法律声明

- 课程详情请咨询
  - 微信公众号：北风教育
  - 官方网址：*http://www.ibeifeng.com/*

AI实战工程师

数据清洗和特征选择

主讲人：Gerry

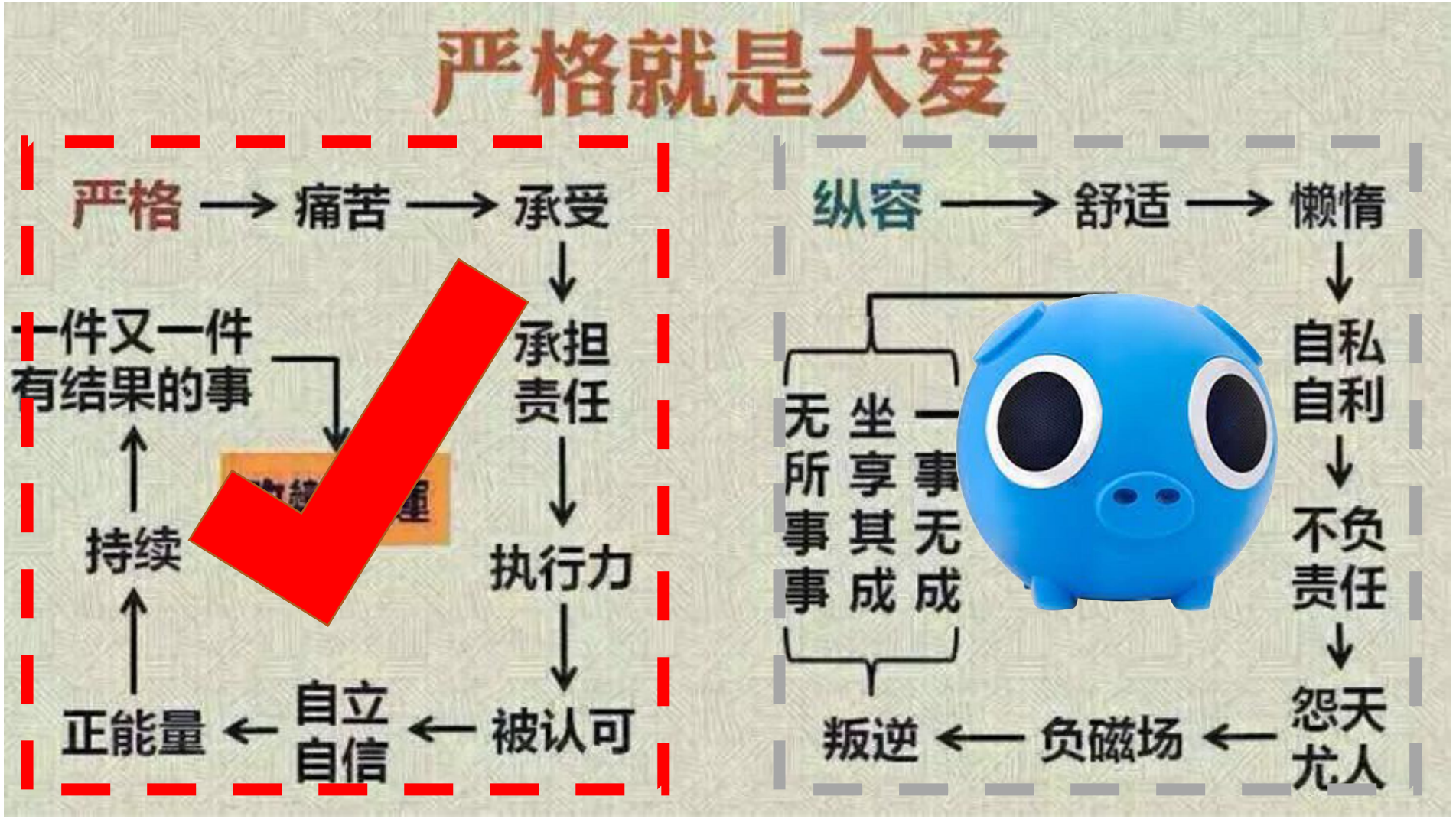
上海育创网络科技有限公司

北风网 IBEIFENG.COM

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

严格是大爱


严格就是大爱
严格
痛苦
承受
承担责任
执行力
被认可
自立自信
正能量
持续
一件又一件有结果的事
纵容
舒适
懒惰
自私自利
不负责任
怨天尤人
负磁场
叛逆
一事无成
坐享其成
无所事事

