**Introduction**

In this blog post, I’ll use the data that I cleaned in a previous  
[blog post](https://www.brodrigues.co/blog/2018-11-14-luxairport/), which you can download  
[here](https://github.com/b-rodrigues/avia_par_lu/tree/master). If you want to follow along,  
download the monthly data. In my [last blog post](https://www.brodrigues.co/blog/2018-11-15-tidy_gridsearch/)  
I showed how to perform a grid search the “tidy” way. As an example, I looked for the right  
hyperparameters of a SARIMA model. However, the goal of the post was not hyperparameter optimization  
per se, so I did not bother with tuning the hyperparameters on a validation set, and used the test  
set for both validation of the hyperparameters and testing the forecast. Of course, this is not great  
because doing this might lead to overfitting the hyperparameters to the test set. So in this blog post  
I split my data into trainig, validation and testing sets and use a genetic algorithm to look  
for the hyperparameters. Again, this is not the most optimal way to go about this problem, since  
the {forecast} package contains the very useful auto.arima() function. I just wanted to see  
what kind of solution a genetic algorithm would return, and also try different cost functions.  
If you’re interested, read on!

**Setup**

Let’s first load some libraries and define some helper functions (the helper functions were explained  
in the previous blog posts):

library(tidyverse)

library(forecast)

library(rgenoud)

library(parallel)

library(lubridate)

library(furrr)

library(tsibble)

library(brotools)

ihs <- function(x){

log(x + sqrt(x\*\*2 + 1))

}

to\_tibble <- function(forecast\_object){

point\_estimate <- forecast\_object$mean %>%

as\_tsibble() %>%

rename(point\_estimate = value,

date = index)

upper <- forecast\_object$upper %>%

as\_tsibble() %>%

spread(key, value) %>%

rename(date = index,

upper80 = `80%`,

upper95 = `95%`)

lower <- forecast\_object$lower %>%

as\_tsibble() %>%

spread(key, value) %>%

rename(date = index,

lower80 = `80%`,

lower95 = `95%`)

reduce(list(point\_estimate, upper, lower), full\_join)

}

Now, let’s load the data:

avia\_clean\_monthly <- read\_csv("https://raw.githubusercontent.com/b-rodrigues/avia\_par\_lu/master/avia\_clean\_monthy.csv")

## Parsed with column specification:

## cols(

## destination = col\_character(),

## date = col\_date(format = ""),

## passengers = col\_integer()

## )

Let’s split the data into a train set, a validation set and a test set:

avia\_clean\_train <- avia\_clean\_monthly %>%

select(date, passengers) %>%

filter(year(date) < 2013) %>%

group\_by(date) %>%

summarise(total\_passengers = sum(passengers)) %>%

pull(total\_passengers) %>%

ts(., frequency = 12, start = c(2005, 1))

avia\_clean\_validation <- avia\_clean\_monthly %>%

select(date, passengers) %>%

filter(between(year(date), 2013, 2016)) %>%

group\_by(date) %>%

summarise(total\_passengers = sum(passengers)) %>%

pull(total\_passengers) %>%

ts(., frequency = 12, start = c(2013, 1))

avia\_clean\_test <- avia\_clean\_monthly %>%

select(date, passengers) %>%

filter(year(date) >= 2016) %>%

group\_by(date) %>%

summarise(total\_passengers = sum(passengers)) %>%

pull(total\_passengers) %>%

ts(., frequency = 12, start = c(2016, 1))

logged\_test\_data <- ihs(avia\_clean\_test)

logged\_validation\_data <- ihs(avia\_clean\_validation)

logged\_train\_data <- ihs(avia\_clean\_train)

I will train the models on data from 2005 to 2012, look for the hyperparameters on data from 2013  
to 2016 and test the accuracy on data from 2016 to March 2018. For this kind of exercise, the ideal  
situation would be to perform cross-validation. Doing this with time-series data is not obvious  
because of the autocorrelation between observations, which would be broken by sampling independently  
which is required by CV. Also, if for example you do leave-one-out CV,  
you would end up trying to predict a point in, say, 2017, with data  
from 2018, which does not make sense. So you should be careful about that. {forecast} is able  
to perform [CV for time series](https://robjhyndman.com/hyndsight/tscv/) and scikit-learn, the  
Python package, is able to perform  
[cross-validation of time series data](https://scikit-learn.org/stable/modules/cross_validation.html#time-series-split)  
too. I will not do it in this blog post and simply focus on the genetic algorithm part.

