Fernando Cerriteño Magaña - A01702790

```
#Bibliotecas para poder trabajar con Spark
!sudo apt update
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://downloads.apache.org/spark/spark-3.5.0//spark-3.5.0-
bin-hadoop3.tgz
!tar xf spark-3.5.0-bin-hadoop3.tgz
#Configuración de Spark con Python
!pip install -q findspark
!pip install pyspark

#Estableciendo variable de entorno
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.5.0-bin-hadoop3"

#Buscando e inicializando la instalación de Spark
import findspark
findspark.init()
findspark.find()

#Probando PySparl
from pyspark.sql import DataFrame, SparkSession
from typing import List
import pyspark.sql.types as T
import pyspark.sql.functions as F

spark = SparkSession \
       .builder \
       .appName("Hola mundo Spark") \
       .getOrCreate()

spark
```

```
Hit:1 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/
InRelease
Hit:2
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/
x86_64  InRelease
Hit:3 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:4 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:5 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:6 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:7 https://ppa.launchpadcontent.net/c2d4u.team/c2d4u4.0+/ubuntu
jammy InRelease
Hit:8 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy
InRelease
Hit:9 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu
jammy InRelease
```

```
Hit:10 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy
InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
18 packages can be upgraded. Run 'apt list --upgradable' to see them.
Requirement already satisfied: pyspark in
/usr/local/lib/python3.10/dist-packages (3.5.0)
Requirement already satisfied: py4j==0.10.9.7 in
/usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)

<pyspark.sql.session.SparkSession at 0x7be954169450>
```

Regresión lineal

Importa regresión lineal de la libreria de pyspark

```
from pyspark.ml.regression import LinearRegression
```

Carga la base de datos de entrenamiento en formato libsvm, más info en como funciona el formato en https://stackoverflow.com/questions/44965186/how-to-understand-the-format-type-of-libsvm-of-spark-mllib#:~:text=The%20LibSVM%20format%20is%20quite,one%20is%20the%20actual%20value.

```
training = spark.read.format("libsvm")\
    .load("/content/sample_linear_regression_data.txt")
```

Se crea el modelo de regresión lineal cmo lr, el primer parámetro representa el numero maximo de iteraciones, el segundo parámetro representa el parámetro de regularización y finalmente el ultimo parámetro es el parámetro de Elastic Net

```
lr = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
```

Ajusta el modelo (entrena el modelo de regresión lineal) utilizando los datos de entrenamiento

```
lrModel = lr.fit(training)
```

Imprime los coeficientes y la intercepción del modelo de regresión lineal

```
print("Coefficients: %s" % str(lrModel.coefficients))
print("Intercept: %s" % str(lrModel.intercept))
```

```
Coefficients: [0.0,0.3229251667740594,-
0.3438548034562219,1.915601702345841,0.05288058680386255,0.76596272045
9771,0.0,-0.15105392669186676,-0.21587930360904645,0.2202536918881343]
Intercept: 0.15989368442397356
```

Obtiene un resumen del modelo sobre el conjunto de entrenamiento y muestra algunas métricas

```
trainingSummary = lrModel.summary
print("numIterations: %d" % trainingSummary.totalIterations)
# Muestra el número total de iteraciones realizadas durante el ajuste.
print("objectiveHistory: %s" % str(trainingSummary.objectiveHistory))
# Muestra el historial de la función objetivo durante el
entrenamiento.
trainingSummary.residuals.show()
# Muestra los residuos del modelo.
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
# Muestra el error cuadrático medio de la raíz (Root Mean Squared
Error).
print("r2: %f" % trainingSummary.r2)
# Muestra el coeficiente de determinación (R-cuadrado) del modelo.

numIterations: 6
objectiveHistory: [0.49999999999999994, 0.4967620357443381,
0.49363616643404634, 0.4936351537897608, 0.4936351214177871,
0.49363512062528014, 0.4936351206216114]
+--------------------+
|           residuals|
+--------------------+
|  -9.889232683103197|
|  0.5533794340053553|
|  -5.204019455758822|
| -20.566686715507508|
|    -9.4497405180564|
|  -6.909112502719487|
|  -10.00431602969873|
|  2.0623978070504845|
|  3.1117508432954772|
|  -15.89360822941938|
|  -5.036284254673026|
|  6.4832158769943335|
|  12.429497299109002|
|  -20.32003219007654|
|    -2.0049838218725|
| -17.867901734183793|
|   7.646455887420495|
| -2.2653482182417406|
|-0.10308920436195645|
|  -1.380034070385301|
+--------------------+
only showing top 20 rows

RMSE: 10.189077
r2: 0.022861
```

