

MÁSTER EN INGENIERÍA INFORMÁTICA

Cloud Computing: Servicios y Aplicaciones

Practica 03 :  BigData y procesamiento de datos en Cloud

**Autores**

Hamada Bouhacida

177339003 bouhcidahamada@correo.ugr.es

hamadabouhcida34@gmail.com



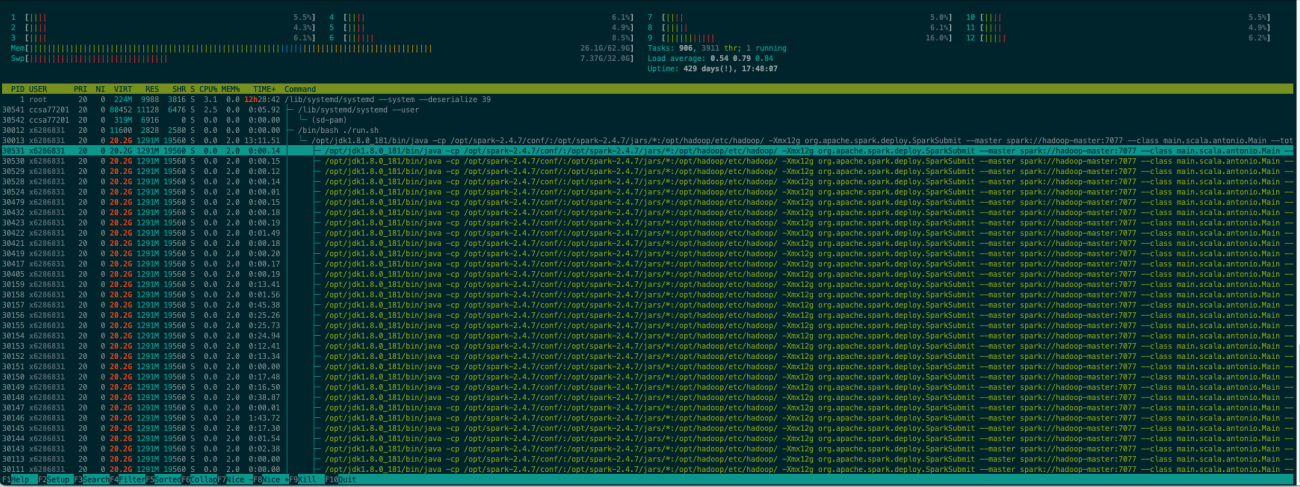
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

-------------

Granada,Junio de 2022

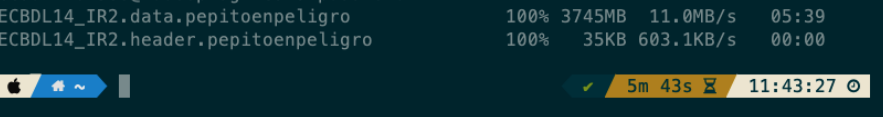
**Introduction :**

During the development of this document, and therefore, of the resolution of the proposed practice, we will try to solve classification problems using computational techniques based on **Big Data**. The framework used is **Spark** using the deep learning library **MLLib** and using the **python wrapper: pyspark.**



On a personal note, I know that we have been given access to a distributed computing cluster, known online as **hadoop.ugr.es**, which has the **HDFS** distributed file system configuration, but I have given up using it, because it is impossible to run any process there.

Therefore, we have only used that cluster to capture the dataset and we have worked locally and later, in our own cloud environment with the provider.



Download both files

At the beginning, we have implemented a composition of services that can be found in the annex and in the delivery, which consists of three Spark containers with the configuration of the Sevillian company Bitnami, in order to carry out execution tests before assembling the system. cloud that we describe in the next chapter. Just keep in mind that you have to install the numpy package before using it:

docker exec -it --user root hamadabouhcida\_spark\_1 pip install numpy

docker exec -it hamadabouhcida \_spark\_1 spark-submit --master spark://spark:707

**Resolution :**

**Infrastructure deployment :**

For the deployment of the infrastructure we have used the **aws cli** to generate the Elastic resource

**Map Reduce** with three instances of type m5.xlarge, two slave and one master.

1 aws emr create-cluster --applications Name=Spark Name=Zeppelin

,→ --ec2-attributes

,→ '{"KeyName":"spark","InstanceProfile":"EMR\_EC2\_DefaultRole",

2 "SubnetId":"subnet-731b7d19",

3 "EmrManagedSlaveSecurityGroup":"sg-0a2bf65c779ae46d3",

4 "EmrManagedMasterSecurityGroup":"sg-0f45869b481f32b74"}'

5

6 --service-role EMR\_DefaultRole --enable-debugging --release-label

,→ emr-6.3.0 --log-uri 's3n://bucket-pepitoenpeligro/' --name

,→ 'hamadaCluster' --instance-groups

,→ '[{"InstanceCount":1,"EbsConfiguration":{"EbsBlockDeviceConfigs":

