ダミー化と補間いろいろ

プログラマたん bot

2021/01/17

ダミー化には、R と Python でいろいろな方法があるので試す。

R でダミー化して補間する

参考資料を元に、いろいろなパッケージの、いろいろなパラメータでダミー化する。

各パッケージの動作をまとめた、1-to-N または 1-to-N-1 ダミー化は、最初に書いたのがパッケージのデフォルト値である。NA を NA のままにするか、NA というカテゴリを作るかも、両方できる場合は最初に書いたのがパッケージのデフォルト値である。

パッケージ	1-to-N または 1-to-N-1 ダミー化	NA(欠測値)
caret::dummyVars	N または N-1	NA のまま
dummies::dummy.data.frame	N-1 のみ	NA のまま
makedummies::makedummies	N-1 または N	NA のまま
fastDummies::dummy_cols	N または N-1	NA というカテゴリを作る/作らない
pandas.get_dummies	N または N-1	NA のまま/NA というカテゴリを作る

データフレームをロードする

列の型は、n が numeric、f が factor である。列名は以下の通り、順に並ぶ。

- rank は、1 を先頭とする行番号
- r, p, q, s はカテゴリ (factor)。NA を含んでいたりいなかったりする。列名はアルファベット順ではない。
- value は数値。NA を含んでいたりいなかったりする。列名はアルファベット順ではない。

```
df <- readr::read_csv('with_na.csv', col_types = 'nfnfnffn')
```

rank	r	value_b	p	value_a	q	s	value_c
1	a	11	e	NA	g	NA	30
2	a	NA	NA	22	g	j	30
3	a	13	e	23	NA	j	30
4	b	14	f	24	NA	k	30
5	b	NA	NA	25	h	k	30
6	b	16	f	NA	h	NA	30

caret::dummyVars

N カテゴリを N 変数にする (全カテゴリを足すと 1s になる)。tibble の NA は NA のまま残り、「NA というカテゴリ」は作らない。ダミー化前の列はなくなり、元々の列名の順序は維持される。

```
mat_predict <- predict(caret::dummyVars(~., df), df)
```

rank	r.a	r.b	value_b	p.e	p.f	value_a	q.g	q.h	s.j	s.k	value_c
1	1	0	11	1	0	NA	1	0	NA	NA	30
2	1	0	NA	NA	NA	22	1	0	1	0	30
3	1	0	13	1	0	23	NA	NA	1	0	30
4	0	1	14	0	1	24	NA	NA	0	1	30
5	0	1	NA	NA	NA	25	0	1	0	1	30
6	0	1	16	0	1	NA	0	1	NA	NA	30

fullRank = TRUE にすると、N カテゴリを N-1 変数にする。

```
mat_predict <- predict(caret::dummyVars(~., df, fullRank = TRUE), df)
```

rank	r.b	value_b	p.f	value_a	q.h	s.k	value_c
1	0	11	0	NA	0	NA	30
2	0	NA	NA	22	0	0	30
3	0	13	0	23	NA	0	30
4	1	14	1	24	NA	1	30
5	1	NA	NA	25	1	1	30
6	1	16	1	NA	1	NA	30

以下の code chunk を cache = TRUE にしなおかつ `df_large <- NULL, mat_predict_large <- NULL` でメモリを解放しないと、PC の DRAM が 8GB だと lazyLoadDBinsertVariable できないというエラーが発生する。

```
n_cols <- 10000
n_samples <- 10000
df_large <- purrr::map(1:n_cols, function(x) {
    v <- list(a = sample(c("p", "q", "r"), n_samples, replace = TRUE))
    names(v) <- paste0('V', x)
    v
}) %>% dplyr::bind_cols()
mat_predict_large <- predict(caret::dummyVars(~., df_large), df_large)
print(pryr::object_size(df_large))
```

```
## 801 MB
```

```
print(pryr::object_size(mat_predict_large))
```

```
## 2.4 GB
```

n_cols <- 20000 にすると、プロテクションスタックがあふれる。

dummies::dummy.data.frame

N カテゴリを N-1 変数にする (baseline カテゴリからの差になる)。tibble の NA は NA のまま残り、NA というカテゴリは作らない。ダミー化前の列はなくなり、元々の列名の順序は維持される。

```
mat_dummies <- dummies::dummy.data.frame(df)
```

