内容简介

基于对数据字段的检视，为解决业务问题提出数据改造方案。

1. 主要内容

- 通过本案例重要字段的描述性统计，对公司样本进行初步筛选，并生成最终的统计文档“short_summary.xlsx”；
- 对所有数值型字段的分布形状进行重新判断，发现与案例 1 中的区别；
- 根据字段检视和业务问题，提出一系列数据改造方案。

2. 学习目标

学完本节，能解决以下问题：

- 在数据检视中，本案例与上一案例之间的变化有哪些？

提出数据改造方案

由于本案例数据集的字段类型与案例一完全一致，仅个别字段在处理方法上有所不同。

故本案例与案例一的区别之处，在提出改造方案中红色警示标注。

1. 明显可以放弃的字段（放弃字段均标灰）

部分字段放弃依据表

英文名称	特征名称	放弃依据
company_code	公司代码	公司标识，不参与数据
company_name	公司名称	建模
trading_days	交易日期	
trading_price	交易平均价格	与本案例主业务问题
total_transaction_amount	交易总金额	无关联，用于后续案例
outway	公司去向	
financing_cnt	融资次数	融资后生成字段

49 个字段，舍弃明显的仍然为该 7 个字段，剩余 42 个字段。

2. 模型输出组变量生成

本案例的主业务问题是“**围绕新三板部分公司数据，挖掘公司融资金额背后的相关因素**”。

因此模型的输出变量应该是融资金额，刚好数据集中有融资金额字段“financing_amount_wan”，为连续变量，且在数据质量中已经对其进行了处理。

3. 文本型字段转换为数值型字段

线性回归模型也不能直接处理文本型字段，需要使用 onehot 函数进行特征提取，文本型字段转换为数值型字段，数据需要的改造与案例 1 一致，故不作介绍。

文本型字段转换为数值型字段依据表

英文名称	特征名称	取值类别	是否转换
transfer_mode	转让方式	2 种	是
layer	公司分层	2 种	

high_executive_edu	高管最高学历	4 种	
industry	第三级投资型行业	67 种	
province	省份	31 种	
accounting_firm	会计事务所	-	类别过多，不利于建模，放弃字段
local_accounting_firm	会计事务所	39 种	
broker	主办券商	-	
local_broker	主办券商	96 种	

注：放弃后剩余 35 个字段

4. 数据清理

（1）缺失/空白分析

在是否发生融资和融资金额异常的处理后，对剩余 3623 个样本进行描述性统计，发现仍然都存在缺失情况，各字段的处理方法与案例 1 一致，故不作介绍。

均值填充依据表

英文名称	特征名称	英文名称	特征名称
listing_days	挂牌时长	liability	负债总计
registered_capital	注册资本	net_asset_per_share	每股净资产
register_days	注册时长	earning_per_share	基本每股收益
customer_rt_1st	第一大客户占比	asset_liablity_rt	资产负债率
customer_rt_5all	五大客户占比	current_rt	流动比率
supplier_rt_1st	第一大供应商占比	gross_cash_flow	现金流量总额
supplier_rt_5all	五大供应商占比	asset_gr	总资产增长率
total_stock_equity	总股本	income_gr	营业收入增长率
business_income	营业收入	net_profit_gr	净利润增长率
gross_profit_rt	毛利率	ave_executive_age	高管平均年龄
net_profit_rt	净利润	ave_executive_edu	高管平均学历
roa	净资产收益率	holder_rt_1st	第一大股东占比
asset	资产总计	holder_rt_10all	前十大股东占比

众数填充表

英文名称	特征名称	英文名称	特征名称
transfer_mode	转让方式	board_num	董事会人数
layer	公司分层	supervisor_num	监事会人数
maker_num	做市商数量	executive_num	高管人数
employee_num	雇员数量	highest_executive_edu	高管最高学历

“0”值填充

英文名称	特征名称
contain_org_holder	存在机构股东

注：缺失值填充需完成：26 个均值填充+8 个众数填充+1 个“0”值填充=35 个字段

（2）异常值分析

同案一：

异常值人工修改依据表

英文名称	特征名称	合理范围	英文名称	特征名称	合理范围
customer_rt_5all	五大客户占比	[0,1]	roa	净资产收益率	[-1,1]
supplier_rt_5all	五大供应商占比	[0,1]	asset_gr	总资产增长率	[-1,5]
gross_profit_rt	毛利率	[-1,1]	income_gr	营业收入增长率	[-1,5]
net_profit_rt	净利润	[-1,1]	net_profit_gr	净利润增长率	[-10,10]

