

# mlr3特征工程

---

特征工程 PipeOp 的三种用法

## 一. 缺失值插补

1. 简单插补
2. 随机抽样插补
3. 直方图法插补
4. 学习器插补
5. 超出范围插补（特别适合基于树的模型）

## 二. 特征工程

1. 特征缩放
2. 特征变换
  - (1) 非线性特征
  - (2) 计算新特征
  - (3) 正态性变换
  - (4) 连续变量分箱
3. 特征降维
  - (1) PCA
  - (2) 核 PCA
  - (3) ICA：独立成分分析
4. 分类特征
  - (1) 因子折叠
  - (2) 因子修正
  - (3) 因子编码
  - (4) 效应编码
5. 处理类不平衡
  - (1) 欠采样与过采样
  - (2) SMOTE 法
  - (3) 样本加权
6. 目标变换

数据预处理，包含数据清洗和特征工程。

- 数据清洗，是在原始数据上执行以消除错误、噪声和冗余，通常独立于机器学习流程之外来做。
- 特征工程，涵盖了将数据输入机器学习模型之前对数据进行的所有其他转换，包括从非结构化数据（如书面文本、时间序列或图像）创建特征、插补缺失值、因子编码、特征和目标变换、不平衡处理以及泛函特征提取等。特征工程的目的是使数据能够被给定的学习器处理，以及提升模型的预测性能。

**特征工程可以构造新的特征来构造更好的模型。**

**有的算法能够直接处理分类变量！有的算法只能处理数值特征，需要把分类变量编码成数值特征。**

**有缺失值，随机森林不能学习。有的算法可以支持缺失值。**

特征工程对于简单的算法有更大的帮助，而高度复杂的模型通常从中获益较少。特征工程能够解决的常见数据问题包括：有偏分布的特征、高维分类特征、缺失值、高维数据和分类任务中的类不平衡。深度学习在自然语言处理和计算机视觉领域的自动化特征工程方面表现良好，但**对于标准的表格数据，基于树的集成方法如随机森林或梯度提升通常仍然优于深度学习（且更容易处理）。**

**高维数据需要做特征工程，类不平衡的数据也要做。**

```
1 library(tidyverse)
2 library(mlr3verse)
```

mlr3pipelines 包提供了常用特征工程的 PipeOPs，能够大大简化手动进行特征工程的过程：

- 选择特征工程步相应的 PipeOp；
- 多个特征工程步通过管道符%>>% 连接；
- 使用 affect\_columns 参数或 po("select") 并行，以选择特征工程步的影响列。

## 特征工程 PipeOp 的三种用法

(1) 调试：查看特征工程步对输入数据做了什么（见上一讲）

(2) 用于机器学习：原任务经过特征工程管道变成预处理之后的**新任务**，再正常用于机器学习流程

```
newtask = gr$train(list(task))[[1]]
```

(3) 用于机器学习：再接一个学习器，转化成图学习器，同普通学习器一样使用，适合对特征工程步、机器学习算法一起联动调参

```
gr = gr %>>% lrn("classif.rpart")
```

```
gr_learner = as_learner(gr)
```

## 一. 缺失值插补

目前支持的插补方法：

```
1 as.data.table(mlr_pipeops)[tags %in% "missings", "key"]
2 #> Key: <key>
3 #> key
4 #> <char>
5 #> 1: imputeconstant
6 #> 2: imputehist
7 #> 3: imputelearner #用学习器插补某个特征，把这个特征作为目
   标变量
8 #> 4: imputemean
9 #> 5: imputemedian
10 #> 6: imputemode #众数插补
11 #> 7: imputeoor
12 #> 8: imputesample #抽样插补。随机抽一个补上
```

也有先用不缺的数据学习模型，在用预测的值插补缺失值。把插补流程放在数据清洗部分。

## 1. 简单插补

```
task = tsk("pima")
```

```
task$missings() # 查看任务的缺失情况
```

```
1 task = tsk("pima")
2 task$missings() # 查看任务的缺失情况
3 #> diabetes age glucose insulin mass pedigree
4 #> 0 0 5 374 11 0
5 #> prtriceps
6 #> 227
```

- 用某常数、均值、中位数、众数插补。

```
po = po("imputeconstant", constant = -999,  
affect_columns = selector_name("glucose"))  
  
# po = po("imputemean") # 均值插补  
  
# po("imputemedian")  
  
# po("imputemode")  
  
# 中位数插补  
  
# 众数插补  
  
newtask = po$train(list(task))[[1]]  
  
newtask$missings()
```

```
1 po = po("imputeconstant", constant = -999,  
2 affect_columns = selector_name("glucose"))  
3 # po = po("imputemean") # 均值插补  
4 # po("imputemedian") # 中位数插补  
5 # po("imputemode") # 众数插补  
6 newtask = po$train(list(task))[[1]]  
7 newtask$missings()  
8 #> diabetes age insulin mass pedigree pregnant pr  
9 #>      0      0      374      11      0      0  
10 #> glucose  
11 #> 0
```

## 2.随机抽样插补

- 通过从非缺失的训练数据中随机抽样来插补特征。

```
po = po("imputesample")
```

```
newtask =po$train(list(task))[[1]]
```

```
newtask$missings()
```

```
#>diabetes agepe
```

```
1 po = po("imputesample")
2 newtask = po$train(list(task))[[1]]
3 newtask$missings()
4 #> diabetes age pedigree pregnant glucose insulin
5 #> 0 0 0 0 0 0
6 #> triceps
7 #> 0
```

## 3.直方图法插补

- 用直方图法插补数值特征，其原理是基于对非缺失值的分布进行建模，通过生成随机值来填补缺失值，以尽可能地保持数据的分布特征

```
po = po("imputehist")
```

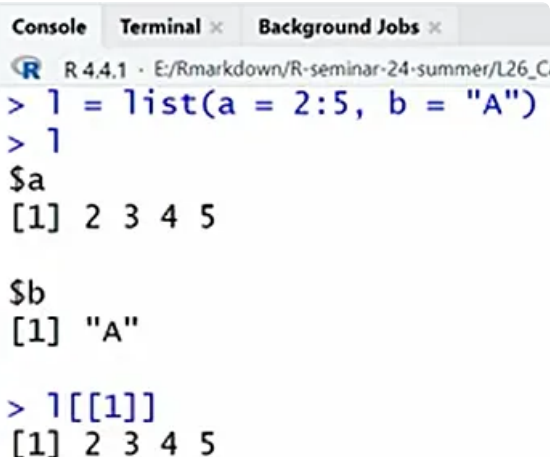
```
newtask =po$train(list(task))[[1]]
```

```
newtask$missings()
```

```

1 po = po("imputehist") # 尽量保持服从同样的分布
2 newtask = po$train(list(task))[[1]]
3 newtask$missings()
4 #> diabetes age pedigree pregnant glucose insulin
5 #> 0 0 0 0 0 0
6 #> triceps
7 #> 0

```



```

Console Terminal × Background Jobs ×
R 4.4.1 · E:/Rmarkdown/R-seminar-24-summer/L26_C
> l = list(a = 2:5, b = "A")
> l
$a
[1] 2 3 4 5

$b
[1] "A"

> l[[1]]
[1] 2 3 4 5

```

列表取出来用两个[[].