rank	r	value_b	p	value_a	q	s	value_c
1	a	11	e	NA	g	NA	30
2	a	NA	NA	22	g	j	30
3	a	13	e	23	NA	j	30
4	b	14	f	24	NA	k	30
5	b	NA	NA	25	h	k	30
6	b	16	f	NA	h	NA	30

makedummies::makedummies

basal_level 引数で、N カテゴリを N 変数にするか N-1 変数にするか指定できる。まず N 変数にする。tibble の NA は NA のまま残り、NA というカテゴリは作らない。ダミー化前の列はなくなり、元々の列名の順序は維持される。つまり caret と同様の動作をする。

```
mat_makedummies_n <- makedummies::makedummies(df, basal_level = TRUE)
```

rank	r_a	r_b	value_b	p_e	p_f	value_a	q_g	q_h	s_j	s_k	value_c
1	1	0	11	1	0	NA	1	0	NA	NA	30
2	1	0	NA	NA	NA	22	1	0	1	0	30
3	1	0	13	1	0	23	NA	NA	1	0	30
4	0	1	14	0	1	24	NA	NA	0	1	30
5	0	1	NA	NA	NA	25	0	1	0	1	30
6	0	1	16	0	1	NA	0	1	NA	NA	30

basal_level = FALSE にすると、N カテゴリを N-1 変数にする。デフォルトは N-1 変数である。

```
mat_makedummies_n_1 <- makedummies::makedummies(df, basal_level = FALSE)
```

rank	r	value_b	p	value_a	q	s	value_c
1	0	11	0	NA	0	NA	30
2	0	NA	NA	22	0	0	30
3	0	13	0	23	NA	0	30
4	1	14	1	24	NA	1	30
5	1	NA	NA	25	1	1	30
6	1	16	1	NA	1	NA	30

fastDummies::dummy_cols

remove_first_dummy 引数で、N カテゴリを N 変数にするか N-1 変数にするか指定できる。まず N 変数にする。NA を含む factor には、NA というカテゴリができる。

```
mat_fastDummies_n <- fastDummies::dummy_cols(df, remove_first_dummy = FALSE)
```

rank	r	value_b	p	value_a	q	s	value_c	r_a	r_b	p_e	p_f	p_NA	q_g	q_h	q_NA
1	a	11	e	NA	g	NA	30	1	0	1	0	0	1	0	0
2	a	NA	NA	22	g	j	30	1	0	NA	NA	1	1	0	0
3	a	13	e	23	NA	j	30	1	0	1	0	0	NA	NA	1
4	b	14	f	24	NA	k	30	0	1	0	1	0	NA	NA	1
5	b	NA	NA	25	h	k	30	0	1	NA	NA	1	0	1	0
6	b	16	f	NA	h	NA	30	0	1	0	1	0	0	1	0

ignore_na = TRUE にすると、NA というカテゴリは作らない。

```
mat_fastDummies_n_na <- fastDummies::dummy_cols(df, remove_first_dummy = FALSE, ignore_na = TRUE)
```

rank	r	value_b	p	value_a	q	s	value_c	r_a	r_b	p_e	p_f	q_g	q_h	s_j	s_k
1	a	11	e	NA	g	NA	30	1	0	1	0	1	0	NA	NA
2	a	NA	NA	22	g	j	30	1	0	NA	NA	1	0	1	0
3	a	13	e	23	NA	j	30	1	0	1	0	NA	NA	1	0
4	b	14	f	24	NA	k	30	0	1	0	1	NA	NA	0	1
5	b	NA	NA	25	h	k	30	0	1	NA	NA	0	1	0	1
6	b	16	f	NA	h	NA	30	0	1	0	1	0	1	NA	NA

remove_first_dummy = TRUE にして、N カテゴリを N-1 変数にする。ignore_na = FALSE にすると NA というカテゴリができ、ignore_na = TRUE にすると、NA というカテゴリは作らない。

```
mat_fastDummies_n_1 <- fastDummies::dummy_cols(df, remove_first_dummy = TRUE, ignore_na = FALSE)
```