Let’s start by defining the cost function to minimize. I’ll try several, in the first one I will  
minimize the RMSE:

cost\_function\_rmse <- function(param, train\_data, validation\_data, forecast\_periods){

order <- param[1:3]

season <- c(param[4:6], 12)

model <- purrr::possibly(arima, otherwise = NULL)(x = train\_data, order = order,

seasonal = season,

method = "ML")

if(is.null(model)){

return(9999999)

} else {

forecast\_model <- forecast::forecast(model, h = forecast\_periods)

point\_forecast <- forecast\_model$mean

sqrt(mean(point\_forecast - validation\_data) \*\* 2)

}

}

If arima() is not able to estimate a model for the given parameters, I force it to return NULL,  
and in that case force the cost function to return a very high cost. If a model was successfully estimated,  
then I compute the RMSE.

Let’s also take a look at what auto.arima() says:

starting\_model <- auto.arima(logged\_train\_data)

summary(starting\_model)

## Series: logged\_train\_data

## ARIMA(1,0,2)(2,1,0)[12]

##

## Coefficients:

## ar1 ma1 ma2 sar1 sar2

## 0.9754 -0.7872 0.2091 -0.7285 -0.4413

## s.e. 0.0261 0.1228 0.1213 0.1063 0.1150

##

## sigma^2 estimated as 0.004514: log likelihood=105.61

## AIC=-199.22 AICc=-198.13 BIC=-184.64

##

## Training set error measures:

## ME RMSE MAE MPE MAPE

## Training set 0.008398036 0.06095102 0.03882593 0.07009285 0.3339574

## MASE ACF1

## Training set 0.4425794 0.02073886

Let’s compute the cost at this vector of parameters:

cost\_function\_rmse(c(1, 0, 2, 2, 1, 0),

train\_data = logged\_train\_data,

validation\_data = logged\_validation\_data,

forecast\_periods = 65)

## [1] 0.1731473

Ok, now let’s start with optimizing the hyperparameters. Let’s help the genetic algorithm a little  
bit by defining where it should perform the search:

domains <- matrix(c(0, 3, 0, 2, 0, 3, 0, 3, 0, 2, 0, 3), byrow = TRUE, ncol = 2)

This matrix constraints the first parameter to lie between 0 and 3, the second one between 0 and 2,  
and so on.

Let’s call the genoud() function from the {rgenoud} package, and use 8 cores:

cl <- makePSOCKcluster(8)

clusterExport(cl, c('logged\_train\_data', 'logged\_validation\_data'))

tic <- Sys.time()

auto\_arima\_rmse <- genoud(cost\_function\_rmse,

nvars = 6,

data.type.int = TRUE,

starting.values = c(1, 0, 2, 2, 1, 0), # <- from auto.arima

Domains = domains,

cluster = cl,

train\_data = logged\_train\_data,

validation\_data = logged\_validation\_data,

forecast\_periods = length(logged\_validation\_data),

hard.generation.limit = TRUE)

toc\_rmse <- Sys.time() - tic

makePSOCKcluster() is a function from the {parallel} package. I must also *export* the global  
variables logged\_train\_data or logged\_validation\_data. If I don’t do that, the workers called  
by genoud() will not *know* about these variables and an error will be returned. The option  
data.type.int = TRUE force the algorithm to look only for integers, and hard.generation.limit = TRUE  
forces the algorithm to stop after 100 generations.

