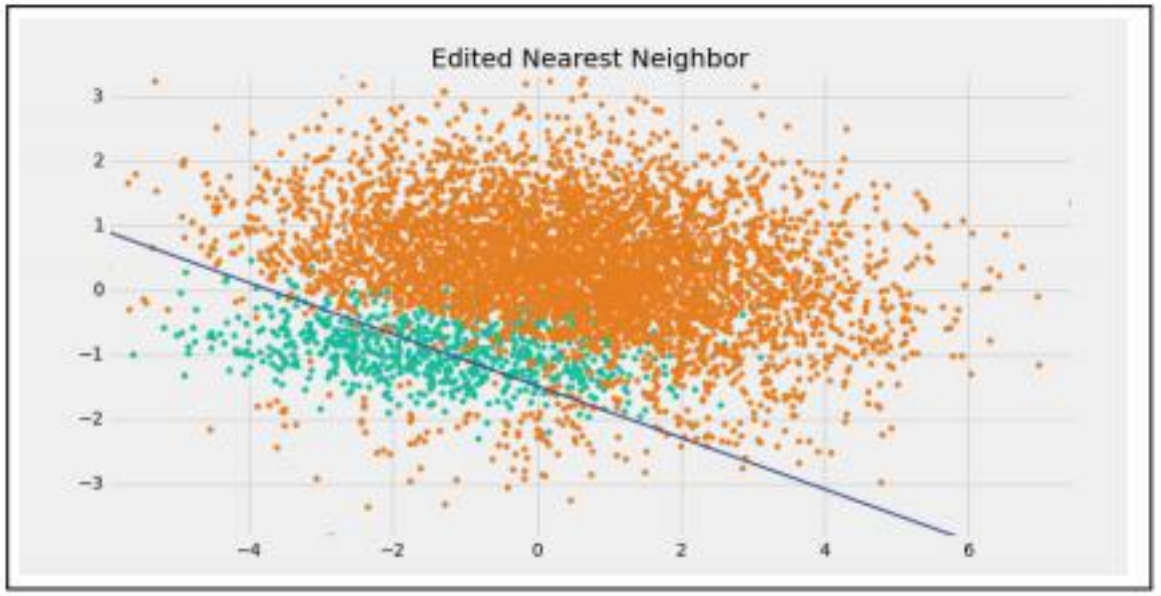
数据不平衡解决方案二

- 集成下采样/欠采样：采用普通的下采样方式会导致信息丢失，所以一般采用集成学习和下采样结合的方式来解决这个问题；主要有两种方式：
  - EasyEnsemble
    - 采用不放回的数据抽取方式抽取多数类别样本数据，然后将抽取出来的数据和少数类别数据组合训练一个模型；多次进行这样的操作，从而构建多个模型，然后使用多个模型共同决策/预测。
  - BalanceCascade
    - 利用Boosting这种增量思想来训练模型；先通过下采样产生训练集，然后使用Adaboost算法训练一个分类器；然后使用该分类器多对所有的大众样本数据进行预测，并将预测正确的样本从大众样本数据中删除；重复迭代上述两个操作，直到大众样本数据量等于小众样本数据量。

数据不平衡解决方案三

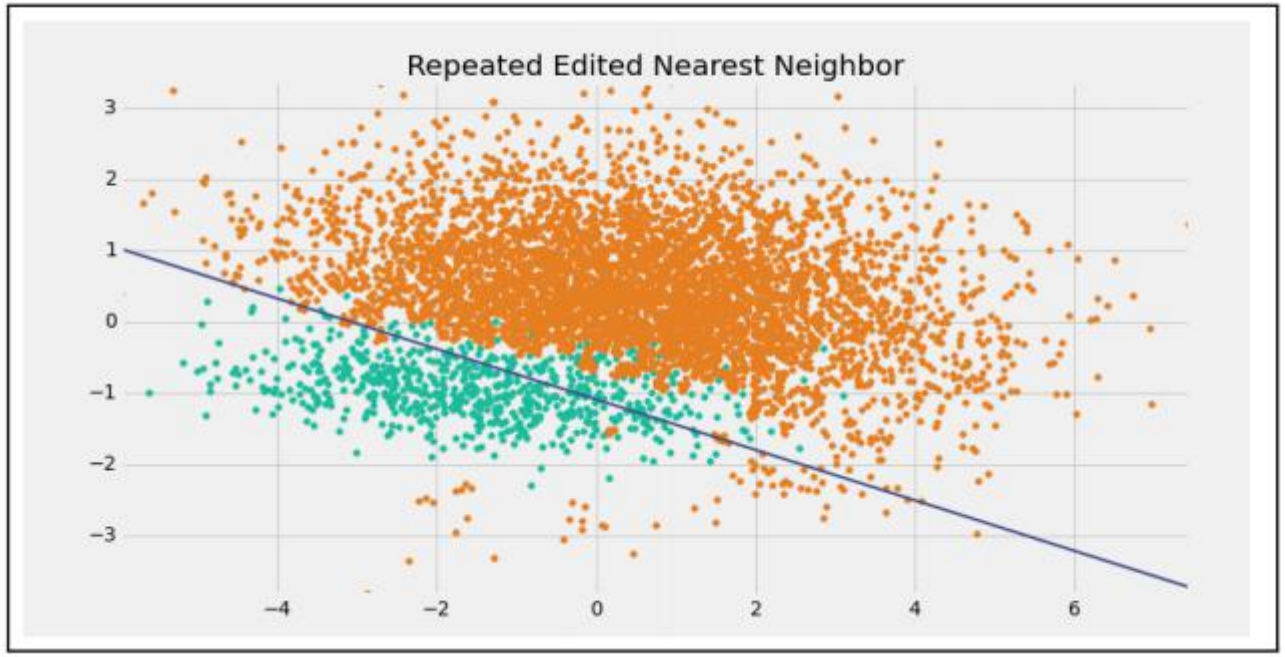
- Edited Nearest Neighbor(ENN)：对于多数类别样本数据而言，如果这个样本的大部分k近邻样本都和自身类别不一样，那我们就将其删除，然后使用删除后的数据训练模型。





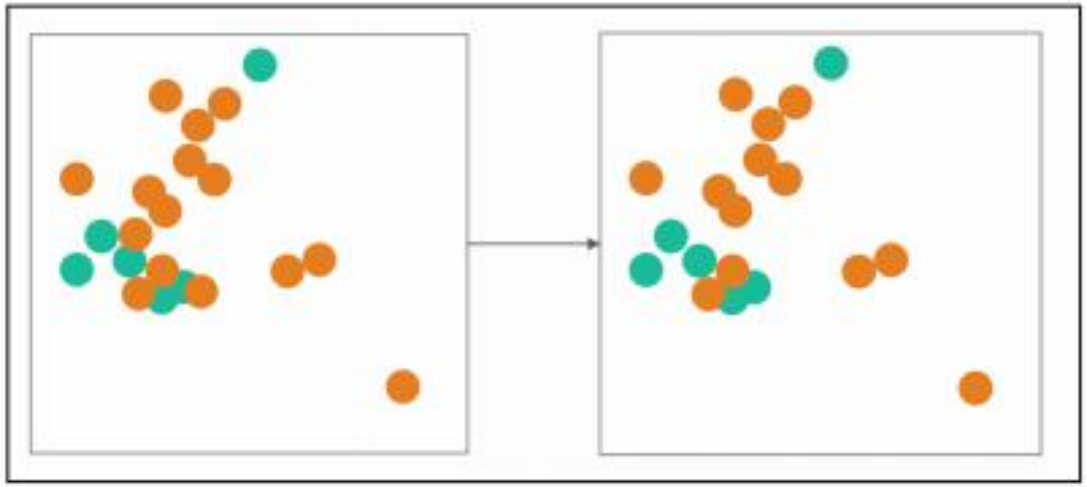
数据不平衡解决方案四

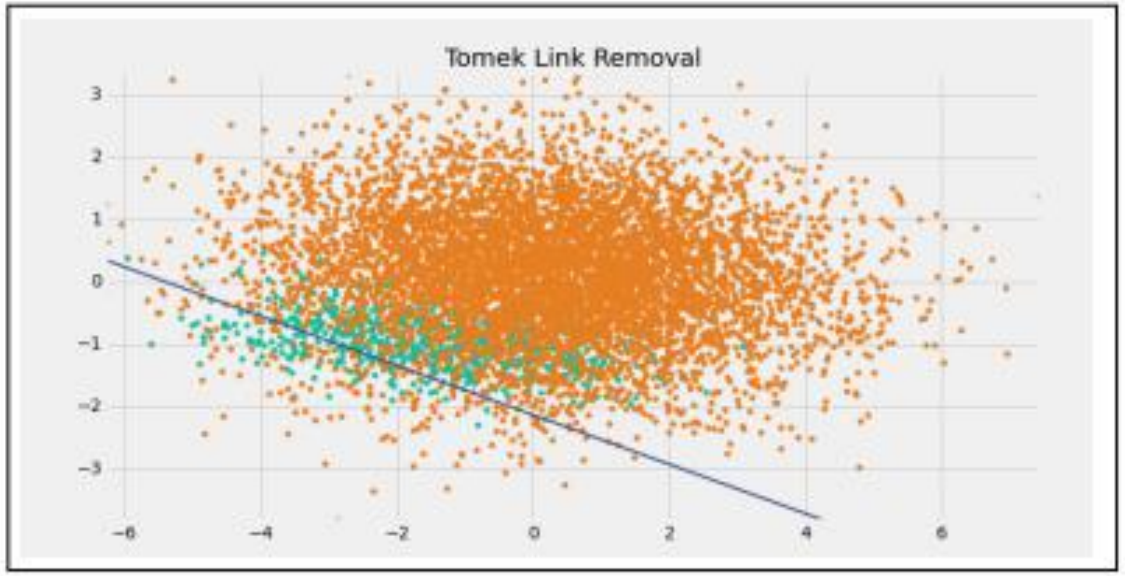
- Repeated Edited Nearest Neighbor(RENN)：对于多数类别样本数据而言，如果这个样本的大部分k近邻样本都和自身类别不一样，那我们就将其删除；重复性的进行上述的删除操作，直到数据集无法再被删除后，使用此时的数据集据训练模型。



数据不平衡解决方案五

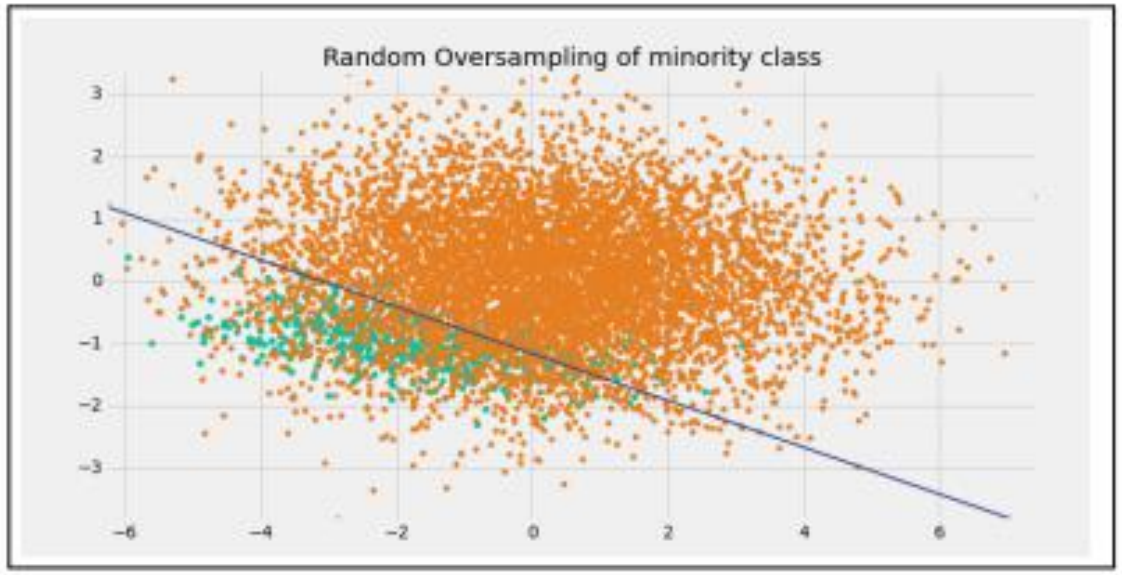
- Tomek Link Removal：如果两个不同类别的样本，它们的最近邻都是对方，也就是A的最近邻是B，B的最近邻也是A，那么A、B就是Tomek Link。将所有Tomek Link中多数类别的样本删除。然后使用删除后的样本来训练模型。





