

▼ **Processamento de Linguagem Natural [2023.Q3]**

Prof. Alexandre Donizeti Alves

▼ **ATIVIDADE PRÁTICA 04 [Uso da API da OpenAI com técnicas de PLN]**

A **ATIVIDADE PRÁTICA 04** deve ser feita utilizando o **Google Colab** com uma conta sua vinculada ao Gmail. O link do seu notebook, armazenado no Google Drive, além do link de um repositório no GitHub e os principais resultados da atividade, devem ser enviados usando o seguinte formulário:

<https://forms.gle/GzwCq3R7ExtE9g9a8>

IMPORTANTE: A submissão deve ser feita até o dia 20/11 (segunda-feira) APENAS POR UM INTEGRANTE DA EQUIPE, até às 23h59. Por favor, lembre-se de dar permissão de ACESSO IRRESTRITO para o professor da disciplina de PLN.

▼ **EQUIPE**

POR FAVOR, PREENCHER OS INTEGRANDES DA SUA EQUIPE:

Integrante 01:

Por favor, informe o seu nome completo e RA: Pedro Victor Marcelino Jordão Motta RA: 11201921599

Integrante 02:

Por favor, informe o seu nome completo e RA: Luccas Vinicius de Faveri Tortorelli Cardoso 11201920991

▼ **LIVRO**

Processamento de Linguagem Natural - Conceitos, Técnicas e Aplicações em Português.

Disponível gratuitamente em:

<https://brasileiraspln.com/livro-pln/1a-edicao/>.

POR FAVOR, PREENCHER OS CAPITULOS SELECIONADOS PARA A SUA EQUIPE:

Primeiro capítulo: 8

▼ DESCRIÇÃO

Implementar um notebook no Google Colab que faça uso da **API da OpenAI** aplicando, no mínimo, 3 técnicas de PLN. As técnicas devem ser aplicadas nos 2 (DOIS) capítulos do livro **Processamento de Linguagem Natural - Conceitos, Técnicas e Aplicações em Português**.

RESTRIÇÃO: É obrigatório usar o *endpoint "Chat Completions"*.

As seguintes técnicas de PLN podem ser usadas:

- Correção Gramatical
- Classificação de Textos
- Análise de Sentimentos
- Detecção de Emoções
- Extração de Palavras-chave
- Tradução de Textos
- Sumarização de Textos
- **Similaridade de Textos**
- **Reconhecimento de Entidades Nomeadas**
- **Sistemas de Perguntas e Respostas**

Os capítulos devem ser os mesmos selecionados na **ATIVIDADE PRÁTICA 02**. Para consultar os capítulos, considere a seguinte planilha:

<https://docs.google.com/spreadsheets/d/1ZutzQ3v1OJgsgzCvCwxXIRIQ3ChXNIHNvB63JQvYsbo/edit?usp=sharing>

IMPORTANTE: É obrigatório usar o e-mail da UFABC. Não é permitido alterar os capítulos já selecionados.

▼ CRITÉRIOS DE AVALIAÇÃO

Serão considerados como critérios de avaliação as técnicas usadas e a criatividade envolvida na aplicação das mesmas.

