Neural Net Exercise

Heitor Gabriel S. Monteiro

06/10/2021

Contents

1 Prelúdio **1**

2 Importação e Divisão **2**

3 Modelo **2**

4 Tratamentos e Fórmula para Alimentar o Modelo **2**

5 Workflow **3**

6 Validação Cruzada **3**

7 Treinamento **3**

8 Testando **5**

1 Prelúdio

O pacote ensinado pelo livro do Lantz, o `neuralnet` está desatualizado e algumas funções importantes não estão funcionando mais. Aproveitando isso, resolvi fazer tudo usando o `nnet` mas dentro do `tidymodels`, para aprender a fazer já algo mais organizado e seriado.

```
setwd('/home/heitor/Área de Trabalho/R Projects/Análise Macro/Lab 9')
library(tidyverse)
library(tidymodels)
library(nnet)
library(NeuralNetTools)
```

2 Importação e Divisão

```
dds     <- read_csv("concrete.csv")
slice_1 <- initial_split(dds)
train   <- training(slice_1)
test    <- testing(slice_1)
```

3 Modelo

Vamos criar a estrutura geral do nosso modelo, deixando espaços livres com `tune()` por serem os parâmetros a serem testados com vários números, reiteragas vezes.

```
nnet_1 <- mlp(
    hidden_units = integer(tune()),
    penalty      = double(tune()),
    activation   = 'linear') %>%
    set_mode("regression") %>%
    set_engine("nnet", verbose = 0)
    # traduzir para pô-lo em termos de nnet:
nnet_1 %>% translate()
```

```
## Single Layer Neural Network Specification (regression)
##
## Main Arguments:
##   hidden_units = integer(tune())
##   penalty = double(tune())
##   activation = linear
##
## Engine-Specific Arguments:
##   verbose = 0
##
## Computational engine: nnet
##
## Model fit template:
## nnet::nnet(formula = missing_arg(), data = missing_arg(), weights = missing_arg(),
##     size = integer(tune()), decay = double(tune()), verbose = 0,
##     trace = FALSE, linout = TRUE)
```

4 Tratamentos e Fórmula para Alimentar o Modelo

Defino como os dados alimentarão o modelo já descrito acima e aplico um tratamento de normalização nos dados, usando desvio da média e desvio-padrão.

```
recipe_1 <- recipe(strength~.,
                   data = train) %>%
    step_normalize(all_numeric_predictors()) %>%
    prep()

recipe_1 %>% bake(new_data=NULL)
```

```
## # A tibble: 772 x 9
##     cement   slag    ash  water superplastic coarseagg fineagg    age strength
##      <dbl>  <dbl>  <dbl>  <dbl>        <dbl>     <dbl>   <dbl>  <dbl>    <dbl>
##  1  0.0569  1.35  -0.831  0.485       -1.02    -0.505   -0.681 -0.272     38.8
##  2  1.57   -0.602  0.402 -0.936        0.880   -0.0580  -0.754 -0.703     25.0
##  3  0.426  -0.881  1.32   0.816        0.749   -2.17     0.256 -0.272     38.6
##  4 -0.192   0.408  0.511  0.627       -0.203   -0.977   -0.491 -0.272     33.7
##  5  0.0569  1.35  -0.831  0.485       -1.02    -0.505   -0.681  0.798     50.5
##  6  0.881   1.32  -0.831 -0.552        0.634   -0.580   -0.193  0.211     63.4
##  7  0.239  -0.881 -0.831  0.532       -1.02    -0.0453   0.500  0.798     32.9
##  8 -1.20    1.86  -0.831  0.485       -1.02    -0.461    0.114 -0.634     16.9
##  9 -1.22    1.25   1.40   0.570        0.486   -1.17    -0.923 -0.272     29.0
## 10 -0.0868  0.338  0.449  1.34         0.453   -0.862   -1.16  -0.272     37.2
## # ... with 762 more rows
```

5 Workflow

Junto o modelo descrito e os dados tratados, formando um workflow:

```
nnet_1_wrkflw <- workflow() %>%
    add_model(nnet_1) %>%
    add_recipe(recipe_1)
```

6 Validação Cruzada

Defino a validação cruzada em grupos de cinco, ou seja, a amostra de treino será $\frac{4}{5}$ passando por várias reamostragens.

```
valid_1 <- vfold_cv(train, v = 5)
```

