

***INTELIGÊNCIA ARTIFICIAL***

**UA02 / LABORATÓRIO # 6 / Mateus**

**Código**

**Importação de Bibliotecas**

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from sentence\_transformers import SentenceTransformer

import pdfplumber

import re

import spacy

from nltk.corpus import stopwords

import chromadb

import nltk

import csv

import pandas as pd

Estas são as bibliotecas necessárias para o funcionamento do código:

- `RecursiveCharacterTextSplitter`: Para dividir textos em pedaços (chunks)

- `SentenceTransformer`: Modelo para gerar embeddings de texto

- `pdfplumber`: Para extrair texto de PDFs

- `re`: Para operações com expressões regulares

- `spacy`: Para processamento de linguagem natural

- `stopwords`: Lista de palavras comuns a serem removidas

- `chromadb`: Banco de dados vetorial

- `nltk`, `csv`, `pandas`: Para manipulação de dados

**Funções para Leitura de Arquivos**

**1.) `ler\_pdf(caminho\_pdf)`**

def ler\_pdf(caminho\_pdf):

with pdfplumber.open(caminho\_pdf) as leitor\_pdf:

texto = "".join([pagina.extract\_text() for pagina in leitor\_pdf.pages])

leitor\_pdf.close()

return texto.replace("\n", " ")

- Lê um arquivo PDF e extrai o texto de todas as páginas

- Concatena o texto de todas as páginas em uma única string

- Remove quebras de linha

**2.) `ler\_csv(caminho\_csv)`**

def ler\_csv(caminho\_csv):

try:

df = pd.read\_csv(caminho\_csv)

return df

except Exception as e:

print(f"Erro ao ler CSV: {e}")

return ""

- Lê um arquivo CSV usando pandas

- Retorna um DataFrame ou string vazia em caso de erro

**3.) `processar\_linhas\_csv(caminho\_csv, coluna\_texto)`**

def processar\_linhas\_csv(caminho\_csv, coluna\_texto):

df = pd.read\_csv(caminho\_csv)

textos\_tratados = []

for texto in df[coluna\_texto]:

texto\_processado = tratamento\_pln(str(texto))

textos\_tratados.append(texto\_processado)

return textos\_tratados

- Processa uma coluna específica de um CSV

- Aplica tratamento de PLN (Processamento de Linguagem Natural) a cada texto

- Retorna lista de textos processados

**4.) `combinar\_colunas\_csv(caminho\_csv, colunas)`**

def combinar\_colunas\_csv(caminho\_csv, colunas):

df = pd.read\_csv(caminho\_csv).to\_dict("records")

texto\_combinado = ""

for obj in df:

for coluna in colunas:

texto\_combinado += f"{obj[coluna]} "

texto\_combinado += "\n"

return texto\_combinado

- Combina valores de múltiplas colunas de um CSV em um único texto

- Cada linha do CSV vira uma linha no texto combinado

**Funções de Processamento de Texto**

**5.) `tratamento\_pln(texto)`**

def tratamento\_pln(texto):

nlp = spacy.load("pt\_core\_news\_sm")

stop\_words = set(stopwords.words('portuguese'))

texto = texto.lower()

texto = re.sub(r'[^a-zA-Záéíóú\s]', '', texto)

doc = nlp(texto)

clean\_tokens = [token.lemma\_ for token in doc

if token.text not in stop\_words and not token.is\_punct]

return ' '.join(clean\_tokens)

- Carrega modelo de PLN em português e stopwords

- Converte texto para minúsculas

- Remove números, pontuações e caracteres especiais

- Tokeniza, lematiza e remove stopwords/pontuações

- Junta os tokens limpos em uma string

**6.) `criar\_chunks(texto, tamanho=30, overlap=10)`**

def criar\_chunks(texto, tamanho=30, overlap=10):

splitter = RecursiveCharacterTextSplitter(

chunk\_size=tamanho,

chunk\_overlap=overlap

)

return splitter.split\_text(texto)

- Divide o texto em pedaços (chunks) com sobreposição

- `chunk\_size`: tamanho máximo de cada pedaço

- `chunk\_overlap`: sobreposição entre pedaços consecutivos

**Funções de Banco de Dados Vetorial**

**7.) `criar\_banco\_vetorial(nome\_colecao="colecao\_teste")`**

def criar\_banco\_vetorial(nome\_colecao="colecao\_teste"):

model = SentenceTransformer('all-MiniLM-L6-v2')

client = chromadb.Client()

try:

collection = client.get\_collection(name=nome\_colecao)

return collection, model

except Exception:

collection = client.create\_collection(name=nome\_colecao)

return collection, model

- Cria ou acessa uma coleção no ChromaDB

- Inicializa modelo de embeddings (all-MiniLM-L6-v2)