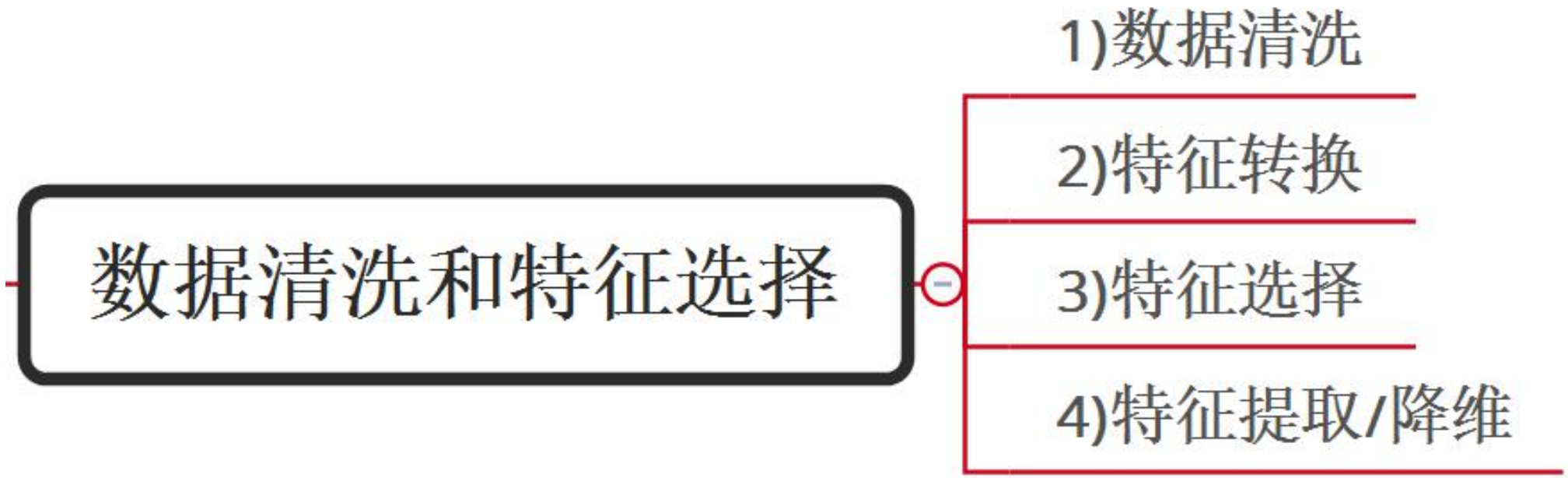
寄语

做别人不愿做的事，

做别人不敢做的事，

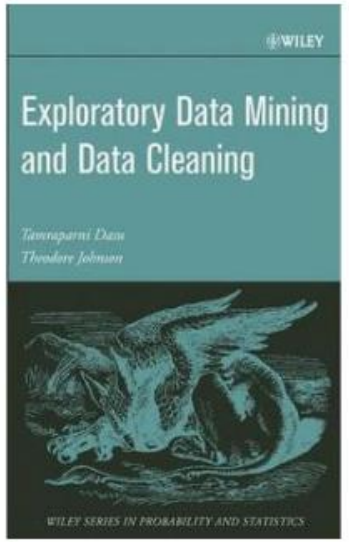
做别人做不到的事。

知识要点



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50%-80%左右的时间。



数据清洗过程


数据清洗
1.预处理
2. 去除/补全有缺失的数据
3. 去除/修改格式和内容错误的数据
4. 去除/修改逻辑错误的数据
5. 去除不需要的数据
6. 关联性验证

数据清洗--预处理

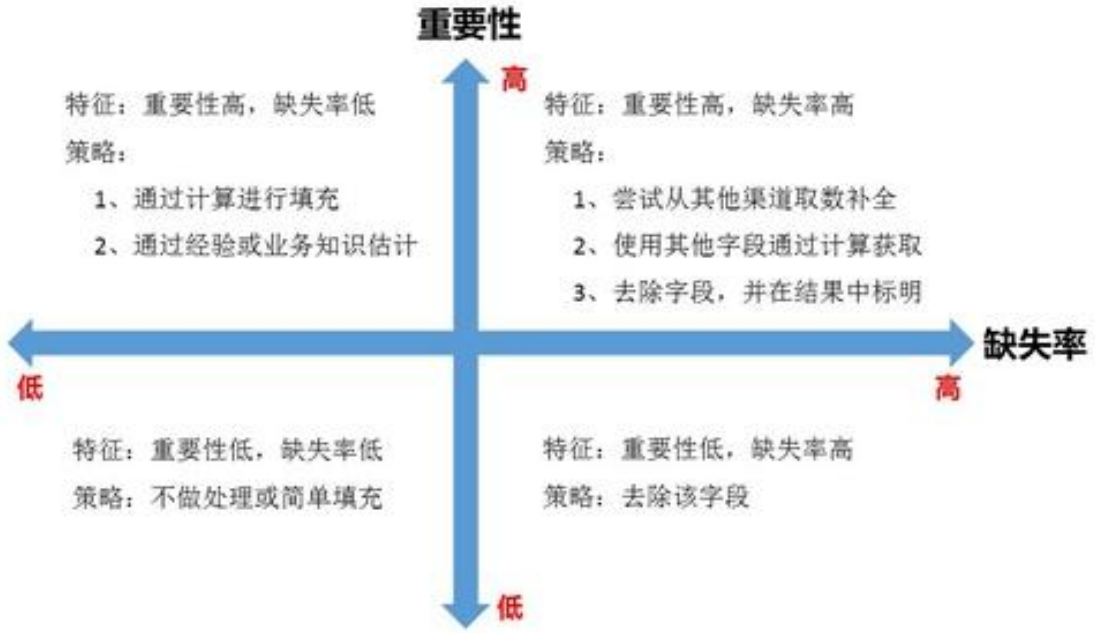
- 在数据预处理过程主要考虑两个方面，如下：
  - 选择数据处理工具：关系型数据库或者Python
  - 查看数据的元数据以及数据特征：一是查看元数据，包括字段解释、数据来源等一切可以描述数据的信息；另外是抽取一部分数据，通过人工查看的方式，对数据本身做一个比较直观的了解，并且初步发现一些问题，为之后的数据处理做准备。

