



ugr

Universidad
de Granada

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

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**.

```

1 [1] 3.93 6 [1] 6.53 7 [1] 5.93 10 [1] 5.93
2 [1] 6.93 8 [1] 6.93 8 [1] 5.93 11 [1] 5.93
3 [1] 6.93 9 [1] 8.93 9 [1] 10.93 12 [1] 6.93
4 [1] 10.93 13 [1] 10.93 14 [1] 10.93 15 [1] 10.93
5 [1] 10.93 16 [1] 10.93 17 [1] 10.93 18 [1] 10.93
6 [1] 10.93 19 [1] 10.93 20 [1] 10.93 21 [1] 10.93
7 [1] 10.93 22 [1] 10.93 23 [1] 10.93 24 [1] 10.93
8 [1] 10.93 25 [1] 10.93 26 [1] 10.93 27 [1] 10.93
9 [1] 10.93 28 [1] 10.93 29 [1] 10.93 30 [1] 10.93
10 [1] 10.93 31 [1] 10.93 32 [1] 10.93 33 [1] 10.93
11 [1] 10.93 34 [1] 10.93 35 [1] 10.93 36 [1] 10.93
12 [1] 10.93 37 [1] 10.93 38 [1] 10.93 39 [1] 10.93
13 [1] 10.93 40 [1] 10.93 41 [1] 10.93 42 [1] 10.93
14 [1] 10.93 43 [1] 10.93 44 [1] 10.93 45 [1] 10.93
15 [1] 10.93 46 [1] 10.93 47 [1] 10.93 48 [1] 10.93
16 [1] 10.93 49 [1] 10.93 50 [1] 10.93 51 [1] 10.93
17 [1] 10.93 52 [1] 10.93 53 [1] 10.93 54 [1] 10.93
18 [1] 10.93 55 [1] 10.93 56 [1] 10.93 57 [1] 10.93
19 [1] 10.93 58 [1] 10.93 59 [1] 10.93 60 [1] 10.93
20 [1] 10.93 61 [1] 10.93 62 [1] 10.93 63 [1] 10.93
21 [1] 10.93 64 [1] 10.93 65 [1] 10.93 66 [1] 10.93
22 [1] 10.93 67 [1] 10.93 68 [1] 10.93 69 [1] 10.93
23 [1] 10.93 70 [1] 10.93 71 [1] 10.93 72 [1] 10.93
24 [1] 10.93 73 [1] 10.93 74 [1] 10.93 75 [1] 10.93
25 [1] 10.93 76 [1] 10.93 77 [1] 10.93 78 [1] 10.93
26 [1] 10.93 79 [1] 10.93 80 [1] 10.93 81 [1] 10.93
27 [1] 10.93 82 [1] 10.93 83 [1] 10.93 84 [1] 10.93
28 [1] 10.93 85 [1] 10.93 86 [1] 10.93 87 [1] 10.93
29 [1] 10.93 88 [1] 10.93 89 [1] 10.93 90 [1] 10.93
30 [1] 10.93 91 [1] 10.93 92 [1] 10.93 93 [1] 10.93
31 [1] 10.93 94 [1] 10.93 95 [1] 10.93 96 [1] 10.93
32 [1] 10.93 97 [1] 10.93 98 [1] 10.93 99 [1] 10.93
33 [1] 10.93 100 [1] 10.93 101 [1] 10.93 102 [1] 10.93
34 [1] 10.93 103 [1] 10.93 104 [1] 10.93 105 [1] 10.93
35 [1] 10.93 106 [1] 10.93 107 [1] 10.93 108 [1] 10.93
36 [1] 10.93 109 [1] 10.93 110 [1] 10.93 111 [1] 10.93
37 [1] 10.93 112 [1] 10.93 113 [1] 10.93 114 [1] 10.93
38 [1] 10.93 115 [1] 10.93 116 [1] 10.93 117 [1] 10.93
39 [1] 10.93 118 [1] 10.93 119 [1] 10.93 120 [1] 10.93
40 [1] 10.93 121 [1] 10.93 122 [1] 10.93 123 [1] 10.93
41 [1] 10.93 124 [1] 10.93 125 [1] 10.93 126 [1] 10.93