rank	r	value_b	p	value_a	q	s	value_c	r_b	p_f	p_NA	q_h	q_NA	s_k	s_NA
1	a	11	e	NA	g	NA	30	0	0	0	0	0	NA	1
2	a	NA	NA	22	g	j	30	0	NA	1	0	0	0	0
3	a	13	e	23	NA	j	30	0	0	0	NA	1	0	0
4	b	14	f	24	NA	k	30	1	1	0	NA	1	1	0
5	b	NA	NA	25	h	k	30	1	NA	1	1	0	1	0
6	b	16	f	NA	h	NA	30	1	1	0	1	0	NA	1

```
mat_fastDummies_n_1_na <- fastDummies::dummy_cols(df, remove_first_dummy = TRUE, ignore_na = TRUE)
```

rank	r	value_b	p	value_a	q	s	value_c	r_b	p_f	q_h	s_k
1	a	11	e	NA	g	NA	30	0	0	0	NA
2	a	NA	NA	22	g	j	30	0	NA	0	0
3	a	13	e	23	NA	j	30	0	0	NA	0
4	b	14	f	24	NA	k	30	1	1	NA	1
5	b	NA	NA	25	h	k	30	1	NA	1	1
6	b	16	f	NA	h	NA	30	1	1	1	NA

列名は元々の列に、ダミー化した列を追加する。ダミー化前の列はデフォルトで残るが、remove_selected_columns = TRUE にするとダミー化前の列を削除し、結果としてダミー化しなかった列が前、ダミー化した列が後にくる (それぞれの中の列の順序は元の順序が維持される)。つまり列の順序は保存されない。

```
mat_fastDummies_n_filtered <- fastDummies::dummy_cols(df, remove_first_dummy = FALSE, remove_selecte
```

rank	value_b	value_a	value_c	r_a	r_b	p_e	p_f	q_g	q_h	s_j	s_k
1	11	NA	30	1	0	1	0	1	0	NA	NA
2	NA	22	30	1	0	NA	NA	1	0	1	0
3	13	23	30	1	0	1	0	NA	NA	1	0
4	14	24	30	0	1	0	1	NA	NA	0	1
5	NA	25	30	0	1	NA	NA	0	1	0	1
6	16	NA	30	0	1	0	1	0	1	NA	NA

```
mat_fastDummies_n_1_filtered <- fastDummies::dummy_cols(df, remove_first_dummy = TRUE, remove_select
```

rank	value_b	value_a	value_c	r_b	p_f	q_h	s_k
1	11	NA	30	0	0	0	NA
2	NA	22	30	0	NA	0	0
3	13	23	30	0	0	NA	0
4	14	24	30	1	1	NA	1
5	NA	25	30	1	NA	1	1
6	16	NA	30	1	1	1	NA

tidyr で NA を補間する

Stack Overflowの内容を参考にした。

N カテゴリを N 変数にするダミー化はこうであった。

```
df <- readr::read_csv('with_na.csv', col_types = 'nfnfnffn')
mat <- makedummies::makedummies(df, basal_level = TRUE)
```

rank	r_a	r_b	value_b	p_e	p_f	value_a	q_g	q_h	s_j	s_k	value_c
1	1	0	11	1	0	NA	1	0	NA	NA	30
2	1	0	NA	NA	NA	22	1	0	1	0	30
3	1	0	13	1	0	23	NA	NA	1	0	30
4	0	1	14	0	1	24	NA	NA	0	1	30
5	0	1	NA	NA	NA	25	0	1	0	1	30
6	0	1	16	0	1	NA	0	1	NA	NA	30

tidyr::fill を使って、各列を補間する。purrr::reduce を使うと、それぞれの列について補間できる。

```
mat_interpolated_1 <- purrr::reduce(.x = colnames(mat), .init = mat, .f = function(df, col_name) {
  df %>%
    tidyr::fill(col_name, .direction = 'down') %>%
```

```
    tidyr::fill(col_name, .direction = 'up')
})
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(col_name)` instead of `col_name` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

rank	r_a	r_b	value_b	p_e	p_f	value_a	q_g	q_h	s_j	s_k	value_c
1	1	0	11	1	0	22	1	0	1	0	30
2	1	0	11	1	0	22	1	0	1	0	30
3	1	0	13	1	0	23	1	0	1	0	30
4	0	1	14	0	1	24	1	0	0	1	30
5	0	1	14	0	1	25	0	1	0	1	30
6	0	1	16	0	1	25	0	1	0	1	30

以下のようにするとメッセージが出なくなる。

