First Part: COVID-19 in Spain

Advanced Regression And Prediction

Roberto Jesús Alcaraz Molina

09/05/2021

Contents

1 INTRODUCTION

This project is the second and final project of the *Advanced Regression and Prediction* course. As we did in the first part, we are going to analyze and try predict the behavior of the coronavirus crisis in Spain, but this time, we are going to use **machine learning** tools.

Even though for the first part of the project we took all days from 22nd of January 2020 until the 11th of March 2021, we will consider also all new observations until the 2nd of May, since the more data, the better will be our analysis.

This project will be divided in four parts: firstly, we are going to review the data cleaning and preprocessing steps we did in the previous part and comment about the results we had; then we will start the machine learning process, where we will explain, tune and apply 6 differents machine learning models (K-Nearest Neighbors (KNN), Support Vector Regression (SVR), Regression Trees, Random Forests, XGBoost and Neural Network (NN)); after that, we will select the best model in addition to creating an ensemble model with the best set of models; and finally we will give our conclusions.

To put us in context, we can remember the preprocessing we did to our data:

- First of all, we selected the target variables of our study, which are the deaths and confirmed cases, and some other variables that could be interesting for our model.
- Secondly, we realized that we had some missing values in our two variables of interest, but they were easy to impute since they indicated that there were no cases.
- After that, we recategorized some of our factor variables because they were encoded as integer, as well as removing and modifiying some levels that were not entirely true in Spain.
- Next, we fixed some errors that were related with the data collection, for instance, there were some days that had lower confirmed cases or deaths than the day before.
- Then, we aggregated the data to have **weekly data** in order to decrease the noise that appear due to different human errors. For the numeric variables, we added all values, and for the categorical features, we took the mode, i.e., the most frequent value in every week. Below, we can observe how our main variables of interest look like:

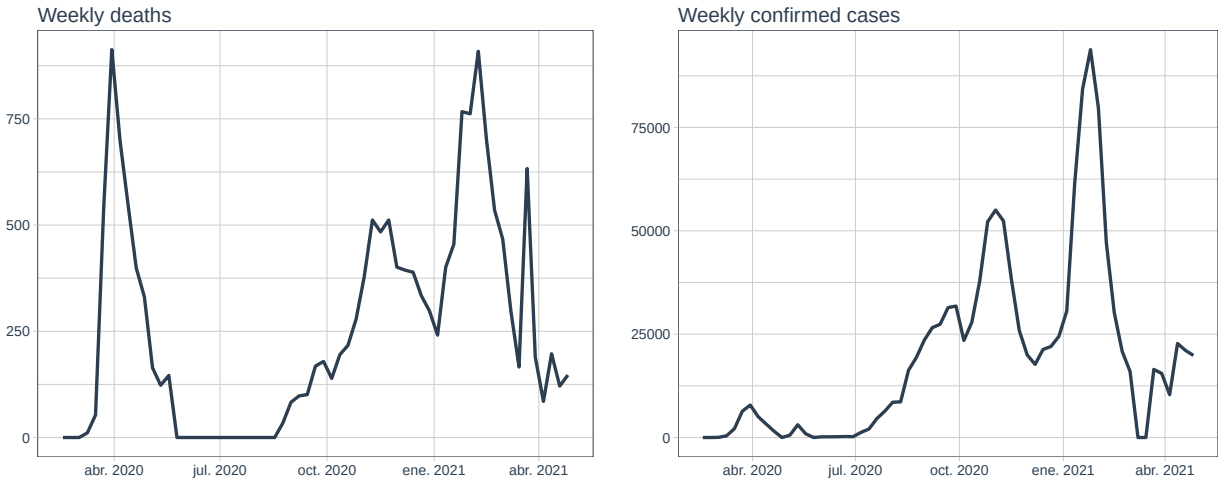

Figure 1: Weekly deaths and confirmed cases due to COVID-19 in Spain

- Finally, we did some feature engineering, adding four new variables that took into account the time (month and year), and past cases (deaths and confirmed cases 3 weeks). Therefore, the variables we

finally considered for our model can be seen below. After all these steps, we are in position to start with the machine learning process.

