

PYSPARK: TREINAMENTO DE MODELO

BEL COGO & BRUNO HOFFMANN



Agenda

- Nossos objetivos
- Mudanças no Script
- Ambiente spark
- Dados utlizados
- Ambiente original
- Comparativo dos ambientes
- Desafios
- Configuração do AWS





Nossos objetivos

PRINCIPAL

 Executar o modelo do Grau A no Spark;

SECUNDÁRIO

- Configurar um ambiente um ambiente com Spark;
- Fazer comparativo de tempo de execução entre Treinamento no Spark vs. Original;
- o Mais dados!



Mudanças no Script - Definição do Schema

```
schema = StructType([
   StructField('Age', IntegerType(), nullable=False),
   StructField('Gender', IntegerType(), nullable=False),
   StructField('BMI', FloatType(), nullable=False),
   StructField('Smoking', IntegerType(), nullable=False),
   StructField('GeneticRisk', IntegerType(), nullable=False),
   StructField('PhysicalActivity', FloatType(), nullable=False),
   StructField('AlcoholIntake', FloatType(), nullable=False),
   StructField('CancerHistory', IntegerType(), nullable=False),
   StructField('Diagnosis', IntegerType(), nullable=False),
# Importa os dados csv
patients_data = spark.read.csv('./data/The_Cancer_data_Generated.csv', header=True, schema=schema)
```

Mudanças no Script – Preparação dos Dados

Novo

```
features_one_hot_encoder = 'GeneticRisk'
features = ['Age', 'Gender', 'BMI', 'Smoking', 'PhysicalActivity',
            'AlcoholIntake', 'CancerHistory', 'GeneticRisk_encoded']
# Realiza OneHotEncoder
one_hot_encoder = OneHotEncoder(inputCol=features_one_hot_encoder,
                                outputCol="GeneticRisk_encoded")
encoded_data = one_hot_encoder.fit(patients_data).transform(patients_data)
# Vetoriza as features
assembler = VectorAssembler(inputCols=features,
                           outputCol="features")
assembled_df = assembler.transform(encoded_data)
# Escalonamento dos dados
standardScaler = StandardScaler(inputCol="features",
                                outputCol="features_scaled")
scaled df = standardScaler.fit(assembled df).transform(assembled df)
# Dividindo o dataset entre treino e teste.
splits = scaled_df.randomSplit([0.7, 0.3], seed=42)
train_patients_data = splits[0]
test_patients_data = splits[1]
```

Antigo

```
Divisão dos dados em X (features) e Y (classe)
X = patients_data.iloc[:, 0:8].values
y = patients_data.iloc[:, 8].values
# Criação do OneHotEncoder para coluna 4 (Genetic Risk)
one_hot_encoder = ColumnTransformer(transformers=[('OneHot',
                                                   OneHotEncoder(),
                                                   [4])],
                                                   remainder='passthrough')
X = one hot encoder.fit transform(X)
# Transformando os dados para a mesma escala
scaler = StandardScaler()
X = scaler.fit_transform(X)
# Divisão dos dados em treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
                                                    random state=42)
```





Mudanças no Script - Treinamento



Novo

Antigo

```
start_time_train_data = time.time()
classifier = RandomForestClassifier(n_estimators=300)
classifier.fit(X_train, y_train)
end_time_train_data = time.time()
```

Mudanças no Script – Métricas

Novo

Antigo

```
y_pred = classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```



 $\bigcirc\bigcirc\bigcirc\bigcirc$

Ambiente Spark

Workers	Master	Historico	TOTAL
7	1	1	9

Configurando o Docker

- Uso da imagem "FROM python:3.10-bullseye as sparkbase"
- Instalando configurações "apt-get install";
- Executar download do Spark;
- Configurações de variáveis de ambiente;
- Instalando dependências do python;
- o Execução de um arquivo ".sh"

