PERFORMANCE COMPARISON OF ML METHODS FOR PREDICTING THE COVID-19 CASES AND DEATHS

A Master's Thesis
Submitted to the KSCHOOL
by
Oscar Monge Villora

In partial fulfilment
of the requirements for the degree of
MASTER IN DATA SCIENCE

Barcelona, August 2020

Title of the thesis: Performance comparison of ML methods for predicting the Covid-19 cases and deaths

Author: Oscar Monge Villora

Abstract

The goal of this work is to compare the performance of different machine learning methods implemented in a proper way for predicting COVID-19 time series (Cases and Deaths).

In order to make predictions for Covid-19 time series, a program has developed integrating different modules that automatize the different steps of the process: choosing dataset of Covid-19, cleaning and preparing the data; eliminating the heteroscedasticity and trend; fitting and predicting using a machine learning method; tuning the result with Bayesian Optimization based on Time Splitting Cross Validation and obtaining different plots and metrics errors as a result.

The main program uses different approaches taken of the literature as Logistic and Gompertz fitting, SIR Model, etc. Furthermore, the program can be adapted for any temporal series and it allows to configure multiple parameters for the prediction of time series.

Using that tool, we execute some simulations for different approaches to compare the performance of the ML Models and visualize some predictions.

KSCHOOL

Table of contents

List of Figures

1. Introduction

This project is based on comparing the performance of different machine learning models tuned through Bayesian Optimization in order to make accurate predictions on time series of Covid-19 (Cases and Deaths) and as a consequence knowing which model is more appropriate to use for these series. This project has been developed as the thesis of the Master of Data Science of KSchool (Barcelona).

Nowadays, COVID-19 virus is affecting all the countries of the world with different impact levels. Each country is registering some data series generated due to the pandemic as the accumulate cases and deaths. There are other time series registered as the number of hospitalized people, the detecting tests applied or the recovered people, that also can be important to analyze the situation of the pandemic in each region. These temporal series registered can be used, not only for analysis, but also for predicting the impact of the pandemic in the next weeks. Knowing the evolution of the number of infected people is very important to help the government to make the proper actuation to control the pandemic.

This project has been executed to collaborate in this scenario. Compiling different approaches taken on the literature to work with these COVID-19 time series, a configurable program has been designed. Some of the relevant ideas of the literature found are using the Logistic and Gompertz curve to fit the trend of the COVID-19 time series. Some other sources are very focused on SEIR and SIR models, using a system of differential equations that implies the cases, deaths and recovered people. They solved in a deterministic way and sometimes they combine it with machine learning model for making predictions. Other sources of the literature use directly machine learning models on the series, as ARIMA models or use LTSM deep learning method, etc.

The program of the project executes a proper processing of the time series data mixing three important ideas: a proper time splitting for cross-validation, the application of Bayesian Optimization for tuning and the Decomposition of the temporal series. For making predictions the program uses common machine learning methods that can be useful in temporal series as XGBoost, SARIMA, NN, etc.

This research project is based on the idea of designing a confident tool that allows to make user-configurations for obtaining the most accurate and reliable predictions possible using only common machine learning methods. The tool is implemented in order to make easier complex simulations of different approaches. As a result, we will obtain the error for the different ML models and then can be compared in terms of performance for every COVID-19 temporal series imported. Furthermore, the project has in account applying some feature engineering to improve the COVID-19 predictions.

We talk about COVID-19 temporal series in a generalized way because the project doesn't focus in studying the situation of COVID-19 in any region in specific. In fact, we use datasets of cases and deaths for the communities of Spain, the entirely Spain, the world countries and all the world, i.e., not focusing only in one region.

Summarizing, the goals of the project are:

- Design a tool that implements a proper Time Splitting, a Bayesian Optimization tuning and Decomposition methods for making good predictions of COVID-19 temporal series.

- Compare the error of different common machine learning methods used on temporal series and simulate them for different configuration of the system.
- Use some techniques of the literature and implement them in the project to improve the COVID-19 predictions.

The strategy of execution of the project hasn’t had relevant timeline guidelines but rather some freedom in order to design and self-orient the project through the literature and the master knowledge learned but also having in account the goals.

2. State of the art

2.1 Review of Time Series aspects in the literature

This section will cover some specific aspects of Times Series in the literature, required in the development of our project.

In order to apply a proper implementation of a machine learning model on a Time Series we need to have in account the tuning of the model. Then we will get the optimal parameters that give us the best predictions results.

There are different methods in the literature to get the optimal parameters as Grid Search, Random Search, Bayesian Optimization, etc. In this section we will study the last one because is very efficient when we need to find hyperparameters that have a continuous range, for example, a learning rate in a XGBoost. These methods are intrinsically based on cross-validation of the data, therefore, we need to learn how to cross-validate time series respecting the chronological and sequential order, so, it's not a good option to carry out the typical cross-validation shuffling the samples. In the next subsections these topics are explained in more detail.

A) Cross-Validation strategies for Times Series Forecasting

The goal of Cross-Validation (CV) is to estimate an unbiased generalized performance. The proper application of Cross-Validation requires a proper splitting of the dataset. First of all, we split the dataset into a subset called training set and another called test set. If any parameters need to be tuned, we split the training set into a training subset and a validation set. The model is trained on the training subset and the parameters that minimize the error on the validation subset are chosen. Finally, the model is trained on the full training set using the chosen parameters, and the error on the test is recorded.

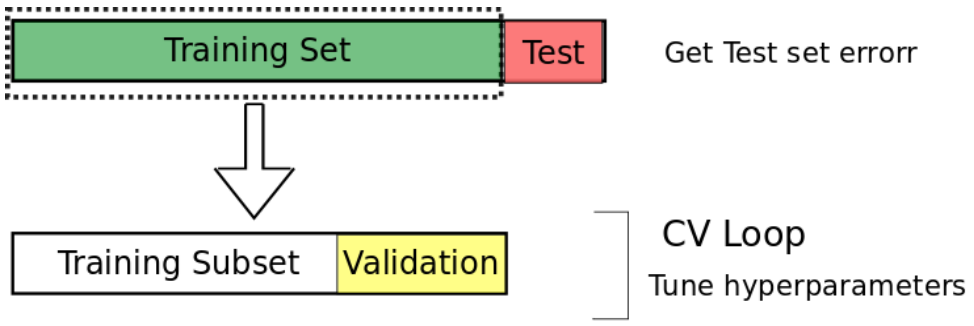


Figure 1. Training /Test Division (Hold out CV)

Using the test set for model selection and estimation is not a good option, because it tends to overfit the test predictions.

Applying a proper cross-validation allows us to find the optimal hyperparameters for the model. In order to find them we define a Loss Function. This Loss Function is minimized using one of these algorithms: Grid Search, Random Search, Bayes Optimization, etc. Some Loss Functions that are commonly used in the literature are the RMSE, R^2 and others as L1-norm, L2-norm or its combination Elastic Net Regression.

We can distinguish some known Cross-Validation methods: the hold-out CV and the Nested CV. The first one consists in using only one subset of training and one subset of validation, as it has been shown in the previous picture. It's the easier method but it hasn't a very good performance because we only execute one training iteration.

The Nested CV or two-round CV divide the train set in various subsets of train and validation instead of one [1]. A part it changes in different iteration the division of the dataset between training and test part. Therefore, it has an inner loop CV in an outer loop CV. The inner loop is responsible for model selection /hyperparameter tuning (validation subset) while the outer loop is for error estimation (test set). We can see how is executed in the next picture:

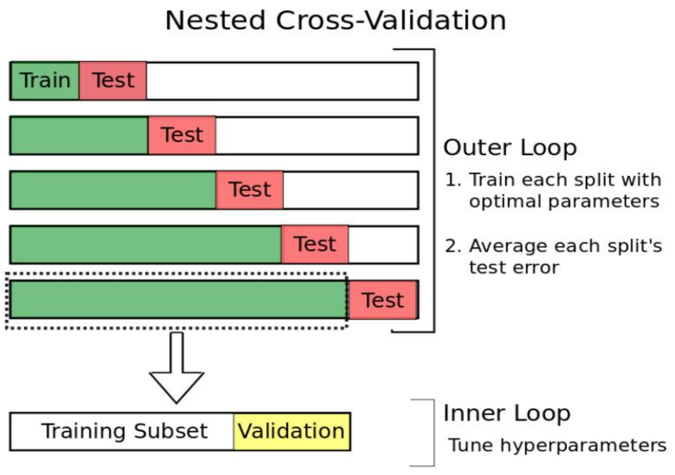


Figure 2. Nested Cross-Validation Strategy

This last method has more performance than the other one and compute a robust estimate of model error.

However, in this project two different strategies from the previous one have been followed for times series forecasting. These strategies have a good performance and don't consider the outer loop fixing the interval of train and test. Doing that we simplify the construction of the algorithm and we get a similar robust estimate of the error comparing to the Nested CV method. They are called Time Series Split and Blocking Times Series Split [2]. These two methods respect the chronologic and sequential order of the time series, a very important consideration.

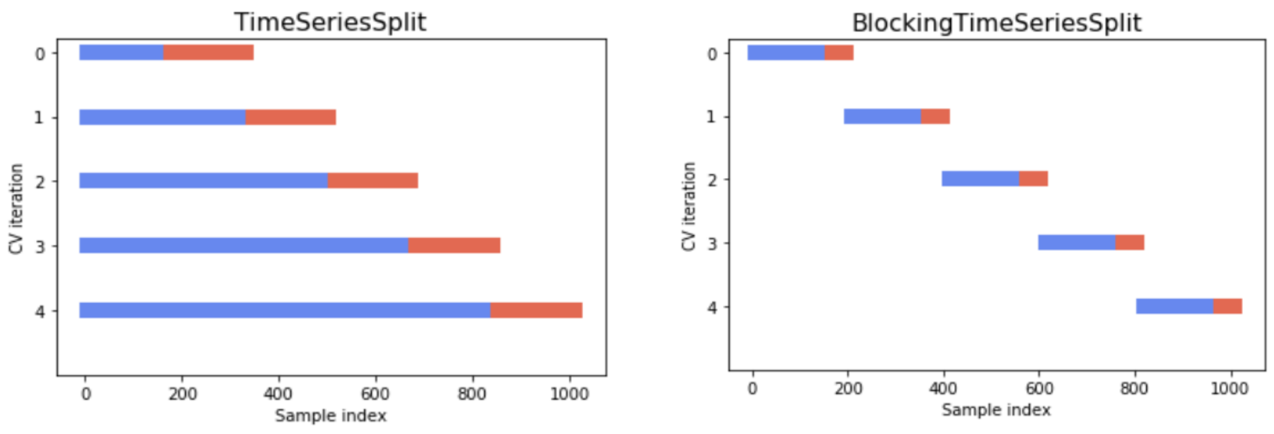


Figure 3. Time Series Splitting Strategies

In the TimeSerieSplit we divide the training set into two folds at each iteration on condition that the validation set is always ahead of the split. However, this may introduce some leakage from future data to the model. The model will observe future patterns to forecast and try to memorize them.

The BlockingTimeSeriesSplit solve this problem. It works by adding margins at two positions:

- First, between the training and validation folds in order to prevent the model from observing values which are used twice, once as a regressor and another as a response. This can be regulated with a new parameter called overlap.
- Second, between the folds used at each iteration to prevent the model from memorizing patterns from an iteration to the next.

B) Bayesian Optimization

Bayesian Optimization is used in different areas as deep learning, sensor networks and a wide range of problems. It is a tool for optimization (minimization) different problems using Gaussian processes. The idea of this method is based on the next explanation:

We have a function called the objective function that we want to minimize choosing the optimal hyperparameters x*.

$$x* = \underset{x \in X}{argmin f(x)} \qquad (1)$$

f(x) is an objective score and x* is the set of hyperparameters that yields the lowest value of the score.

Applied to machine learning field, the f(x) is a score as the RMSE or error rate calculated on the predictions at the validation set for each set of hyperparameters values given. To calculate this score, we apply a cross-validation adapted for times series, calculating the score for each validation subset and averaging the result.

Bayesian approaches in contrast to random or grid search, keep track of past evaluations results which they use to form a probabilistic model mapping hyperparameters to a probability of a score on the objective function, called the surrogate model:

$$P(score|hyperparameters) = P(y|x) \qquad (2)$$

The Bayesian Optimization method is implemented with a sequential process called *Sequential Model-Based Optimization (SMBO)*: First of all, it builds a surrogate probability model of the objective function. Then, it finds the hyperparameters that perform best on the surrogate. After that, it applies these hyperparameters on the true objective function f(x). After evaluating the score, it updates the surrogate model incorporating the new results. We repeat these steps until the number of max iterations is reached.

The aim of Bayesian reasoning is to become “less wrong” with more data. Spend a little more time selecting the next hyperparameters in order to make fewer calls to the objective function. Therefore, we evaluate more promising hyperparameters and we need fewer iterations to find better results than Random Search for example.

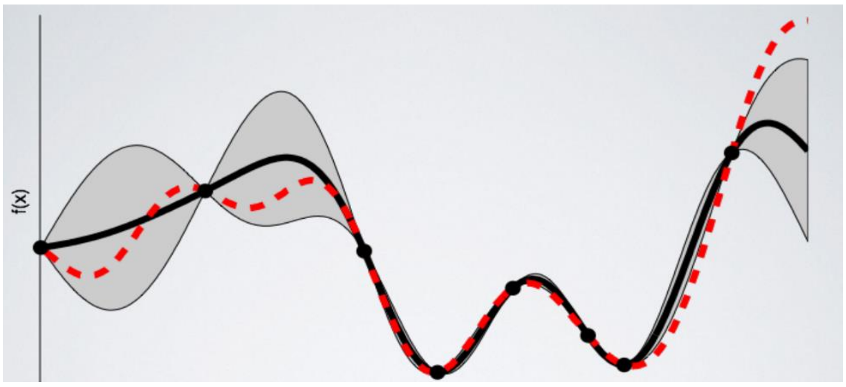


Figure 4. Shows the surrogate function after 8 evaluations

The Bayesian Optimization method has the next elements [3]: A **domain of hyperparameters** over which to search; an **objective function** which takes in hyperparameters and outputs a score that we want to minimize; the **surrogate model** of the objective function; a criteria, called a **selection function** or **acquisition function**, for evaluating which hyperparameters to choose next from the surrogate model; and a **history** consisting of (score, hyperparameters) pairs used by the algorithm to update the surrogate model.