Table 1: Description of the variables

Variable	Description
`date`	Observation date (identifier).
`deaths_week`	Number of deaths per week.
`lag_3_deaths_week`	Number of deaths per week three weeks ago.
`confirmed_week`	Number of confirmed cases per week.
`lag_3_confirmed_week`	Number of confirmed cases per week three weeks ago.
`vaccines_week`	Number of doses administered (single dose) per week.
`month`	Observation month .
`year`	Observation year.
`stay_home`	Indicates the measures of staying at home.
`school_closing`	Indicates the measures in education.
`workplace_closing`	Indicates the measures of the workplace.
`gatherings_restrictions`	Indicates the measures of gatherings.
`internal_movement_restrictions`	Indicates the measures of the movements between regions.

2 MACHINE LEARNING

In the following chapter, some well-known machine learning tools will be applied. We will explain the preprocessing steps that every algorithm needs, which are their main hyperparameters and how they work in order to tune their hyperparameters and draw comparison between the models.

First of all, since we have new data, we have to repeat the recursive feature selection to select the most important features by means of a Random Forest. For the data set that will be used for deaths models, it removes the variables `stay_home` and `gathering_restrictions`, nonetheless, for the confirmed cases, it removes `workplace_closing` and again `gathering_restrictions`.

Next, we need to divide our training data into folds to tune the model parameters and to compare them. To do that, we will use again the function called `rolling_origin` from the `rsample` package (from `tidymodels`), but now there will be 36 weeks to train in each fold, since as we saw in the previous part, 12 weeks were too little. Moreover, the number of folds will increase up to 25 folds, so the parameters will be tuned in a more efficient way.

After all these explanations, let's start with the first model: **K-Nearest Neighbors**.

2.1 K-Nearest Neighbors (K-NN)

The nearest neighbors algorithm **(k-NN)** is a supervised machine learning model normally used for both classification and regression. Firstly, it is important to define a distance between the observations, usually the euclidean is the most popular. Then, the distance between the observation to classify and all points in the sample are calculated. After that, we have to select a number for k, which will be the number of the k closest observations from the point that we want to classify. Finally, the observation will be classified as the average of these others k observations.

Regarding the preprocessing steps, there are not too many assumptions, but there are some changes that are recommended. For instance, we will decorrelate predictors because it improves the prediction accuracy even though it is not mandatory. Also, we will convert all categorical variables to dummy (note that most of the algorithms do it without any specification), we will normalize all predictors and finally, if necessary, we will remove those features with zero variance (those that only have one value).

With respect to the hyperparameter tuning, we have to focus on two main parameters: neighbors (obviously), and the weight function (`weight_func`), which is the type of kernel function that weights the distance between samples (*rectangular*, *triangular*, *epanechnikov*, etc). Also, we will use the Euclidean distance to carry out these steps.

2.2 Support Vector Regression (SVR)

The support vector machine, or preferably, the support vector regression **(SVR)** is an algorithm used for both classification and regression. In regression it fits a function to create a tube with width ε, where the points inside that tube will be used to predict the new points.

For this algorithm, the preprocessing steps we need to do are the same that we did for the k-NN algorithm: decorrelate predictors, create dummy variables, normalize the predictors and remove those with zero variance.

We will consider two algorithms from this family: the *polynomial SVR* and the *radial basis function SVR*. The difference between them is the kernel function used to map the original data set into a higher dimensional space. Therefore, for the polynomial SVR we will have a parameter for the polynomial degree (`degree`) and another one for the scaling factor (`scale_factor`) for the kernel; and for the rbf SVR we will have an hyperparameter σ (`rbf_sigma`), which determines the reach of a single training instance. Also, there are another two common hyperparameters for both algorithms: the cost (`cost`) of predicting a sample within or on the wrong side of the margin, and the ε (`margin`) we have said before.