#### 4. 学习器插补

通过为每个特征拟合一个学习器来插补特征(**插补哪个特征就把该特征做目标**)，使用参数 `context_columns` 所指示的特征作为特征来训练插补学习器。

注意，只有插补学习器支持的特征，即**回归学习器只能插补整数和数值特征**，分类学习器可以插补因子、有序因子和逻辑值特征。

用于插补的学习器是在所有的 `context_columns` 上训练的；如果这些列包含缺失值，学习器通常需要能够自己处理缺失值，或者需要做先行

插补。

- 决策树插补

```
po = po("imputelearner",lrn("regr.rpart"))
```

```
newtask =po$train(list(task))[[1]]
```

```
newtask$missings()
```

```
1 po = po("imputelearner", lrn("regr.rpart"))
2 newtask = po$train(list(task))[[1]]
3 newtask$missings()
4 #> diabetes age pedigree pregnant glucose insulin
5 #> 0 0 0 0 0 0
6 #> triceps
7 #> 0
```

- KNN插补，训练KNN学习器时，先用直方图法插补

```
po = po("imputelearner",
```

```
po("imputehist")%>>% lrn("regr.kknn")) # knn不能有缺失值，直方图插补不影响原任务，直方图处理后交给knn，对原始任务没有影响。外层是有NA的，由knn学习器插补。
```

```
newtask =po$train(list(task))[[1]]
```

```
newtask$missings()
```

```
#>diabetes agepedigree
```



```

1 po = po("imputelearner",
2 po("imputehist") %>% lrn("regr.kknn")) # 直方图临时先插
   补数据，用knn学习器，插补原缺失值
3
4 newtask = po$train(list(task))[[1]]
5 newtask$missings()
6 #> diabetes age pedigree pregnant glucose insulin
7 #> 0 0 0 0 0 0
8 #> triceps
9 #> 0

```

## 5.超出范围插补（特别适合基于树的模型）

- 通过增加一个新的水平".MISSING"来插补因子特征。

- 通过使用 $\min(x) - \text{offset} - \text{multiplier} * \text{diff}(\text{range}(x))$ 或

$\max(x) + \text{offset} + \text{multiplier} * \text{diff}(\text{range}(x))$ ，移到最小值以下或最大值以上的常量值插补数值特征。# **缺失值可以落在 $[\min, \max]$ 之外**

该插补法在基于树的模型中尤其合理。（树模型不怕缺失值，当然补了后更好）

```
set.seed(123)
```

```
data = tsk("pima")$data()
```

```
data$y= factor(c(NA,sample(letters, 766,
```

```
replace= TRUE), NA))
```

```
data$z= ordered(c(NA, sample(1:10,767, replace= TRUE)))
```

```
task = as_task_classif(data, target= "diabetes")
```

```
po = po("imputeoor")
```

```
newtask = po$train(list(task))[[1]]
```

```
newtask$missings()
```

```
#> diabetes
```

```
1  set.seed(123)
2  data = tsk("pima")$data()
3  data$y = factor(c(NA, sample(letters, 766,
4  replace = TRUE), NA)) #增加两列
5  data$z = ordered(c(NA, sample(1:10, 767, replace = TRUE)))
6  task = as_task_classif(data, target = "diabetes")
7  po = po("imputeoor")
8  newtask = po$train(list(task))[[1]]
9  newtask$missings() #生成的y, z有缺少, 外插补
10 #> diabetes age pedigree pregnant glucose insulin
11 #> 0 0 0 0 0 0
12 #> triceps y z
13 #> 0 0 0
```

另外，跟缺失值相关的特征工程还有 `po("missind")`：在任务中增加是否缺失指示列（将删除原始特征），通常与 `po("featureunion")` 和各插补PipeOps 结合使用。

注意 `affect_columns` 是用

```
selector_invert(selector_type(c("factor", "ordered",
```

"character")) 初始化的，因为因子列的缺失值通常是用超出范围的插补（po("imputeoor"))。

注：当然也可以在 mlr3 框架外面，单独对数据做缺失值插补，再用来创建任务。

## 二. 特征工程

### 1. 特征缩放

不同数值型特征的数据量纲可能相差多个数量级，这对很多数据模型会有很大影响，所以有必要做归一化处理，就是将列或行对齐并转化为一

致。

```
df = as_tibble(iris) %>%
```

```
  set_names(str_c("x", 1:4), "Species")
```

```
  task = as_task_classif(df, target = "Species")
```

把数据变成0-1之间。

```
1 df = as_tibble(iris) %>%
2 set_names(str_c("x", 1:4), "Species") #修改列名，短一点好看
3 task = as_task_classif(df, target = "Species")
```

#### (1) 标准化

标准化也称为  $Z$  标准化，将数据变成均值为0，标准差为1：

$$z = \frac{x - \mu}{\sigma}$$

注：0 就代表均值，更方便模型解释；若只减去均值不除以标准差，则是做中心化。

```
pos = po("scale") # scale=FALSE 只做中心化
```

```
pos$train(list(task))[[1]]$data() # 标准化后的结果
```

```
1 pos = po("scale") # scale=FALSE 只做中心化
2 pos$train(list(task))[[1]]$data()
3 #> Species x1 x2 x3 x4
4 #> <fctr> <num> <num> <num> <num>
5 #> 1: setosa -0.8977 1.0156 -1.336 -1.311
6 #> 2: setosa -1.1392 -0.1315 -1.336 -1.311
7 #> 3: setosa -1.3807 0.3273 -1.392 -1.311
8 #> 4: setosa -1.5015 0.0979 -1.279 -1.311
9 #> 5: setosa -1.0184 1.2450 -1.336 -1.311
```

## (2) 归一化

归一化是将数据线性放缩到 [0,1]（根据需要也可以线性放缩到 [a,b]），一般还同时考虑指标一致化，将正向指标（值越大越好）和负向指标（值越小越好）都变成正向。

正向指标：

$$x'_i = \frac{x_i - \min x_i}{\max x_i - \min x_i}$$

负向指标：

$$x'_i = \frac{\max x_i - x_i}{\max x_i - \min x_i}$$

```
pop = po("scalerange", lower= 0, upper= 1)
```

```
pop$train(list(task))[[1]]$data()
```

```
1 pop = po("scalerange", lower = 0, upper = 1)
2 pop$train(list(task))[[1]]$data()
3 #> Species x1 x2 x3 x4
4 #> <fctr> <num> <num> <num> <num>
5 #> 1: setosa 0.2222 0.625 0.0678 0.0417
6 #> 2: setosa 0.1667 0.417 0.0678 0.0417
7 #> 3: setosa 0.1111 0.500 0.0508 0.0417
8 #> 4: setosa 0.0833 0.458 0.0847 0.0417
9 #> 5: setosa 0.1944 0.667 0.0678 0.0417
10 #> ---
11 #> 146: virginica 0.6667 0.417 0.7119 0.9167
12 #> 147: virginica 0.5556 0.208 0.6780 0.7500
13 #> 148: virginica 0.6111 0.417 0.7119 0.7917
14 #> 149: virginica 0.5278 0.583 0.7458 0.9167
15 #> 150: virginica 0.4444 0.417 0.6949 0.7083
```