▼ IMPLEMENTAÇÃO

▼ Instalação de Libs

```
# Instalação da biblioteca Faiss para operações eficientes de busca e análise em conjunto
# A versão '-cpu' indica que a instalação é para CPUs, apropriada quando não se utiliza a
!pip install faiss-cpu
```

```
Requirement already satisfied: faiss-cpu in /usr/local/lib/python3.10/dist-packages (
```



```
# Instalação da versão específica 0.28.1 da biblioteca OpenAI.
# Fixar a versão pode ser útil para manter consistência em projetos e evitar problemas de
!pip install openai==0.28.1
```

```
Requirement already satisfied: openai==0.28.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: requests>=2.20 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-pac
```



```
# Instalação da versão específica 0.0.329 da biblioteca Langchain.
# Fixar a versão pode ser útil para manter consistência em projetos e evitar problemas de
!pip install langchain==0.0.329
```

```
Requirement already satisfied: langchain==0.0.329 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: SQLAlchemy<3,>=1.4 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in /usr/local/lib/python3.10/dis
Requirement already satisfied: anyio<4.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: async-timeout<5.0.0,>=4.0.0 in /usr/local/lib/python3.
Requirement already satisfied: dataclasses-json<0.7,>=0.5.7 in /usr/local/lib/python3
Requirement already satisfied: jsonpatch<2.0,>=1.33 in /usr/local/lib/python3.10/dist
Requirement already satisfied: langsmith<0.1.0,>=0.0.52 in /usr/local/lib/python3.10/
Requirement already satisfied: numpy<2,>=1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pydantic<3,>=1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: requests<3,>=2 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: tenacity<9.0.0,>=8.1.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in /usr/local/lib/python3
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (
```

```
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: marshmallow<4.0.0,>=3.18.0 in /usr/local/lib/python3.1
Requirement already satisfied: typing-inspect<1,>=0.4.0 in /usr/local/lib/python3.10/
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.10/
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: packaging>=17.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: mypy-extensions>=0.3.0 in /usr/local/lib/python3.10/di
```

```
# Instalação da biblioteca tiktoken, que fornece uma contagem de tokens para textos no fo
!pip install tiktoken
```

```
Requirement already satisfied: tiktoken in /usr/local/lib/python3.10/dist-packages (0
Requirement already satisfied: regex>=2022.1.18 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: requests>=2.26.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-p
```



▼ Importação de todas as Libs para a tarefa

```
# Importação de bibliotecas necessárias para o script.
from bs4 import BeautifulSoup # Para realizar o parsing de HTML.
import requests # Para fazer requisições HTTP.
import re # Para manipulação de expressões regulares.
import time # Para manipulação de tempo.
import json # Para manipulação de dados em formato JSON.
import openai # Biblioteca OpenAI para tarefas de linguagem natural.
from google.colab import files # Para manipulação de arquivos no ambiente Google Colab.
import textwrap # Para facilitar a formatação de texto.
from langchain.text_splitter import CharacterTextSplitter # Módulo para dividir texto em
from langchain.embeddings.openai import OpenAIEmbeddings # Módulo para incorporar vetore
from langchain.vectorstores import FAISS # Módulo para armazenamento eficiente de vetore
from langchain.llms import OpenAI # Módulo para modelos de linguagem utilizando OpenAI.
import httpx # Cliente HTTP assíncrono.
```


▼ Definindo a chave da API

```
# fazer upload do arquivo de texto
upload_arquivo = files.upload()

# obter o nome do arquivo
nome_arquivo = list(upload_arquivo.keys())[0]

# ler o conteúdo do arquivo
with open(nome_arquivo, 'r') as file:
    chave_api = file.read()

# definir a chave da API
openai.api_key = chave_api
```

 Nenhum arquivo escolhido Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving apiOpenai.txt to apiOpenai (2).txt

