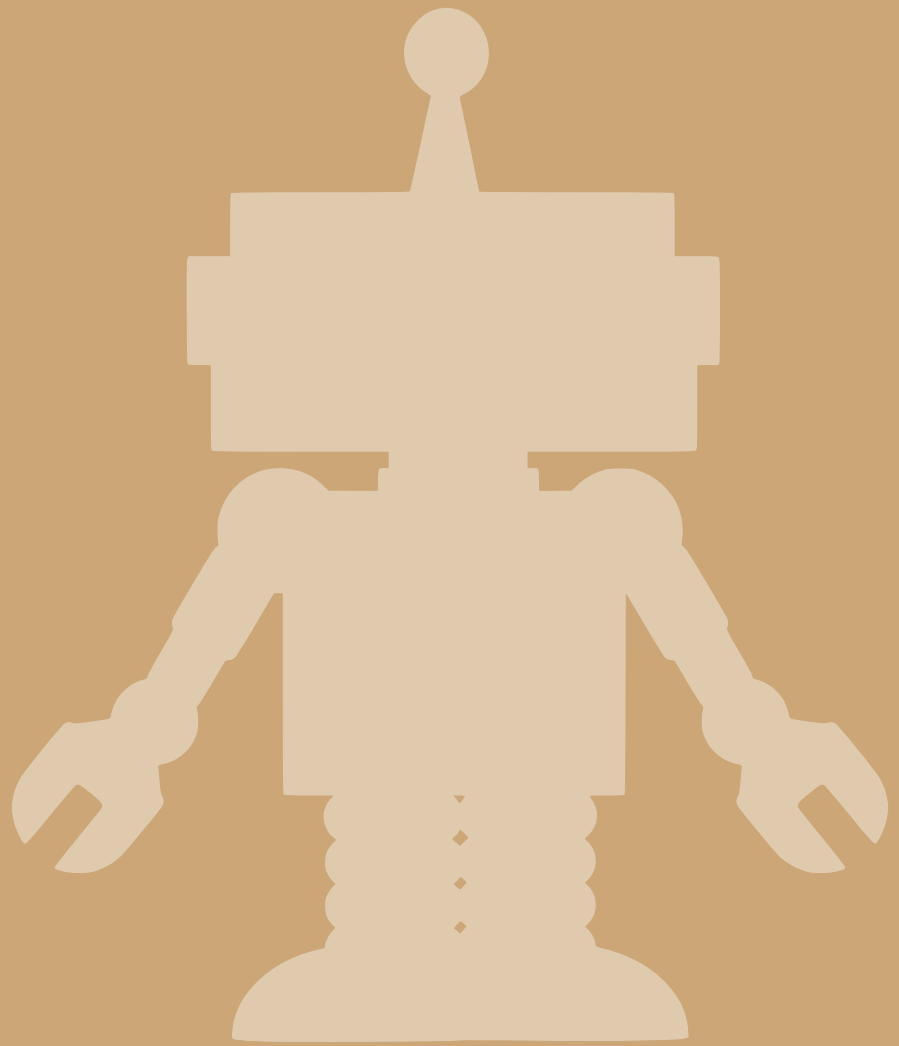
Aprendizado de Máquina 2

Exercício – Árvores de Decisão e Ensembles

Professora: Patrícia Pampanelli

patricia.pampanelli@usp.br



Objetivos

Preparação dos Dados: Preparar os dados para treinamento e avaliação de modelos de aprendizado de máquina.

Implementação de Modelos: Implementar modelos de aprendizado de máquina utilizando bibliotecas populares como Scikit-learn e XGBoost.

Avaliação e Ajustes: Avaliar o desempenho dos modelos e realizar ajustes necessários para melhorar a precisão.

Dockerização e Deploy: Dockerizar a solução para facilitar a implantação em diferentes ambientes e deployar o modelo utilizando Flask e uma imagem Docker.

Inferência e Cliente: Criar um notebook cliente que realize inferências direto do servidor onde o modelo foi deployado.