2.3 Regression Trees, Random Forest and eXtreme Gradient Boosting

A decision or regression tree **(RT)** is the result of asking an ordered sequences of questions finishing in a prediction for the observation. The starting point of the classification tree is called root node and consist of the entire training set in the top of the tree. A node is a subset of the set of variables and it can be a parent node, which is a node that splits into another two nodes if the condition is satisfied or not, and a terminal node, which is a node that does not split and is assigned a value. This algorithm has the advantage of interpretability or small bias, however, the variance tend to be very large. Therefore, to reduce that variance, we combine some decision trees, which leeds to Random Forest.

The Random Forest **(RF)** algorithm is an ensemble of regression trees. Firstly, we start with B boostrap samples drawn from the training set, which reduce variance, due to aggregation, and bias, if the trees are fully grown. For each sample, a decision tree is grown. Then, we predict every observation in each tree, and the final value for the observation will be the average of the results of each tree.

The third one, e**X**treme **G**radient **B**oosting or **XGBoost**, is a decision tree ensemble machine learning algorithm that, unlike random forest, uses a boosting framework, i.e., instead of taking the average of multiple decision trees as the RF does, the models are built sequentially by minimizing the errors from previous models while boosting the influence of high-performing models.

The principal advantages of the first two models is that they are scale invariant, i.e., it does not matter the scale of the features, so in these models we will not apply any preprocessing steps for the variables. Another advantage is that it is robust to irrelevant predictors, because if the variables is not very relevant, it is very possible that they will not use it to split the data. In contrast, for the XGBoost we should decorrelate predictors, create dummy variables and remove the zero variance variables.

For the decision trees, we will have to tune three hyperparameters: a cost parameter for the complexity of the tree (`Cp`), the maximum depth for the tree (`tree_depth`) and the minimum number of observations in a node that are required for the node to be split further (`min_n`). Secondly, for the Random Forest we will need to tune the number of predictors that will be randomly sampled at each split when creating decision trees (`mtry`), the number of trees contained in the ensemble (`trees`) and again, `min_n`. Finally, for the XGBoost, we have the same hyperparameters than the RF, the `tree_depth` from the decision tree and two additional hyperparameters: a number for the rate at which the boosting algorithm adapts from iteration to iteration (`learn_rate`), and the reduction in the loss function required to split further (`loss_reduction`).

2.4 Neural network

A multi-layer perceptron is a multivariate statistical technique that maps the inputs variables nonlinearly with the output variable. Between the inputs and output there is also hidden variables arranged in layers. The hidden and output layers are called nodes, neurons or processing units. Therefore, to model our problem we will have n inputs nodes, $X_1, \dots, X_n$ which are the variables, one or more layers of hidden nodes and an output node Y.

The preprocessing steps we need to follow are the same as in the majority of models: decorrelate predictors, create dummy variables, normalize the data and remove zero-variance features.

Regarding the hyperparameters, we have to tune four: the number of hidden units (`hidden_units`), a penalty for the amount of weight regularization to reduce overfitting (`penalty`), the number of training iterations (`epoch`) and the activation function, which could be linear, softmax, relu or elu (`activation`).

2.5 Model tuning and selection

Now that we have already explained all the models with their characteristics, it is time to tune their hyperparameters. To do that, since our data set is very small, we are going to consider a grid by default of size 100, which added to the 25 folds we have to tune the parameters, results in fitting 2500 models for each algorithm. To compare and tune them, we will select the one with lowest root mean square error (RMSE) and we will look also to the r-square (RSQ).

After fitting the first models, we will look at the model results to see possible errors and also, better parameter grids. We will give more appropiate values for the set of parameters based on the results obtained and we will again consider another grid of size 100 around the grid we have specified. Therefore, we will obtain 25 different metrics for each set of parameters, so we will select the one with the lowest average RMSE. These results can be seen below:

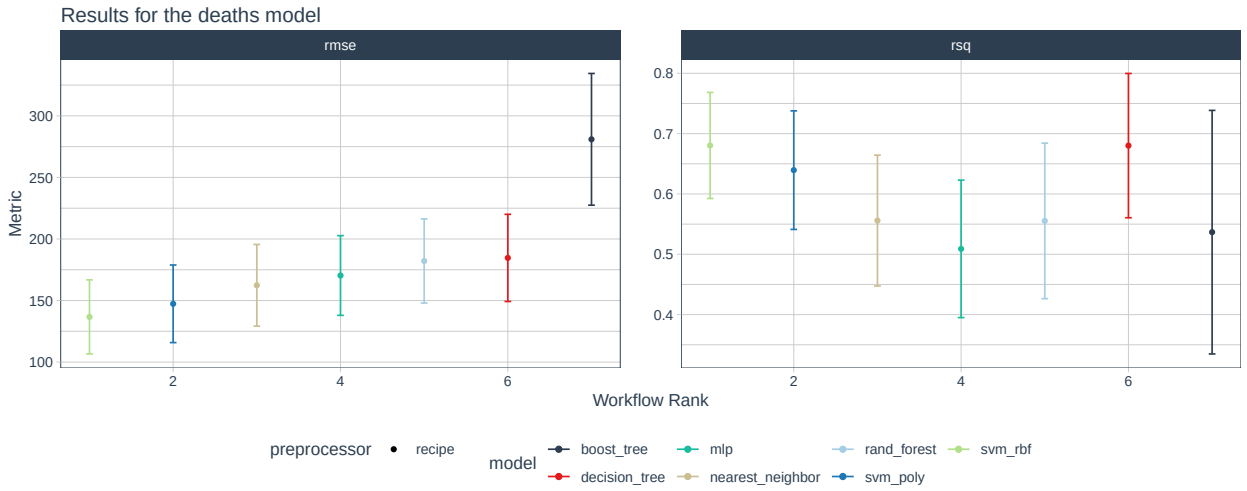

Figure 2: Confidence interval for the metrics of the best set of hyperparameters of each model for the deaths

In the graphic above, we can observe the metrics obtained from the best set of parameters and their confidence interval. For the deaths model, the model that has outperformed the rest is the **radial basis function support vector machines**, which has around 130 RMSE and almost 0.7 RSQ on average. The set of hyperparameters can be seen below:

Table 2: Final set of parameters for the radial basis function SVM

Hyperparameters	`rbf_svm`
`cost`	18.2275
`rbf_sigma`	0.00155
`margin`	0.10298

On the other hand, as we can see below, we have the results for the confirmed cases model. In this case, we can observe that the best model seem to be the decision tree, however, it does not have any value for RSQ. It occurs when the model predicts a single value for all samples, so it is not what we want. Therefore, we will select the second best in RMSE, which is also the best in RSQ: the **XGBoost**.

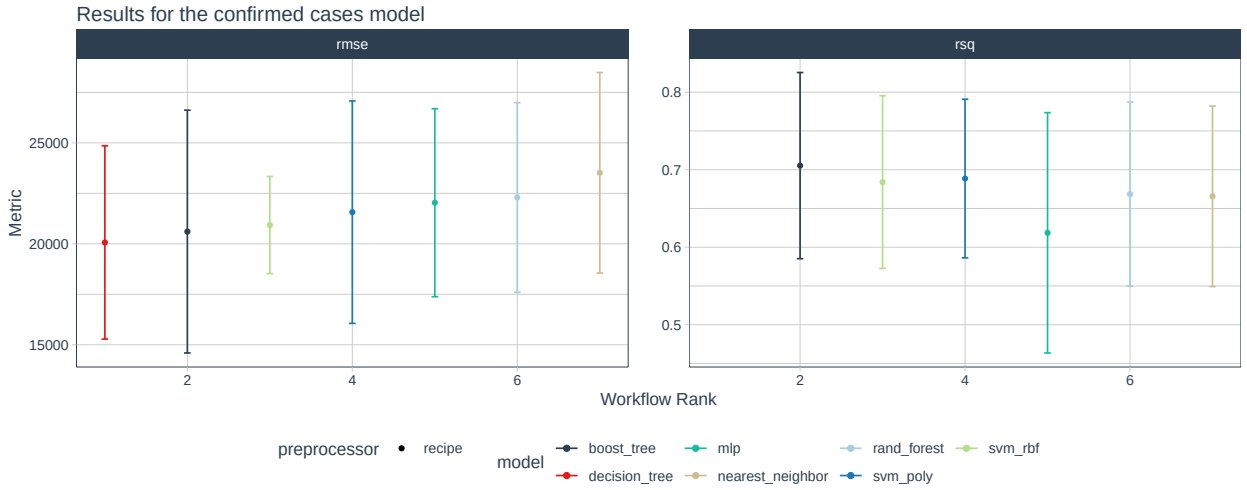

Figure 3: Confidence interval for the metrics of the best set of hyperparameters of each model for the confirmed cases