▼ Funções para a tarefa de resumir

```
# Definição da função para obter texto de um item.
def getText(item):
    # Mensagem de início da captura do capítulo.
    print(f"Iniciando a captura do capítulo: {item[0]}")

    # Faz uma requisição HTTP para a URL do item e obtém o conteúdo HTML.
    response = requests.get(item[1])

    # Utiliza BeautifulSoup para fazer o parsing do conteúdo HTML.
    soup = BeautifulSoup(response.content, 'html.parser')

    # Encontra o elemento principal no HTML com a classe 'content' e o id 'quarto-document-content'
    main_element = soup.find('main', {'class': 'content', 'id': 'quarto-document-content'})

    # Verifica se o elemento principal foi encontrado.
    if main_element:
        # Obtém o texto do elemento principal em minúsculas e sem normalização.
        texto = main_element.get_text().lower()
        textoOfffNormalize = main_element.get_text()

    # Mensagem indicando sucesso na conexão e coleta do texto.
    print("Conexão e coleta do texto com sucesso")

    # Cria um dicionário com os dados do capítulo.
    json_data = {
        "Capitulo": item[0],
        "texto": texto,
        "textoOfffNormalize": textoOfffNormalize,
    }

    # Retorna o dicionário com os dados do capítulo.
    return json_data
```

```
# Definição da função para quebrar um texto em chunks.
def quebraTexto(text):
    # Tamanho máximo de cada chunk (em caracteres).
    tamanho_chunk = 16384
    chunks = [] # Lista para armazenar os chunks resultantes.
    chunk_atual = "" # String para construir o chunk atual.

    # Itera sobre as frases do texto.
    for sentence in text.split("."):
        # Verifica se adicionar a frase atual ao chunk não excede o tamanho máximo.
        if len(chunk_atual) + len(sentence) < tamanho_chunk:
            chunk_atual += sentence + "." # Adiciona a frase ao chunk atual.
        else:
            chunks.append(chunk_atual.strip()) # Adiciona o chunk atual à lista.
            chunk_atual = sentence + "." # Inicia um novo chunk com a frase atual.

    # Verifica se há algum chunk não processado ao final.
    if chunk_atual:
        chunks.append(chunk_atual.strip())

    # Retorna a lista de chunks.
    return chunks
```

```

# Definição da função para resumir um trecho de texto utilizando a API da OpenAI.
def resumir_trecho(texto, indice, max_retries=3, delay_between_retries=5):
    # Loop para realizar tentativas de resumo.
    for tentativa in range(max_retries + 1):
        chunk = texto # Inicializa o chunk com o texto fornecido.

        # Lista de mensagens a serem enviadas para a API da OpenAI.
        mensagens = []
        # Mensagem de sistema explicando o propósito da IA no resumo de texto.
        mensagens.append({'role': 'system', 'content': "Você é uma inteligência artificial"})
        # Mensagem do usuário contendo o chunk de texto a ser resumido.
        mensagens.append({'role': 'user', 'content': chunk})
        print(mensagens)

        # Chamada à API da OpenAI para obter um resumo do texto.
        response = openai.ChatCompletion.create(
            model="gpt-3.5-turbo-16k-0613",
            messages=mensagens,
            max_tokens=2048,
            temperature=0.4,
        )

        # Verifica se a resposta da API contém escolhas válidas.
        if "choices" in response and response["choices"]:
            summary = response["choices"][0]["message"]["content"]
            return summary # Retorna o resumo obtido.
        else:
            # Caso a tentativa não seja bem-sucedida, imprime mensagem de falha.
            print(f"Falha na tentativa {tentativa + 1}.")
            # Verifica se há mais tentativas disponíveis.
            if tentativa < max_retries:
                print(f"Tentando novamente em {delay_between_retries} segundos...")
                time.sleep(delay_between_retries) # Aguarda antes de tentar novamente.
            else:
                print("Número máximo de tentativas atingido. Abortando.")
                return None # Retorna None caso todas as tentativas falhem.

# Definição da função para visualizar resumos de trechos.
def visualizar_resumo(tuplaResumo):
    largura_maxima = 200 # Largura máxima para formatação do texto.

    # Itera sobre os resumos na tupla.
    for i, value in enumerate(tuplaResumo):
        # Quebra o texto do resumo em linhas de acordo com a largura máxima.
        texto_quebrado = textwrap.wrap(value[1], largura_maxima)

        # Imprime o número do trecho e o resumo formatado.
        print("Resumo do trecho", value[0], ":")
        for linha in texto_quebrado:
            print(linha)
        print('\n') # Adiciona uma linha em branco entre os resumos.

