Does Predictor Order Affect the Results of Linear Regression: Simulation Study

This R notebook provides the details and replicable code for the simulation study presented in this blog post about the effect on the coefficients when playing around with the input order of your predictors in R's linear regression.

Environment Setup

We are using `renv` to manage the environment.

```
source("renv/activate.R")
library(tidyverse)
library(magrittr)
library(farff)
library(gtools)
library(OpenML)
library(glue)
library(fastDummies)
theme_set(theme_light())
```

Getting Datasets from OpenML

We are using OpenML's dataset repository for this study. The dataset repository is queried with the following parameters.

```
num_obs_min <-
  25
num_obs_max <-
  1000
num_features_min <-
  3
num_features_max <-
  59
num_missing_values <-
  0
num_classes <- 0 # OpenML's way to identify datasets with numeric target variable
```

These choices are obviously quite subjective and could be debated. For example, we can not rule out a potential relationship between the number of predictors and the our issue of interest (fluctuations in coefficients under predictor permutations). However, this study is not designed to be "representative" of the overall population of datasets but is meant to provide us with a number of educational example cases along which we can analyse the issue from a more theoretical perspective.

```
oml_datasets <-
  listOMLDataSets(
    number.of.instances = c(num_obs_min,
                            num_obs_max),
```

```r
    number.of.features = c(num_features_min,
                           num_features_max),
    number.of.missing.values = num_missing_values
  ) %>%
  filter(number.of.classes == num_classes,
         format == "ARFF") # exclude sparse format which seems to cause problems

datasets_raw <-
  lapply(oml_datasets$data.id,
         getOMLDataSet)

names(datasets_raw) <-
  as.character(oml_datasets$data.id)

print(glue("Downloaded {length(datasets_raw)} datasets from OpenML repository."))
```

```
## Downloaded 207 datasets from OpenML repository.
```

Preprocessing

We will apply some minimal pre-processing to the data. Most importantly we dummify the categorical variables. While the `lm` function can handle categorical variables directly, this way will give us more control over the actual predictor ordering.

```r
preprocess_oml_dataset <-
  function(dataset) {
    if (any(!is.na(dataset$desc$ignore.attribute))) {
      dataset$data %<>%
        select(-dataset$desc$ignore.attribute)
    }

    if (any(sapply(dataset$data, class) %in% c("factor", "character"))) {
      dataset$data <-
        dummy_cols(
          dataset$data,
          remove_selected_columns = T,
          remove_first_dummy = T
        )
    }

    return(
      dataset$data %>% nest(data = everything()) %>%
        mutate(
          dataset_id = dataset$desc$id,
          dep_var = dataset$desc$default.target.attribute,
          n_obs = nrow(dataset$data),
          n_cols = ncol(dataset$data)
        )
    )
  }

datasets_processed <-
  lapply(datasets_raw,
         FUN = preprocess_oml_dataset)
```

Create Datasets with Permuted Predictors

For our simulation we will run several models on the same dataset with different input orders for the predictors. Since it will become computationally unfeasible (and also probably unnecessary for our purposes) to run through all permutations we are limiting the number of permutations per dataset to 20.

Actually calculating all permutations and then choosing a subset of 20 from those permutations is also not computationally feasible for some of the number of predictors in the dataset. Therefore we are generating 200 permutations by randomly sampling the column indices and then choosing a subsample of up to 20 from the results.

```
generate_predictor_permutations <-
  function(data,
           dep_var_name,
           perms_to_keep =  20,
           perms_to_generate = 200) {
    dep_var_col <-
      which(names(data) == dep_var_name)

    pred_cols <-
      setdiff(1:ncol(data),
              dep_var_col)
    perms <-
      unique(replicate(
        perms_to_generate,
        sample(pred_cols,
               size = length(pred_cols),
               replace = F),
        simplify = F
      ))
    perms <-
      perms[1:min(length(perms), perms_to_keep)]

    perms <-
      lapply(perms,
             function(x)
               c(dep_var_col, x))

    return(perms)
  }
```

With the number of datasets and permutations we can actually generate all the permuted datasets and keep them in memory (as opposed to permuting the inputs on the fly right before the fitting of the model).