Several common choices for the surrogate model are Gaussian Processes, Random Forest Regressions and Tree Parzen Estimators (TPE). For the acquistion function the most common is the Expected Improvement. This is defined in the next way:

$$\mathrm{EI}_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)p(y|x)dy \tag{3}$$

There are some external libraries in the literature that implement the Bayesian Optimization method as the skopt library and GpyOpt library. In this project, the second one has been chosen. The reason is that this module need to configure and personalize the objective function that is minimized, and this is good for implementing our proper time splitting method for time series. In the code, we import the BayesOptimization function of GPyOpt.methods.

2.2 Review of COVID-19 forecasting in the literature

In this section, we are going to talk about some knowledge and techniques of COVID-19 forecasting of the literature.

Growth in epidemics: basic knowledge

The first stage of an epidemics is an exponential growth [4]. Visualizing the accumulate cases curve, every day the number of cases is obtained multiplying the number of cases of the previous day by some constant. For example, in China, the COVID-19 growth has a constant of 1.15 -1.25.

In an epidemic the new cases are caused by the existent cases. We can conclude intuitively that:

(4)

$$\Delta N_d = E \cdot p \cdot N_d$$

Where Nd is the number of cases on a given day, E is the average number of people someone infected is exposed to each day and p is the probability of each exposure becoming an infection. When Nd increases the growth rate increases too: the increment of cases is proportional to the existent cases.

If we factorize the equation we obtain that:

$$N_{d+1} = (1 + E \cdot p) \cdot N_d \tag{5}$$

Where (1 + E · p) acts as a constant, as we mentioned at the beginning of the subsection. If we want to calculate how many cases will be in “d” days, we can apply the previous formula in a recursive way obtaining:

$$N_{d+1} = (1 + E \cdot p)^d \cdot N_0 \tag{6}$$

This equation describes an infinite exponential curve that never stops and continues growing. However, there can’t be more cases than human exist, so, there is always a limit to grow. As a consequence we expect a moment the curve begins to decelerate. The only way to allow this deceleration on the formula is reducing on time the parameters E or p.

The probability of an exposure becoming a new infection (p), must have in account a factor referring to the probability that a person that is exposed to the virus is already infected. This can be modelized as:

$$p = (1 - \frac{N}{pop.size}) \tag{7}$$

Where the pop.size is the population size of all the community. If we include this probability in the first equation of the subsection we arrive to the next differential equation:

$$\frac{dN}{dt} = c \cdot (1 - \frac{N}{pop.size}) \cdot N \tag{8}$$

The solution to this differential equation is a logistic curve. At the beginning, it has the same behaviour than an exponential curve but when it reaches near the population size it becomes flatten. This curve has a point at the middle called inflection point. The slope of the logistic curve starts to decrease after this point:

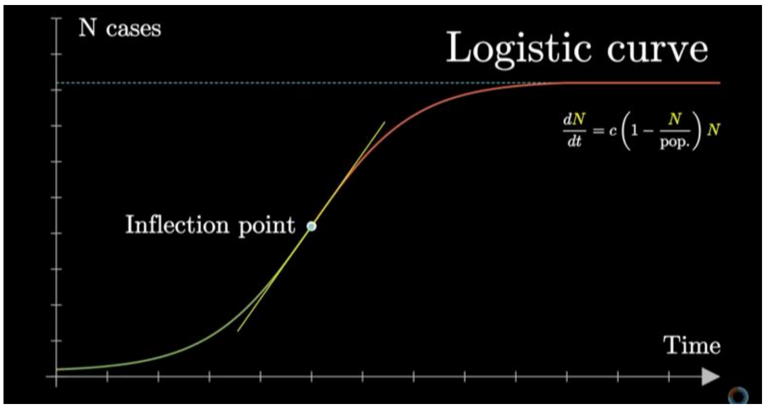


Figure 5. Inflection point in the Logistic curve.

In fact, the inflection point is very important because is the point where the growth factor is equal to one, i.e, when the increment of number of cases is equal to the previous day. The growth factor is defined as:

$$GrowthFactor = \frac{\Delta N_d}{\Delta N_{d-1}} \quad (9)$$

If GF > 1, we are in the logistic part; if GF <1 we are still in the exponential part. When GF = 1 you are at the inflection point and you can conclude the total number of cases will max out around 2 times the number of cases of this point.

There are more causes to shrink the incremental Nd. We put our attention in the parameters E and p of the first equation of the subsection. The average number of people someone infected is exposed (E) can be reduced if the population is limited in terms of mobility, travelling, confinment, reducing the number of people in social groups, etc. The probability p, of each exposure becoming an infection can be reduced if some measures are applied as using masks, washing the hands, using disinfectant, etc.

There is another aspect to have in account for pandemics. World people is clustered in groups (countries or communities). Therefore, taking the population size of the world as the limitant factor of p is a simplification of the problem. It needs simulations of the interaction and exposure of people in all the clusters. However, it has been demostrated that the behaviour considering a free travelling between clusters is similar to consider all the population.

Trend Fitting Models

We define three models commonly used by the community of scientists to fit the curve of COVID-19 cases: The Exponential model, the Logistic model and Gompertz model [4]:

- The **Exponential model** is defined by the next equation:

$$y(x) = N_0(1+p)^{(x-x_0)} \quad (10)$$

  where “N0” is the initial number of cases. The variable p is what we call in the previous section “E · p” and represent the daily percentage increase (if x counts in days). X0 represents a shifting constant.

- The **Logistic model** is defined by the next equation:

$$y(x) = \frac{c}{1+e^{-(x-b)/a}} \quad (11)$$

  It has been explained in the previous subsection. The parameter “c” is the limiting value of the population, “x” is the number of days, and “a” and “b” are parameters that controls the transition between the growth of new cases and the slowing of the growth.

- The **Gompertz model**, defined by:

$$y(x) = c \cdot e^{-b \cdot e^{-x/a}} \quad (12)$$

It's the most suggested model by the scientific community for fitting COVID-19 time series. It is a sigmoid function: The curve has an early, almost exponential growth rate followed by slower growth rate which reaches a plateau. Therefore, it is a not symmetric function, as a difference with the logistic function.

The "a" and "b" parameters have the same function as in the logistic curve.

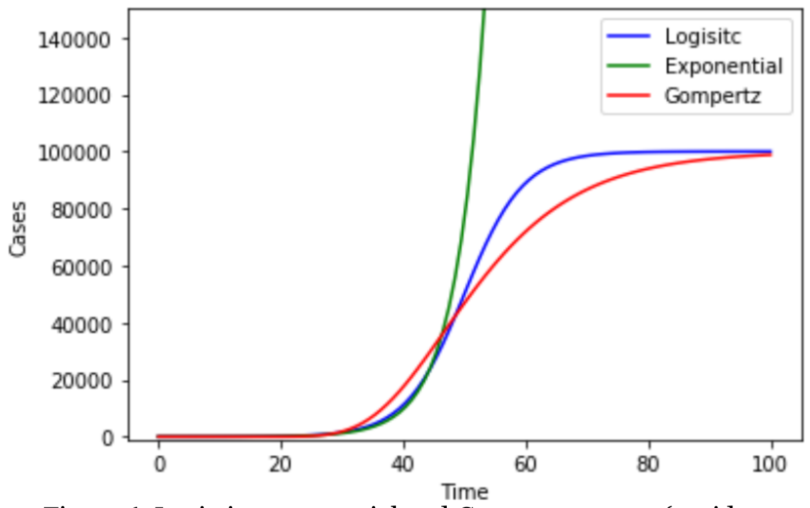


Figure 6. Logistic, exponential and Gompertz curves (covid-19 cases)

We can define in code these models as functions and fit to the data using the function of the Scipy module: *scipy.optimize.curve_fit.* In order to test the fitting of this functions it is recommended to use the mean squared logarithmic error (MSLE), a proper metric to use when targets have an exponential growth, as epidemics or population growth. Other metric commonly used is the R2-score or the coefficient of determination, used for testing the goodness of fit of a model.

SIR model and Effective Reproduction Number R

One of the most used models for predicting on COVID-19 time series is the SIR model. It models the infectious disease dynamics [6] [7]. The strategy and the perspective are different: instead of fitting the trend curve of the cases, the SIR model solves a system of differential equations for the susceptible-infected-removed (SIR) people.

In this system there are four states: susceptible (S), exposed (E), infected (I) and removed (R). The people that belong to R state can be recovered or died. For simplifying the problem, we make the assumption that who have recovered for the virus have acquired immunity for long term.

The flow across the states follow the paths: $S \to E \to I \to R$

The system of differential equations that the model defines is the next one:

$$\begin{aligned} \dot{s}(t) &= -\beta(t)s(t)i(t) \\ \dot{e}(t) &= \beta(t)s(t)i(t) - \sigma e(t) \\ \dot{i}(t) &= \sigma e(t) - \gamma i(t) \end{aligned} \tag{13}$$

Where β(t) is the transmission rate (rate at which people is exposed to infected people), σ(t) is the infection rate (rate at which people exposed becomes infected), and γ(t) is the removed rate (rate at which people is recovered or died).

The system variables “s”, “e”, and “i” are partitions of all the population. Therefore, we can obtain “r” as the number of removed fraction of the population calculating:

$$r = 1 - s - e - i \tag{14}$$

To calculate the cumulative cases of the epidemics in a specific day we only need to sum the infected fraction of population with the removed fraction.

The infection rate and removed rate are considered fixed in this COVID-19 model. Their values adopted for the scientific community are 1/5.2 (5.2 period of incubation) and 1/18 (18 days of average illness duration) respectively.

The transmission rate is modeled as:

$$\beta(t) = R(t)\gamma \tag{15}$$

Where R(t) is the effective reproduction number at time t. This parameter is very important to model the epidemics. R(t) is the number of people that can be infected by an individual at any specific time. It can change through time due to different scenarios of the epidemics:

- Population becomes increasingly immunized, either through individuals gaining immunity after being infected or through vaccination and also as people die.
- Imposed social distancing though confinements or other restrictive measures.

Therefore, making good assumptions for R(t) give us a more robust model. In this context, we can use the SIR models in different scenarios: considering **R(t) is constant**: R0; **changing mitigation**, adoption of stricter measures for social distancing; and **Ending Lockdown**, confining the population. In each scenario R(t) is modelized with a different time function.

To solve the system equations with python a good solution is using the odeint function of the library scipy.integrate. In our project, we will try to improve our predictions using the SIR Model to fit the trend of the COVID-19 time series.

3. Project development

3.1) Structure of the project

The project is divided in 6 modules: Datasets, Bayes Optimization, Decompose, Models ML, Utils and Plots and Error. The input of the system are the datasets of time series of COVID-19 (Datasets module) and the *config_generator.py* file that allows to choose different option configurations. The output are the day predictions in the COVID-19 temporal series returned through plots and errors calculated using the Plots and Errors module.

The project runs since *__main__.py* calling the module main_project.py, that contains the main instructions of the project. Basically, it imports the configuration parameters, prepares the data of the time series and run the Bayes Optimization module to obtain an accurate prediction of the series. After that, it makes the instructions to plot and evaluate it in terms of error.

Therefore, the *main_project.py* obtain one result for a determined configuration. Executing our project, we call this module a unique time. In order to do simulations obtaining various results for dynamic changes of the configured parameters we only need to create a new script or notebook and import the *main_project.py* module to run it various times for different configurations.

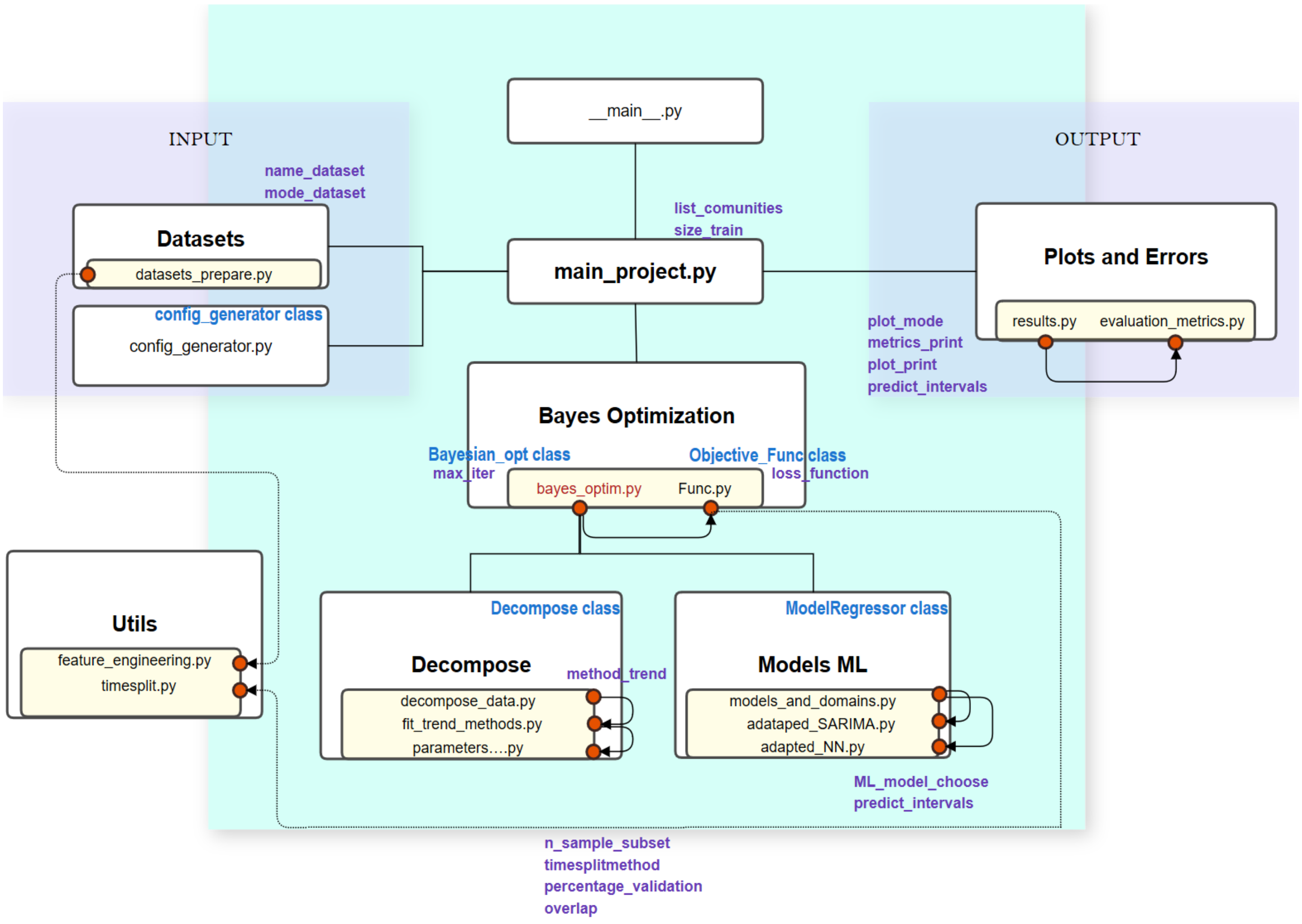


Figure 7. Schematic representation of the modules of the program and interactions

The ***Bayes Optimization module***, specifically, the *bayes_optim.py* is one of the most important modules of the project because it connects the major part of the modules and its respective classes.

