Utilizando Redes Neurais para combinar modelos

J. Renato Leripio

09/08/2019

No post "Combinando modelos de previsão", apresentei duas formas simples de combinar modelos de previsão. A primeira delas envolvia obter uma combinação linear das previsões individuais de cada modelo, através de um método de otimização (OLS, por exemplo, ou equivalente). O segundo método utilizava a mediana das projeções dos modelos individuais. Embora operacionalmente mais simples, esta última abordagem performou melhor que a anterior – um resultado nada surpreendente, pois existe literatura mostrando que é mesmo difícil obter resultados melhores que a combinação por mediana daqueles modelos empregados.

Combinação de modelos tornou-se um novo paradigma em previsões. Conforme citado naquele post:

[...] a estratégia parece ter definido um novo padrão no campo de previsões uma vez que **12 dos 17 modelos mais acurados na competição M4 foram combinações**.

Na ocasião, mencionei também que haviam formas mais sofisticadas de combinar modelos. Segue a passagem:

[...] Métodos mais sofisticados permitem, por exemplo, mudanças no valor dos parâmetros ao longo do tempo e até mesmo a utilização de **algoritmos de Machine Learning** para aprender o valor destes parâmetros.

O post de hoje busca justamente isso: utilizar uma **rede neural** para combinar modelos. Para isto, vamos utilizar aqueles mesmos quatro modelos: **CES**, **ETS**, **DOTM** e **ARIMA**. Entretanto, optei desta vez por utilizar a série temporal do índice VIX, divulgado pelo CBOE (Chicago Board Options Exchange). Para os menos familiarizados, o índice VIX busca medir a volatilidade esperada na bolsa americana. A escolha da série, dentre outros motivos, justifica-se por seu tamanho: 355 observações mensais. Sabe-se que redes neurais são *data hungry*, logo séries curtas podem não apresentar bons resultados. A principal vantagem de utilizar uma rede neural para aprender qual a melhor forma de combinar os modelos é sua flexibilidade, isto é, os pesos atribuídos a cada modelo poderá ser uma função não-linear, potencialmente complexa.

Entendido o objetivo, vamos iniciar o exercício. De modo a poder contar com um número maior de observações, criei a amostra da seguinte forma:

1. Computei as projeções um-passo-à-frente de cada modelo através de validação-cruzada. Utilizando uma janela móvel de 100 observações para as projeções, obtive um total próximo a 250 previsões (fora da amostra).
2. Desta amostra, separei cerca de 70% para treino e 30% para teste. A amostra de treino foi utilizada para calcular os pesos por OLS e também para treinar a Rede Neural. A combinação por mediana foi calculada diretamente sobre a amostra de teste, pois não envolve nenhuma estimativa de parâmetros.
3. Por fim, foram computados o Mean Squared Error (MSE) de todos os métodos.