数据清洗--缺省值清洗1

- 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。**

数据清洗--缺省值清洗2

- 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。



数据清洗--缺省值清洗3

- 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。

数据清洗--缺省值清洗4

- 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

数据清洗--缺省值清洗5

- 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。

数据清洗--格式内容清洗

- 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。

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

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

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: 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，表示该单词与常见，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，不计算词频，而是计算单词进行hash后的hash值的数量(有的模型中可能存在正则化操作)；
- 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
```

```
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()
```

```
[[ 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
[u'another', u'cd', u'example', u'hadoop', u'hbase', u'hive', u'is', u'sample', u'spark
转换另外的文档数据
[[ 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.]]
```

```
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.        ]]
[u'another', u'cd', u'example', u'hadoop', u'hbase', u'hive', u'is', u'sample', u'spark', u'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.        ]]
```

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)
```

```
[[ 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.]]
```

```
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]])
print enc.n_values_
```

```
[2 3 4]
```

```
enc.transform([[1, 0, 2]]).toarray()
```

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

```
dv = DictVectorizer(sparse=False)
D = [{'foo':1, 'bar':2}, {'foo':3, 'baz': 2}]
X = dv.fit_transform(D)
print X
print dv.get_feature_names()
print dv.transform({'foo':4, 'unseen':3})
```

```
[[ 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

- 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]

```
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

- 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.        ]]
```

归一化

- 归一化：和标准化不同，归一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='l1')
normalizer2 = Normalizer(norm='l2')
normalizer1.fit(X)
normalizer2.fit(X)
```

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

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

```
[[ 0.25 -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]]
```

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.]]
```

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，比如方差解决于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]
])
```

```
variance = VarianceThreshold(threshold=0.1)
print variance
variance.fit(X)
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相关系数的P值，然后获取大于阈值的特征属性。

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])
```

```
sk1 = SelectKBest(f_regression, k=2)
sk1.fit(X, Y)
print sk1
print sk1.scores_
print sk1.transform(X)
```

```
SelectKBest(k=2, score_func=<function f_regression at 0x000000000A8A4BA8>)
[  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 0x000000000A8A4B38>)
[ 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]

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

特征选择-基于惩罚项的特征选择法

- 在使用惩罚项的基模型，除了可以筛选出特征外，同时还可以进行降维操作

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,  3.4,  5.4,  2.3],
    [ 5.9,  3. ,  5.1,  1.8]
], dtype=np.float64)
Y2 = np.array([0, 0, 2, 2])
estimator = LogisticRegression(penalty='l2', C=0.1)
sfm = SelectFromModel(estimator)
sfm.fit(X2, Y2)
print sfm.transform(X2)
```

```
[[ 1.4]
 [ 1.4]
 [ 5.4]
 [ 5.1]]
```

特征选择-基于惩罚项的特征选择法

- 在使用惩罚项的基模型，除了可以筛选出特征外，同时还可以进行降维操作

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-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，LDA是为了让映射后的样本有最好的分类性能。

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],
    [ 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)
pca = PCA(n_components=2)
pca.fit(X2)
print pca.mean_
print pca.components_
print pca.transform(X2)
```

