Métodos de Reamostragem:

k-Fold-CV, Monte Carlo, LOO-CV e Bootstrap no Tidymodels

Heitor Gabriel S. Monteiro

15/11/2021

Contents

1 Prelúdio

Nosso objetivo é exercitar os algoritmos de reamostragem, que servem para validação dos modelos, visto que a amostragem usada como treino e teste podem influenciar as estimativas dos parâmetros. PAra fins de replicação, fixarei a aleatoriedade em *123*.

```
setwd('/home/heitor/Área de Trabalho/R Projects/Análise Macro/Labs/Lab 13')

library(tidyverse)
library(tidymodels)
library(tensorflow)
library(plotly)
library(GGally) # para os gráficos de correlação
library(ISLR)   # para a base de dados usada

set.seed(123)
```

Os dados são até bem organizados, as únicas alterações que fiz foi na variável-fator de `origin`, do modelo de carros:

```
dds <- as.data.frame(Auto) %>%
    mutate(made_in =
            case_when(Auto$origin==1 ~ 'america',
                      Auto$origin==2 ~ 'europe',
                      Auto$origin==3 ~ 'japan')) %>%
    mutate(made_in = as.factor(made_in)) %>%
    select(-origin) %>%
    as_tibble()

dds %>% str()
```

```
## tibble [392 x 9] (S3: tbl_df/tbl/data.frame)
##  $ mpg         : num [1:392] 18 15 18 16 17 15 14 14 14 15 ...
##  $ cylinders   : num [1:392] 8 8 8 8 8 8 8 8 8 8 ...
##  $ displacement: num [1:392] 307 350 318 304 302 429 454 440 455 390 ...
##  $ horsepower  : num [1:392] 130 165 150 150 140 198 220 215 225 190 ...
##  $ weight      : num [1:392] 3504 3693 3436 3433 3449 ...
##  $ acceleration: num [1:392] 12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year        : num [1:392] 70 70 70 70 70 70 70 70 70 70 ...
##  $ name        : Factor w/ 304 levels "amc ambassador brougham",..: 49 36 231 14 161
##  $ made_in     : Factor w/ 3 levels "america","europe",..: 1 1 1 1 1 1 1 1 1 1 ...
```

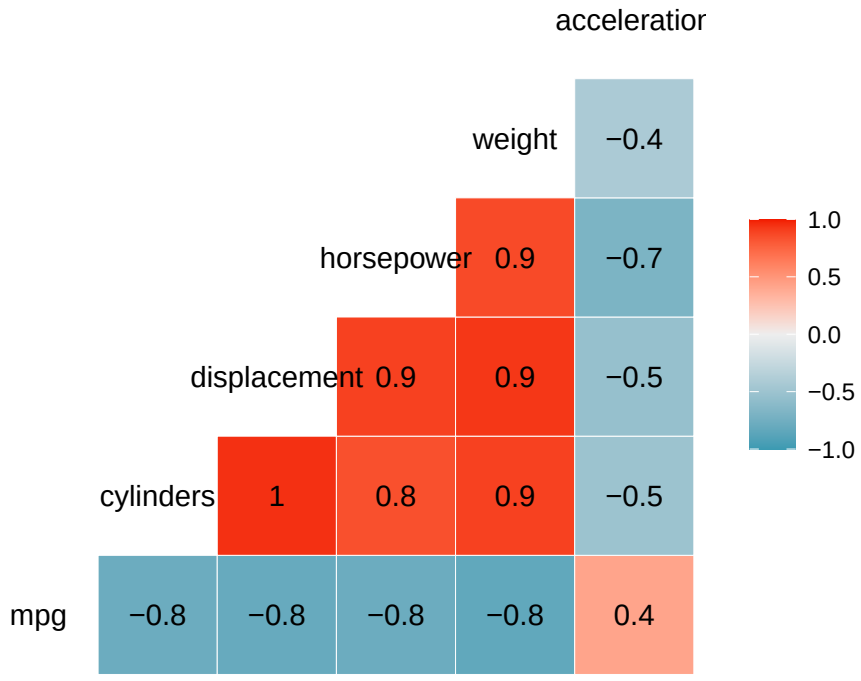
2 Estatísticas Descritivas

A seguir, algumas estatísticas descritivas sobre os dados usados.