In order to be able to consistently work with `dplyr` pipes we will put the resulting datasets in a nested dataframe instead of keeping the list structure.

```
set.seed(41125)
datasets_simulation <-
  datasets_processed %>%
  bind_rows() %>%
  rowwise() %>%
  mutate(predictor_permutations = list(generate_predictor_permutations(data,
                                                                       dep_var))) %>%
```

```
  unnest(predictor_permutations) %>%
  mutate(simulation_id = row_number()) %>%
  rowwise() %>%
  mutate(data = data %>%
           select(all_of(predictor_permutations)) %>%
           list) %>%
  select(simulation_id,
         dataset_id,
         dep_var,
         data)
print(glue("Generated {nrow(datasets_simulation)} datasets for simulation."))
```

```
## Generated 3764 datasets for simulation.
```

Fitting the Models

With the nested dataframe structure fitting the models can be done easily with a `rowwise` and `mutate` statement.

```
results_lm <-
  datasets_simulation %>%
  rowwise() %>%
  mutate(model_fit = list(lm(
    formula = as.formula(paste0(dep_var, " ~ .")),
    data = data
  ))) %>%
  select(simulation_id,
         dataset_id,
         model_fit) %>%
  ungroup()
```

We extract the coefficients from the models and put them in a for analysis.

```
coefficients_fit <-
  results_lm %>%
  rowwise() %>%
  mutate(coefficients = list(bind_rows(coefficients(model_fit)) %>%
                               gather(key, value))) %>%
  select(-model_fit) %>%
  ungroup() %>%
  unnest(cols = coefficients)
print(glue("Generated {nrow(coefficients_fit)} fits for {nrow(coefficients_fit %>% distinct(dataset_id,k
```

```
## Generated 79470 fits for 4035 different coefficients.
```

```
coefficients_fit %>% head
```

```
## # A tibble: 6 x 4
##   simulation_id dataset_id key             value
##           <int>      <int> <chr>           <dbl>
## 1             1          8 (Intercept) -15.0
## 2             1          8 gammagt       0.0199
## 3             1          8 alkphos       0.00761
## 4             1          8 sgot          0.0496
## 5             1          8 sgpt         -0.0103
## 6             1          8 mcv           0.180
```

Measuring Coefficient Fluctuations

To analyse the fluctuations in the coefficients we will calculate the coefficient of variation (CV) for each of the fit coefficient.