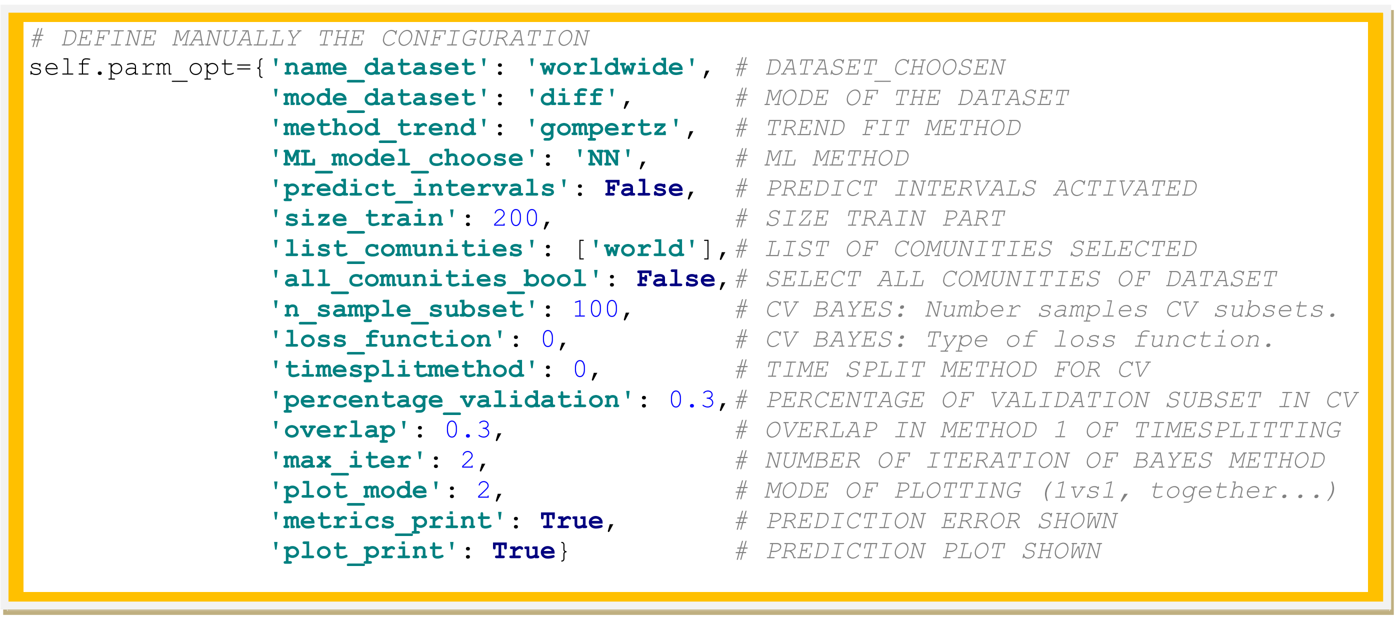
It contains the Bayesian_opt class that define the objects of the class Decompose (**Decompose module**), ModelRegressor (**Models ML module**) and Objective_Func (*Func.py*) in its own methods, and calls the external library GpyOpt.BayesOptimization to execute the search of the optimal parameters minimizing the objective function (declared in the Objective_Func class). Also, it defines in its methods, the functions to fit the optimal ModelRegressor and making the optimal prediction.

The other modules of the project mentioned before are the **Decompose module**, that decompose the temporal series subtracting its heteroscedasticity and trend components (with different fitting approaches); the **Models ML module**, that choses and applies a machine learning method on the temporal series without trend for making predictions on test; and finally the **Utils module**, that incorporate the time splitting function to generate the temporal cross-validation on the objective function and the engineering feature function that stacks interesting features on the vector of features that enters as input in the ML model.

When running the project, the *main_project.py* declares a Bayesian_opt object with the configuration parameters given by the user and execute, with a proper order, the instructions of the different methods of the Bayesian class: run a number of iterations, fit the best ML model and predict.

Configuration file (*config_generator.py*)

The different configuration parameters can be configured manually in the config_generator.py and are shown in the next picture:



```
# DEFINE MANUALLY THE CONFIGURATION
self.parm_opt={'name_dataset': 'worldwide', # DATASET_CHOOSEN
               'mode_dataset': 'diff',       # MODE OF THE DATASET
               'method_trend': 'gompertz',   # TREND FIT METHOD
               'ML_model_choose': 'NN',      # ML METHOD
               'predict_intervals': False,   # PREDICT INTERVALS ACTIVATED
               'size_train': 200,            # SIZE TRAIN PART
               'list_comunities': ['world'],# LIST OF COMUNITIES SELECTED
               'all_comunities_bool': False,# SELECT ALL COMUNITIES OF DATASET
               'n_sample_subset': 100,       # CV BAYES: Number samples CV subsets.
               'loss_function': 0,           # CV BAYES: Type of loss function.
               'timesplitmethod': 0,         # TIME SPLIT METHOD FOR CV
               'percentage_validation': 0.3,# PERCENTAGE OF VALIDATION SUBSET IN CV
               'overlap': 0.3,               # OVERLAP IN METHOD 1 OF TIMESPLITTING
               'max_iter': 2,                # NUMBER OF ITERATION OF BAYES METHOD
               'plot_mode': 2,               # MODE OF PLOTTING (1vs1, together...)
               'metrics_print': True,        # PREDICTION ERROR SHOWN
               'plot_print': True}           # PREDICTION PLOT SHOWN
```

Also, these parameters can be configured manually through the file config.ini of the project. In fact, when we put the values manually in the *config_generator.py* and run the project, an object of the class config_generator is given in the *main_project.py* as an input. Then it replaces the values of config.ini that after that, are read in the same module *main_project.py*.

```
def main_project(config_object = config_generator()):
   list_comunities = config_object.config_replace_ini(activate = True)#Replace config
   conf = config_function()    # Read the values of config.ini
    ...
```

In order to make simulations and running various sets of configurations, the config_generator file has two functions that make easier this task: the 'import_options_template' function and the 'modify_options'. Calling the first one we can put an initial template of configuration for running the simulations. Using the second function, we can indicate only which are the modifications of configuration that have to be applied respect the template to run again the project.

Main file (*main_project.py*)

The main file *main_project.py* to run the instructions of the project contains 4 different functions. One of them is the most important, that is the *main_project()* function that is mentioned in the previous picture. It runs all the important instructions of the project.

Another function is the *config_function()* that reads the configuration values of the config.ini and keep them in a dictionary to use them. The third function is the *check_programmed_ways(),* that verifies the configuration parameters set are compatible and adjust other internal parameters that depends on the configuration imposed. Finally, the last function is the *data_prepare().* This function handle import aspects: selects the community or region of the total dataset imported, applies the train/test division and makes the feature engineering task: the general engineering feature applied in all datasets (adding the seasonal feature) and more specific engineering feature for concrete datasets which is provided by the Utils module.

The *main_project()* function instructions are implemented in the next logical sequence:

- First, we read the configuration file with the *config_function()* and check and adjust the correct parameters with *check_programmed_ways().* Then, we import the specified dataset of the Datasets module and prepare the data frame with the *data_prepare()* function.
- After that, we make a loop for the different communities or regions selected to treat their temporal series. We declare the Bayesian_opt class with the configuration parameters imported and make the fitting and prediction with the optimal parameters for each community saving the results on a dictionary called *store_data*.
- Finally, we plot the results and the error using the plot mode specified in the configuration for all the communities

3.2) Modules of the project

3.2.1 DATASET MODULE

CODE (datasets_prepare.py)

This is the module responsible of reading any dataset of COVID-19 that has been download and saved on Datasets folder. It obtains and prepares the data as a data frame in a predetermined and common form: using the rows for the different Communities or Regions and the columns for the cases or death temporal series. In case we have only one region as considering the full World, we only have one row.

The dataset module selects the valid dates will become an input of the system (manual way).

Some of the datasets download are time series of the cumulative cases and deaths of COVID-19 virus and others contains the daily incremental cases and deaths. Therefore, we can have datasets in cumulative or incremental (differential) form.

It is possible to work with every dataset download in the mode we want. The module contains a function **diff_cum_conv()** that makes the conversion differential-cumulative and vice versa. In fact, it is configurable in the configuration file of the project through the parameter 'mode_dataset', that can be 'diff' or 'cum'.

We have different datasets ready to be used, everyone has been named, therefore we can select the dataset chosen through the configuration parameter 'name_dataset':

- **'deaths-old':** It has the time series of COVID-19 accumulated deaths until 20-May for each community of Spain.

  Converting the series to the incremental mode and drawing it for a Spain community, we notice the last point is at the descending part of the curve. Therefore, one of the possible interesting goals of the predictions here would be to know where the incremental deaths reaches zero or a stabilized low number.
- **'deaths-update':** It has the time series of COVID-19 differential deaths until 23-July for each community of Spain. This series has been corrected trying to minimize the effect of the change of counting criteria that was done at finals of May.

  Drawing the series, we realize the curve of the series finishes in a long-stabilized zone of low deaths values nearly zero after the first wave of the COVID-19 pandemic.
- **'casos-update':** It has the time series of COVID-19 differential cases until 19-July for each community of Spain. This series has been corrected trying to minimize the effect of the change of counting criteria that was done at finals of May.

  Drawing the series, we realize the curve describes the first wave of the pandemic, a stable period of low cases after it and finishes at a point the cases are increasing again in more or less quantity depending the Spain community. Therefore, one of the possible interesting goals of the predictions here would be to know if it will be a second wave and which levels will reach the peak of this wave.

- **‘national’:** Cases, hospitalized, deaths and recovers for the entirely Spain until 18-May.
- **‘worldwide’:** Incremental and accumulated cases and deaths of COVID-19 for each country of the World until 10-Aug.

  The data frame obtained from this dataset is prepared adding all the temporal series for all countries. Then, we work with one temporal series for all the world COVID-19 cases and deaths.

  In this dataset, for much countries series there are some days that seems to be outliers. They can be produced by different causes like changing the counting way, recollect some cases of past days, discard some others, etc. For this reason, the idea of adding all countries series obtaining higher values on the final series than compensates these outliers gives more credibility to our data. Furthermore, in order to compensate these outliers and the abrupt seasonality change of cases reported between labor days and the weekend, an exponential moving average of 7 days is applied, obtaining a smoother curve.

The datasets ‘**deaths_old’**, ‘**deaths_update’** and ‘**casos_update’** are obtained from the source of Datadista, a communication media that collects the information mainly from the Ministerio de Sanidad and the Institute of Salud Carlos III of Spain.

The dataset ‘**national**’ is obtained from the web of the Centro Nacional de Epidemiología of Spain while the dataset ‘**worldwide**’ is obtained from the World Health Organization (WHO).

3.2.2 DECOMPOSE MODULE

CODE (decompose_data.py)

This module allows to decompose the temporal series in heteroscedasticity and trend components. It subtracts the trend component to the temporal series becoming our input to the machine learning model that will predict for the next days. Apart, the module can recover again the results adding the tendency and heteroscedasticity.

The seasonality component is not considered due to the COVID series don't manifest a clearly and repeatable season component. However, we can appreciate a low signal of seasonality for each week, because the data recompiled about cases and deaths for COVID-19 are notified in a non-homogenous way due to the weekend. For this reason, we apply some feature engineering adding a seasonal feature that distinguish every week (this is done in another module, main_project.py).

To obtain the trend component we fit an adjusted curve to the temporal series. This is executed through the methods of the trend_method object initiated at the **fit_trend_apply** function. This object belongs to a class (coded in other file: fit_trend_methods.py) that defines different strategies or fitting functions to adjust to the curve: polynomic, logistic and gompertz are the most important among others. We can choose what fitting method will use in our simulations specifying the **trend_method_name** in the configuration file.

```
class Decompose(object):
    ...

    def fit_trend_apply(self):
        self.model_fit_trend = trend_method(self.x_train_features,
                                            self.data_train,
                                            self.trend_method_name,
                                            self.parameters_trend,
                                            self.mode_dataset)
        self.model_fit_trend.fit()
        trend_train = self.model_fit_trend.predict(self.x_train_features)
        trend_train = pd.DataFrame(trend_train)
        data_without_trend = self.data_train - trend_train.values
        return data_without_trend

    ...
```

This module has been implemented because some machine learning models that we use to make predictions on the COVID-19 temporal series are not good enough to capture the trend of the series as the Random Forest and the XGBoost. SARIMA, another method used on the project needs the input series to be seasonal stationary, i.e., without trend, so it justifies more its use.

CODE (fit_trend_methods.py.)

The fitting functions are defined in the class trend_method in *fit_trend_methods.py.* There are several aspects to have in account in these strategies.

1) One of them is the type of dataset. The dataset as, we have explained, can be in incremental (differential) mode or cumulative mode. There are some adjusted fitting models that works better using one form of the dataset than the other. For example, the logistic and gompertz fitting models

are good (as literature confirms) to adjust to cumulative curves of cases and deaths of epidemic series. Furthermore, applying a log(function) to work on them is not a good option because deforms the temporal curve and then they can't be adjusted with these methods.