### (3) 行规范化

行规范化，**常用于文本数据或聚类算法**，是保证每行具有单位范数，即每行的向量“长度”相同。想象一下， $m$ 个特征下，每行数据都是 $m$ 维空间中的一个点，做行规范化能让这些点都落在单位球面上（到原点的距离均为 1）。

行规范化，一般采用  $L2$  范数：

$$x'_{ij} = \frac{x_{ij}}{\|x_i\|} = \frac{x_{ij}}{\sqrt{\sum_{j=1}^m x_{ij}^2}}$$

```
pop = po("spatialsign")
```

```
pop$train(list(task))[[1]]$data()
```

▼ 不常用，按行来做

R

```
1 pop = po("spatialsign")
2 pop$train(list(task))[[1]]$data()
3 #> Species x1 x2 x3 x4
4 #> <fctr> <num> <num> <num> <num>
5 #> 1: setosa 0.804 0.552 0.221 0.0315
6 #> 2: setosa 0.828 0.507 0.237 0.0338
7 #> 3: setosa 0.805 0.548 0.223 0.0343
8 #> 4: setosa 0.800 0.539 0.261 0.0348
```

## 2. 特征变换

### (1) 非线性特征

#### 有某些模型是非线性的！

对于数值特征  $x_1, x_2, \dots$ ，可以创建更多的多项式项特征： $x_1^2, x_1 \cdot x_2, x_2^2, \dots$ ，这相当于是用自变量的更高阶泰勒公式去逼近因变量。

使用 `po("modelmatrix")`，再结合如下的 formula 模型公式语法：

- $y \sim .$ ：包含所有自变量的主效应
- $x_1:x_2$ ：交互效应，即  $x_1 x_2$  项
- $x_1 \cdot x_2$ ：包含全部主效应和交互效应， $x_1 + x_2 + x_1:x_2$  的简写
- $l()$ ：打包式子作为整体
- $y \sim \text{poly}(x, 2, \text{raw} = \text{TRUE})$ ：一元二次多项式回归，同  $y \sim x + l(x^2)$

- $y \sim \text{polym}(x_1, x_2, \text{degree} = 2, \text{raw} = \text{TRUE})$ : 二元二次多项式回归, 不能超过3次, 龙格现象

- $\log(y) \sim x$ : 对  $y$  做对数变换

```
pop = po("modelmatrix",
```

```
formula = ~ . ^ 2 + I(x1 ^ 2) + log(x2))
```

```
pop$train(list(task))[[1]]$data()
```

```
1 pop = po("modelmatrix",
2 formula = ~ . ^ 2 + I(x1 ^ 2) + log(x2)) #把每个的平方项
   加进来
3 pop$train(list(task))[[1]]$data()
4 #> Species (Intercept) x1 x2 x3 x4 I(x1^
5 #> <fctr> <num> <num> <num> <num> <num> <nu
6 #> 1: setosa 1 5.1 3.5 1.4 0.2 26
7 #> 2: setosa 1 4.9 3.0 1.4 0.2 24
8 #> 3: setosa 1 4.7 3.2 1.3 0.2 22
9 #> 4: setosa 1 4.6 3.1 1.5 0.2 21
10 #> 5: setosa 1 5.0 3.6 1.4 0.2 25
11 #> ---
12 #> 146: virginica 1 6.7 3.0 5.2 2.3 44
13 #> 147: virginica 1 6.3 2.5 5.0 1.9 39
14 #> 148: virginica 1 6.5 3.0 5.2 2.0 42
15 #> 149: virginica 1 6.2 3.4 5.4 2.3 38
16 #> 150: virginica 1 5.9 3.0 5.1 1.8 34
17 #> x1:x4 x2:x3 x2:x4 x3:x4 23
```

另一种常用的非线性特征是基于自然样条的样条特征:

```
pop = po("modelmatrix", formula = ~ splines::ns(x1, 5))
```

```
pop$strain(list(task))[[1]]$data()
```

```
1 #另一种常用的非线性特征是基于自然样条的样条特征:
2 pop = po("modelmatrix", formula = ~ splines::ns(x1, 5
  ) )#ns自然样条
3 pop$strain(list(task))[[1]]$data()
4 #> Species (Intercept) splines::ns(x1, 5)1 splines::
5 #> <fctr> <num> <num>
6 #> 1: setosa 1 3.09e-01
7 #> 2: setosa 1 1.32e-01
8 #> 3: setosa 1 3.91e-02
9 #> 4: setosa 1 1.65e-02
10 #> 5: setosa 1 2.09e-01
11 #> ---
12 #> 146: virginica 1 0.00e+00
13 #> 147: virginica 1 1.81e-02
14 #> 148: virginica 1 1.36e-05
15 #> 149: virginica 1 5.58e-02
16 #> 150: virginica 1 3.63e-01
17 #> splines::ns(x1, 5)3 splines::ns(x1, 5)4 splines::ns
18
```

探索:

```
df1 = tibble(n = 1:150, x = iris$Sepal.Length)
```

```
mdl = lm(x ~ splines::ns(n, 5), df1)
```

```
summary(mdl)
```



```
Call:
lm(formula = x ~ splines::ns(n, 5), data = df1)
```

Residuals:

Min	1Q	Median	3Q	Max
-1.45889	-0.36871	-0.06182	0.32425	1.35940

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )	
(Intercept)	5.1301	0.1988	25.811	< 2e-16	***
splines::ns(n, 5)1	1.1794	0.2529	4.664	7.02e-06	***
splines::ns(n, 5)2	0.3671	0.3218	1.141	0.2559	
splines::ns(n, 5)3	2.1235	0.2495	8.510	2.07e-14	***
splines::ns(n, 5)4	1.0097	0.5036	2.005	0.0468	*
splines::ns(n, 5)5	1.4967	0.2187	6.842	2.07e-10	***