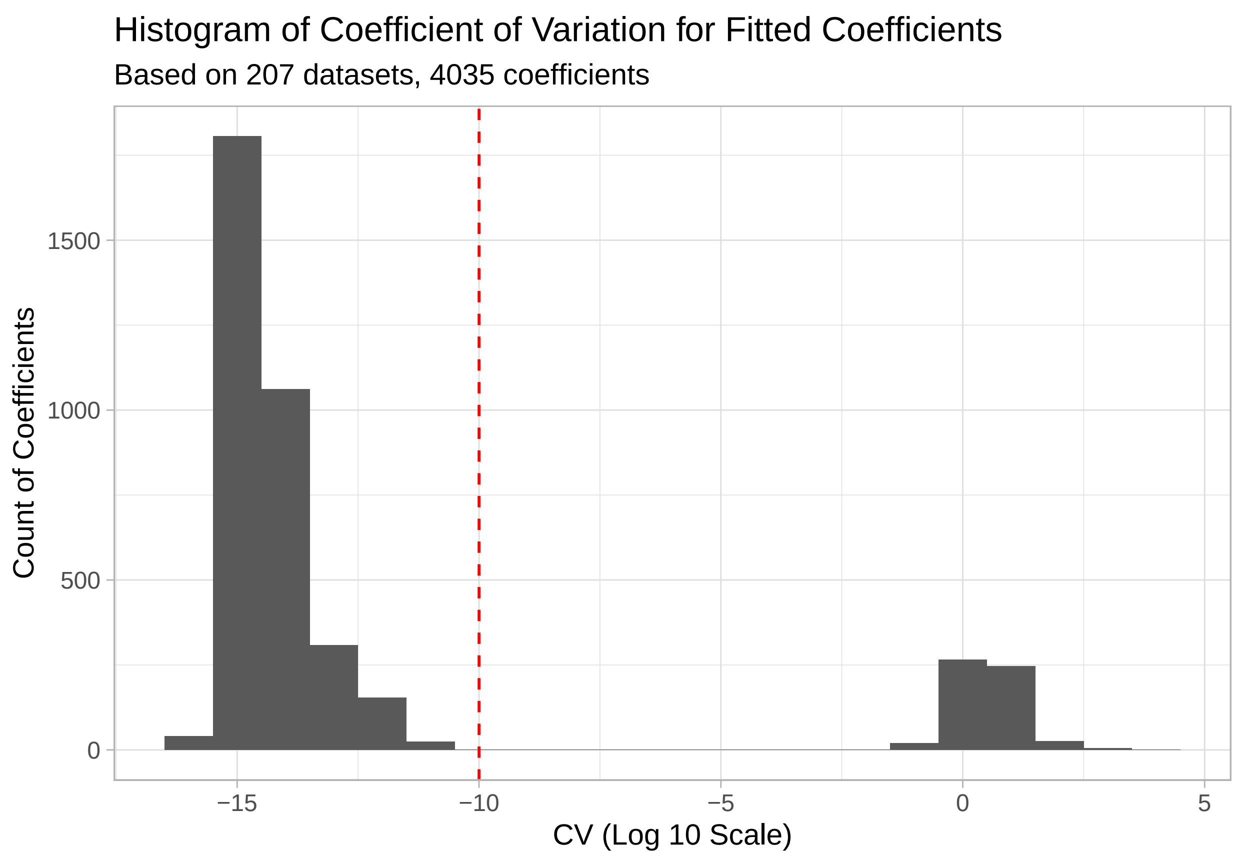
```
coefficients_summary <-
  coefficients_fit %>%
  group_by(dataset_id, key) %>%
  summarise(total_coefficients_fit = n(),
            total_missing = sum(is.na(value)),
            cv = sd(value,na.rm = T)/abs(mean(value,na.rm=T)),
            .groups = "drop")
coefficients_summary
```

```
## # A tibble: 4,035 x 5
##    dataset_id key         total_coefficients_fit total_missing       cv
##  *      <int> <chr>                        <int>         <int>    <dbl>
##  1          8 (Intercept)                     20             0 5.10e-16
##  2          8 alkphos                         20             0 1.08e-15
##  3          8 gammagt                         20             0 1.03e-15
##  4          8 mcv                             20             0 4.88e-16
##  5          8 sgot                            20             0 1.58e-15
##  6          8 sgpt                            20             0 1.72e-15
##  7        190 (Intercept)                      2             0 1.97e-16
##  8        190 GMAT                             2             0 1.14e-16
##  9        190 sex_1                            2             0 0.
## 10        191 (Intercept)                     20             0 2.03e-14
## # ... with 4,025 more rows
```

We define a threshold for the CV of 10^{-10} above which we flag the fluctuations of the coefficient as too high. This threshold is somewhat arbitrary but it should limit the maximum fluctuation of a given coefficient to $10^{-10} * \sqrt{N-1}$ which should be acceptable for most practical applications. N here is the number of fits for the coefficient which in our case ranges between 2 and 20.

```
cv_threshold <-
  1e-10
coefficients_summary %>%
  ggplot(aes(x = log10(cv))) +
  geom_histogram(binwidth = 1) +
  geom_vline(xintercept = log10(cv_threshold),
             color = "red",
             linetype ="dashed") +
  ylab("Count of Coefficients") +
  xlab("CV (Log 10 Scale)") +
  ggtitle("Histogram of Coefficient of Variation for Fitted Coefficients",
          subtitle = paste0("Based on ", length(datasets_processed)," datasets, ",
                            nrow(coefficients_summary), " coefficients"))
```

```
## Warning: Removed 72 rows containing non-finite values (stat_bin).
```



We can see that there is actually quite a substantial number of coefficients that fluctuate more than our allowed threshold. What is more we can see that some of the coefficients could not be fit (are missing):

```
coefficients_summary %>%
  summarise(
    perc_coefficients_missing = sum(total_missing) / sum(total_coefficients_fit),
    high_cv = mean(cv > cv_threshold, na.rm = T)
  )
```

```
## # A tibble: 1 x 2
##   perc_coefficients_missing high_cv
##                       <dbl>   <dbl>
## 1                    0.0390   0.143
```

We will try to find out how the fluctuations come about by looking at the dataset level.

