MÁSTER EN INGENIERÍA INFORMÁTICA

Cloud Computing: Servicios y Aplicaciones

Practica 03 : BigData y procesamiento de datos en Cloud

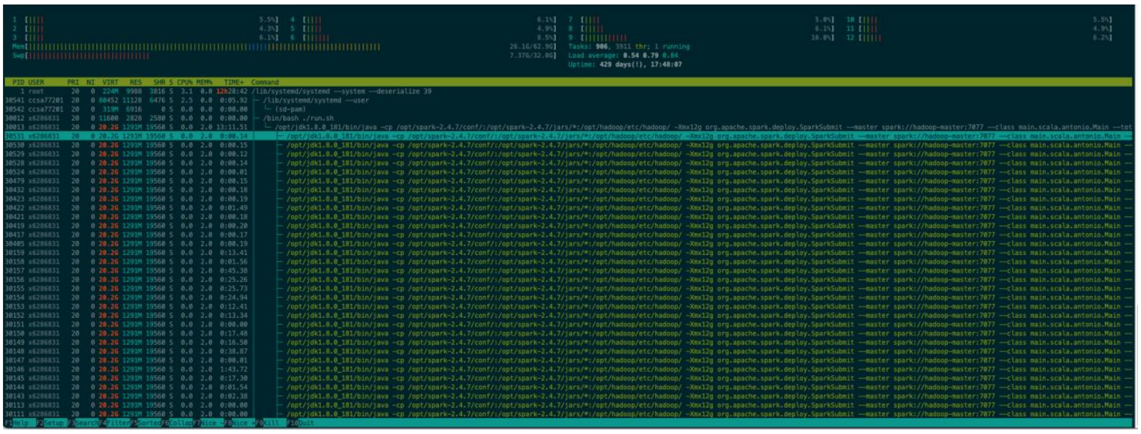
Autores

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 :

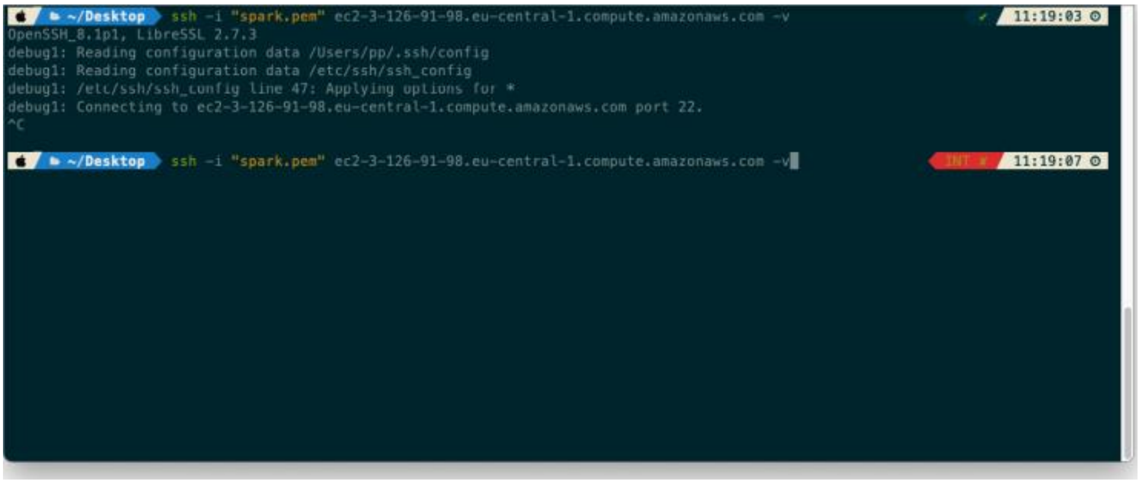
Infrastructure deployment :

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.