42 [1] 10.93 127 [1] 10.93 128 [1] 10.93 129 [1] 10.93
43 [1] 10.93 130 [1] 10.93 131 [1] 10.93 132 [1] 10.93
44 [1] 10.93 133 [1] 10.93 134 [1] 10.93 135 [1] 10.93
45 [1] 10.93 136 [1] 10.93 137 [1] 10.93 138 [1] 10.93
46 [1] 10.93 139 [1] 10.93 140 [1] 10.93 141 [1] 10.93
47 [1] 10.93 142 [1] 10.93 143 [1] 10.93 144 [1] 10.93
48 [1] 10.93 145 [1] 10.93 146 [1] 10.93 147 [1] 10.93
49 [1] 10.93 148 [1] 10.93 149 [1] 10.93 150 [1] 10.93
50 [1] 10.93 151 [1] 10.93 152 [1] 10.93 153 [1] 10.93
51 [1] 10.93 154 [1] 10.93 155 [1] 10.93 156 [1] 10.93
52 [1] 10.93 157 [1] 10.93 158 [1] 10.93 159 [1] 10.93
53 [1] 10.93 160 [1] 10.93 161 [1] 10.93 162 [1] 10.93
54 [1] 10.93 163 [1] 10.93 164 [1] 10.93 165 [1] 10.93
55 [1] 10.93 166 [1] 10.93 167 [1] 10.93 168 [1] 10.93
56 [1] 10.93 169 [1] 10.93 170 [1] 10.93 171 [1] 10.93
57 [1] 10.93 172 [1] 10.93 173 [1] 10.93 174 [1] 10.93
58 [1] 10.93 175 [1] 10.93 176 [1] 10.93 177 [1] 10.93
59 [1] 10.93 178 [1] 10.93 179 [1] 10.93 180 [1] 10.93
60 [1] 10.93 181 [1] 10.93 182 [1] 10.93 183 [1] 10.93
61 [1] 10.93 184 [1] 10.93 185 [1] 10.93 186 [1] 10.93
62 [1] 10.93 187 [1] 10.93 188 [1] 10.93 189 [1] 10.93
63 [1] 10.93 190 [1] 10.93 191 [1] 10.93 192 [1] 10.93
64 [1] 10.93 193 [1] 10.93 194 [1] 10.93 195 [1] 10.93
65 [1] 10.93 196 [1] 10.93 197 [1] 10.93 198 [1] 10.93
66 [1] 10.93 199 [1] 10.93 200 [1] 10.93 201 [1] 10.93
67 [1] 10.93 202 [1] 10.93 203 [1] 10.93 204 [1] 10.93
68 [1] 10.93 205 [1] 10.93 206 [1] 10.93 207 [1] 10.93
69 [1] 10.93 208 [1] 10.93 209 [1] 10.93 210 [1] 10.93
70 [1] 10.93 211 [1] 10.93 212 [1] 10.93 213 [1] 10.93
71 [1] 10.93 214 [1] 10.93 215 [1] 10.93 216 [1] 10.93
72 [1] 10.93 217 [1] 10.93 218 [1] 10.93 219 [1] 10.93
73 [1] 10.93 220 [1] 10.93 221 [1] 10.93 222 [1] 10.93
74 [1] 10.93 223 [1] 10.93 224 [1] 10.93 225 [1] 10.93
75 [1] 10.93 226 [1] 10.93 227 [1] 10.93 228 [1] 10.93
76 [1] 10.93 229 [1] 10.93 230 [1] 10.93 231 [1] 10.93
77 [1] 10.93 232 [1] 10.93 233 [1] 10.93 234 [1] 10.93
78 [1] 10.93 235 [1] 10.93 236 [1] 10.93 237 [1] 10.93
79 [1] 10.93 238 [1] 10.93 239 [1] 10.93 240 [1] 10.93
80 [1] 10.93 241 [1] 10.93 242 [1] 10.93 243 [1] 10.93
81 [1] 10.93 244 [1] 10.93 245 [1] 10.93 246 [1] 10.93
82 [1] 10.93 247 [1] 10.93 248 [1] 10.93 249 [1] 10.93
83 [1] 10.93 250 [1] 10.93 251 [1] 1
```