5. 线性回归对数据的要求

①量纲相同；②尽可能服从标准正态分布。通过“数据质量”整理得到下表：

数值型字段分布情况统计表

分布类型	英文名称	特征名称	分布类型	英文名称	特征名称
近似泊松分布（4 个）	maker_num	做市商数量		liability	负债总计
	employee_num	雇员数量		gross_cash_flow	现金流量总额
	board_num	董事会人数		current_rt	流动比率
	executive_num	高管人数	特殊分布	holder_rt_10all	前十大股东占比
近似正态分布（7 个）	customer_rt_5all	五大客户占比	其他分布（8 个）	supervisor_num	监事会人数
	supplier_rt_5all	五大供应商占比		gross_profit_rt	毛利率
	net_asset_per_share	每股净资产		net_profit_rt	净利润
	earning_per_share	基本每股收益		roa	净资产收益率
	asset_liablity_rt	资产负债率		asset_gr	总资产增长率
	ave_executive_age	高管平均年龄		income_gr	营业收入增长率
	holder_rt_1st	第一大股东占比		net_profit_gr	净利润增长率
近似对数正态分布（11 个）	listing_days	挂牌时长		ave_executive_edu	高管平均学历
	registered_capital	注册资本	输出变量	financing_amount_wan	融资金额（万）
	register_days	注册时长	放弃字段（5 个）（文本型）	financing_cnt	融资次数
	customer_rt_1st	第一大客户占比		total_transaction_amount	交易总金额
	supplier_rt_1st	第一大供应商占比		trading_days	交易日期
	total_stock_equity	总股本		trading_price	交易平均价格
	business_income	营业收入		highest_executive_edu	高管最高学历
	asset	资产总计	"0"填充	contain_org_holder	存在机构股东

注：与案例一相比“current_rt”发生变化，由其他分布→近似对数正态分布。

至此已完成数据预处理的改造方案。

具体处理详见“资料下载”中的“数据字段处理记录.xlsx”表格。

6. 数据分割

线性回归模型为有监督学习算法，需要对数据集进行分割。

本案例采用“直接下定义”和“k-fold 交叉验证”相结合的分割方法，具体内容可参考后续“数据分割”章节中的“3.分割方法”文档介绍。

7. 训练集中是否存在非平衡数据？

该问题只针对输出变量为类别型变量的情况，本案例以连续型变量“financing_amount_wan”作为模型的输出变量，故不考虑该问题。

数据检视总结

1、在数据检视中本案例与上一案例之间的变化

（1）根据业务问题的不同，删除了 1329 个异常样本

首先对于非发生融资的公司，其数据会影响到模型对于融资金额影响因素的判断。

案例 1（主业务问题：是否发生交易）中“company_code”非空，“trading_days”取值空，则默认该公司并未发生交易。

而案例 2（主业务问题：融资金额预测）中“company_code”非空，“financing_cnt”空，则默认该公司未发生融资，共有 1328 个样本，为无用样本。

其次“financing_amount_wan”的均值远高于 75%的样本，查看并筛选掉 1 个异常公司数据。

（2）“current_rt”和“financing_amount_wan”两个字段，由于样本量的减少，都从其他分布变成近似对数正态分布。

（3）数据改造方案中不需要考虑对非平衡训练集的问题。

内容简介

本节主要进行数据的预处理工作，基本内容上一案例中的相同，但在代码实现上新增加流水线结构和 lambda 函数来优化代码。

1. 主要内容

- 按照数据改造方案中的内容进行本案例的数据预处理工作；
- 通过使用流水线结构和 lambda 函数来优化数据预处理部分代码。

2. 学习目标

学完本节，能解决以下问题：

- 如何使用流水线结构和 lambda 函数进行数据预处理部分的代码优化？

数据预处理总结

1、使用流水线结构和 lambda 函数优化数据预处理部分的代码

流水线结构

流水线结构主要对本案例中，单个字段多步数据预处理工作进行打包，使得相同或较多步骤的字段处理起来更简洁。

（1）使用 sklearn 包中的 Pipeline()函数

首先从 sklearn 包中导入 pipeline，代码：

from sklearn.pipeline import Pipeline

其次在 Pipeline()函数的括号中放入（命名，数据预处理的函数），例如案例中的代码：

```
impute_dummy_transformer = Pipeline(
    [('imputer', SimpleImputer(strategy='most_frequent')),
     ('onehot', OneHotEncoder(categories="auto", sparse=False))])
```