```

▼ Resgate do texto do capítulo 8 e 18

```
# Lista de URLs a serem processadas.
urls = [
    (8, "https://brasileiraspln.com/livro-pln/1a-edicao/parte5/cap8/cap8.html"),
    (18, "https://brasileiraspln.com/livro-pln/1a-edicao/parte8/cap18/cap18.html")
]

# Lista para armazenar os resultados da função getText para cada URL.
texto = []

# Itera sobre as URLs e obtém o texto de cada capítulo.
for i in urls:
    texto.append(getText(i))

    Iniciando o captura do capitulo:8
    Conexão e coleta do texto com sucesso
    Iniciando o captura do capitulo:18
    Conexão e coleta do texto com sucesso
```

▼ Aplicação da Tarefa de Resumo de texto

```
# Obtendo os chunks do texto do capítulo 8 usando a função quebraTexto.
chunks_8 = quebraTexto(texto[0]['textoOffNormalize'])

# Obtendo os chunks do texto do capítulo 18 usando a função quebraTexto.
chunks_18 = quebraTexto(texto[1]['textoOffNormalize'])

# Lista para armazenar os resumos do Capítulo 8.
resumo_Capitulo_8 = []

# Itera sobre os chunks do Capítulo 8 e obtém os resumos.
for i, value in enumerate(chunks_8):
    resumo = resumir_trecho(value, i)

    # Verifica se o resumo foi bem-sucedido.
    if resumo is not None:
        print(f"\n\nResumo do trecho {i} foi bem-sucedido. Segue o resumo: {resumo}\n\n")
        resumo_Capitulo_8.append((i, resumo))
    else:
        print(f"Falha no resumo do trecho {i}. Você pode tentar novamente ou lidar com o

[{'role': 'system', 'content': 'Você é uma inteligência artificial que irá resumir pe

Resumo do trecho 0 foi bem-sucedido. Segue o resumo: A semântica estuda o significado

[{'role': 'system', 'content': 'Você é uma inteligência artificial que irá resumir pe
```


Resumo do trecho 1 foi bem-sucedido. Segue o resumo: O termo "trabalho" possui divers

```
# Lista para armazenar os resumos do Capítulo 18.
resumo_Capitulo_18 = []

# Itera sobre os chunks do Capítulo 18 e obtém os resumos.
for i, value in enumerate(chunks_18):
    resumo = resumir_trecho(value, i)

    # Verifica se o resumo foi bem-sucedido.
    if resumo is not None:
        print(f"\n\nResumo do trecho {i} foi bem-sucedido. Segue o resumo: {resumo}\n\n")
        resumo_Capitulo_18.append((i, resumo))
    else:
        print(f"Falha no resumo do trecho {i}. Você pode tentar novamente ou lidar com o
```

[{'role': 'system', 'content': 'Você é uma inteligência artificial que irá resumir o conteúdo do capítulo 18 do livro "Tradução Automática: Fundamentos e Aplicações". O resumo deve ser conciso e focar nos pontos principais de cada trecho. O texto a ser resumido é o seguinte:'}]

Resumo do trecho 0 foi bem-sucedido. Segue o resumo: A tradução automática (TA) é uma tecnologia que permite a conversão de texto de um idioma para outro sem a necessidade de intervenção humana.

[{'role': 'system', 'content': 'Você é uma inteligência artificial que irá resumir o conteúdo do capítulo 18 do livro "Tradução Automática: Fundamentos e Aplicações". O resumo deve ser conciso e focar nos pontos principais de cada trecho. O texto a ser resumido é o seguinte:'}]

Resumo do trecho 1 foi bem-sucedido. Segue o resumo: A tradução automática estatística (TA estatística) é uma abordagem que utiliza modelos estatísticos para gerar traduções automáticas.

[{'role': 'system', 'content': 'Você é uma inteligência artificial que irá resumir o conteúdo do capítulo 18 do livro "Tradução Automática: Fundamentos e Aplicações". O resumo deve ser conciso e focar nos pontos principais de cada trecho. O texto a ser resumido é o seguinte:'}]

Resumo do trecho 2 foi bem-sucedido. Segue o resumo: A avaliação da Tradução Automática (TA) é um processo essencial para garantir a qualidade das traduções geradas automaticamente.