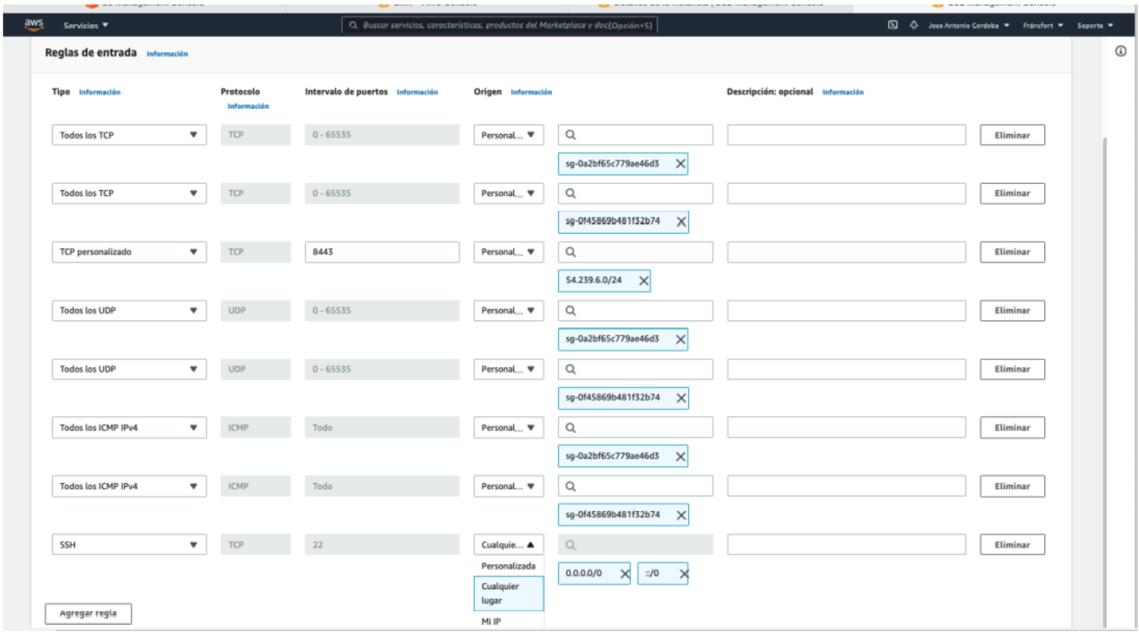
```
aws emr create-cluster --applications Name=Spark Name=Zeppelin
    ,→ --ec2-attributes
    ,→ '{"KeyName":"spark","InstanceProfile":"EMR_EC2_DefaultRole",
    "SubnetId":"subnet-731b7d19",
    "EmrManagedSlaveSecurityGroup":"sg-0a2bf65c779ae46d3",
    "EmrManagedMasterSecurityGroup":"sg-0f45869b481f32b74"}'

    --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":
        [{"VolumeSpecification":{"SizeInGB":32,"VolumeType":"gp2"},
        "VolumesPerInstance":2}]},

    ,→ "InstanceGroupType":"MASTER","InstanceType":"m5.xlarge","Name":"Master
    ,→ Instance Group"},{"InstanceCount":2,"EbsConfiguration"
            :{"EbsBlockDeviceConfigs":
                [{"VolumeSpecification":
                    {"SizeInGB":32,"VolumeType":"gp2"},
                    "VolumesPerInstance":2}]},
                    "InstanceGroupType":"CORE","InstanceType":"m5.xlarge",
                    "Name":"Core Instance Group"}]'
    --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.

```
import sys
import time
from pyspark import SparkContext, SparkConf, sql
from pyspark.ml.classification import LogisticRegression
from functools import reduce

configurationSpark = SparkConf().setAppName("CC-P4-Preprocesado")
sparkContexto = SparkContext.getOrCreate(conf=configurationSpark)
sqlContext = sql.SQLContext(sparkContexto)

name_output="hamadabouhcida"

if __name__ == "__main__":
    start = time.time()
    print("Comenzando el preprocesado")
    ficheroCabeceras = sparkContexto.textFile("
    s3n://bucket-pepitoenpeligro/raw_data/ECBDL14_IR2.header").collect()
    cabecerasFiltradas = filter(lambda line: "@attribute" in line
```