Roteiro do Exercício

1. **Treinamento do Modelo Baseado em Árvore de Decisão:** Treinar um modelo de árvore de decisão utilizando o conjunto de dados MNIST.
2. **Avaliação dos Ganhos com a Utilização de Modelos Ensemble:** Avaliar os ganhos obtidos ao utilizar modelos Ensemble em comparação com o modelo de árvore de decisão simples.
3. **Visualização da Árvore de Decisão e Medida de Impureza:** Explorar a estrutura da árvore de decisão e entender como a medida de impureza é utilizada para avaliar a qualidade das divisões nos nós da árvore.
4. **Preparação do Container para Deploy do Modelo:** Preparar um container Docker para deploy do modelo de árvore de decisão treinado.
5. **Deploy do Modelo Usando Flask Utilizando uma Imagem Docker:** Deployar o modelo de árvore de decisão treinado utilizando o framework Flask e uma imagem Docker.
6. **Notebook Cliente com Inferência Direto do Servidor:** Criar um notebook cliente que realize inferências direto do servidor onde o modelo de árvore de decisão foi deployado.

Componentes básicos do projeto

Arquivos e Pastas

Estrutura básica do projeto

```
project/
|— images/
|        └— test_images/
|— models/
|        └— trained_model.pkl
|— Dockerfile
|— main.py
|— training_notebook.ipynb
|— client_notebook.ipynb
|— requirements.txt
```

images
- Contém imagens de teste para o modelo.

models/
- Armazena os modelos treinados e persistidos.

Dockerfile
- Define as instruções para construir a imagem Docker do serviço do servidor.

main.py
- Contém a aplicação servidor construída com Flask API.

training_notebook.ipynb
- Utilizado para treinar, experimentar e validar o modelo.

client_notebook.ipynb
- Utilizado para realizar inferências no servidor.

requirements.txt
- Lista os pacotes necessários para a aplicação.

Desenvolvimento do Projeto

Passo a Passo

Roteiro do Exercício

1. ~~**Treinamento do Modelo Baseado em Árvore de Decisão:** Treinar um modelo de árvore de decisão/XGBoost utilizando o conjunto de dados MNIST.~~
2. **Avaliação dos Ganhos com a Utilização de Modelos Ensemble:** Avaliar os ganhos obtidos ao utilizar modelos Ensemble em comparação com o modelo de árvore de decisão simples.
3. **Visualização da Árvore de Decisão e Medida de Impureza:** Explorar a estrutura da árvore de decisão e entender como a medida de impureza é utilizada para avaliar a qualidade das divisões nos nós da árvore.
4. **Preparação do Container para Deploy do Modelo:** Preparar um container Docker para deploy do modelo de árvore de decisão treinado.
5. **Deploy do Modelo Usando Flask Utilizando uma Imagem Docker:** Deployar o modelo de árvore de decisão treinado utilizando o framework Flask e uma imagem Docker.
6. **Notebook Cliente com Inferência Direto do Servidor:** Criar um notebook cliente que realize inferências direto do servidor onde o modelo de árvore de decisão foi deployado.

Roteiro do Exercício

1. ~~**Treinamento do Modelo Baseado em Árvore de Decisão:** Treinar um modelo de árvore de decisão/XGBoost utilizando o conjunto de dados MNIST.~~
2. ~~**Avaliação dos Ganhos com a Utilização de Modelos Ensemble:** Avaliar os ganhos obtidos ao utilizar modelos Ensemble em comparação com o modelo de árvore de decisão simples.~~
3. **Visualização da Árvore de Decisão e Medida de Impureza:** Explorar a estrutura da árvore de decisão e entender como a medida de impureza é utilizada para avaliar a qualidade das divisões nos nós da árvore.
4. **Preparação do Container para Deploy do Modelo:** Preparar um container Docker para deploy do modelo de árvore de decisão treinado.
5. **Deploy do Modelo Usando Flask Utilizando uma Imagem Docker:** Deployar o modelo de árvore de decisão treinado utilizando o framework Flask e uma imagem Docker.
6. **Notebook Cliente com Inferência Direto do Servidor:** Criar um notebook cliente que realize inferências direto do servidor onde o modelo de árvore de decisão foi deployado.

Roteiro do Exercício

1. ~~**Treinamento do Modelo Baseado em Árvore de Decisão:** Treinar um modelo de árvore de decisão/XGBoost utilizando o conjunto de dados MNIST.~~
2. ~~**Avaliação dos Ganhos com a Utilização de Modelos Ensemble:** Avaliar os ganhos obtidos ao utilizar modelos Ensemble em comparação com o modelo de árvore de decisão simples.~~
3. ~~**Visualização da Árvore de Decisão e Medida de Impureza:** Explorar a estrutura da árvore de decisão e entender como a medida de impureza é utilizada para avaliar a qualidade das divisões nos nós da árvore.~~
4. **Preparação do Container para Deploy do Modelo:** Preparar um container Docker para deploy do modelo de árvore de decisão treinado.
5. **Deploy do Modelo Usando Flask Utilizando uma Imagem Docker:** Deployar o modelo de árvore de decisão treinado utilizando o framework Flask e uma imagem Docker.
6. **Notebook Cliente com Inferência Direto do Servidor:** Criar um notebook cliente que realize inferências direto do servidor onde o modelo de árvore de decisão foi deployado.