```
SPARK_WORKLOAD=$1
echo "SPARK_WORKLOAD: $SPARK_WORKLOAD"
    "$SPARK_WORKLOAD" == "master" ];
then
 start-master.sh -p 7077
elif [ "$SPARK_WORKLOAD" == "worker" ];
then
 start-worker.sh spark://spark-master:7077
elif [ "$SPARK_WORKLOAD" == "history" ]
then
 start-history-server.sh
```

Configurando Docker-compose

- o Definição dos serviços:
 - Spark-master
 - Spark-history-server
 - Spark-worker
- Uso da imagem que definimos antes;
- Entrypoint é o script .sh;
- o Definição de volumes:
 - o Pasta resources;
 - o Pasta src;









https://www.youtube.com/watch?v=0zjmmkgBnVA



Spark Master at spark://spark-master:7077

URL: spark://spark-master:7077

Alive Workers: 7

Cores in use: 14 Total, 14 Used

Memory in use: 61.0 GiB Total, 7.0 GiB Used

Resources in use:

Applications: 1 Running, 0 Completed Drivers: 0 Running, 0 Completed

Status: ALIVE

→ Workers (7)

Worker Id	Address	State	Cores	Memory	Resources
worker-20241128122137-172.19.0.5-36687	172.19.0.5:36687	ALIVE	2 (2 Used)	8.7 GiB (1024.0 MiB Used)	
worker-20241128122138-172.19.0.4-44927	172.19.0.4:44927	ALIVE	2 (2 Used)	8.7 GiB (1024.0 MiB Used)	