7 [{"VolumeSpecification":{"SizeInGB":32,"VolumeType":"gp2"},

8 "VolumesPerInstance":2}]},

9

,→ "InstanceGroupType":"MASTER","InstanceType":"m5.xlarge","Name":"Master

,→ Instance Group"},{"InstanceCount":2,"EbsConfiguration"

10 :{"EbsBlockDeviceConfigs":

11 [{"VolumeSpecification":

12 {"SizeInGB":32,"VolumeType":"gp2"},

13 "VolumesPerInstance":2}]},

14 "InstanceGroupType":"CORE","InstanceType":"m5.xlarge",

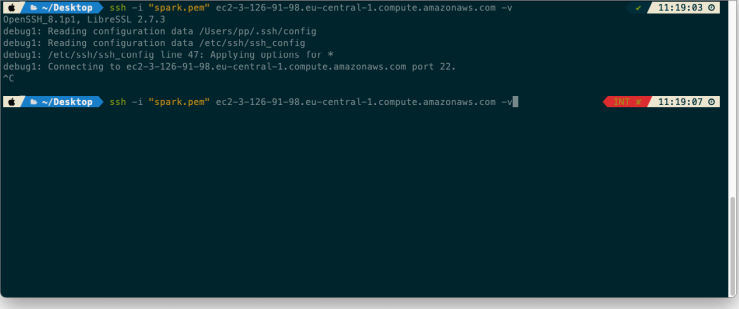
15 "Name":"Core Instance Group"}]'

16 --configurations '[{"Classification":"spark","Properties":{}}]'

,→ --scale-down-behavior TERMINATE\_AT\_TASK\_COMPLETION --region

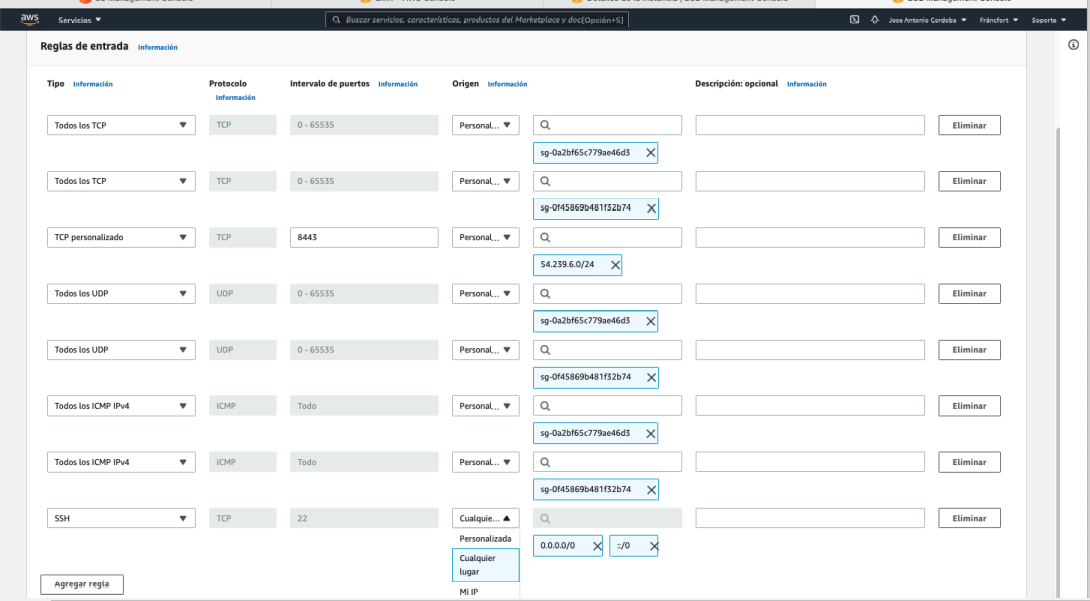
,→eu-central-1

We tried to connect via ssh to the master machine but we couldn't because port 22 by default is only available within the **AWS** **VPC**.



Connecting to the cluster master machine. Access denied

We have adapted the ingress rules of the master machine to be able to ssh in from outside the **AWS VPC**. For simplicity and speed, we have gone to do it through the web client of the console.



Add an inbound rule to allow access to port 22 from any IP

**Get Columns :**

To obtain the columns that interest us, we have uploaded the header and data files to an AWS S3 bucket and then we have applied a column extraction process.

Next, we take advantage of the cloud environment to directly read from the previously defined bucket and extract the columns that have been assigned to us:

• PredCN\_central\_2

• PredSS\_r1\_4

• PSSM\_r1\_3\_T

• AA\_freq\_central\_M

• PSSM\_r2\_-1\_L

• PSSM\_r1\_-2\_R

To do this we have modeled the following python script that summarizes creating a spark execution context, reading the header and the data file from the bucker, performing the map operation and reducing with the data itself later, selecting the columns that we are interested in and export the dataframe to a bucket.