For these reasons, when we enter a COVID-19 dataset in differential mode and we apply logistic or gompertz methods, a conversion to cumulative mode is applied before the fitting. Then we predict and convert it again to differential mode (to obtain the incremental trend).

```
def fit(self):
  ...

  if self.method == 'gompertz':
    if self.mode_dataset == 'diff':
        self.data_train = diff_cum_conv(type='cum', data=self.data_train, axis=0)

    x0 = self.parameters_trend
    self.gompertz_model = lambda x,a,b,c: c * np.exp(-b*np.exp(-(x - x0) / a))

    def gompertz_model(x, a, b, c):
        return c * np.exp(-b * np.exp(-(x - x0) / a))

    fit_model = curve_fit(gompertz_model,
                          xdata=self.x_train_features[:, 0].squeeze(),
                          ydata=self.data_train.squeeze(), maxfev=100000,
                          bounds=([0, 0, 0], [10, 100, 150000]))
    self.parameters_optim = fit_model[0]

def predict(self, x_features):
  ...

  if self.method == 'gompertz':
    a, b, c = self.parameters_optim
    self.trend = self.gompertz_model(x_features[:, 0].squeeze(), a, b, c)

    # We convert it again into the trend in differential version
    if self.mode_dataset == 'diff':
        self.trend = diff_cum_conv(type='diff', data=self.trend, axis=0)

  return self.trend
```

The polynomic fitting method works better for datasets with incremental mode. Applying the log(series) we get a curve that can be easier to fit with a polynomic curve. In fact, this method isn't allowed to work with the cumulative mode of the dataset (despite we use conversion to incremental mode before fitting).

The reason is that it would create problems during the conversion due to the structure and packaging of the different modules of the project: the problem is that the conversion incremental-cumulative and vice versa and the operation of log(series) are not commutative and to adapt the code of the project having this in account requires a high restructuration.

2) There are some parameters that are required for the fit trend methods: for the polynomic method, the order of the polynomic curve; for the gompertz and logistic function, the delay x0 that indicates the starting point of the curve. Depending the COVID-19 series we input in our system: statal

(Spain), communities of Spain, Worldwide, etc, the required parameters that works better changes, so, the value for these parameters for the different datasets and fit trend methods are saved in another file called *parameters_adjusted_dataset.py*

3) The methods of gompertz and logistic are adjusted with the function **curve_fit** from the external module scipy.optimize.

3.2.3 BAYES OPTIMIZATION MODULE

CODE (bayes_optim.py):

This file is the main module that connects the different modules to run the Bayes optimization tuning. It calls the modules through the methods defined in its unique class: the class Bayesian_opt.

First of all, it defines the Decompose object to eliminate the tendency through a chosen method that fits the trend and other components of the series. After that, it defines the ModelRegressor object that will carry out the fitting and prediction to the temporal series. Then it defines the Objective_Func Object, i.e., the objective function that will be minimized through the BayesOptimization Object that comes from the external package GpyOpt. Finally, we will declare the run method that will initiate every object explained and will run the optimization with a number of iterations given (max_iter).

```
def descompose_train_data(self):
    self.decompose_model = Decompose(self.x_train_features, self.data_train,
                                     self.log_transform, self.fit_trend_method,
                                     self.parameters_trend, self.mode_dataset)
    self.data_train_log_trend = self.decompose_model.descompose_train_data()

def choose_model(self):
    self.model_cv = ModelRegressor(self.model_choose, self.predict_intervals)
    self.domain = self.model_cv.domains_models()

def func_to_optimize(self):
    self.func = Objective_Func(self.x_train_features, self.data_train,
                               self.data_train_log_trend, self.n_sample_subset,
                               self.timesplitmethod, self.opt_train_size,
                               self.model_cv, self.loss_function,
                               self.decompose_model, self.percentage_validation,
                               self.overlap)
def Bayesdefinition(self):
    self.opt_search = BayesianOptimization(self.func.objective_function,
                                           domain=self.domain,
                                           model_type='GP',
                                           acquisition_type='EI',
                                           num_cores=-1,
                                           verbosity=False)
def run(self, max_iter=100):
    self.descompose_train_data()
    self.choose_model()
    self.func_to_optimize()
    self.Bayesdefinition()

    self.opt_search.run_optimization(max_iter)
    ...
```

This class have the three more functions: **best_model, fit and predict.** The first one defines the ModelRegressor with the optimal parameters of the Bayesian search and the other two functions defines the method to fit the data to the model and get the optimal prediction.

The Objective Function Class is described in another file in the same folder than the Bayes Optimization (*Func.py*). Its main function **objective_function()** receives the hyperparameter vector that the Bayesian Optimization method tries in each iteration on the Regressor Model chosen. Basically, it is constituted by three parts: applying the Time Splitting method to obtain the subsets of train and validation; fit and predict of the Regressor Model with the hyperparameter vector tried in this iteration and calculate the average Loss Function (for all the Cross Validation Folds) in this iteration that is returned as an output and it is going to be minimized.

3.2.4 UTILS MODULE: TIME SPLITTING

The Time Splitting module, as it has been explained in the second section of the work, allows to divide the train part of the data in different subsets of train and validation (different folds) in a cross-validation way, respecting the temporal order of the events. This is made to apply the Bayesian Optimization method calculating the loss function in each fold. The **time_split()** function defined in the timesplit.py module allows choosing one of two methods of Time Splitting: the Time Series Splitting (timesplitmethod == 0) or the Blocking Time Series Splitting (timesplitmethod == 1).

In the configuration file of the project we can configure: the **timesplitmethod**, the Time Splitting method used; the **n_sample_subset**, number of observations in each subset or fold for cross-validating, that determines the number of subsets constructed; the **percentage_validation**, that is the percentage of division of each subset between train and validation; the **overlap** for the second time-splitting method, that is the percentage of overlapping between a subset and the next one. When overlap is equal to zero the second fold starts where the first fold ends (in the Blocking Time Split method).

Furthermore, the code is prepared to profit all the samples, including the remaining samples that grouped are not enough to construct a subset. These are included in the validation subset in the last fold.

```
    n_subsets = int(round(opt_train_size / n_sample_subset))
    n_validation = int(round(n_sample_subset * percentage_validation))
    n_subtrain = n_sample_subset - n_validation
    idx = np.array(range(opt_train_size))
    i = iteration

    if timesplitmethod == 0:
        # Times Series Split

        start = 0
        split = start + (i + 1) * n_subtrain
        stop = split + (i + 1) * n_validation

        idx_subset_train = idx[start: split]
        idx_subset_valid = idx[split: stop]

        if i == (n_subsets - 1):
            idx_subset_valid = idx[split:]

    elif timesplitmethod == 1:
        # Blocking Time Series Split

        start = int(round(i * n_sample_subset * (1-overlap)))
        split = start + n_subtrain
        stop = split + n_validation

        idx_subset_train = idx[start: split]
        idx_subset_valid = idx[split: stop]


        if i == (n_subsets - 1):
            idx_subset_valid = idx[split:]
```

3.2.5 MODELS OF ML module

In this module, we have created the class ModelRegressor, which function *regressor()* defines the different machine learning methods to make predictions for the temporal series of COVID-19 cases or deaths. The method receives as an input variable the list of hyperparameters values that are used for each ML method object.

A part of the definition of the ML model objects, we define the search domain of the parameters of each model that the Bayesian Optimization module uses for finding the optimal ones.

We can choose on the configuration parameter **ML_model_choose** which method we want to use. Furthermore, there are some methods that have the option of using predicting intervals as the XGBoost. This option can be activated through the configuration parameter **predict_intervals.**

The different machine learning methods defined are the next one:

- **XGBoost**, (ML_model_choose = 'xgb'). The hyperparameters used in this model are the n_estimators, max_depth and learning_rate.

  In case the configuration parameter **predict_intervals** is activated, a Gradient Boosting Machine is defined instead of the XGBoost. The reason is because this external module of sklearn has the possibility to predict the intervals for different quantiles [8].

- **Random Forest** (ML_model_choose = 'rf'). The hyperparameters used in this model are the n_estimators and the max_depth.

- **Support Vector Regressor** (ML_model_choose = 'svr') [9]. This method is not very used in our simulations because the predictions obtained are not so good despite we are running the model with its optimal parameters found by the Bayesian Optimization method.

- **SARIMA** (ML_model_choose = 'SARIMA'). We are using SARIMA instead of ARIMA because we have in account the low signal of week seasonality. The hyperparameters used in this model are the (p, d, q) and (P, D, Q) orders.

  This method imports the class of *adapted_SARIMA.py,* another file of the module that configure the fit and predict methods in the same way that the previous three methods: with the same input and outputs. This is done in order to be called and used by the Bayes Optimization module in a generalized way.

- **Neuronal Network** (ML_model_choose = 'NN'). It has the same problem as SARIMA to be called by the Bayes Optimization module. Therefore, it has been adapted in another file *adapted_NN.py*. The hyperparameters used in this model are the train window and the learning rate.

  We are using a specific NN, a Long Short Term Memory Networks (**LSTM**), that are capable of capturing patterns in the time series data. It is implemented using the PyTorch library [10] [11].

3.2.6 PLOTS AND ERRORS MODULE

In this module we have two files: the *evaluation_metrics.py* and the *results.py*. The first one prints and returns the prediction error for a metric chosen, being the most used the 'rmse'. There are other available metrics as the 'mae' or the 'mape' (these can be specified with the input variable list_metrics).

The *results.py* module carries out the plots of the train and test part of the temporal series with the prediction on test. This module receives the data_storage variable with all the train, test and predicted vectors for the list of communities or regions chosen in the configuration parameters as **list_communities**.

In order to show together the different predictions for different communities we have developed three methods or plot modes to visualize them in different perspectives. Furthermore, as *evaluation_metrics.py* can only be applied for one temporal series (i.e., only one region) it is executed since this module in a loop for obtaining all the error metrics results for all the communities.

We can configure which of the three plot modes we want to use with the configuration parameter **plot_mode** . These three methods are the next:

- *evaluate():* This method plots the prediction of the temporal series for each region calculating and printing its metrics error one after one (plot_mode = 0).
- *plot_comunities():* This method prepares a unique plot with an arrange of subfigures, each one for each region, in order to see all the communities at the same time (plot_mode = 1). It calls the metrics_evaluation function with the option (mode = 'Inplot') to include the prediction error in every subfigure. The selection of the number of subfigures and its proper arrangement is prepared in this section of the code:

```
#store_data.keys are the different communities or regions

elem = len(store_data.keys())

matrix_k = int(np.sqrt(elem))
rest = elem - matrix_k * matrix_k
columns = matrix_k
rows = matrix_k
if rest != 0:
    if rest <= matrix_k:
        rows = rows + 1
    elif rest <= matrix_k * 2:
        rows = rows + 2
    elif rest <= matrix_k * 3:
        rows = rows + 3
```

- *global_prediction_plot ():* This method is based on the sum of the temporal series and the test predictions of every region specified in order to have a global prediction plot and error (plot_mode = 2). For example: we can predict for all the communities of Spain and obtain the global prediction for the entirely country.

3.3) Project Timeline development

Our project strategy hasn't considered a very detailed work plan because the project has been focused in a research point of view based on discovering new aspects and useful implementations in the main program to improve our predictions.

However, we can classify the different task events executed for the last four months in Research, Planning and Design, Coding, Simulations and Analysis of the results and see their timeline development.

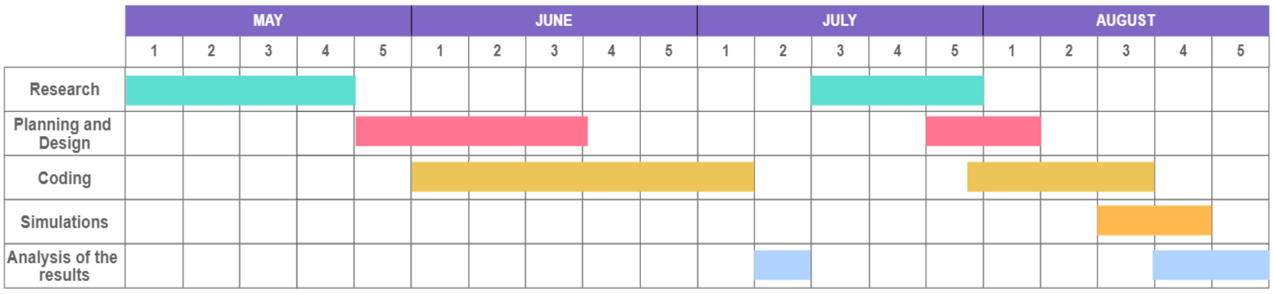


Figure 8. Work Timeline Development of the project.

As the picture shows, it seems there are two periods of Timeline Development: one since May to the beginning of July and the second one since July until finals of August.

In the first period the research was focused on understanding and developing the techniques associated with generalized temporal series as how to implement the Bayesian Optimization for tuning or how to applies a proper time splitting that respects the temporal sequence for cross-validation. After that, we designed the code for applying various known machine learning methods with some COVID-19 times series and started to assemble the main program. We checked the results obtained until the next period of Timeline Development starts, to assure the project was in the good direction.

In the second period, we focused the research on the COVID-19 Forecasting literature. Some aspects of what we learn is how to fit the trend of the time series of COVID-19 cases and which ML are more frequently used for this specific series. Furthermore, we collected information about other strategies of forecasting as SIR models. Then, our program has been adapted to implement these new techniques that improves the predictions. Finally, the last week of August we executed some simulations to compare the different ML models used and analysed the results.

4. Results and simulations

The simulations have been done in the SIMULATIONS_NOTEBOOK (1, 2 and 3). We have imported the configuration script *config_generator.py* and the script *main_project.py*. Using the class of the first one we can configure the parameters of the simulation. Running the class of the main_project.py we run our program with the configuration applied and return us the RMSE error and the prediction results.

4.1 Main explanation of the set of simulations

Structure of the simulations

The idea that is repeated all the time to develop our simulations is to make an initial template to adjust all the parameters one time (function *import_options_template*). Then we decide in the simulation which is going to be our dynamic parameters that will change. After that, we run in loop the project modifying in each iteration this parameter using the function *modify_options*. While the loop is running, we are constructing our dictionary with the results of interest of looping the parameter as could be the RMSE, or the predictions. Finally, we make plots to visualize the dynamic parameters response. If we want to do another simulation similar to the template used, it is not necessary to set again the full template, only we need to change the parameters that are different respect before.