数据不平衡解决方案六

- 过采样/上采样(Over Sampling)：和欠采样采用同样的原理，通过抽样来增加少数样本的数目，从而达到数据平衡的目的。一种简单的方式就是通过有放回抽样，不断的从少数类别样本数据中抽取样本，然后使用抽取样本+原始数据组成训练数据集来训练模型；不过该方式比较容易导致过拟合，一般抽样样本不要超过50%。

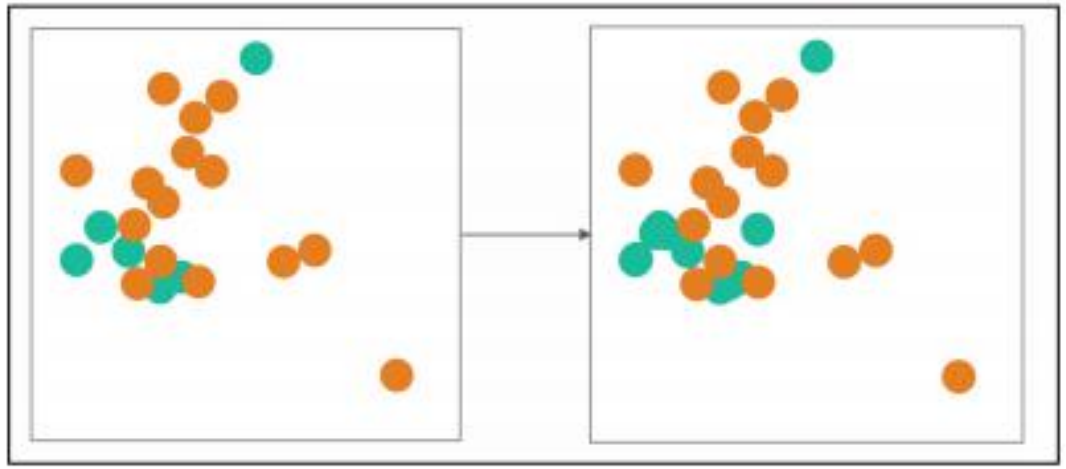


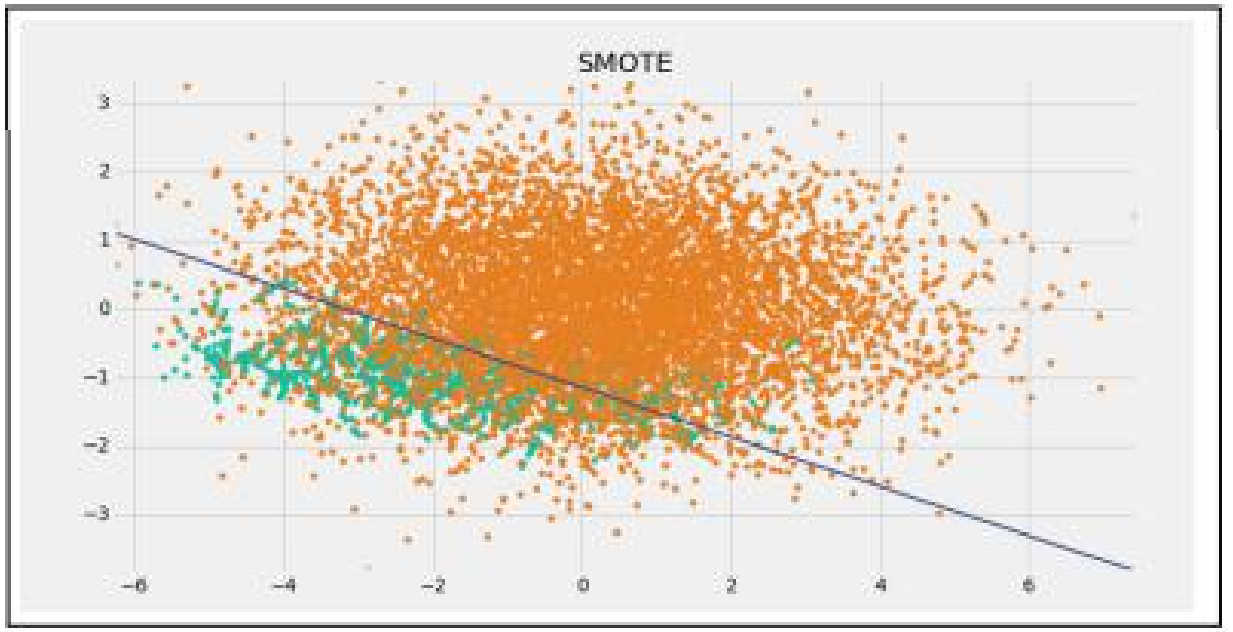
数据不平衡解决方案六

- 过采样/上采样(Over Sampling)：因为在上采样过程中，是进行是随机有放回的抽样，所以最终模型中，数据其实是相当于存在一定的重复数据，为了防止这个重复数据导致的问题，我们可以加入一定的随机性，也就是说：在抽取数据后，对数据的各个维度可以进行随机的小范围变动，eg: (1,2,3) --> (1.01, 1.99, 3)；通过该方式可以相对比较容易的降低上采样导致的过拟合问题。

数据不平衡解决方案七

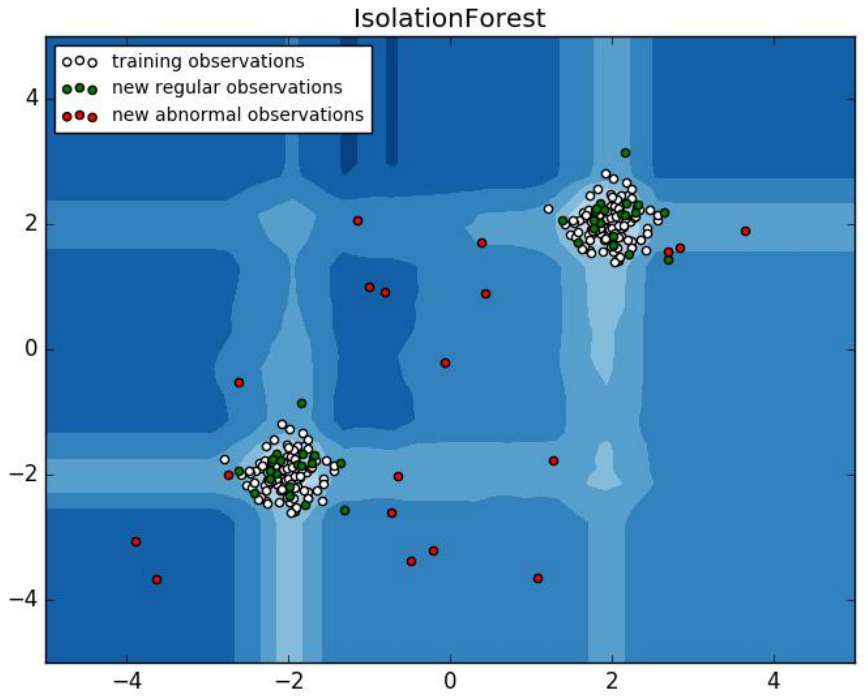
- 采用数据合成的方式生成更多的样本，该方式在小数据集场景下具有比较成功的案例。常见算法是SMOTE算法，该算法利用小众样本在特征空间的相似性来生成新样本。





数据不平衡解决方案八

- 对于正负样本极不平衡的情况下，其实可以换一种思路/角度来看待这个问题：可以将其看成一分类(One Class Learning)或者异常检测(Novelty Detection)问题，在这类算法应用中主要就是对于其中一个类别进行建模，然后对所有不属于这个类别特征的数据就认为是异常数据，经典算法包括：One Class SVM、IsolationForest等。



特征转换

- 特征转换主要指将原始数据中的字段数据进行转换操作，从而得到适合进行算法模型构建的输入数据(数值型数据)，在这个过程中主要包括但不限于以下几种数据的处理：
  - 文本数据转换为数值型数据
  - 缺省值填充
  - 定性特征属性哑编码
  - 定量特征属性二值化
  - 特征标准化与归一化
  - 基于业务产生新的特征属性

分词

- 分词是指将文本数据转换为一个一个的单词，是NLP自然语言处理过程中的基础；因为对于文本信息来讲，我们可以认为文本中的单词可以体现文本的特征信息，所以在进行自然语言相关的机器学习的时候，第一步操作就是需要将文本信息转换为单词序列，使用单词序列来表达文本的特征信息。
- 分词：通过某种技术将连续的文本分隔成更具有语言语义学上意义的词。这个过程就叫做分词。
- Python中汉字分词包：jieba
  - 安装方式： pip install jieba
  - Github：https://github.com/fxsjy/jieba

```
C:\Users\ibf>pip install jieba
Collecting jieba
  Downloading jieba-0.39.zip (7.3MB)
    100% |████████████████████████████████|
Building wheels for collected packages: jieba
  Running setup.py bdist_wheel for jieba ...
  Stored in directory: C:\Users\ibf\AppData\L
811b0d
Successfully built jieba
Installing collected packages: jieba
Successfully installed jieba-0.39

C:\Users\ibf>python
Python 3.6.0 |Anaconda 4.3.1 (64-bit)| (defau
Type "help", "copyright", "credits" or "licen
>>> import jieba
```

分词

- 分词的常见方法
  - 按照文本/单词特征进行划分：对于英文文档，可以基于空格进行单词划分。
  - 词典匹配：匹配方式可以从左到右，从右到左。对于匹配中遇到的多种分段可能性，通常会选取分隔出来词的数目最小的。
  - 基于统计的方法：隐马尔可夫模型（HMM）、最大熵模型（ME），估计相邻汉字之间的关联性，进而实现切分
  - 基于深度学习：神经网络抽取特征、联合建模

Jieba分词

- jieba：中文分词模块
- jieba分词原理：
  - 字符串匹配：把汉字串与词典中的词条进行匹配，识别出一个词
  - 理解分词法：通过分词子系统、句法语义子系统、总控部分来模拟人对句子的理解。（试验阶段）
  - 统计分词法：建立大规模语料库，通过隐马尔可夫模型或其他模型训练，进行分词（主流方法）

Jieba分词使用

- jieba分词模式：
  - 全模式jieba.cut(str,cut_all=True)
  - 精确模式jieba.cut(str)
  - 搜索引擎模式jieba.cut_for_search(str)
- 分词特征提取：返回TF/IDF权重最大的关键词，默认返回20个
  - jieba.analyse.extract_tags(str,topK=20)
- 自定义词典：帮助切分一些无法识别的新词，加载词典：jieba.load_userdict（‘dict.txt’）
- 调整词典：add_word(word, freq=None, tag=None) 和 del_word(word) 可在程序中动态修改词典。使用 suggest_freq(segment, tune=True) 可调节单个词语的词频

文本特征属性转换

- 机器学习的模型算法均要求输入的数据必须是数值型的，所以对于文本类型的特征属性，需要进行文本数据转换，也就是需要将文本数据转换为数值型数据。常用方式如下：
  - 词袋法(BOW/TF)
  - TF-IDF(Term frequency-inverse document frequency)
  - HashTF
  - Word2Vec(主要用于单词的相似性考量)

词袋法

- 词袋法(Bag of words, BOW)是最早应用于NLP和IR领域的一种文本处理模型，该模型忽略文本的语法和语序，用一组无序的单词(words)来表达一段文字或者一个文档，词袋法中使用单词在文档中出现的次数(频数)来表示文档。

d1:this is a sample is a sample

d2:this is anthoer example anthoer example

dict(词典、字典、特征属性): this sample anthoer example

	this	another	sample	example
d_1	1	0	2	0
d_2	1	2	0	2

词集法

- 词集法(Set of words, SOW)是词袋法的一种变种，应用的比较多，和词袋法的原理一样，是以文档中的单词来表示文档的一种的模型，区别在于：词袋法使用的是单词的频数，而在词集法中使用的是单词是否出现，如果出现赋值为1，否则为0。

d1:this is a sample is a sample

d2:this is anthoer example anthoer example

dict: this sample anthoer example

	this	another	sample	example
d_1	1	0	1	0
d_2	1	1	0	1

TF-IDF

- 在词袋法或者词集法中，使用的是单词的词频或者是否存在来进行表示文档特征，但是不同的单词在不同文档中出现的次数不同，而且有些单词仅仅在某一些文档中出现(eg：专业名称等等)，也就是说不同单词对于文本而言具有不同的重要性，那么，如何评估一个单词对于一个文本的重要性呢?
  - **单词的重要性随着它在文本中出现的次数成正比增加，也就是单词的出现次数越多，该单词对于文本的重要性就越高。**
  - **同时单词的重要性会随着在语料库中出现的频率成反比下降，也就是单词在语料库中出现的频率越高，表示该单词越常见，也就是该单词对于文本的重要性越低。**

