

PYSPARK: TREINAMENTO DE MODELO

BEL COGO & BRUNO HOFFMANN

Agenda

- Nossos objetivos
- Mudanças no Script
- Ambiente spark
- Dados utilizados
- 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;
- 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

```
start_time_train_data = time.time()
rf_classifier = RandomForestClassifier(featuresCol="features_scaled",
                                     labelCol="Diagnosis", numTrees=300)
model = rf_classifier.fit(train_patients_data)
end_time_train_data = time.time()
```

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

```
predictions = model.transform(test_patients_data)
classifications = predictions.select("prediction", "Diagnosis")

evaluator = MulticlassClassificationEvaluator(labelCol="Diagnosis",
                                             predictionCol="prediction")

start_time_metrics_data = time.time()
accuracy = evaluator.evaluate(classifications,
                             {evaluator.metricName: "accuracy"})
precision = evaluator.evaluate(classifications,
                               {evaluator.metricName: "weightedPrecision"})
recall = evaluator.evaluate(classifications,
                            {evaluator.metricName: "weightedRecall"})
f1_score = evaluator.evaluate(classifications,
                              {evaluator.metricName: "f1"})
```

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)
```

Ambiente Spark

Workers	Master	Historico	TOTAL
7	1	1	9

START



Configurando o Docker

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

```
SPARK_WORKLOAD=$1

echo "SPARK_WORKLOAD: $SPARK_WORKLOAD"

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

Configurando Docker-compose

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

```
spark-worker:  
  image: da-spark-image  
  entrypoint: ['./entrypoint.sh', 'worker']  
  depends_on:  
    - spark-master  
  env_file:  
    - .env.spark  
  volumes:  
    - ./resources:/opt/spark/data  
    - ./src:/opt/spark/apps  
    - spark-logs:/opt/spark/spark-events
```



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-20241128122559-0000 (kill)	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 spark-  
worker=3
```