The process took 7 minutes, which is faster than doing the grid search.  
What was the solution found?

auto\_arima\_rmse

## $value

## [1] 0.0001863039

##

## $par

## [1] 3 2 1 1 2 1

##

## $gradients

## [1] NA NA NA NA NA NA

##

## $generations

## [1] 11

##

## $peakgeneration

## [1] 1

##

## $popsize

## [1] 1000

##

## $operators

## [1] 122 125 125 125 125 126 125 126 0

Let’s train the model using the arima() function at these parameters:

best\_model\_rmse <- arima(logged\_train\_data, order = auto\_arima\_rmse$par[1:3],

season = list(order = auto\_arima\_rmse$par[4:6], period = 12),

method = "ML")

summary(best\_model\_rmse)

##

## Call:

## arima(x = logged\_train\_data, order = auto\_arima\_rmse$par[1:3], seasonal = list(order = auto\_arima\_rmse$par[4:6],

## period = 12), method = "ML")

##

## Coefficients:

## ar1 ar2 ar3 ma1 sar1 sma1

## -0.6999 -0.4541 -0.0476 -0.9454 -0.4996 -0.9846

## s.e. 0.1421 0.1612 0.1405 0.1554 0.1140 0.2193

##

## sigma^2 estimated as 0.006247: log likelihood = 57.34, aic = -100.67

##

## Training set error measures:

## ME RMSE MAE MPE MAPE

## Training set -0.0006142355 0.06759545 0.04198561 -0.005408262 0.3600483

## MASE ACF1

## Training set 0.4386693 -0.008298546

Let’s extract the forecasts:

best\_model\_rmse\_forecast <- forecast::forecast(best\_model\_rmse, h = 65)

best\_model\_rmse\_forecast <- to\_tibble(best\_model\_rmse\_forecast)

## Joining, by = "date"

## Joining, by = "date"

starting\_model\_forecast <- forecast(starting\_model, h = 65)

starting\_model\_forecast <- to\_tibble(starting\_model\_forecast)

## Joining, by = "date"

## Joining, by = "date"

and plot the forecast to see how it looks:

avia\_clean\_monthly %>%

group\_by(date) %>%

summarise(total = sum(passengers)) %>%

mutate(total\_ihs = ihs(total)) %>%

ggplot() +

ggtitle("Minimization of RMSE") +

geom\_line(aes(y = total\_ihs, x = date), colour = "#82518c") +

scale\_x\_date(date\_breaks = "1 year", date\_labels = "%m-%Y") +

geom\_ribbon(data = best\_model\_rmse\_forecast, aes(x = date, ymin = lower95, ymax = upper95),

fill = "#666018", alpha = 0.2) +

geom\_line(data = best\_model\_rmse\_forecast, aes(x = date, y = point\_estimate),

linetype = 2, colour = "#8e9d98") +

geom\_ribbon(data = starting\_model\_forecast, aes(x = date, ymin = lower95, ymax = upper95),

fill = "#98431e", alpha = 0.2) +

geom\_line(data = starting\_model\_forecast, aes(x = date, y = point\_estimate),

linetype = 2, colour = "#a53031") +

theme\_blog()

The yellowish line and confidence intervals come from minimizing the genetic algorithm, and the  
redish from auto.arima(). Interesting; the point estimate is very precise, but the confidence  
intervals are very wide. Low bias, high variance.

Now, let’s try with another cost function, where I minimize the BIC, similar to the auto.arima() function:

cost\_function\_bic <- function(param, train\_data, validation\_data, forecast\_periods){

order <- param[1:3]

season <- c(param[4:6], 12)

model <- purrr::possibly(arima, otherwise = NULL)(x = train\_data, order = order,

seasonal = season,

method = "ML")

if(is.null(model)){

return(9999999)

} else {

BIC(model)

}

}

Let’s take a look at the cost at the parameter values returned by auto.arima():

cost\_function\_bic(c(1, 0, 2, 2, 1, 0),

train\_data = logged\_train\_data,

validation\_data = logged\_validation\_data,

forecast\_periods = 65)