Analysing Problematic Datasets

```
datasets_overview <-
  coefficients_summary %>%
  group_by(dataset_id) %>%
  summarise(
    max_cv = max(cv, na.rm = T),
    high_cv = any(cv > 1e-10, na.rm = T),
    perc_coefficients_missing = sum(total_missing) / sum(total_coefficients_fit),
    .groups = "drop"
  ) %>%
```

```
  left_join(datasets_processed %>%
              bind_rows() %>%
              select(dataset_id, dep_var, n_obs, n_cols),
            by = "dataset_id")

datasets_summary <-
  datasets_overview %>%
  summarise(perc_high_cv = mean(high_cv),
            total_high_cv = sum(high_cv))
datasets_summary
```

```
## # A tibble: 1 x 2
##   perc_high_cv total_high_cv
##          <dbl>         <int>
## 1       0.0773            16
```

On the dataset level we can see that 16 out of 207 (7.7 %) datasets show coefficient fluctuations.

```
datasets_overview %>%
  filter(high_cv)
```

```
## # A tibble: 16 x 7
##    dataset_id  max_cv high_cv perc_coefficients_missi~ dep_var      n_obs n_cols
##         <int>   <dbl> <lgl>                      <dbl> <chr>        <int>  <int>
##  1        195    2.64 TRUE                      0.0476 price          159     21
##  2        421 5321.   TRUE                      0.426  oz54            31     54
##  3        482   11.5  TRUE                      0.0217 events         559     46
##  4        487  410.   TRUE                      0.268  response_1      30     41
##  5        513    3.91 TRUE                      0.0217 events         559     46
##  6        518    1.02 TRUE                      0.222  Violence_ti~    74      9
##  7        521 1284.   TRUE                      0.118  Team_1_wins    120     34
##  8        527   11.3  TRUE                      0      Gore00          67     15
##  9        530    1.17 TRUE                      0.0833 GDP             66     12
## 10        533   35.0  TRUE                      0.0217 events         559     46
## 11        536   24.1  TRUE                      0.0217 events         559     46
## 12        543 2462.   TRUE                      0.111  LSTAT          506    117
## 13        551   30.5  TRUE                      0.188  Accidents      108     16
## 14       1051   70.4  TRUE                      0.238  ACT_EFFORT      60     42
## 15       1076  194.   TRUE                      0.383  act_effort      93    107
## 16       1091   49.8  TRUE                      0      NOx             59     16
```

Underdetermined Problems

The first thing we notice is that three of the datasets have less observations than predictors in the model, i.e. the problem is underdetermined. This is also reflected in the missing coefficients (column `perc_coefficients_missing`):

```
datasets_overview %>%
  filter(high_cv, n_obs < n_cols)
```

```
## # A tibble: 3 x 7
##   dataset_id max_cv high_cv perc_coefficients_missing dep_var    n_obs n_cols
##        <int>  <dbl> <lgl>                       <dbl> <chr>      <int>  <int>
## 1        421  5321. TRUE                        0.426 oz54          31     54
## 2        487   410. TRUE                        0.268 response_1    30     41
```

```
## 3       1076   194. TRUE                         0.383 act_effort    93    107
```

Note that while `n_cols` counts all the columns (target variable and predictors) in the dataset it still happens to be equal to the number of coefficients we want to estimate since we are fitting a model with an intercept.