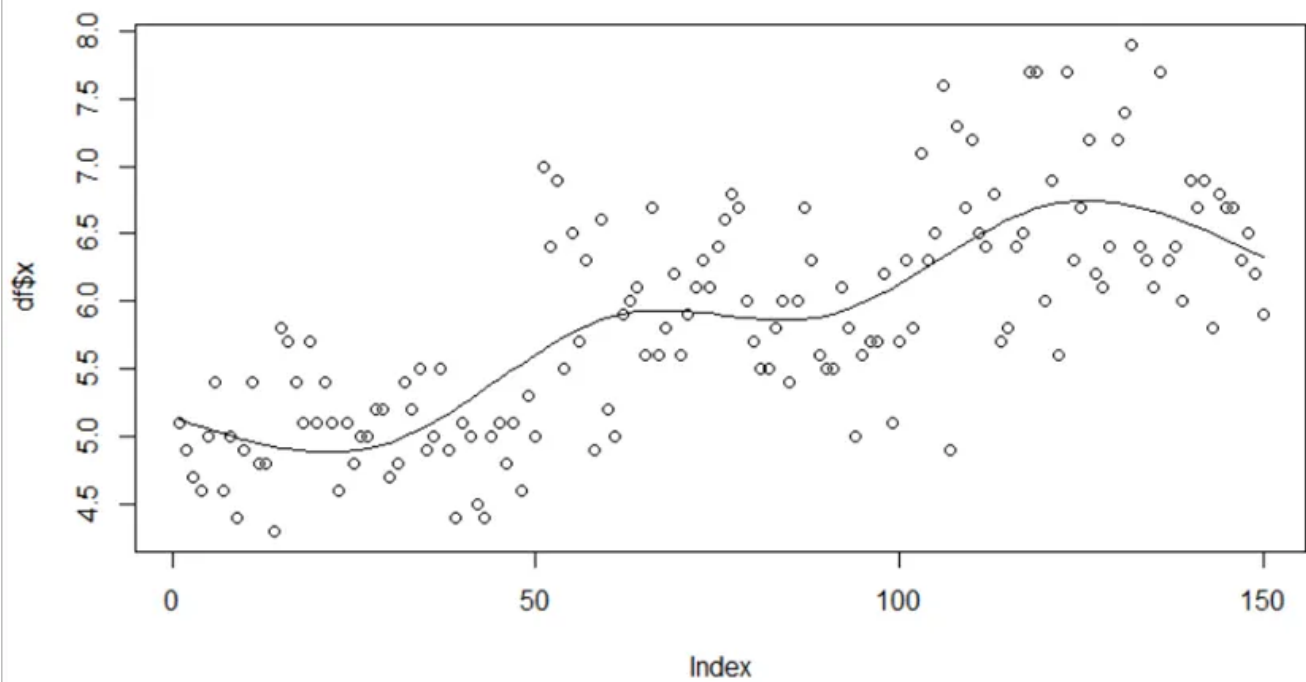
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5553 on 144 degrees of freedom  
Multiple R-squared: 0.5654, Adjusted R-squared: 0.5503  
F-statistic: 37.46 on 5 and 144 DF, p-value: < 2.2e-16

▼

R |

```
1 plot(df1$x)
2 lines(augment mdl)$fitted)
3
```



```
1 library(tidymodels)
2 rec = recipe(n ~ ., data = df1) %>%
3   step_ns(x, deg_free = 5)
4 rec %>%
5   prep() %>%
6   bake(new_data = NULL)
7
```

R |

```
# A tibble: 150 × 6
      n    x_ns_1    x_ns_2    x_ns_3    x_ns_4    x_ns_5
  <int>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
1     1  0.309    0.000997 -0.203    0.468   -0.265
2     2  0.132     0         -0.217    0.499   -0.282
3     3  0.0391     0         -0.175    0.404   -0.229
4     4  0.0165     0         -0.140    0.322   -0.182
5     5  0.209     0         -0.217    0.501   -0.284
6     6  0.610    0.0638    -0.107    0.247   -0.140
7     7  0.0165     0         -0.140    0.322   -0.182
8     8  0.209     0         -0.217    0.501   -0.284
9     9  0.000611     0        -0.0496    0.114   -0.0647
10    10  0.132     0         -0.217    0.499   -0.282
# ... with 140 more rows
```

---

## (2) 计算新特征

管道运算"mutate", 根据以公式形式给出的表达式添加特征, 这些表达式可能取决于其他特征的值。这可以增加新的特征, 也可以改变现有的特征。

```
pom = po("mutate", mutation = list(
```

```
  x1_p = ~ x1 + 1, #创建新列
```

```
  Area1 = ~ x1 * x2,
```

```
  Area2 = ~ x3 * x4,
```

```
  Area = ~ Area1 + Area2))
```

```
pom$train(list(task))[[1]]$data()
```

```

1  pom = po("mutate", mutation = list(
2  x1_p = ~ x1 + 1,
3  Area1 = ~ x1 * x2,
4  Area2 = ~ x3 * x4,
5  Area = ~ Area1 + Area2))
6  pom$train(list(task))[[1]]$data()
7  #> Species x1 x2 x3 x4 x1_p Area1 Area2
8  #> <fctr> <num> <num> <num> <num> <num> <num> <num>
9  #> 1: setosa 5.1 3.5 1.4 0.2 6.1 17.8 0.28
10 #> 2: setosa 4.9 3.0 1.4 0.2 5.9 14.7 0.28
11 #> 3: setosa 4.7 3.2 1.3 0.2 5.7 15.0 0.26
12 #> 4: setosa 4.6 3.1 1.5 0.2 5.6 14.3 0.30
13 #> 5: setosa 5.0 3.6 1.4 0.2 6.0 18.0 0.28
14 25

```

titanic任务，利用正则表达式从复杂字符串列提取出相关信息，并转化为因子特征：

```

po_ftextract = po("mutate",mutation=list(

fare_per_person = ~fare /(parch + sib_sp +1),

deck= ~factor(str_sub(cabin,1,1)),

title=~ factor(str_extract(name, "(?<=,).*?(?=\\.|)")),

surname= ~factor(str_extract(name, "^.*?(?=,)")),

ticket_prefix =~ factor(

str_extract(ticket,"^.*?(?=)"))

))

```

```
newtask =po_ftextract$train(list(tsk("titanic")))[[1]]
```

```
1 po_ftextract = po("mutate", mutation = list(  
2   fare_per_person = ~ fare / (parch + sib_sp + 1),  
3   deck = ~ factor(str_sub(cabin, 1, 1)), #从字符串中提取船  
   票等级, 反映身份  
4   title = ~ factor(str_extract(name, "(?<=, ).*(?=\\.|.)"  
   )), #头衔  
5   surname = ~ factor(str_extract(name, "^.*(?=,)")), #正  
   则表达式, 零宽断言  
6   ticket_prefix = ~ factor(  
7     str_extract(ticket, "^.*(?= )")  
8   )) # 多了很多新特征  
9 newtask = po_ftextract$train(list(tsk("titanic")))[[1  
  ]]  
10
```