## [1] -184.6397

Let the genetic algorithm run again:

cl <- makePSOCKcluster(8)

clusterExport(cl, c('logged\_train\_data', 'logged\_validation\_data'))

tic <- Sys.time()

auto\_arima\_bic <- genoud(cost\_function\_bic,

nvars = 6,

data.type.int = TRUE,

starting.values = c(1, 0, 2, 2, 1, 0), # <- from auto.arima

Domains = domains,

cluster = cl,

train\_data = logged\_train\_data,

validation\_data = logged\_validation\_data,

forecast\_periods = length(logged\_validation\_data),

hard.generation.limit = TRUE)

toc\_bic <- Sys.time() - tic

This time, it took 6 minutes, a bit slower than before. Let’s take a look at the solution:

auto\_arima\_bic

## $value

## [1] -201.0656

##

## $par

## [1] 0 1 1 1 0 1

##

## $gradients

## [1] NA NA NA NA NA NA

##

## $generations

## [1] 12

##

## $peakgeneration

## [1] 1

##

## $popsize

## [1] 1000

##

## $operators

## [1] 122 125 125 125 125 126 125 126 0

Let’s train the model at these parameters:

best\_model\_bic <- arima(logged\_train\_data, order = auto\_arima\_bic$par[1:3],

season = list(order = auto\_arima\_bic$par[4:6], period = 12),

method = "ML")

summary(best\_model\_bic)

##

## Call:

## arima(x = logged\_train\_data, order = auto\_arima\_bic$par[1:3], seasonal = list(order = auto\_arima\_bic$par[4:6],

## period = 12), method = "ML")

##

## Coefficients:

## ma1 sar1 sma1

## -0.6225 0.9968 -0.832

## s.e. 0.0835 0.0075 0.187

##

## sigma^2 estimated as 0.004145: log likelihood = 109.64, aic = -211.28

##

## Training set error measures:

## ME RMSE MAE MPE MAPE

## Training set 0.003710982 0.06405303 0.04358164 0.02873561 0.3753513

## MASE ACF1

## Training set 0.4553447 -0.03450603

And let’s plot the results:

best\_model\_bic\_forecast <- forecast::forecast(best\_model\_bic, h = 65)

best\_model\_bic\_forecast <- to\_tibble(best\_model\_bic\_forecast)

## Joining, by = "date"

## Joining, by = "date"

avia\_clean\_monthly %>%

group\_by(date) %>%

summarise(total = sum(passengers)) %>%

mutate(total\_ihs = ihs(total)) %>%

ggplot() +

ggtitle("Minimization of BIC") +

geom\_line(aes(y = total\_ihs, x = date), colour = "#82518c") +

scale\_x\_date(date\_breaks = "1 year", date\_labels = "%m-%Y") +

geom\_ribbon(data = best\_model\_bic\_forecast, aes(x = date, ymin = lower95, ymax = upper95),

fill = "#5160a0", alpha = 0.2) +

geom\_line(data = best\_model\_bic\_forecast, aes(x = date, y = point\_estimate),

linetype = 2, colour = "#208480") +

geom\_ribbon(data = starting\_model\_forecast, aes(x = date, ymin = lower95, ymax = upper95),

fill = "#98431e", alpha = 0.2) +

geom\_line(data = starting\_model\_forecast, aes(x = date, y = point\_estimate),

linetype = 2, colour = "#a53031") +

theme\_blog()

The solutions are very close, both in terms of point estimates and confidence intervals. Bias  
increased, but variance lowered… This gives me an idea! What if I minimize the RMSE, while  
keeping the number of parameters low, as a kind of regularization? This is somewhat what minimising  
BIC does, but let’s try to do it a more “naive” approach:

cost\_function\_rmse\_low\_k <- function(param, train\_data, validation\_data, forecast\_periods, max.order){

order <- param[1:3]

season <- c(param[4:6], 12)

if(param[1] + param[3] + param[4] + param[6] > max.order){

return(9999999)

} else {

model <- purrr::possibly(arima, otherwise = NULL)(x = train\_data,

order = order,

seasonal = season,

method = "ML")

}

if(is.null(model)){

return(9999999)

} else {

forecast\_model <- forecast::forecast(model, h = forecast\_periods)

point\_forecast <- forecast\_model$mean

sqrt(mean(point\_forecast - validation\_data) \*\* 2)

}

}

This is also similar to what auto.arima() does; by default, the max.order argument in auto.arima()  
is set to 5, and is the sum of p + q + P + Q. So I’ll try something similar.