```
,→ ,ficheroCabeceras)
print("Mapeando")
mapHeaders = list(map(lambda line: line.split()[1],
 ,→ cabecerasFiltradas))

print("Leyendo en un dataframe los datos del dataset pesado")
df =
,→ sqlContext.read.csv("s3://bucket-pepitoenpeligro/raw_data/ECBDL14_IR2.data",
 header=False,sep=",",inferSchema=True)
 print("Reduciendo")
 dfReducido = reduce(lambda data, idx:
 ,→data.withColumnRenamed(df.schema.names[idx], mapHeaders[idx]),
 ,→range(len(df.schema.names)), df)

 dfReducido.createOrReplaceTempView("sql_dataset")

 columns= ['`PredCN_central_2`', '`PredSS_r1_4`', '`PSSM_r1_3_T`',
  ,→ '`AA_freq_central_M`', '`PSSM_r2_-1_L`', '`PSSM_r1_-2_R`']
 print("Seleccionando las columnas {%s, %s, %s, %s, %s, %s}" %
  ,→(columns[0], columns[1], columns[2], columns[3], columns[4],
  ,→columns[5]))
 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]))

  print("Escribiendo en el fichero csv")
  sqlDF.write.format('csv').option('header',True)
    .save('s3n://bucket-pepitoenpeligro/%s' % (name_output))

  print("Fin del preprocesado")
  print("[3] Hemos Seleccionando las columnas {%s, %s, %s, %s, %s, %s}" %
```

```
,→(columns[0], columns[1], columns[2], columns[3], columns[4],
,→columns[5]))
end = time.time()
print("Tiempo consumido en la seleccion de columnas: %s" % (end –
,→ start))
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.

```
def predictions(estimator, paramGrid, dataTrain, dataTest):
    # binary clasification
    # https://spark.apache.org/docs/latest/
    # mllib-evaluation-metrics.html#binary-classification
    train_validator = TrainValidationSplit(estimator=estimator,
     ↪ estimatorParamMaps=paramGrid,
     ↪ evaluator=BinaryClassificationEvaluator(), trainRatio=portionTrain)
    model = train_validator.fit(dataTrain)
    predictions = model.transform(dataTest)
    predictionAndLabel = predictions.select("prediction","label")

    # convierte labels y prediccones a float
    predictionAndLabel = predictionAndLabel.withColumn("prediction",
     ↪ func.round(predictionAndLabel['prediction']).cast('float'))
    predictionAndLabel = predictionAndLabel.withColumn("label",
     ↪ func.round(predictionAndLabel['label']).cast('float'))
    metrics=MulticlassMetrics(predictionAndLabel
    .select("prediction","label").rdd.map(tuple))


    evaluator = BinaryClassificationEvaluator()
    auRocRF = evaluator.evaluate(predictions)


    # la matriz de confusion revienta
```

```python
    cnf_matrix = metrics.confusionMatrix()
    accuracy = round(metrics.accuracy*100, 3)
    f1 = metrics.fMeasure(1.0)
    recall = metrics.recall(1.0)

    print("Results of model %s"  % (estimator.__dict__['uid']))
    print("Accuracy %s" % accuracy)
    print("F1 %s" % f1)
    print("Recall %s" % recall)
    print("AUC %s" % auRocRF)

    return predictions, model
```

Random Forest :

```python
def random_forest_2(trainingData,testData):
    print("[Random Forest] init")
    start_time = time()
    rf = RandomForestClassifier(labelCol="label", featuresCol="features",
     seed=12345)
    # ParamGridBuilder params:
    # https://spark.apache.org/docs/latest/ml-tuning.html
    paramGridRF = ParamGridBuilder().addGrid(rf.numTrees, [5, 10,
     20]).addGrid(rf.maxDepth, [2, 3, 6]).build()

    predictionsRF, mRF = predictions(rf,paramGridRF,trainingData,testData)
    end_time = time()
    elapsed_time = end_time - start_time
    print("[Random Forest] With params %s" % paramGridRF)
    print("[Random Forest] time %s" %(elapsed_time))
```

```python
def random_forest_2(trainingData,testData):
    print("[Random Forest] init")
    start_time = time()
    rf = RandomForestClassifier(labelCol="label", featuresCol="features", seed=2021)
    # ParamGridBuilder params:
    # https://spark.apache.org/docs/latest/ml-tuning.html

    paramGridRF = ParamGridBuilder().addGrid(rf.numTrees, [10, 30, 60]).addGrid(rf.maxDepth, [3, 6, 12]).build()
    predictionsRF, mRF = predictions(rf,paramGridRF,trainingData,testData)
    end_time = time()
    elapsed_time = end_time - start_time
    print("[Random Forest] With params %s" % paramGridRF)
    print("[Random Forest] time %s" %(elapsed_time))
```