Datasets used for simulations

We have different datasets available that describes the cases and deaths of communities of Spain ('cases_update' and 'deaths_update' respectively) and the cases and deaths for each country of the world and as a consequence the entirely world ('worlwide' dataset).

Using one level of regionality to predict or another have a remarkable difference: considering the worlwide curve we have a curve of cases that is still growing, same as deaths and we can make predictions about how much it is going to increase.

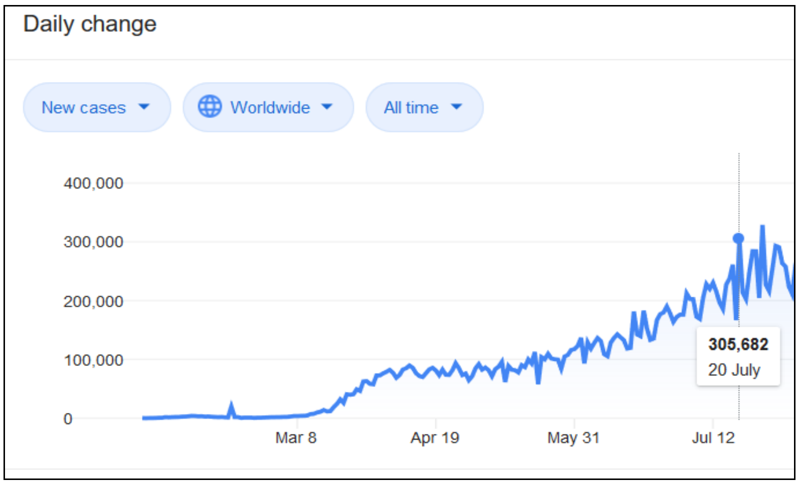


Figure 9. COVID-19 worlwide cases time series shown by Google

However, if we consider a community of Spain curve, we notice there was a wave around April and then it falls to low values of cases and a second wave is appearing at finals of July. This behaviour is repeated in other communities and some countries of the world. If we want to make predictions is very important to consider since which point we are going to predict. As our model hasn't learned about new outbreaks, it can't predict these second waves and the results checked are poor in this point. Therefore, using these temporal series we consider a good point of interest to predict at the descending part of the first curve (as since 19 April). It has less interest than predicting a second wave but is a more reliable goal.

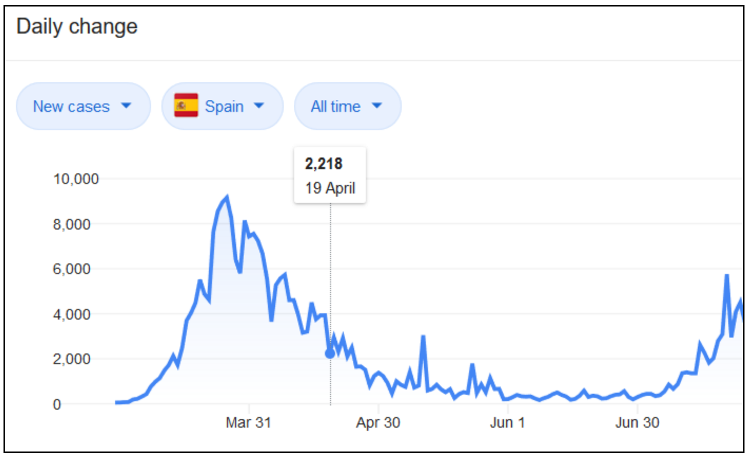


Figure 10. COVID-19 Spanish cases time series shown by Google

The problem to focus on predicting at the descending part of the first wave for this dataset is that we have few points to train the model. Therefore, we don't expect a high signal to make very accurate predictions. We are talking about training a model with 70 samples as much.

Despite these datasets have few data samples, the reality is that the COVID-19 has appeared and need to be analysed and make an effort to predict its behaviour.

Finally, we have decided to make the simulations and predictions for the COVID-19 cases and deaths for the full world community using the 'worldwide' dataset and the COVID-19 deaths for the Cataluña and Madrid communities of the 'deaths_update' dataset. Therefore, using different datasets and different levels of regionality we allow us to contrast and obtain more reliable conclusions.

DIFFERENT SET OF SIMULATIONS

We are going to explain the different approaches taken for execute the simulations and obtain interesting results that answer these questions: **1) Which configuration and model is the best for predictions? 2) How good are the predictions? 3) It is decomposing the Time Series need?**

In order to answer these questions, we are going to compare the RMSE errors obtained for each ML model varying the percentage train/test and we will obtain the predictions at an optimal percentage. Furthermore, we will conclude which ML model is working better and which method for adjusting the trend we have to select.

The set of simulations are divided in three different set that depends on the Time Series used of predicting.

4.2 Set of Simulations

A. Predictions on COVID-19 worldwide cases

We can consider the Time Series in cumulative mode or differential mode to make predictions. Therefore, we are going to execute simulations for both cases.

We have used different models to fit the trend of the Time Series: gompertz, logistic, polynomic, etc. The one that adjust better to the curve is the Gompertz Curve. Therefore, in the next figure we compare two ways of proceeding: fitting the trend of the cumulative time series with the Gompertz method and subtracting it to the Time Series for predicting, or no subtracting the trend and applying the ML models directly.

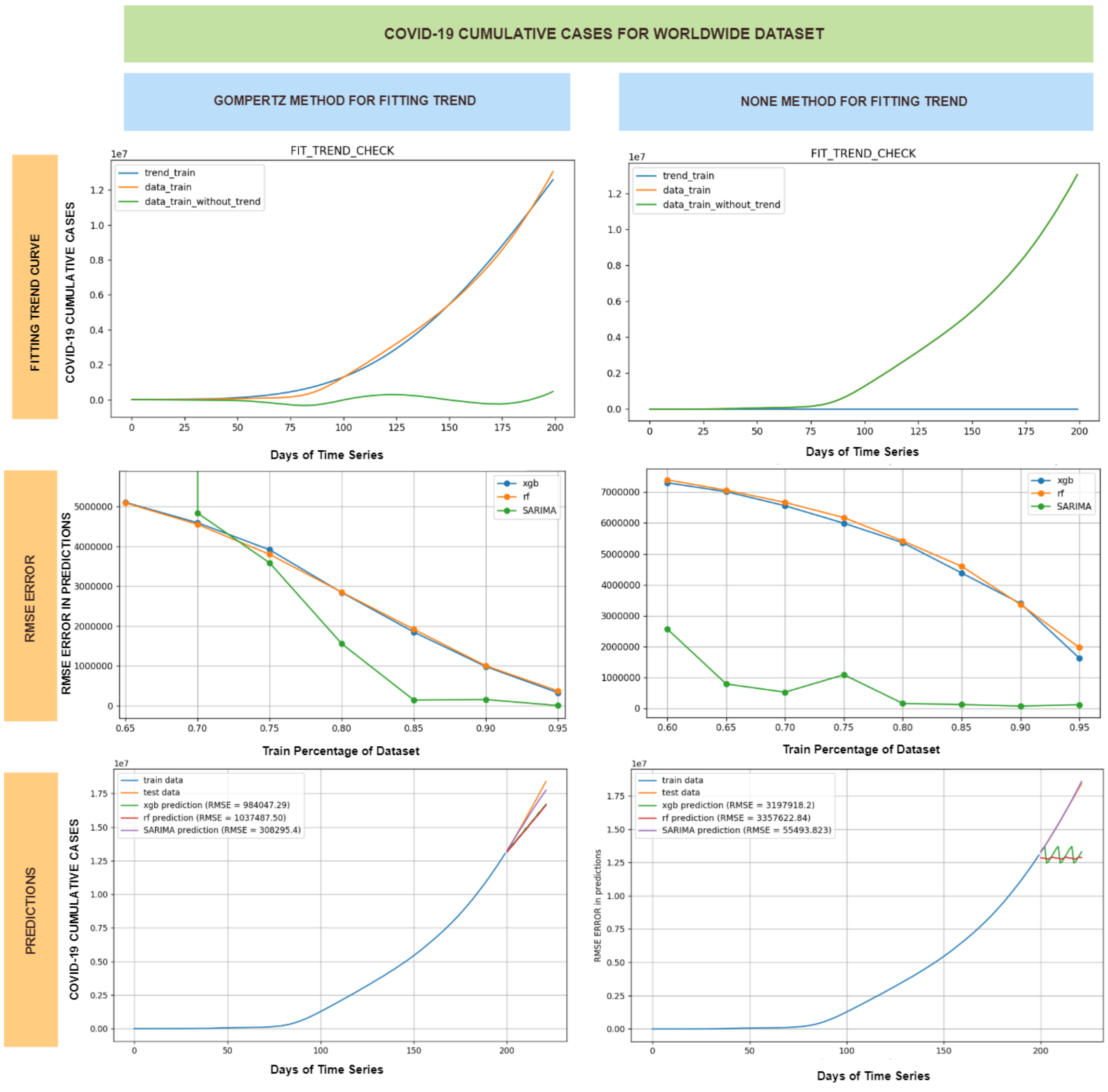


Figure 11. SIMULATION 1: Cumulative worldwide cases. Trend fitting curves, RMSE error comparative for different ML methods and Predictions for 90 % train/test.

In this simulation (Figure 11), we can conclude the **SARIMA** is the method of ML that **works better** obtaining less error for all the partitions of train / test dataset.

Using **Gompertz method** to fit the curve we are **helping the XGBoost and Random Forest** methods to improve their predictions: we obtain less RMSE error for different partitions; also, the distance between the RMSE error curve for the tree methods and the SARIMA is smaller and finally the predictions are also better. These tree methods have difficulties on detecting the trend of a Series, so we give it to them.

Another conclusion is that **SARIMA doesn't need the Gompertz fitting**, in fact, it works better in the case we don't apply a trend fitting, obtaining less MSE error.

One of the reasons of being worse is that the Gompertz fitting is not perfectly accurate, and we have some divergences on the trend fitting that after causes this error in SARIMA prediction.

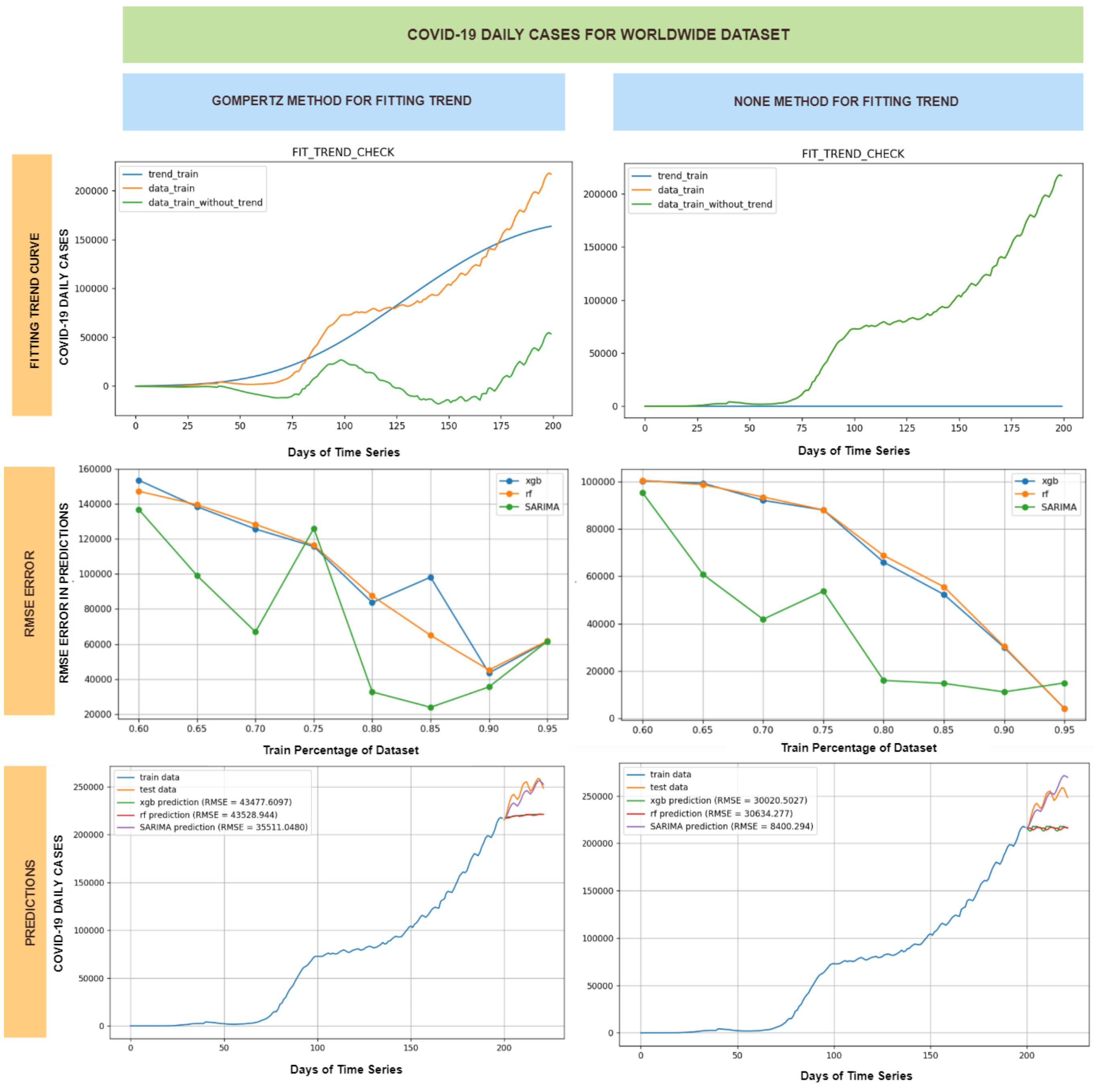


Figure 12. SIMULATION 2: Daily worldwide cases. Trend fitting curves, RMSE error comparative for different ML methods and Predictions for 90 % train/test.

Looking the second simulation (Figure 12) for the Daily worlwide cases we also conclude **SARIMA is working the best**. However, now using the **gompertz curve** (in differential mode) we **don’t obtain a proper adjusting** of the trend. This is producing more error in the RMSE for all methods and is not helping the trees method. Therefore, we conclude fitting the trend with accuracy is very relevant to make reliable predictions.