7 Treinamento

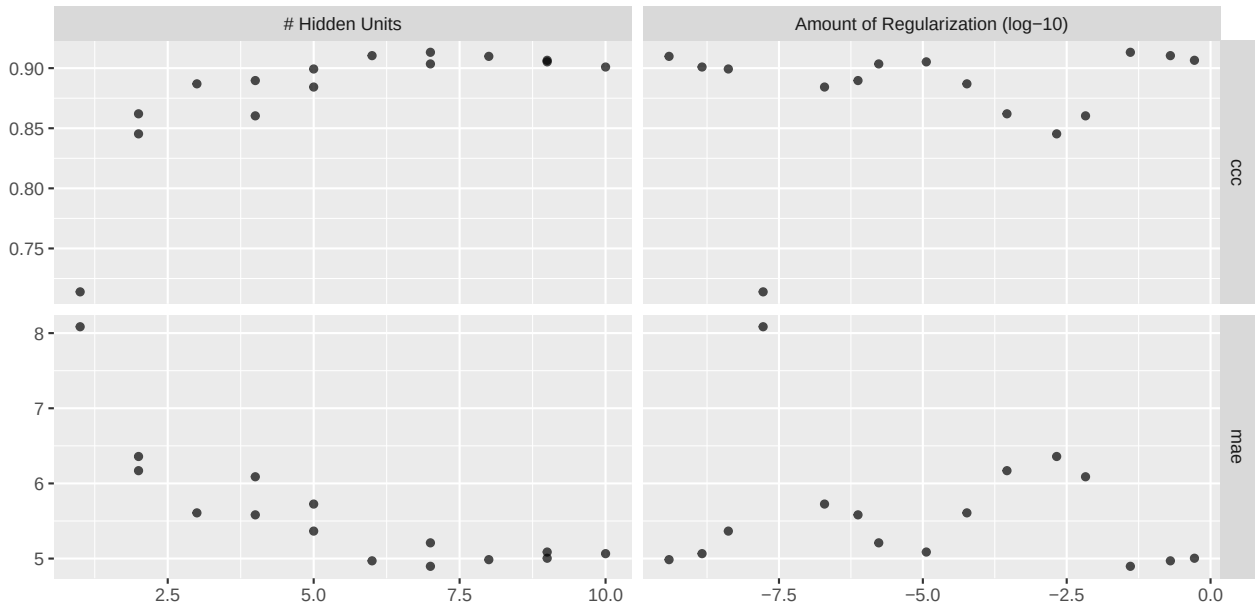
Treinaremos o modelo com vários parâmetros e selecionaremos de acordo com `ccc`: coeficiente de concordância de correlação. Mostraremos um gráfico com os parâmetros testados.

```r
nnet_1_trained <- nnet_1_wrkflw %>%
    tune_grid(valid_1,
              grid    = 15,
              control = control_grid(save_pred = T),
              metrics = metric_set(ccc, mae))

nnet_1_trained %>% show_best(n=15)
```

```
## # A tibble: 15 x 8
##    hidden_units  penalty .metric .estimator  mean     n std_err .config
##           <int>    <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1            7 4.04e- 2 ccc     standard   0.913     5 0.00728 Preprocessor1_M~
##  2            6 2.00e- 1 ccc     standard   0.910     5 0.00728 Preprocessor1_M~
##  3            8 3.89e-10 ccc     standard   0.910     5 0.00419 Preprocessor1_M~
##  4            9 5.25e- 1 ccc     standard   0.907     5 0.00750 Preprocessor1_M~
##  5            9 1.15e- 5 ccc     standard   0.905     5 0.00691 Preprocessor1_M~
##  6            7 1.71e- 6 ccc     standard   0.903     5 0.00686 Preprocessor1_M~
##  7           10 1.46e- 9 ccc     standard   0.901     5 0.00721 Preprocessor1_M~
##  8            5 4.17e- 9 ccc     standard   0.899     5 0.00929 Preprocessor1_M~
##  9            4 7.47e- 7 ccc     standard   0.890     5 0.00602 Preprocessor1_M~
## 10            3 5.83e- 5 ccc     standard   0.887     5 0.00557 Preprocessor1_M~
## 11            5 1.96e- 7 ccc     standard   0.884     5 0.00711 Preprocessor1_M~
## 12            2 2.90e- 4 ccc     standard   0.862     5 0.0151  Preprocessor1_M~
## 13            4 6.72e- 3 ccc     standard   0.860     5 0.0407  Preprocessor1_M~
## 14            2 2.12e- 3 ccc     standard   0.845     5 0.0247  Preprocessor1_M~
## 15            1 1.68e- 8 ccc     standard   0.714     5 0.0811  Preprocessor1_M~
```

```r
# (6.1) Auto-plot ---
ggplot2::autoplot(nnet_1_trained)
```



8 Testando

Selecionaremos o melhor modelo, usando o `ccc`.

```
best_tune  <- select_best(nnet_1_trained, 'ccc')
nnet_final <- nnet_1 %>%
    finalize_model(best_tune)
```

