R 机器学习

第 05 讲　特征工程

张敬信

2022 年 10 月 23 日

哈尔滨商业大学

自变量通常称为特征，**特征工程**（Feature Engineering），就是发现或构建对因变量有明显影响作用的特征，具体来说是将原始特征转化成更好的表达问题本质的特征的过程，将这些特征运用到预测模型中能提高对不可见数据的模型预测精度。

数据清洗、特征工程属于机器学习中的数据预处理环节，其实现是用 `mlr3verse` 框架下的管道包：`mlr3pipelines`.

```
library(mlr3verse)
```

- 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;
- 很多 PipeOp 都支持 affect_columns 参数, 接受 Selector 选择器函数以选择该 PipeOp 影响的列

Selector 选择器函数（接受 Task）有：

- selector_all(): 选择所有特征；
- selector_none(): 不选择任何特征；
- selector_type(types): 根据类型来选择特征：“logical”, “integer”, “numeric”, “character”, “factor”, “ordered”, “POSIXct”；
- selector_grep(pattern): 根据正则表达式匹配（特征名字）选择特征；
- selector_name(feature_names): 根据特征名字选择特征；
- selector_invert(selector): 选择某选择器的反选特征；
- selector_intersect/union/setdiff(selector_x, selector_y): 选择两个选择器特征的交/并/差集；
- selector_missing(): 选择包含缺失值的特征；
- selector_cardinality_greater_than(min_cardinality): 选择分类特征的基数大于指定阈值的特征

```
po()  # 查看所有`PipeOp`, 获取帮助: ?PipeOp
#> <DictionaryPipeOp> with 64 stored values
#> Keys: boxcox, branch, chunk, classbalancing, classifavg, c
#>   colapply, collapsefactors, colroles, copy, datefeatures,
#>   encodeimpact, encodelmer, featureunion, filter, fixfacto
#>   ica, imputeconstant, imputehist, imputelearner, imputeme
#>   imputemedian, imputemode, imputeoor, imputesample, kerne
#>   learner, learner_cv, missind, modelmatrix, multiplicitye
#>   multiplicityimply, mutate, nmf, nop, ovrsplit, ovrunite,
#>   quantilebin, randomprojection, randomresponse, regravg,
#>   removeconstants, renamecolumns, replicate, scale, scalem
#>   scalerange, select, smote, spatialsign, subsample, targe
#>   targetmutate, targettrafoscalerange, textvectorizer, thr
#>   tunethreshold, unbranch, vtreat, yeojohnson
```

```
gr = po("scale") %>>% po("pca", rank. = 2)
gr$plot()
```

特征工程管道的两种用法：

(1) 调试：查看特征工程步对输入数据做了什么

- 训练特征工程管道：提供任务，访问特征工程之后的数据：

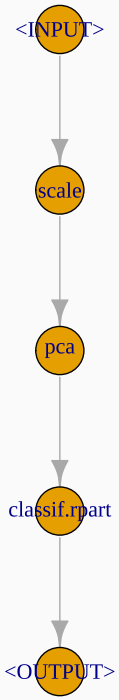
```
gr$train(tsk("iris"))[[1]]$data()
#>         Species    PC1     PC2
#>    1:    setosa -2.257 -0.4784
#>    2:    setosa -2.074  0.6719
#>    3:    setosa -2.356  0.3408
#>    4:    setosa -2.292  0.5954
#>    5:    setosa -2.382 -0.6447
#>   ---
#> 146: virginica  1.864 -0.3857
#> 147: virginica  1.559  0.8937
#> 148: virginica  1.516 -0.2682
#> 149: virginica  1.368 -1.0079
#> 150: virginica  0.957  0.0243
```

- 将训练好的特征工程管道，用于新数据：

```
gr$predict(tsk("iris")$filter(1:5))[[1]]$data()
#>    Species   PC1    PC2
#> 1:  setosa -2.26 -0.478
#> 2:  setosa -2.07  0.672
#> 3:  setosa -2.36  0.341
#> 4:  setosa -2.29  0.595
#> 5:  setosa -2.38 -0.645
```

