Analisis overfitting

Jimena Murillo

2022-06-15

Paquetes

```
library(keras) # for deep learning
library(tidyverse) # general utility functions
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(caret) # machine learning utility functions
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(tibble)
library(readr)
library(ggplot2)
library(tensorflow)
```

```
##
## Attaching package: 'tensorflow'

## The following object is masked from 'package:caret':
##
##     train
```

```
library(neuralnet)
```

```
##
## Attaching package: 'neuralnet'
```

```
## The following object is masked from 'package:dplyr':
##
##     compute
```

Datos

```
load("C:/Users/usuario1/Desktop/CIMPA/Github_CIMPA/PRACTICA_CIMPA/base_cantones.RData")


Alajuela <- basecanton %>% filter(Canton == "Alajuela") %>%

  dplyr::select(Year,Month,Nino12SSTA, Nino3SSTA, Nino4SSTA,Nino34SSTA,Nino34SSTA1, Nino34SSTA2, Nino34S

  arrange(Year,Month) %>% ungroup() %>% mutate(Month=as.numeric(Month))


if(anyNA(Alajuela)){
  Alajuela <- na.omit(Alajuela)
}

#Escala


normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

max <- apply(Alajuela,2,max)
min <- apply(Alajuela,2,min)

Alajuela2 <- apply(Alajuela, 2, normalize)
```

```
#Train y test

Fechas = c(1, 0.95, 0.90, 0.85, 0.80, 0.75, 0.70, 0.65, 0.60, 0.55, 0.50)
Eval = matrix(data = NA, nrow = length(Fechas), ncol = 2)
Evalc = matrix(data = NA, nrow = length(Fechas), ncol = 2)

for (i in 1:length(Fechas)) {

data_train = as.data.frame(Alajuela2) %>% filter(Year < Fechas[i])#PARA ENTRENAR HASTA 2018
data_test = as.data.frame(Alajuela2) %>% filter(Year >= Fechas[i])

X_train = as.matrix(data_train[,-ncol(data_train)])
y_train = as.matrix(data_train[,ncol(data_train)])
```

```
X_test = as.matrix(data_test[,-ncol(data_test)])
y_test = as.matrix(data_test[,ncol(data_test)])



## Generar un Wrapper para el learning dropout



# R6 wrapper class, a subclass of KerasWrapper
ConcreteDropout <- R6::R6Class("ConcreteDropout",

  inherit = KerasWrapper,

  public = list(
    weight_regularizer = NULL,
    dropout_regularizer = NULL,
    init_min = NULL,
    init_max = NULL,
    is_mc_dropout = NULL,
    supports_masking = TRUE,
    p_logit = NULL,
    p = NULL,

    initialize = function(weight_regularizer,
                          dropout_regularizer,
                          init_min,
                          init_max,
                          is_mc_dropout) {
      self$weight_regularizer <- weight_regularizer
      self$dropout_regularizer <- dropout_regularizer
      self$is_mc_dropout <- is_mc_dropout
      self$init_min <- k_log(init_min) - k_log(1 - init_min)
      self$init_max <- k_log(init_max) - k_log(1 - init_max)
    },

    build = function(input_shape) {
      super$build(input_shape)

      self$p_logit <- super$add_weight(
        name = "p_logit",
        shape = shape(1),
        initializer = initializer_random_uniform(self$init_min, self$init_max),
        trainable = TRUE
      )

      self$p <- k_sigmoid(self$p_logit)

      input_dim <- input_shape[[2]]

      weight <- private$py_wrapper$layer$kernel

      kernel_regularizer <- self$weight_regularizer *
```

```
                            k_sum(k_square(weight)) /
                            (1 - self$p)

      dropout_regularizer <- self$p * k_log(self$p)
      dropout_regularizer <- dropout_regularizer +
                             (1 - self$p) * k_log(1 - self$p)
      dropout_regularizer <- dropout_regularizer *
                             self$dropout_regularizer *
                             k_cast(input_dim, k_floatx())

      regularizer <- k_sum(kernel_regularizer + dropout_regularizer)
      super$add_loss(regularizer)
    },

