BIG DATA

PROYECTO FINAL

Atanas Turlakov

New Technology School

Tokio.

New
Technology
School

Atanas Turlakov

RESPONSABLE DEL DESARROLLO DE LA APLICACIÓN

Fecha: 11/02/2024

BIG DATA
PROYECTO FINAL
Atanas Turlakov

Tokio.

RESUMEN

El siguiente documento es un informe tecnico explicando los pasos como los que se ha realizado el estudio. Se trata de analizar el trafico saliente desde el aeropuerto de San Francisco a Tokio y otras ciudades de Japon. Se van a proporcionar ejemplos de codigo con explicaciones.

METODOLOGIA

En el estudio se ha usado multiples tecnologias para llegar al objetivo tanto herramientas de programacion como base de datos y otras.
Lenguajes y librerias – Python, PySpark, Bash Scripting
Systemas Operativos – Debian GNU/Linux 12
Base de datos – Apache Cassandra
Plataformas de contenedores - Docker

DESARROLLO

La primera tarea es catagorizar los datos según su estructora. Lo hacemos atravez de un script categorizar_datos.py

```python
import subprocess
import pkg_resources
import pandas as pd
from tabulate import tabulate

def install(package):
    subprocess.check_call([sys.executable, "-m", "pip", "install", package])

try:
    pkg_resources.get_distribution('tabulate')
except pkg_resources.DistributionNotFound:
    install('tabulate')

archivo_csv = 'Air_Traffic_Passenger_Statistics.csv'

df = pd.read_csv(archivo_csv)
estructura_datos = []

for columna in df.columns:
    tipo_dato = "Categórico" if df[columna].dtype == 'object' else "Numérico"
    estructura_datos.append([columna, tipo_dato])

print(tabulate(estructura_datos, headers=["Nombre del Campo", "Tipo de Dato"], tablefmt="fancy_grid"))
```

El script crea un dataframe de pandas tabulado tipo “fancy_grid” que crea dos columnas “Nombre de campo” y “Tipo de dato” haciendo un resumen.

La segunda tarea se trata de realizar varias consultas obteniendo resultados que ayudaran a tomar futuras deseciones. Es una tarea completamente automatizada a traves de scripts de Bash y Python ejecutandose en un contenedor Docker de Cassandra

El nombre de script escassandra_project.sh el que se debe dar permisos de ejecucion "chmod +x cassandra_project.sh" y ejecutarlo con permiso de administrador.

Lo primero lo que hace el script es comprobar si esta instalado docker y si no lo esta lo instala, añade el usuario actual al grupo docker y modifica el archivo sudoers para que no pida contraseña al docker

```
if [ ! -n "$(command -v docker)" ]; then
    sudo apt update
    sudo apt upgrade -y
    sudo curl -fsSL https://get.docker.com | sh
    sudo systemctl start docker
        sudo systemctl enable docker
fi

usermod -aG docker ${USER}
echo "%sudo ALL=(ALL) NOPASSWD: /usr/bin/docker" | sudo tee -a /etc/sudoers >/dev/null
```

Descargamos la imagen de Cassandra y creamos el contenedor

```
# Install Cassandra container
docker pull cassandra:latest > /dev/null
docker run -d --name cassandra-container -p 9042:9042 cassandra > /dev/null
```

Exportamos las variables para obtener el ID y el IP del contenedor

```
# Set variables
export GET_CONTAINER_ID=$(docker ps | grep cassandra-container | awk '{print $1}')
export GET_CONTAINER_IP=$(docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'  ${GET_CONTAINER_ID})
```

Tenemos preparados dos scripts.

get_csv_headers.py – extraemos los headers del .csv que vamos a nesecitar para crear la tabla de Cassandra

```
import csv
import os

file_name = os.path.join(os.getcwd(), "Air_Traffic_Passenger_Statistics.csv")

with open(file_name, "r") as f:
    lector_csv = csv.reader(f)
    header_list = next(lector_csv)

text_var = 'text'
int_var = 'int'
l = []
for i in header_list:
    if i == 'Activity Period' or i == 'Passenger Count' or i == 'Adjusted Passenger Count' or i == 'Year':
        l.append('"{}" {} '.format(i, int_var))
    else:
        l.append('"{}" {} '.format(i, text_var))
l.insert(0, '"id" INT PRIMARY KEY')
res = ",".join(l)
print("CREATE TABLE air_traffic ({});".format(res))
```

set_csv_id.py – creamos otro archivo modificado con campos con id

```
import csv

input_file = 'Air_Traffic_Passenger_Statistics.csv'
output_file = 'Air_Traffic_Passenger_Statistics_with_ID.csv'

with open(input_file, 'r') as csv_input, open(output_file, 'w', newline='') as csv_output:
    reader = csv.reader(csv_input)
    writer = csv.writer(csv_output)

    header = next(reader)
    header.insert(0, 'id')
    writer.writerow(header)
    id_counter = 1

    for row in reader:
        row.insert(0, id_counter)
        writer.writerow(row)
        id_counter += 1
```