TF-IDF

- TF-IDF(Item frequency-inverse document frequency)是一种常用的用于信息检索与数据挖掘的常用加权技术，TF的意思是词频(Item Frequency)，IDF的意思是逆向文件频率(Inverse Document Frequency)。
- TF-IDF可以反映语料中单词对文档/文本的重要程度。

TF-IDF

- 假设单词用t表示，文档用d表示，语料库用D表示，那么N(t,D)表示包含单词t的文档数量，|D|表示文档数量，|d|表示文档d中的所有单词数量。N(t,d)表示在文档d中单词t出现的次数。

$$TFIDF(t,d,D) = TF(t,d) * IDF(t,D)$$

$$TF(t,d) = \frac{N(t,d)}{|d|}$$

$$IDF(t,D) = \log\left(\frac{|D|+1}{N(t,D)+1}\right)$$

TF-IDF

- TF-IDF除了使用默认的tf和idf公式外，tf和idf公式还可以使用一些扩展之后公式来进行指标的计算，常用的公式有：

Variants of term frequency (TF) weight

weighting scheme	TF weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$
log normalization	$1 + \log(f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K) \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

Variants of inverse document frequency (IDF) weight

weighting scheme	IDF weight ($n_t = \lvert\{d \in D : t \in d\}\rvert$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log\left(1 + \frac{N}{n_t}\right)$
inverse document frequency max	$\log\left(\frac{\max_{\{t' \in d\}} n_{t'}}{1 + n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N - n_t}{n_t}$

TF-IDF

- 有两个文档，单词统计如下，请分别计算各个单词在文档中的TF-IDF值以及这些文档使用单词表示的特征向量。

单词	单词次数
this	1
is	1
a	2
sample	1

单词	单词次数
this	2
is	1
another	2
example	3

$$tf("this", d_1) = \frac{1}{5} \quad tf("this", d_2) = \frac{2}{8}$$

$$idf("this", D) = \log\left(\frac{2+1}{2+1}\right) = 0$$

$$tfidf("this", d_1) = tf("this", d_1) * idf("this", D) = 0$$

$$tfidf("this", d_2) = 0$$

	this	is	a	another	sample	example
d_1	0	0	0.191	0	0.095	0
d_2	0	0	0	0.119	0	0.179

HashTF-IDF

- 不管是前面的词袋法还是TF-IDF，都避免不了计算文档中单词的词频，当文档数量比较少、单词数量比较少的时候，我们的计算量不会太大，但是当这个数量上升到一定程度的时候，程序的计算效率就会降低下去，这个时候可以通过HashTF的形式来解决该问题。HashTF的计算规则是：**在计算过程中，不计算单词的词频，而是计算单词进行hash后的hash值对应的频次**；
- HashTF的特点：运行速度快，但是无法获取高频词，有可能存在单词碰撞问题(hash值一样)

Scikit Text Feature Extraction

- 在scikit中，对于文本数据主要提供了三种方式将文本数据转换为数值型的特征向量，同时提供了一种对TF-IDF公式改版的公式。所有的转换方式均位于模块：sklearn.feature_extraction.text

名称	描述
CountVectorizer	以词袋法的形式表示文档
HashingVectorizer	以HashingTF的模型来表示文档的特征向量
TfidfVectorizer	以TF-IDF的模型来表示文档的特征向量，等价于先做CountVectorizer，然后做TfidfTransformer转换操作的结果
TfidfTransformer	使用改进的TF-IDF公式对文档的特征向量矩阵(数值型的)进行重计算的操作，TFIDF=TF*(IDF+1)；<备注：该转换常应用到CountVectorizer或者HashingVectorizer之后>

Scikit Text Feature Extraction

class sklearn.feature_extraction.text. **CountVectorizer** (*input='content', encoding='utf-8', decode_error='strict', strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, stop_words=None, token_pattern='(?u)\b\w\w+\b', ngram_range=(1, 1), analyzer='word', max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dtype=<class 'numpy.int64'>*) ¶ [source]

fit (raw_documents[, y])	Learn a vocabulary dictionary of all tokens in the raw documents.
fit_transform (raw_documents[, y])	Learn the vocabulary dictionary and return term-document matrix.
get_feature_names ()	Array mapping from feature integer indices to feature name
get_params ([deep])	Get parameters for this estimator.
get_stop_words ()	Build or fetch the effective stop words list
inverse_transform (X)	Return terms per document with nonzero entries in X.
set_params (**params)	Set the parameters of this estimator.
transform (raw_documents)	Transform documents to document-term matrix.

Scikit Text Feature Extraction

class sklearn.feature_extraction.text. **HashingVectorizer** (*input='content', encoding='utf-8', decode_error='strict', strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, stop_words=None, token_pattern='(?u)\b\w\w+\b', ngram_range=(1, 1), analyzer='word', n_features=1048576, binary=False, norm='l2', non_negative=False, dtype=<class 'numpy.float64'>*) [source]

`fit` (X[, y])	Does nothing: this transformer is stateless.
`fit_transform` (X[, y])	Transform a sequence of documents to a document-term matrix.
`get_params` ([deep])	Get parameters for this estimator.
`get_stop_words` ()	Build or fetch the effective stop words list
`partial_fit` (X[, y])	Does nothing: this transformer is stateless.
`set_params` (**params)	Set the parameters of this estimator.
`transform` (X[, y])	Transform a sequence of documents to a document-term matrix.

Scikit Text Feature Extraction

class sklearn.feature_extraction.text. **TfidfVectorizer** (*input='content', encoding='utf-8', decode_error='strict', strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, analyzer='word', stop_words=None, token_pattern='(?u)\b\w\w+\b', ngram_range=(1, 1), max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dtype=<class 'numpy.int64'>, norm='l2', use_idf=True, smooth_idf=True, sublinear_tf=False*) [source]

fit (raw_documents[, y])	Learn a vocabulary dictionary of all tokens in the raw documents.
fit_transform (raw_documents[, y])	Learn the vocabulary dictionary and return term-document matrix.
get_feature_names ()	Array mapping from feature integer indices to feature name
get_params ([deep])	Get parameters for this estimator.
get_stop_words ()	Build or fetch the effective stop words list
inverse_transform (X)	Return terms per document with nonzero entries in X.
set_params (**params)	Set the parameters of this estimator.
transform (raw_documents)	Transform documents to document-term matrix.

Scikit Text Feature Extraction

class sklearn.feature_extraction.text. **TfidfTransformer** (*norm='l2'*, *use_idf=True*, *smooth_idf=True*, *sublinear_tf=False*) ¶ [source]

fit (X[, y])	Learn the idf vector (global term weights)
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[, copy])	Transform a count matrix to a tf or tf-idf representation

Scikit Text Feature Extraction


```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer,HashingVectorizer,TfidfTransformer
```

```
arr1 = [
    "This is spark, spark sql a every good",
    "Spark Hadoop Hbase",
    "This is sample",
    "This is anthor example anthor example",
    "spark hbase hadoop spark hive hbase hue oozie",
    "hue oozie spark"
]
arr2 = [
    "this is a sample a example",
    "this c c cd is another another sample example example",
    "spark Hbase hadoop Spark hive hbase"
]
df = arr2
```

```
tfidf = TfidfVectorizer(min_df=0, dtype=np.float64)
df2 = tfidf.fit_transform(df)
print (df2.toarray())
print (tfidf.get_feature_names())
print (tfidf.get_stop_words())
print ("转换另外的文档数据")
print (tfidf.transform(arr1).toarray())
```

```
[[0.         0.         0.5        0.         0.         0.
  0.5        0.5        0.         0.5       ]
 [0.66486672 0.33243336 0.50564828 0.         0.         0.
  0.25282414 0.25282414 0.         0.25282414]
 [0.         0.         0.         0.31622777 0.63245553 0.31622777
  0.         0.         0.63245553 0.        ]]
['another', 'cd', 'example', 'hadoop', 'hbase', 'hive', 'is', 'sample', 'spark', 'this']
None
转换另外的文档数据
[[0.         0.         0.         0.         0.         0.
  0.3349067  0.         0.88072413 0.3349067 ]
 [0.         0.         0.         0.57735027 0.57735027 0.
  0.         0.         0.57735027 0.        ]
 [0.         0.         0.         0.         0.         0.
  0.57735027 0.57735027 0.         0.57735027]
 [0.         0.         0.81649658 0.         0.         0.
  0.40824829 0.         0.         0.40824829]
 [0.         0.         0.         0.31622777 0.63245553 0.31622777
  0.         0.         0.63245553 0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         1.         0.        ]]
```

```
count = CountVectorizer(min_df=0.1, dtype=np.float64, ngram_range=(0,1))
df4 = count.fit_transform(df)
print (df4.toarray())
print (count.get_stop_words())
print (count.get_feature_names())
print ("转换另外的文档数据")
print (count.transform(arr1).toarray())
print(df4)
```

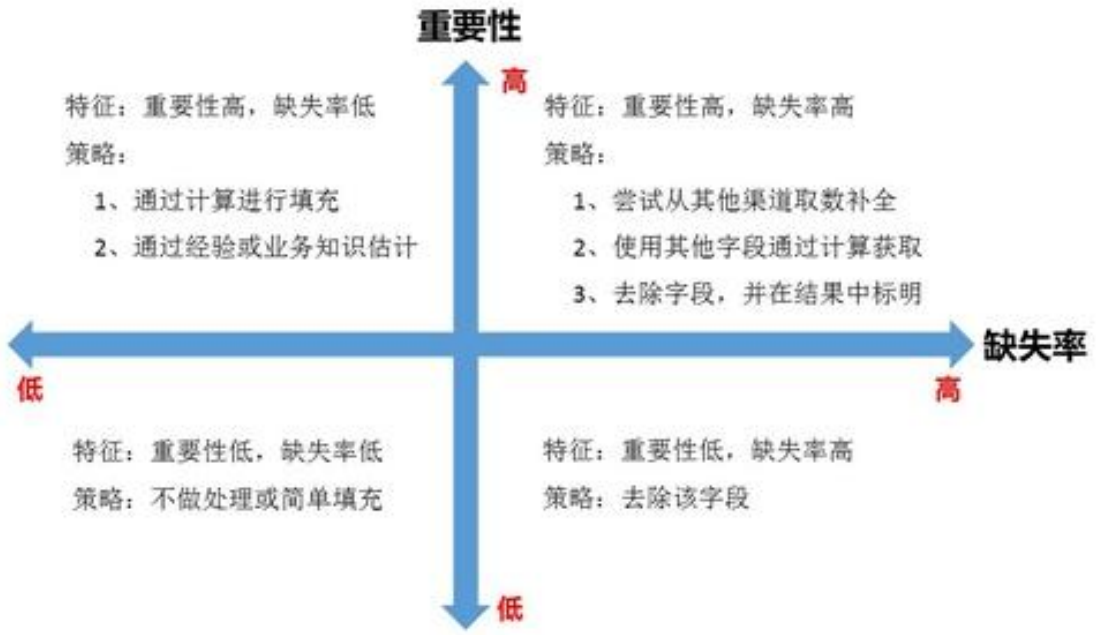
```
[[0. 0. 1. 0. 0. 0. 1. 1. 0. 1.]
 [2. 1. 2. 0. 0. 0. 1. 1. 0. 1.]
 [0. 0. 0. 1. 2. 1. 0. 0. 2. 0.]]
None
['another', 'cd', 'example', 'hadoop', 'hbase', 'hive', 'is', 'sample', 'spark', 'this']
转换另外的文档数据
[[0. 0. 0. 0. 0. 0. 1. 0. 2. 1.]
 [0. 0. 0. 1. 1. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 1. 1. 0. 1.]
 [0. 0. 2. 0. 0. 0. 1. 0. 0. 1.]
 [0. 0. 0. 1. 2. 1. 0. 0. 2. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]]
  (0, 2)        1.0
  (0, 7)        1.0
  (0, 6)        1.0
  (0, 9)        1.0
  (1, 0)        2.0
  (1, 1)        1.0
  (1, 2)        2.0
  (1, 7)        1.0
  (1, 6)        1.0
  (1, 9)        1.0
  (2, 5)        1.0
  (2, 3)        1.0
  (2, 4)        2.0
  (2, 8)        2.0
```