Let’s take a look at the cost at the parameter values returned by auto.arima():

cost\_function\_rmse\_low\_k(c(1, 0, 2, 2, 1, 0),

train\_data = logged\_train\_data,

validation\_data = logged\_validation\_data,

forecast\_periods = 65,

max.order = 5)

## [1] 0.1731473

Let’s see what will happen:

cl <- makePSOCKcluster(8)

clusterExport(cl, c('logged\_train\_data', 'logged\_validation\_data'))

tic <- Sys.time()

auto\_arima\_rmse\_low\_k <- genoud(cost\_function\_rmse\_low\_k,

nvars = 6,

data.type.int = TRUE,

starting.values = c(1, 0, 2, 2, 1, 0), # <- from auto.arima

max.order = 5,

Domains = domains,

cluster = cl,

train\_data = logged\_train\_data,

validation\_data = logged\_validation\_data,

forecast\_periods = length(logged\_validation\_data),

hard.generation.limit = TRUE)

toc\_rmse\_low\_k <- Sys.time() - tic

It took 1 minute to train this one, quite fast! Let’s take a look:

auto\_arima\_rmse\_low\_k

## $value

## [1] 0.002503478

##

## $par

## [1] 1 2 0 3 1 0

##

## $gradients

## [1] NA NA NA NA NA NA

##

## $generations

## [1] 11

##

## $peakgeneration

## [1] 1

##

## $popsize

## [1] 1000

##

## $operators

## [1] 122 125 125 125 125 126 125 126 0

And let’s plot it:

best\_model\_rmse\_low\_k <- arima(logged\_train\_data, order = auto\_arima\_rmse\_low\_k$par[1:3],

season = list(order = auto\_arima\_rmse\_low\_k$par[4:6], period = 12),

method = "ML")

summary(best\_model\_rmse\_low\_k)

##

## Call:

## arima(x = logged\_train\_data, order = auto\_arima\_rmse\_low\_k$par[1:3], seasonal = list(order = auto\_arima\_rmse\_low\_k$par[4:6],

## period = 12), method = "ML")

##

## Coefficients:

## ar1 sar1 sar2 sar3

## -0.6468 -0.7478 -0.5263 -0.1143

## s.e. 0.0846 0.1171 0.1473 0.1446

##

## sigma^2 estimated as 0.01186: log likelihood = 57.88, aic = -105.76

##

## Training set error measures:

## ME RMSE MAE MPE MAPE

## Training set 0.0005953302 0.1006917 0.06165919 0.003720452 0.5291736

## MASE ACF1

## Training set 0.6442205 -0.3706693

best\_model\_rmse\_low\_k\_forecast <- forecast::forecast(best\_model\_rmse\_low\_k, h = 65)

best\_model\_rmse\_low\_k\_forecast <- to\_tibble(best\_model\_rmse\_low\_k\_forecast)

## Joining, by = "date"

## Joining, by = "date"

avia\_clean\_monthly %>%

group\_by(date) %>%

summarise(total = sum(passengers)) %>%

mutate(total\_ihs = ihs(total)) %>%

ggplot() +

ggtitle("Minimization of RMSE + low k") +

geom\_line(aes(y = total\_ihs, x = date), colour = "#82518c") +

scale\_x\_date(date\_breaks = "1 year", date\_labels = "%m-%Y") +

geom\_ribbon(data = best\_model\_rmse\_low\_k\_forecast, aes(x = date, ymin = lower95, ymax = upper95),

fill = "#5160a0", alpha = 0.2) +

geom\_line(data = best\_model\_rmse\_low\_k\_forecast, aes(x = date, y = point\_estimate),

linetype = 2, colour = "#208480") +

geom\_ribbon(data = starting\_model\_forecast, aes(x = date, ymin = lower95, ymax = upper95),

fill = "#98431e", alpha = 0.2) +

geom\_line(data = starting\_model\_forecast, aes(x = date, y = point\_estimate),

linetype = 2, colour = "#a53031") +

theme\_blog()

Looks like this was not the right strategy. There might be a better cost function than what I have  
tried, but looks like minimizing the BIC is the way to go.