    concrete_dropout = function(x) {
      eps <- k_cast_to_floatx(k_epsilon())
      temp <- 0.1

      unif_noise <- k_random_uniform(shape = k_shape(x))

      drop_prob <- k_log(self$p + eps) -
                   k_log(1 - self$p + eps) +
                   k_log(unif_noise + eps) -
                   k_log(1 - unif_noise + eps)
      drop_prob <- k_sigmoid(drop_prob / temp)

      random_tensor <- 1 - drop_prob

      retain_prob <- 1 - self$p
      x <- x * random_tensor
      x <- x / retain_prob
      x
    },

    call = function(x, mask = NULL, training = NULL) {
      if (self$is_mc_dropout) {
        super$call(self$concrete_dropout(x))
      } else {
        k_in_train_phase(
          function()
            super$call(self$concrete_dropout(x)),
          super$call(x),
          training = training
        )
      }
    }
  )
)

# function for instantiating custom wrapper
layer_concrete_dropout <- function(object,
                                   layer,
                                   weight_regularizer = 1e-6,
```

```
                                    dropout_regularizer = 1e-5,
                                    init_min = 0.1,
                                    init_max = 0.1,
                                    is_mc_dropout = TRUE,
                                    name = NULL,
                                    trainable = TRUE) {
  create_wrapper(ConcreteDropout, object, list(
    layer = layer,
    weight_regularizer = weight_regularizer,
    dropout_regularizer = dropout_regularizer,
    init_min = init_min,
    init_max = init_max,
    is_mc_dropout = is_mc_dropout,
    name = name,
    trainable = trainable
  ))
}

# sample size (training data)
n_train <- nrow(data_train)
# sample size (validation data)
n_val <- nrow(data_test)
# prior length-scale
l <- 1e-4
# initial value for weight regularizer
wd <- l^2/n_train
# initial value for dropout regularizer
dd <- 2/n_train



## Modelo Dropout


# we use one-dimensional input data here, but this isn't a necessity
input_dim <- 32
# this too could be > 1 if we wanted
output_dim <- 1


input <- layer_input(shape = input_dim)

output <- input %>% layer_concrete_dropout(
  layer = layer_dense(units = 100, activation = "relu"),
  weight_regularizer = wd,
  dropout_regularizer = dd
  ) %>% layer_concrete_dropout(
  layer = layer_dense(units = 50, activation = "relu"),
  weight_regularizer = wd,
  dropout_regularizer = dd
  ) %>% layer_concrete_dropout(
  layer = layer_dense(units = 50, activation = "relu"),
  weight_regularizer = wd,
```

```
  dropout_regularizer = dd
) %>% layer_concrete_dropout(
  layer = layer_dense(units = 50, activation = "relu"),
  weight_regularizer = wd,
  dropout_regularizer = dd
) %>% layer_concrete_dropout(
  layer = layer_dense(units = 25, activation = "relu"),
  weight_regularizer = wd,
  dropout_regularizer = dd
) %>% layer_concrete_dropout(
  layer = layer_dense(units = 25, activation = "relu"),
  weight_regularizer = wd,
  dropout_regularizer = dd
) %>% layer_concrete_dropout(
  layer = layer_dense(units = 25, activation = "relu"),
  weight_regularizer = wd,
  dropout_regularizer = dd
) %>% layer_concrete_dropout(
  layer = layer_dense(units = 12, activation = "relu"),
  weight_regularizer = wd,
  dropout_regularizer = dd
) %>% layer_concrete_dropout(
  layer = layer_dense(units = 12, activation = "relu"),
  weight_regularizer = wd,
  dropout_regularizer = dd
) %>% layer_concrete_dropout(
  layer = layer_dense(units = 6, activation = "relu"),
  weight_regularizer = wd,
  dropout_regularizer = dd
) %>% layer_concrete_dropout(
  layer = layer_dense(units = 6, activation = "relu"),
  weight_regularizer = wd,
  dropout_regularizer = dd