A avaliação da TA pode ser realizada de várias maneiras diferentes, devido à falta de consenso sobre a melhor abordagem. No entanto, existem algumas métricas comuns que são utilizadas para avaliar a qualidade das traduções automáticas.

A avaliação da TA tem uma longa história, com várias avaliações influentes ao longo dos anos. No entanto, a avaliação da TA continua a ser uma área de pesquisa ativa, com novos métodos sendo desenvolvidos constantemente.

A avaliação de TA é importante para todos os campos relacionados à tradução, mas é especialmente importante para a tradução automática. É importante que a avaliação de TA seja replicável e baseada em evidências. No entanto, a avaliação da TA é uma tarefa desafiadora, devido à subjetividade da avaliação humana.

Em resumo, a avaliação da TA é uma prática essencial para melhorar os sistemas de tradução automática. No entanto, a avaliação da TA é uma tarefa desafiadora, devido à subjetividade da avaliação humana.

[{'role': 'system', 'content': 'Você é uma inteligência artificial que irá resumir o conteúdo do capítulo 18 do livro "Tradução Automática: Fundamentos e Aplicações". O resumo deve ser conciso e focar nos pontos principais de cada trecho. O texto a ser resumido é o seguinte:'}]

Resumo do trecho 3 foi bem-sucedido. Segue o resumo: As métricas de avaliação automática (TA automática) são utilizadas para avaliar a qualidade das traduções geradas automaticamente.

A avaliação humana é essencial para uma compreensão mais detalhada e abrangente da qualidade das traduções automáticas. No entanto, a avaliação humana é uma tarefa desafiadora, devido à subjetividade da avaliação humana.

A adequação é uma métrica que avalia a fidelidade semântica entre o texto-fonte e o texto-alvo. No entanto, a adequação é uma métrica subjetiva, o que torna a sua avaliação uma tarefa desafiadora.

A pós-edição é uma ferramenta importante na avaliação de sistemas de TA, pois ofe

É importante considerar que a avaliação humana pode variar dependendo dos avalia

```
[{'role': 'system', 'content': 'Você é uma inteligência artificial que irá resum:
```

Resumo do trecho 4 foi bem-sucedido. Segue o resumo: A avaliação humana na avali

A avaliação dependente de contexto na TA tem se mostrado necessária para avaliar

O futuro da TA é promissor, com a tecnologia se tornando cada vez mais importante

Em resumo, a avaliação humana na avaliação de TA levanta questões metodológicas e

```
[{'role': 'system', 'content': 'Você é uma inteligência artificial que irá resum:
```

Função para visualizar os resumos do Capítulo 8.

```
visualizar_resumo(resumo_Capitulo_8)
```

Resumo do trecho 0 :

A semântica estuda o significado das palavras e frases na linguagem. No entanto, a de principais abordagens no estudo do significado: a representacionista/essencialista e uma essência comum nos diferentes usos da palavra. Já a abordagem pragmática entende essas diferentes perspectivas se refletem em técnicas simbólicas e representações dis palavras e seus significados em contextos específicos. Essas abordagens têm impacto n

Resumo do trecho 1 :

O termo "trabalho" possui diversas acepções de acordo com o dicionário. Essas acepção confecção de uma obra, o grande esforço, a ação contínua de uma força da natureza, o responsabilidade, o conjunto de atividades humanas empregadas na produção de bens e a significado das palavras enquanto estão sendo usadas. Estudos têm mostrado a dificuldade distribuídas tem sido uma abordagem promissora para lidar com o sentido das palavras, maneiras de trabalhar com o significado no Processamento de Linguagem Natural.