worker-20241128122138-172.19.0.6-33871	172.19.0.6:33871	ALIVE	2 (2 Used)	8.7 GiB (1024.0 MiB Used)	
worker-20241128122138-172.19.0.7-41095	172.19.0.7:41095	ALIVE	2 (2 Used)	8.7 GiB (1024.0 MiB Used)	
worker-20241128122139-172.19.0.10-35597	172.19.0.10:35597	ALIVE	2 (2 Used)	8.7 GiB (1024.0 MiB Used)	
worker-20241128122139-172.19.0.8-45103	172.19.0.8:45103	ALIVE	2 (2 Used)	8.7 GiB (1024.0 MiB Used)	
worker-20241128122139-172.19.0.9-44047	172.19.0.9:44047	ALIVE	2 (2 Used)	8.7 GiB (1024.0 MiB Used)	

→ Running Applications (1)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20241128122559-	PySpark-Get- Started	14	1024.0 MiB		2024/11/28 12:25:59	root	RUNNING	3 s



Comando para Executar:

Buildar nossa imagem docker

docker build . -t da-spark-image

Rodar o docker-compose

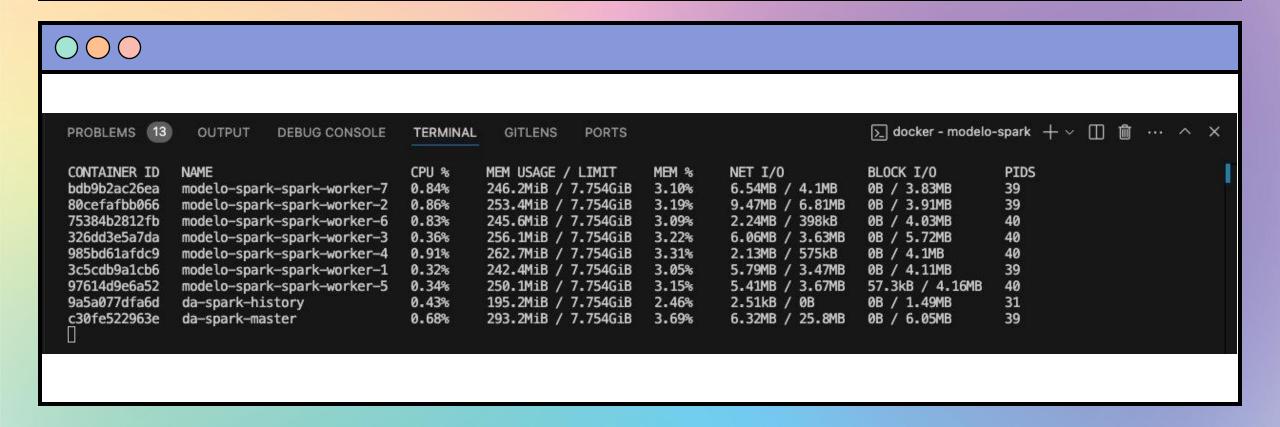
docker-compose up --scale sparkworker=3

Executar o script python

docker exec da-spark-master sparksubmit --master spark://spark-master:7077 --deploy-mode client ./apps/train_script.py

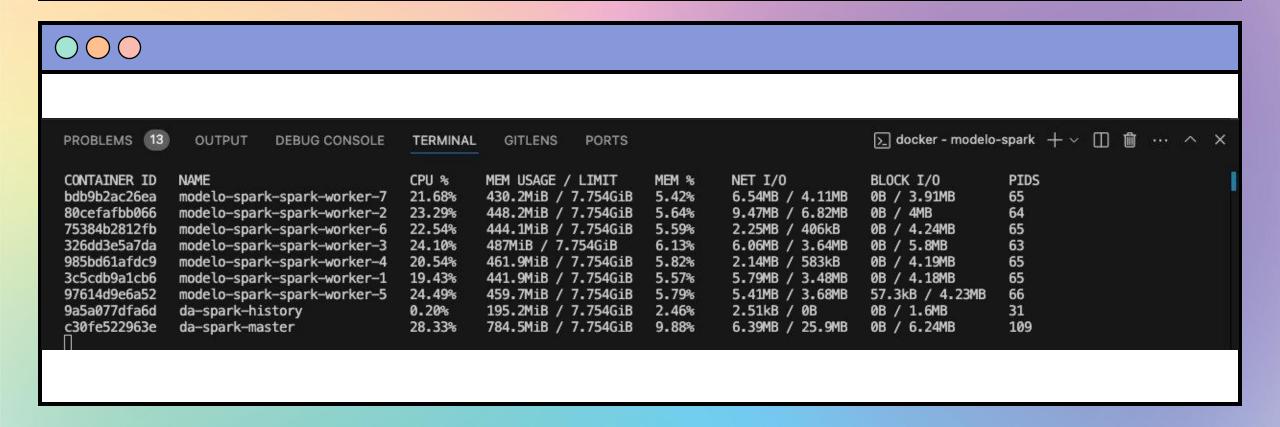


Métricas Spark - Antes





Métricas Spark - Durante



Mas e os dados?