(2) 用于机器学习：再接一个学习器，转化成图学习器（和普通学习器一样使用）

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



```
gr_learner = as_learner(gr)
gr_learner
#> <GraphLearner:scale.pca.classif.rpart>
#> * Model: -
#> * Parameters: scale.robust=FALSE, pca.rank.=2, classif.rpa
#> * Packages: mlr3, mlr3pipelines, rpart
#> * Predict Types:  [response], prob
#> * Feature Types: logical, integer, numeric, character, fac
#>   POSIXct
#> * Properties: featureless, hotstart_backward, hotstart_for
#>   importance, loglik, missings, multiclass, oob_error,
#>   selected_features, twoclass, weights
```

一. 缺失值插补

- 目前支持的插补方法:

```
as.data.table(mlr_pipeops)[tags %in% "missings", list(key)]
#>                key
#> 1: imputeconstant
#> 2:     imputehist
#> 3:  imputelearner
#> 4:     imputemean
#> 5:   imputemedian
#> 6:     imputemode
#> 7:      imputeoor
#> 8:   imputesample
```

1. 简单插补

- 用某常数、均值、中位数、众数插补。

```
task = tsk("pima")
task$missings()
#> diabetes      age  glucose  insulin     mass pedigree preg
#>        0        0        5      374       11        0
#>  triceps
#>      227
po = po("imputeconstant", param_vals = list(
  constant=-999, affect_columns=selector_name("glucose")))
new_task = po$train(list(task = task))[[1]]
new_task$missings()
#> diabetes      age  insulin     mass pedigree pregnant pres
#>        0        0      374       11        0        0
#>  glucose
#>        0
```

```
po = po("imputemean")
# po("imputemedian")
# po("imputemode")
new_task = po$train(list(task))[[1]]
new_task$missings()
#> diabetes      age pedigree pregnant  glucose  insulin
#>        0        0        0        0        0        0
#>  triceps
#>        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")
new_task = po$train(list(task = task))[[1]]
new_task$missings()
#> diabetes      age pedigree pregnant  glucose  insulin
#>        0        0        0        0        0        0
#>  triceps
#>        0
```

3. 直方图法插补

- 用直方图法插补数值特征

```
po = po("imputehist")
new_task = po$train(list(task = task))[[1]]
new_task$missings()
#> diabetes      age pedigree pregnant  glucose  insulin
#>        0        0        0        0        0        0
#>  triceps
#>        0
```

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。

使用参数 `context_columns` 所指示的特征作为特征来训练插补学习器。

只有学习器支持的特征可以被插补；即 `regr` 类型的学习器只能插补整数和数值特征，而 `classif` 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，或者需要做自我插补。

```
po = po("imputelearner", lrn("regr.rpart"))
new_task = po$train(list(task = task))[[1]]
new_task$missings()
#> diabetes      age pedigree pregnant  glucose  insulin
#>        0        0        0        0        0        0
#>  triceps
#>        0
po = po("imputelearner",
  po("imputehist") %>>% lrn("regr.kknn"))
new_task = po$train(list(task = task))[[1]]
new_task$missings()
#> diabetes      age pedigree pregnant  glucose  insulin
#>        0        0        0        0        0        0
#>  triceps
#>        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。

用 `min(x)-offset-multiplier*diff(range(x))` 或 `max(x)+offset+multiplier*diff(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。

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

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

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。

(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:

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

其中，μ 为均值，σ 为标准差。Z 值反映了该值偏离均值的标准差的倍数。

```
pos = po("scale")      # 参数 scale = FALSE, 只做中心化
pos$train(list(task))[[1]]$data()
```

注：中心化后，0 就代表均值，更方便模型解释。

(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", param_vals = list(lower=0, upper=1))
pop$train(list(task))[[1]]$data()
```

(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 L_2 范数：

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

```
pop = po("spatialsign")
pop$train(list(task))[[1]]$data()
```

(1) 非线性特征

对于数值特征 `x1, x2, ...`, 可以创建更多的多项式项特征: `x1^2, X1 x2, x2^2, ...`, 这相当于是用自变量的更高阶泰勒公式去逼近因变量。

管道运算"modelmatrix", 再结合如下的 `formula` 模型公式语法:

- `y ~ .`: 包含所有自变量的主效应
- `x1:x2`: 交互效应, 即 $x_1 x_2$ 项
- `x1*x2`: 包含全部主效应和交互效应, `x1 + x2 + x1:x2` 的简写
- `I()`: 打包式子作为整体
- `y ~ poly(x, 2, raw = TRUE)`: 一元二次多项式回归, 同 `y ~ x + I(x^2)`
- `y ~ polym(x1, x2, degree = 2, raw = TRUE)`: 二元二次多项式回归
- `log(y) ~ x`: 对 `y` 做对数变换

```
pop = po("modelmatrix",
         formula = ~ . ^ 2 + I(x1 ^ 2) + log(x2))
pop$train(list(task))[[1]]$data()
```

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

(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,
  Area1 = ~ x1 * x2,
  Area2 = ~ x3 * x4,
  Area = ~ Area1 + Area2
))
pom$train(list(task))[[1]]$data()
```

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 = ~ slide_dbl(x1, mean, .before = 2, .after = 2)
))
dat = pom$train(list(task = task))[[1]]$data()
```

注：其它平滑法还有指数平滑、滤波、光滑样条等。

另外，还有：

- “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)
```

- “renamecolumns”，修改列名

```
pop = po("renamecolumns",
         param_vals = list(renaming=c("Petal.Length"="PL")))
```

(3) 正态性变换

- Box-Cox 变换

非常神奇的正态性变换，用最大似然估计选择最优的 λ 值，让非负的非正态数据变成正态数据：

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

```
pop = po("boxcox")
pop$train(list(task))[[1]]$data()
```

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：

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

```
pop = po("yeojohnson")
pop$train(list(task))[[1]]$data()
```

注： 使用 bestNormalize 包，也可逆变换回到原数据量级。

(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：

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

- 分位数分箱

```
pop = po("quantilebin", numsplits = 4)
pop$train(list(task))[[1]]$data()
```

注：其他基于模型的分箱方法还有，基于 k-means 聚类、决策树、ROC 曲线、广义可加模型、最大秩统计量等。

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，可以用降维技术进压缩特征，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$train(list(tsk("iris")))[[1]]$data()
#>         Species   PC1     PC2
#>    1:    setosa -2.68  0.3194
#>    2:    setosa -2.71 -0.1770
#>    3:    setosa -2.89 -0.1449
#>    4:    setosa -2.75 -0.3183
#>    5:    setosa -2.73  0.3268
#>   ---
#> 146: virginica  1.94  0.1875
#> 147: virginica  1.53 -0.3753
#> 148: virginica  1.76  0.0789
#> 149: virginica  1.90  0.1166
#> 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, KPCA）可实现数据的非线性降维，用于处理线性不可分的数据集。

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

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

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

(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。

```
po = po("removeconstants")
po$train(list(task_ex))[[1]]$data()
```

4. 分类特征

(1) 因子折叠

管道运算"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；
- 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。

```
poc = po("collapsefactors", target_level_count = 5)
poc$train(list(task))[[1]]$data()
```

(2) 因子修正

管道运算 “fixfactors”（参数：droplevels = TRUE）确保预测过程中的因子水平与训练过程中的相同；可能会在之前丢弃空的训练因子水平。

注意，如果发现未见过的因子水平，这可能会在预测期间引入缺失值。

```
op = po("fixfactors")
op$train(list(tasktrain))
op$predict(list(tasktest))[[1]]$data()
```

(3) 因子编码

对因子、有序因子、字符特征进行编码。参数 `method` 指定编码方法：

- “one-hot”：独热编码；
- “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（查阅 `stats::conc.treatment()`）；
- “helmert”：根据 Helmert 对比度创建列（查阅 `stats::conc.helmert()`）；
- “poly”：根据正交多项式创建对比列（查阅 `stats::conc.poly()`）；
- “sum”：创建对比度相加为零的列，（查阅 `stats::conc.sum()`）

```
poe = po("encode", method = "one-hot")  # 独热编码
poe$train(list(task))[[1]]$data()
poe = po("encode", method = "treatment")  # 虚拟编码
poe$train(list(task))[[1]]$data()
```

(4) 效应编码

条件目标效应编码，对因子、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；
- 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。

在预测过程中，将未出现的水平视为缺失值。