Exercício 4 – Preparação do Container para deploy do modelo

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . /app/

EXPOSE 8000

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Roteiro do Exercício

1. ~~**Treinamento do Modelo Baseado em Árvore de Decisão:** Treinar um modelo de árvore de decisão/XGBoost utilizando o conjunto de dados MNIST.~~
2. ~~**Avaliação dos Ganhos com a Utilização de Modelos Ensemble:** Avaliar os ganhos obtidos ao utilizar modelos Ensemble em comparação com o modelo de árvore de decisão simples.~~
3. ~~**Visualização da Árvore de Decisão e Medida de Impureza:** Explorar a estrutura da árvore de decisão e entender como a medida de impureza é utilizada para avaliar a qualidade das divisões nos nós da árvore.~~
4. ~~**Preparação do Container para Deploy do Modelo:** Preparar um container Docker para deploy do modelo de árvore de decisão treinado.~~
5. **Deploy do Modelo Usando Flask Utilizando uma Imagem Docker:** Deployar o modelo de árvore de decisão treinado utilizando o framework Flask e uma imagem Docker.
6. **Notebook Cliente com Inferência Direto do Servidor:** Criar um notebook cliente que realize inferências direto do servidor onde o modelo de árvore de decisão foi deployado.

Exercício 5 - Deploy do Modelo Usando Flask Utilizando uma Imagem Docker

main.py > predict

```python
from fastapi import FastAPI, File
from pydantic import BaseModel
import xgboost as xgb
import numpy as np
import pickle
import warnings

import base64
from PIL import Image
import io

warnings.simplefilter(action='ignore', category=DeprecationWarning)

app = FastAPI()

# Definição dos tipos de dados
class PredictionResponse(BaseModel):
    prediction: float

class ImageRequest(BaseModel):
    image: str

```

Exercício 5 - Deploy do Modelo Usando Flask Utilizando uma Imagem Docker

```
23  # Carregamento do Modelo de Machine Learning
24  def load_model():
25      global xgb_model_carregado
26      with open("xgb_model.pkl", "rb") as f:
27          xgb_model_carregado = pickle.load(f)
28
29  # Inicialização da Aplicação
30  @app.on_event("startup")
31  async def startup_event():
32      load_model()
```

Exercício 5 - Deploy do Modelo Usando Flask Utilizando uma Imagem Docker

```python
# Definição do endpoit /predict que aceita as requisições via POST
# Esse endpoit que irá receber a imagem em base64 e irá convertê-la para fazer inferência
@app.post("/predict", response_model=PredictionResponse)
async def predict(request: ImageRequest):
    # Processamento da Imagem
    img_bytes = base64.b64decode(request.image)
    img = Image.open(io.BytesIO(img_bytes))
    img = img.resize((8, 8))
    img_array = np.array(img)

    # Converter a imagem para escala de cinza
    img_array = np.dot(img_array[...,:3], [0.2989, 0.5870, 0.1140])

    img_array = img_array.reshape(1, -1)

    # Predição do Modelo de Machine Learning
    prediction = xgb_model_carregado.predict(img_array)

    return {"prediction": prediction}
```

Exercício 5 - Deploy do Modelo Usando Flask Utilizando uma Imagem Docker

```python
# Endpoint de Healthcheck
@app.get("/healthcheck")
async def healthcheck():
    # retorna um objeto com um campo status com valor "ok" se a aplicação estiver funcionando corretamente
    return {"status": "ok"}
```

Exercício 5 - Deploy do Modelo Usando Flask Utilizando uma Imagem Docker

Nesta etapa vamos criar uma imagem Docker que contenha a aplicação e suas dependências.

Isso permite que a aplicação seja executada de forma isolada e portátil em qualquer ambiente que suporte Docker, sem a necessidade de instalar dependências ou configurar o ambiente. Além disso, a imagem pode ser facilmente compartilhada e reutilizada em diferentes ambientes.