```
[-0.525  1.725  3.325  1.125]
[[-0.90319637 -0.24920124  0.31484024  0.15169269]
 [ 0.03919747 -0.80616444 -0.27357029 -0.52318424]]
[[-6.26919501 -0.19988789]
 [-5.96395512  0.19535484]
 [ 6.28736349 -0.33667759]
 [ 5.94578665  0.34121064]]
```

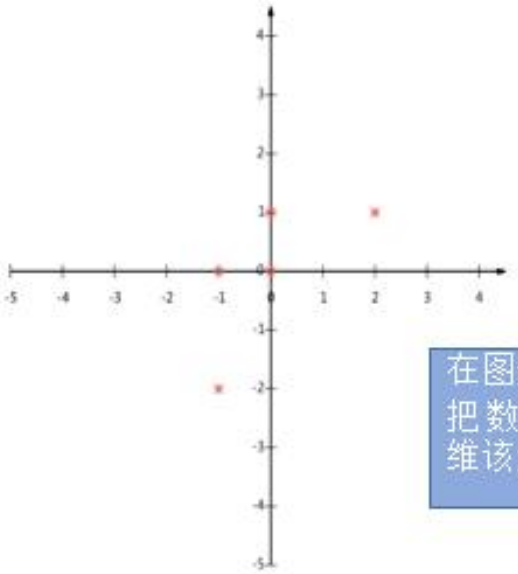
特征选取/降维-PCA

- 协方差矩阵及优化目标
- 二维到一维降维

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

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



在图形上，把数据降低到一维该怎么做？

方差最大化让投影分散

$$var(a) = \frac{1}{m}\sum_{i=1}^{m}(a_i - \mu)^2 = \frac{1}{m}\sum_{i=1}^{m} a_i^{\Sigma}$$

特征选取/降维-PCA

协方差矩阵及优化目标

多维情况——协方差矩阵

假如X,Y,Z三个维度，那么他们的协方差矩阵如下。C可以作为刻画不同维度（分量）之间相关性的一个评判量。若不同分量之间的相关性越小，则C的非对角线元素的值就越小，特别地，若不同分量彼此不想关，那么C就变成了一个对角阵。

$$C_{3\times 3} = \begin{pmatrix} cov(X,X) & cov(X,Y) & cov(X,Z) \\ cov(Y,X) & cov(Y,Y) & cov(Y,Z) \\ cov(Z,X) & cov(Z,Y) & cov(Z,Z) \end{pmatrix}$$

特征选取/降维-PCA

PCA数学过程

假设样本集$X = \{x_1, x_2, ..., x_m\}$,样本点$x_i$新空间中超平面上的投影是$W^T x_i$，欲投影分散，应该让投影后的方差最大化，投影后的方差为$\sum_i W^T x_i x_i^T W$ ，把优化目标数学化：

$$\max_W \ tr(W^T X X^T W) \ s.t. \ \ W^T W = I$$

对于这个式子使用拉格朗日乘子法

$$xx^T w = \lambda w$$

最后，对xx^T进行特征值分解，对特征值λ_i进行排序，去前n个特征值对应的特征向量就构成 $W = \{w_1, w_2, ..., w_m\}$

特征选取/降维-LDA

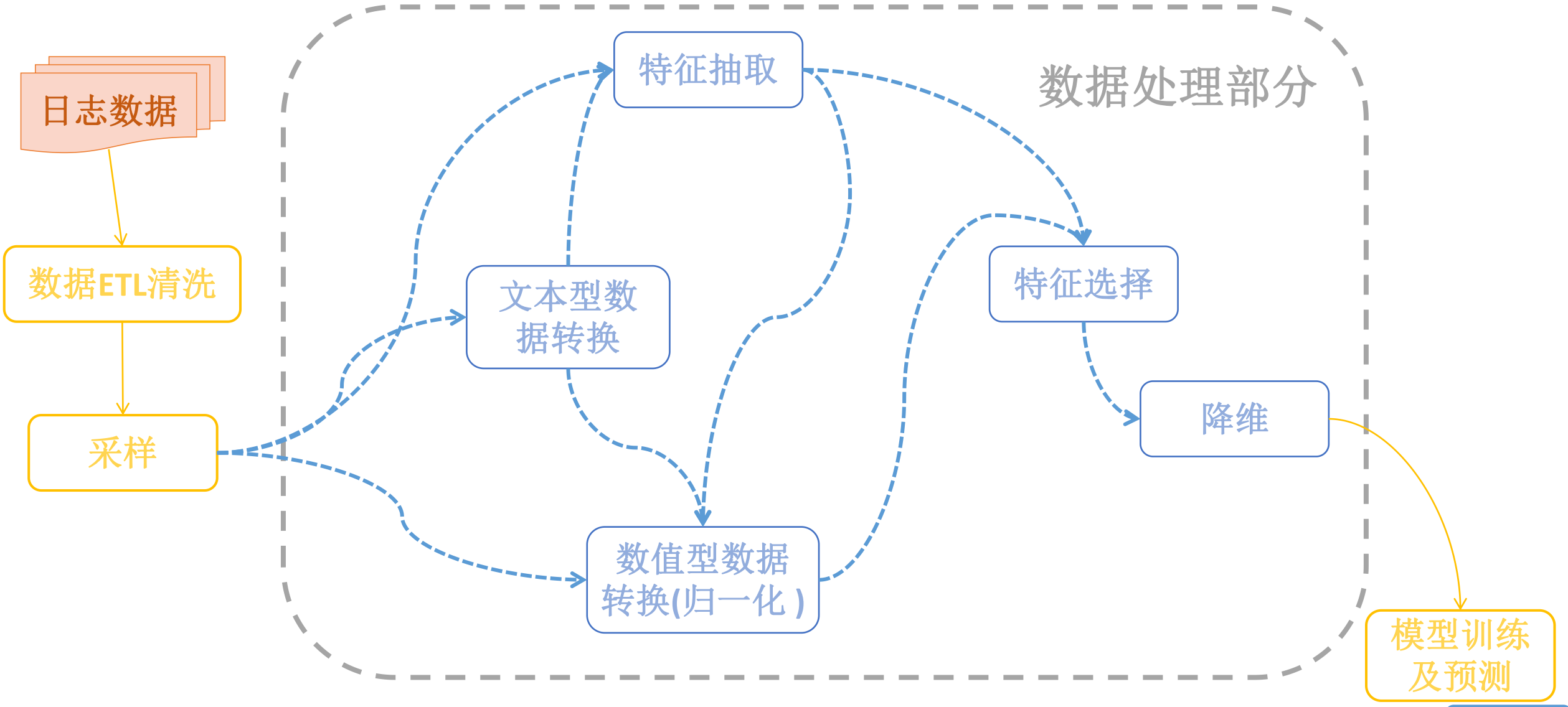
- 线性判断分析(LDA)：LDA是一种基于分类模型进行特征属性合并的操作，是一种有监督的降维方法。

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

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
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])
lda = LinearDiscriminantAnalysis(n_components=2)
lda.fit(X2, Y2)
print lda.transform(X2)
```