```
mat_interpolated_2 <- purrr::reduce(.x = colnames(mat), .init = mat, .f = function(df, col_name) {
  df %>%
    dplyr::mutate(.prev_val = col_name, .next_val = col_name) %>%
    tidyr::fill(col_name, .direction = 'down') %>%
    tidyr::fill(col_name, .direction = 'up') %>%
    dplyr::mutate(col_name = ifelse(.prev_val == .next_val, .prev_val, col_name)) %>%
    dplyr::select(-c(.prev_val, .next_val))
})
```

rank	r_a	r_b	value_b	p_e	p_f	value_a	q_g	q_h	s_j	s_k	value_c	col_name
1	1	0	11	1	0	NA	1	0	NA	NA	30	rank
2	1	0	NA	NA	NA	22	1	0	1	0	30	rank
3	1	0	13	1	0	23	NA	NA	1	0	30	rank
4	0	1	14	0	1	24	NA	NA	0	1	30	rank
5	0	1	NA	NA	NA	25	0	1	0	1	30	rank
6	0	1	16	0	1	NA	0	1	NA	NA	30	rank

Python でダミー化して補間する

Python では Pandas を使う。Windows では Anadonda の python.exe を PATH で見つけられるようにすると、Python の実行結果をこの R markdown ファイルに埋め込むことができる。

```
import pandas as pd
print(sys.version)
```

```
## 3.9.1 (tags/v3.9.1:1e5d33e, Dec  7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)]
```

```
print(pd.__version__)
```

```
## 1.2.0
```

Pandas でダミー化する

Pandas で CSV ファイルを読み込む。R の NA は、Python では NaN である。

```
df = pd.read_csv('with_na.csv')
print(df)
```

```
##    rank  r  value_b    p  value_a    q    s  value_c
## 0     1  a     11.0    e      NaN    g  NaN       30
## 1     2  a      NaN  NaN     22.0    g    j       30
## 2     3  a     13.0    e     23.0  NaN    j       30
## 3     4  b     14.0    f     24.0  NaN    k       30
## 4     5  b      NaN  NaN     25.0    h    k       30
## 5     6  b     16.0    f      NaN    h  NaN       30
```

N カテゴリを N 変数およびは N-1 変数にする。Data frame の NA は NA のまま残り、デフォルトでは「NA というカテゴリ」は作らない。ダミー化前の列はなくなる。

列の順序は **保存されない**。ダミー化しなかった列が前、ダミー化した列が後にくる (それぞれの中の列の順序は元の順序が維持される)。fastDummies::dummy_cols(remove_selected_columns = TRUE) と同様の出力になる。

```
df_dummy_n = pd.get_dummies(data=df, drop_first=False)
print(df_dummy_n)
```

```
##    rank  value_b  value_a  value_c  r_a  r_b  p_e  p_f  q_g  q_h  s_j  s_k
## 0     1     11.0      NaN       30    1    0    1    0    1    0    0    0
## 1     2      NaN     22.0       30    1    0    0    0    1    0    1    0
## 2     3     13.0     23.0       30    1    0    1    0    0    0    1    0
## 3     4     14.0     24.0       30    0    1    0    1    0    0    0    1
## 4     5      NaN     25.0       30    0    1    0    0    0    1    0    1
## 5     6     16.0      NaN       30    0    1    0    1    0    1    0    0
```

```
df_dummy_n_1 = pd.get_dummies(data=df, drop_first=True)
print(df_dummy_n_1)
```