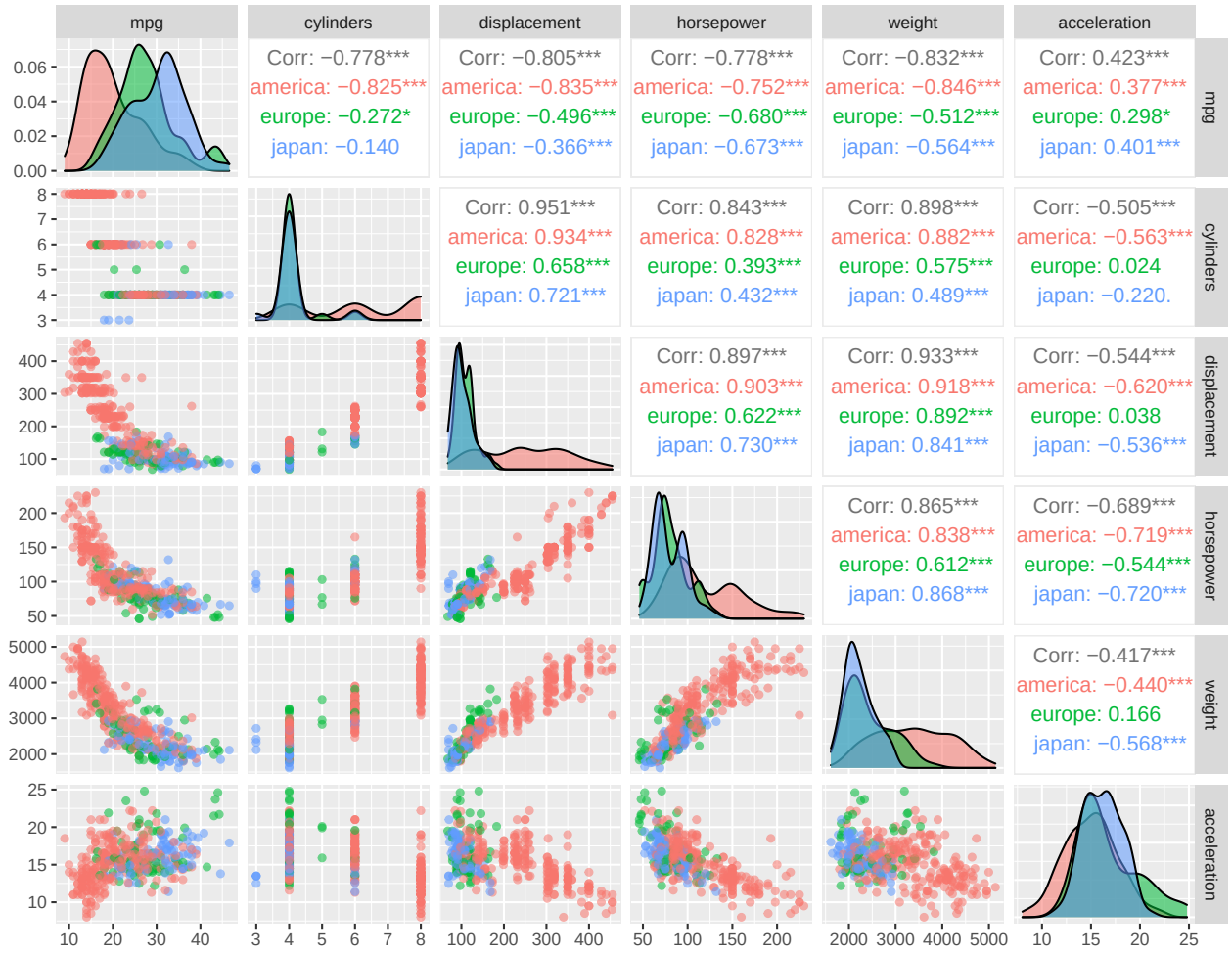
```
dds %>% summary()
```

```
##       mpg          cylinders      displacement     horsepower        weight
##  Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0   Min.   :1613
##  1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0   1st Qu.:2225
##  Median :22.75   Median :4.000   Median :151.0   Median : 93.5   Median :2804
##  Mean   :23.45   Mean   :5.472   Mean   :194.4   Mean   :104.5   Mean   :2978
##  3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0   3rd Qu.:3615
##  Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0   Max.   :5140
##
##   acceleration        year                       name        made_in
##  Min.   : 8.00   Min.   :70.00   amc matador       :  5   america:245
##  1st Qu.:13.78   1st Qu.:73.00   ford pinto        :  5   europe : 68
##  Median :15.50   Median :76.00   toyota corolla    :  5   japan  : 79
##  Mean   :15.54   Mean   :75.98   amc gremlin       :  4
##  3rd Qu.:17.02   3rd Qu.:79.00   amc hornet        :  4
##  Max.   :24.80   Max.   :82.00   chevrolet chevette:  4
##                                  (Other)           :365
```

```
ggcorr(dds %>%
        select(!c('year', 'name', 'made_in')),
       label = T)
```

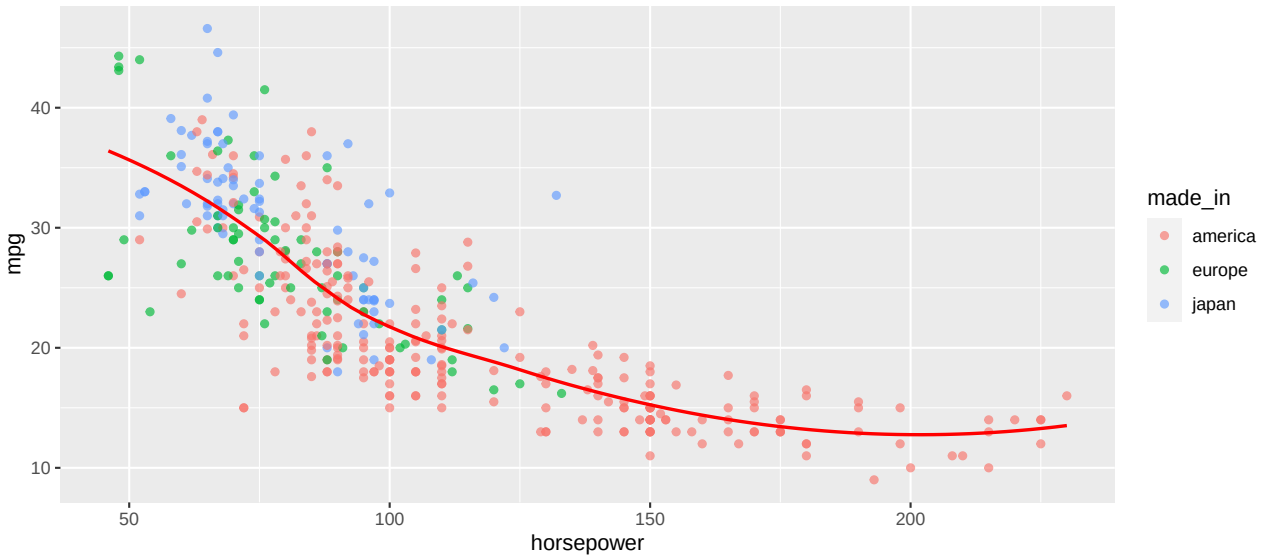


```
ggpairs(dds %>%
            select(!c('year', 'name')),
    columns = 1:6,
    ggplot2::aes(colour=made_in, alpha=.3)) #%>% ggplotly()
```



