How to put an R model in production

To create a machine learning algorithm is cool but... what's next? How can I put my model (in this case made in R) in production?

In this blog, we have already learned some ways of squeezing out our R scripts, from automating them in Google Cloud or doing so in Windows or Mac. But sometimes our R codes are not auto sufficient: some other services might need to use them. But how?

To achieve that we usually build APIs. As I explained in this webinar at Deusto University (https://www.youtube.com/watch?v=sRQrxoTDsGw) (sorry it is in Spanish), an API is a way that enables two computing services to communicate.

Imagine that we have trained an algorithm that classifies flowers depending on four variables. The idea is that users will have a form on a website with those same variables as placeholders. The idea is simple: when they submit the form correctly, they are told which type of flower it is.

It sounds cool, right? Sure it does! So, let's learn how we can put our R models in production!

How to build an API with Plumber

Understanding the basic concepts of an API

Before we go in and see how to create an API, it is important to first understand some fundamental concepts of an API. If you saw the webinar on how to work with APIs in R or, you already know about APIs, you can skip this section.

As we have said before, an API allows two computing services to communicate. One service sends an order to the other. If the order has been sent correctly, this second service executes an action and returns a result.

Now, what types of actions or methods can be done? Well, there are mainly the following:

- **POST**: it allows us to create or send data.
- **GET**: it allows us to retrieve data.
- **PUT**: it allows us to update data.
- **DELETE**: it allows us to delete data.

In our case, our algorithm will receive data in order to predict, so we will use a POST method.

Besides, an API can have several endpoints, that is, different ways by which we can connect with the other service. For example, this Football API (link (https://www.api-football.com)) for example, offers us info about seasons at this endpoint (link (https://www.api-football.com/documentation#seasons)), but also team info at this other endpoint (link (https://www.api-football.com/documentation#teams)).

Each endpoint has a different URL and it might (or not) require different parameters.

Lastly, an API will most likely have some required and some optional parameters. As the name suggests, the required parameters are just that, required: if we don't pass the required parameters, the API will return an error.

In our case, the required parameters will be the data that we need for our model: the height and width of the petal and the height and width of the sepal.

On the other hand, the optional parameters or filters usually serve to make our request easier. For example, in the football API leagues endpoint, the optional parameters allow us to filter the data by a team, by country, or even a year to obtain the data of the leagues that interest us.

Parameter	Type	Required	Description
league_id	integer	false	Fails if field contains anything other than an integer
team_id	integer	false	Fails if field contains anything other than an integer
search	string	false	3 characters minimum | Fails if field has anything other than alphabetic characters
country	string	false	Fails if field contains anything other than alpha-numeric characters, underscores or dashes

Football Leagues API parameters

current	string	false	Fails if field has anything other than alphabetic characters
type	string	false	Fails if field has anything other than alphabetic characters

Now that you already know what an API is, let's learn how to build an API to put our R machine learning model in production.

How to put an R model in production

Example of an R model to upload to production

Before putting a model into production, in order to make the example is as close to reality as possible, we are going to train a model. It will be a simple classification model, based on the classic iris dataset.

To do so, I will train a classification model with `RandomForest`.

```
model = randomForest(Species ~ ., data = iris)
model
```

```
                Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 4.67%
Confusion matrix:
           setosa versicolor virginica class.error
setosa         50          0         0        0.00
versicolor      0         47         3        0.06
virginica       0          4        46        0.08
```

Lastly, we need to save the model so that in production we don't need to train anything, but rather just load the model and make the prediction. This will make the process much faster.

```
save(model, file = "API/model.RData")
```

We already have our model trained and saved. But how can we put this R model into production? Let's see it!

How to create an API with Plumber

First of all is to convert our model into an API. For this we will use the Plumber package. How do you create an API? Well, you just have to add embellished comments, like the following, to a script:

```
#* @apiTitle API Title
#* @param parameter_example
#* @get /endpoint_name
```

First of all, `@apiTitle` is the name of our API. On the other hand, with `@param` we can pass parameters to our API. In this case, there is only a single parameter, called the test parameter, but we could add more.