)


## Output del Modelo


mean <- output %>% layer_concrete_dropout(
  layer = layer_dense(units = output_dim),
  weight_regularizer = wd,
  dropout_regularizer = dd
)

log_var <- output %>% layer_concrete_dropout(
  layer_dense(units = output_dim),
  weight_regularizer = wd,
  dropout_regularizer = dd
)

output <- layer_concatenate(list(mean, log_var))
```

```r
model <- keras_model(input, output)


## Entrenar al modelo


model %>% compile(
  optimizer = "adam",
  loss = "mse",
  metrics = "mae")

history <- model %>% fit(
  X_train,
  y_train,
  epochs = 100,
  batch_size = 18,
  validation_split = 0.1
)




## MonteCarlo sampling


denorm <- function(x) {
  return (x*(max(Alajuela$RR) - min(Alajuela$RR))+min(Alajuela$RR))
}


num_MC_samples <- 100

MC_samples <- array(0, dim = c(num_MC_samples, nrow(X_test), 2 * output_dim))
for (k in 1:num_MC_samples) {
  MC_samples[k, , ] <- denorm((model %>% predict(X_test)))
}


## Generar intervalo de confianza


# First, we determine the predictive mean as an average of the MC samples' mean output:

# the means are in the first output column
means <- NULL
means <- MC_samples[, , 1:output_dim]
# average over the MC samples
predictive_mean <- apply(means, 2, mean)



# To calculate epistemic uncertainty, we again use the mean output, but this time we're interested in t
```

```r
epistemic_uncertainty <- apply(means, 2, var)



# Then aleatoric uncertainty is the average over the MC samples of the variance output. 1 .

logvar = NULL
logvar <- MC_samples[, , (output_dim + 1):(output_dim * 2)]
aleatoric_uncertainty <- exp(colMeans(logvar))

y_test = denorm(y_test)
# Note how this procedure gives us uncertainty estimates individually for every prediction. How do they

df1 <- data.frame(
  x = 1:nrow(X_test),
  y = y_test,
  y_pred = predictive_mean,
  e_u_lower = predictive_mean - sqrt(epistemic_uncertainty),
  e_u_upper = predictive_mean + sqrt(epistemic_uncertainty),
  a_u_lower = predictive_mean - sqrt(aleatoric_uncertainty),
  a_u_upper = predictive_mean + sqrt(aleatoric_uncertainty),
  u_overall_lower = predictive_mean -
                    sqrt(epistemic_uncertainty) -
                    sqrt(aleatoric_uncertainty),
  u_overall_upper = predictive_mean +
                    sqrt(epistemic_uncertainty) +
                    sqrt(aleatoric_uncertainty)
)



#Here, first, is epistemic uncertainty, with shaded bands indicating one standard deviation above resp.

ggplot(df1, aes(x, y_pred)) +
  geom_line(colour = "blue") +
  geom_line( aes(x, y = y_test, colour = "red"))+
  geom_ribbon(aes(ymin = e_u_lower, ymax = e_u_upper), alpha = 0.3)



metricas <- function(tabla){
  NRMSE <- mean((tabla$y_pred-tabla$y)^2)/mean(tabla$y)
  NIS_95 <- mean((tabla$e_u_upper-tabla$e_u_lower)+
                   (2/0.05)*(tabla$e_u_lower-tabla$y)*(tabla$y<tabla$e_u_lower)+
                   (2/0.05)*(tabla$y-tabla$e_u_upper)*(tabla$y>tabla$e_u_upper))/mean(tabla$y)
  return(data.frame(NRMSE,NIS_95))
}

Eval[i, 1:2] = as.numeric(metricas(df1))



#Valores aproximados
```

```r
num_MC_samples <- 100

MC_samples <- array(0, dim = c(num_MC_samples, nrow(Alajuela), 2 * output_dim))
for (k in 1:num_MC_samples) {
  MC_samples[k, , ] <- denorm((model %>% predict(Alajuela2[,-33])))
}


## Generar intervalo de confianza


# First, we determine the predictive mean as an average of the MC samples' mean output:

# the means are in the first output column
means <- NULL
means <- MC_samples[, , 1:output_dim]
# average over the MC samples
predictive_mean <- apply(means, 2, mean)



# To calculate epistemic uncertainty, we again use the mean output, but this time we're interested in t

epistemic_uncertainty <- apply(means, 2, var)