缺省值填充

- 缺省值是数据中最常见的一个问题，处理缺省值有很多方式，主要包括以下四个步骤进行缺省值处理：
  - 确定缺省值范围
  - 去除不需要的字段
  - 填充缺省值内容
  - 重新获取数据
- 注意：最重要的是**缺省值内容填充。**

缺省值填充

- 在进行确定缺省值范围的时候，对每个字段都计算其缺失比例，然后按照缺失比例和字段重要性分别指定不同的策略。



缺省值填充

- 在进行去除不需要的字段的时候，需要注意的是：删除操作最好不要直接操作与原始数据上，最好的是抽取部分数据进行删除字段后的模型构建，查看模型效果，如果效果不错，那么再到全量数据上进行删除字段操作。总而言之：该过程简单但是必须慎用，不过一般效果不错，删除一些丢失率高以及重要性低的数据可以降低模型的训练复杂度，同时又不会降低模型的效果。

缺省值填充

- 填充缺省值内容是一个比较重要的过程，也是我们常用的一种缺省值解决方案，一般采用下面几种方式进行数据的填充：
  - 以业务知识或经验推测填充缺省值
  - 以同一字段指标的计算结果(均值、中位数、众数等)填充缺省值
  - 以不同字段指标的计算结果来推测性的填充缺省值，比如通过身份证号码计算年龄、通过收货地址来推测家庭住址、通过访问的IP地址来推测家庭/公司/学校的家庭住址等等

缺省值填充

- 如果某些指标非常重要，但是缺失率有比较高，而且通过其它字段没法比较精准的计算出指标值的情况下，那么就需要和数据产生方(业务人员、数据收集人员等)沟通协商，是否可以通过其它的渠道获取相关的数据，也就是进行重新获取数据的操作。

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，在scikit中主要通过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

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

```
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]
]
```

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

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

```
[[2.         6.         3.         1.        ]
 [1.66666667 5.         3.         1.        ]
 [4.         1.         3.         5.        ]
 [1.66666667 2.         3.         1.        ]]
------------------------
[[2.         6.         3.         1.        ]
 [3.         5.         3.         1.        ]
 [4.         1.         3.33333333 5.        ]
 [1.         1.         1.         1.        ]]
```

```
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)
```

```
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.]]
```

哑编码

- 哑编码(OneHotEncoder)：对于定性的数据(也就是分类的数据)，可以采用N位的状态寄存器来对N个状态进行编码，每个状态都有一个独立的寄存器位，并且在任意状态下只有一位有效；是一种常用的将特征数字化的方式。比如有一个特征属性:['male','female']，那么male使用向量[1,0]表示，female使用[0,1]表示。

class sklearn.preprocessing. **OneHotEncoder** (*n_values='auto', categorical_features='all', dtype=<class 'float'>, sparse=True, handle_unknown='error'*) [source]

class sklearn.feature_extraction. **DictVectorizer** (*dtype=<class 'numpy.float64'>, separator='=', sparse=True, sort=True*) ¶ [source]

class sklearn.feature_extraction. **FeatureHasher** (*n_features=1048576, input_type='dict', dtype=<class 'numpy.float64'>, non_negative=False*) ¶ [source]

哑编码

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.feature_extraction import DictVectorizer,FeatureHasher
```

```
enc = OneHotEncoder()
enc.fit([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2],[1,1,1]])
a=np.array([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]])
print(a)
print("编码结果",enc.n_values_)
```

```
[[0 0 3]
 [1 1 0]
 [0 2 1]
 [1 0 2]]
编码结果 [2 3 4]
```

```
# sparse: 最终产生的结果是否是稀疏化矩阵，默认为True，一般不改动
dv = DictVectorizer(sparse=False)
D = [{'foo':1, 'bar':2.2}, {'foo':3, 'baz': 2}]
X = dv.fit_transform(D)
print (X)
# 直接把字典中的key作为特征，value作为特征值，然后构建特征矩阵
print (dv.get_feature_names())
print (dv.transform({'foo':4, 'unseen':3}))
```

```
[[2.2 0.  1. ]
 [0.  2.  3. ]]
['bar', 'baz', 'foo']
[[0. 0. 4.]]
```

```
h = FeatureHasher(n_features=10)
D = [{'dog': 1, 'cat':2, 'elephant':4},{'dog': 2, 'run': 5}]
f = h.transform(D)
f.toarray()
```

```
array([[ 0.,  0., -4., -1.,  0.,  0.,  0.,  0.,  0.,  2.],
       [ 0.,  0.,  0., -2., -5.,  0.,  0.,  0.,  0.,  0.]])
```

二值化/连续数据区间化

- 二值化(Binarizer)：对于定量的数据（特征取值连续）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]

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

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

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

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

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

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

标准化（z-score）

- 标准化：基于特征属性的数据(也就是特征矩阵的列)，获取均值和方差，然后将特征值转换至服从标准正态分布。计算公式如下：

$$x' = \frac{x - \overline{X}}{S}$$

class sklearn.preprocessing. **StandardScaler** (*copy=True, with_mean=True, with_std=True*) [source]

```
from sklearn.preprocessing import StandardScaler
```

```
X = [
    [1, 2, 3, 2],
    [7, 8, 9, 2.01],
    [4, 8, 2, 2.01],
    [9, 5, 2, 1.99],
    [7, 5, 3, 1.99],
    [1, 4, 9, 2]
]
```

```
ss = StandardScaler(with_mean=True, with_std=True)
ss.fit(X)
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
print(ss.mean_)
print(ss.n_samples_seen_)
print(ss.scale_)
```

```
[4.83333333 5.33333333 4.66666667 2.        ]
6
[3.07769755 2.13437475 3.09120617 0.00816497]
```

```
print(ss.transform(X))
```

```
[[-1.24551983 -1.56173762 -0.53916387  0.        ]
 [ 0.70398947  1.2493901   1.40182605  1.22474487]
 [-0.27076518  1.2493901  -0.86266219  1.22474487]
 [ 1.3538259  -0.15617376 -0.86266219 -1.22474487]
 [ 0.70398947 -0.15617376 -0.53916387 -1.22474487]
 [-1.24551983 -0.62469505  1.40182605  0.        ]]
```

区间缩放法

- 区间缩放法：是指按照数据(特征属性，也就是列)的取值范围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.        ]]
```

正则化

- 正则化：和标准化不同，正则化是基于矩阵的行进行数据处理，其目的是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]

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

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

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

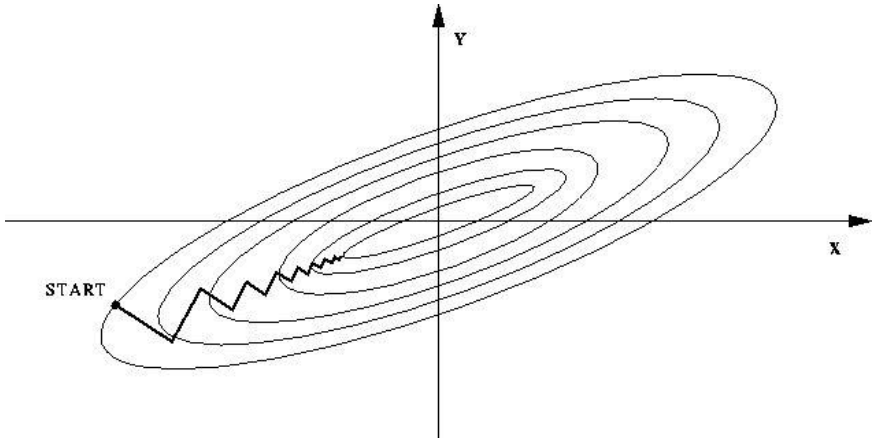
```
Normalizer(copy=True, norm='l2')
```

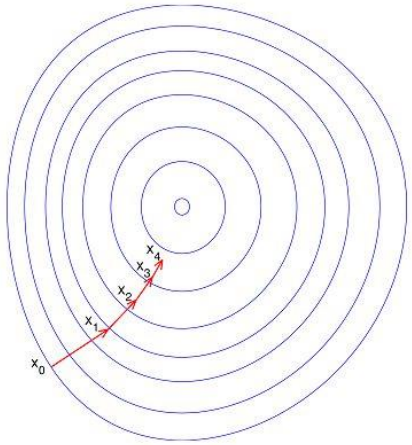
```
print(normalizer1.transform(X))
print("--------------------------------")
print(normalizer2.transform(X))
```

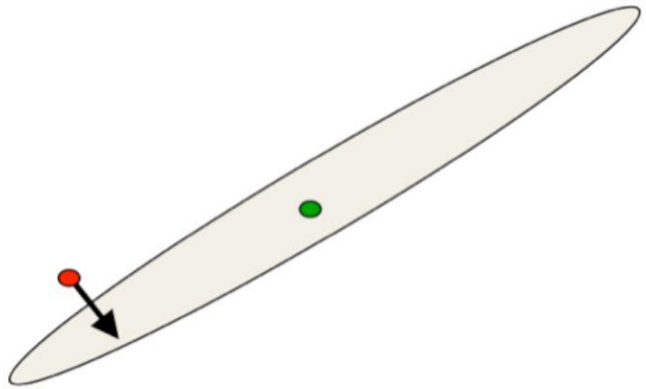
```
[[ 0.5 -0.5  1. ]
 [ 1.   0.   0. ]
 [ 0.   1.  -1. ]]
--------------------------------
[[ 0.40824829 -0.40824829  0.81649658]
 [ 1.          0.          0.        ]
 [ 0.          0.70710678 -0.70710678]]
```

标准化、区间缩放法(归一化)、正则化

- 标准化的目的是为了降低不同特征的不同范围的取值对于模型训练的影响；比如对于同一个特征，不同的样本的取值可能会相差的非常大，那么这个时候一些异常小或者异常大的数据可能会误导模型的正确率；另外如果数据在不同特征上的取值范围相差很大，那么也有可能导致最终训练出来的模型偏向于取值范围大的特征，特别是在使用梯度下降求解的算法中；通过改变数据的分布特征，具有以下两个好处：1. 提高迭代求解的收敛速度；2. 提高迭代求解的精度。







标准化、区间缩放法(归一化)、正则化

- 归一化对于不同特征维度的伸缩变换的主要目的是为了使得不同维度度量之间特征具有可比性，同时不改变原始数据的分布(相同特性的特征转换后，还是具有相同特性)(不改变的意思是：多个特征之间的关系不改变)。和标准化一样，也属于一种无量纲化的操作方式。
- 正则化则是通过范数规则来约束特征属性，通过正则化我们可以降低数据训练处来的模型的过拟合可能，和之前在机器学习中所讲述的L1、L2正则的效果一样。
- **备注**：广义上来讲，标准化、区间缩放法、正则化都是具有类似的功能。在有一些书籍上，将标准化、区间缩放法统称为标准化，把正则化称为归一化操作。
    - 如果面试有人问标准化和归一化的区别：标准化会改变数据的分布情况，归一化不会，标准化的主要作用是提高迭代速度，降低不同维度之间影响权重不一致的问题。而归一化的主要目的是为了解决过拟合问题的。

数据多项式扩充变换

- 多项式数据变换主要是指基于输入的特征数据按照既定的多项式规则构建更多的输出特征属性，比如输入特征属性为[a,b]，当设置degree为2的时候，那么输出的多项式特征为[1, a, b, a^2, ab, b^2]

class sklearn.preprocessing. **PolynomialFeatures** (*degree=2, interaction_only=False, include_bias=True*) ¶ [source]

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

```
X = np.arange(6).reshape(3, 2)
X
```

```
array([[0, 1],
       [2, 3],
       [4, 5]])
```

```
poly1 = PolynomialFeatures(2)
poly1.fit(X)
print(poly1)
print(poly1.transform(X))
```

```
PolynomialFeatures(degree=2, include_bias=True, interaction_only=False)
[[ 1.  0.  1.  0.  0.  1.]
 [ 1.  2.  3.  4.  6.  9.]
 [ 1.  4.  5. 16. 20. 25.]]
```

```
poly2 = PolynomialFeatures(interaction_only=True)
poly2.fit(X)
print(poly2)
print(poly2.transform(X))
```