Logistic Regression :

```python
def logistic_regresion1(trainingData, testData):
    print("[Logistic Regression] init")
    start_time = time()
    lr = LogisticRegression(featuresCol="features",labelCol="label",maxIter=100,family="mul
    lrGrid = ParamGridBuilder().addGrid(lr.regParam, [0.1, 0.01, 0.001]).addGrid(lr.elasticNetParam, [0.5, 0.6, 0.8]).build()
    predictionsRL, mRL = predictions(lr,lrGrid,trainingData,testData)
    end_time = time()

    elapsed_time = end_time - start_time
    print("[Logistic Regression] With params %s" % predictionsRL)
    print("[Logistic Regression] time %s" %(elapsed_time))
```

```python
def logistic_regression_2(trainingData, testData):
    print("[Logistic Regression] init")
    start_time = time()
```

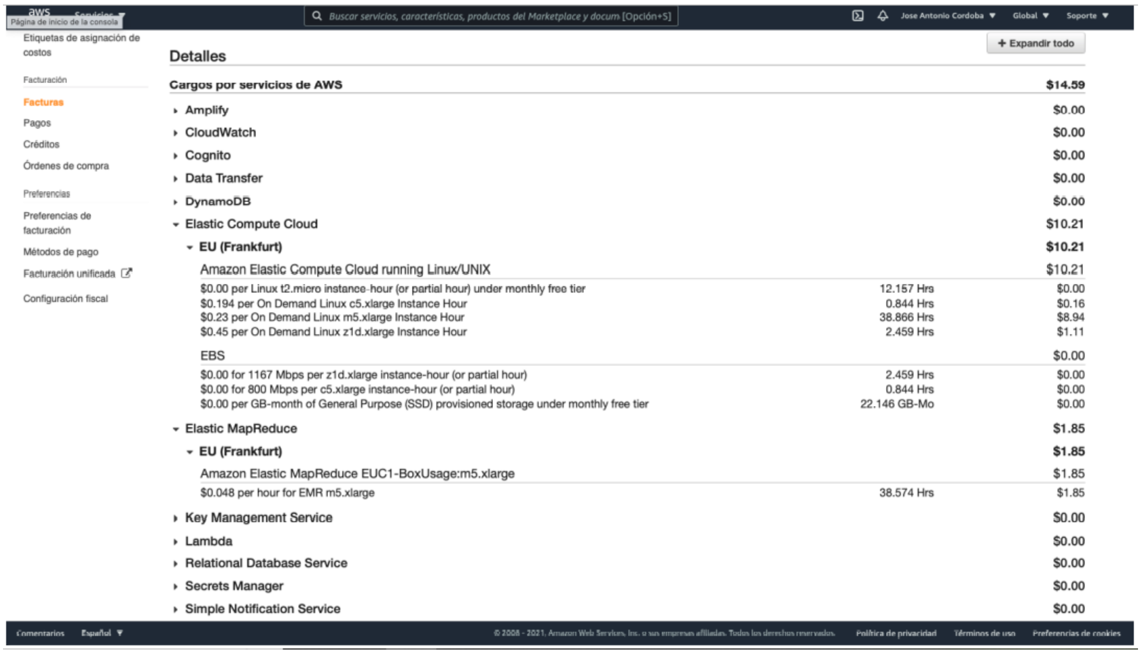
```python
    lr = LogisticRegression(featuresCol="features",labelCol="label",maxIter=100,family="mul
    lrGrid = ParamGridBuilder().addGrid(lr.regParam, [0.1, 0.01, 0.001]).addGrid(lr.elasticNetParam, [0.6, 0.7, 0.9]).build()
    predictionsRL, mRL = predictions(lr,lrGrid,trainingData,testData)
    end_time = time()
    elapsed_time = end_time - start_time
    print("[Logistic Regression] With params %s" % predictionsRL)
    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 :



Detalles

Cargos por servicios de AWS		$14.59
Amplify		$0.00
CloudWatch		$0.00
Cognito		$0.00
Data Transfer		$0.00
DynamoDB		$0.00
Elastic Compute Cloud		$10.21
EU (Frankfurt)		$10.21
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

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 :