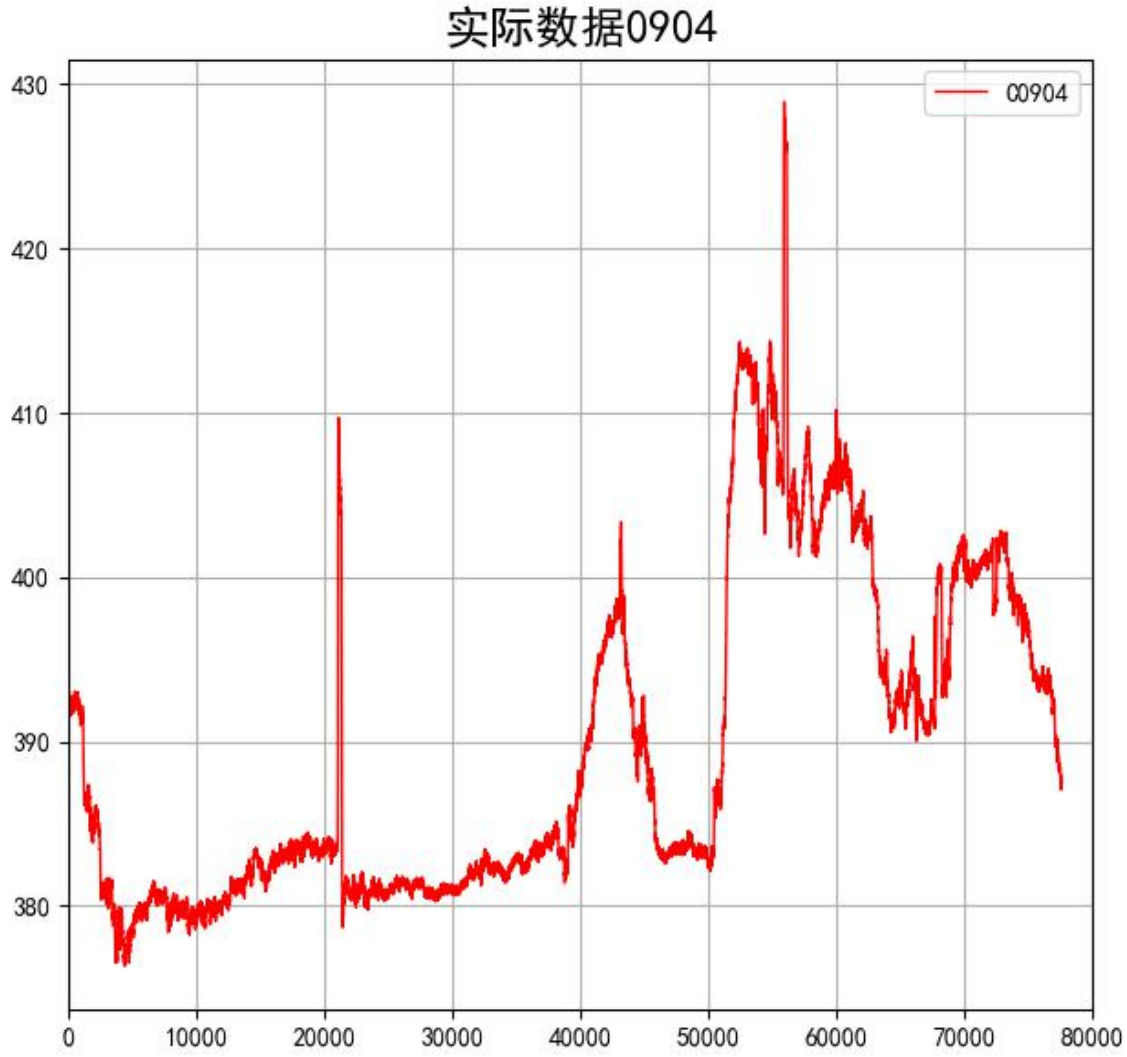
```
[[-27.18468036]
 [-25.44008732]
 [ 26.80136112]
 [ 25.82340656]]
```