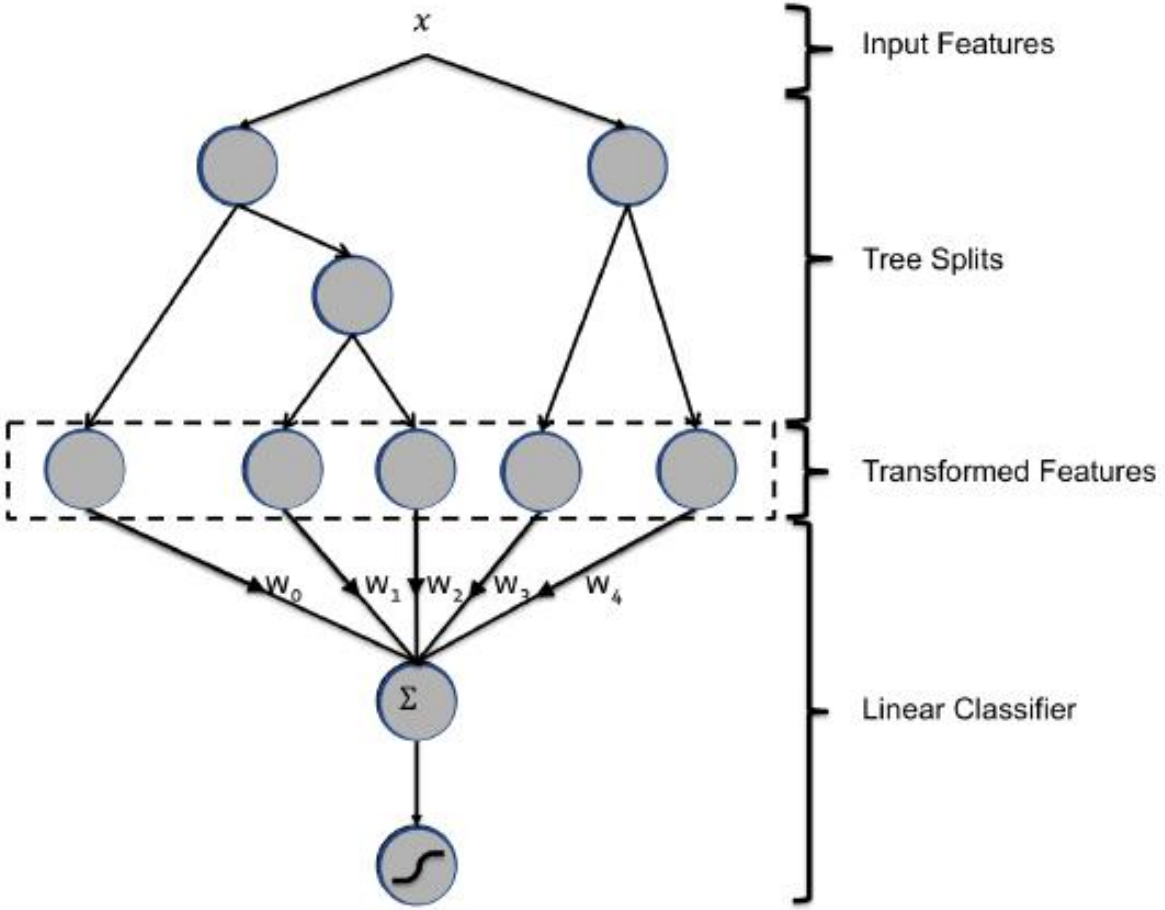
```
PolynomialFeatures(degree=2, include_bias=True, interaction_only=True)
[[ 1.  0.  1.  0.]
 [ 1.  2.  3.  6.]
 [ 1.  4.  5. 20.]]
```

```
poly3 = PolynomialFeatures(include_bias=False)
poly3.fit(X)
print(poly3)
print(poly3.transform(X))
```

```
PolynomialFeatures(degree=2, include_bias=False, interaction_only=False)
[[ 0.  1.  0.  0.  1.]
 [ 2.  3.  4.  6.  9.]
 [ 4.  5. 16. 20. 25.]]
```

GBDT/RF+LR

- 认为每个样本在决策树落在决策树的每个叶子上就表示属于一个类别，那么我们可以进行基于GBDT或者随机森林的维度扩展，经常我们会将其应用在GBDT将数据进行维度扩充，然后使用LR进行数据预测，这也是我们进行所说的GBDT+LR做预测。



特征选择

- 当做完特征转换后，实际上可能会存在很多的特征属性，比如：多项式扩展转换、文本数据转换等等，但是太多的特征属性的存在可能会导致模型构建效率降低，同时模型的效果有可能会变的不好，那么这个时候就需要从这些特征属性中选择出影响最大的特征属性作为最后构建模型的特征属性列表。
- 在选择模型的过程中，通常从两方面来选择特征：
  - 特征是否发散：如果一个特征不发散，比如方差等于0，也就是说这样的特征对于样本的区分没有什么作用。
  - 特征与目标的相关性：如果与目标相关性比较高，应当优先选择。

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 包装法-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]

```
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.svm import SVR
```

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

```
estimator = SVR(kernel='linear')
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  True  True False]
2
[3 1 1 2]
[[2 0]
 [1 4]
 [1 1]]
```

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, prefit=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]]
```

特征选择 嵌入法-基于树模型的特征选择法

- 树模型中GBDT在构建的过程会对特征属性进行权重的给定，所以GBDT也可以应用在基模型中进行特征选择。

class sklearn.feature_selection. **SelectFromModel** (*estimator, threshold=None, prefit=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
from sklearn.ensemble import GradientBoostingClassifier
```

```
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 = GradientBoostingClassifier()
sfm = SelectFromModel(estimator)
sfm.fit(X2, Y2)
print(sfm.transform(X2))
```

```
[[ 5.1  0.2]
 [ 4.9  0.2]
 [-6.2  2.3]
 [-5.9  1.8]]
```

特征选取/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。**
- 除了使用PCA和LDA降维外，还可以使用主题模型来达到降维的效果。

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；
  - 高纬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

特征选取/降维-PCA

- 主成分析(PCA)：将高纬的特征向量合并称为低纬度的特征属性，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))
```

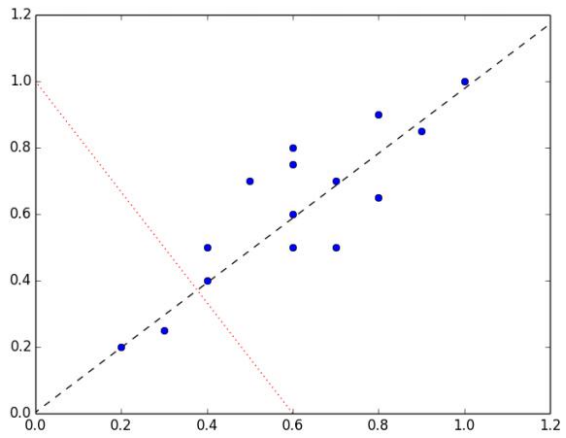
```
[-0.525  1.725  3.325  1.125  1.825 12.775]
[[ 0.02038178 -0.01698103 -0.01350052 -0.0149724   0.03184796 -0.99893718]
 [ 0.9024592   0.25030511 -0.31422084 -0.15092666 -0.03185873  0.01965141]
 [-0.08872116 -0.06952185 -0.06858116 -0.3074396  -0.94204108 -0.02512755]
 [-0.22421522  0.94883808  0.17610905 -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]]
```

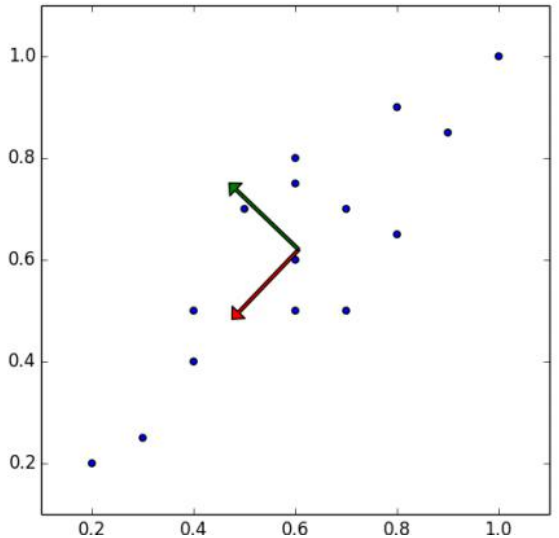
PCA原理

- 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”。





PCA原理

- 假设X是已经**中心化（z-score）**过的数据矩阵，每列一个样本（每行一个特征）；样本点x_i在新空间中的超平面上的投影是：W^Tx_i；若所有样本点的投影能够尽可能的分开，则表示投影之后的点在各个维度上的方差应该最大化，那么投影样本点的各个维度方差和可以表示为：$\frac{1}{n}\sum_i W^T x_i x_i^T W$；从而我们可以得到PCA的最优目标函数是：

$$\max_W tr\left(W^T XX^T W\right)$$

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

PCA原理

- 在PCA的目标函数基础上，带入拉格朗日求解最终，可以得到最终的拉格朗日函数函数为：

$$L = W^T XX^T W + \lambda\left(I - W^T W\right)$$

- 对拉格朗日函数求偏导数0：

$$\frac{\partial L}{\partial W} = 2XX^T W - 2\lambda W$$

$$\xrightarrow{令\frac{\partial L}{\partial w}=0} XX^T W = \lambda W$$

$$W^T XX^T W = W^T \lambda W = \lambda$$

- 可以发现如果，此时将XX^T看成一个整体A，那么求解W的过程恰好就是求解矩阵A的特征向量的过程，所以我们可以认为PCA的计算其实就是对进行去中心化后的数据的协方差矩阵求解特征值和特征向量。

PCA的执行过程

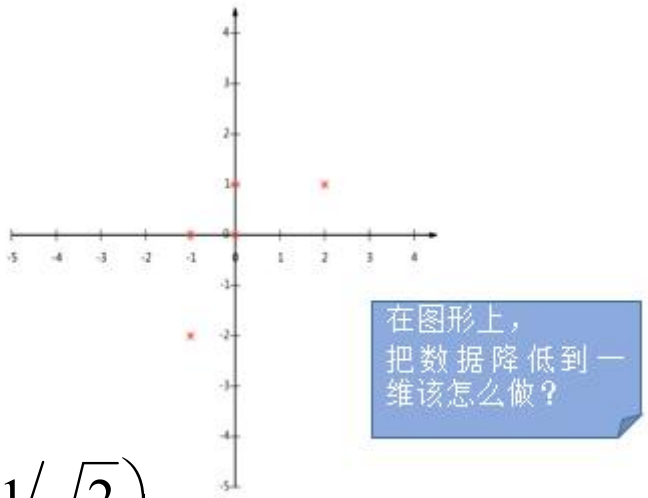
- 输入：样本集X={x1,x2,...,xn}；每个样本有m维特征，X是一个m行n列的矩阵
- 步骤：
  - 数据中心化：对X中的每一行(即一个特征属性)进行零均值化，即减去这一行的均值
  - 求出数据中心化后矩阵X的协方差矩阵(即特征与特征之间的协方差构成的矩阵)
  - 求解协方差矩阵的特征值和特征向量
  - 将特征向量按照特征值从大到小按列进行排列称为矩阵，获取最前面的k列数据形成矩阵W
  - 利用矩阵W和样本集X进行矩阵的乘法得到降低到k维的最终数据矩阵

PCA案例

假如有五条记录，用矩阵表示

为了后续处理方便，把每个字段都减去字段的均值

$$\begin{pmatrix} 1 & 1 & 2 & 4 & 2 \\ 1 & 3 & 3 & 4 & 4 \end{pmatrix} \xrightarrow{\text{减均值}} \begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix} \xrightarrow{\text{画图}}$$



$$X = \begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix}$$

$$\begin{cases} \lambda_1 = 0.5 \\ \lambda_2 = 2.5 \end{cases} \Rightarrow \begin{cases} w_1 = \begin{pmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \\ w_2 = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \end{cases}$$

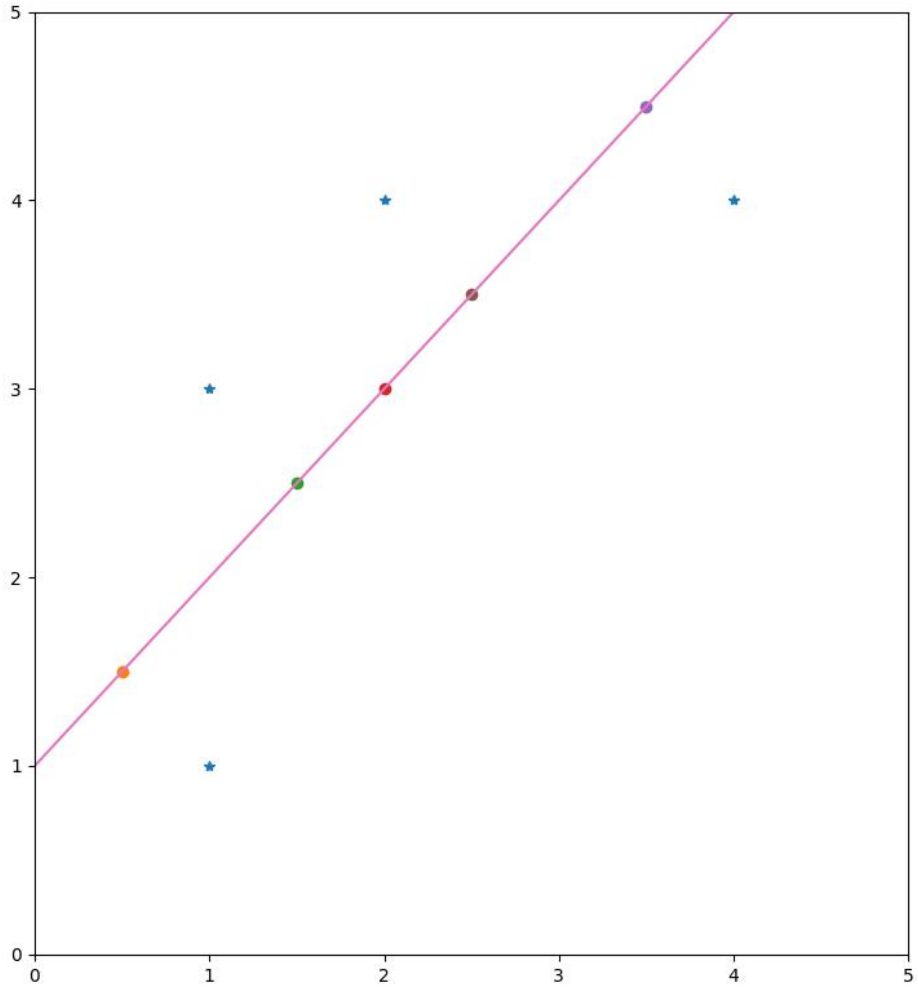
$$C = \frac{1}{m-1} XX^T = \begin{pmatrix} 1.5 & 1 \\ 1 & 1.5 \end{pmatrix}$$

$$X' = W^T X = \left(-3/\sqrt{2}, -1/\sqrt{2}, 0, 3/\sqrt{2}, 1/\sqrt{2},\right)$$

PCA案例

$$\begin{pmatrix} 1 & 1 & 2 & 4 & 2 \\ 1 & 3 & 3 & 4 & 4 \end{pmatrix} \xrightarrow{\text{减均值}} \begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix}$$

$$\left(-3/\sqrt{2}, -1/\sqrt{2}, 0, 3/\sqrt{2}, 1/\sqrt{2},\right)$$



PCA降维的SVD求解方式

- PCA的求解相当于是求解XX^T的特征向量和特征值的求解。

$$XX^TW = \lambda W$$

- 而且此时恰好XX^T是对角矩阵，所以我们可以将其进行特征分解：

$$XX^T = W\Lambda W^T$$

- 另外对矩阵X进行SVD矩阵分解，那么可以得到下列式子：

$$X = D\Sigma V^T \Rightarrow XX^T = D\Sigma V^T\left(D\Sigma V^T\right)^T = D\Sigma^2 D^T \Rightarrow W = D$$

$$\Downarrow$$

$$X' = W^T X = D^T D\Sigma V^T = \Sigma V^T$$

特征选取/降维-LDA

- 线性判断分析(LDA)：LDA是一种基于分类模型进行特征属性合并的操作，是一种有监督的降维方法。

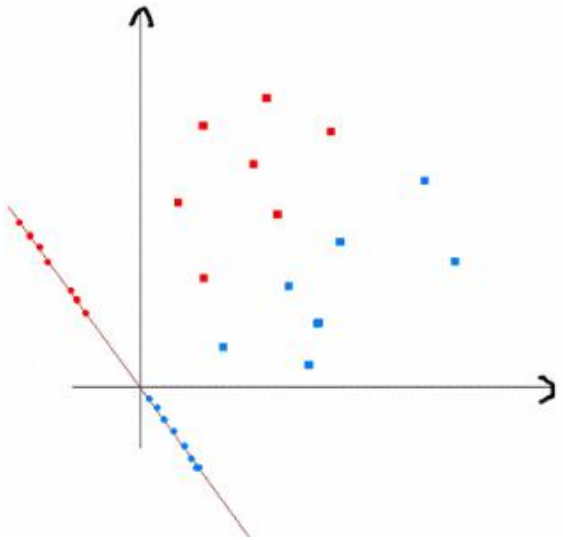
class sklearn.discriminant_analysis. **LinearDiscriminantAnalysis** (*solver='svd', shrinkage=None, priors=None, n_components=None, store_covariance=False, tol=0.0001*) [source]