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 `aws` provider.

```
ECBDL14_IR2.data.pepitoenpeligro 100% 3745MB 11.0MB/s 05:39
ECBDL14_IR2.header.pepitoenpeligro 100% 35KB 603.1KB/s 00:00
```

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
```

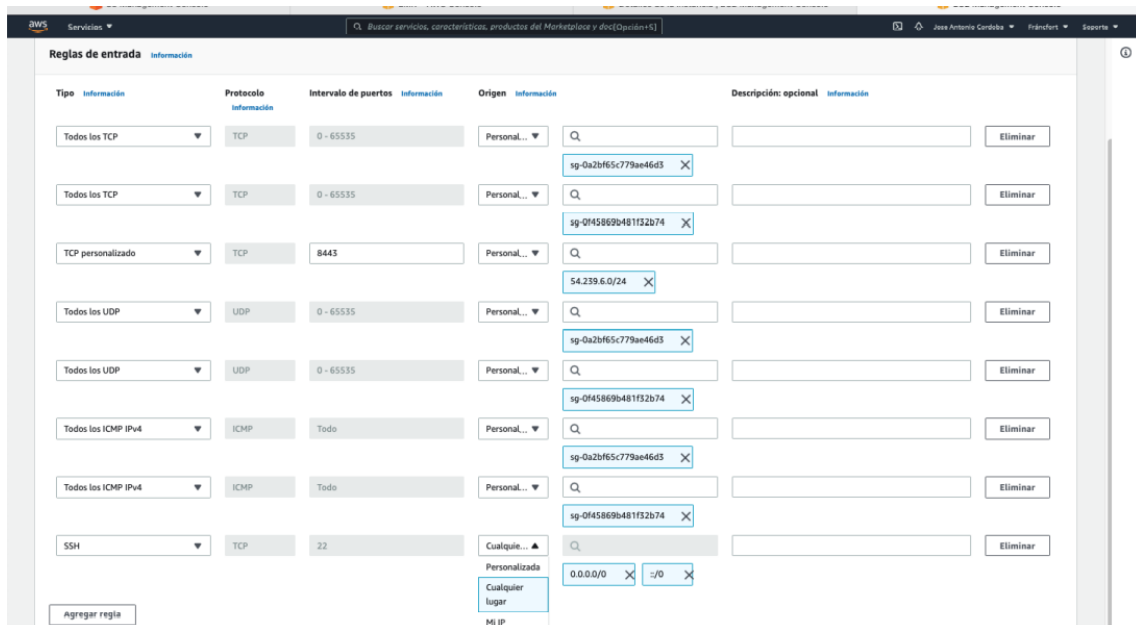
We tried to connect via ssh to the master machine but we couldn't because port 22 by default is only available within the **AWS VPC**.

```
~/Desktop ssh -i "spark.pem" ec2-3-126-91-98.eu-central-1.compute.amazonaws.com ~v
OpenSSH_8.1p1, LibreSSL 2.7.3
debug1: Reading configuration data /Users/pp/.ssh/config
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: /etc/ssh/ssh_config line 47: Applying options for *
debug1: Connecting to ec2-3-126-91-98.eu-central-1.compute.amazonaws.com port 22.
^C

~/Desktop ssh -i "spark.pem" ec2-3-126-91-98.eu-central-1.compute.amazonaws.com ~v
```

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 bucket, 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
11 name_output="hamadabouhcida"
12
13 if __name__ == "__main__":
14     start = time.time()
15     print("Comenzando el preprocesado")
16     ficheroCabeceras = sparkContexto.textFile("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 {%, %, %, %, %, %}" %
    ,→(columns[0], columns[1], columns[2], columns[3], columns[4],
    ,→columns[5]))
31 sqlDF = sqlContext.sql('SELECT %, %, %, %, %, %, 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 {%, %, %, %, %, %}" %

```

```

        ,→(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.

```

1  def predictions(estimator, paramGrid, dataTrain, dataTest):
2      # binary clasification
3      # https://spark.apache.org/docs/latest/
4      # mllib-evaluation-metrics.html#binary-classification
5      train_validator = TrainValidationSplit(estimator=estimator,
        ~ estimatorParamMaps=paramGrid,
        ~ evaluator=BinaryClassificationEvaluator(), trainRatio=portionTrain)
6      model = train_validator.fit(dataTrain)
7      predictions = model.transform(dataTest)
8      predictionAndLabel = predictions.select("prediction","label")
9
10     # convierte labels y predicciones a float
11     predictionAndLabel = predictionAndLabel.withColumn("prediction",
        ~ func.round(predictionAndLabel['prediction']).cast('float'))
12     predictionAndLabel = predictionAndLabel.withColumn("label",
        ~ func.round(predictionAndLabel['label']).cast('float'))
13     metrics=MulticlassMetrics(predictionAndLabel
14     .select("prediction","label").rdd.map(tuple))
15
16
17     evaluator = BinaryClassificationEvaluator()
18     auRocRF = evaluator.evaluate(predictions)
19
20
21     # la matriz de confusion revienta

```

```

22     cnf_matrix = metrics.confusionMatrix()
23     accuracy = round(metrics.accuracy*100, 3)
24     f1 = metrics.fMeasure(1.0)
25     recall = metrics.recall(1.0)
26
27     print("Results of model %s" % (estimator.__dict__['uid']))
28     print("Accuracy %s" % accuracy)
29     print("F1 %s" % f1)
30     print("Recall %s" % recall)
31     print("AUC %s" % auRocRF)
32
33     return predictions, model

```

Random Forest :

```

1  def random_forest_2(trainingData,testData):
2      print("[Random Forest] init")
3      start_time = time()
4      rf = RandomForestClassifier(labelCol="label", featuresCol="features",
5          ↪ seed=12345)
6      # ParamGridBuilder params:
7      # https://spark.apache.org/docs/latest/ml-tuning.html
8      paramGridRF = ParamGridBuilder().addGrid(rf.numTrees, [5, 10,
9          ↪ 20]).addGrid(rf.maxDepth, [2, 3, 6]).build()
10
11     predictionsRF, mRF = predictions(rf,paramGridRF,trainingData,testData)
12     end_time = time()
13     elapsed_time = end_time - start_time
14     print("[Random Forest] With params %s" % paramGridRF)
15     print("[Random Forest] time %s" %(elapsed_time))

```