Despite that, SARIMA can overcome this problem and make a reasonable prediction.

Both simulations have demonstrated that SARIMA has good power of prediction using this dataset (222 samples).

In order to remark again the results, the next two pictures show the comparison between the RMSE error for different ML models varying the train/test percentage with Gompertz fitting model and without that.

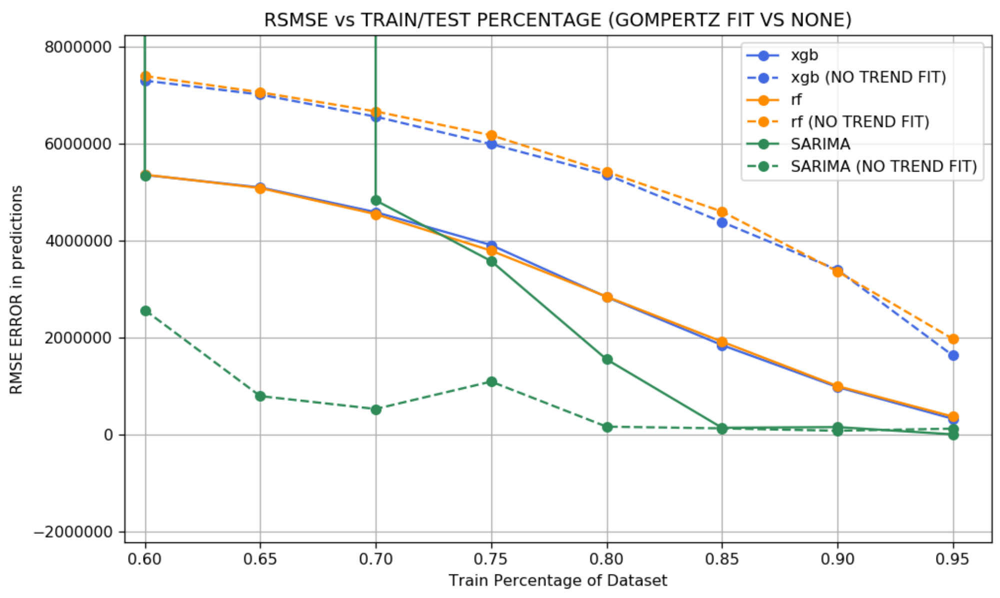


Figure 13. RMSE error comparison between Gompertz fitting and None fitting. Results for different ML methods varying the train/test percentage in Simulation 1.

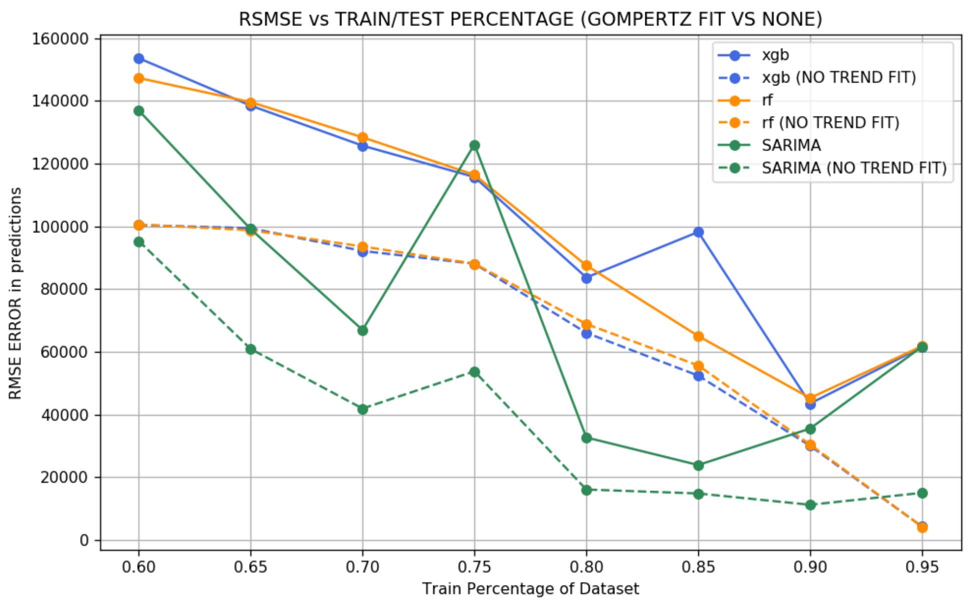


Figure 14. RMSE error comparison between Gompertz fitting and None fitting. Results for different ML methods varying the train/test percentage in Simulation 2.

Another thing to comment of these simulations is that increasing the Train Percentage of the time series we obtain less RMSE Error. This is a reasonable result that verifies the program is working well: how much less training data we have and much days we have to predict we will have less accuracy, i.e., we work with more uncertainty.

Finally, it is important to notice we haven't shown the results for the NN method implemented in our program. We haven't included them on the previous simulations because it is not working well for predicting: it is unstable and with more RMSE error than the others. The reasons could be mainly: we are working with few samples and Neuronal Networks works fine with a big quantity of data; we haven't selected the proper neuronal network and it needs more tuning and as a consequence, more time to work on that. This behaviour of the NN implemented is repeated for all the simulations.

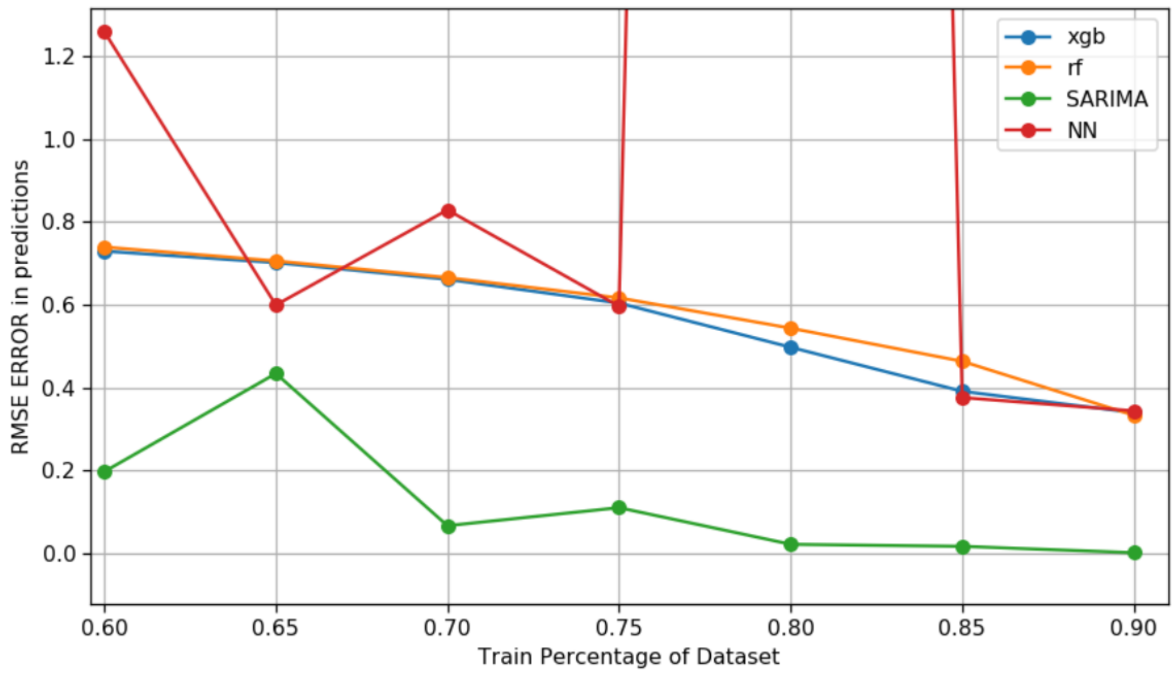


Figure 15. RMSE error for different ML methods varying the train/test percentage in Simulation 1

Looking the picture, we can see it doesn't improve respect RF and XGBoost predictions and is more unstable.

KSCHOOL

B. Predictions on COVID-19 worldwide deaths

Now we make the same simulations for the COVID-19 worlwide deaths:

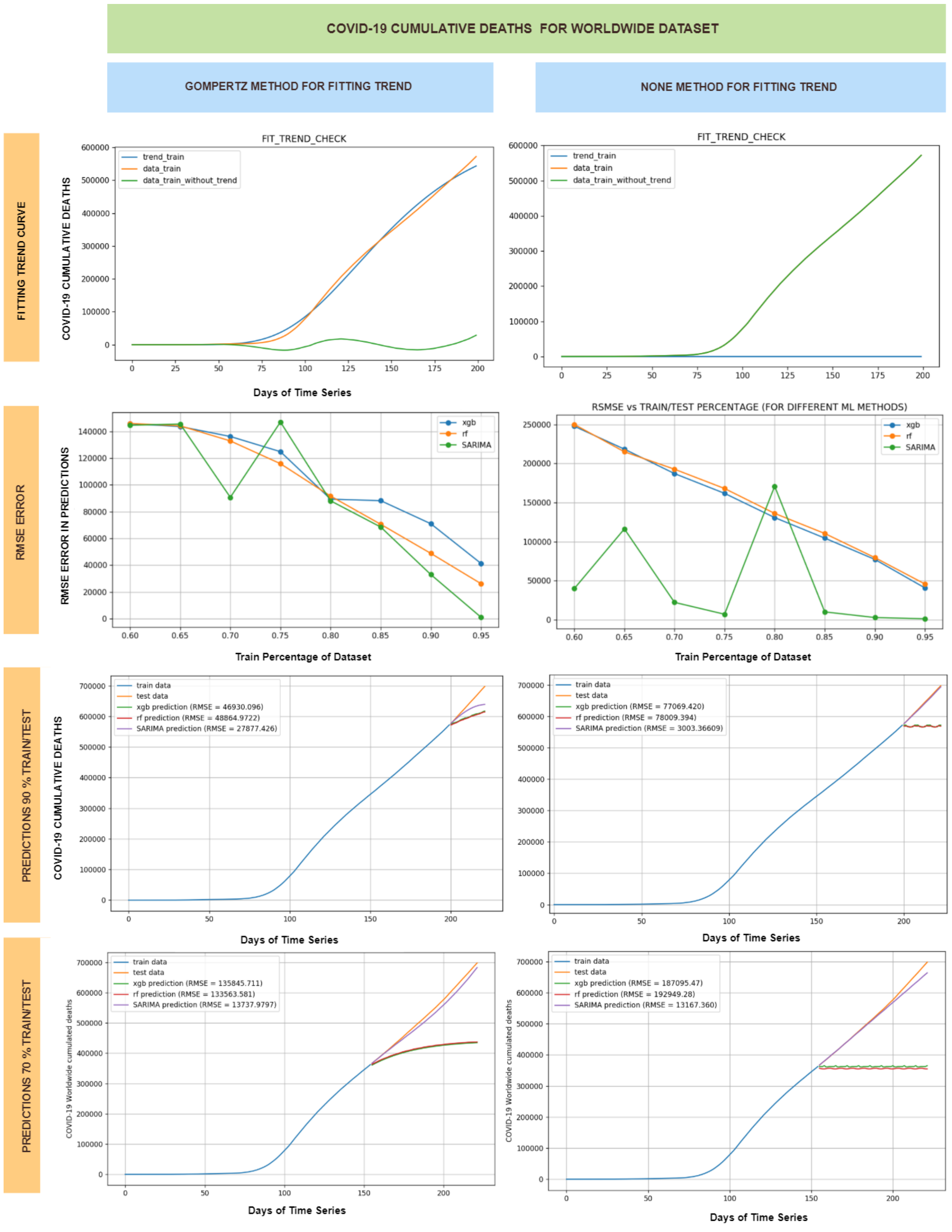


Figure 16. SIMULATION 3: Cumulative worldwide deaths. Trend fitting curves, RMSE error comparative for some ML methods and Predictions for 90% and 70% train/test.

In this simulation (Figure 16), we can obtain similar conclusions to the cumulative worlwide cases (Simulation 1). SARIMA is the best method with enough power of predicting. The method of fitting that adjust better to the curve is the Gompertz method. It helps the trees method to capture the trend but SARIMA works better without it. Random Forest and XGBoost share similar error test results. The difference in this simulation is we have obtained the prediction for 90 % train/test partition and the 70%.