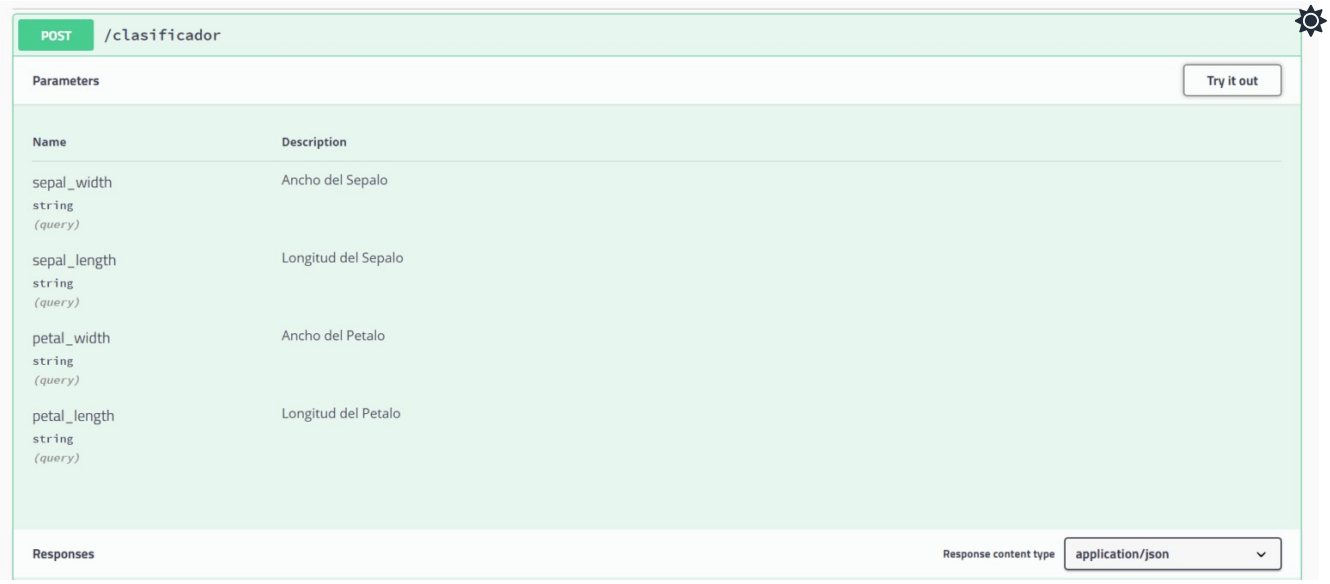
Although these parameters would be enough, we could configure our API even more. For example, with the following line we could include an optional filter:

```
#* @filter filter_example
```

Once we have this, we simply have to create a function that uses those input parameters as arguments. What this function returns will be the response of our API.

Let's see how we can build the API for our classification problem:

```
library(plumber)
library(randomForest)
#* @apiTitle API de clasificacion de flores
#* @param petal_length Longitud del Petalo
#* @param petal_width Ancho del Petalo
#* @param sepal_length Longitud del Sepalo
#* @param sepal_width Ancho del Sepalo
#* @post /clasificador

function(petal_length, petal_width, sepal_length, sepal_width){

  load(&quot;model.RData&quot;)

  test = c(sepal_length, sepal_width, petal_length, petal_width)
  test = sapply(test, as.numeric)
  test = data.frame(matrix(test, ncol = 4))

  colnames(test) = colnames(iris[,1:4])
  predict(model, test)

}
```



Help on making the API call.

Now, if we click on ‘Try it out’, the parameters description will change to some placeholders. We have to fil those placeholders with their corresponding values. Once those fields are filled in, we can click on ‘Execute’. As a result, a request will be made with the parameters that we have included to our API.

If the API call is correct, we will get a 200 response and will get the answer:

Response of the API that we have build with Plumber to put our machine learning model made in R in production

We already have our API working! Now we just need to be able to upload this API to the Cloud and... we will have already learned how to put any R model into production!

Hosting an R API in Google Cloud

Google Cloud Setup

Before uploading our API to Google Cloud, AWS or any other site, we must create a docker image that contains this API. Although I already explained it in this post (https://anderfernandez.com/en/blog/how-to-create-a-virtual-machine-with-r-on-google-cloud/) in more detail, a docker image allows you to run the code inside of it on any server with docker.

This is something very interesting, since this way it is not necessary for that server to have R and the packages that we use installed, since it will simply be installed when you run docker itself.

The process of uploading the API to Google Cloud will be done from R. But first we need to:

1. Create a service account.
2. Grant access for: Cloud Functions developer, Compute Admin, project editor, Service Account User, Cloud Run Admin and Storage objects Admin.
3. Download the accound key in JSON file.
4. Create a Bucker at Cloud Storage. You can do it on this url (https://console.cloud.google.com/storage/create-bucket).

If you don't know how to exactly follow this steps, check out this post (https://anderfernandez.com/en/blog/how-to-create-a-virtual-machine-with-r-on-google-cloud/) where I explained it step by step.