```python
# En hadoop: /opt/spark-3.0.1/bin/pyspark --master spark://hadoop-master:7077
# spark-submit --conf spark.jars.ivy=/tmp/.ivy /intercambio/models.py
# exec(open('/intercambio/models.py', encoding="utf-8").read())

# sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose
# -$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
# sudo chmod +x /usr/local/bin/docker-compose


import sys
import os.path
from time import *
import pyspark.sql.functions as func

# Librerias Core de spark
from pyspark import SparkContext, SparkConf, sql

from pyspark.sql.functions import udf
from pyspark.ml.feature import StringIndexer
from pyspark.sql.types import StringType, DoubleType, IntegerType
from pyspark.sql import SparkSession
from functools import reduce
from pyspark.mllib.evaluation import MulticlassMetrics
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.ml import Pipeline
```

```python

# Libreria MLKit de Spark
from pyspark.ml.linalg import *
from pyspark.ml.feature import *
from pyspark.ml.tuning import *
from pyspark.ml.evaluation import *
from pyspark.ml.classification import *
from pyspark.ml import *

# Neural Network: https://runawayhorse001.github.io/LearningApacheSpark/fnn.html
# Random Forest: https://runawayhorse001.github.io/LearningApacheSpark/regression.html?
# highlight=random
#%20forest#random-forest-regression
# Decision Tree: https://runawayhorse001.github.io/LearningApacheSpark/classification.html#id5
# Gradient Boost Tree: https://runawayhorse001.github.io/LearningApacheSpark/classification.html#
# gradient-boosted-tree-classification
# Binomial Logistic Regression: https://runawayhorse001.github.io/LearningApacheSpark/classification.html#
# binomial-logistic-regression

# docker exec -it --user root ubuntu_spark_1 pip install numpy
# docker exec -it  ubuntu_spark_1 /bin/bash
# spark-submit --master spark://spark:7077 --total-executor-cores 4 --executor-memory 4g /intercambio/models.py
# docker exec -it  ubuntu_spark_1 /bin/bash spark-submit --master spark://spark:7077 --total-executor-cores 4 --executor-memory 8g /intercambio/models.py

title = "CC-P4-Modelos"
name_file="/Intercambio/pepitoenpeligro-training.csv"

columns = ['`PredCN_central_2`', '`PredSS_r1_4`', '`PSSM_r1_3_T`', '`AA_freq_central_M`', '`PSSM_r2_-1_L`', '`PSSM_r1_-2_R`']
columns_asIndex= ['PredCN_central_2', 'PredSS_r1_4', 'PSSM_r1_3_T', 'AA_freq_central_M', 'PSSM_r2_-1_L', 'PSSM_r1_-2_R']

portionTrain = 0.8
portionTest = 0.2
```

```
def predictions(estimator, paramGrid, dataTrain, dataTest):
    # binary clasification
    # https://spark.apache.org/docs/latest/
    # mllib-evaluation-metrics.html#binary-classification
    train_validator = TrainValidationSplit(estimator=estimator,
    ↪ estimatorParamMaps=paramGrid,
    ↪ evaluator=BinaryClassificationEvaluator(), trainRatio=portionTrain)
    model = train_validator.fit(dataTrain)
    predictions = model.transform(dataTest)
    predictionAndLabel = predictions.select("prediction","label")