Como exercício, usaremos `horsepower` para explicar `mpg`:

```
ggplot(
    dds, aes(x=horsepower, y=mpg))+
    geom_point(aes(color=made_in), alpha=.66)+
    geom_smooth(method = 'auto',
                colour='red',
                size = .8,
                se=FALSE) #%>% ggplotly()
```

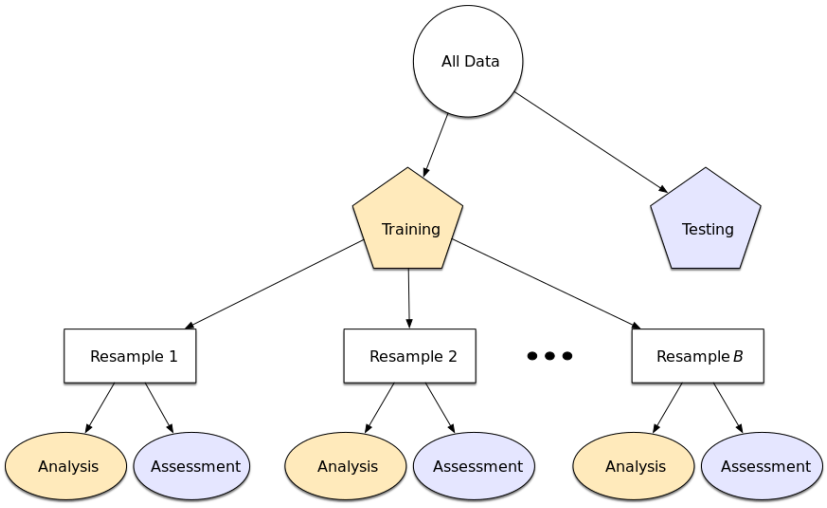


3 Divisão: Treino & Teste

A primeira divisão é feita abaixo.

```
slice_1 <-
    initial_split(dds %>%
                      select(!c('name', 'year',
                                'made_in')))
train_dds   <- training(slice_1)
test_dds    <- testing(slice_1)
```

Digo primeira pois o próprio conjunto de treino é dividido em treino e teste novamente, chamados de *Conjunto de Análise* (para treinar o modelo) e *Conjunto de Avaliação* (para fazer comparar as estimações) para não confundir com a primeira divisão, de acordo com a figura:



Autors: Max Kuhn and Kjell Johnson

A reamostragem é feita apenas na amostra de treino. Se fazemos `k` reamostragens, então 1) `k` grupos amostrais são coletados do conjunto treino, formando `k`'s conjuntos de análise; 2) `k` modelos são formados e aplicados nos conjuntos de avaliação; 3) `k` estatísticas de desempenho são criadas. A estatística final de avaliação do modelo é a média das `k` informações de desempenho.

4 Modelo

Façamos agora o modelo, que funciona como um esqueleto onde colocaremos as variáveis predita e preditora. Observe que deixamos o `penalty` em aberto, usando um `tune()` no lugar. Esse parâmetro será ajustado conforme o menor erro quadrático médio (definiremos mais adiante), o processo de reamostragem nos dirá qual é a melhor regularização a ser usada. No caso do pacote *keras*, `penalty` é a penalização de um *ridge regression*, portanto, um parâmetro de regularização de Tikhonov.

```
glm_algorit <- linear_reg( penalty = tune()) %>%
                           # mixture = 1) %>%
    set_mode('regression') %>%
    set_engine("keras")

glm_algorit %>% translate()
```

```
## Linear Regression Model Specification (regression)
##
## Main Arguments:
##   penalty = tune()
##
## Computational engine: keras
```

```
##
## Model fit template:
## parsnip::keras_mlp(x = missing_arg(), y = missing_arg(), penalty = tune(),
##     hidden_units = 1, act = "linear")
```

5 Fórmula

Agora, dizemos quais e como as variáveis `x` e `y` alimentarão esse modelo. No caso, o grau do polinômio está em aberto e será *tunado* mais a diante. Por esse motivo, não finalizamos a cadeia da "receita" com `%>% prep()` nem veremos como ficam os termos usados aplicados aos dados com o `bake()`, como de costume.

```
recipe_glm <- recipe(mpg ~ horsepower,
                     data=train_dds) %>%
    step_poly( horsepower,
               role = "predictor",
               trained = FALSE,
               objects = NULL,
               degree = tune())
```

6 Workflow

Definiremos um `workflow()`, alimentando o modelo com as variáveis:

```
glm_wrkflw <- workflow() %>%
    add_model(glm_algorit) %>%
    add_recipe(recipe_glm)
```

7 Validações

Definiremos agora três processos de reamostragens. Um ótimo material para aprendê-las, além dos listados nas referências, é o Kuhn & Johnson (2019).