Os resultados seguem abaixo. Diferente dos resultados do post anterior, a mediana não foi capaz de superar todos os modelos individuais. Isto, provavelmente, é consequência de existirem dois modelos cujas performances se distanciam bastante dos demais – ARIMA e CES.

```
## 1. Carregar pacotes necessários

library(keras)
library(forecast)
library(tidyverse)
library(smooth)
library(forecTheta)
library(alfred)
library(lubridate)
```

```r
## 2. Importar os dados

vix <- alfred::get_fred_series(series_id = "VIXCLS",
                               series_name = "vix",
                               observation_end = "2019-07-31") %>%

  dplyr::mutate(date = lubridate::ymd(date)) %>%

  dplyr::mutate_at(vars(date), funs(year, month, day)) %>%

  dplyr::group_by(year, month) %>%

  dplyr::summarise(vix = mean(vix, na.rm = T))

## 3. Computar as previsões one-step-ahead para os modelos de referência

vix_ts <- ts(vix$vix, end = c(2019, 7), freq = 12)

modelo_i <- list(

  "ets" = function(x,h) forecast(ets(x), h = h),

  "ces" = function(x,h) forecast(smooth::auto.ces(x), h = h),

  "arima" = function(x,h) forecast(auto.arima(x), h = h),

  "dotm" = function(x,h) forecTheta::dotm(x, h = h)

)

fc_i <- purrr::map2(.f = forecast::tsCV,
                    .x = list(vix_ts),
                    .y = modelo_i,
                    h = 1, window = 100)

fc_i_mean <- fc_i %>%

  magrittr::set_names(names(modelo_i)) %>%

  purrr::map(.f = as_tibble) %>%

  plyr::ldply(bind_rows) %>%

  dplyr::group_by(.id) %>%

  dplyr::mutate(t = seq(from = 1, to = n(), by = 1)) %>%

  tidyr::spread(key = .id, value = x) %>%

  dplyr::mutate(y = vix_ts) %>%

  tidyr::drop_na() %>%
```

```
  dplyr::mutate_at(vars(-t, -y), ~ y - .) %>%

  dplyr::select(-t)

## 2. Definir uma amostra de treino e outra de teste

fc_i_treino <- fc_i_mean %>% dplyr::slice(1:174)

fc_i_teste <- fc_i_mean %>% dplyr::slice(175:n())

## 3. Observar acurácia dos modelos individuais e da mediana

acc_fc <- fc_i_teste %>%

  dplyr::rowwise() %>%

  dplyr::mutate(mediana = median(c(arima, ces, ets, dotm))) %>%

  tidyr::gather(key = modelo, value = y_hat, -y) %>%

  dplyr::group_by(modelo) %>%

  dplyr::summarise(MSE = mean((y - y_hat)^2)) %>%

  dplyr::arrange(MSE)

knitr::kable(acc_fc)
```

modelo	MSE
dotm	9.041486
ets	9.090884
mediana	9.120418
arima	9.361085
ces	10.519399

A combinação, por sua vez, também não foi capaz de entregar resultados superiores. Em outras palavras, na presença de modelos ruins, métodos mais convencionais de combinação tendem a não performar muito bem.

```
library(tidyverse)

## 4. Computar os pesos para combinação linear através de OLS

pesos <- lm(y ~ . -1, data = fc_i_treino)$coefficients

pesos_norm <- exp(pesos)/sum(exp(pesos))

acc_comb <- fc_i_teste %>%

  dplyr::mutate(comb = pesos_norm["arima"]*arima + pesos_norm["ces"]*ces +
                  pesos_norm["dotm"]*dotm + pesos_norm["ets"]*ets)

acc_comb_valor <- (acc_comb$y - acc_comb$comb) %>% `^`(2) %>% mean()
```

```r
acc_fc_aux <- acc_fc %>%

  tibble::add_row(modelo = "combinação",
                  MSE = acc_comb_valor) %>%

  dplyr::arrange(MSE)

knitr::kable(acc_fc_aux)
```