    # convierte labels y prediccones a float
    predictionAndLabel = predictionAndLabel.withColumn("prediction",
    ↪ func.round(predictionAndLabel['prediction']).cast('float'))
    predictionAndLabel = predictionAndLabel.withColumn("label",
    ↪ func.round(predictionAndLabel['label']).cast('float'))
    metrics=MulticlassMetrics(predictionAndLabel
    .select("prediction","label").rdd.map(tuple))


    evaluator = BinaryClassificationEvaluator()
    auRocRF = evaluator.evaluate(predictions)


    # la matriz de confusion revienta
    cnf_matrix = metrics.confusionMatrix()
    accuracy = round(metrics.accuracy*100, 3)
    f1 = metrics.fMeasure(1.0)
    recall = metrics.recall(1.0)


    print("Results of model %s"  % (estimator.__dict__['uid']))
    print("Accuracy %s" % accuracy)
    print("F1 %s" % f1)
    print("Recall %s" % recall)
    print("AUC %s" % auRocRF)

    return predictions, model

# Ya tengo captura de esta ejecucion
def random_forest_1(trainingData,testData):
    print("[Random Forest] init")
```

```
    start_time = time()
    rf = RandomForestClassifier(labelCol="label", featuresCol="features", seed=12345)
    # ParamGridBuilder params:
    # https://spark.apache.org/docs/latest/ml-tuning.html
    paramGridRF = ParamGridBuilder().addGrid(rf.numTrees, [5, 10, 20]).addGrid(rf.maxDepth, [2, 3, 6]).build()

    predictionsRF, mRF = predictions(rf,paramGridRF,trainingData,testData)
    end_time = time()
    elapsed_time = end_time - start_time
    print("[Random Forest] With params %s" % paramGridRF)
    print("[Random Forest] time %s" %(elapsed_time))

# Ya tengo captura de esta ejecucion
def random_forest_2(trainingData,testData):
    print("[Random Forest] init")
    start_time = time()
    rf = RandomForestClassifier(labelCol="label", featuresCol="features", seed=2021)
    # ParamGridBuilder params:
    # https://spark.apache.org/docs/latest/ml-tuning.html
    paramGridRF = ParamGridBuilder().addGrid(rf.numTrees, [10, 30, 60]).addGrid(rf.maxDepth, [3, 6, 12]).build()
    predictionsRF, mRF = predictions(rf,paramGridRF,trainingData,testData)
    end_time = time()
    elapsed_time = end_time - start_time
    print("[Random Forest] With params %s" % paramGridRF)
    print("[Random Forest] time %s" %(elapsed_time))



def gradient_boosted_tree_1(trainingData, testData):
    print("[Gradient Boosted Tree] init")
    start_time = time()
    gbt = GBTClassifier(labelCol="label", featuresCol="features", seed=2021)
    #paramGridGBT = ParamGridBuilder().addGrid(gbt.maxIter, [10, 15, 20]).addGrid(gbt.maxDepth, [3, 6, 12]).build()
    paramGridGBT = ParamGridBuilder().addGrid(gbt.maxIter, [5, 10, 15]).addGrid(gbt.maxDepth, [2, 3, 9]).build()
    predictionsGBT, mGBT = predictions(gbt,paramGridGBT,trainingData,testData)
```

```
    end_time = time()
    elapsed_time = end_time - start_time
    print("[Gradient Boosted Tree] With params %s" % paramGridGBT)
    print("[Gradient Boosted Tree] time %s" %(elapsed_time))


def gradient_boosted_tree_2(trainingData, testData):
    print("[Gradient Boosted Tree] init")
    start_time = time()
    gbt = GBTClassifier(labelCol="label", featuresCol="features",
     ↪ seed=2021)
    paramGridGBT = ParamGridBuilder().addGrid(gbt.maxIter, [10, 15,
     ↪ 20]).addGrid(gbt.maxDepth, [3, 6, 12]).build()