Função para visualizar os resumos do Capítulo 18.

```
visualizar_resumo(resumo_Capitulo_18)
```

Resumo do trecho 0 :

A tradução automática (TA) é a tradução de um texto eletrônico de uma língua para neurais. Atualmente, é amplamente utilizada em diversas aplicações de comunicação exemplos e estatística. A tradução direta mapeia palavras-fonte para palavras-alvo realizar mapeamentos mais complexos, considerando a sintaxe das línguas envolvidas tradução baseada em exemplos utiliza exemplos de traduções existentes para aprend corpora bilíngues.

Resumo do trecho 1 :

A tradução automática estatística baseada em frases (PBSMT) alinha fragmentos de melhor tradução. Esses modelos são treinados a partir de grandes quantidades de corpus paralelo. A PBSMT foi amplamente utilizada até a introdução da tradução automática baseada em sentença-fonte de uma vez e usando embeddings e modelos de atenção para melhorar o desempenho ruim em condições fora do domínio e para idiomas com recursos limitados. A qualidade da tradução.

Resumo do trecho 2 :

A avaliação da Tradução Automática (TA) é a prática de analisar a qualidade da tradução, e muitas vezes uma combinação das duas é utilizada. A avaliação de desempenho precisa de melhorias. A avaliação da TA pode ser realizada de várias maneiras e é dividida em avaliação "caixa de vidro" e "caixa-preta". A avaliação "caixa de vidro" do sistema, sem considerar seus mecanismos internos. A avaliação da TA tem uma longa história em sistemas de TA nos Estados Unidos e resultou em uma redução significativa no financiamento na avaliação de TA e utilizou julgamentos humanos para avaliar as traduções (2005) e QTLaunchPad (2012). A avaliação de TA é importante para todos os campos de qualidade. Além disso, a avaliação automática de TA é realizada por meio de métricas que têm sido aprimoradas ao longo dos anos, mas ainda apresentam limitações. É importante considerar desnecessária e alguns estudos utilizam apenas uma métrica para avaliar a avaliação e evitar promessas exageradas sobre a capacidade da TA. Em resumo, a avaliação de TA é feita com várias métricas para avaliar a qualidade da tradução, mas é importante considerar o contexto.

Resumo do trecho 3 :

As métricas de avaliação automática (MAAs) são usadas para medir a qualidade da tradução supervisionadas, que podem ser baseadas em word-embeddings ou contextual-embeddings. A avaliação humana é essencial para uma compreensão mais detalhada e abrangente do desempenho e anotação de erros. Além disso, métricas secundárias, como legibilidade, fluência, coerência e semântica entre o texto-fonte e a tradução. A fluência, por sua vez, avalia a naturalidade das traduções, enquanto a anotação de erros identifica e classifica os erros presentes. A avaliação humana é aprofundada do esforço envolvido no processo de tradução. O esforço pode ser medido em termos de dados de treinamento ou teste. É importante considerar que a avaliação humana pode ser demorada, e a escolha da abordagem de avaliação mais adequada é um ponto de reflexão.

Resumo do trecho 4 :

A avaliação humana na avaliação de Tradução Automática (TA) levanta várias questões sobre as competências dos avaliadores, a inclusão de tradutores, linguistas ou especialistas. Além disso, considerações pragmáticas, como o custo associado aos avaliadores e a geração de resultados, têm se mostrado necessária para avaliar sistemas neurais mais complexos e que levam em conta o contexto. A avaliação humana com contexto é necessária para uma análise mais abrangente. A avaliação humana em nível de documento. Estudos têm mostrado que a avaliação de sentenças é mais promissora, com a tecnologia se tornando cada vez mais importante para aprimorar a avaliação de sistemas neurais, mas ainda há desafios a serem superados, como preconceitos nos dados de treinamento.