```
# Executar o script python  
docker exec da-spark-master spark-  
submit --master spark://spark-master:7077  
--deploy-mode client  
./apps/train_script.py
```

Métricas Spark - Antes

PROBLEMS13

OUTPUT

DEBUG CONSOLE

TERMINAL

GITLENS

PORTS

docker - modelo-spark

Métricas Spark - Durante

PROBLEMS 13

OUTPUT

DEBUG CONSOLE

TERMINAL

GIT LENS

PORTS

docker - modelo-spark + v [icon] [icon] ... ^ X

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
bdb9b2ac26ea	modelo-spark-spark-worker-7	21.68%	430.2MiB / 7.754GiB	5.42%	6.54MB / 4.11MB	0B / 3.91MB	65
80cefafbb066	modelo-spark-spark-worker-2	23.29%	448.2MiB / 7.754GiB	5.64%	9.47MB / 6.82MB	0B / 4MB	64
75384b2812fb	modelo-spark-spark-worker-6	22.54%	444.1MiB / 7.754GiB	5.59%	2.25MB / 406kB	0B / 4.24MB	65
326dd3e5a7da	modelo-spark-spark-worker-3	24.10%	487MiB / 7.754GiB	6.13%	6.06MB / 3.64MB	0B / 5.8MB	63
985bd61afdc9	modelo-spark-spark-worker-4	20.54%	461.9MiB / 7.754GiB	5.82%	2.14MB / 583kB	0B / 4.19MB	65
3c5cdb9a1cb6	modelo-spark-spark-worker-1	19.43%	441.9MiB / 7.754GiB	5.57%	5.79MB / 3.48MB	0B / 4.18MB	65
97614d9e6a52	modelo-spark-spark-worker-5	24.49%	459.7MiB / 7.754GiB	5.79%	5.41MB / 3.68MB	57.3kB / 4.23MB	66
9a5a077dfa6d	da-spark-history	0.20%	195.2MiB / 7.754GiB	2.46%	2.51kB / 0B	0B / 1.6MB	31
c30fe522963e	da-spark-master	28.33%	784.5MiB / 7.754GiB	9.88%	6.39MB / 25.9MB	0B / 6.24MB	109

START



13

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

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

- **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;
 - 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;
 - Toda vez que executava as etapas, ele gerava erro;
 - Erros de permissionamento;
 - Provavelmente por conta de limitação de configurações de segurança;

Configurações AWS - EMR

emr-train-data-gb

Atualizado há menos de um minuto  [Encerrar](#) [Clonar na AWS CLI](#) [Clonar](#)

► Resumo

[Propriedades](#) | [Ações de bootstrap](#) | [Instâncias \(hardware\)](#) | **[Etapas](#)** | [Aplicativos](#) | [Configurações](#) | [Monitoramento](#) | [Eventos](#) | [Tags \(0\)](#)

Etapas (1) [Informações](#)

[Atualizar tabela](#) [Etapas de cancelamento](#) [Clonar etapa](#) [Adicionar etapa](#)


Cada etapa é uma unidade de trabalho que contém instruções para manipular dados processados pelo software instalado no cluster.

Etapas simultâneas: 1 [✎](#)

Filtrar etapas por status ▼

🔍 Encontrar etapas

< 1 > ⚙️

<input type="checkbox"/>	ID da etapa ▼	Status ▼	Nome ▼	Arquivos de log 🔗	Horário de criação (UTC-03:00) ▼
<input type="checkbox"/>	s-07575271Y6AV0MPWPRRX	✖ Failed	etapa-treinamento	controller syslog stderr stdout 	November 28, 2024 at 03:38

- 1 instância master – m5.xlarge
- 3 instâncias worker – m5.xlarge
- Uso do S3
- m5.xlarge:
 - 2 vCPUs;
 - 16gb de RAM;

Configurações AWS - S3

build/

Copiar URI do S3

Objetos

Propriedades

Objetos (4) Informações

 Copiar URI do S3 Copiar URL Fazer download Abrir Excluir Ações Criar pasta Carregar

Os objetos são as entidades fundamentais armazenadas no Amazon S3. Você pode usar o [inventário do Amazon S3](#) para obter uma lista de todos os objetos em seu bucket. Para outras pessoas acessarem seus objetos, você precisará conceder permissões explicitamente a eles. [Saiba mais](#)

Localizar objetos por prefixo

Mostrar versões

< 1 > ⚙

<input type="checkbox"/>	Nome ▲	Tipo ▼	Última modificação ▼	Tamanho ▼	Classe de armazenamento ▼
<input type="checkbox"/>	input/	Pasta	-	-	-
<input type="checkbox"/>	logs/	Pasta	-	-	-
<input type="checkbox"/>	outputs/	Pasta	-	-	-
<input type="checkbox"/>	train_script.py	py	28 Nov 2024 03:25:12 AM -03	4.4 KB	Padrão

- Armazenamento de:
 - Input: Dados;
 - Logs: Logs;
 - Outputs: Modelo;
 - Script;

Configurações AWS - Instâncias

Instâncias (4) [Informações](#)

Última atualização
less than a minute atrás



Conectar

Estado da instância ▼

Ações ▼

Executar instâncias ▼

Localizar Instância por atributo ou tag (case-sensitive)

Todos os ... ▼

< 1 > ⚙

<input type="checkbox"/>	Name	ID da instância	Estado da inst... ▼	Tipo de inst... ▼	Verificação de status	Status do alarm	Zona de dispon... ▼	DNS IP
<input type="checkbox"/>		i-054be5849835cb8af	⌚ Pendente	m5.xlarge	⌚ Inicializando	Exibir alarmes +	us-east-1b	ec2-44
<input type="checkbox"/>		i-0170dabc3c7d3eb1a	⌚ Pendente	m5.xlarge	⌚ Inicializando	Exibir alarmes +	us-east-1b	ec2-3-
<input type="checkbox"/>		i-033933d3010b0339f	⌚ Pendente	m5.xlarge	⌚ Inicializando	Exibir alarmes +	us-east-1b	ec2-34
<input type="checkbox"/>		i-00f298fe4aff61052	⌚ Pendente	m5.xlarge	⌚ Inicializando	Exibir alarmes +	us-east-1b	ec2-44



Obrigado!
Dúvidas?

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/@MarinAglil/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>