    predictionsGBT, mGBT =
     ↪ predictions(gbt,paramGridGBT,trainingData,testData)
    end_time = time()
    elapsed_time = end_time - start_time
    print("[Gradient Boosted Tree] With params %s" % paramGridGBT)
    print("[Gradient Boosted Tree] time %s" %(elapsed_time))


def perceptron_1(trainingData, testData):
    print("[Peceptron] init")
    start_time = time()
    mlp = MultilayerPerceptronClassifier(
       featuresCol="features",
       labelCol="label",
       predictionCol="prediction",
       maxIter=100
    )

    mlpGrid = ParamGridBuilder().addGrid(mlp.layers, [[7, 3, 2], [7, 9, 3,
     ↪ 2], [7, 5, 2]]).build()
    predictionsMLP, mMLP = predictions(mlp, mlpGrid, trainingData,
     ↪ testData)
    end_time = time()
    elapsed_time = end_time - start_time
    print("[Peceptron] With params %s" % predictionsMLP)
    print("[Peceptron] time %s" %(elapsed_time))


# Ya tengo captura de esta ejecucion
```

```
def logistic_regresion1(trainingData, testData):
    print("[Logistic Regression] init")
    start_time = time()
    lr = LogisticRegression(featuresCol="features",
    labelCol="label",maxIter=100,family="multinomial")
    lrGrid = ParamGridBuilder().addGrid(lr.regParam, [0.1, 0.01, 0.001]).addGrid(lr.elasticNetParam, [0.5, 0.6, 0.8]).build()
    predictionsRL, mRL = predictions(lr,lrGrid,trainingData,testData)
    end_time = time()
    elapsed_time = end_time - start_time
    print("[Logistic Regression] With params %s" % predictionsRL)
    print("[Logistic Regression] time %s" %(elapsed_time))

def logistic_regresion_2(trainingData, testData):
    print("[Logistic Regression] init")
    start_time = time()
    lr = LogisticRegression(featuresCol="features",
    labelCol="label",maxIter=100,family="multinomial")
    lrGrid = ParamGridBuilder().addGrid(lr.regParam, [0.1, 0.01, 0.001]).addGrid(lr.elasticNetParam, [0.6, 0.7, 0.9]).build()
    predictionsRL, mRL = predictions(lr,lrGrid,trainingData,testData)
    end_time = time()
    elapsed_time = end_time - start_time
    print("[Logistic Regression] With params %s" % predictionsRL)
    print("[Logistic Regression] time %s" %(elapsed_time))


def naive_bayes_1(trainingData, testData):
    print("[NaiveBayes] init")
    start_time = time()
    nb = NaiveBayes(modelType="multinomial", featuresCol="features", labelCol="label", smoothing=1.0)
    nbGrid = ParamGridBuilder().addGrid(1.0, [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]).build()
    predictionsNB, mNB = predictions(nb,nbGrid,trainingData,testData)
    end_time = time()
    elapsed_time = end_time - start_time
    print("[NaiveBayes] With params %s" % predictionsNB)
    print("[NaiveBayes] time %s" %(elapsed_time))


if __name__ == "__main__":
```

```
print("Iniciando el contexto de Spark %s", title)
configurationSpark = SparkConf().setAppName(title)
sparkContexto = SparkContext.getOrCreate(conf=configurationSpark)
sqlContext = sql.SQLContext(sparkContexto)

df_columns = sqlContext.read.csv(name_file, sep=",", header=True, inferSchema=True)
indexer = StringIndexer(inputCol="PredSS_r1_4", outputCol="PredSS_r1_4_indexado")
df_columns = indexer.fit(df_columns).transform(df_columns)
df_columns = df_columns.drop("PredSS_r1_4")
df_columns = df_columns.withColumnRenamed("PredSS_r1_4_indexado","PredSS_r1_4")
df_columns.show(20)

clases_negativas = df_columns.filter(df_columns['class']==0).count()
clases_positivas = df_columns.filter(df_columns['class']==1).count()
print("El balanceo negativo/positivio es: %s / %s" % (clases_negativas, clases_positivas))

#Me quedo con el numero de clases de la menor
tam_partition = clases_positivas
if(clases_positivas > clases_negativas):
    tam_partition = clases_negativas