1 import sys

2 import time

3 from pyspark import SparkContext, SparkConf, sql

4 from pyspark.ml.classification import LogisticRegression

5 from functools import reduce

6

7 configurationSpark = SparkConf().setAppName("CC-P4-Preprocesado")

8 sparkContexto = SparkContext.getOrCreate(conf=configurationSpark)

9 sqlContext = sql.SQLContext(sparkContexto)

10 name\_output="hamadabouhcida"

11

12 if \_\_name\_\_ == "\_\_main\_\_":

13 start = time.time()

14 print("Comenzando el preprocesado")

15 ficheroCabeceras = sparkContexto.textFile("

16 s3n://bucket-pepitoenpeligro/raw\_data/ECBDL14\_IR2.header").collect()

17 cabecerasFiltradas = filter(lambda line: "@attribute" in line

,→ ,ficheroCabeceras)

18 print("Mapeando")

19 mapHeaders = list(map(lambda line: line.split()[1],

,→ cabecerasFiltradas))

20

21 print("Leyendo en un dataframe los datos del dataset pesado")

22 df =

,→ sqlContext.read.csv("s3://bucket-pepitoenpeligro/raw\_data/ECBDL14\_IR2.data",

23 header=False,sep=",",inferSchema=True)

24 print("Reduciendo")

25 dfReducido = reduce(lambda data, idx:

,→data.withColumnRenamed(df.schema.names[idx], mapHeaders[idx]),

,→range(len(df.schema.names)), df)

26

27 dfReducido.createOrReplaceTempView("sql\_dataset")

28

29 columns= ['`PredCN\_central\_2`', '`PredSS\_r1\_4`', '`PSSM\_r1\_3\_T`',

,→ '`AA\_freq\_central\_M`', '`PSSM\_r2\_-1\_L`', '`PSSM\_r1\_-2\_R`']

30 print("Seleccionando las columnas {%s, %s, %s, %s, %s, %s}" %

,→(columns[0], columns[1], columns[2], columns[3], columns[4],

,→columns[5]))

31 sqlDF = sqlContext.sql('SELECT %s, %s, %s, %s, %s, %s, class FROM

,→sql\_dataset' % (columns[0], columns[1], columns[2], columns[3],

,→columns[4], columns[5]))

32

33 print("Escribiendo en el fichero csv")

34 sqlDF.write.format('csv').option('header',True)

35 .save('s3n://bucket-pepitoenpeligro/%s' % (name\_output))

36

37 print("Fin del preprocesado")

38 print("[3] Hemos Seleccionando las columnas {%s, %s, %s, %s, %s, %s}" %

,→(columns[0], columns[1], columns[2], columns[3], columns[4],

,→columns[5]))

39 end = time.time()

40 print("Tiempo consumido en la seleccion de columnas: %s" % (end –

,→ start))

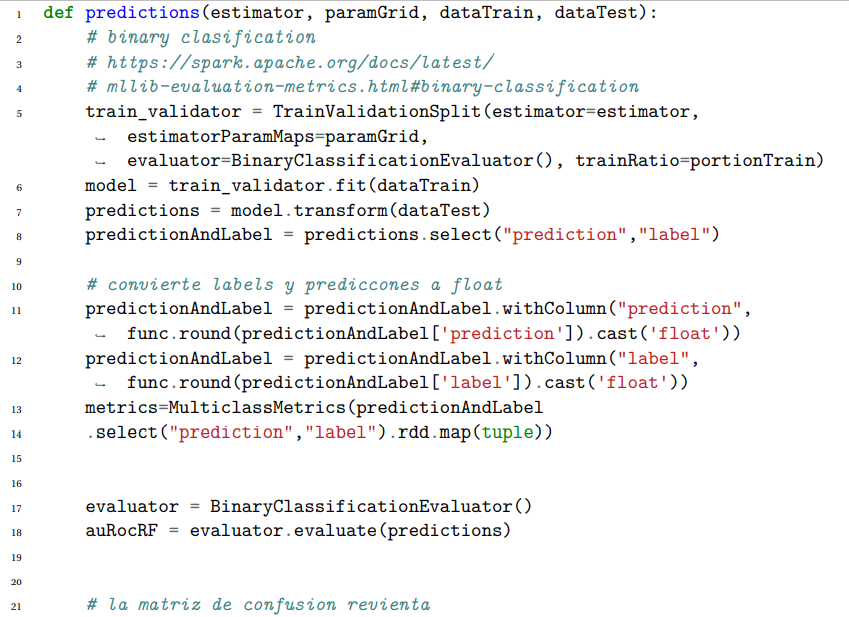
41 sparkContexto.stop()

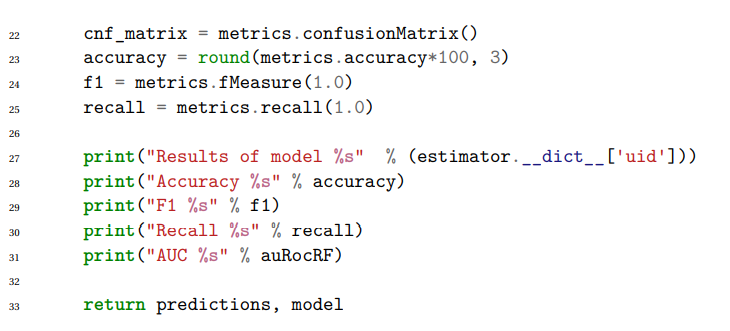
**Models :**

We define a proportion of 80% training and 20% testing. We use all the rows of the dataset. For each model we measure the time it takes to train and evaluate so that we can make a comparison later.