Damos permiso de ejecucion a los archivos y creamos una funcion para preparar el cluster de Cassandra

```
chmod +x get_csv_headers.py set_csv_id.py
python3 set_csv_id.py

export GET_CSV_HEADERS=$(python3 get_csv_headers.py)

cluster_prepair(){
    if time grep -qi "starting listening" <(docker logs -f cassandra-container 2>&1); then
        docker exec -it cassandra-container cqlsh ${GET_CONTAINER_IP} -e "CREATE KEYSPACE final_project WITH replication = {
            'class': 'SimpleStrategy', 'replication_factor': 1}"
        docker exec -it cassandra-container cqlsh ${GET_CONTAINER_IP} -e "USE final_project; ${GET_CSV_HEADERS}"
        echo ${GET_CSV_HEADERS} > air_traffic.cql
        docker exec -it ${GET_CONTAINER_ID} cqlsh -e "CREATE TABLE --keyspace=final_project -f air_traffic.cql"
        docker cp Air_Traffic_Passenger_Statistics_with_ID.csv ${GET_CONTAINER_ID}:/
        docker exec -it ${GET_CONTAINER_ID} cqlsh -e "COPY final_project.air_traffic (\"id\", \"Activity Period\", \"Operating
    fi
}
cluster_prepair
```

Punto importante es que el contenedor nesecita un cierto tiempo para inicializarse asi que la condiccion

“if time grep -qi ‘starting listening’ < (docker logs -f cassandra-container)”

pone la ejecucucion en espera hasta que aparezca ‘starting listening’ en el log de docker.

La siguiente accion es instalar las librerias python nesesarias o comprobar si ya estan: cassandra-driver, tabulate y docker

```
cassandra_drivers(){
    if [ ! command -n pip ]; then
        sudo apt install -y pip
    fi

    if [ ! pip show cassandra-driver >/dev/null 2>&1 ] && [ ! pip show tabulate >/dev/null 2>&1 ] && [ ! pip show docker >/dev/null 2>&1 ]; then
        pip install cassandra-driver tabulate docker
    fi
}
cassandra_drivers
```

la ultima accion de esta tarea es ejecutar las consultas requeridas. Estan definidas en dos scripts de python

```
chmod +x air_china_registers.py air_berlin_registers.py

python3 air_china_registers.py
python3 air_berlin_registers.py
```

air_china registers.py

```
from tabulate import tabulate
import os

def space():
        print("\n\n\n")

system_ip = os.environ.get('GET_CONTAINER_IP')
cluster = Cluster([system_ip])
session = cluster.connect('final_project')

rows = session.execute('SELECT "Published Airline", "Published Airline IATA Code", "Terminal", "Boarding Area", "Passenger Count", "Month",

data = [list(row) for row in rows]
columns = rows.column_names
output = tabulate(data, headers=columns, tablefmt="fancy_grid")

space()
print('############################   Recuperar todos los registros de la aerolínea “Air China”   ############################')
space()
print(output)
```

air_berlin_registers.py

```
from tabulate import tabulate
import os

def space():
        print("\n\n\n")

system_ip = os.environ.get('GET_CONTAINER_IP')
cluster = Cluster([system_ip])
session = cluster.connect('final_project')

rows = session.execute('SELECT "Published Airline", "Published Airline IATA Code", "Terminal", "Boarding Area", "Passenger Count", "Month",

data = [list(row) for row in rows]
columns = rows.column_names
output = tabulate(data, headers=columns, tablefmt="fancy_grid")

space()
print('############################  Recuperar todos los vuelos de la compañía “AirBerlín”embarcados porla puerta “G” ##################
space()

print(output)
```