7.1 Reamostragem Cruzada k-Fold

```
vc_kfold <- vfold_cv(train_dds,
                     v=10,
                     repeats = 2)
```

7.2 Reamostragem Bootstrap

```
v_boot <- bootstraps(train_dds,
                     times = 5)
```

7.3 Reamostragem Monte-Carlo

```
vc_mc <- mc_cv(train_dds,
               prop = 4/5,
               times=3)
```

8 Ajustes e Treinamentos

Definiremos quais intervalos os dois parâmetros ajustáveis serão testados com `update()` e, para cada conjunto de parâmetros, uma quantidade de três valores com `grid_regular()`.

```
grid_stand <-
    glm_wrkflw %>%
    parameters() %>%
    update(
        penalty = penalty(range = c(-1, 2),
                trans = scales::pseudo_log_trans()),
        degree = degree(range = c(1, 3))
    ) %>%
    grid_regular(levels = 3)
```

Agora a parte que exigirá mais esforço computacional: aplicar os modelos do workflow criado em cada método de reamostragem. Definimos duas mátricas de performace: `rmse`` emae. A principal diferença entre eles é que o erro quadrático médio` (rmse') põe mais peso nos outliers, é mais sensível a tais observações justamente por elevar a diferença ao quadrado.

```
glm_train_kfold <- glm_wrkflw %>%
    tune_grid(resamples = vc_kfold,
              grid      = grid_stand,
              control   = control_grid(save_pred = T),
              metrics   = metric_set(rmse, mae))
```

```
glm_train_boot <- glm_wrkflw %>%
    tune_grid(resamples = v_boot,
              grid      = grid_stand,
              control   = control_grid(save_pred = T),
              metrics   = metric_set(rmse, mae))

glm_train_mc <- glm_wrkflw %>%
    tune_grid(resamples = vc_mc,
              grid      = grid_stand,
              control   = control_grid(save_pred = T),
              metrics   = metric_set(rmse, mae))
```

9 Seleção do Melhor Modelo

Vejamos os objetos criados para os três métodos de reamostragem. Vemos que não é um dataset convencional, cada item desse objeto é um conjunto de outras informações que podem ser expandidas com `tidy()`, `augment()` e `unnest()`, caso queira.

```
glm_train_kfold
```

```
## # Tuning results
## # 10-fold cross-validation repeated 2 times
## # A tibble: 20 x 6
##    splits           id      id2    .metrics          .notes      .predictions
##    <list>           <chr>   <chr>  <list>            <list>      <list>
##  1 <split [264/30]> Repeat1 Fold01 <tibble [12 x 6]> <tibble [3~ <tibble [180 x~
##  2 <split [264/30]> Repeat1 Fold02 <tibble [12 x 6]> <tibble [3~ <tibble [180 x~
##  3 <split [264/30]> Repeat1 Fold03 <tibble [12 x 6]> <tibble [3~ <tibble [180 x~
##  4 <split [264/30]> Repeat1 Fold04 <tibble [12 x 6]> <tibble [3~ <tibble [180 x~
##  5 <split [265/29]> Repeat1 Fold05 <tibble [12 x 6]> <tibble [3~ <tibble [174 x~
##  6 <split [265/29]> Repeat1 Fold06 <tibble [12 x 6]> <tibble [3~ <tibble [174 x~
##  7 <split [265/29]> Repeat1 Fold07 <tibble [12 x 6]> <tibble [3~ <tibble [174 x~
##  8 <split [265/29]> Repeat1 Fold08 <tibble [12 x 6]> <tibble [3~ <tibble [174 x~
##  9 <split [265/29]> Repeat1 Fold09 <tibble [12 x 6]> <tibble [3~ <tibble [174 x~
## 10 <split [265/29]> Repeat1 Fold10 <tibble [12 x 6]> <tibble [3~ <tibble [174 x~
## 11 <split [264/30]> Repeat2 Fold01 <tibble [12 x 6]> <tibble [3~ <tibble [180 x~
## 12 <split [264/30]> Repeat2 Fold02 <tibble [12 x 6]> <tibble [3~ <tibble [180 x~
## 13 <split [264/30]> Repeat2 Fold03 <tibble [12 x 6]> <tibble [3~ <tibble [180 x~
## 14 <split [264/30]> Repeat2 Fold04 <tibble [12 x 6]> <tibble [3~ <tibble [180 x~
## 15 <split [265/29]> Repeat2 Fold05 <tibble [12 x 6]> <tibble [3~ <tibble [174 x~
## 16 <split [265/29]> Repeat2 Fold06 <tibble [12 x 6]> <tibble [3~ <tibble [174 x~
## 17 <split [265/29]> Repeat2 Fold07 <tibble [12 x 6]> <tibble [3~ <tibble [174 x~
```

```
## 18 <split [265/29]> Repeat2 Fold08 <tibble [12 x 6]> <tibble [3~ <tibble [174 x~
## 19 <split [265/29]> Repeat2 Fold09 <tibble [12 x 6]> <tibble [3~ <tibble [174 x~
## 20 <split [265/29]> Repeat2 Fold10 <tibble [12 x 6]> <tibble [3~ <tibble [174 x~
```

```
glm_train_boot
```

```
## # Tuning results
## # Bootstrap sampling
## # A tibble: 5 x 5
##   splits            id         .metrics          .notes           .predictions
##   <list>            <chr>      <list>            <list>           <list>
## 1 <split [294/110]> Bootstrap1 <tibble [12 x 6]> <tibble [3 x 1]> <tibble [660 ~
## 2 <split [294/108]> Bootstrap2 <tibble [12 x 6]> <tibble [3 x 1]> <tibble [648 ~
## 3 <split [294/110]> Bootstrap3 <tibble [12 x 6]> <tibble [3 x 1]> <tibble [660 ~
## 4 <split [294/107]> Bootstrap4 <tibble [12 x 6]> <tibble [3 x 1]> <tibble [642 ~
## 5 <split [294/104]> Bootstrap5 <tibble [12 x 6]> <tibble [3 x 1]> <tibble [624 ~
```

```
glm_train_mc
```

```
## # Tuning results
## # Monte Carlo cross-validation (0.8/0.2) with 3 resamples
## # A tibble: 3 x 5
##   splits           id        .metrics          .notes           .predictions
##   <list>           <chr>     <list>            <list>           <list>
## 1 <split [235/59]> Resample1 <tibble [12 x 6]> <tibble [3 x 1]> <tibble [354 x ~
## 2 <split [235/59]> Resample2 <tibble [12 x 6]> <tibble [3 x 1]> <tibble [354 x ~
## 3 <split [235/59]> Resample3 <tibble [12 x 6]> <tibble [3 x 1]> <tibble [354 x ~
```

```
collect_metrics(glm_train_kfold)
```

```
## # A tibble: 12 x 8
##    penalty degree .metric .estimator  mean     n std_err .config
##      <dbl>  <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1    1.04      1 mae     standard    23.3    20   0.425 Preprocessor1_Model2
##  2    1.04      1 rmse    standard    24.5    20   0.450 Preprocessor1_Model2
##  3    7.25      1 mae     standard    23.3    20   0.434 Preprocessor1_Model3
##  4    7.25      1 rmse    standard    24.6    20   0.458 Preprocessor1_Model3
##  5    1.04      2 mae     standard    23.3    20   0.431 Preprocessor2_Model2
##  6    1.04      2 rmse    standard    24.5    20   0.458 Preprocessor2_Model2
##  7    7.25      2 mae     standard    23.3    20   0.426 Preprocessor2_Model3
##  8    7.25      2 rmse    standard    24.6    20   0.453 Preprocessor2_Model3
##  9    1.04      3 mae     standard    23.3    20   0.429 Preprocessor3_Model2
## 10    1.04      3 rmse    standard    24.6    20   0.456 Preprocessor3_Model2
## 11    7.25      3 mae     standard    23.3    20   0.438 Preprocessor3_Model3
## 12    7.25      3 rmse    standard    24.6    20   0.463 Preprocessor3_Model3
```

```
collect_metrics(glm_train_boot)
```