```

1 def random_forest_2(trainingData, testData):
2     print("[Random Forest] init")
3     start_time = time()
4     rf = RandomForestClassifier(labelCol="label", featuresCol="features",
5                               ↪ seed=2021)
6     # ParamGridBuilder params:
7     # https://spark.apache.org/docs/latest/ml-tuning.html
8
9     paramGridRF = ParamGridBuilder().addGrid(rf.numTrees, [10, 30,
10    ↪ 60]).addGrid(rf.maxDepth, [3, 6, 12]).build()
11     predictionsRF, mRF = predictions(rf, paramGridRF, trainingData, testData)
12     end_time = time()
13     elapsed_time = end_time - start_time
14     print("[Random Forest] With params %s" % paramGridRF)
15     print("[Random Forest] time %s" %(elapsed_time))

```

Logistic Regression :

```

1 def logistic_regression1(trainingData, testData):
2     print("[Logistic Regression] init")
3     start_time = time()
4     lr =
5     ↪ LogisticRegression(featuresCol="features", labelCol="label", maxIter=100, family="mul
6     lrGrid = ParamGridBuilder().addGrid(lr.regParam, [0.1, 0.01,
7     ↪ 0.001]).addGrid(lr.elasticNetParam, [0.5, 0.6, 0.8]).build()
8     predictionsRL, mRL = predictions(lr, lrGrid, trainingData, testData)
9     end_time = time()
10
11     elapsed_time = end_time - start_time
12     print("[Logistic Regression] With params %s" % predictionsRL)
13     print("[Logistic Regression] time %s" %(elapsed_time))

```

```

1  def logistic_regression_2(trainingData, testData):
2      print("[Logistic Regression] init")
3      start_time = time()

4      lr =
5          LogisticRegression(featuresCol="features",labelCol="label",maxIter=100,family="mul
6      lrGrid = ParamGridBuilder().addGrid(lr.regParam, [0.1, 0.01,
7          0.001]).addGrid(lr.elasticNetParam, [0.6, 0.7, 0.9]).build()
8      predictionsRL, mRL = predictions(lr,lrGrid,trainingData,testData)
9      end_time = time()
10     elapsed_time = end_time - start_time
11     print("[Logistic Regression] With params %s" % predictionsRL)
12     print("[Logistic Regression] time %s" %(elapsed_time))

```

Comparison of the models :

	Precisión	AUC	F1	Recall	Tiempo
Random Forest 1	62%	0.65	0.62	0.62	101 segundos
Random Forest 2	62%	0.66	0.62	0.61	451 segundos
Gradient Boost Tree 1	62.68%	0.66	0.62	0.61	200 segundos
Gradient Boost Tree 2	62.48%	0.66	0.61	0.60	359 segundos
Linear Regression 1	55.95%	0.56	0.58	0.63	79 segundos
Linear Regression 2	58.88%	0.56	0.58	0.63	81 segundos

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 :

AWS <small>Consolidación</small>		Buscar servicios, características, productos del Marketplace y docum [Opción+5]		Jose Antonio Cordoba Global Soporte	
Etiquetas de asignación de costos		+ Expandir todo			
Facturación		Detalles			
Facturas		Cargos por servicios de AWS		\$14.59	
Pagos		» Amplify		\$0.00	
Créditos		» CloudWatch		\$0.00	
Órdenes de compra		» Cognito		\$0.00	
Preferencias		» Data Transfer		\$0.00	
Preferencias de facturación		» DynamoDB		\$0.00	
Métodos de pago		» Elastic Compute Cloud		\$10.21	
Facturación unificada ⓘ		» EU (Frankfurt)		\$10.21	
Configuración fiscal		Amazon Elastic Compute Cloud running Linux/UNIX		\$10.21	
		\$0.00 per Linux t2.micro instance-hour (or partial hour) under monthly free tier		12.157 Hrs	\$0.00
		\$0.194 per On Demand Linux c5.xlarge Instance Hour		0.844 Hrs	\$0.16
		\$0.23 per On Demand Linux m5.xlarge Instance Hour		38.866 Hrs	\$8.94
		\$0.45 per On Demand Linux z1d.xlarge Instance Hour		2.459 Hrs	\$1.11
		EBS		\$0.00	
		\$0.00 for 1167 Mbps per z1d.xlarge instance-hour (or partial hour)		2.459 Hrs	\$0.00
		\$0.00 for 800 Mbps per c5.xlarge instance-hour (or partial hour)		0.844 Hrs	\$0.00
		\$0.00 per GB-month of General Purpose (SSD) provisioned storage under monthly free tier		22.146 GB-Mo	\$0.00
		» Elastic MapReduce		\$1.85	
		» EU (Frankfurt)		\$1.85	
		Amazon Elastic MapReduce EUC1-BoxUsage:m5.xlarge		\$1.85	
		\$0.048 per hour for EMR m5.xlarge		38.574 Hrs	\$1.85
		» Key Management Service		\$0.00	
		» Lambda		\$0.00	
		» Relational Database Service		\$0.00	
		» Secrets Manager		\$0.00	
		» Simple Notification Service		\$0.00	
Comentarios		Español		© 2008 - 2021 Amazon Web Services, Inc. o sus empresas afiliadas. Todos los derechos reservados. Política de privacidad Términos de uso Preferencias de cookies	

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 :