```
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])
clf = LinearDiscriminantAnalysis(n_components=2)
clf.fit(X, y)
print(clf.transform(X))
```

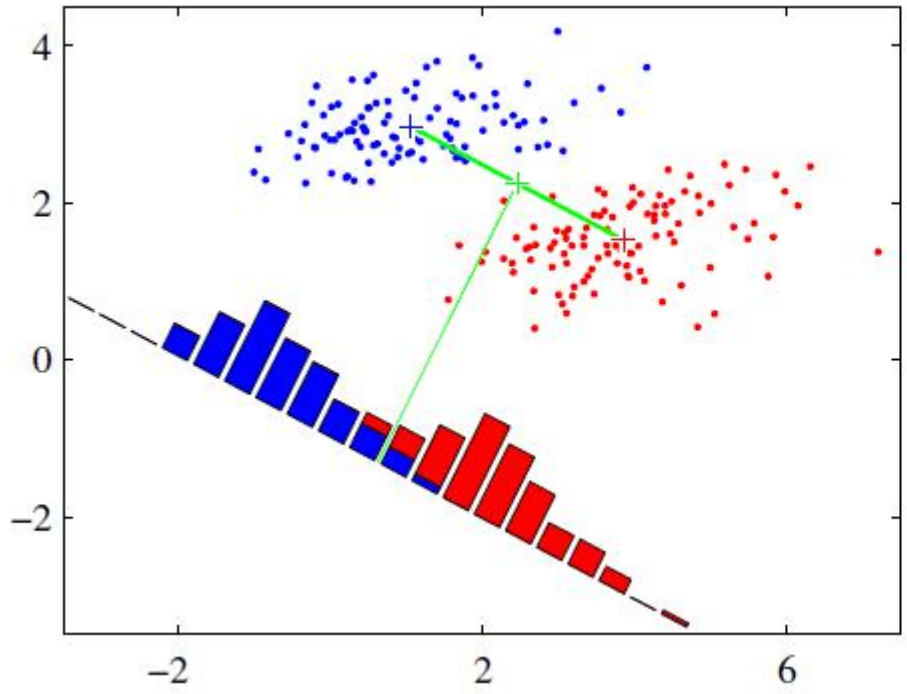
```
[[-3.2688434  -0.38911349]
 [-1.25507558 -1.78088569]
 [ 5.26064254 -0.49688862]
 [ 6.34385833  1.16134391]
 [-4.05800618  3.58297801]
 [-3.02257571 -2.07743411]]
```

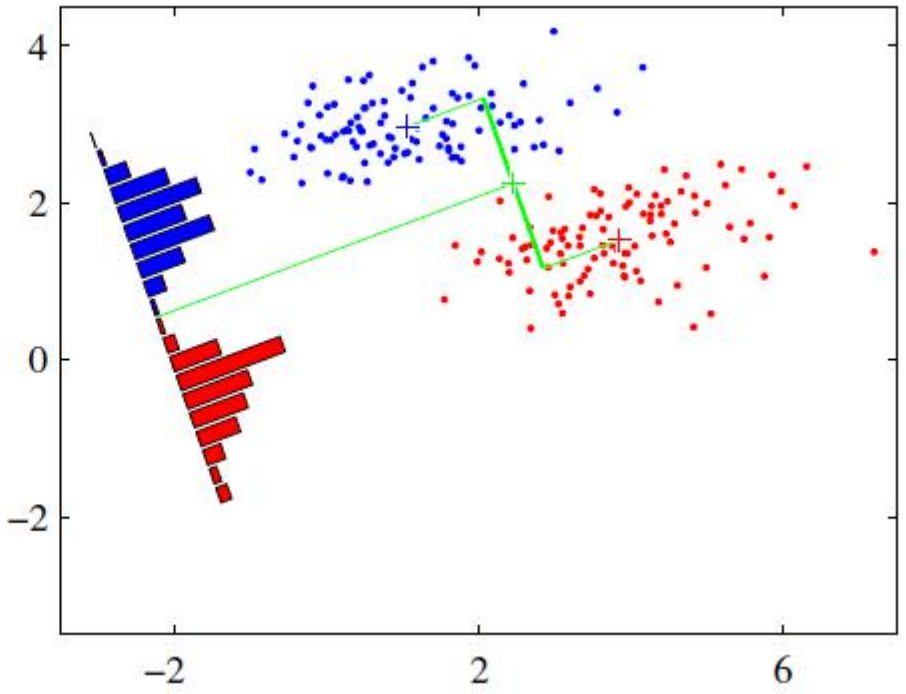
LDA原理

- 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原理

- 假定转换为w，那么线性转换函数为：$x' = w^T x$ ；并且转换后的数据是一维的
- 考虑二元分类的情况，认为转换后的值大于某个阈值，属于某个类别，小于等于某个阈值，属于另外一个类别，使用类别样本的中心点来表示类别信息，那么这个时候其实就相当于让这两个中心的距离最远：

$$\mu_j = \frac{1}{N_j}\sum_{x \in X_j} x \qquad \mu_j' = \frac{1}{N_j}\sum_{x \in X_j} x' = \frac{1}{N_j}\sum_{x \in X_j} w^T x = w^T \mu_j$$

$$J = \left|\mu_1' - \mu_2'\right| = w^T \left|\mu_1 - \mu_2\right|$$

LDA原理

- 同时又要求划分之后同个类别中的样本数据尽可能的接近，也就是同类别的投影点的协方差要尽可能的小。