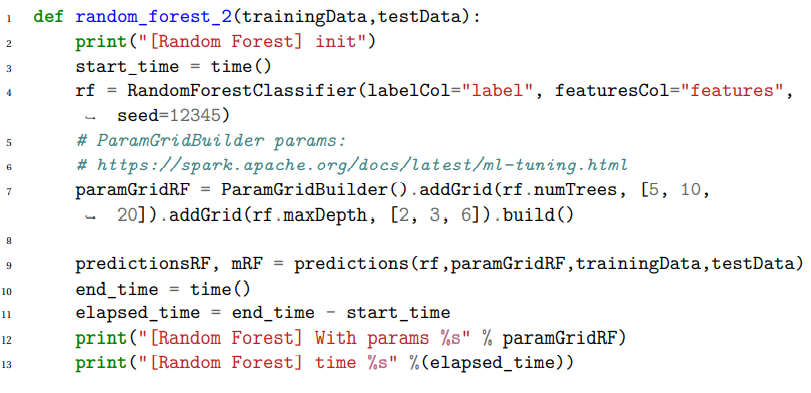
**Evaluation of the models:**

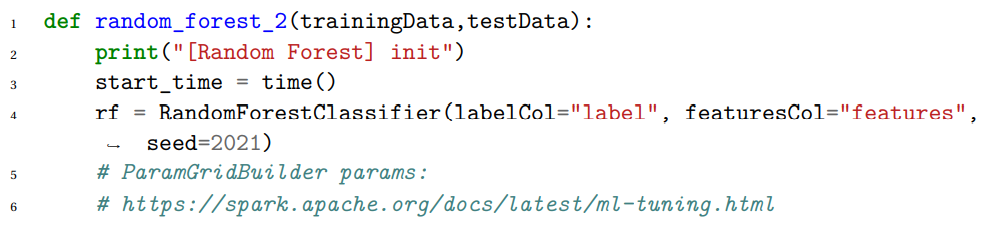
For the evaluation of the models, we have defined a function that needs the model, the parameter grid, the training set and the test set. Inside we fit the model to the training set and get the predictions and get the **precision**, **f1**, **auc** and **recall** values.