Its best set of hyperparameters is:

Table 3: Final set of parameters for the XGBoost

Hyperparameters	`xgboost`
`trees`	1279
`min_n`	9
`tree_depth`	6
`learn_rate`	0.0008
`loss_reduction`	$2.11e-09$

In the following section, in order to improve our models, we are going to make an stacking ensemble with the best set of models we have fitted before.

3 STACKING ENSEMBLE

In this section, we are going to combine those models we have created in order to make another one with better predictive power. All this process can be done thanks to the R package `stacks` from `tidymodels`,

which have the necessary tools to carry it out.

Basically, we start by creating a data set with the outcome in the validation set (in our case, every 3 weeks validation sets) and all the predictions from each candidate member. Once we have our data set ready, we will apply a model to select the best set of candidates. This process is usually called *metalearning*.

Then, since the outputs of each member can be highly correlated, we should consider a model that skips some of the methods. Therefore, the models that are used in these steps are the ones we used in the previous project: ridge regression, lasso or elastic net, depending on the degree of regularization we want. After fitting one of these models, we get a coefficient for each member, which is called *weight*, that determines which models will be taken as final members of the ensemble. After we have the weights for every member of the ensemble, we can fit the model in all the data set and predict on new data.

For our deaths model, we add all candidate members to the stack object, having finally 619 candidate members from the 7 model definitions. We do not have the 700 candidates (one for each set of hyperparameters for each model) because some models configurations of the k-NN and decision tree had errors. After that, we fitted a elastic net model, and below we can see their results:

```
-- A stacked ensemble model -------------------------------------

Out of 619 possible candidate members, the ensemble retained 10.
Penalty: 0.1.
Mixture: 1.

The 10 highest weighted members are:
# A tibble: 10 x 3
   member                    type             weight
   <chr>                     <chr>             <dbl>
 1 xgboost_deaths_wf_6_1_046 boost_tree 187867.
 2 svm_rbf_deaths_wf_3_1_025 svm_rbf         0.492
 3 mlp_deaths_wf_7_1_050     mlp             0.206
 4 mlp_deaths_wf_7_1_056     mlp             0.155
 5 mlp_deaths_wf_7_1_033     mlp             0.141
 6 mlp_deaths_wf_7_1_040     mlp             0.132
 7 mlp_deaths_wf_7_1_004     mlp             0.0824
 8 mlp_deaths_wf_7_1_045     mlp             0.0806
 9 mlp_deaths_wf_7_1_048     mlp             0.0184
10 mlp_deaths_wf_7_1_075     mlp             0.00210
```

It finally select ten models: XGBoost, rfb SVM and 8 multilayer perceptrons. As we can observe, for the first model, it gives a very large weight. Below we can see why: it predicts very small numbers, so it needs a larger weight.

Table 4: Predictions from the XGBoost model

.pred	deaths_week
0.5001799	279
0.5001799	379
0.5001799	512
0.5002698	379
0.5002698	512

Also, we need to ensure that the predictions are not very correlated. To do that, we can observe a correlation

plot between the ten selected members. There are a couple of them that have a high correlation (>0.8) but the majority are fine.