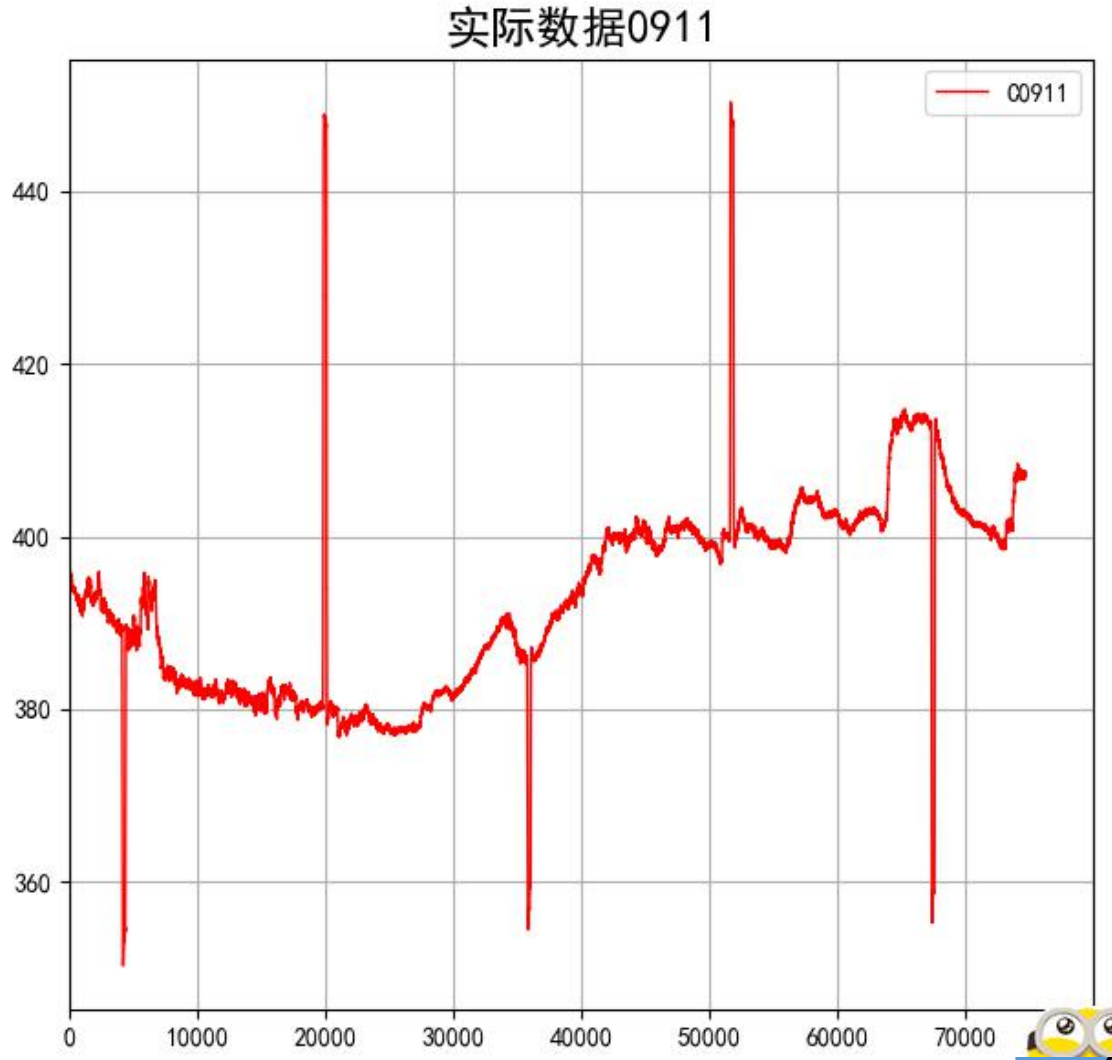
数据清洗常见流程


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

异常数据处理

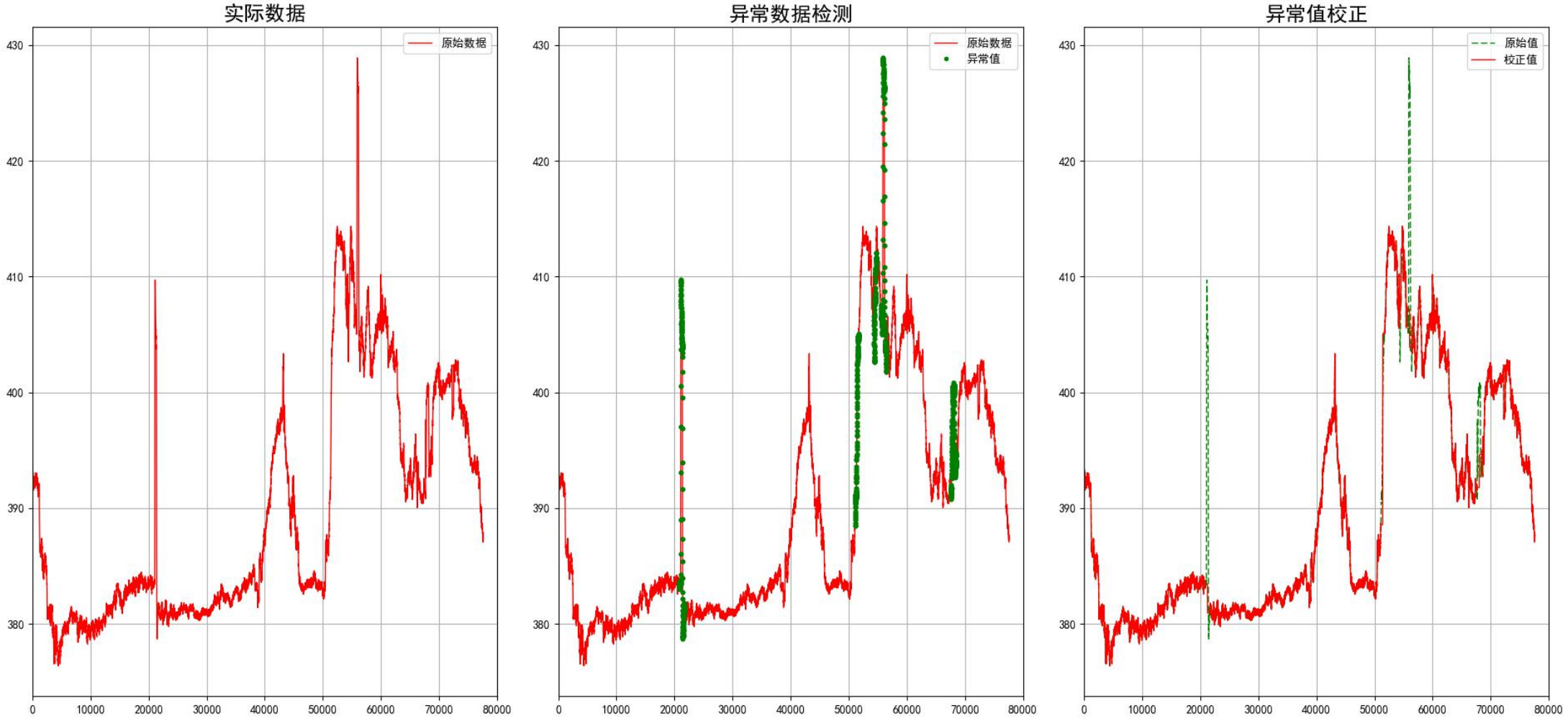
如何找到下图中的异常值



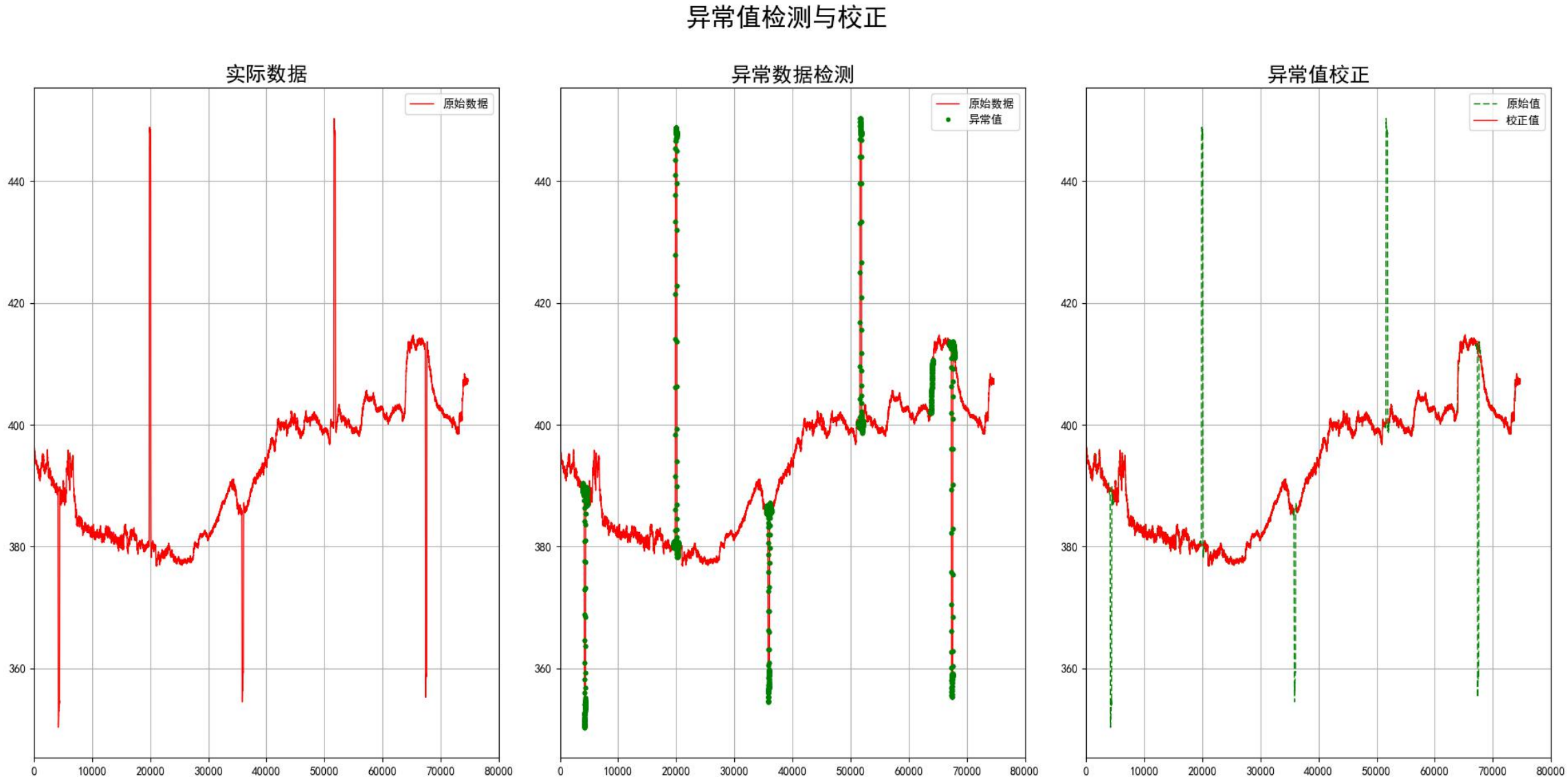


异常数据处理

异常值检测与校正



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

车辆数据预处理

- 每个样本具有7个特征，描述如下：
  - 购买价格：high、low、med、vhigh
  - 维护价格：high、low、med、vhigh
  - 车门数量：2、3、4、5more
  - 载人数目：2、4、more
  - 后备箱大小：big、med、small
  - 安全程度：high、low、med
  - 接受程度：acc、good、unacc、vgood

	A	B	C	D	E	F	G
1	buy	maintain	doors	persons	boot	safety	accept
2	vhigh	vhigh	2	2	small	low	unacc
3	vhigh	vhigh	2	2	small	med	unacc
4	vhigh	vhigh	2	2	small	high	unacc
5	vhigh	vhigh	2	2	med	low	unacc
6	vhigh	vhigh	2	2	med	med	unacc
7	vhigh	vhigh	2	2	med	high	unacc
8	vhigh	vhigh	2	2	big	low	unacc
9	vhigh	vhigh	2	2	big	med	unacc
10	vhigh	vhigh	2	2	big	high	unacc
11	vhigh	vhigh	2	4	small	low	unacc
12	vhigh	vhigh	2	4	small	med	unacc
13	vhigh	vhigh	2	4	small	high	unacc
14	vhigh	vhigh	2	4	med	low	unacc
15	vhigh	vhigh	2	4	med	med	unacc
16	vhigh	vhigh	2	4	med	high	unacc