```
1 # En hadoop: /opt/spark-3.0.1/bin/pyspark --master
   ~ spark://hadoop-master:7077
2 # spark-submit --conf spark.jars.ivy=/tmp/.ivy /intercambio/models.py
3 # exec(open('/intercambio/models.py', encoding="utf-8").read())
4
5 # sudo curl -L
   ~ "https://github.com/docker/compose/releases/download/1.29.2/docker-compose
6 # -$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
7 # sudo chmod +x /usr/local/bin/docker-compose
8
9
10 import sys
11 import os.path
12 from time import *
13 import pyspark.sql.functions as func
14
15 # Librerias Core de spark
16 from pyspark import SparkContext, SparkConf, sql
17
18 from pyspark.sql.functions import udf
19 from pyspark.ml.feature import StringIndexer
20 from pyspark.sql.types import StringType, DoubleType, IntegerType
21 from pyspark.sql import SparkSession
22 from functools import reduce
23 from pyspark.mllib.evaluation import MulticlassMetrics
24 from pyspark.mllib.evaluation import BinaryClassificationMetrics
25 from pyspark.ml import Pipeline
```

```

26
27 # Libreria MLKit de Spark
28 from pyspark.ml.linalg import *
29 from pyspark.ml.feature import *
30 from pyspark.ml.tuning import *
31 from pyspark.ml.evaluation import *
32 from pyspark.ml.classification import *
33 from pyspark.ml import *
34
35 # Neural Network:
36   - https://runawayhorse001.github.io/LearningApacheSpark/fnn.html
37 # Random Forest:
38   - https://runawayhorse001.github.io/LearningApacheSpark/regression.html?
39   highlight=random
40   #forest#random-forest-regression
41 # Decision Tree:
42   - https://runawayhorse001.github.io/LearningApacheSpark/classification.html?id5
43 # Gradient Boost Tree:
44   - https://runawayhorse001.github.io/LearningApacheSpark/classification.html#
45   gradient-boosted-tree-classification
46 # Binomial Logistic Regression:
47   - https://runawayhorse001.github.io/LearningApacheSpark/classification.html#
48   binomial-logistic-regression
49
50 # docker exec -it --user root ubuntu_spark_1 pip install numpy
51 # docker exec -it ubuntu_spark_1 /bin/bash
52 # spark-submit --master spark://spark:7077 --total-executor-cores 4
53   - --executor-memory 4g /intercambio/models.py
54 # docker exec -it ubuntu_spark_1 /bin/bash spark-submit --master
55   - spark://spark:7077 --total-executor-cores 4 --executor-memory 8g
56   - /intercambio/models.py
57
58
59
60 title = "CC-P4-Modelos"
61 name_file="/Intercambio/pepitoenpeligro-training.csv"
62
63 columns = ['PredCN_central_2', 'PredSS_r1_4', 'PSSM_r1_3_T',
64   - 'AA_freq_central_M', 'PSSM_r2_1_L', 'PSSM_r1_2_R']
65 columns_asIndex= ['PredCN_central_2', 'PredSS_r1_4', 'PSSM_r1_3_T',
66   - 'AA_freq_central_M', 'PSSM_r2_1_L', 'PSSM_r1_2_R']
67
68
69
70 portionTrain = 0.8
71 portionTest = 0.2

```