For a more detailed analysis let's add the rank of the QR decomposition for each model fit.

```
results_lm$rank_qr <-
  lapply(results_lm$model_fit,
         function(x) {
           x$qr$rank
         }) %>% unlist

# check if rank of QR decomp is invariant under column permutations so we can
# safely append to dataset overview
if (nrow(results_lm %>% distinct(dataset_id, rank_qr)) !=
    nrow(results_lm %>% distinct(dataset_id))) {
  stop("QR decomposition rank not invariant under predictor permutations.")
}

ranks_qr <-
  results_lm %>%
  distinct(dataset_id,
           rank_qr)

datasets_overview %<>%
  left_join(ranks_qr,
            by = "dataset_id") %>%
  mutate(singular_qr = rank_qr < n_cols)
datasets_overview %>%
  filter(high_cv)
```

```
## # A tibble: 16 x 9
##    dataset_id max_cv high_cv perc_coefficien~ dep_var n_obs n_cols rank_qr
##         <int>  <dbl> <lgl>              <dbl> <chr>   <int>  <int>   <int>
##  1        195 2.64e0 TRUE              0.0476 price     159     21      20
##  2        421 5.32e3 TRUE              0.426  oz54       31     54      31
##  3        482 1.15e1 TRUE              0.0217 events    559     46      45
##  4        487 4.10e2 TRUE              0.268  respon~    30     41      30
##  5        513 3.91e0 TRUE              0.0217 events    559     46      45
##  6        518 1.02e0 TRUE              0.222  Violen~    74      9       7
##  7        521 1.28e3 TRUE              0.118  Team_1~   120     34      30
##  8        527 1.13e1 TRUE              0      Gore00     67     15      15
##  9        530 1.17e0 TRUE              0.0833 GDP        66     12      11
## 10        533 3.50e1 TRUE              0.0217 events    559     46      45
## 11        536 2.41e1 TRUE              0.0217 events    559     46      45
## 12        543 2.46e3 TRUE              0.111  LSTAT     506    117     104
## 13        551 3.05e1 TRUE              0.188  Accide~   108     16      13
## 14       1051 7.04e1 TRUE              0.238  ACT_EF~    60     42      32
## 15       1076 1.94e2 TRUE              0.383  act_ef~    93    107      66
## 16       1091 4.98e1 TRUE              0      NOx        59     16      16
## # ... with 1 more variable: singular_qr <lgl>
```

We can see that with the exception of two datasets all the datasets with high coefficient fluctuations are showing a singular QR decomposition.

Multicollinearity

We now focus on the problematic datasets which are not underdetermined and will investigate what is causing the singular QR decomposition:

```
datasets_overview %>%
  filter(high_cv, singular_qr, n_obs > n_cols)
```

```
## # A tibble: 11 x 9
##    dataset_id max_cv high_cv perc_coefficien~ dep_var n_obs n_cols rank_qr
##         <int>  <dbl> <lgl>              <dbl> <chr>   <int>  <int>   <int>
##  1        195 2.64e0 TRUE              0.0476 price     159     21      20
##  2        482 1.15e1 TRUE              0.0217 events    559     46      45
##  3        513 3.91e0 TRUE              0.0217 events    559     46      45
##  4        518 1.02e0 TRUE              0.222  Violen~    74      9       7
##  5        521 1.28e3 TRUE              0.118  Team_1~   120     34      30
##  6        530 1.17e0 TRUE              0.0833 GDP        66     12      11
##  7        533 3.50e1 TRUE              0.0217 events    559     46      45
##  8        536 2.41e1 TRUE              0.0217 events    559     46      45
##  9        543 2.46e3 TRUE              0.111  LSTAT     506    117     104
## 10        551 3.05e1 TRUE              0.188  Accide~   108     16      13
## 11       1051 7.04e1 TRUE              0.238  ACT_EF~    60     42      32
## # ... with 1 more variable: singular_qr <lgl>
```

As we will show below each of the datasets in the list above contains multicollinearity in the predictors.