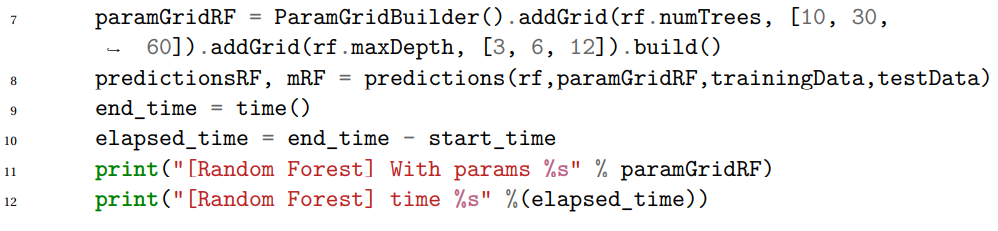




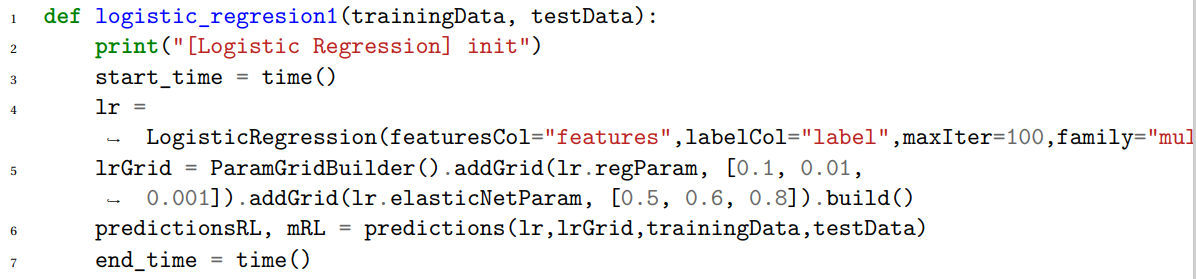
**Random Forest :**

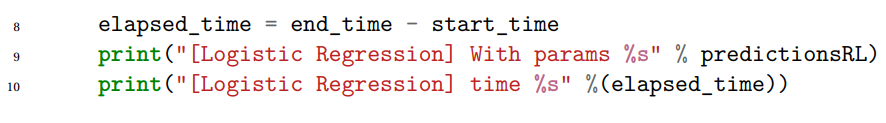


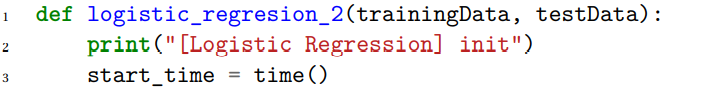


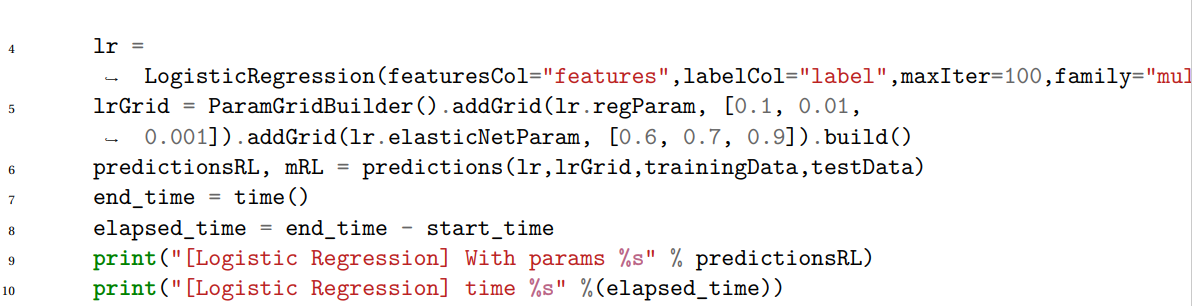


Logistic Regression :









**Comparison of the models :**

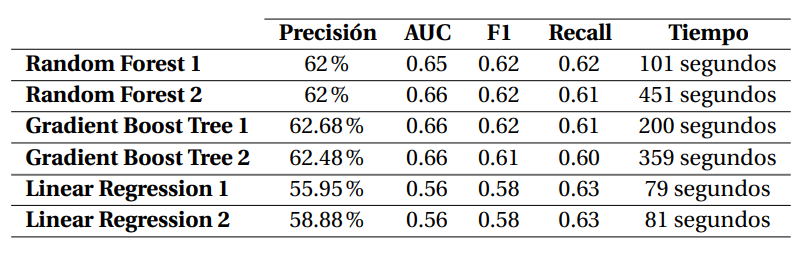
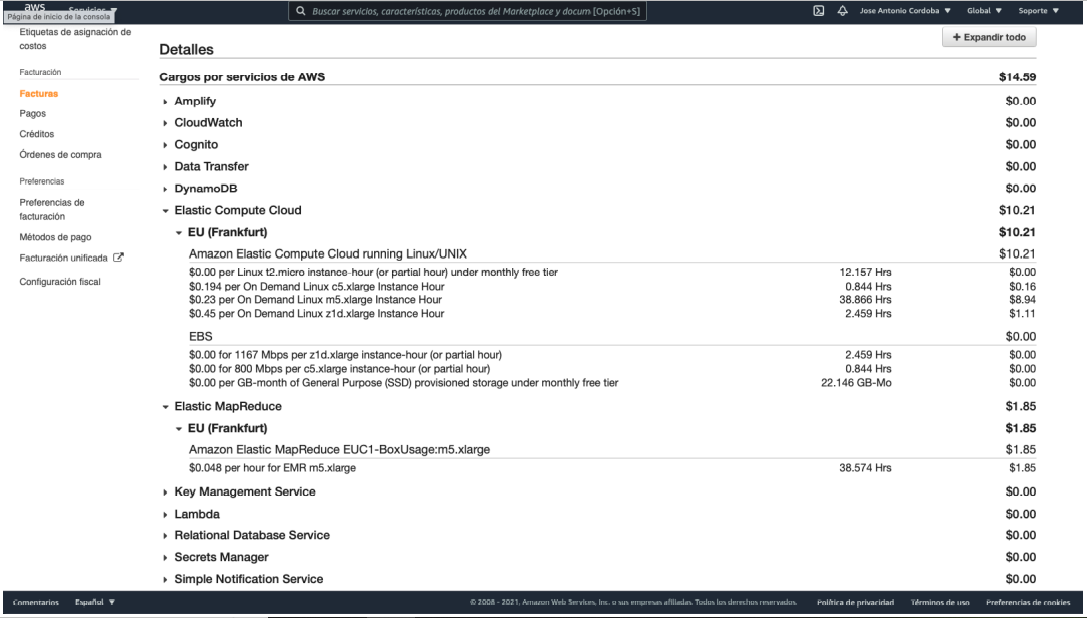


Table 2.1: As we can see, of all the parameter variations that we have made in all the models, the only one that has found improvement is the linear regression. Comparing the training times and their trade-off with the accuracy and the area under the ROC curve, we can find that the smartest solution would be to use either the RandomForest or the Gradient Boost Tree with the simplest parameters.