# Reduzco ambos al tamaño de la particion anterior: Undersampling
df_0 = df_columns.filter(df_columns['class'] == 0).limit(tam_partition)
df_1 = df_columns.filter(df_columns['class'] == 1).limit(tam_partition)

df_balanced = df_1.union(df_0)
df_train, df_test = df_balanced.randomSplit([portionTrain, portionTest])
df_balanced_count = df_balanced.select('class').count()

df_train_count = df_train.select('class').count()
df_train_negative_count = df_train.filter(df_columns['class']==0).select('class').count()
df_train_positive_count = df_train.filter(df_columns['class']==1).select('class').count()

df_test_count = df_test.select('class').count()
```

```python
df_test_negative_count = df_test.filter(df_columns['class']==0).select('class').count()
df_test_positive_count = df_test.filter(df_columns['class']==1).select('class').count()

print("[Global] total: %s", df_balanced_count)
print("[Train] positivas: %s, negativas %s, total %s" % (df_train_positive_count, df_train_negative_count, df_train_count ))
print("[Test] positivas: %s, negativas %s, total %s" % (df_test_positive_count, df_test_negative_count, df_test_count ))

# Feature Transformer VectorAssembler in PySpark ML Feature
# https://medium.com/@nutanbhogendrasharma/
# feature-transformer-vectorassembler-in-pyspark
# -ml-feature-part-3-b3c2c3c93ee9
assembler = VectorAssembler(inputCols=columns_asIndex, outputCol='features')
trainingData = assembler.transform(df_train).select("features","class")
.withColumnRenamed("class","label")
testData = assembler.transform(df_test).select("features","class")
.withColumnRenamed("class","label")


# RandomForest - OK
random_forest_1(trainingData, testData)
random_forest_2(trainingData, testData)


# Gradient Boosted Tree - OK
gradient_boosted_tree_1(trainingData,testData)
gradient_boosted_tree_2(trainingData,testData)

# Regresion logistica- OK
logistic_regresion1(trainingData, testData)
logistic_regresion_2(trainingData, testData)


# Perceptron multicapa - No funca
# perceptron_1(trainingData,testData)
```

```
# Naive Bayes - No funca
# naive_bayes_1(trainingData, testData)


# https://stackoverflow.com/questions/60772315/
# how-to-evaluate-a-classifier-with-pyspark-2-4-5
# https://stackoverflow.com/questions/41714698/
# how-to-get-accuracy-precision-recall
# -and-roc-from-cross-validation-in-spark-ml
print("FIN")
```

Spark composition:

```yaml
version: '2'

services:
  spark:
    image: docker.io/bitnami/spark:3
    environment:
      - SPARK_MODE=master
      - SPARK_RPC_AUTHENTICATION_ENABLED=no
      - SPARK_RPC_ENCRYPTION_ENABLED=no
      - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
      - SPARK_SSL_ENABLED=no
    ports:
      - '8080:8080'
    volumes:
      - ./intercambio:/intercambio
  spark-worker-1:
    image: docker.io/bitnami/spark:3
    environment:
      - SPARK_MODE=worker
      - SPARK_MASTER_URL=spark://spark:7077
      - SPARK_WORKER_MEMORY=2G
      - SPARK_WORKER_CORES=2
      - SPARK_RPC_AUTHENTICATION_ENABLED=no
      - SPARK_RPC_ENCRYPTION_ENABLED=no
      - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
      - SPARK_SSL_ENABLED=no
    volumes:

      - ./intercambio:/intercambio
  spark-worker-2:
    image: docker.io/bitnami/spark:3
    environment:
      - SPARK_MODE=worker
      - SPARK_MASTER_URL=spark://spark:7077
      - SPARK_WORKER_MEMORY=2G
      - SPARK_WORKER_CORES=2
      - SPARK_RPC_AUTHENTICATION_ENABLED=no
      - SPARK_RPC_ENCRYPTION_ENABLED=no
      - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
      - SPARK_SSL_ENABLED=no
    volumes:
      - ./intercambio:/intercambio
```

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

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