$$\Sigma_j = \sum_{x \in X_j} (x - \mu_j)(x - \mu_j)^T \qquad \Sigma_j' = \sum_{x \in X_j} (x' - \mu_j')(x' - \mu_j')^T = w^T \Sigma_j w$$

$$\Sigma = \Sigma_1 + \Sigma_2$$

- 结合着两者，那么我们最终的目标函数就是：

$$\max_w J(w) = \frac{\left\| w^T \mu_1 - w^T \mu_2 \right\|_2^2}{w^T \Sigma_1 w + w^T \Sigma_2 w} = \frac{w^T (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T w}{w^T (\Sigma_1 + \Sigma_2) w}$$

LDA原理

- 对目标函数进行转换（A、B为方阵，A为正定矩阵）：

$$\max_w J(w)=\frac{\left\|w^T\mu_1-w^T\mu_2\right\|_2^2}{w^T\Sigma_1 w+w^T\Sigma_2 w}=\frac{w^T(\mu_1-\mu_2)(\mu_1-\mu_2)^T w}{w^T(\Sigma_1+\Sigma_2)w}$$

$$\xrightarrow{令A=\Sigma_0+\Sigma_1,B=(\mu_1-\mu_2)(\mu_1-\mu_2)^T} \max_w J(w)=\frac{w^T B w}{w^T A w}$$

$$\xrightarrow{令w'=A^{\frac{-1}{2}}w} \max_w J(w)=\frac{w'^T A^{\frac{-1}{2}} B A^{\frac{-1}{2}} w'}{w'^T w'}$$

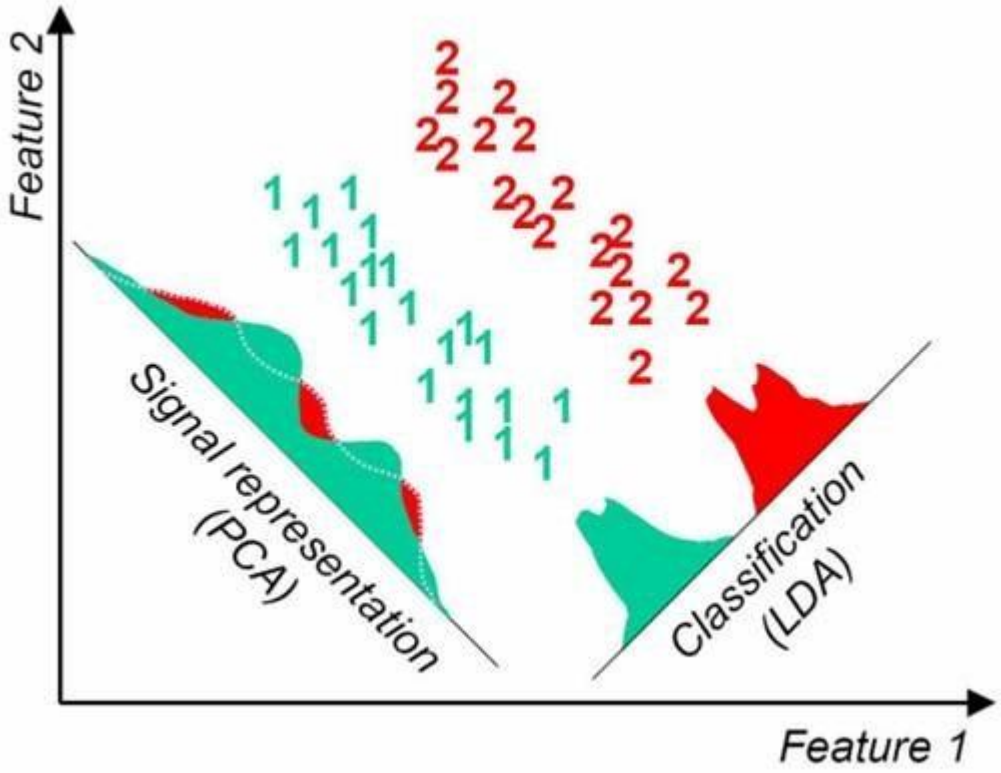
$$\xrightarrow{因w'^T w'=1} \max_w J(w)=w'^T A^{\frac{-1}{2}} B A^{\frac{-1}{2}} w'$$