若数据噪声太多的问题，通常就需要做数据平滑。

最简单的数据平滑方法是移动平均，即用一定宽度的小窗口滑过曲线，会把曲线的毛刺尖峰抹掉，能一定程度上去掉噪声还原原本曲线。

窗口宽度越大，平滑的效果越明显。

```
pom = po("mutate", mutation = list( # 做五点移动平均
```

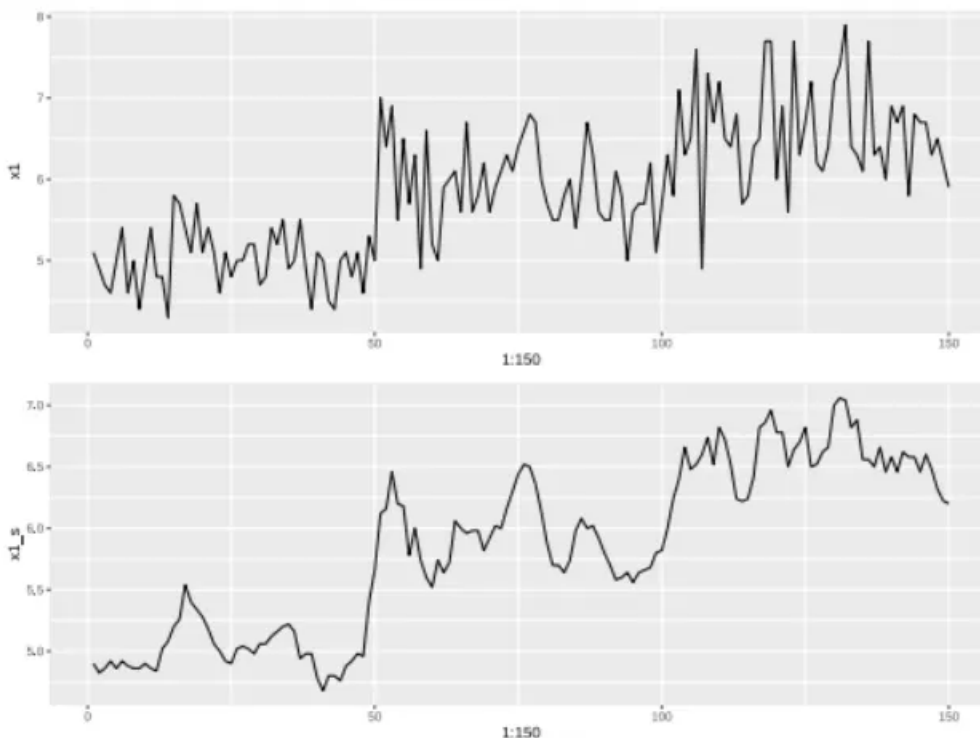
```
  x1_s = ~ slider::slide_dbl(x1, mean,
```

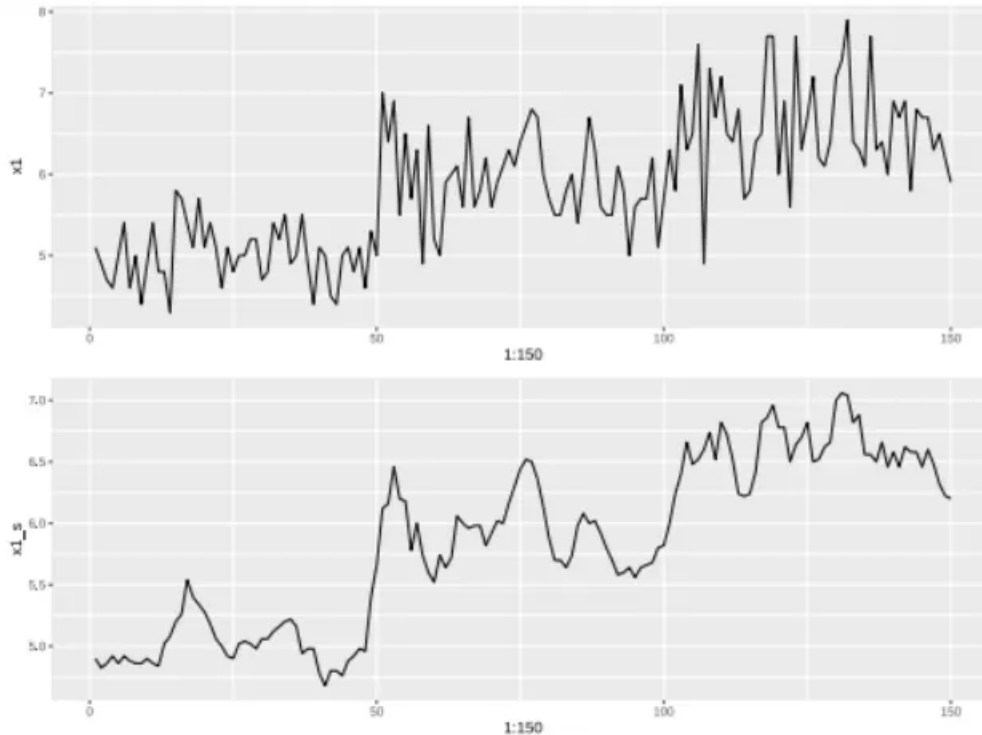
```
  .before = 2, .after = 2)))
```

```
dat = pom$train(list(task))[[1]]$data()
```

```
library(patchwork) p1 = ggplot(dat, aes(1:150, x1)) + geom_line()
p2 = ggplot(dat, aes(1:150, x1_s)) + geom_line() p1 / p2
```

```
1 pom = po("mutate", mutation = list( # 做五点移动平均
2   x1_s = ~ slider::slide_dbl(x1, mean,
3     .before = 2, .after = 2))) # 滑窗迭代, 前2后2, 移动平均
4 dat = pom$train(list(task))[[1]]$data()
5 library(patchwork)
6 p1 = ggplot(dat, aes(1:150, x1)) +
7   geom_line()
8 p2 = ggplot(dat, aes(1:150, x1_s)) +
9   geom_line()
10 p1 / p2
```





注：其它平滑法还有指数平滑、滤波、光滑样条等。

另外，还有：

- `po("colapply")`, #应用函数到任务的每一列，常用于类型转换：

```
poca = po("colapply", applicator = as.character) #列应用，把函数应用到列上
```

- `po("renamecolumns")`, #修改列名

```
pop = po("renamecolumns",  
renaming = c("Petal.Length"="PL", "Petal.Width"="PW"))
```

```

1 • po("colapply"), 应用函数到任务的每一列, 常用于类型转换:
2 poca = po("colapply", applicator = as.character)
3 • po("renamecolumns"), 修改列名, 可以在特征工程外做
4 pop = po("renamecolumns",
5   renaming = c("Petal.Length"="PL", "Petal.Width"="PW"))

```

### (3) 正态性变换

#### • Box-Cox 变换

Box-Cox 变换是更神奇的**正态性变换**, 用最大似然估计选择最优的  $\lambda$  值, 让非负的非正态数据变成正态数据:

$$y' = \begin{cases} \ln(y), & \lambda = 0 \\ (y^\lambda - 1)/\lambda, & \lambda \neq 0 \end{cases}$$

```

1 pop = po("boxcox")
2 newtask = pop$train(list(task))[[1]]

```

若数据包含 0 或负数, 则Box-Cox 变换不再适用, 可以改用同样原理的Yeo-Johnson 变换:

$$y' = \begin{cases} \ln(y + 1), & \lambda = 0, y \geq 0 \\ \frac{(y + 1)^\lambda - 1}{\lambda}, & \lambda \neq 0, y \geq 0 \\ -\ln(1 - y), & \lambda = 2, y < 0 \\ \frac{(1 - y)^{2-\lambda} - 1}{\lambda - 2}, & \lambda \neq 2, y < 0 \end{cases}$$



```
1 pop = po("yeojohnson")
2 newtask = pop$train(list(task))[[1]] #希望这些变量能逆回去
```

#### (4) 连续变量分箱

在统计和机器学习中，有时需要将连续变量转化为离散变量，称为连续变量离散化或分箱，常用于银行风控建模，特别是线性回归或 Logistic 回归模型。

分箱的好处有：

- (i) 使得结果更便于分析和解释。比如，年龄从中年到老年，患高血压比例增加 25%，而年龄每增加一岁，患高血压比例不一定有显著变化；
- (ii) 简化模型，将自变量与因变量间非线性的潜在的关系，转化为简单的线性关系。