```

58
59 def predictions(estimator, paramGrid, dataTrain, dataTest):
60     # binary clasification
61     # https://spark.apache.org/docs/latest/
62     # mllib-evaluation-metrics.html#binary-classification
63     train_validator = TrainValidationSplit(estimator=estimator,
64     ~ estimatorParamMaps=paramGrid,
65     ~ evaluator=BinaryClassificationEvaluator(), trainRatio=portionTrain)
66     model = train_validator.fit(dataTrain)
67     predictions = model.transform(dataTest)
68     predictionAndLabel = predictions.select("prediction", "label")
69
70     # convierte labels y predicciones a float
71     predictionAndLabel = predictionAndLabel.withColumn("prediction",
72     ~ func.round(predictionAndLabel['prediction']).cast('float'))
73     predictionAndLabel = predictionAndLabel.withColumn("label",
74     ~ func.round(predictionAndLabel['label']).cast('float'))
75     metrics=MulticlassMetrics(predictionAndLabel
76     .select("prediction", "label").rdd.map(tuple))
77
78
79     evaluator = BinaryClassificationEvaluator()
80     auRocRF = evaluator.evaluate(predictions)
81
82
83     # la matriz de confusion revienta
84     cnf_matrix = metrics.confusionMatrix()
85     accuracy = round(metrics.accuracy*100, 3)
86     f1 = metrics.fMeasure(1.0)
87     recall = metrics.recall(1.0)
88
89
90
91     print("Results of model %s" % (estimator.__dict__['uid']))
92     print("Accuracy %s" % accuracy)
93     print("F1 %s" % f1)
94     print("Recall %s" % recall)
95     print("AUC %s" % auRocRF)
96
97     return predictions, model
98
99 # Ya tengo captura de esta ejecucion
100 def random_forest_1(trainingData, testData):
101     print("[Random Forest] init")

```

```

96     start_time = time()
97     rf = RandomForestClassifier(labelCol="label", featuresCol="features",
98         ~ seed=12345)
99     # ParamGridBuilder params:
100     # https://spark.apache.org/docs/latest/ml-tuning.html
101     paramGridRF = ParamGridBuilder().addGrid(rf.numTrees, [5, 10,
102         ~ 20]).addGrid(rf.maxDepth, [2, 3, 6]).build()
103
104     predictionsRF, mRF = predictions(rf,paramGridRF,trainingData,testData)
105     end_time = time()
106     elapsed_time = end_time - start_time
107     print("[Random Forest] With params %s" % paramGridRF)
108     print("[Random Forest] time %s" %(elapsed_time))
109
110     # Ya tengo captura de esta ejecucion
111     def random_forest_2(trainingData,testData):
112         print("[Random Forest] init")
113         start_time = time()
114         rf = RandomForestClassifier(labelCol="label", featuresCol="features",
115             ~ seed=2021)
116         # ParamGridBuilder params:
117         # https://spark.apache.org/docs/latest/ml-tuning.html
118         paramGridRF = ParamGridBuilder().addGrid(rf.numTrees, [10, 30,
119             ~ 60]).addGrid(rf.maxDepth, [3, 6, 12]).build()
120         predictionsRF, mRF = predictions(rf,paramGridRF,trainingData,testData)
121         end_time = time()
122         elapsed_time = end_time - start_time
123         print("[Random Forest] With params %s" % paramGridRF)
124         print("[Random Forest] time %s" %(elapsed_time))
125
126     def gradient_boosted_tree_1(trainingData, testData):
127         print("[Gradient Boosted Tree] init")
128         start_time = time()
129         gbt = GBTClassifier(labelCol="label", featuresCol="features",
130             ~ seed=2021)
131         #paramGridGBT = ParamGridBuilder().addGrid(gbt.maxIter, [10, 15,
132             ~ 20]).addGrid(gbt.maxDepth, [3, 6, 12]).build()
133         paramGridGBT = ParamGridBuilder().addGrid(gbt.maxIter, [5, 10,
134             ~ 15]).addGrid(gbt.maxDepth, [2, 3, 9]).build()
135         predictionsGBT, mGBT =
136             ~ predictions(gbt,paramGridGBT,trainingData,testData)

```



```

130     end_time = time()
131     elapsed_time = end_time - start_time
132     print("[Gradient Boosted Tree] With params %s" % paramGridGBT)
133     print("[Gradient Boosted Tree] time %s" %(elapsed_time))
134
135
136 def gradient_boosted_tree_2(trainingData, testData):
137     print("[Gradient Boosted Tree] init")
138     start_time = time()
139     gbt = GBTClassifier(labelCol="label", featuresCol="features",
140         ~ seed=2021)
141     paramGridGBT = ParamGridBuilder().addGrid(gbt.maxIter, [10, 15,
142         ~ 20]).addGrid(gbt.maxDepth, [3, 6, 12]).build()
143
144     predictionsGBT, mGBT =
145         ~ predictions(gbt,paramGridGBT,trainingData,testData)
146     end_time = time()
147     elapsed_time = end_time - start_time
148     print("[Gradient Boosted Tree] With params %s" % paramGridGBT)
149     print("[Gradient Boosted Tree] time %s" %(elapsed_time))
150
151
152 def perceptron_1(trainingData, testData):
153     print("[Peceptron] init")
154     start_time = time()
155     mlp = MultilayerPerceptronClassifier(
156         featuresCol="features",
157         labelCol="label",
158         predictionCol="prediction",
159         maxIter=100
160     )
161
162     mlpGrid = ParamGridBuilder().addGrid(mlp.layers, [[7, 3, 2], [7, 9, 3,
163         ~ 2], [7, 5, 2]]).build()
164     predictionsMLP, mMLP = predictions(mlp, mlpGrid, trainingData,
165         ~ testData)
166     end_time = time()
167     elapsed_time = end_time - start_time
168     print("[Peceptron] With params %s" % predictionsMLP)
169     print("[Peceptron] time %s" %(elapsed_time))
170
171
172 # Ya tengo captura de esta ejecucion

```