```
##    rank  value_b  value_a  value_c  r_b  p_f  q_h  s_k
## 0     1     11.0      NaN       30    0    0    0    0
## 1     2      NaN     22.0       30    0    0    0    0
## 2     3     13.0     23.0       30    0    0    0    0
## 3     4     14.0     24.0       30    1    1    0    1
## 4     5      NaN     25.0       30    1    0    1    1
## 5     6     16.0      NaN       30    1    1    1    0
```

dummy_na = True にすると、NA をいうカテゴリを作る。

```
df_dummy_n_na = pd.get_dummies(data=df, dummy_na=True)
print(df_dummy_n_na)
```

```
##    rank  value_b  value_a  value_c  r_a  r_b  r_nan  p_e  p_f  p_nan  q_g  \
## 0     1     11.0      NaN       30    1    0      0    1    0      0    1
## 1     2      NaN     22.0       30    1    0      0    0    0      1    1
## 2     3     13.0     23.0       30    1    0      0    1    0      0    0
## 3     4     14.0     24.0       30    0    1      0    0    1      0    0
## 4     5      NaN     25.0       30    0    1      0    0    0      1    0
## 5     6     16.0      NaN       30    0    1      0    0    1      0    0
##
##    q_h  q_nan  s_j  s_k  s_nan
## 0    0      0    0    0      1
## 1    0      0    1    0      0
## 2    0      1    1    0      0
## 3    0      1    0    1      0
## 4    1      0    0    1      0
## 5    1      0    0    0      1
```

Pandas で NA を補間する

Pandas interpolate を使うと、NA を前後の行から補間する。

```
print(df)
```

```
##    rank  r  value_b    p  value_a    q    s  value_c
## 0     1  a     11.0    e      NaN    g  NaN       30
## 1     2  a      NaN  NaN     22.0    g    j       30
## 2     3  a     13.0    e     23.0  NaN    j       30
## 3     4  b     14.0    f     24.0  NaN    k       30
## 4     5  b      NaN  NaN     25.0    h    k       30
## 5     6  b     16.0    f      NaN    h  NaN       30
```

```
df_forward_interpolated = df_dummy_n.interpolate(limit_direction='forward')
print(df_forward_interpolated)
```

```
##    rank  value_b  value_a  value_c  r_a  r_b  p_e  p_f  q_g  q_h  s_j  s_k
## 0     1     11.0      NaN       30    1    0    1    0    1    0    0    0
## 1     2     12.0     22.0       30    1    0    0    0    1    0    1    0
## 2     3     13.0     23.0       30    1    0    1    0    0    0    1    0
## 3     4     14.0     24.0       30    0    1    0    1    0    0    0    1
## 4     5     15.0     25.0       30    0    1    0    0    0    1    0    1
## 5     6     16.0     25.0       30    0    1    0    1    0    1    0    0
```

```
df_backward_interpolated = df_dummy_n.interpolate(limit_direction='backward')
print(df_backward_interpolated)
```

```
##    rank  value_b  value_a  value_c  r_a  r_b  p_e  p_f  q_g  q_h  s_j  s_k
## 0     1     11.0     22.0       30    1    0    1    0    1    0    0    0
## 1     2     12.0     22.0       30    1    0    0    0    1    0    1    0
## 2     3     13.0     23.0       30    1    0    1    0    0    0    1    0
## 3     4     14.0     24.0       30    0    1    0    1    0    0    0    1
## 4     5     15.0     25.0       30    0    1    0    0    0    1    0    1
## 5     6     16.0      NaN       30    0    1    0    1    0    1    0    0
```

```
df_both_interpolated = df_dummy_n.interpolate(limit_direction='both')
print(df_both_interpolated)
```

```
##    rank  value_b  value_a  value_c  r_a  r_b  p_e  p_f  q_g  q_h  s_j  s_k
## 0     1     11.0     22.0       30    1    0    1    0    1    0    0    0
## 1     2     12.0     22.0       30    1    0    0    0    1    0    1    0
## 2     3     13.0     23.0       30    1    0    1    0    0    0    1    0
## 3     4     14.0     24.0       30    0    1    0    1    0    0    0    1
## 4     5     15.0     25.0       30    0    1    0    0    0    1    0    1
## 5     6     16.0     25.0       30    0    1    0    1    0    1    0    0
```