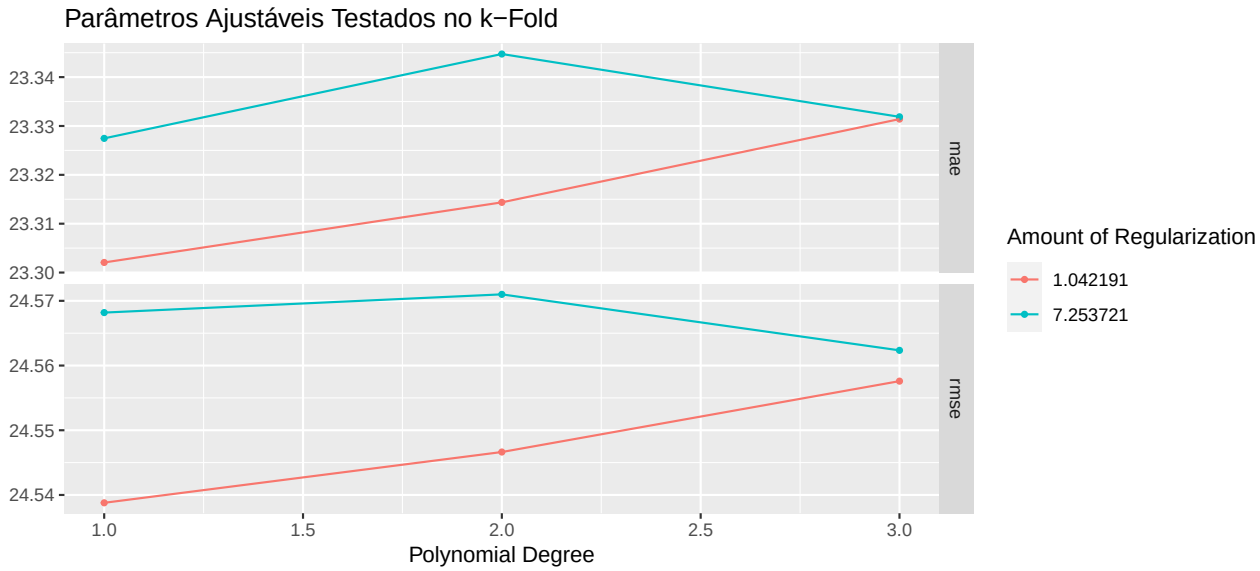
```
## # A tibble: 12 x 8
##    penalty degree .metric .estimator  mean     n std_err .config
##      <dbl>  <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1    1.04      1 mae     standard    22.3     5  0.111  Preprocessor1_Model2
##  2    1.04      1 rmse    standard    23.5     5  0.139  Preprocessor1_Model2
##  3    7.25      1 mae     standard    22.3     5  0.103  Preprocessor1_Model3
##  4    7.25      1 rmse    standard    23.5     5  0.127  Preprocessor1_Model3
##  5    1.04      2 mae     standard    22.4     5  0.115  Preprocessor2_Model2
##  6    1.04      2 rmse    standard    23.6     5  0.146  Preprocessor2_Model2
##  7    7.25      2 mae     standard    22.3     5  0.0603 Preprocessor2_Model3
##  8    7.25      2 rmse    standard    23.5     5  0.0983 Preprocessor2_Model3
##  9    1.04      3 mae     standard    22.3     5  0.135  Preprocessor3_Model2
## 10    1.04      3 rmse    standard    23.5     5  0.159  Preprocessor3_Model2
## 11    7.25      3 mae     standard    22.2     5  0.0935 Preprocessor3_Model3
## 12    7.25      3 rmse    standard    23.5     5  0.127  Preprocessor3_Model3
```

```
collect_metrics(glm_train_mc)
```