是将缺失值填充使用的 SimpleImputer()函数和文本型转换为数值型使用的 OneHotEncoder()函数放在 Pipeline()函数的括号中，并分别命名为‘impute’和‘onehot’。最后将这个流水线命名为“impute_dummy_transformer”。

最后在使用该流水线的字段后，直接使用该流水线即可，例如本案例中'transfer_mode'、'layer'和'highest_executive_edu'三个字段，均需要先进行众数填充，再进行文本转换。

对比使用流水线前后的字段转换部分：

未优化前代码

```
(['transfer_mode'], [SimpleImputer(strategy='most_frequent'),
                     OneHotEncoder(categories='auto', sparse=False)]),
(['layer'], [SimpleImputer(strategy='most_frequent'),
             OneHotEncoder(categories='auto', sparse=False)]),
(['highest_executive_edu'], [SimpleImputer(strategy='most_frequent'),
                             OneHotEncoder(categories='auto', sparse=False)])
```

优化后代码：

```
(['transfer_mode'], impute_dummy_transformer),
(['layer'], impute_dummy_transformer),
(['highest_executive_edu'], impute_dummy_transformer)
```

差异还是较为明显的。

（2）sklearn 包中的 make_pipeline()函数

首先从 sklearn 包中导入 make_pipeline，代码：

from sklearn.pipeline import make_Pipeline

使用 pipeline()函数和使用 make_pipeline()函数的区别在于括号中需不要放每种方法的命名，两种代码在同一流水线的对比：

Pipeline()函数

```
impute_dummy_transformer = Pipeline(
    [('imputer', SimpleImputer(strategy='most_frequent')),
     ('onehot', OneHotEncoder(categories="auto", sparse=False))])
```

make_pipeline()函数

```
impute_dummy_transformer = make_pipeline(SimpleImputer(strategy='most_frequent'),
                                         OneHotEncoder(categories="auto", sparse=False))
```

比较而言，后者在书写时更加简便些。

函数 make_pipeline()定义的流水线与 Pipeline()定义的流水线，使用方法一致。

lambda 函数

对于涉及异常数据压缩处理的流水线，虽然得到了优化，但仍然感觉在定义流水线时比较复杂，因为按照不同的压缩区间需要定义的不同流水线。lambda 函数通过将压

缩区间范围设置成两个参数，实现了函数调用过程中压缩区间的自定义设置。

（1）生成流水线时有无 lambda 函数的代码差异：

无 lambda 函数代码

```
truncate_transformer_1 = make_pipeline( SimpleImputer(),
                                        FunctionTransformer(truncate_wrapper(0, 1), validate=True),
                                        StandardScaler())
truncate_transformer_2 = make_pipeline( SimpleImputer(),
                                        FunctionTransformer(truncate_wrapper(-1, 1), validate=True),
                                        StandardScaler())
truncate_transformer_3 = make_pipeline( SimpleImputer(),
                                        FunctionTransformer(truncate_wrapper(-1, 5), validate=True),
                                        StandardScaler())
truncate_transformer_4 = make_pipeline( SimpleImputer(),
                                        FunctionTransformer(truncate_wrapper(-10, 10), validate=True),
                                        StandardScaler())
```

存在四种不同压缩区间，则需要定义四个流水线。

有 lambda 函数代码

```
truncate_transformer = lambda l, r: make_pipeline( SimpleImputer(),
                                                   FunctionTransformer(truncate_wrapper(l, r), validate=True),
                                                   StandardScaler())
```

只需要定义一个流水线。

（2）两种流水线在使用时的代码区别：

无 lambda 函数代码

```
(['customer_rt_5all'], truncate_transformer_1),
(['supplier_rt_5all'],truncate_transformer_1),
(['gross_profit_rt'], truncate_transformer_2),
(['net_profit_rt'], truncate_transformer_2),
(['roa'], truncate_transformer_2),
(['asset_gr'], truncate_transformer_3),
(['income_gr'], truncate_transformer_3),
(['net_profit_gr'], truncate_transformer_4),
```

有 lambda 函数代码

```
(['customer_rt_5all'], truncate_transformer(0, 1)),
(['supplier_rt_5all'], truncate_transformer(0, 1)),
(['gross_profit_rt'], truncate_transformer(-1, 1)),
(['net_profit_rt'], truncate_transformer(-1, 1)),
(['roa'], truncate_transformer(-1, 1)),
(['asset_gr'], truncate_transformer(-1, 5)),
(['income_gr'], truncate_transformer(-1, 5)),
(['net_profit_gr'], truncate_transformer(-10, 10)),
```