当然，分箱也可能带来问题：简化的模型关系可能与潜在的模型关系不一致（甚至发现的是错误的模型关系）、删除数据中的细微差别、切分点可能没有实际意义

- 等宽分箱：

```
pop = po("histbin", breaks = 4) # 分四段
```

```
newtask = pop$train(list(task))[[1]]
```

- 分位数分箱

```
pop = po("quantilebin", numplits = 4) #前25%的数据落在前25%
```

```
newtask = pop$train(list(task))[[1]]
```

### 3. 特征降维

有时数据集可能包含过多特征，甚至是冗余特征，可以用降维技术压缩特征，但通常会降低模型性能。

#### (1) PCA

最常用的特征降维方法是主成分分析（PCA），是利用协方差矩阵的特征值分解原理，实现多个特征向少量综合特征（称为主成分）的转化，每个主成分都是多个原始特征的线性组合，且各个主成分之间互不相关，第一主成分是解释数据变异（方差）最大的，然后是次大的，依此类推。

$n$ 个特征，若转化为 $n$ 个主成分，则会保留原始数据的100%信息，但这就失去了降维的意义。所以一般是只选择前若干个主成分，一般原则是选择至保留 85% 以上信息的主成分。

```
pop = po("pca", rank.= 2)
```

```
pop$train(list(task))[[1]]$data()
```

```

1 pop = po("pca", rank. = 2)
2 pop$train(list(task))[[1]]$data()
3 #> Species PC1 PC2
4 #> <fctr> <num> <num>
5 #> 1: setosa -2.68 -0.3194
6 #> 2: setosa -2.71 0.1770
7 #> 3: setosa -2.89 0.1449
8 #> 4: setosa -2.75 0.3183
9 #> 5: setosa -2.73 -0.3268
10 #> ---
11 #> 146: virginica 1.94 -0.1875
12 #> 147: virginica 1.53 0.3753
13 #> 148: virginica 1.76 -0.0789
14 #> 149: virginica 1.90 -0.1166
15 #> 150: virginica 1.39 0.2827

```

## (2) 核 PCA

PCA 适用于数据的线性降维。而核主成分分析 (Kernel PCA) 可实现数据

的非线性降维，用于处理线性不可分的数据集。

核 PCA 的大致思路是：对于输入空间中的矩阵  $X$ ，先用一个非线性映射把  $X$  中的所有样本映射到一个高维甚至是无穷维的特征空间（使其线性可分），然后在这个高维空间进行 PCA 降维。

```
pop = po("kernelpca", features = 3) # 需要 kernelab 包
```

```
newtask = pop$train(list(task))[[1]]
```

```

1 pop = po("kernelpca", features = 3) # 需要 kernelab 包
2 newtask = pop$train(list(task))[[1]]

```

### (3) ICA: 独立成分分析

- 提取统计意义上的独立成分

```
pop = po("ica", n.comp = 3) # 需要 fastICA 包
```

```
newtask = pop$train(list(task))[[1]]
```

### (4) NMF: 非负矩阵分解

对于任意给定的非负矩阵  $V$ , NMF 算法能够寻找到非负矩阵  $W$  和非负矩阵  $H$ , 使它们的积为矩阵  $V$ 。非负矩阵分解的方法在保证矩阵的非负性的同时能够减少数据量, 相当于把  $n$  维的数据降维到  $r$  维。

```
# BiocManager::install("Biobase")
```

```
pop = po("nmf", rank = 3) # 需要 NMF 包, Biobase 包
```

```
newtask = pop$train(list(task))[[1]]
```

```

1 # BiocManager::install("Biobase") #生信方向的, 需要装好多包
2 pop = po("nmf", rank = 3) # 需要 NMF 包, Biobase 包
3 newtask = pop$train(list(task))[[1]]

```

### (5) 剔除常量特征

从任务中剔除常量特征。对于每个特征，计算不同于其众数值的比例。所有比例低于可设置阈值的特征都会从任务中剔除。缺失值可以被忽略，也可以视为与非缺失值不同的常规值。

```
library(data.table)
data = data.table(y = runif(10), a = 1:10,
                  b = rep(1, 10),
                  c = rep(1:2, each = 5))
task_ex = as_task_regr(data, target = "y")
```

- 剔除常量特征 b:

```
po = po("removeconstants")
po$train(list(task_ex))[[1]]$data()
#>           y      a      c
#>      <num> <int> <int>
#>  1: 0.1745      1      1
#>  2: 0.5986      2      1
#>  3: 0.0806      3      1
#>  4: 0.7073      4      1
#>  5: 0.5637      5      1
#>  6: 0.2828      6      2
#>  7: 0.3921      7      2
#>  8: 0.6727      8      2
#>  9: 0.1622      9      2
#> 10: 0.1085     10      2
```

- 剔除近似常量特征：若不同值比例小于5%，予以剔除

```
por = po("removeconstants", ratio = 0.05)
```

```
newtask = por$train(list(task))[[1]]
```

```
1 # 剔除近似常量特征：若不同值比例小于 5%，予以剔除
2 por = po("removeconstants", ratio = 0.05)
3 newtask = por$train(list(task))[[1]]
```

## 4. 分类特征

### (1) 因子折叠

使用 `po("collapsefactors")` 对因子或有序因子进行折叠，折叠训练样本中最少的水平，直到剩下 `target_level_count` 个水平。然而，那些出现率高于 `no_collapse_above_prevalence` 的水平会被保留。

对于因子变量，它们被折叠到下一个更大的水平，对于有序变量，稀有变量被折叠到相邻的类别，以样本数较少者为准。

训练集中没有出现的水平在预测集中不会被使用，因此经常与因子修正结合起来使用。

```
dat = tibble(color = factor(starwars$skin_color), y = 1)
```

```
count(dat, color) # 原数据 color 有 31 个水平
```

```
1 dat = tibble(color = factor(starwars$skin_color),  
2 y = 1)  
3 count(dat, color) # 原数据 color 有 31 个水平  
4 #> # A tibble: 31 x 2  
5 #> color n  
6 #> <fct> <int>  
7 #> 1 blue 2  
8 #> 2 blue, grey 2  
9 #> 3 brown 4  
10 #> # i 28 more rows
```

```
task = as_task_regr(dat, target = "y")
```

```
poc = po("collapsefactors", target_level_count = 5)
```

```
poc$train(list(task))[[1]]$data() %>%
```

```
count(color)
```

```
#> color
```



```

1 task = as_task_regr(dat, target = "y")
2 poc = po("collapsefactors", target_level_count = 5)
3 poc$train(list(task))[[1]]$data() %>%
4 count(color)
5 #> color n
6 #> <fctr> <int>
7 #> 1: dark 6
8 #> 2: fair 17
9 #> 3: green 6
10 #> 4: grey 47
11 #> 5: light 11

```

## (2)因子修正

使用`po("fixfactors")`对因子或有序因子做修正，确保预测过程中的因子水平与训练过程中的相同；可能在此之前删除训练中为空的因子水平。

注意，若在预测过程中发现未见过的因子水平，可能会导致缺失值的出现。

```

dattrain =data.table(
  a= factor(c("a","b", "c",NA),
    levels= letters),
  b= ordered(c("a", "b","c",NA)),target=1:4)
dattest =data.table(
  a= factor(c("a","b", "c","d")),
  b= ordered(c("a", "b","c","d"),