Para buildar a imagem docker através do terminal dentro do codespace:

```
docker build -t my-app-fastapi .
```

Exercício 4 – Preparação do Container para deploy do modelo

```
PROBLEMS    OUTPUT    TERMINAL    PORTS 3

See 'docker buildx build --help'.
@patriciapampanelli ➜ /workspaces/mba-deeplearning-iad-006-2024 (main) $ docker build -t my-app-fastapi .
[+] Building 3.2s (11/11) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 245B
 => [internal] load metadata for docker.io/library/python:3.9-slim
 => [auth] library/python:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/5] FROM docker.io/library/python:3.9-slim@sha256:1e3437daae1d9cebce372794eacfac254dd108200e47c8b7f56a633ebd3e2a0a
 => [internal] load build context
 => => transferring context: 1.61MB
 => CACHED [2/5] WORKDIR /app
 => CACHED [3/5] COPY requirements.txt .
 => CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt
 => [5/5] COPY . /app/
 => exporting to image
 => => exporting layers
 => => writing image sha256:dcaeae680cd869e7805729d7648c384055e7429eaf3c044c867ec90e8217c20f
 => => naming to docker.io/library/my-app-fastapi
@patriciapampanelli ➜ /workspaces/mba-deeplearning-iad-006-2024 (main) $
```

Roteiro do Exercício

1. **Treinamento do Modelo Baseado em Árvore de Decisão:** Treinar um modelo de árvore de decisão/XGBoost utilizando o conjunto de dados MNIST.
2. **Avaliação dos Ganhos com a Utilização de Modelos Ensemble:** Avaliar os ganhos obtidos ao utilizar modelos Ensemble em comparação com o modelo de árvore de decisão simples.
3. **Visualização da Árvore de Decisão e Medida de Impureza:** Explorar a estrutura da árvore de decisão e entender como a medida de impureza é utilizada para avaliar a qualidade das divisões nos nós da árvore.
4. **Preparação do Container para Deploy do Modelo:** Preparar um container Docker para deploy do modelo de árvore de decisão treinado.
5. **Deploy do Modelo Usando Flask Utilizando uma Imagem Docker:** Deployar o modelo de árvore de decisão treinado utilizando o framework Flask e uma imagem Docker.
6. **Notebook Cliente com Inferência Direto do Servidor:** Criar um notebook cliente que realize inferências direto do servidor onde o modelo de árvore de decisão foi deployado.

Exercício 6 – Executando a Imagem Docker

Nesta etapa vamos executar a imagem Docker que contenha a aplicação.

Para executar a imagem docker através do terminal dentro do codespace:

```
docker run -p 8000:8000 my-app-fastapi
```

```
@patriciapampanelli ➜ /workspaces/mba-deeplearning-iad-006-2024 (main) $ docker run -p 8000:8000 my-app-fastapi
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     172.17.0.1:56518 - "POST /predict HTTP/1.1" 200 OK
INFO:     172.17.0.1:38680 - "POST /predict HTTP/1.1" 200 OK
INFO:     172.17.0.1:38686 - "POST /predict HTTP/1.1" 200 OK
INFO:     172.17.0.1:36608 - "POST /predict HTTP/1.1" 200 OK
INFO:     172.17.0.1:47648 - "POST /predict HTTP/1.1" 200 OK
```

Exercício 6 - Notebook Cliente com Inferência Direto do Servidor

```
import base64
from PIL import Image
import requests
import io

img = Image.open('imagens/imagem_positiva.png')
img = img.resize((8, 8))

buffered = io.BytesIO()
img.save(buffered, format="PNG")
img_str = base64.b64encode(buffered.getvalue()).decode('utf-8')

url = 'http://localhost:8000/predict'
response = requests.post(url, json={'image': img_str})

print(response.json())
```

Roteiro do Exercício

1. **Treinamento do Modelo Baseado em Árvore de Decisão:** Treinar um modelo de árvore de decisão/XGBoost utilizando o conjunto de dados MNIST.
2. **Avaliação dos Ganhos com a Utilização de Modelos Ensemble:** Avaliar os ganhos obtidos ao utilizar modelos Ensemble em comparação com o modelo de árvore de decisão simples.
3. **Visualização da Árvore de Decisão e Medida de Impureza:** Explorar a estrutura da árvore de decisão e entender como a medida de impureza é utilizada para avaliar a qualidade das divisões nos nós da árvore.
4. **Preparação do Container para Deploy do Modelo:** Preparar um container Docker para deploy do modelo de árvore de decisão treinado.
5. **Deploy do Modelo Usando Flask Utilizando uma Imagem Docker:** Deployar o modelo de árvore de decisão treinado utilizando o framework Flask e uma imagem Docker.
6. **Notebook Cliente com Inferência Direto do Servidor:** Criar um notebook cliente que realize inferências direto do servidor onde o modelo de árvore de decisão foi deployado.