Dataset 195

There is collinearity in the dummy columns belonging to the `symboling` variable. If you were to simply apply a one hot encoding without any other precautions you will get collinearity in the resulting dummy columns. In our dummification code we therefore remove the set the flag `remove_first_dummy` of the `dummy_cols` function to `TRUE` which will remove the dummy column corresponding to the first level in the data. However, this approach fails here since the `symboling` variable is supposed to have 7 levels (`-3`, `-2`, `-1`, `0`, `1`, `2`, `3`) but only the last six levels are ever encountered in the data. Therefore the approach to remove collinearity from the dummy columns fails in this case and we get collinearity between the dummy columns.

```
current_ds_id <-
  "195"
current_ds <- datasets_raw[[current_ds_id]]$data
current_ds_processed <- datasets_processed[[current_ds_id]]$data[[1]]
all((1-(current_ds_processed$`symboling_-2` + current_ds_processed$`symboling_-1`  + current_ds_process
```

```
## [1] TRUE
```

Datasets 482, 513, 533, 536

Those datasets seem to be variants of the same dataset which all have the same collinearity pattern. The `conc` variable has a unique value within each level of `group`. Therefore after dummification `conc` can be represented as the weighted sum of the `group` columns where the weights are the values of `conc` within that group. See code below.

```
current_ds_id <-
  "482"
current_ds <- datasets_raw[[current_ds_id]]$data
current_ds_processed <-
  datasets_processed[[current_ds_id]]$data[[1]]
weights <-
```

```
  current_ds %>% distinct(group, conc) %>% pull(conc)
weights <-
  weights[2:length(weights)]

all(
  apply(
    current_ds_processed %>% select(matches("^group")) * matrix(
      rep(weights, nrow(current_ds_processed)),
      nrow = nrow(current_ds_processed),
      byrow = T
    ),
    FUN = sum,
    MARGIN = 1
  ) == current_ds_processed$conc
)
```

```
## [1] TRUE
```

Dataset 518

The variable `Injuries` is the sum of `Good.neutral_injuries` and `Bad_injuries` (same applies for `Fatalities`).

```
current_ds_id <-
  "518"
current_ds <- datasets_raw[[current_ds_id]]$data
current_ds_processed <-
  datasets_processed[[current_ds_id]]$data
all(all((current_ds$Good.neutral_injuries + current_ds$Bad_injuries) == current_ds$Injuries))
```

```
## [1] TRUE
```

Dataset 521

The predictors `Seed_team_1`, `Seed_team_2` and the `X*`s are collinear:

```
current_ds_id <-
  "521"
current_ds <- datasets_raw[[current_ds_id]]$data %>%
  mutate_all(.funs = list( ~ as.numeric(as.character(.))))
current_ds %<>%
  mutate(
    Seed_team_2_comb = (
      Seed_team_1 - X2 - 2 * X3 - 3 * X4 - 4 * X5 - 5 * X6 - 6 * X7 -
        7 * X8 - 8 * X9 - 9 * X10 - 10 * X11 - 11 *
        X12 - 12 * X13 -
        13 * X14 - 14 * X15 - 15 * X16
    )
  )
all(current_ds$Seed_team_2 == current_ds$Seed_team_2_comb)
```

```
## [1] TRUE
```

Dataset 530

Variable `Total2000` is the sum of `Gold2000`, `Silver2000` and `Bronze2000`:

```
current_ds_id <- "530"
current_ds <- datasets_raw[[current_ds_id]]$data
all((current_ds$Gold2000 + current_ds$Silver2000 + current_ds$Bronze2000) == (current_ds$Total2000))
```

```
## [1] TRUE
```