▼ Funções para a tarefa de Similaridade de Textos

```
# Definição da função para dividir um texto em chunks e obter embeddings utilizando OpenAI
def splitter_OpenAIEmbeddings(text):
    # Configuração do text splitter para separar o texto por quebras de linha.
    text_splitter = CharacterTextSplitter(
        separator="\n",
        chunk_size=2500,
        chunk_overlap=200,
        length_function=len
    )

    # Divisão do texto em chunks.
    chunks = text_splitter.split_text(text)

    # Configuração e criação de embeddings utilizando OpenAIEmbeddings.
    embeddings = OpenAIEmbeddings(
        request_timeout=180,
        deployment='text-embedding-ada-002',
        chunk_size=10,
        openai_api_key=openai.api_key,
        show_progress_bar=True
    )

    return chunks, embeddings
```

```

# Definição da função para buscar trechos similares na base de conhecimento do Capítulo 8
def similarty_base_cap_8(query_user, k_user, chunks_8, embeddings_8, knowledge_base_8):
    # Obtém a representação vetorial da consulta do usuário.
    query = embeddings_8.embed_query(query_user)

    # Realiza a busca por similaridade na base de conhecimento do Capítulo 8.
    docs = knowledge_base_8.similarity_search_with_score_by_vector(query, k=k_user)

    # Lista para armazenar os resultados em formato JSON.
    json_data = []
    context = ''

    # Itera sobre os documentos retornados.
    for index, document in enumerate(docs):
        data = {
            "indice": index,
            "page_content": document[0].page_content,
            "score": float(document[1])
        }
        json_data.append(data)

    # Ordena os dados por score em ordem decrescente.
    sorted_data = sorted(json_data, key=lambda x: x["score"], reverse=True)

    print("\nOs trechos com a semelhança são os seguintes:\n\n")

    # Imprime os trechos similares ordenados por score.
    for i in sorted_data:
        print(f"O Trecho {i['indice']} com o score {i['score']} com o seguinte conteúdo:\n")
        print('\n\n')

```

```

# Definição da função para buscar trechos similares na base de conhecimento do Capítulo 1
def similarty_base_cap_18(query_user, k_user, chunks_18, embeddings_18, knowledge_base_18
    # Obtém a representação vetorial da consulta do usuário.
    query = embeddings_18.embed_query(query_user)

    # Realiza a busca por similaridade na base de conhecimento do Capítulo 18.
    docs = knowledge_base_18.similarity_search_with_score_by_vector(query, k=k_user)

    # Lista para armazenar os resultados em formato JSON.
    json_data = []
    context = ''

    # Itera sobre os documentos retornados.
    for index, document in enumerate(docs):
        data = {
            "indice": index,
            "page_content": document[0].page_content,
            "score": float(document[1])
        }
        json_data.append(data)

    # Ordena os dados por score em ordem decrescente.
    sorted_data = sorted(json_data, key=lambda x: x["score"], reverse=True)

    print(f"\nOs trechos com a semelhança com o texto ({query_user}) são os seguintes:\n\

    # Imprime os trechos similares ordenados por score.
    for i in sorted_data:
        print(f"0 Trecho {i['indice']} com o score {i['score']} com o seguinte conteúdo:\
        print('\n\n')

    # A chamada OpenAIEmbeddings.update_forward_refs() é usada para atualizar as referências
    # Isso pode ser útil em situações específicas, como resolver dependências cíclicas entre
    # No entanto, é importante revisar a documentação ou contexto específico do código para e
    OpenAIEmbeddings.update_forward_refs()

```

▼ Criação de base de dados para a tarefa de Similaridade de Textos

```

# Obtendo chunks e embeddings para o Capítulo 8 utilizando a função splitter_OpenAIEmbedd
chunks_8, embeddings_8 = splitter_OpenAIEmbeddings(texto[0]['textOffNormalize'])

# Criando a base de conhecimento FAISS a partir dos chunks e embeddings do Capítulo 8.
knowledge_base_8 = FAISS.from_texts(chunks_8, embeddings_8)