可以发现使用 lambda 函数的流水线更加灵活方便。

内容简介

本节中首先会介绍模型泛化和数据分割的一些基本概念，随后对本案例的数据集进行数据分割。

1. 主要内容

- 模型的泛化、过拟合和欠拟合；
- 数据分割方法与操作；

2. 学习目标

学完本节，能解决以下问题：

- 数据分割中的相关概念有哪些？分别代表什么意思？
- 在本案例中如果实现对数据的分割？

模型的泛化、过拟合和欠拟合

在有监督学习中，我们会在训练集上建立一个模型，之后会把这个模型用于新的之前从未见过的数据中，这个过程称为**模型泛化**（generalization）。当然我们希望模型对于新数据的预测能够尽可能准确，这样才能说模型泛化的准确度比较高。

用什么标准来判断一个模型的泛化是好还是差呢？我们可以使用测试集对模型的表现进行评估。

过拟合（over-fitting）可以理解为构建的模型在训练集中表现得过于优越，但是在测试集的表现非常差，这种情况说明模型出现了过拟合的问题。

例如：一个分类模型的目标是根据图片分类图片中的物体是否为树叶，如果训练集图片中的所有树叶的边缘都是锯齿形的，那么基于这个训练集得到的模型可能也就会认为，只有边缘是锯齿形的物体，才有可能被分类为树叶。但是，这样的规则显然不适用于所有的树叶图片。这种模型学习到训练集中过于“严格”的假设的情况，就是过拟合。

产生过拟合的原因有很多，例如非平衡数据：模型只需要全部输出大类对应标签即可达到较高准确率；异常数据：使得分类器将部分噪音认为是特征从而扰乱了预设的分类规则；模型复杂度过高：当模型的参数过多时，导致模型过度解读现有数据集，而丧失一般性，因此设置倾向于稀疏化的正则化“惩罚”部分参数，来控制模型的复杂度等。

欠拟合（under-fitting）是指如果模型过于简单，在训练集和测试集的得分都非常差，连训练集的特点都不能完全考虑到的情况。

只有模型在训练集和测试集得分都比较高的情况下，我们才会认为模型对数据拟合的程度刚刚好，同时泛化的表现也会更出色。

分割方法

1. 训练集、验证集和测试集

在较为严谨的分析过程中，涉及到有监督算法的数据建模主要使用三种数据集：训练集、验证集、测试集。

数据分割的主要目标就是将现有的数据集分为训练集、验证集和测试集（在很多情况下验证集可以省略）。

训练集：学习样本数据集特征来训练模型；

验证集：评估模型的训练效果，调整模型的超参数；

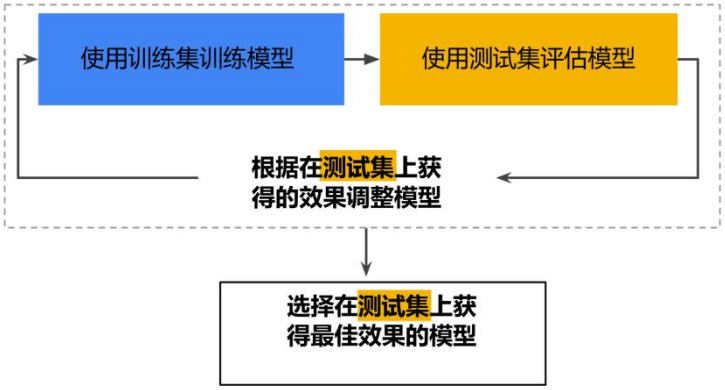
测试集：用来评估最终模型的泛化能力，但不能作为模型调参的依据。

在上一个案例中我们已经了解到，每个算法模型都涉及到不同的超参数，超参数的设定会明显影响算法的效果。因此，数据建模往往可以分为两步：

首先确定合适的超参数，再建立最终的模型。

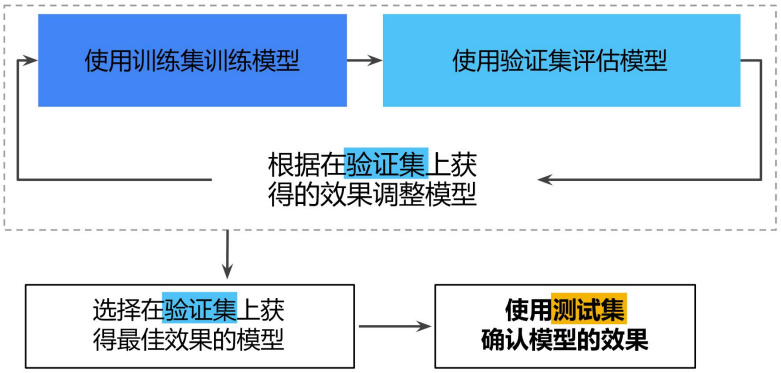
一般来说，需要调整超参数的模型，分割验证集都会使得结果更严谨可信，所以首选都是要划分验证集的。