Las siguientes tareas se van a realizar con pyspark en entorno Google Colab

Montamos el drive, instalamos pyspark y cargamos el archivo .csv el que vamos a usar

```
from google.colab import drive
drive.mount('/content/drive')
```

```
[ ] pip install pyspark
```

```
[ ] from pyspark.sql import SparkSession
    from pyspark.sql.functions import col, collect_set, format_number, max
    import pandas as pd

    spark = SparkSession.builder\
      .master("local")\
      .appName("Pyspark_SQL")\
      .config("spark.ui.port", '4050')\
      .getOrCreate()
    df = spark.read.option("Header", True).csv("/content/Air_Traffic_Passenger_Statistics.csv")
```

Una vez preparado el entorno hacemos la primera consulta sobre el numero de companias aereas que aparecen en el archivo. Seleccionamos la columna “Operating Airlines” del dataframe, eliminamos los valores duplicados, mostramos el resultado y despues contampos el numero total de companias

```
df.select('Operating Airline').distinct().show()
number_airlines = df.select('Operating Airline').distinct().count()

print("En el registro hay {} diferentes companias".format(number_airlines))
```

Convertimos los valores de “Passenger Count” a enteros, contampos el promedio de cada compania, formateando hasta dos decimales

```
df = df.withColumn("Passenger Count", col("Passenger Count").cast("integer"))
result = df.groupBy("Operating Airline").avg("Passenger Count")
result = result.withColumn("avg(Passenger Count)", format_number(col("avg(Passenger Count)"), 2))
result.show()
```

Eliminamos los registros duplicados por el campo “GEO Region” manteniendo aquel con mayor numero de pasajeros

```
df = df.withColumn("Passenger Count", col("Passenger Count").cast("integer"))
deduplicated_df = df.groupBy("GEO Region").agg(max("Passenger Count")\
                                               .alias("Max Passenger Count"))

deduplicated_df.show()
```

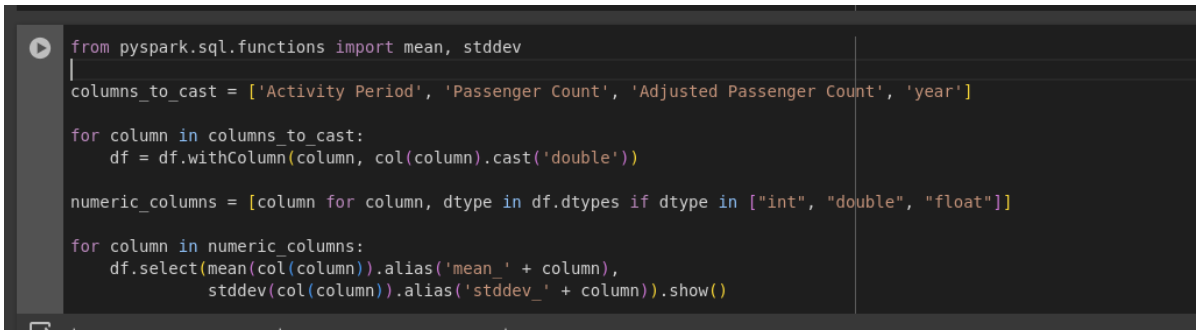
Volcamos los resultados a un archivo .csv

```
[ ] result.write.mode("overwrite").csv("/content/sample_data/passenger_average.csv", header=True)
    deduplicated_df.write.mode("overwrite").csv("/content/single_registers.csv", header=True)
```