#Regresión logística binomial

Importación de librerías

```
from pyspark.ml.classification import LogisticRegression
```

Carga la base de datos de entrenamiento en formato libsvm

```
training =
spark.read.format("libsvm").load("/content/sample_binary_classificatio
n_data.txt")
```

Se crea el modelo de regresión logística binaria con los parámetros: Iteración máxima, parámetro de regularización y el parámetro de Elastic Net

```
lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
```

Se ajusta el modelo utilizando los datos de entrenamiento

```
lrModel = lr.fit(training)
```

Imprime los coeficientes y la intercepción del modelo de regresión logística binaria

```
print("Coefficients: " + str(lrModel.coefficients))
print("Intercept: " + str(lrModel.intercept))
```

```
Coefficients: (692,
[272,300,323,350,351,378,379,405,406,407,428,433,434,435,455,456,461,4
62,483,484,489,490,496,511,512,517,539,540,568],[-7.520689871384186e-
05,-8.115773146847071e-05,3.814692771846355e-
05,0.00037764905404243413,0.00034051483661944043,0.0005514455157343107
,0.0004085386116096918,0.000419746733274946,0.0008119171358670032,0.00
050527708372668753,-2.392926040660163e-
05,0.0005745048020902295,0.0009037546426803719,7.818229700243984e-05,-
2.1787551952912656e-05,-3.4021658217896046e-
05,0.0004966517360637638,0.0008190557828370373,-8.017982139522677e-
05,-2.743169403783598e-
05,0.00048108322622389907,0.00048408017626778754,-8.92647292001121e-
06,-0.0003414881233042733,-8.950592574121474e-
05,0.00048645469116892124,-8.478698005186183e-05,-
0.00042347832158317684,-7.296535777631314e-05])
Intercept: -0.599146028640144
```

Como extra se utiliza la familia multinomial para clasificación binaria

```
mlr = LogisticRegression(maxIter=10, regParam=0.3,
elasticNetParam=0.8, family="multinomial")
```

```
# Ajusta el modelo de regresión logística utilizando la familia multinomial
mlrModel = mlr.fit(training)
```

Imprime los coeficientes e intercepciones para la regresión logística con familia multinomial

```
print("Multinomial coefficients: " + str(mlrModel.coefficientMatrix))
print("Multinomial intercepts: " + str(mlrModel.interceptVector))
```

```
Multinomial coefficients: 2 X 692 CSRMatrix
(0,272) 0.0001
(0,300) 0.0001
(0,350) -0.0002
(0,351) -0.0001
(0,378) -0.0003
(0,379) -0.0002
(0,405) -0.0002
(0,406) -0.0004
(0,407) -0.0002
(0,433) -0.0003
(0,434) -0.0005
(0,435) -0.0001
(0,456) 0.0
(0,461) -0.0002
(0,462) -0.0004
(0,483) 0.0001
..
..
Multinomial intercepts: [0.2750587585718083,-0.2750587585718083]
```

K-Means

Carga de las librerías

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
```

Carga de la base de datos de entrenamiento en formato libsvm

```
dataset =
spark.read.format("libsvm").load("/content/sample_kmeans_data.txt")
```

Se entrena un modelo de k-means con 2 clústeres y una semilla específica

```
kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(dataset)
```

Realiza predicciones utilizando el modelo de k-means en los datos

```
predictions = model.transform(dataset)
```

Se evalua el cluster con el puntaje silhuette

```
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " +
str(silhouette))
```

```
Silhouette with squared euclidean distance = 0.9997530305375207
```

Se muestran los resultados

```
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Cluster Centers:
[9.1 9.1 9.1]
[0.1 0.1 0.1]
```