Dataset 543

The values in `TOWN_ID` and `TOWN` have a 1-to-1 relationship. Therefore after the dummification `TOWN_ID` can be represented as the weighted sum over all `TOWN` dummy columns where the weights for each `TOWN` column is the value of the corresponding `TOWN_ID`.

```
current_ds_id <- "543"
current_ds <- datasets_raw[[current_ds_id]]$data
current_ds %>% distinct(TOWN, TOWN_ID)
```

```
##                          TOWN TOWN_ID
## 0                      Nahant       0
## 1                  Swampscott       1
## 3                  Marblehead       2
## 6                       Salem       3
## 13                       Lynn       4
## 35                     Sargus       5
## 39                  Lynnfield       6
## 41                    Peabody       7
## 50                    Danvers       8
## 54                  Middleton       9
## 55                  Topsfield      10
## 56                   Hamilton      11
## 57                     Wenham      12
## 58                    Beverly      13
## 64                 Manchester      14
## 65              North_Reading      15
## 67                 Wilmington      16
## 70                 Burlington      17
## 74                     Woburn      18
## 80                    Reading      19
## 84                  Wakefield      20
## 88                    Melrose      21
## 92                   Stoneham      22
## 95                 Winchester      23
## 100                   Medford      24
## 111                    Malden      25
## 120                   Everett      26
## 127                Somerville      27
## 142                 Cambridge      28
## 172                 Arlington      29
## 179                   Belmont      30
## 187                 Lexington      31
## 193                   Bedford      32
## 195                   Lincoln      33
## 196                   Concord      34
## 199                   Sudbury      35
## 201                   Wayland      36
## 203                    Weston      37
```

```
## 205                    Waltham     38
## 216                  Watertown     39
## 220                     Newton     40
## 238                     Natick     41
## 244                 Framingham     42
## 254                    Ashland     43
## 256                   Sherborn     44
## 257                  Brookline     45
## 269                     Dedham     46
## 274                    Needham     47
## 279                  Wellesley     48
## 283                      Dover     49
## 284                   Medfield     50
## 285                     Millis     51
## 286                    Norfolk     52
## 287                    Walpole     53
## 290                   Westwood     54
## 293                    Norwood     55
## 298                     Sharon     56
## 301                     Canton     57
## 304                     Milton     58
## 308                     Quincy     59
## 320                  Braintree     60
## 328                   Randolph     61
## 331                   Holbrook     62
## 333                   Weymouth     63
## 341                   Cohasset     64
## 342                       Hull     65
## 343                    Hingham     66
## 345                   Rockland     67
## 347                    Hanover     68
## 348                    Norwell     69
## 349                   Scituate     70
## 351                 Marshfield     71
## 353                    Duxbury     72
## 354                   Pembroke     73
## 356 Boston_Allston-Brighton     74
## 364            Boston_Back_Bay     75
## 370         Boston_Beacon_Hill     76
## 373          Boston_North_End     77
## 375         Boston_Charlestown     78
## 381         Boston_East_Boston     79
## 393        Boston_South_Boston     80
## 406            Boston_Downtown     81
## 414             Boston_Roxbury     82
## 433          Boston_Savin_Hill     83
## 456          Boston_Dorchester     84
## 467            Boston_Mattapan     85
## 473        Boston_Forest_Hills     86
## 480        Boston_West_Roxbury     87
## 484          Boston_Hyde_Park     88
## 488                    Chelsea     89
## 493                     Revere     90
## 501                   Winthrop     91
```

Dataset 551

The dataset contains factor variabls for `Season` and `Month`. Therefore after dummification the Season variables can be rewritten as a sum of a subset of the month variables (and vice versa).

```
current_ds_id <- "551"
current_ds_processed <- datasets_processed[[current_ds_id]]$data[[1]]
all(current_ds_processed$Season_Summer == (current_ds_processed$Month_June + current_ds_processed$Month_
```

```
## [1] TRUE
```

Dataset 1051

In this dataset it's somewhat tricky to spot the collinearity pattern as it can not be derived from the meaning of the variables. It seems to have only occured randomly due to the high number of categorical variables relative to the amount of observations.

```
current_ds_id <- "1051"
current_ds <- datasets_raw[[current_ds_id]]$data
current_ds_processed <- datasets_processed[[current_ds_id]]$data[[1]]
all(current_ds_processed$MODP_Very_High == (current_ds_processed$STOR_Very_High - current_ds_processed$M
```

```
## [1] TRUE
```

Remaining Datasets

For the remaining datasets with coefficient fluctuations we were able to spot collinearity between the *target variable* and some of the predictors. This is obviously not a reasonable setting to run regression in but it shows nicely how `lm` handles this case.