- Retorna a coleção e o modelo

**8.) `adicionar\_chunks(chunks, collection, model)`**

def adicionar\_chunks(chunks, collection, model):

embeddings = model.encode(chunks)

ids = [f"doc\_{i}" for i in range(len(chunks))]

collection.add(

documents=chunks,

embeddings=embeddings.tolist(),

ids=ids

)

return

- Converte chunks em embeddings

- Gera IDs sequenciais para os documentos

- Adiciona documentos e embeddings à coleção

**9.) `consultar\_banco(colecao, modelo, consulta, n\_resultados=1)`**

def consultar\_banco(colecao, modelo, consulta, n\_resultados=1):

embedding\_consulta = modelo.encode([consulta])

resultados = colecao.query(

query\_embeddings=embedding\_consulta.tolist(),

n\_results=n\_resultados

)

return resultados

- Converte a consulta em embedding

- Busca os n documentos mais similares na coleção

- Retorna os resultados

**10.) `ExibirResultados(resultados)`**

def ExibirResultados(resultados):

for i in range(len(resultados['ids'][0])):

print(f"ID: {resultados['ids'][0][i]}")

print(f"Documento: {resultados['documents'][0][i]}")

print(f"Distância: {resultados['distances'][0][i]}")

print("-" \* 40)

- Formata e exibe os resultados de uma consulta

- Mostra ID, conteúdo do documento e distância (similaridade)

**11.) Fluxo Principal**

O código executa dois fluxos principais: um para CSV e outro para PDF.

**Fluxo CSV:**

1.) Extrai e combina colunas do CSV ("First Name", "Last Name", "Job Title")

2.) Pré-processa o texto combinado

3.) Divide em chunks (tamanho 40, overlap 10)

4.) Cria banco vetorial "peolple" (typo no nome)

5.) Adiciona chunks ao banco

6.) Consulta por "Game Developer" e mostra 3 resultados

**Saída CSV:**

ID: doc\_0

Documento: shelby terrell game developer

Distância: 0.7823794484138489

----------------------------------------

ID: doc\_39

Documento: service engineer

Distância: 1.2176907062530518

----------------------------------------

ID: doc\_67

Documento: agent

Distância: 1.2247307300567627

----------------------------------------

- O primeiro resultado é o mais relevante (menor distância)

- "shelby terrell game developer" contém exatamente o termo buscado

- Os outros resultados são menos relevantes

**Fluxo PDF:**

1.) Extrai texto do PDF "os\_3\_porquinhos.pdf"

2.) Pré-processa o texto

3.) Divide em chunks (tamanho 100, overlap 30)

4.) Cria banco vetorial "tres\_porquinhos"

5.) Adiciona chunks ao banco

6.) Consulta por "lobo mal assoprou" e mostra 3 resultados

**Saída PDF:**

ID: doc\_47

Documento: so saboroso abrir porta ir derrubar tudo lar vigoroso gritar luizinho lobo soprar soprar

Distância: 1.0531325340270996

----------------------------------------

ID: doc\_35

Documento: brincadeira saber assim estar seguro contra lobo mau após algum dia casa joozinho luizinho pron to

Distância: 1.1412612199783325

----------------------------------------

ID: doc\_55

Documento: teto porquinho colocar fogo lareira fa zer labareda queimar rabo lobo auuu uivar dor lobo correr

Distância: 1.2308220863342285

----------------------------------------

- Todos os resultados mencionam "lobo" de alguma forma

- O primeiro resultado menciona "lobo soprar soprar" que é mais próximo da consulta

- As distâncias mostram a similaridade decrescente

**Conclusão**

Este código é um pipeline completo para:

1.) Extrair texto de diferentes fontes (PDF, CSV)

2.) Processar e limpar o texto

3.) Dividir em pedaços adequados

4.) Armazenar em um banco de dados vetorial

5.) Permitir buscas por similaridade semântica

As saídas mostram que o sistema está funcionando corretamente, encontrando os trechos mais relevantes para as consultas feitas.