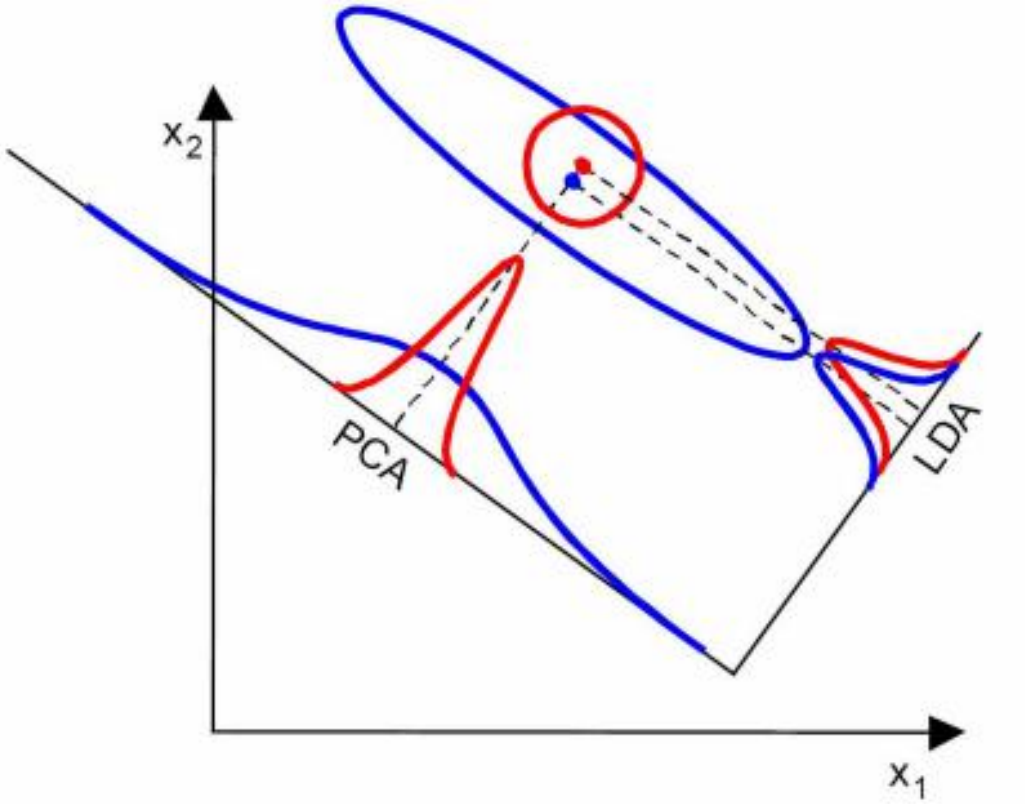
- 该式子和PCA降维中的优化函数一模一样，所以直接对中间的矩阵进行矩阵分解即可。

PCA和LDA

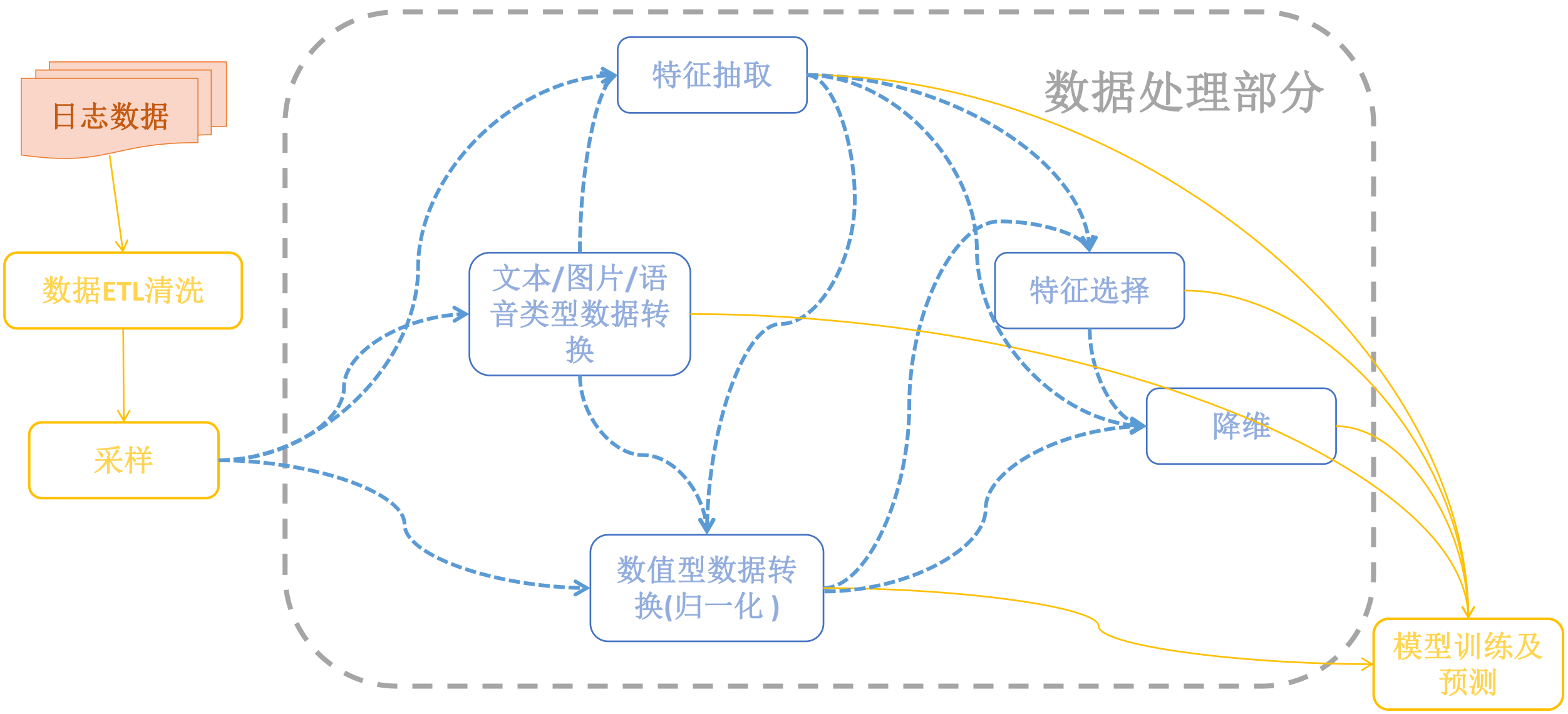
- 相同点：
  - 两者均可以对数据完成降维操作
  - 两者在降维时候均使用矩阵分解的思想
  - 两者都假设数据符合高斯分布
- 不同点：
  - LDA是监督降维算法，PCA是无监督降维算法
  - LDA降维最多降到类别数目k-1的维数，而PCA降维最多降到min(样本数目,特征属性数目)的维度数目
  - LDA除了降维外，还可以应用于分类
  - LDA选择的是分类性能最好的投影，而PCA选择样本点投影具有最大方差的方向

PCA和LDA


Feature 2
Feature 1
Signal representation
(PCA)
Classification
(LDA)

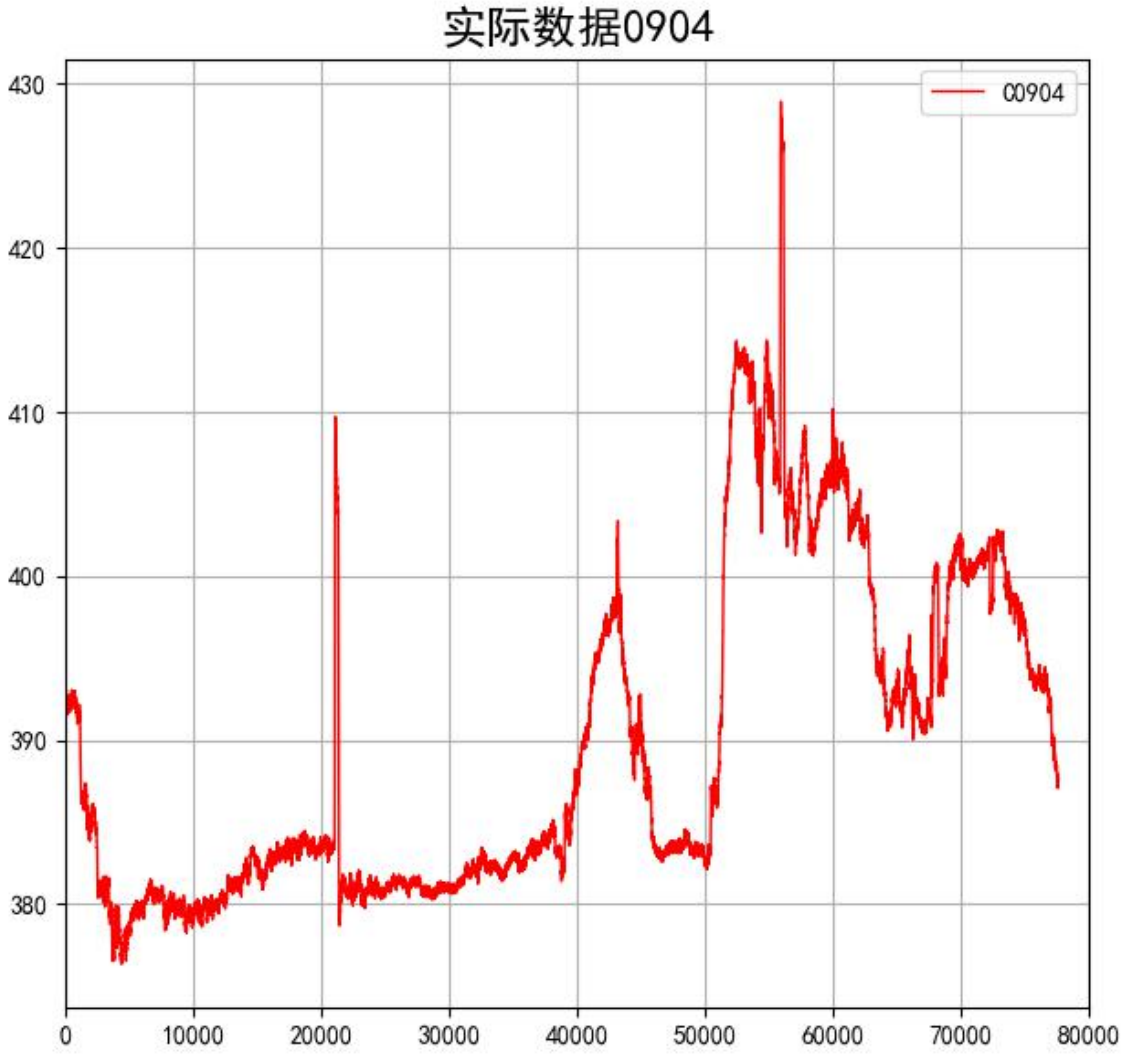

x2
x1
PCA
LDA

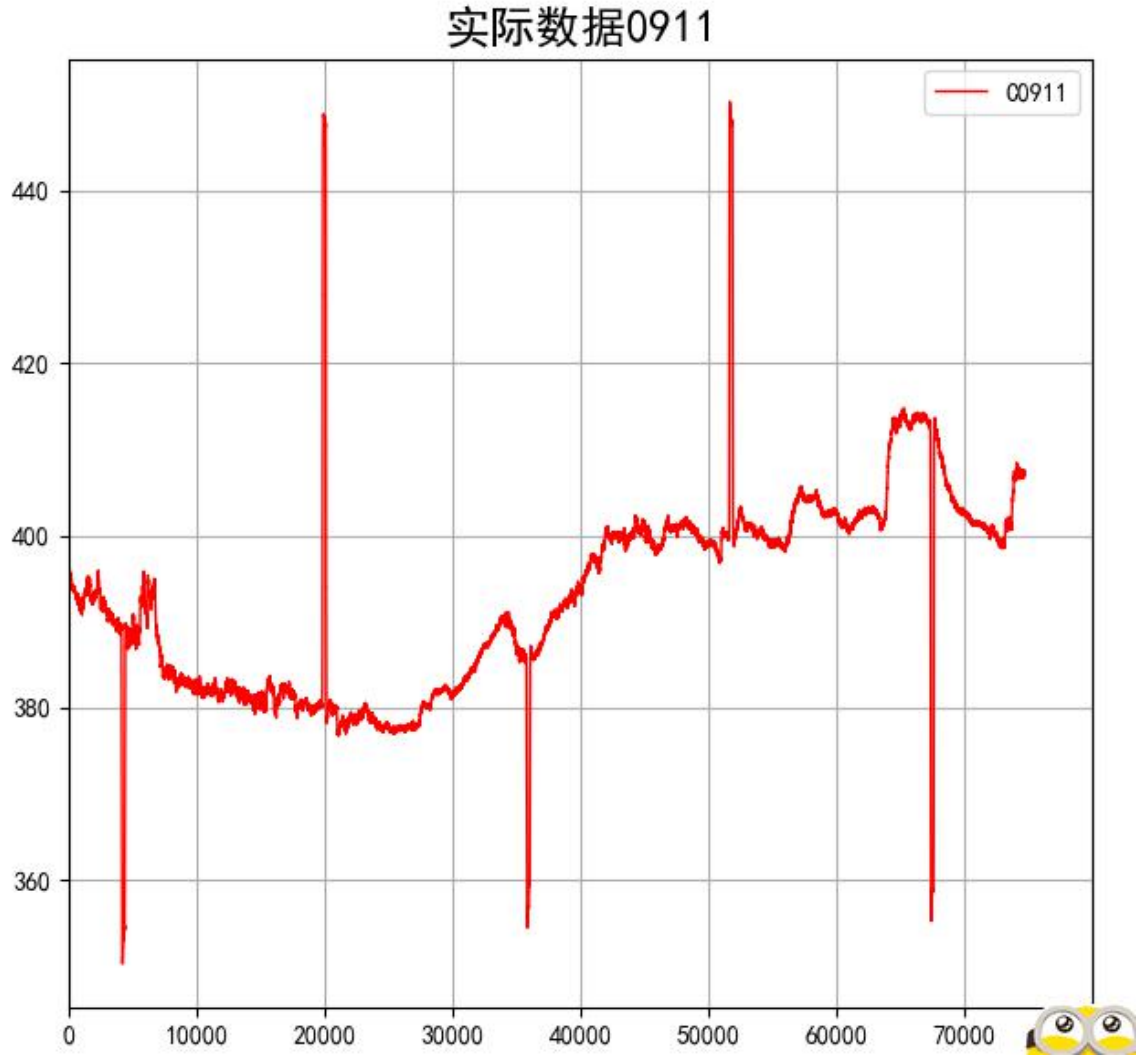
数据清洗常见流程


日志数据
数据ETL清洗
采样
特征抽取
文本/图片/语音类型数据转换
数值型数据转换(归一化)
特征选择
降维
数据处理部分
模型训练及预测

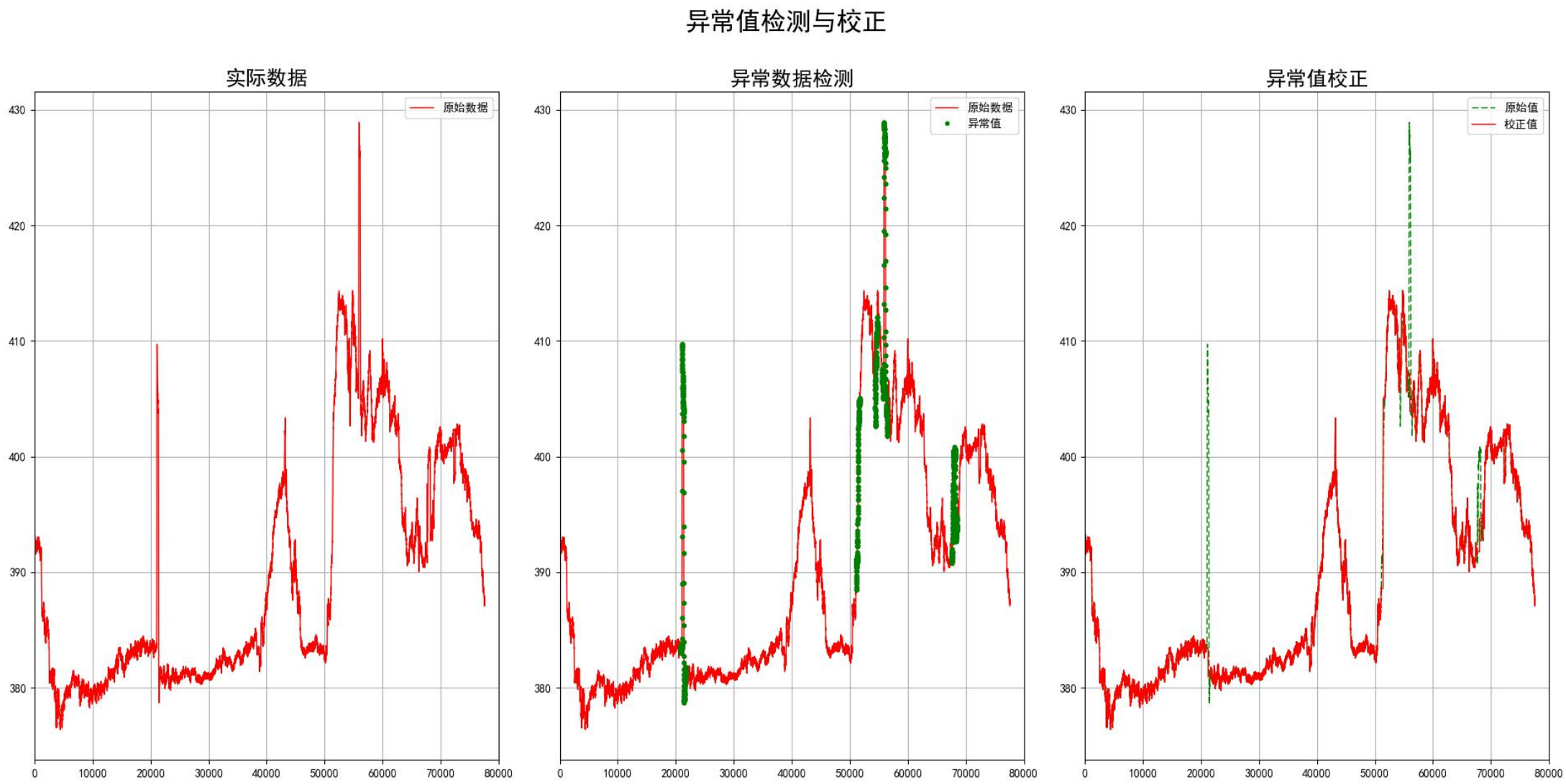
异常数据处理

如何找到下图中的异常值

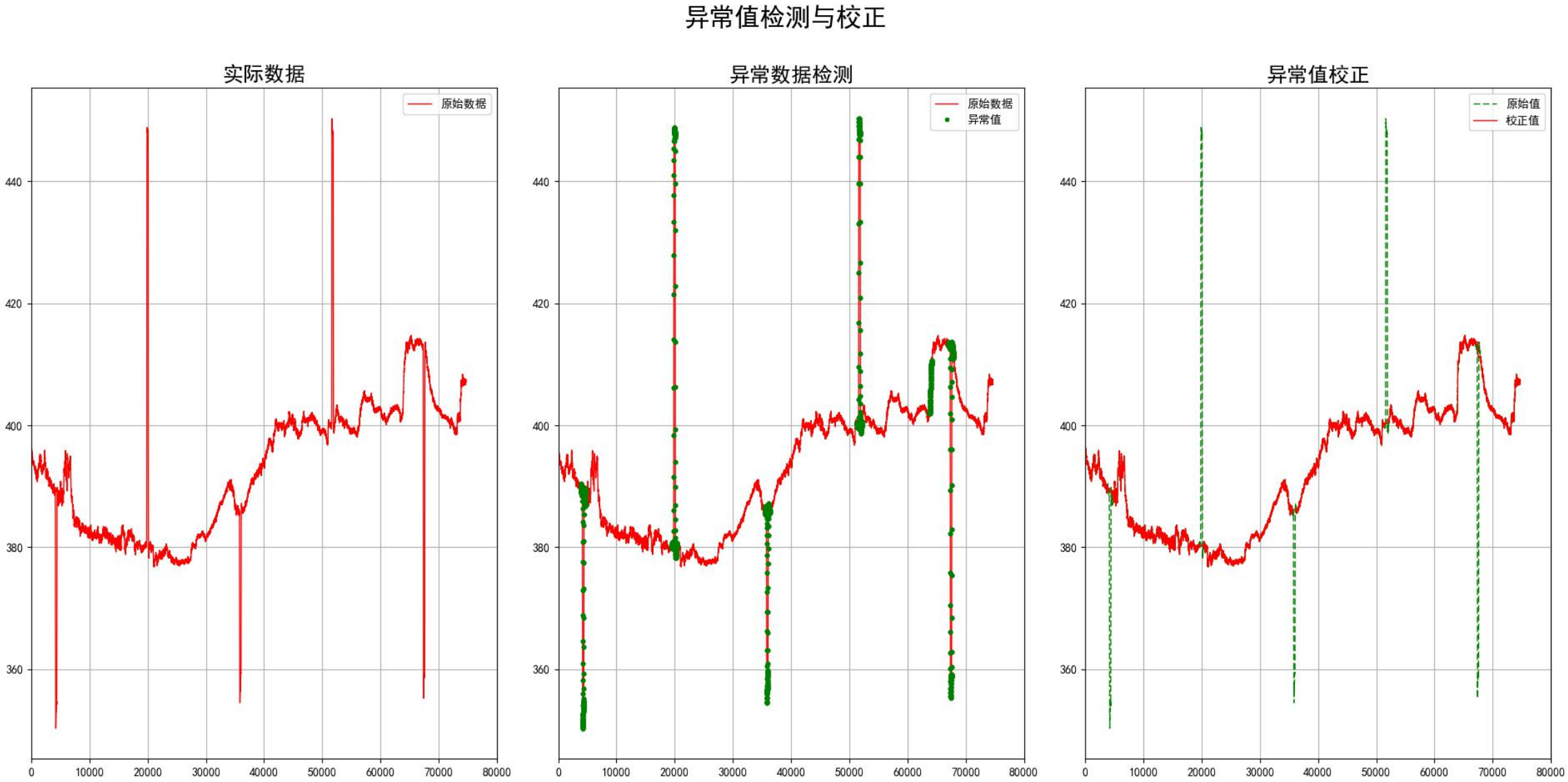




异常数据处理



异常数据处理


异常值检测与校正
实际数据
原始数据
异常数据检测
原始数据
异常值
异常值校正
原始值
校正值
440
420
400
380
360
0
10000
20000
30000
40000
50000
60000
70000
80000


北风网
IBEIFENG.COM