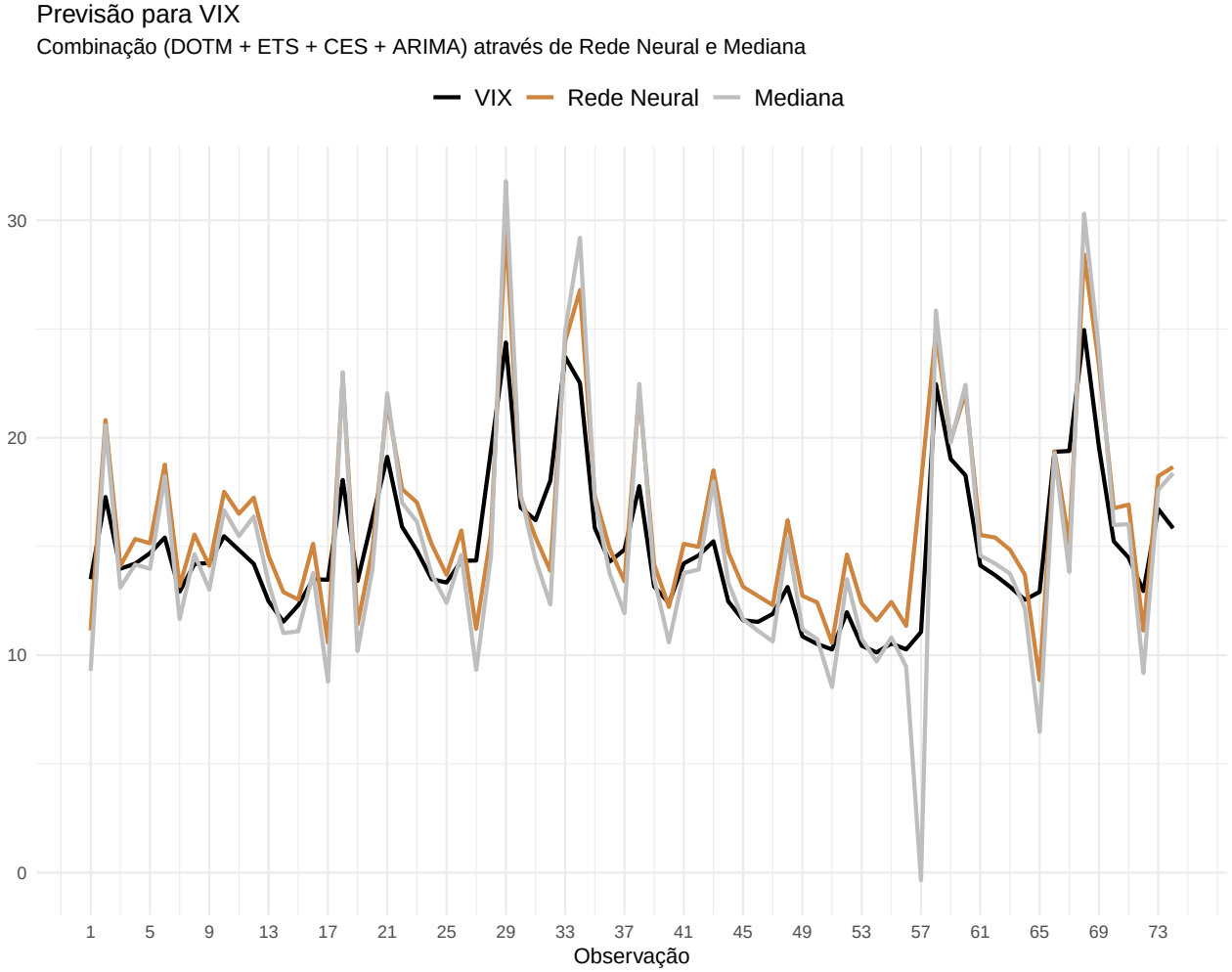
modelo	MSE
dotm	9.041486
ets	9.090884
mediana	9.120418
combinação	9.157094
arima	9.361085
ces	10.519399