```
import pandas as pd
from ctgan import CTGAN
# Carregar o CSV original
patients_data = pd.read_csv("/content/The_Cancer_data_1500_V2.csv")
# Treinar o modelo CTGAN
model = CTGAN(epochs=10)
model.fit(patients_data)
# Criar dados sintéticos
synthetic_data = model.sample(150000)
# Salvar os novos dados em um arquivo CSV
synthetic_data.to_csv("/content/The_Cancer_data_Generated.csv", index=False)
print("Dados sintéticos gerados e salvos em 'The_Cancer_data_Generated.csv'.")
```



Configuração do modelo original

- Memória: 1GB
- Mesma quantidade de dados
- Mesma configuração de quantidade de árvores

 $\bigcirc\bigcirc\bigcirc$

Comparativo Tempo de execução

	Modelo Original	Modelo Spark			
Preparação de dados (s)	0,71	84.4			
Treinamento (s)	33.79	100,12			
Testes (s)	1,02	0,3			
Métricas (s)	0,015	32,03			



Desafios

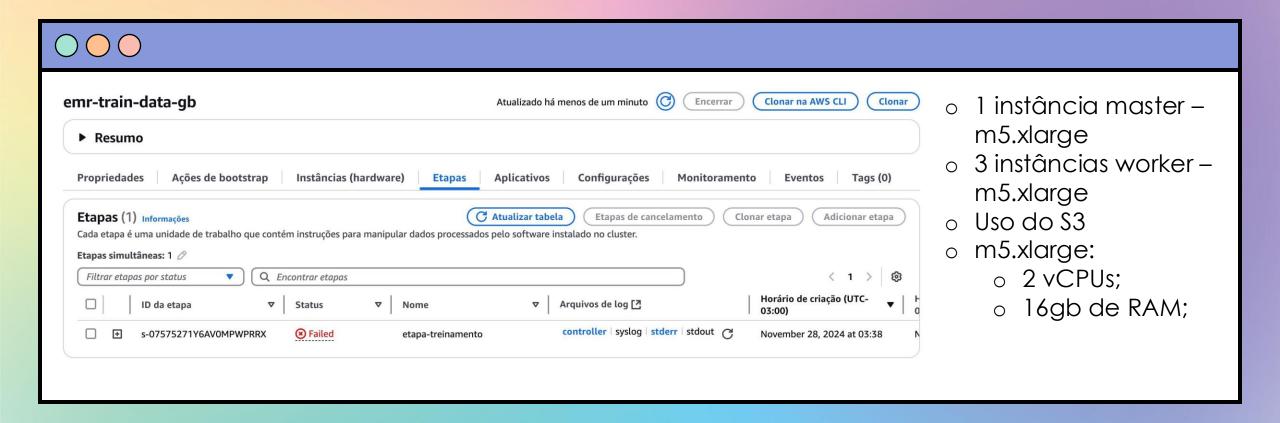


- o **Limitações de uma única máquina**, mesmo que containerizada
 - Tentamos diminuir os recursos do container com código original;
 - Aumentamos recursos dos workers com código Spark;