```

167 def logistic_regression1(trainingData, testData):
168     print("[Logistic Regression] init")
169     start_time = time()
170     lr = LogisticRegression(featuresCol="features",
171                             labelCol="label",maxIter=100,family="multinomial")
172     lrGrid = ParamGridBuilder().addGrid(lr.regParam, [0.1, 0.01,
173                                                         ~ 0.001]).addGrid(lr.elasticNetParam, [0.5, 0.6, 0.8]).build()
174     predictionsRL, mRL = predictions(lr,lrGrid,trainingData,testData)
175     end_time = time()
176     elapsed_time = end_time - start_time
177     print("[Logistic Regression] With params %s" % predictionsRL)
178     print("[Logistic Regression] time %s" %(elapsed_time))
179
180 def logistic_regression_2(trainingData, testData):
181     print("[Logistic Regression] init")
182     start_time = time()
183     lr = LogisticRegression(featuresCol="features",
184                             labelCol="label",maxIter=100,family="multinomial")
185     lrGrid = ParamGridBuilder().addGrid(lr.regParam, [0.1, 0.01,
186                                                         ~ 0.001]).addGrid(lr.elasticNetParam, [0.6, 0.7, 0.9]).build()
187     predictionsRL, mRL = predictions(lr,lrGrid,trainingData,testData)
188     end_time = time()
189     elapsed_time = end_time - start_time
190     print("[Logistic Regression] With params %s" % predictionsRL)
191     print("[Logistic Regression] time %s" %(elapsed_time))
192
193 def naive_bayes_1(trainingData, testData):
194     print("[NaiveBayes] init")
195     start_time = time()
196     nb = NaiveBayes(modelType="multinomial", featuresCol="features",
197                     ~ labelCol="label", smoothing=1.0)
198     nbGrid = ParamGridBuilder().addGrid(1.0, [0.0, 0.2, 0.4, 0.6, 0.8,
199                                                         ~ 1.0]).build()
200     predictionsNB, mNB = predictions(nb,nbGrid,trainingData,testData)
201     end_time = time()
202     elapsed_time = end_time - start_time
203     print("[NaiveBayes] With params %s" % predictionsNB)
204     print("[NaiveBayes] time %s" %(elapsed_time))
205
206 if __name__ == "__main__":

```

```

205 print("Iniciando el contexto de Spark %s", title)
206 configurationSpark = SparkConf().setAppName(title)
207 sparkContexto = SparkContext.getOrCreate(conf=configurationSpark)
208 sqlContext = sql.SQLContext(sparkContexto)
209
210 df_columns = sqlContext.read.csv(name_file, sep=",", header=True,
    ~ inferSchema=True)
211 indexer = StringIndexer(inputCol="PredSS_r1_4",
    ~ outputCol="PredSS_r1_4_indexado")
212 df_columns = indexer.fit(df_columns).transform(df_columns)
213 df_columns = df_columns.drop("PredSS_r1_4")
214 df_columns =
    ~ df_columns.withColumnRenamed("PredSS_r1_4_indexado", "PredSS_r1_4")
215 df_columns.show(20)
216
217 clases_negativas = df_columns.filter(df_columns['class']==0).count()
218 clases_positivas = df_columns.filter(df_columns['class']==1).count()
219 print("El balanceo negativo/positivo es: %s / %s" % (clases_negativas,
    ~ clases_positivas))
220
221 #Me quedo con el numero de clases de la menor
222 tam_partition = clases_positivas
223 if(clases_positivas > clases_negativas):
224     tam_partition = clases_negativas
225
226 # Reduzco ambos al tamaño de la particion anterior: Undersampling
227 df_0 = df_columns.filter(df_columns['class'] == 0).limit(tam_partition)
228 df_1 = df_columns.filter(df_columns['class'] == 1).limit(tam_partition)
229
230 df_balanced = df_1.union(df_0)
231 df_train, df_test = df_balanced.randomSplit([portionTrain,
    ~ portionTest])
232 df_balanced_count = df_balanced.select('class').count()
233
234 df_train_count = df_train.select('class').count()
235 df_train_negative_count =
    ~ df_train.filter(df_columns['class']==0).select('class').count()
236 df_train_positive_count =
    ~ df_train.filter(df_columns['class']==1).select('class').count()
237
238 df_test_count = df_test.select('class').count()

```