E o que dizer da combinação através da rede neural? Antes de apresentar o resultado, vou descrever um pouco melhor a especificação utilizada. A rede neural usada aqui é do tipo *feed forward*, com quatro *layers* contendo 16 *units* cada. As funções de ativação dos três primeiros *layers* é *relu*, enquanto a do último é *linear*. No total, foram treinados 641 parâmetros. **A combinação através da rede neural retornou um MSE de 4.49**, cerca de metade daquele reportado pelo melhor modelo individual – DOTM. O resultado é muito bom, mas valem alguns avisos. O primeiro é usual, mas nunca demais repetir: não necessariamente este aumento de performance vai acontecer para qualquer série. O segundo é mais específico: o treinamento de redes neurais envolve alguns processos estocásticos, de modo que pode ocorrer instabilidade nos parâmetros estimados e, portanto, sobre a própria acurácia do modelo. Uma solução seria treinar a rede neural um grande número de vezes e reportar a acurácia mediana. Obviamente isto envolve um grande custo operacional.

```r
## 5. Treinar rede neural para combinação dos modelos

library(keras)
library(tidyverse)


build_nn <- keras_model_sequential() %>%

  layer_dense(units = 16, activation = "relu",
              input_shape = ncol(fc_i_treino)-1) %>%

  layer_dense(units = 16, activation = "relu") %>%

  layer_dense(units = 16, activation = "relu") %>%

  layer_dense(units = 1, activation = "linear")

compile_nn <-  build_nn %>%

  compile(loss = "mse",

          optimizer = optimizer_rmsprop(),

          metrics = list("mean_squared_error"))
```

```
early_stop <- callback_early_stopping(monitor = "val_loss",
                                      patience = 20)

modelo_treinado <- compile_nn %>%

  fit(as.matrix(fc_i_treino[, c(1:4)]),

      as.matrix(fc_i_treino[, 5]),

      epochs = 1000,

      validation_split = 0.25,

      verbose = 0,

      callbacks = list(early_stop))

mse <- compile_nn %>%

  evaluate(as.matrix(fc_i_teste[, c(1:4)]),
           as.matrix(fc_i_teste[, 5]),
           verbose = 0)

nn_fc <- compile_nn %>% predict(as.matrix(fc_i_teste[, 1:4]))


fc_mediana <- fc_i_teste %>%

  dplyr::rowwise() %>%

  dplyr::mutate(mediana = median(c(arima, ces, ets, dotm)))

fc_tibble <- tibble::tibble(obs = 1:74,
                            "VIX" = acc_comb$y,
                            "Combinação linear" = acc_comb$comb,
                            "Mediana" = fc_mediana$mediana,
                            "Rede Neural" = nn_fc)

fc_tibble %>%

  tidyr::gather(key = modelo, value = valor, -obs) %>%

  dplyr::filter(modelo %in% c("VIX", "Mediana", "Rede Neural")) %>%

  ggplot(aes(x = obs, y = valor,
             color = factor(modelo, levels = c("VIX",
                                               "Rede Neural",
                                               "Mediana")))) +

  geom_line(lwd = 1) +

  scale_color_manual(values = c("black", "peru", "grey")) +
```

```
  labs(x = "Observação",
       y = "", color = "",
       title = "Previsão para VIX",
       subtitle = "Combinação (DOTM + ETS + CES + ARIMA) através de Rede Neural e Mediana") +

  theme_minimal() +

  scale_x_continuous(breaks = seq(1,74,4)) +

  theme(legend.position = "top",
        legend.text = element_text(size = 12))
```



O que chama mais atenção nos resultados é a capacidade da rede neural em evitar as projeções mais acentuadas para baixo. Note que em diversos casos isto faz com que a projeção se distancie muito do valor realizado. A princípio, tudo indica que a rede neural foi capaz de aprender a não dar muito peso para o modelo (ou os modelos) com essa tendência de subestimar demais a série em certos momentos. Adicionalmente, a combinação através da rede neural também não superestima demais alguns valores. Em suma, parece que em situações mais usuais ambos os métodos são razoáveis. Porém, na presença de desvios mais acentuados, a rede neural é capaz de entregar resultados mais próximos do observado. Essa é uma virtude importante, sobretudo por se tratar de uma série que mede volatilidade e, portanto, apresenta regiões mais "nervosas". Neste caso, é importante que o modelo não perca a mão quando esse momento aparecer.

Por fim, uma possível estratégia para melhorar o método de combinação linear ou de mediana seria remover

aquele modelo com desempenho muito ruim – no nosso caso, o CES. Uma generalização desse processo seria estimar diversos (ou todos) subconjuntos de modelos, calcular a acurácia da combinação de cada subconjunto e selecionar aquele com melhor poder preditivo. Talvez possamos abordar isso numa próxima oportunidade.

Ficou alguma dúvida ou tem sugestões? Entre em contato!

Os códigos dos exercícios encontram-se disponíveis no repositório do blog no github.

Siga nossa página **RLeripio – Economia e Data Science** no Facebook e fique sabendo de todas as nossas publicações!

Aviso legal: Todo o conteúdo desta página é de responsabilidade pessoal do autor e não expressa a visão da instituição a qual o autor tem vínculo profissional.