```

```
levels= letters[10:1]),target= 1:4)

tasktrain = as_task_regr(dattrain,"target")

tasktest =as_task_regr(datatest,"target")

op = po("fixfactors")

op$train(list(tasktrain))

op$predict(list(tasktest))[[1]]$data()
```

```

1  dattrain = data.table(
2  a = factor(c("a", "b", "c", NA),
3  levels = letters),
4  b = ordered(c("a", "b", "c", NA)), target = 1:4)
5  dattest = data.table(
6  a = factor(c("a", "b", "c", "d")),
7  b = ordered(c("a", "b", "c", "d"),
8  levels = letters[10:1]), target = 1:4)
9
10 tasktrain = as_task_regr(dattrain, "target")
11 tasktest = as_task_regr(dattest, "target")
12 op = po("fixfactors")
13 op$train(list(tasktrain))
14 #> $output
15 #> <TaskRegr:dattrain> (4 x 3)
16 #> * Target: target
17 #> * Properties: -
18 #> * Features (2):
19 #> - fct (1): a
20 #> - ord (1): b
21
22
23 op$predict(list(tasktest))[[1]]$data()
24 #> target a b
25 #> <int> <fctr> <ord>
26 #> 1: 1 a a
27 #> 2: 2 b b
28 #> 3: 3 c c
29 #> 4: 4 <NA> <NA>
30

```

### (3) 因子编码

对因子（也包括有序因子、字符型）特征进行重新编码。**把类别的变成数值**

用 `po("encode")` 实现，其参数 `method` 指定编码方法：

- “one-hot”：独热编码； #类别多的话，效果不好，维度灾难
- “treatment”：虚拟编码，创建  $n-1$  列，留出每个因子变量的第一个因子水平（`?stats::conc.treatment()`）； # 跟上面，差不多
- “helmert”：根据 Helmert 对比度创建列  
(`?stats::conc.helmert()`) ；
- “poly”：根据正交多项式创建对比列（`?stats::conc.poly()`）；
- “sum”：创建对比度相加为零的列，（`?stats::conc.sum()`）

新创建的列按模式 `[column-name].[x]` 命名，其中 `x` 是“one-hot”和“treatment”编码的各自因子水平，否则是一个整数序列。

```
task = tsk("penguins") #企鹅数据集
```

```
poe = po("encode", method = "one-hot") # 独热编码
```

```
poe = po("encode", method = "treatment") # 虚拟编码
```

```
poe = po("encode", method = "helmert") # Helmert 编码
```

```
poe = po("encode", method = "poly") # 多项编码
```

```
poe = po("encode", method = "sum") # sum 编码
```

```
newtask = poetrain(list(task))[[1]]
```

```

1 task = tsk("penguins")
2 poe = po("encode", method = "one-hot") # 独热编码
3 poe = po("encode", method = "treatment") # 虚拟编码
4 poe = po("encode", method = "helmert") # Helmert 编码
5 poe = po("encode", method = "poly") # 多项编码
6 poe = po("encode", method = "sum") # sum 编码
7 newtask = poe$train(list(task))[[1]]

```

#### (4) 效应编码

条件目标效应编码，对因子（也包括有序因子、字符列）进行编码：

- 分类任务的效应编码将每个因子列的因子水平转换为给定该水平的目标条件对数似然与全局对数似然之差；**（相对于目标的效应，提前把因变量的影响拿进来）**
- 回归任务的效应编码将每个因子列的因子水平转换为给定该水平的目标条件均值与目标全局均值之差；**（相对于目标的效应，提前把因变量的影响拿进来）**

在预测过程中，将未出现的水平视为缺失值。

```

1 poe = po("encodeimpact")
2 newtask = poe$train(list(task))[[1]]

```

**注：**回归任务中也有另一种简单的效应编码，比如预测房价时，将小区编码为该小区的房价中位数。

简单，不增加特征。把目标的信息用上了，容易出现目标泄露问题，过拟合。作弊。

## 5. 处理类不平衡

类不平衡问题是指在分类任务中，不同类别的样本数量存在明显的不平衡情况，即某些类别的样本数量远远多于或远远少于其他类别的样本数量，比如信用卡违约数据中，不违约的人比违约的人往往要多很多。

```
1 task = tsk("german_credit")
2 table(task$truth())
3 #>
4 #> good bad
5 #> 700 300
```

这种不平衡可能会对机器学习模型的性能产生负面影响。通过采样对任务进行类平衡处理，可能有利于提升模型的预测性能。

注意，正确的做法是在训练集上处理类不平衡问题，即原始数据划分为训练集+ 测试集，对训练集做类平衡化处理然后拟合模型，再在测试集上评估模型的泛化性能。

### (1) 欠采样与过采样

- 欠采样：只保留多数类的一部分行；
- 过采样：对少数类进行超量采样（重复数据点）。

使用 `po("classbalancing")` 实现，其参数：

- `ratio`：相对于 `$reference` 值，要保留的类的行数的比率，默认为1；

- reference: \$ratio 的值是根据什么来衡量的, 可选"all" (默认值, 所有类的平均观测数), "major" (最多数类的观测数), "minor" (最少数类的观测数) 等;
- adjust: 哪些类要欠/过采样, 可选"all" (默认)、"major"、"minor"、"upsample" (只过采样) 和"downsample" (只欠采样) 等;
- shuffle: 是否对结果任务的行进行洗牌, 默认为 TRUE。

## 欠采样案例

```
opb_down = po("classbalancing", reference="minor",
adjust = "major")
```

#默认ratio=1,若ratio= 2,结果是600good,300bad

```
newtask = opb_down$train(list(task))[[1]]
```

```
table(newtask$truth())
```

```

1  opb_down = po("classbalancing", reference = "minor",
2  adjust = "major")
3  # 默认 ratio = 1, 若 ratio = 2, 结果是 600 good, 300 bad
4  newtask = opb_down$train(list(task))[[1]]
5  table(newtask$truth())
6  #>
7  #> good bad
8  #> 300 300
```

## 过采样

```
opb_up = po("classbalancing", reference= "major",  
adjust= "minor")  
  
#默认ratio=1,若ratio= 2,结果是700good,1400bad  
  
newtask =opb_up$train(list(task))[[1]]  
  
table(newtask$truth())
```

```
1  opb_up = po("classbalancing", reference = "major",  
2  adjust = "minor")  
3  # 默认 ratio = 1, 若 ratio = 2, 结果是 700 good, 1400 bad  
4  newtask = opb_up$train(list(task))[[1]]  
5  table(newtask$truth())  
6  #>  
7  #> good bad  
8  #> 700 700
```

## (2) SMOTE 法

用 SMOTE 算法创建少数类别的合成观测，生成一个更平衡的数据集。该算法为每个少数类观测取样，基于该观测的  $K$  个最近邻居生成新观测。**它只能应用于具有纯数值特征的任务。重新伪造一些观测值**