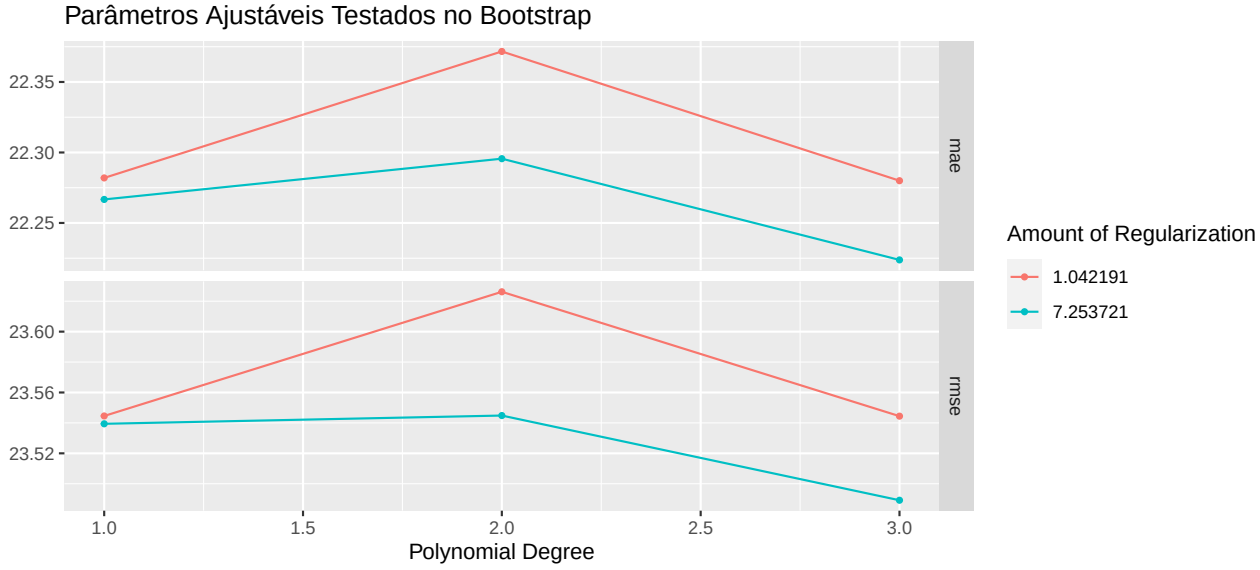
```
## # A tibble: 12 x 8
##    penalty degree .metric .estimator  mean     n std_err .config
##      <dbl>  <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1    1.04      1 mae     standard    23.2     3   0.458 Preprocessor1_Model2
##  2    1.04      1 rmse    standard    24.5     3   0.483 Preprocessor1_Model2
##  3    7.25      1 mae     standard    23.1     3   0.586 Preprocessor1_Model3
##  4    7.25      1 rmse    standard    24.5     3   0.618 Preprocessor1_Model3
##  5    1.04      2 mae     standard    23.2     3   0.558 Preprocessor2_Model2
##  6    1.04      2 rmse    standard    24.5     3   0.582 Preprocessor2_Model2
##  7    7.25      2 mae     standard    23.1     3   0.562 Preprocessor2_Model3
##  8    7.25      2 rmse    standard    24.5     3   0.594 Preprocessor2_Model3
##  9    1.04      3 mae     standard    23.2     3   0.563 Preprocessor3_Model2
## 10    1.04      3 rmse    standard    24.5     3   0.586 Preprocessor3_Model2
## 11    7.25      3 mae     standard    23.1     3   0.515 Preprocessor3_Model3
## 12    7.25      3 rmse    standard    24.5     3   0.527 Preprocessor3_Model3
```

Podemos visualizar os resultados nos plots a seguir.

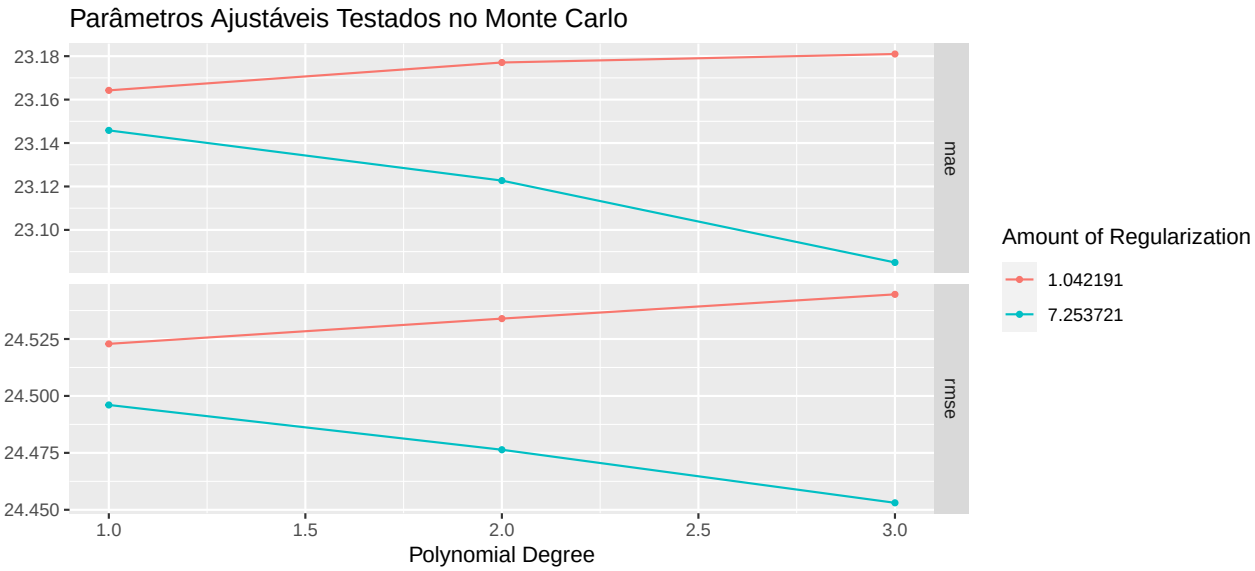
```
ggplot2::autoplot(glm_train_kfold) +
    labs(title = 'Parâmetros Ajustáveis Testados no k-Fold ') # %>% ggplotly()
```




```
ggplot2::autoplot(glm_train_boot) +
    labs(title = 'Parâmetros Ajustáveis Testados no Bootstrap')  # %>% ggplotly()
```




```
ggplot2::autoplot(glm_train_mc) +
    labs(title = 'Parâmetros Ajustáveis Testados no Monte Carlo')    # %>% ggplotly()
```