**Invoice :**



**Conclusions :**

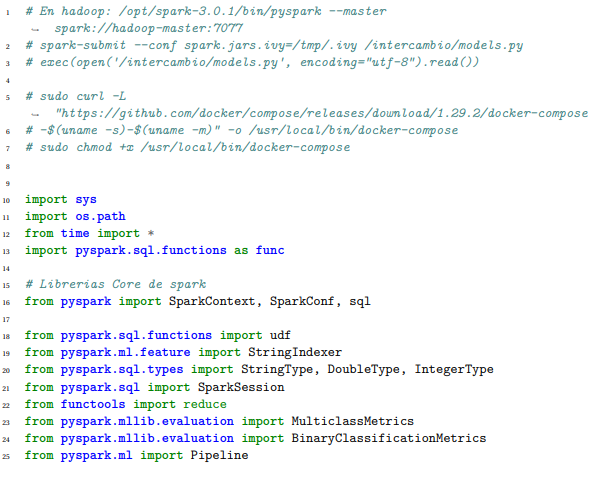
After doing this work, we can summarize the following questions:

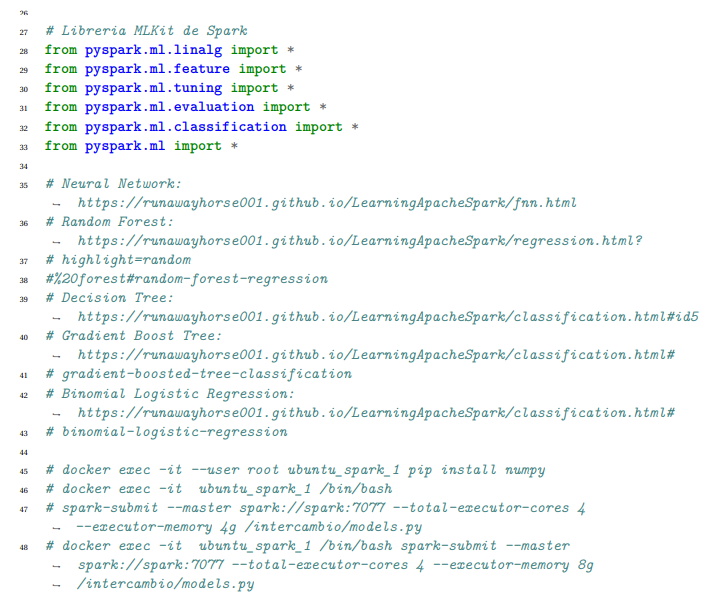
• Deploying a cloud environment for development with Big Data techniques under a major provider like AWS is highly recommended, simple and cheap.

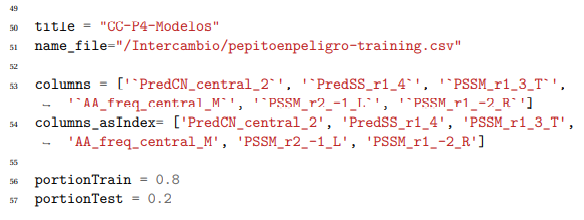
• I find the Big Data technique tools with Spark extremely fast and easy to use, for huge volumes of data. In our case, the training and evaluation of the heaviest model has consumed 451 seconds, a time that if we had invested a local standard machine, with the same set of data, it would not have taken those 451 seconds, not even close to the time consumed would have been much higher.

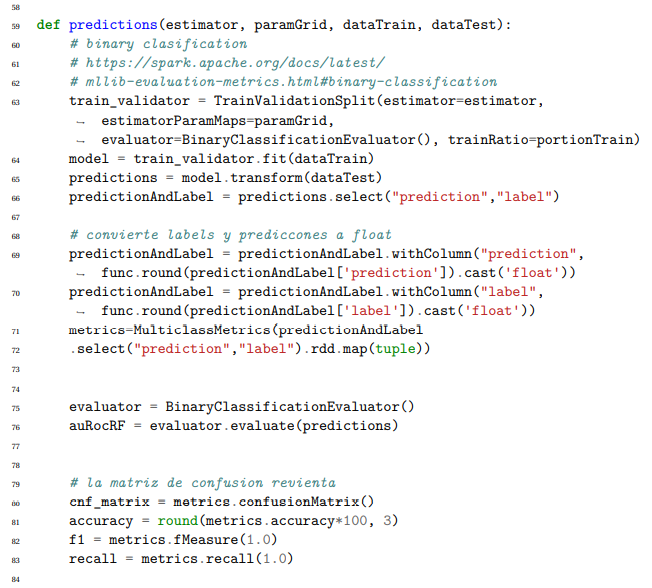
• I would have liked to have had another planning in the master to be able to deepen with Scala, or to have had the opportunity to have done some performance evaluation by increasing the number of slave nodes. I have not done the latter so as not to increase the bill, but I would have found it highly recommended to know what the gain in performance can be from scaling a Big Data system horizontally. I'll have a chance to do it in the summer.