Once we have done this, we will have to create the following environment variables in our R session. **It is important that we define the variables before loading the library, otherwise, it will not work**.

```
bucket = "trexamplebucket"

Sys.setenv(
  GCE_AUTH_FILE = account_key,
  GCE_DEFAULT_PROJECT_ID = project,
  CR_REGION = region,
  GCS_DEFAULT_BUCKET = bucket,
  PORT = 8080
)


library(googleCloudRunner)
```

In this way, the googleCloudRunner library already has access to our service account. Now, we must create the dockerfile to upload it to our Google account.

Now, let's create our docker image!

Build our API on Docker

To create a docker image, we can use the `containerit` package. However, even if we are based on `containerit`, we will need to make some modifications to the dockerfile it generates.

So in my case, I have created the dockerfile myself. You will see that it is very simple. At the end of the day, they are nothing more than sequences that you must be executing.

```
# Copy model and script
RUN mkdir /data
COPY model.RData /data
COPY api.R /data
WORKDIR /data

# Plumb & run server
EXPOSE 8080
ENTRYPOINT ["R", "-e", \
    "pr <- plumber::plumb('/data/api.R'); pr$run(host='0.0.0.0', port=8080
```

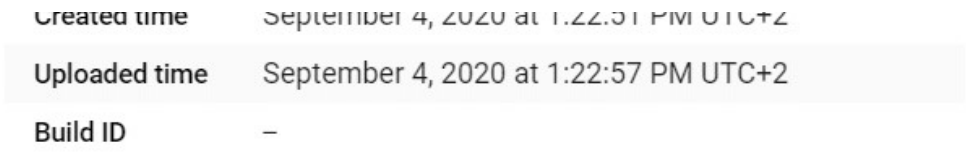
Lastly, we are going to use both the R file and the dockerfile to publish our API. To do this, first we are going to change the directory to the folder we have created. This way we make sure that only the api.R and Dockerfile files are uploaded to the bucket.

```
setwd(paste0(getwd(),"/API"))
cr <- cr_deploy_plumber(getwd())
```

Basically, this code uploads the files to the bucket we have chosen. From there, it mounts the docker image in Container Registry and finally, deploys the image in Cloud Run.

It may be that the previous step gives you an error when mounting the docker image. If that is the case, you should go to Container Registry and publish the image from there, as in the image.

How to publish a Docker image from Container Registry



Once done, it will redirect you to Cloud Run. In case everything has been correct, a green dot will appear and you will be able to see the URL of your API, as below.

Checl that the deployment of the API on Cloud Run is OK

With this, we would already have our API published! In my case, I can access the API at this URL: https://api-rk6gh2l6da-uc.a.run.app (https://api-rk6gh2l6da-uc.a.run.app). Thus, we already have our machine learning model made in R “exposed” so that any other service that needs it can obtain a prediction through a call to the API that we have created.

Check that the R model has been put into production correctly

Finally, we are going to check that our API works correctly and it returns the prediction of our algorithm. To do this, we are going to make a POST request to the API, to see what it returns:

```
library(httr)
library(jsonlite)

url = "https://api-rk6gh2l6da-ew.a.run.app/clasificador?sepal_width=2&sepa

response = POST(url)

fromJSON(content(response, type = "text", encoding = "utf-8"))
```

it works. We have uploaded our Random Forest model made in R to Google Cloud and we see that if we pass it the appropriate parameters, it works correctly!

Now, we only miss to see our machine learning model work in a real production example.... So let's do it!

Example of how a machine learning model works in R in production

To demonstrate an example of how our R model would work in production, I have created a small HTML form. This form has 4 fields, where the width and length of both the petal and the sepal must be indicated.

Thus, once the form is submitted, it is verified that the fields have been filled in correctly and, if so, a POST request is made to the API that we have created.

In addition, we show the response that it returns in an API through an alert, thus indicating the type of flower that corresponds to those sizes that we have indicated.

Petal Width:

Petal Length:

Sepal Width:

Sepal Length:

And this is just a very simple model with a very simple example! But, can you imagine the potential of being able to put your machine learning model into production? And it is that, if we do not put our models (in this case of R), in production ... what are they useful for beyond obtaining analysis on our computer?

In short, I hope you have found it interesting to learn how to put an R model into production. For any suggestions, do not hesitate to write to me on Linkedin or by answering the form on the page. In fact, if I have written this post it is because a reader asked me! So you know, whatever, feel free to contact. See you next time!