```
poe = po("encodeimpact")
poe$train(list(task))[[1]]$data()
```

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。

另外，类似的效应编码还有随机截距模型编码：`po("encodelmer")`.

5. 日期时间特征

基于数据的 `POSIXct` 列（若没有则不计算），计算出一组与日期相关的特征，并添加到输出任务的特征集中。

该功能是基于 `fastai` 包中的 `add_datepart()` 和 `add_cyclic_datepart()` 函数。

日期时间特征包含了年、月、日、时、分、秒，以及年中的一周，星期中的一天等大量信息，这些信息大多具有周期性，比如一天中的 11 点和 12 点相差一小时，但一天的 23 点和下一天的 0 点也相差一小时。

要处理这种周期性，就需要设置 `cyclic = TRUE` 以计算循环特征。其方法是计算特征 $\frac{2\pi x}{x_{max}}$ 的正弦和余弦变换，若是小时，则 $x_{max} = 24$, 若是分钟和秒，则是 60.

```
dat = tsk("bike_sharing")$data()
dat[, date := as.POSIXct(paste0(date, " ", hour, ":00:00"),
                         tz = "GMT",
                         format = "%Y-%m-%d %H:%M:%S")]
task = as_task_regr(dat, target = "count")
pipeop_date = po("datefeatures", cyclic = TRUE,
                 minute = FALSE, second = FALSE)
pipeop_date$train(list(task))[[1]]$data() %>% colnames()
#>  [1] "count"                  "apparent_temperature"  "holi
#>  [4] "hour"                   "humidity"              "mont
#>  [7] "season"                 "temperature"           "weat
#> [10] "weekday"                "windspeed"             "work
#> [13] "year"                   "date.year"             "date
#> [16] "date.week_of_year"      "date.day_of_year"      "date
#> [19] "date.day_of_week"       "date.hour"             "date
#> [22] "date.month_sin"         "date.month_cos"        "date
#> [25] "date.week_of_year_cos" "date.day_of_year_sin"  "date
```

6. 文本特征

使用 quanteda 包的 dfm(), dfm_trim(), dfm_tfidf() 等函数，将文本特征列，转化为词袋表示。

一般流程包括：

- 用 quanteda::tokens 对文本进行分词；
- 用 quanteda::tokens_ngrams 计算出 Ngrams；
- 用 quanteda::dfm 计算文档频率矩阵；
- 在训练期间，使用 quanteda::dfm_trim 对文档频率矩阵进行修剪；
- 若 scheme_df 未被设置为"unary"，对文档频率矩阵重新加权。

注：目前，仍然更建议直接在 quanteda 框架下做文本挖掘。

```r
library(quanteda)
library(readtext)
docs = readtext("job/*.txt", docvarsfrom = "filenames",
                encoding = "UTF-8")
task = as_task_regr(docs, target = "docvar1")
po_text = po("textvectorizer", param_vals = list(
    stopwords_language = "en",    # 不支持"ch"
    scheme_df = "inverse",
    remove_punct = TRUE,
    remove_symbols = TRUE,
    remove_numbers = TRUE),
  affect_columns = selector_name("text"))
dat = po_text$train(list(task))[[1]]$data()
```

三. 处理不均衡数据

通过采样对任务进行类平衡，可能有利于对不平衡的训练数据进行分类，采样只发生在训练阶段。

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）

管道运算"classbalancing"，其参数有：

- `reference`：设置参照类，可以是"all"（默认，所有类的平均），"major"（最大类），"minor"（最小类），"nonmajor"，"nonminor"，"one"；
- `ratio`：相对于参照类，保留类的行数比率，默认为 1。
- `adjust`：哪些类要向上/向下采样。可以是"all"（默认）、"major"、"minor"、"nonmajor"、"nonminor"、"upsample"（只过采样）和"downsample"（只欠采样）。
- `shuffle`：是否对结果行进行随机重排序，默认为 TRUE。

```r
task = tsk("german_credit")
table(task$truth())
#>
#> good  bad
#>  700  300
## 欠采样
opb_down = po("classbalancing", reference = "minor",
              adjust = "major")
# 默认 ratio = 1, 若 ratio = 2, 结果是 600 good, 300 bad
result = opb_down$train(list(task))[[1]]
table(result$truth())
#>
#> good  bad
#>  300  300
```

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

2. SMOTE 法

用 SMOTE 算法创建少数类别的合成实例，生成一个更平衡的数据集。该算法为每个少数群体实例取样，基于该数据点的 K 个最近的邻居的新数据点。它只能应用于具有纯数值特征的任务。

管道运算"smote"，其参数有：

- 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