Comparing the 90% train/test predictions and 70% train/test predictions we can notice the predictions **for the tree methods get worse considerably reducing the train size**. The reason that explains this is that only using the 70% for the train, the trend of the data series is fitted with the **Gompertz method in a so early stage that** the method doesn't calculate with precision the final value "c" of the Gompertz curve. Now we observe the daily worldwide deaths and their predictions:

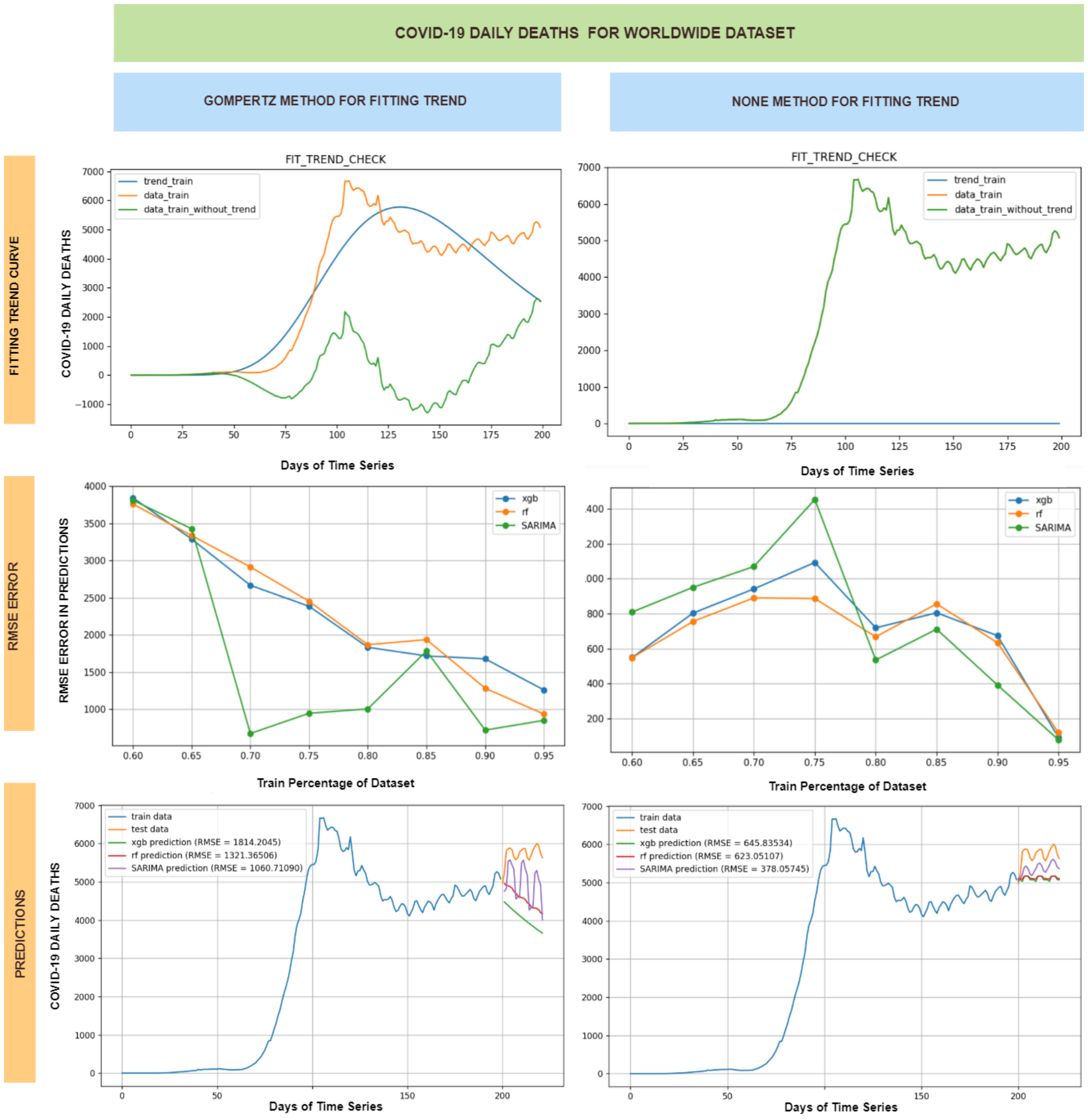


Figure 17. SIMULATION 4: Daily worldwide deaths. Trend fitting curves, RMSE error comparative for different ML methods and Predictions for 90 % train/test.

This simulation is similar to Simulation 2 because the fitting using Gompertz curve is not proper and it penalizes the results a lot for all the ML methods.

In order to remark again the results, the next two pictures show the comparison between the RMSE error for different ML models varying the train/test percentage with Gompertz fitting model and without that for the previous two simulations:

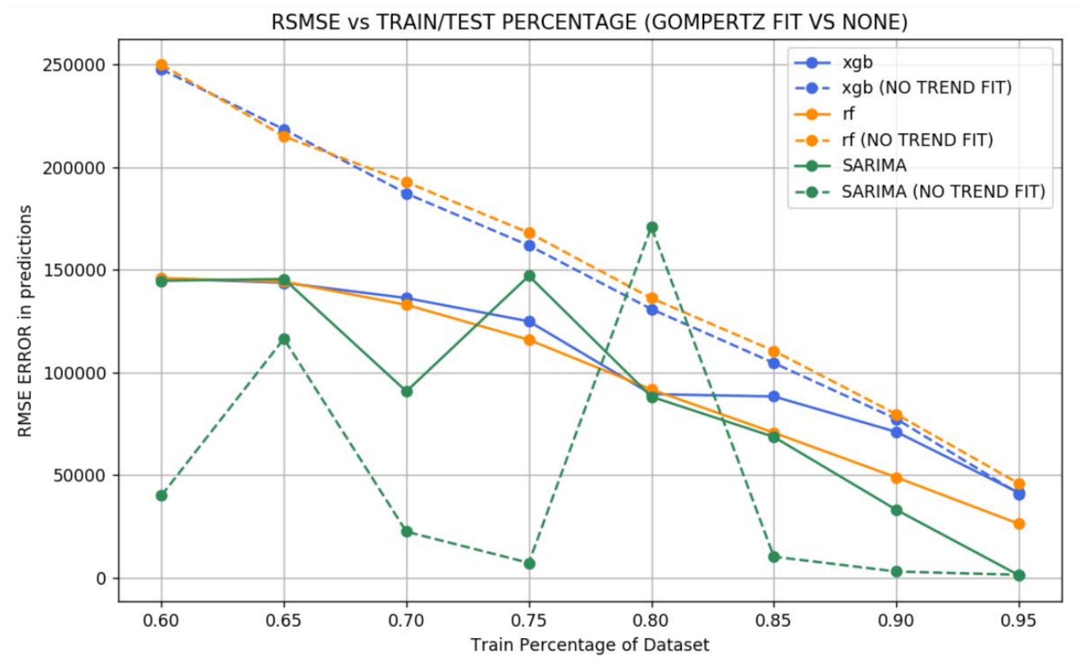


Figure 18. RMSE error comparison between Gompertz fitting and None fitting. Results for different ML methods varying the train/test percentage in Simulation 3.

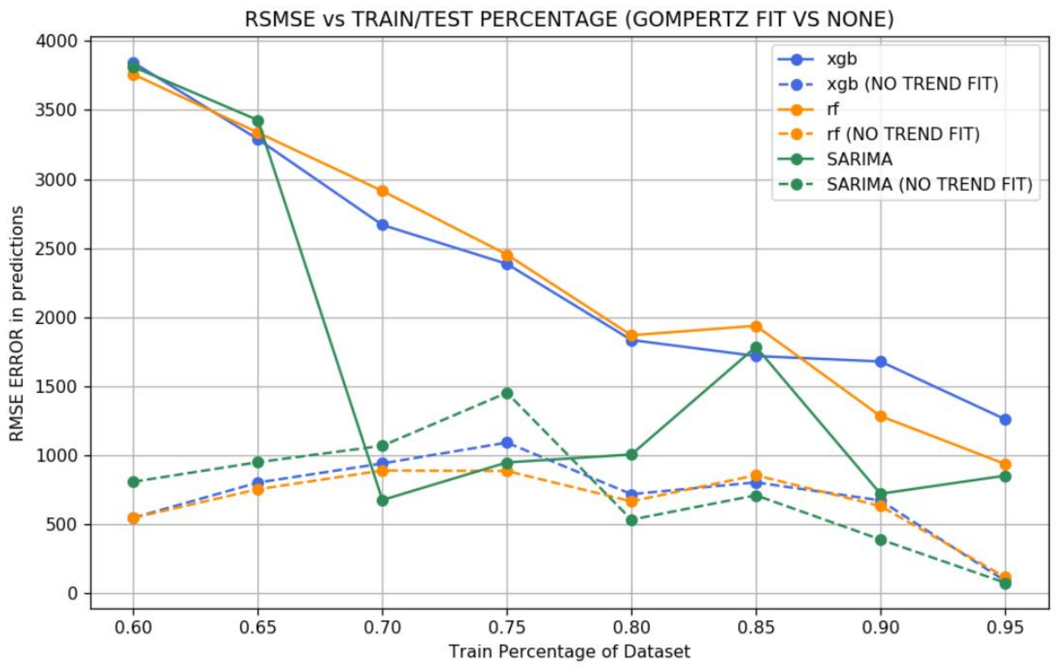


Figure 19. RMSE error comparison between Gompertz fitting and None fitting. Results for different ML methods varying the train/test percentage in Simulation 4.

The prediction for the daily worldwide deaths made with SARIMA without trend fitting seems good and reasonable. We have executed the program 15 times with this configuration to see if the prediction results are consistent (Figure 20). The RMSE error for these iterations varies between 150 – 700 unities. We can see the range of error prediction for these iterations where the orange curve is the test data. Therefore, we can verify the program has working well and these predictions are stable and consistent. These 15 executions have been done with max_iter = 80 for the Bayes Optimization method.

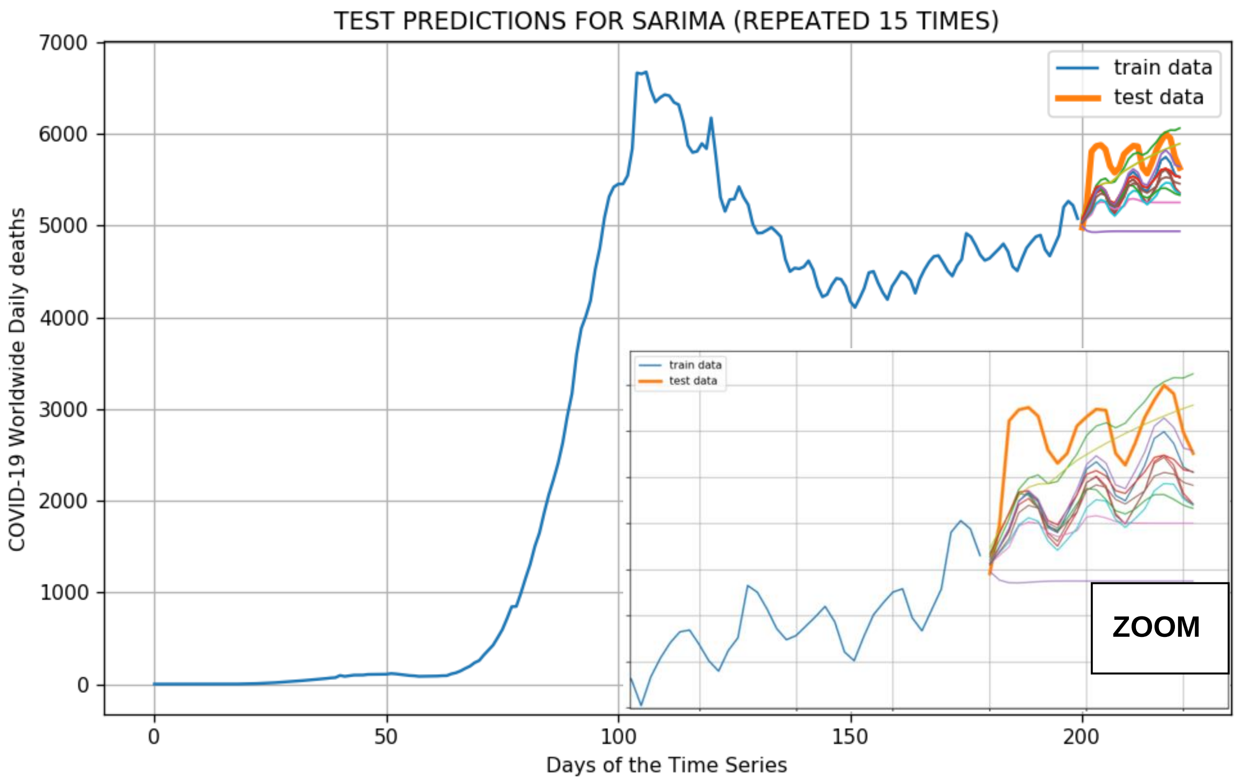


Figure 20. Repeated predictions for COVID-19 daily worldwide deaths using SARIMA without any trend fitting.

Lastly, another aspect to comment of the simulation 3 (Fig 16) is that we can use polynomic fitting to capture the trend. It adapts very well to the train part but it has very bad results for predicting the trend on the test part. Trying different simulations, we see the curve at test part grows or decrease a lot, being far away of the trend of the real Time Series. The main cause of that is that this fitting is producing overfitting.

Logistic fitting is very similar to Gompertz curve but have a faster growth rate reaching the plateau, and it contrasts with an epidemic behaviour. For all these reasons, we are using all time Gompertz fitting in our simulations.

C. Predictions on COVID-19 deaths in Cataluña and Madrid communities

In this simulation we are going to work with Time Series in differential mode to make predictions: the number of COVID-19 daily deaths in Cataluña and Madrid communities, counted together. We are fitting a dataset with less samples than before (80 samples as the maximum number of available samples for training). Therefore, we will expect poor power of prediction. The simulation is done with a Gompertz method for fitting the trend and without applying any fitting as the previous simulations.

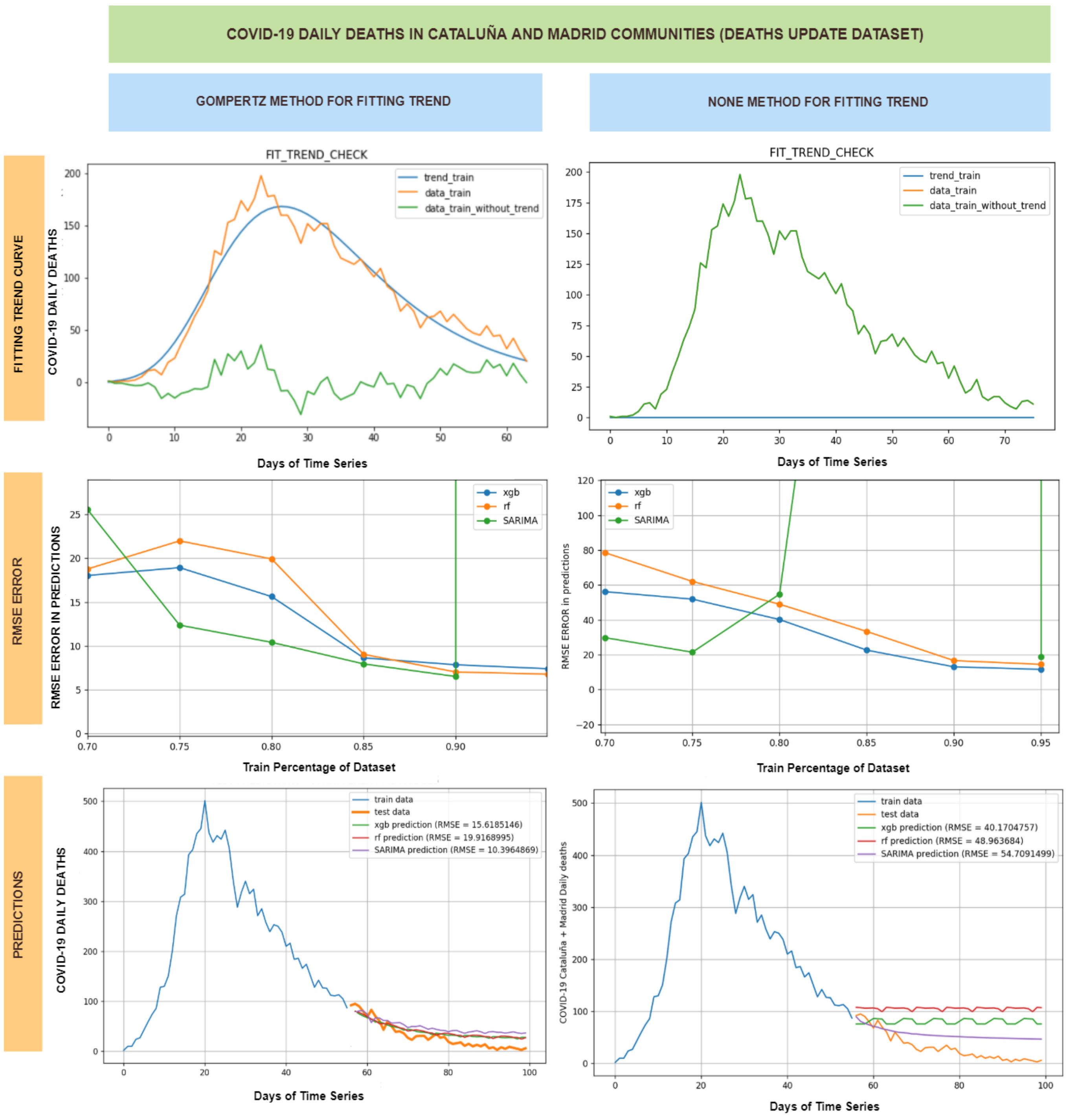


Figure 21. SIMULATION 5: Daily deaths for Cataluña and Madrid. Trend fitting curves, RMSE error comparative for different ML methods and Predictions for 90 % train/test.

