Roteiro Modelos em Produção

Neste roteiro vamos ver algumas definições para execução de modelos em produção. Teremos acesso à modelos via API e estes conceitos, podem ser aplicados em cenários com Docker (contêiner).

**Uma etapa anterior: validação dos dados**

Para esta tarefa, vamos utilizar uma ferramenta do TensorFlow, chamada TFDV – TensorFlow Dava Validation. Para a validação, vamos informar o arquivo contendo os dados e depois explorar diretamente na ferramenta.

Instalar a biblioteca

!pip install tensorflow-data-validation

Executar os comandos

import tensorflow\_data\_validation as tfdv

stats = tfdv.generate\_statistics\_from\_csv(data\_location='data.csv', delimiter=',') #Usar aqui o arquivo .csv que usamos na primeira aula!

tfdv.visualize\_statistics(stats)

stats.datasets[0].features[7].string\_stats.rank\_histogram

**Acessando modelo via API**

Abrir PowerShell do Windows

Acessar uma pasta do seu computador, como por exemplo:

C:/Users/<usuário>/Documents

Você pode criar uma pasta pelo próprio Shell ou manualmente

mkdir resnet\_api

Acessa a pasta

cd resnet\_api

Executar o comando abaixo para instalar os pacotes necessários (importante ter o Python instalado na máquina)

pip install torch torchvision fastapi uvicorn opencv-python-headless pillow prometheus\_client

Na pasta resnet\_api, criar um arquivo Python com o nome api.py

Comandos dentro do arquivo:

from fastapi import FastAPI, UploadFile, File, Response

import torch

import torchvision.transforms as transforms

from torchvision import models

from PIL import Image

import io

from prometheus\_client import Counter, Summary, generate\_latest, CONTENT\_TYPE\_LATEST

# Criar a API

app = FastAPI()

# Carregar modelo ResNet-18 pré-treinado

model = models.resnet18(pretrained=True)

model.eval()

# Transformação de imagem

transform = transforms.Compose([

transforms.Resize((224, 224)),

transforms.ToTensor()

])

# Definir métricas

PREDICTION\_COUNT = Counter("inferences\_total", "Número total de inferências")

PREDICTION\_TIME = Summary("inference\_duration\_seconds", "Duração da inferência")

@app.get("/")

def home():

return {"message": "API ResNet-18 funcionando!"}

# Endpoint para métricas do Prometheus

@app.get("/metrics")

def get\_metrics():

return Response(generate\_latest(), media\_type=CONTENT\_TYPE\_LATEST)

# Endpoint de inferência

@app.post("/predict/")

async def predict(file: UploadFile = File(...)):

start\_time = time.time()

PREDICTION\_COUNT.inc()

image\_bytes = await file.read()

image = Image.open(io.BytesIO(image\_bytes)).convert("RGB")

# Aplicar transformações

image = transform(image).unsqueeze(0)

# Fazer predição

with torch.no\_grad():

output = model(image)

prediction = torch.argmax(output, dim=1).item()

duration = time.time() - start\_time

PREDICTION\_TIME.observe(duration)

return {"prediction": prediction, "duration":duration}

Este arquivo vai ser usado para instanciar a API e solicitar a inferência ao modelo.

Vamos rodar localmente a API e testar o acesso e a execução

No PowerShell:

uvicorn api:app --host 0.0.0.0 --port 8000

Para acessar a página via Browser, vamos utilizar http://127.0.0.1:8000. Por lá, vamos testar a inferência enviando uma imagem para o modelo, utilizando o /predict do arquivo API.py

Além do Browser, a inferência pode ser solicitada via Python:

import requests

# Enviar uma imagem para a API

image\_path = “caminho/para/imagem”

image\_data = requests.get(image\_path).content

response = requests.post(

"http://127.0.0.1:8000/predict/",

files={"file": ("image.jpg", image\_data, "image/jpeg")}

)

print(response.json()) # Exibe a predição

**Monitoramento**

Para monitoramento utilizando Prometheus, primeiro precisamos de instalar (download):

Link: <https://prometheus.io/download/>

Após download, a pasta pode ser extraída e armazenada para um local de fácil acesso (por exemplo, mesma pasta já utilizada anteriormente C:/Users/<usuário>/Documents/resnet\_api

Será preciso configurar o arquivo **prometheus.yml** que existe na pasta.

Adicionar:

global:

scrape\_interval: 5s

scrape\_configs:

- job\_name: 'My\_API'

static\_configs:

- targets: ['localhost:8000'] #URL da sua API FastAPI (a mesma utilizada anteriormente)

Executar o arquivo prometheus.exe, acessar localhost:9090 e verificar as métricas, como por exemplo, request HTTP que são enviadas via API e as configuradas na api.py, como **inferences\_total**.

**Monitoramento**

Visualização em Dashboard

A última etapa é a integração de uma ferramenta de visualização, como o Grafana, ou outro qualquer que receba as métricas. Para isso, vamos precisar:

Download Grafana, via link: <https://grafana.com/grafana/download?platform=windows>

Após instalar, execute no Browser: <http://localhost:3000>

Um login padrão será solicitado, login: admin e senha: admin. Após primeiro acesso vai ser solicitado que troque a senha. Ao final, você estará dentro da plataforma.

A configuração desta ferramenta é direta e bastante simples:

1. Configuração do Source Data
2. Configuração do Painel com as visualizações necessárias

**Monitoramento de Drift**

Um pacote que pode ser utilizado é o **evidently**, em python. Nele um conjunto de dado é comparado com outro, em todas as colunas/*features* e então um índice é calculado, informando se houve ou não **drift**.

Para a execução:

Em qualquer ambiente python, por exemplo o Colab:

import pandas as pd

from evidently.report import Report

from evidently.metric\_preset import DataDriftPreset

# Carregar os dados

df\_treinamento = pd.read\_csv("dados\_treinamento.csv")

df\_producao = pd.read\_csv("novos\_dados.csv")

# Criar o relatório de drift

report = Report(metrics=[DataDriftPreset()])

# Alimentar os dados de referência (treinamento) e os novos dados (produção)

report.run(reference\_data=df\_treinamento, current\_data=df\_producao)

# Salvar o relatório em HTML

report.save\_html("relatorio\_drift.html")

print("Relatório gerado: 'relatorio\_drift.html'")

A análise é feita diretamente no arquivo HTML apresentado. Basta fazer download (caso esteja no Colab) e/ou abrir diretamente no seu navegador padrão.