Agora, veremos e coletaremos os melhores conjuntos de parâmetros testados.

```
glm_train_kfold %>% show_best(n=1)
```

```
## # A tibble: 1 x 8
##   penalty degree .metric .estimator  mean     n std_err .config
##     <dbl>  <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    1.04      1 rmse    standard    24.5    20   0.450 Preprocessor1_Model2
```

```
glm_train_boot  %>% show_best(n=1)
```

```
## # A tibble: 1 x 8
##   penalty degree .metric .estimator  mean     n std_err .config
##     <dbl>  <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    7.25      3 rmse    standard    23.5     5   0.127 Preprocessor3_Model3
```

```
glm_train_mc  %>% show_best(n=1)
```

```
## # A tibble: 1 x 8
##   penalty degree .metric .estimator  mean     n std_err .config
##     <dbl>  <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    7.25      3 rmse    standard    24.5     3   0.527 Preprocessor3_Model3
```

```
best_tune_1  <- select_best(glm_train_kfold, 'rmse')
best_tune_2  <- select_best(glm_train_mc, 'rmse')
```

10 Aplicação do Melhor Modelo ao Conjunto de Teste

Como o melhor `penalty` é o mesmo em ambos os best's que escolhemos, farei somente um `finalize_model()` mas dois `finalize_recipe()` para o caso curioso que encontramos de dois `degree`.

```
final_algotim <- glm_algorit %>%
    finalize_model(parameters = best_tune_1 %>%
                     dplyr::select(`penalty`))
final_recip_1 <- recipe_glm %>%
    finalize_recipe(parameters = best_tune_1 %>%
                         dplyr::select(`degree`))
final_recip_2 <- recipe_glm %>%
    finalize_recipe(parameters = best_tune_2 %>%
                         dplyr::select(`degree`))

glm_final_wrkflw_1 <- workflow() %>%
    add_model(final_algotim) %>%
    add_recipe(final_recip_1) %>%
    last_fit(slice_1)

glm_final_wrkflw_2 <- workflow() %>%
    add_model(final_algotim) %>%
    add_recipe(final_recip_2) %>%
    last_fit(slice_1)

glm_final_wrkflw_1$.metrics
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard      23.5   Preprocessor1_Model1
## 2 rsq     standard       0.604 Preprocessor1_Model1
```

```
glm_final_wrkflw_1$.predictions
```

```
## [[1]]
## # A tibble: 98 x 4
##    .pred  .row   mpg .config
##    <dbl> <int> <dbl> <chr>
##  1 0.375     1    18 Preprocessor1_Model1
##  2 0.380     3    18 Preprocessor1_Model1
```

```
##  3 0.393     6    15 Preprocessor1_Model1
##  4 0.398     8    14 Preprocessor1_Model1
##  5 0.365    15    24 Preprocessor1_Model1
##  6 0.366    17    18 Preprocessor1_Model1
##  7 0.362    18    21 Preprocessor1_Model1
##  8 0.363    19    27 Preprocessor1_Model1
##  9 0.396    28    11 Preprocessor1_Model1
## 10 0.384    38    14 Preprocessor1_Model1
## # ... with 88 more rows
```

```
glm_final_wrkflw_2$.metrics
```

```
## [[1]]
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard     23.4    Preprocessor1_Model1
## 2 rsq     standard      0.0392 Preprocessor1_Model1
```

```
glm_final_wrkflw_2$.predictions
```