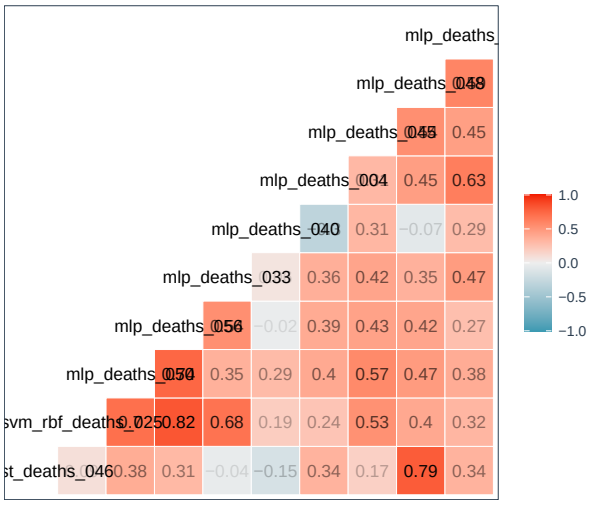

Figure 4: Correlation plot between the selected members

On the other hand, let's see the ensemble model for the confirmed cases. Below, we can observe that it only chooses 4: a rbf SVM, a multilayer perceptron and two decision trees, giving the largest weight to the SVM model. Also, as we can observe in the correlation plot, the two decision tree models are very correlated but since the rest are not, and the weights are very low, it is not a big problem.

Figure 5: Weights for the selected members and correlation plot between them

Once we have all these models fitted and selected, we are going how they predict in the training and testing sets, and we will give our conclusions.

4 PREDICTIONS AND CONCLUSIONS

As we saw above, the best model for the deaths was the radial basis function SVM, and for the confirmed cases, the XGBoost model. First of all, we are going to see both model predictions in the training and testing sets and then, we will see the behavior of the ensemble models.

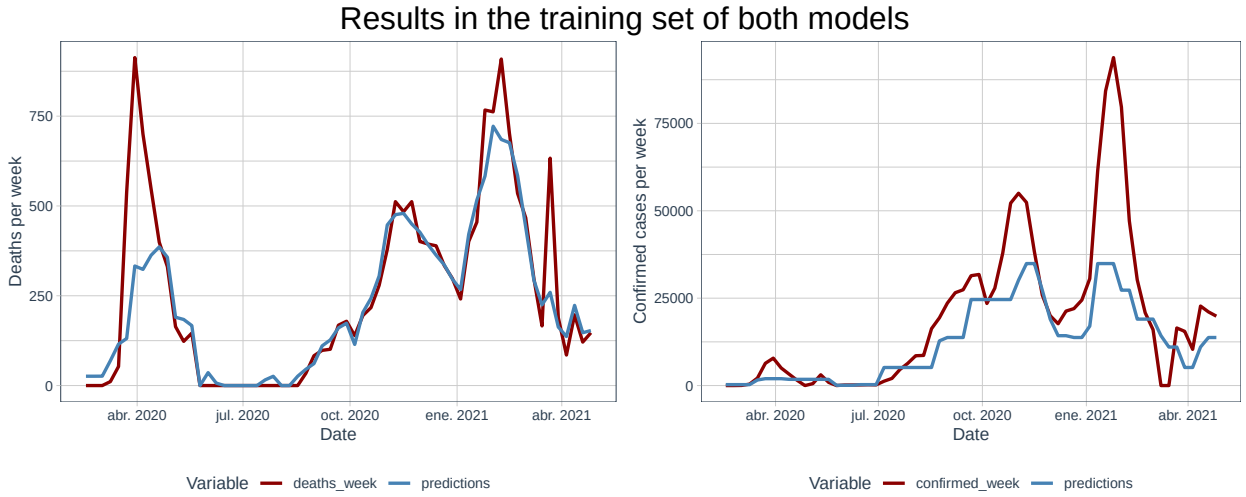

Figure 6: Predictions in the training set from the radial basis function SVM (deaths) and XGBoost (confirmed)

As we can observe, both models work in a good way but they are missing some peaks. For the deaths model, it does not get the peaks in April 2020 and either in April 2021. In the confirmed cases model it happens practically the same, in January 2021 it predicts a much lower value than it was. Now, let's see the predictions in the testing set:

Table 5: Results in the testing set

Date	Deaths	Pred. Deaths	Dif. Deaths	Confirmed	Pred. Confirmed	Dif. Confirmed
2021-05-03	77	407	-330	16353	24607	-8254
2021-05-10	103	53	50	13984	13748	236
2021-05-17	93	47	46	11061	13748	-2687

The results in the testing sets are ver good. For the deaths model, the difference in the first week is very large but for the next two, the differences are very similar. Moreover, for the confirmed model, the second week is almost perfectly predicted, and the other differences are not very large, which is very good. Now, let's see the results for the ensemble models:

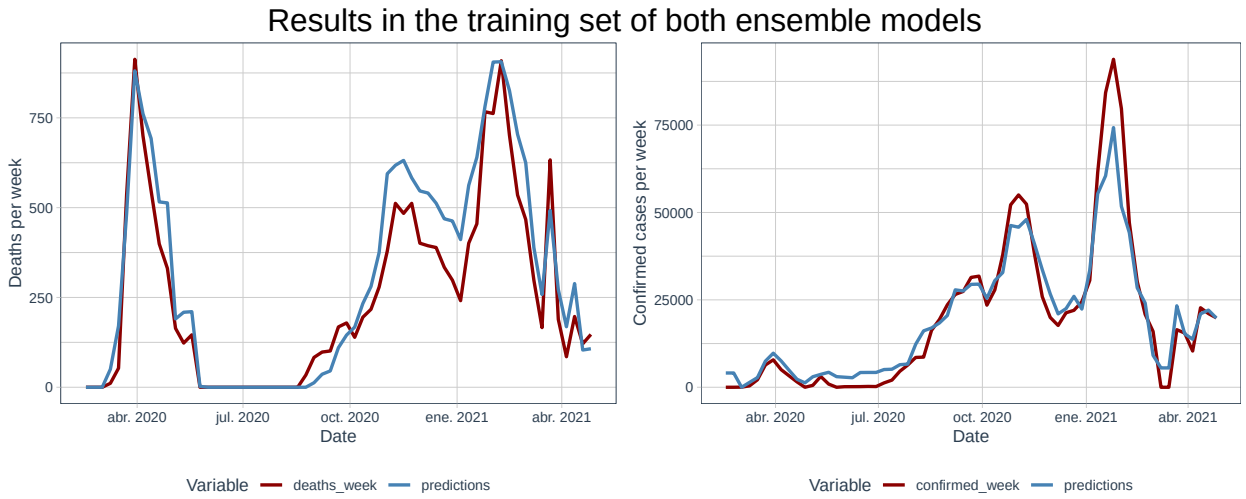

Figure 7: Results in the training set for the ensemble models

Now, it can be seen that the predictions fits way better to the real values, reaching the peaks that, with the previous models, we were not able to get. Let's see the results in the testing set.

Table 6: Results in the testing set for the ensemble model

Date	Deaths	Pred. Deaths	Dif. Deaths	Confirmed	Pred. Confirmed	Dif. Confirmed
2021-05-03	77	503	-426	16353	8286	8067
2021-05-10	103	0	103	13984	26090	-12106
2021-05-17	93	0	93	11061	27355	-16294

Now, the results are much worse, and it may happen due to overfitting. Due to the small quantity of data we have, an ensemble does not seem to be the best option. It learns in a very optimal way the training set but it has huge errors in the testing set, so at the end, it is not an useful model to predict.

As a conclusion, we have to admit that the results are very good taking into account the size of the data set, which only has 63 weeks and the quickness that the situation is changing. For instance, everybody knows that thanks to the vaccination process to old people, deaths will decrease drastically and also since the people are getting used to this pandemic time, the confirmed cases also goes down. However, those variables are not in our data set and could improve much more the predictions, but again, with only 63 weeks it is very difficult to get prefect predictions.

5 REFERENCES

- Wickham et al., (2019). Welcome to the tidyverse. Journal of Open Source Software, 4(43), 1686, https://doi.org/10.21105/joss.01686
- Kuhn et al., (2020). Tidymodels: a collection of packages for modeling and machine learning using tidyverse principles. https://www.tidymodels.org
- Guidotti, E., Ardia, D., (2020), "COVID-19 Data Hub," Journal of Open Source Software 5(51):2376.
- Steven Pawley (2021). recipeselectors: Extra Recipes Steps for Supervised Feature Selection. R package version 0.0.1. https://github.com/stevenpawley/recipeselectors
- Kuhn, Max, and Kjell Johnson. 2019. "Feature Selection Overview." In, 227–40. Chapman; Hall/CRC.