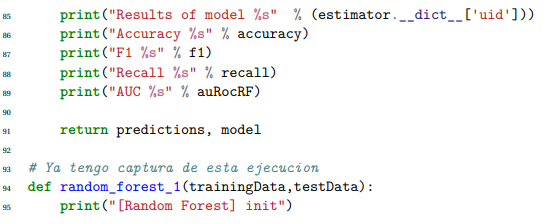
**Complete model code :**

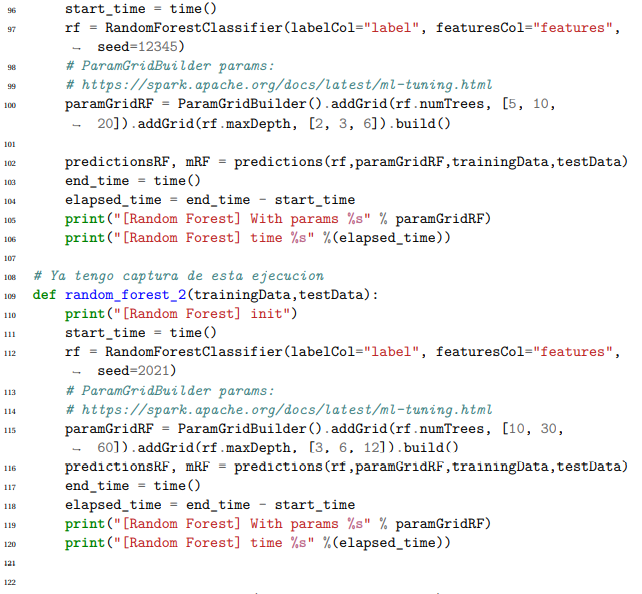


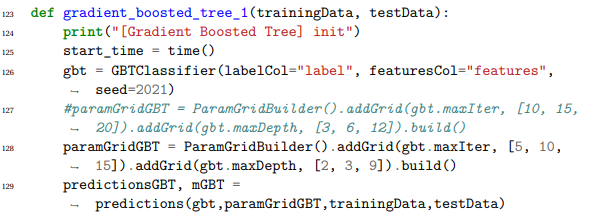




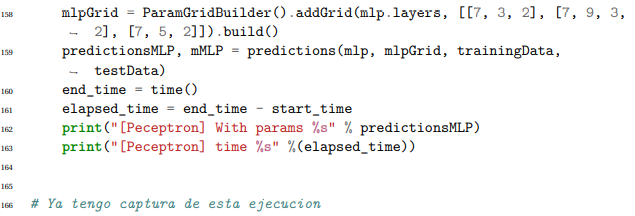


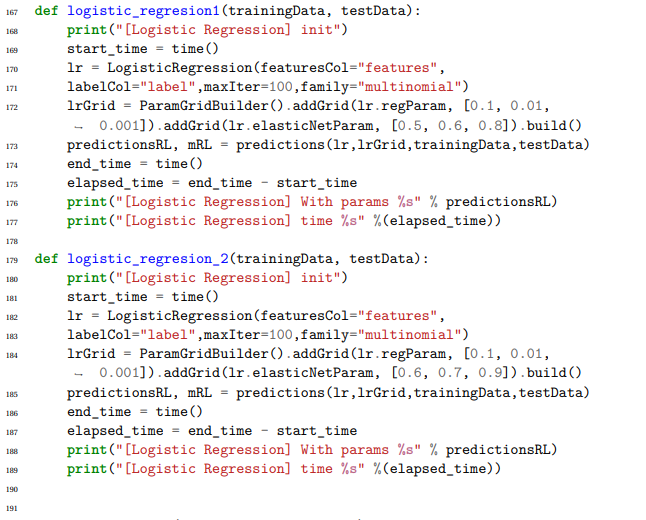


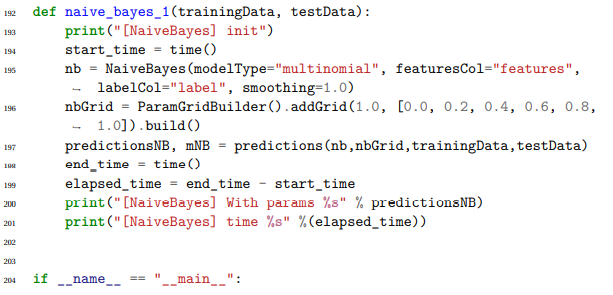


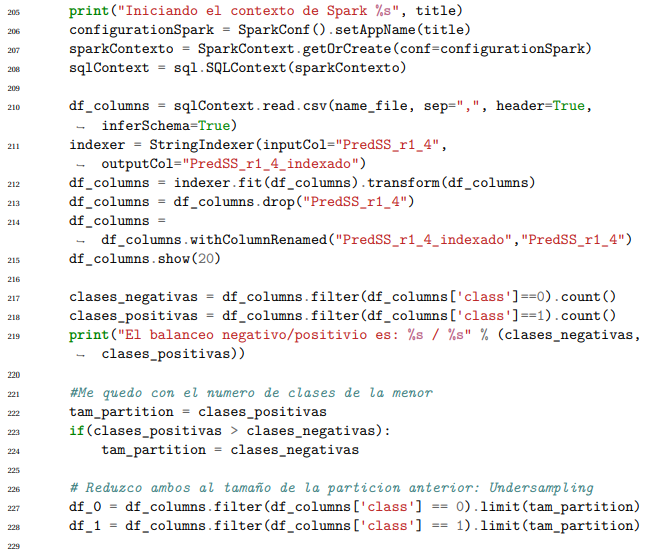


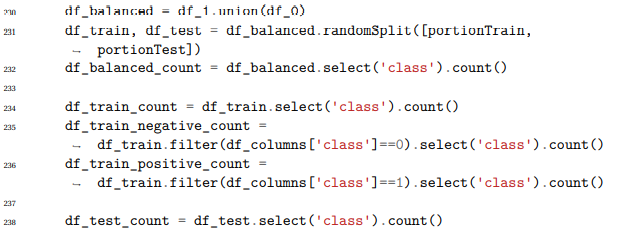


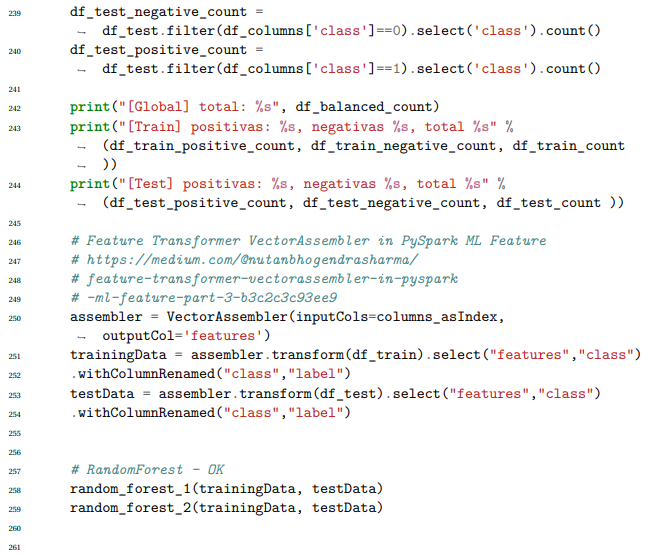


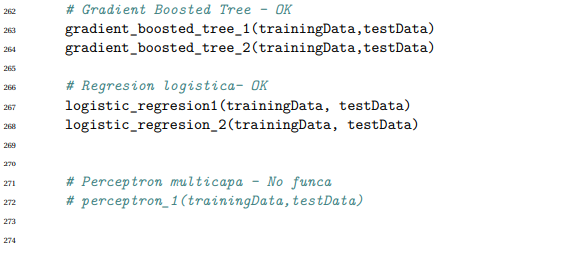


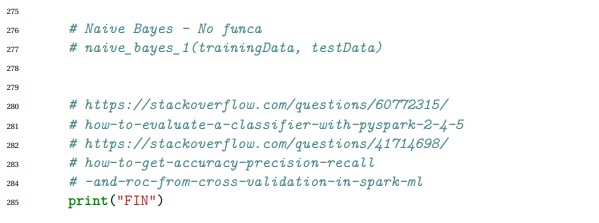






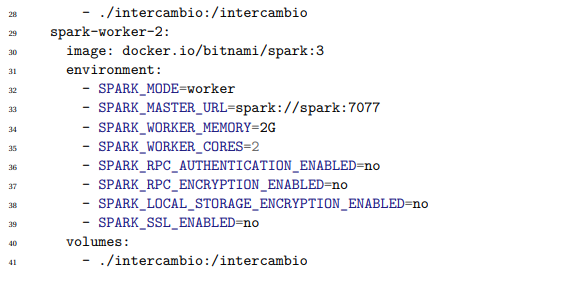






**Spark composition:**





**Bibliography :**

[1] Feature Transformer VectorAssembler in PySpark ML Feature https://medium.com/@nut anbhogendrasharma/feature-transformer-vectorassembler-in-pyspark-ml-fe ature-part-3-b3c2c3c93ee9

[2] How to get Accuracy precision recall and ROC https://stackoverflow.com/questions/ 41714698/how-to-get-accuracy-precision-recall-and-roc-from-cross-valid ation-in-spark-ml

[3] Random Forest https://runawayhorse001.github.io/LearningApacheSpark/regres sion.html?highlight=random%20forest#random-forest-regression

[4] Gradient Boost Tree https://runawayhorse001.github.io/LearningApacheSpark/cl assification.html#id5

[5] Logistic Regression https://runawayhorse001.github.io/LearningApacheSpark/cl assification.html#binomial-logistic-regression

[6] AWS CLI S3 [https://docs.aws.amazon.com/es\_es/cli/latest/userguide/cli-ser vices-s3-commands.html](https://docs.aws.amazon.com/es_es/cli/latest/userguide/cli-ser%20vices-s3-commands.html)

[7] AWS EMR <https://aws.amazon.com/es/emr/>