Aplicaremos esse modelo, `nnet_final` na partição feita em `slice_1` e com a organização dos dados de acordo com `recipe_1`. Vemos que conseguimos aumentar para 92.37% a correlação entre previsto e verdadeiro.

```
nnet_final_wrkflw <- workflow() %>%
    add_recipe(recipe_1) %>%
    add_model(nnet_final) %>%
    last_fit(slice_1) %>%
    collect_predictions()
nnet_final_wrkflw
```

```
## # A tibble: 258 x 5
##    id               .pred  .row strength .config
##    <chr>            <dbl> <int>    <dbl> <chr>
##  1 train/test split  28.0     1     29.9 Preprocessor1_Model1
##  2 train/test split  22.4     5     18.3 Preprocessor1_Model1
##  3 train/test split  25.7     9     21.6 Preprocessor1_Model1
##  4 train/test split  25.1    13     27.9 Preprocessor1_Model1
##  5 train/test split  46.4    16     50.0 Preprocessor1_Model1
##  6 train/test split  12.4    21     13.4 Preprocessor1_Model1
```

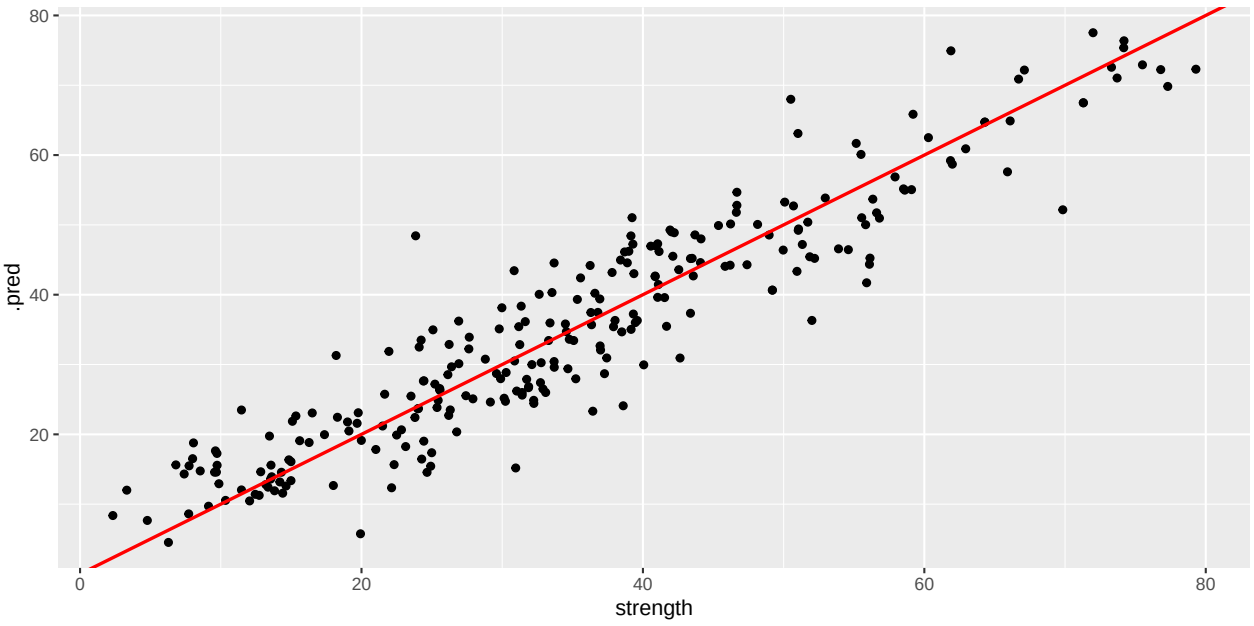
```
##  7 train/test split  33.4    24     33.3 Preprocessor1_Model1
##  8 train/test split  27.9    25     31.7 Preprocessor1_Model1
##  9 train/test split  47.2    32     39.3 Preprocessor1_Model1
## 10 train/test split  35.4    35     37.9 Preprocessor1_Model1
## # ... with 248 more rows
```

```
cor(nnet_final_wrkflw$.pred, nnet_final_wrkflw$strength)
```

```
## [1] 0.934763
```

Por fim, faço duas ilustrações: um plot dos dados originais contra os previstos, com a estimação perfeita sendo a linha vermelha; e a representação da rede neural.

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



Para fazer a representação da rede neural, precisamos traduzir o que foi feito em `mlp()` para colocar como um objeto `nnet` e fazer o gráfico.

```
nnet_final %>% translate()
```

```
## Single Layer Neural Network Specification (regression)
##
## Main Arguments:
##   hidden_units = 7
##   penalty = 0.0404219222790908
##   activation = linear
##
## Engine-Specific Arguments:
##   verbose = 0
##
## Computational engine: nnet
##
## Model fit template:
## nnet::nnet(formula = missing_arg(), data = missing_arg(), weights = missing_arg(),
##     size = 7L, decay = 0.0404219222790908, verbose = 0, trace = FALSE,
##     linout = TRUE)
```

```
nnet3 <-nnet(strength ~ cement+slag+ash+water+superplastic+coarseagg+fineagg+age,
             size = 8, # as penalty
             data = recipe(strength~., data = train) %>%
                step_normalize(all_numeric_predictors()) %>%
                prep() %>% bake(new_data=NULL),
             decay = 0.0218099079606513,
             verbose = 0, trace = FALSE,
             linout = TRUE)
nnet3 %>% plotnet()
```