17	vhigh	vhigh	2	4	big	low	unacc
18	vhigh	vhigh	2	4	big	med	unacc
19	vhigh	vhigh	2	4	big	high	unacc
20	vhigh	vhigh	2	more	small	low	unacc
21	vhigh	vhigh	2	more	small	med	unacc
22	vhigh	vhigh	2	more	small	high	unacc
23	vhigh	vhigh	2	more	med	low	unacc
24	vhigh	vhigh	2	more	med	med	unacc
25	vhigh	vhigh	2	more	med	high	unacc
26	vhigh	vhigh	2	more	big	low	unacc
27	vhigh	vhigh	2	more	big	med	unacc
28	vhigh	vhigh	2	more	big	high	unacc
29	vhigh	vhigh	3	2	small	low	unacc
30	vhigh	vhigh	3	2	small	med	unacc
31	vhigh	vhigh	3	2	small	high	unacc
32	vhigh	vhigh	3	2	med	low	unacc
33	vhigh	vhigh	3	2	med	med	unacc
34	vhigh	vhigh	3	2	med	high	unacc
35	vhigh	vhigh	3	2	big	low	unacc
36	vhigh	vhigh	3	2	big	med	unacc

车辆数据预处理

```
for i in range(7):
    print i, np.unique(data[i])
```

```
0 ['high' 'low' 'med' 'vhigh']
1 ['high' 'low' 'med' 'vhigh']
2 ['2' '3' '4' '5more']
3 ['2' '4' 'more']
4 ['big' 'med' 'small']
5 ['high' 'low' 'med']
6 ['acc' 'good' 'unacc' 'vgood']
```

```
### 字符串转换为序列id（数字）
X = data.apply(lambda x: pd.Categorical(x).codes)
X.head(5)
```

	0	1	2	3	4	5	6
0	3	3	0	0	2	1	2
1	3	3	0	0	2	2	2
2	3	3	0	0	2	0	2
3	3	3	0	0	1	1	2
4	3	3	0	0	1	2	2

```
### 原始数据
data.head(5)
```

	0	1	2	3	4	5	6
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

```
### 转换后数据
# print X.toarray()
pd.DataFrame(X.toarray()).head(5)
```

	0	1	2	3	4	5	6	7	8	9	...	15	16	17	18	19	20	21	22	23	24
0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	...	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0
2	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	...	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	...	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	...	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0

NLP自然语言处理

- 自然语言处理：即实现人机间自然语言通信，或实现自然语言理解和自然语言生成。
- 主要技术：汉字词法分析、句法分析、语义分析、文本生成、语言识别
- 应用场景：文本分类和聚类、信息检索和过滤、机器翻译
- Python中汉字分词包：jieba

NLP—中文分词

- 分词：来把连续的汉字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。
- 分词的常见方法
  - 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分词：python开发的中文分词模块
- 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

THANK YOU

上海育创网络科技有限公司