 - Aumentamos a quantidade de dados de 1.500 para 150.000;
 - o No fim, continuávamos com o mesmo resultado final;
- Dificuldade para executar o EMR via AWS Academy;
 - Configuramos o S3, o VPC e o cluster EMR;
 - Subia tudo certinho;
 - o Toda vez que executava as etapas, ele gerava erro;
 - Erros de permissionamento;
 - Provavelmente por conta de limitação de configurações de segurança;



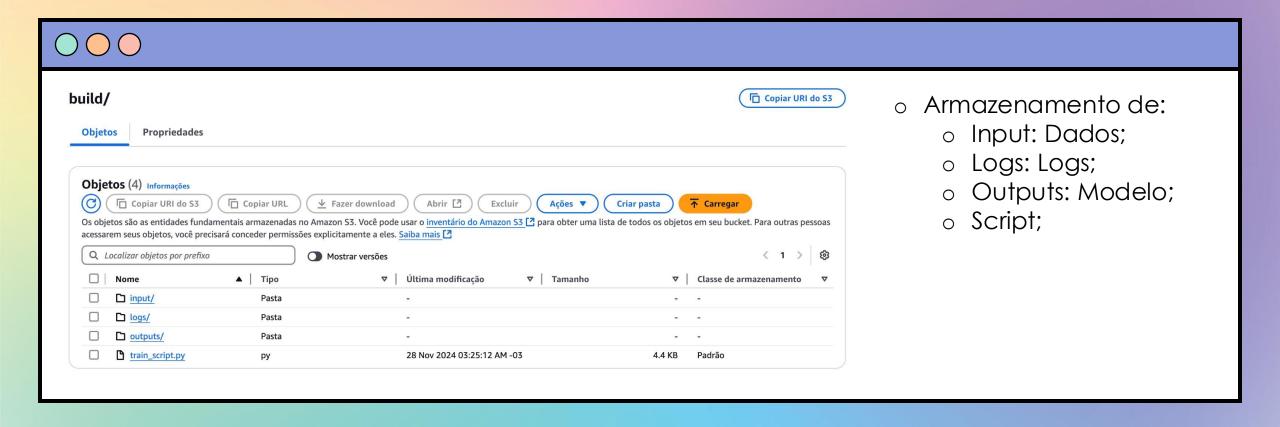


Configurações AWS - EMR



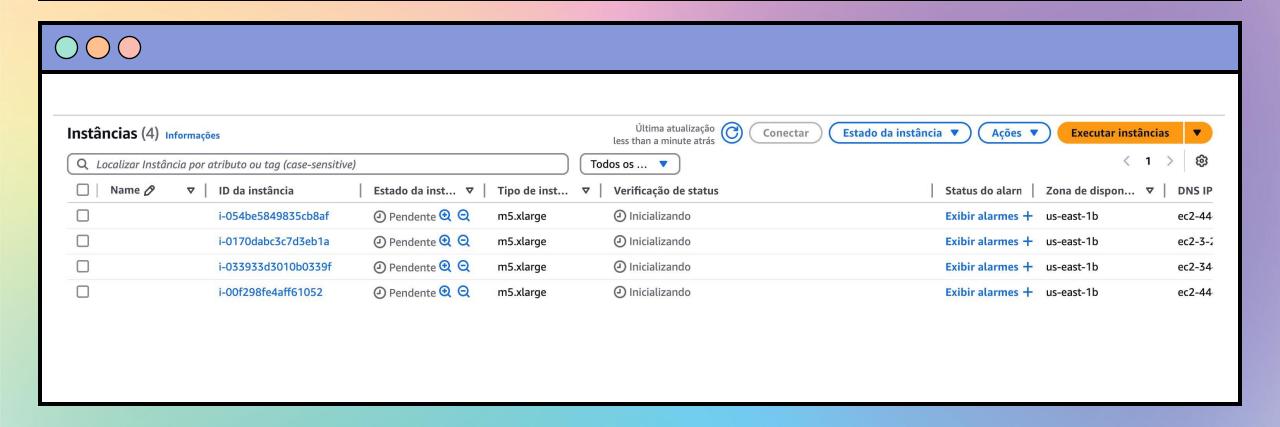


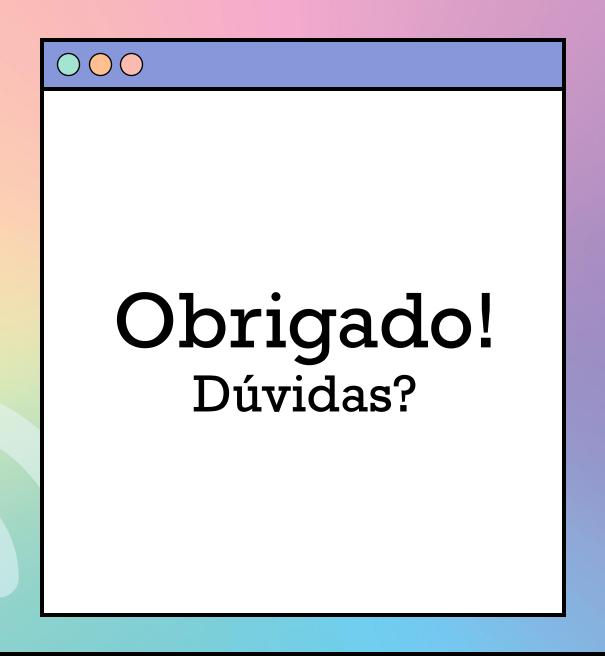
Configurações AWS - S3





Configurações AWS - Instâncias







Referências

- How to Run a Spark Cluster with Multiple Workers Locally Using Docker https://www.youtube.com/watch?v=FteThJ-YvXk
- Setting up a Spark standalone cluster on Docker in layman terms https://medium.com/@MarinAgli1/setting-up-a-spark-standalone-cluster-on-docker-in-layman-terms-8cbdc9fdd14b
- Intro to Amazon EMR Big Data Tutorial using Spark https://www.youtube.com/watch?v=8bOgOvz6Tcg&t=76s