Since $X^t X$ is not rank deficient the QR decomposition algorithm does not hit its stopping criterion and therefore does not return `NA`s. Instead it returns values close to zeros (on the order of 10^{-16}) for some of the coefficients in each fit.

This probably makes sense as multiple regression can be interpreted as fitting a sequence of univariate regressions in which each member of the sequence fits against the residual of its predecessor (see Elements of Statistical Learning Section 3.2.3).

That means after all the predictors which are involved in the collinearity with the target variable have been fit the residual will become zero (or very close to zero) and all remaining coefficients will be set so accordingly.

Dataset 527

The `Total00` is the sum of a subset of the predictors and the target variable.

```
current_ds_id <- "527"
current_ds <- datasets_raw[[current_ds_id]]$data
current_ds_processed <-
  datasets_processed[[current_ds_id]]$data[[1]]
all((
  datasets_processed[[current_ds_id]]$data[[1]]$Total00 - apply(
    datasets_processed[[current_ds_id]]$data[[1]] %>% select(Bush00:Phillips00),
    MARGIN = 1,
    sum
  )
) == datasets_processed[[current_ds_id]]$data[[1]]$Gore00)
```

```
## [1] TRUE
```

Dataset 1091

The target variable `NOx` is included under a different name (`NOxPot`) as a predictor.

```
current_ds_id <- "1091"
current_ds <- datasets_raw[[current_ds_id]]$data
current_ds_processed <-
  datasets_processed[[current_ds_id]]$data[[1]]
all(current_ds_processed$NOx == current_ds_processed$NOxPot)
```

```
## [1] TRUE
```

Pivoting in Fortran

As one final thing we check whether the pivoting that Fortran applies internally to improve the numerical stability of the QR decomposition may have affected some of our results. Luckily they did not:

```
results_lm$pivoting_applied <-
  lapply(results_lm$model_fit, function(x) {
    !all(x$qr$pivot == 1:length(x$qr$pivot))
  }) %>% unlist


datasets_overview %<>%
  left_join(results_lm %>%
              group_by(dataset_id) %>%
              summarise(pivoting_applied = any(pivoting_applied)),
            by = "dataset_id")

datasets_overview %>% filter(pivoting_applied)
```

```
## # A tibble: 18 x 10
##    dataset_id   max_cv high_cv perc_coefficien~ dep_var n_obs n_cols rank_qr
##         <int>    <dbl> <lgl>              <dbl> <chr>   <int>  <int>   <int>
##  1        195 2.64e+ 0 TRUE              0.0476 price     159     21      20
##  2        199 6.72e-15 FALSE             0.143  class     125      7       6
##  3        217 2.00e-14 FALSE             0.0357 activi~    74     28      27
##  4        421 5.32e+ 3 TRUE              0.426  oz54       31     54      31
##  5        482 1.15e+ 1 TRUE              0.0217 events    559     46      45
##  6        494 0.       FALSE             0.333  Aberra~   649      3       2
##  7        513 3.91e+ 0 TRUE              0.0217 events    559     46      45
##  8        518 1.02e+ 0 TRUE              0.222  Violen~    74      9       7
##  9        521 1.28e+ 3 TRUE              0.118  Team_1~   120     34      30
## 10        530 1.17e+ 0 TRUE              0.0833 GDP        66     12      11
## 11        536 2.41e+ 1 TRUE              0.0217 events    559     46      45
## 12        543 2.46e+ 3 TRUE              0.111  LSTAT     506    117     104
## 13        546 1.94e-13 FALSE             0.0769 Score     576     26      24
## 14        551 3.05e+ 1 TRUE              0.188  Accide~   108     16      13
## 15        703 1.17e-11 FALSE             0.104  col_6     526    337     302
## 16       1051 7.04e+ 1 TRUE              0.238  ACT_EF~    60     42      32
## 17       1076 1.94e+ 2 TRUE              0.383  act_ef~    93    107      66
## 18       1245 8.26e-15 FALSE             0.115  OS_yea~   442     26      23
## # ... with 2 more variables: singular_qr <lgl>, pivoting_applied <lgl>
```