Preparamos un entorno Google Colab para la suiguiente tarea

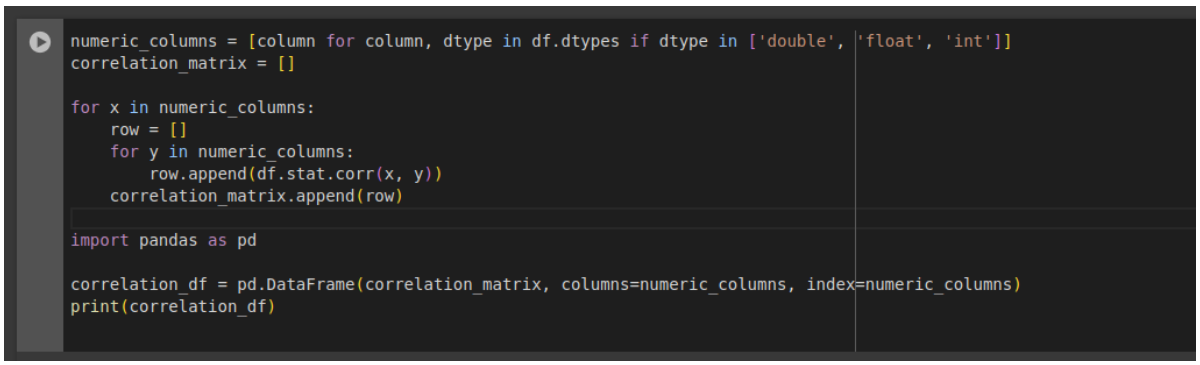
```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

    pip install pyspark

[ ] from pyspark.sql import SparkSession
    from pyspark.sql.functions import col, collect_set, format_number, max
    import pandas as pd
    from pyspark.ml.feature import StringIndexer, OneHotEncoder
    from pyspark.ml import Pipeline

    spark = SparkSession.builder\
      .master("local")\
      .appName("Pyspark_SQL")\
      .config("spark.ui.port", '4050')\
      .getOrCreate()
    df = spark.read.option("Header", True).csv("/content/Air_Traffic_Passenger_Statistics.csv")
```

Hacemos un analisis descriptivo calculando la media y la desviacion estandar para sacar ciertas concluciones



```
from pyspark.sql.functions import mean, stddev

columns_to_cast = ['Activity Period', 'Passenger Count', 'Adjusted Passenger Count', 'year']

for column in columns_to_cast:
    df = df.withColumn(column, col(column).cast('double'))

numeric_columns = [column for column, dtype in df.dtypes if dtype in ["int", "double", "float"]]

for column in numeric_columns:
    df.select(mean(col(column)).alias('mean_' + column),
              stddev(col(column)).alias('stddev_' + column)).show()
```

Hacemos un analizis de correlacion de la mañera de que estan relacionadas las variables. Primero seleccionamos las columnas numericas, inicializamos la matriz de correlacion, calculamos la matriz y la convertimos en un dataframe pandas



```
numeric_columns = [column for column, dtype in df.dtypes if dtype in ['double', 'float', 'int']]
correlation_matrix = []

for x in numeric_columns:
    row = []
    for y in numeric_columns:
        row.append(df.stat.corr(x, y))
    correlation_matrix.append(row)

import pandas as pd

correlation_df = pd.DataFrame(correlation_matrix, columns=numeric_columns, index=numeric_columns)
print(correlation_df)
```

Por ultimo aplicamos un algoritmo. Preparamos los datos convertiendo algunas en double, se hace una indexacion y codificacion de las columnas categoricas. Se utiliza VectorAssembler para combinar columnas categoricas codificadas y columnas numericas. Despues entrenamos el modelo de regrecion lineal. El conjunto de datos se divide a subconjuntos de entrenamiento y prueba usando randomSplit. Evaluacion del modelo – se realizan predicciones en el conjunto de prueba, se evalua el

rendimiento del modelo usando RMSE y R2 a traves de RegressionEvaluator

```python
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql.functions import col

columns_to_cast = ['Activity Period', 'Passenger Count', 'Adjusted Passenger Count', 'year']
for column in columns_to_cast:
    df = df.withColumn(column, col(column).cast('double'))

numeric_columns = ['Activity Period', 'Adjusted Passenger Count', 'year']

indexers = [
    StringIndexer(inputCol=c, outputCol=c+"_index", handleInvalid="keep")
    for c in categorical_columns
]

encoders = [
    OneHotEncoder(inputCol=c+"_index", outputCol=c+"_vec")
    for c in categorical_columns
]

assemblerInputs = [c + "_vec" for c in categorical_columns] + numeric_columns

vecAssembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")

lr = LinearRegression(featuresCol="features", labelCol="Passenger Count")

pipeline = Pipeline(stages=indexers + encoders + [vecAssembler, lr])

(train_data, test_data) = df.randomSplit([0.8, 0.2])
model = pipeline.fit(train_data)

predictions = model.transform(test_data)
evaluator_rmse = RegressionEvaluator(labelCol="Passenger Count", predictionCol="prediction", metricName="rmse")
evaluator_r2 = RegressionEvaluator(labelCol="Passenger Count", predictionCol="prediction", metricName="r2")

rmse = evaluator_rmse.evaluate(predictions)
r2 = evaluator_r2.evaluate(predictions)

print(f"RMSE: {rmse}")
print(f"R2: {r2}")
```