使用 `po("smote")` 实现 (`?smotefamily::SMOTE()`)，其参数：

- $K$ ：用于抽取新值的近邻的数量；
- `dup_size`：合成的少数实例在原始多数实例数量上的期望次数。

# 只支持 double 型特征, 需安装 `smotefamily` 包

```
pop = po("colapply", applicator = as.numeric,
```



```

affect_columns = selector_type("integer")) %>>%
po("encodeimpact") %>>%

po("smote", K = 5, dup_size = 1) # 少数类增加 1 倍

newtask = pop$train(task)[[1]]

table(newtask$truth())

```

```

1 # 只支持 double 型特征, 需安装 smotefamily 包
2 pop = po("colapply", applicator = as.numeric,
3 affect_columns = selector_type("integer")) %>>% #整数
  转double型
4 po("encodeimpact") %>>% #把因子型效应编码
5 po("smote", K = 5, dup_size = 1) # 少数类增加 1 倍
6 newtask = pop$train(task)[[1]]
7 table(newtask$truth())
8 #>
9 #> good bad
10 #> 700 600 #相对平衡

```

### (3) 样本加权

处理类不平衡问题的另一种方法是将少数类别中的样本点权重加大，这可能会提高模型对该类别的预测性能。 **#在tidymodel框架下支持的更好**

样本加权，可以先根据目标列计算一列权重，再将其列角色设置为"weight"来实现：

```
dat = task$data()
```

```
dat$w = ifelse(dat$credit_risk == "good", 1, 2)
```

```
newtask = as_task_classif(dat, target = "credit_risk")
```

```
newtask$set_col_roles("w", roles = "weight")
```

```
newtask
```

```
1  dat = task$data()
2  dat$w = ifelse(dat$credit_risk == "good", 1, 2)
3  newtask = as_task_classif(dat, target = "credit_risk")
4  newtask$set_col_roles("w", roles = "weight")
5
6  newtask  # 多了一个权重w特征
7  #> <TaskClassif:dat> (1000 x 21)
8  #> * Target: credit_risk
9  #> * Properties: twoclass, weights
10 #> * Features (20):
11 #> - fct (14): credit_history, employment_duration, fo
    rei
12 #> housing, job, other_debtors, other_installment_plan
    s
13 #> people_liable, personal_status_sex, property, purpo
    s
14 #> status, telephone
15 #> - int (3): age, amount, duration
16 #> - ord (3): installment_rate, number_credits, presen
    t_r
17 #> * Weights: w
```

## 6. 目标变换

为了提高预测能力，对于异方差或右偏的因变量数据，经常需要做取对数变换、或 Box-Cox 变换，以变成正态数据，再进行回归建模，预测值要回到原数据量级，还要做逆变换。太麻烦了，之间用ppl方便！

ppl("targettrafo", graph), 是预定义的目标变换的 Graph，其参数 targetmutate.trafo 用于在训练过程中对目标变量进行变换，

targetmutate.inverter 用于在预测过程中对原变换进行逆变换。这样构建的图学习器还能用于更复杂的重抽样或基准测试。

示例：对目标做取对数变换

```
task = tsk("mtcars")

gr_log = ppl("targettrafo", graph=ln("regr.lm"),
targetmutate.trafo = \(x)log(x),
targetmutate.inverter = \(x)
list(response =exp(x$response)))

gr_log$plot(horizontal= TRUE)
```

```
1 task = tsk("mtcars")
2 gr_log = ppl("targettrafo", graph = ln("regr.lm"),
3 targetmutate.trafo = \(x) log(x),
4 targetmutate.inverter = \(x)
5 list(response = exp(x$response))) # x$response指x的响应
6 gr_log$plot(horizontal = TRUE)
```



```
gl = as_learner(gr_log)
```

```
gl$train(task)
```

```
gl$predict(task)
```

```
1 gl = as_learner(gr_log)
2 gl$train(task)
3 gl$predict(task)
4 #> <PredictionRegr> for 32 observations:
5 #> row_ids truth response
6 #> 1 21.0 21.7
7 #> 2 21.0 21.1
8 #> 3 22.8 25.7
9 #> ---
10 #> 30 19.7 19.6
11 #> 31 15.0 14.1
12 #> 32 21.4 23.1
13 # 预测的结果跟原来一样
```

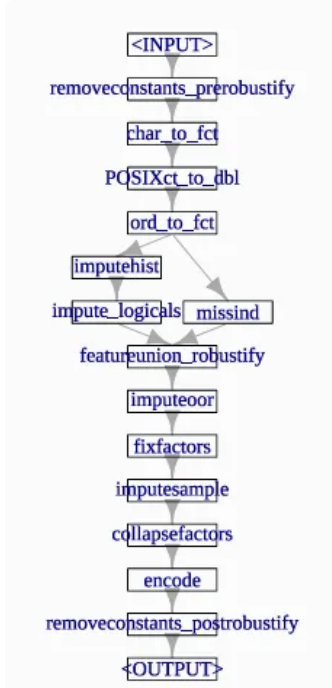
最后注：mlr3pipelines提供了一个稳健化、自动预处理的预定义图 `ppl("robustify")`，包含了适用于大多数缺失值插补和因子编码等场景的默认处理，还有可选参数 `task` 和 `learner` 以针对专门任务和学习器：

(1) `po("removeconstants")`：删除常数特征。

(2) `po("colapply")`：将字符和有序特征编码为分类特征，并将日期/时间特征编码为数值特征。

- (3) `po("imputehist")`: 通过直方图采样对数值特征进行插补。
- (4) `po("imputesample")`: 通过从经验分布中采样来插补逻辑特征– 这仅影响 `$predict()`–步骤。
- (5) `po("missind")`: 添加缺失值指示器以用于插补的数值和逻辑变量。
- (6) `po("imputeoor")`: 使用新水平对分类特征的缺失值进行编码。
- (7) `po("fixfactors")`: 修复分类特征的水平, 使得在预测和训练过程中存在相同的水平 (可能涉及丢弃空因子水平) 。
- (8) `po("imputesample")`: 通过从经验分布中采样来插补由前一步骤中的水平丢弃引入的分类特征的缺失值。
- (9) `po("collapsefactors")`: 折叠分类特征的水平 (从训练数据中最稀有的因子开始), 直到水平数量少于 `max_cardinality` 参数所控制的特定数量 (具有保守的默认值 1000) 。
- (10) `po("encode")`: 对分类特征进行独热编码。
- (11) `po("removeconstants")`: 删除可能在前面步骤中创建的常数特征。

```
graph = ppl("robustify") # 自动化的处理
graph$plot()
```



简言之，不能直接用于学习器的数据经过该图管道的自动处理将可用于学习器，甚至不弱于自己一般性的手动预处理。

## 参考文献

(Bernd Bischl and Lang, 2024), (Marc Becker, 2021),

(Bernd Bischl, 2021), (Martin Binder, 2022)。

Bernd Bischl, e. a. (2021). Machine Learning Pipelines in R.

Bernd Bischl, Raphael Sonabend, L. K. and Lang, M., editors

(2024). Applied Machine Learning Using mlr3 in R. CRC Press.

Marc Becker, e. a. (2021). mlr3 gallery.

Martin Binder, e. a. (2022). mlr3pipelines: Preprocessing

Operators and Pipelines for 'mlr3'. version 0.4.2.

Molnar, C. (2022). Interpretable Machine Learning. 2 edition.