```

239 df_test_negative_count =
    - df_test.filter(df_columns['class']==0).select('class').count()
240 df_test_positive_count =
    - df_test.filter(df_columns['class']==1).select('class').count()
241
242 print("[Global] total: %s", df_balanced_count)
243 print("[Train] positivas: %s, negativas %s, total %s" %
    - (df_train_positive_count, df_train_negative_count, df_train_count
    - ))
244 print("[Test] positivas: %s, negativas %s, total %s" %
    - (df_test_positive_count, df_test_negative_count, df_test_count ))
245
246 # Feature Transformer VectorAssembler in PySpark ML Feature
247 # https://medium.com/@nutanbhogendrasharma/
248 # feature-transformer-vectorassembler-in-pyspark
249 # -ml-feature-part-3-b3c2c3c93ee9
250 assembler = VectorAssembler(inputCols=columns_asIndex,
    - outputCol='features')
251 trainingData = assembler.transform(df_train).select("features","class")
252 .withColumnRenamed("class","label")
253 testData = assembler.transform(df_test).select("features","class")
254 .withColumnRenamed("class","label")
255
256
257 # RandomForest - OK
258 random_forest_1(trainingData, testData)
259 random_forest_2(trainingData, testData)
260
261
262 # Gradient Boosted Tree - OK
263 gradient_boosted_tree_1(trainingData, testData)
264 gradient_boosted_tree_2(trainingData, testData)
265
266 # Regresion logistica- OK
267 logistic_regresion1(trainingData, testData)
268 logistic_regresion_2(trainingData, testData)
269
270
271 # Perceptron multicapa - No funca
272 # perceptron_1(trainingData, testData)
273
274

```

```
275
276 # Naive Bayes - No funca
277 # naive_bayes_1(trainingData, testData)
278
279
280 # https://stackoverflow.com/questions/60772315/
281 # how-to-evaluate-a-classifier-with-pyspark-2-4-5
282 # https://stackoverflow.com/questions/41714698/
283 # how-to-get-accuracy-precision-recall
284 # -and-roc-from-cross-validation-in-spark-ml
285 print("FIN")
```

Spark composition:

```
1  version: '2'
2
3  services:
4    spark:
5      image: docker.io/bitnami/spark:3
6      environment:
7        - SPARK_MODE=master
8        - SPARK_RPC_AUTHENTICATION_ENABLED=no
9        - SPARK_RPC_ENCRYPTION_ENABLED=no
10       - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
11       - SPARK_SSL_ENABLED=no
12      ports:
13        - '8080:8080'
14      volumes:
15        - ./intercambio:/intercambio
16    spark-worker-1:
17      image: docker.io/bitnami/spark:3
18      environment:
19        - SPARK_MODE=worker
20        - SPARK_MASTER_URL=spark://spark:7077
21        - SPARK_WORKER_MEMORY=2G
22        - SPARK_WORKER_CORES=2
23        - SPARK_RPC_AUTHENTICATION_ENABLED=no
24        - SPARK_RPC_ENCRYPTION_ENABLED=no
25        - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
26        - SPARK_SSL_ENABLED=no
27      volumes:
28
29        - ./intercambio:/intercambio
29    spark-worker-2:
30      image: docker.io/bitnami/spark:3
31      environment:
32        - SPARK_MODE=worker
33        - SPARK_MASTER_URL=spark://spark:7077
34        - SPARK_WORKER_MEMORY=2G
35        - SPARK_WORKER_CORES=2
36        - SPARK_RPC_AUTHENTICATION_ENABLED=no
37        - SPARK_RPC_ENCRYPTION_ENABLED=no
38        - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
39        - SPARK_SSL_ENABLED=no
40      volumes:
41        - ./intercambio:/intercambio
```

Bibliography :

- [1] Feature Transformer VectorAssembler in PySpark ML Feature <https://medium.com/@nutanbhogendrasharma/feature-transformer-vectorassembler-in-pyspark-ml-feature-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-validation-in-spark-ml>
- [3] Random Forest <https://runawayhorse001.github.io/LearningApacheSpark/regression.html?highlight=random%20forest#random-forest-regression>
- [4] Gradient Boost Tree <https://runawayhorse001.github.io/LearningApacheSpark/classification.html#id5>
- [5] Logistic Regression <https://runawayhorse001.github.io/LearningApacheSpark/classification.html#binomial-logistic-regression>
- [6] AWS CLI S3 https://docs.aws.amazon.com/es_es/cli/latest/userguide/cli-services-s3-commands.html
- [7] AWS EMR <https://aws.amazon.com/es/emr/>