```r
# 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
result = pop$train(task)[[1]]
table(result$truth())
#>
#> good  bad
#>  700  600
```

四. 目标变换

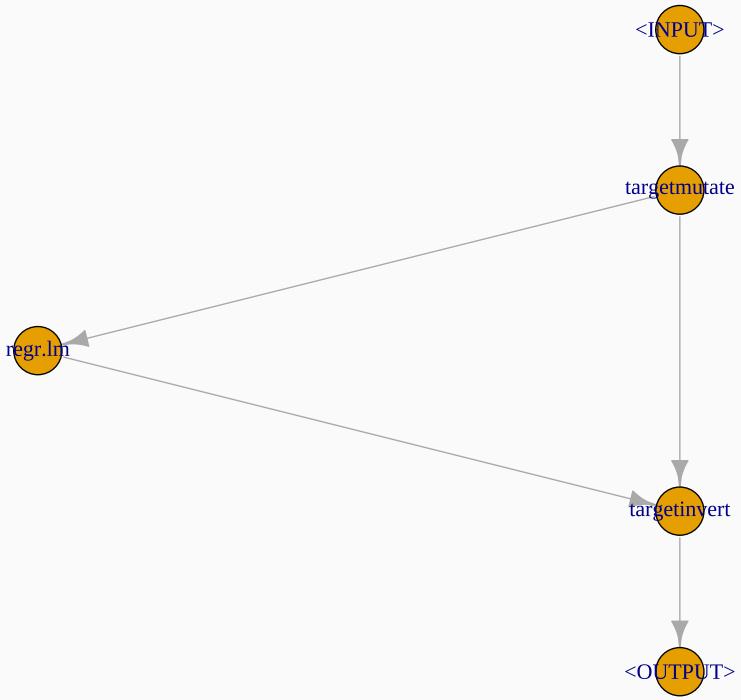
为了提高预测能力，对于异方差或右偏的因变量数据，经常需要做取对数变换、或 Box-Cox 变换，以变成正态数据，再进行回归建模，预测值要回到原数据量级，还要做逆变换。

mlr3pipelines 包还提供了目标变换管道，将目标变换及其逆变换，都封装到图学习器，这样就省了手动做两次变换，还能整体进行调参或基准测试。

以对数变换为例：

```
task = tsk("mtcars")
learner = lrn("regr.lm")
g_ppl = ppl("targettrafo", graph = learner)
g_ppl$param_set$values$targetmutate.trafo =
  function(x) log(x)
g_ppl$param_set$values$targetmutate.inverter =
  function(x) list(response = exp(x$response))
```

```
g_ppl$plot()
```



```
gl = as_learner(g_ppl)
gl$train(task)
gl$predict(task)
#> <PredictionRegr> for 32 observations:
#>     row_ids truth response
#>           1  21.0     21.7
#>           2  21.0     21.1
#>           3  22.8     25.7
#> ---
#>          30  19.7     19.6
#>          31  15.0     14.1
#>          32  21.4     23.1
```

另外，对目标做 Box-Cox 变换，mlr3 官方博文提供了实现。

本篇主要参阅 (张敬信, 2022), (锡南·厄兹代米尔, 2019), (Hadley Wickham, 2017), 以及 `mlr3pipelines` 包文档，模板感谢 (黄湘云, 2021), (谢益辉, 2021).

参考文献

Hadley Wickham, G. G. (2017). *R for Data Science*. O' Reilly, 1 edition. ISBN 978-1491910399.

张敬信 (2022). *R 语言编程：基于 tidyverse*. 人民邮电出版社, 北京.

谢益辉 (2021). *rmarkdown: Dynamic Documents for R*.

锡南·厄兹代米尔, 迪夫娅·苏萨拉, [?]. (2019). 特征工程入门与实践. 人民邮电出版社, 北京.

黄湘云 (2021). *Github: R-Markdown-Template*.