# Then aleatoric uncertainty is the average over the MC samples of the variance output. 1 .

logvar = NULL
logvar <- MC_samples[, , (output_dim + 1):(output_dim * 2)]
aleatoric_uncertainty <- exp(colMeans(logvar))


# Note how this procedure gives us uncertainty estimates individually for every prediction. How do they

df1 <- data.frame(
  x = 1:nrow(Alajuela),
  y = Alajuela$RR,
  y_pred = predictive_mean,
  e_u_lower = predictive_mean - sqrt(epistemic_uncertainty),
  e_u_upper = predictive_mean + sqrt(epistemic_uncertainty),
  a_u_lower = predictive_mean - sqrt(aleatoric_uncertainty),
  a_u_upper = predictive_mean + sqrt(aleatoric_uncertainty),
  u_overall_lower = predictive_mean -
                    sqrt(epistemic_uncertainty) -
                    sqrt(aleatoric_uncertainty),
  u_overall_upper = predictive_mean +
                    sqrt(epistemic_uncertainty) +
                    sqrt(aleatoric_uncertainty)
)
```

```
#Here, first, is epistemic uncertainty, with shaded bands indicating one standard deviation above resp.

ggplot(df1, aes(x, y_pred)) +
  geom_line(colour = "blue") +
  geom_line( aes(x, y = Alajuela$RR, colour = "red"))+
  geom_ribbon(aes(ymin = e_u_lower, ymax = e_u_upper), alpha = 0.3)



metricas <- function(tabla){
  NRMSE <- mean((tabla$y_pred-tabla$y)^2)/mean(tabla$y)
  NIS_95 <- mean((tabla$e_u_upper-tabla$e_u_lower)+
                   (2/0.05)*(tabla$e_u_lower-tabla$y)*(tabla$y<tabla$e_u_lower)+
                   (2/0.05)*(tabla$y-tabla$e_u_upper)*(tabla$y>tabla$e_u_upper))/mean(tabla$y)
  return(data.frame(NRMSE,NIS_95))
}

Evalc[i, 1:2] = as.numeric(metricas(df1))

}
```

```
## Loaded Tensorflow version 2.8.0
```

Resultados

```
Eval
```

```
##            [,1]      [,2]
##  [1,] 0.1266148  4.058797
##  [2,] 0.9587178 27.393274
##  [3,] 0.4597486 13.074184
##  [4,] 0.4461742 20.377201
##  [5,] 0.5166130 10.321580
##  [6,] 0.6004216 10.725275
##  [7,] 0.6258576 10.348060
##  [8,] 0.5649877  9.016314
##  [9,] 0.4522885  7.341923
## [10,] 0.5508476  8.938748
## [11,] 1.0313515 18.237314
```

```
Evalc
```

```
##            [,1]      [,2]
##  [1,] 0.3982420 10.184789
##  [2,] 0.3979075  9.471780
##  [3,] 0.3696414  8.369108
##  [4,] 0.9357669 27.328770
##  [5,] 0.4547781  8.784812
##  [6,] 0.4485212  9.441314
```

```
##  [7,] 0.4906603 10.483206
##  [8,] 0.5061812 10.431880
##  [9,] 0.4455770  8.061086
## [10,] 0.5489292 10.164983
## [11,] 0.8695342 15.639672
```