Looking the results (Figure 21) we are seeing a **different behaviour for the previous simulations** in the other dataset. **SARIMA is more unstable than the other methods** and it is not clear it is the best method for predicting here. Depending the train/test percentage the errors of this method or the tree methods are similar. Now the gompertz method is given better results than not applying any fitting on the trend. The reason is that now, the machine learning model have low power of prediction and the **predictions mainly are based on the calculation on test of the gompertz extrapolation trend.**

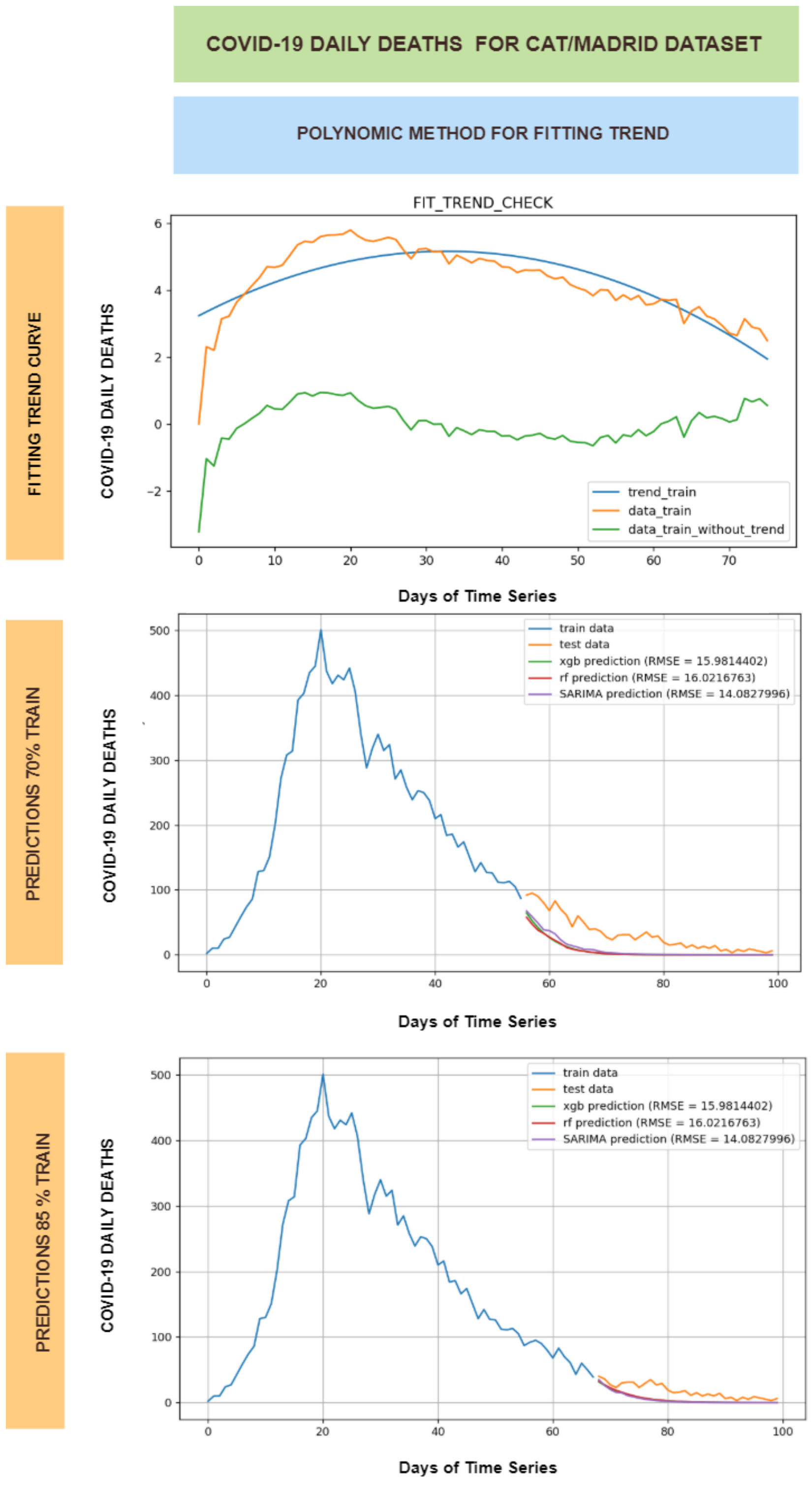


Figure 22. SIMULATION 6: Daily deaths for Cataluña and Madrid. Trend fitting using polynomic curve. Predictions on 70% and 85% of percentage train/test

In Simulation 6 (Figure 22), a **polynomic fitting** has been applied to obtain the trend. It has a correct fitting and the results aren't so bad. The RMSE errors are similar than using the polynomic method. Another time, all the weight of prediction is on fitting the trend. However, this method is low reliable because it usually gives results with overfitting.

The program implements the **SIR Model** to fitting the trend of the curve. However, the curve doesn't fit as well as gompertz curve. The reasons are: SIR Model is strongly conditioned by initial conditions for solving their differential equations and it can cause variability in the fitting ; furthermore, SIR Model is a simply version of the real behaviour of expansion of the pandemic because doesn't consider interaction between cluster and other factors, there are more complex models as SEIR Model; the cases and deaths notified by the state don't follow these equations at all, due to the variability in PCR test, in confirmation of deaths, in changing the way of counting, etc. Due to all these reasons, a SIR Model is not easy to adjust for fitting COVID-19 curves. It requires another work focusing only on that kind of model. Therefore, it is not a good idea to implement it in a "fast" way to predict the trend.

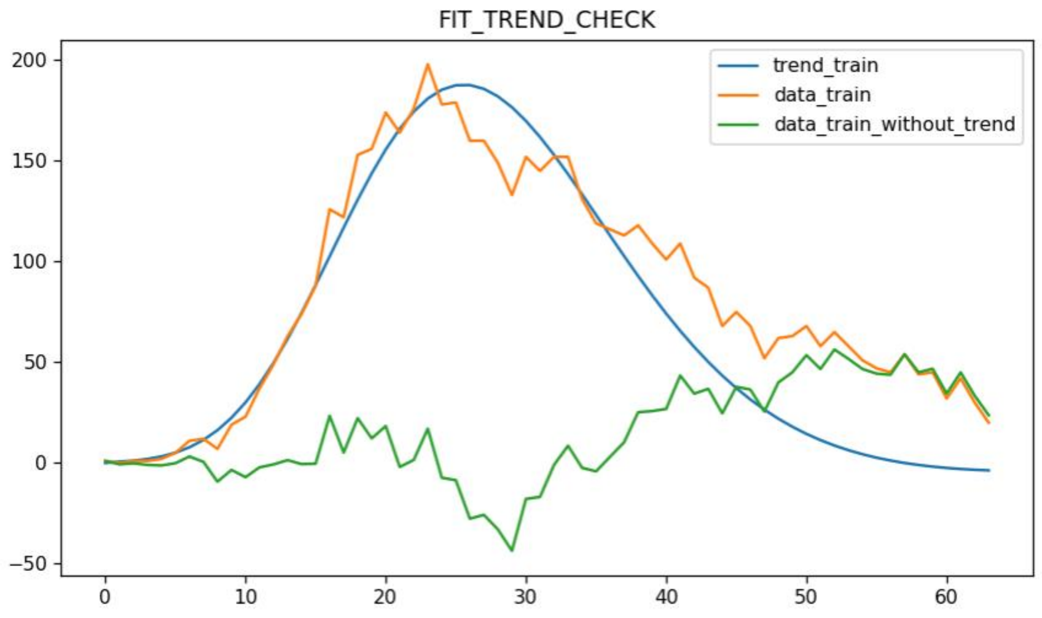


Figure 23. Fitting the trend for death cases with SIR Model. X-axis: days, Y-axis: number of deaths.

4.3 Summary of the simulation results

Summarizing the results, the SARIMA model without fitting the trend has been the best configuration for our predictions in the worlwide cases and deaths. Using this dataset, SARIMA has enough power of prediction. The other methods as XGBoost and Random Forest are in most of the occasions with a higher RMSE error than SARIMA. They are helped by gompertz methods to fitting the trend but it not enough to overcome SARIMA.

For the predictions of the daily deaths in Madrid and Cataluña in the first wave, all the machine methods obtain similar results in terms of RMSE and SARIMA model become very unstable. They don’t have enough power of prediction and all the weight of the results depends mainly on fitting the trend with a proper method as Gompertz or Polynomic. However, it is important to know they can have overfitting, over all, the second one.

Other aspects of the simulations have shown that the implemented Neuronal Network doesn’t have enough power of prediction with these datasets or need more tuning and research to apply it. The SIR model hasn’t been a good solution to fit the trend of the curves too.

5. Conclusions and future development

As a conclusion, we have reached the goals of the project. First, we have created a configurable program that implements different modules as Bayesian Optimization tuning with Time Splitting, Decomposition Methods, etc. for making predictions of COVID-19 temporal series.

The program allows to execute simulations in a very automatic way. Profiting that, we have compared the error of different common machine learning methods applied on different datasets on COVID-19 cases and deaths.

Summarizing in a brief way the results, the SARIMA model is the most proper one to make predictions with the COVID-19 datasets available. The best model for fitting the trend of the curves is the Gompertz model. We can fit the trend of the COVID-19 Time Series applying this method (especially for cumulative time series) and it will help some machine learning methods as Random Forest or XGBoost for their predictions but it is not necessary for SARIMA. Despite this help, the other methods don't overcome SARIMA. This behavior is not seen in the second dataset tested, where we have lower samples and the power of prediction for ML methods was lower. In fact, for low samples as the daily deaths of Cataluña and Madrid the results depends mainly on fitting the trend with a proper method as Gompertz or Polynomic.

For future development it will be interesting to implement a more complex SIR Model making more research to adjust the COVID-19 Times Series in a more accurate way. Furthermore, it will be interesting to apply other strategies of the literature to forecast COVID-19 times series and implements them in our program. Some of these strategies are: using a Kalman filter to forecasting [13]; fitting the trend of the cases curve with Weibull function and Gaussian function and compare with the already implemented ones [14]; using the FBProphet [15] of Facebook for forecasting and compare the predictions obtained with the predictions of our program.

Bibliography

[1] Courtney Cochrane. Time Series Nested Cross-Validation. *Towards data science*. May 2018.

[2] Packt Editorial Staff. Cross-Validation strategies for Time Series forecasting. *Technology news, analysis, and tutorials from Packt.* May 2019.

[3] Will Koehrsen. A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning. *Towards data science*. June 2018.

[4] Santiago Casas. Using Python scientific tools for a (rough) analysis of COVID-19 data. *Medium*. Mar 2017.

[5] Martí Català; Sergio Alonso; Enrique Alvarez-Lacalle; Daniel López; Pere-Joan Cardona; Clara Prats. Empiric model for short-time prediction of COVID-19 spreading. *Towards data science*. May 2020.

[6*]* Thomas J.Sargent; John Stachurski. Modeling Covid19. Quantitative Economics with Python*. QuantEcon. 2020.*

[7] Henri Froese. Infectious Disease Modelling: Beyond the Basic SIR Model. Towards data science. April 2020.

[8] Will Koehrsen. How to Generate Prediction Intervals with Scikit-Learn and Python. *Towards data science*. May 2019.

[9] Milind Yadav; Murukessan Perumal; Dr.M Srinivas. Analysis on novel coronavirus (COVID-19) using machine learning methods. *Chaos, Solitons & Fractals*. Volume 138. 2020 May 2018.

[10] Venelin. Time Series Forecasting with LSTMs for Daily Coronavirus Cases using PyTorch in Python. *Curiousily blog.* March 2020.

[11] Usman Malik. Time Series Prediction using LSTM with PyTorch in Python. *Stack Abuse Tutorials*. 2020

[12] Peipei Wang; Xinqi Zheng; Jiayang Li; Bangren Zhu. Prediction of epidemic trends in COVID-19 with logistic model and machine learning technics. *Chaos, Solitons & Fractals*. Volume 139. October 2020.

[13] Ran Kremer. Using Kalman Filter to Predict Coronavirus Spread. *Medium*. Feb. 2020

[14] Shreshth Tuli; Shikhar Tuli; Rakesh Tuli; Sukhpal Singh Gilld. Predicting the growth and trend of COVID-19 pandemic using machine learning and cloud computing. *Internet of Things*. Volume 11 September 2020

[15] Nikolaus Herjuno Sapto Dwi Atmojo. Hacking Time-Series Forecasting Like a Pro with FBProphet. *Medium.* Jul 2019.