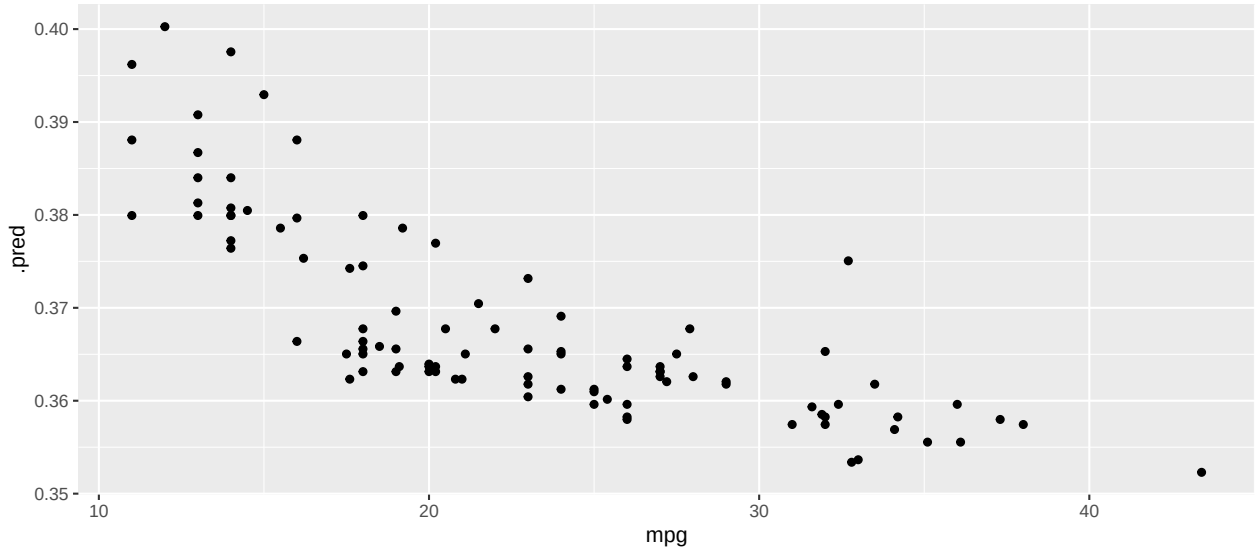
```
## [[1]]
## # A tibble: 98 x 4
##    .pred  .row   mpg .config
##    <dbl> <int> <dbl> <chr>
##  1 0.556     1    18 Preprocessor1_Model1
##  2 0.577     3    18 Preprocessor1_Model1
##  3 0.436     6    15 Preprocessor1_Model1
##  4 0.281     8    14 Preprocessor1_Model1
##  5 0.488    15    24 Preprocessor1_Model1
##  6 0.492    17    18 Preprocessor1_Model1
##  7 0.475    18    21 Preprocessor1_Model1
##  8 0.478    19    27 Preprocessor1_Model1
##  9 0.334    28    11 Preprocessor1_Model1
## 10 0.570    38    14 Preprocessor1_Model1
## # ... with 88 more rows
```

Vamos ficar somente com os estimados e os valores verdadeiros para vê-los plotados:

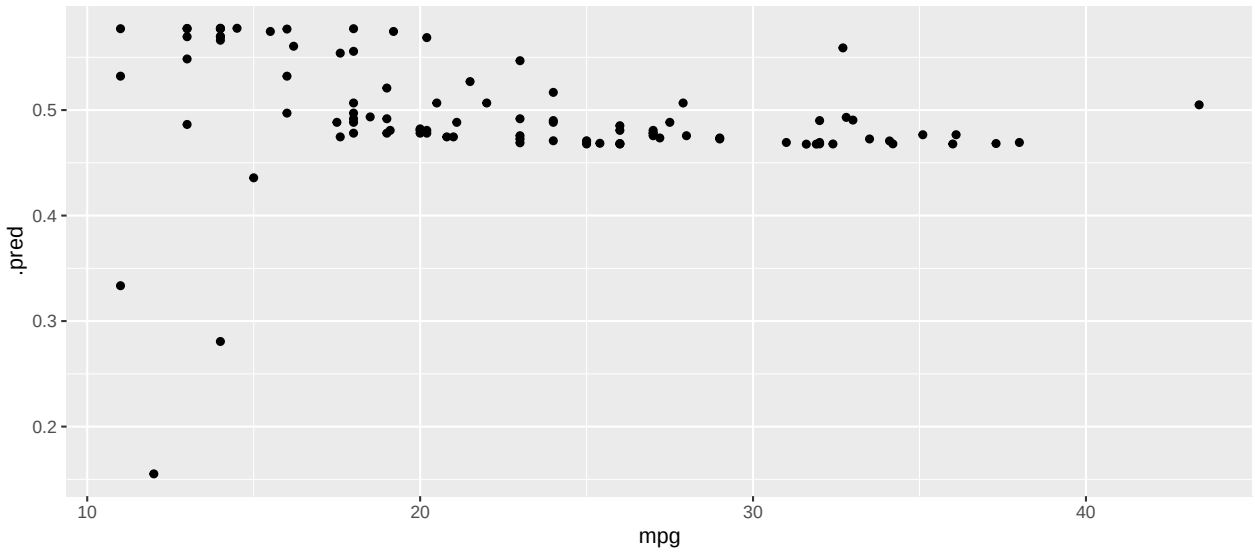
```
glm_final_wrkflw_1 <- glm_final_wrkflw_1 %>%
    collect_predictions()

glm_final_wrkflw_2 <- glm_final_wrkflw_2 %>%
    collect_predictions()
```

```
glm_final_wrkflw_1 %>%
    select(.row, .pred, mpg) %>%
    ggplot() +
    aes(x = mpg,
        y = .pred) +
    geom_point() +
    geom_abline(intercept = 0,
                slope     = 1,
                color     ='red',
                size      = .8) #%>% ggplotly()
```




```
glm_final_wrkflw_2 %>%
    select(.row, .pred, mpg) %>%
    ggplot() +
    aes(x = mpg,
        y = .pred) +
    geom_point() +
    geom_abline(intercept = 0,
                slope     = 1,
                color     ='lightseagreen',
                size      = .8) #%>%    ggplotly()
```



11 Referências

- Resampling for evaluating performance
- Samuel Macêdo - Intro ao Tidymodels
- Resampling Methods - ISLR tidymodels Labs
- Evaluate your model with resampling
- Bootstrap resampling and tidy regression models
- V-Fold Cross-Validation
- Leave-One-Out Cross-Validation
- Bootstrap Sampling
- Andrew MacDonald - bootstrapping regressions with dplyr