基于我们在之前有监督算法中对于训练集和测试集的理解，在加入验证集前，模型训练流程如下图所示：



无论是使用 train_test_split()函数还是 k 折交叉验证的分割方法，都会导致测试集参与了模型参数的调整，使得测试集的知识“泄漏”到模型中，从而导致模型存在过拟合问题。为了解决这个问题，我们增加验证集替代测试集在模型训练时的作用，而测试集作为独立的数据集来最终确认模型的效果。

增加验证集后的模型训练流程如下图所示：



三者的关系可以理解为课本、题库和考试的关系，学生根据课本里的内容来掌握知识，同时题库反映了学习的效果和控制学习的进度，最后考试的题是平常都没有见过的，考察学生举一反三的能力。而有些时候学习完了并非去刷题库就直接参加考试，因此验证集并不是必须的，同时也不够严谨。

2. 本案例中的数据分割方法

我们已经学习了两种数据分割方法：

①直接下定义；②k-fold 交叉验证。

本案例使用的**线性回归模型**，线性回归模型属于一种有监督模型，按照以训练集训练模型，使用验证集来搜寻模型超参数，以及通过测试集评估模型的泛化能力进行数据分割：

①首先采用直接下定义的方法（sklearn 包中 train_test_split()函数）将数据集拆

分为训练集和测试集；

②其次使用 k-fold 交叉验证从训练集中拆分出验证集，用于网格搜索超参数。

数据分割总结

1、数据分割中的相关概念有哪些？

（1）模型的泛化

模型的泛化是指：在有监督学习中，我们会在训练集上建立一个模型，之后会把这个模型用于新的之前从未见过的数据中，这个过程称为模型泛化。

（2）过拟合和欠拟合

过拟合可以理解为构建的模型在训练集中表现得过于优越，但是在测试集的表现非常差，这种情况说明模型出现了过拟合的问题。

欠拟合是指如果模型过于简单，在训练集和测试集的得分都非常差，连训练集的特点都不能完全考虑到的情况。

（3）训练集、验证集和测试集

训练集：学习样本数据集特征来训练模型；

验证集：评估模型的训练效果，调整模型的超参数；

测试集：用来评估最终模型的泛化能力，但不能作为模型调参的依据。

三者的关系可以理解为课本、题库和考试的关系，学生根据课本里的内容来掌握知识，同时题库反映了学习的效果和控制学习的进度，最后考试的题是平常都没有见过的，考察学生举一反三的能力。而有些时候学习完了并非去刷题库就直接参加考试，因此验证集并不是必须的，同时也不够严谨。

2、在本案例中如果实现对数据的分割？

为了得到更严谨的模型，本案例采用两步，分别对测试集、训练集和验证集进行分割。

（1）直接下定义的方法将数据集拆分为训练集和测试集

使用 sklearn 中提供的 train_test_split()函数分割训练集和测试集，代码如下：

```
# 导入数据集分割工具
from sklearn.model_selection import train_test_split
# 将数据集分割为训练集与测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1000, random_state=0)
```

将样本分成测试集尺寸为 1000 个样本，剩余样本为训练集的两部分。

（2）k-fold 交叉验证从训练集中拆分出验证集，用于网格搜索超参数

首先需要从 sklearn.model_selection 包中导入 KFold()函数，代码如下：

from sklearn.model_selection import KFold

其次需要设置 KFold 中的三个参数，代码如下：

kf = KFold(n_splits=4, shuffle=True, random_state=0)

其中 n_spilts=4 表示将训练集拆分为 4 等份进行交叉验证；

shuffle 表示在切分前是否对数据进行洗牌：取值 False 时，不进行随机排序，每次切分结果相同，取值 True 时，会在切分前对数据进行重新排序，每次切分的结果不同；

random_state 为随机状态。

设置好参数后，将分割方法命名为 kf。

最后执行该分割方法，分割本案例的数据集，代码为：

kf.split(X_train, y_train)

这样就将训练集分割成 4 等份，若查看切分后得到训练集和验证集，可以使用 for 循环查看得到的训练集和验证集的索引。代码为：

```
for train_index, validate_index in kf.split(X_train, y_train):
    print('train_index:', train_index, 'validate_index:', validate_index)
```