```

```
# Obtendo chunks e embeddings para o Capítulo 18 utilizando a função splitter_OpenAIEmbed
chunks_18, embeddings_18 = splitter_OpenAIEmbeddings(texto[1]['textoOffNormalize'])
```

```
# Criando a base de conhecimento FAISS a partir dos chunks e embeddings do Capítulo 18.
knowledge_base_18 = FAISS.from_texts(chunks_18, embeddings_18)
```

100%

5/5 [01:07<00:00, 17.00s/it]

```
WARNING:langchain.embeddings.openai:Retrying langchain.embeddings.openai.embed_with_r
WARNING:langchain.embeddings.openai:Retrying langchain.embeddings.openai.embed_with_r
WARNING:langchain.embeddings.openai:Retrying langchain.embeddings.openai.embed_with_r
WARNING:langchain.embeddings.openai:Retrying langchain.embeddings.openai.embed_with_r
WARNING:langchain.embeddings.openai:Retrying langchain.embeddings.openai.embed_with_r
WARNING:langchain.embeddings.openai:Retrying langchain.embeddings.openai.embed_with_r
WARNING:langchain.embeddings.openai:Retrying langchain.embeddings.openai.embed_with_r
WARNING:langchain.embeddings.openai:Retrying langchain.embeddings.openai.embed_with_r
WARNING:langchain.embeddings.openai:Retrying langchain.embeddings.openai.embed_with_r
WARNING:langchain.embeddings.openai:Retrying langchain.embeddings.openai.embed_with_r
WARNING:langchain.embeddings.openai:Retrying langchain.embeddings.openai.embed_with_r
WARNING:langchain.embeddings.openai:Retrying langchain.embeddings.openai.embed_with_r
WARNING:langchain.embeddings.openai:Retrying langchain.embeddings.openai.embed_with_r
```

▼ Aplicação da Tarefa de Similaridade de Textos

```
# Loop principal para interação contínua com o usuário.
```

```
while True:
```

```
    # Solicitação para digitar o número do capítulo desejado.
```

```
    entrada_usuario = int(input("Digite qual capítulo você quer recuperar informação: "))
```

```
    # Verifica se o número do capítulo é válido.
```

```
    if entrada_usuario == 8 or entrada_usuario == 18:
```

```
        # Solicitação para digitar a informação desejada.
```

```
        informacao = input("\nDigite qual informação quer recuperar: ")
```

```
    # Solicitação para informar a quantidade de trechos desejada.
```

```
    trechos = int(input("\nInforme a quantidade de informações que quer recuperar: "))
```

```
    # Verifica qual capítulo foi escolhido e chama a função correspondente.
```

```
    if entrada_usuario == 8:
```

```
        similarty_base_cap_8(informacao, trechos, chunks_8, embeddings_8, knowledge_b
```

```
    elif entrada_usuario == 18:
```

```
        similarty_base_cap_18(informacao, trechos, chunks_18, embeddings_18, knowledg
```

```
    # Solicitação para digitar 0 para sair ou qualquer outra tecla para continuar.
```

```
    sair_ou_ficar = int(input("\nDigite 0 para sair ou qualquer tecla para continuar:"))
```

```
    if sair_ou_ficar == 0:
```

```
        break
```

```
else:
```

```
    print("\nVocê digitou um número de capítulo inválido na base de dados.")
```

Digite qual capítulo você quer recuperar informação: 1

Digitou um capítulo inválido na base de dados

Digite qual capítulo você quer recuperar informação: 8

Digite qual informação quer recuperar: o que é pnl

Informe quantidade de informações que quer recuperar: 3

100%

1/1 [00:00<00:00, 2.97it/s]

Os trechos com a semelhança são os seguintes:

O Trecho 2 com o score 0.6119176149368286 com o seguinte conteúdo:

Entre as técnicas simbólicas e as representações distribuídas existem ainda os (Por exemplo, tomando a palavra “trabalho” destacada no parágrafo anterior, a tarefa Quadro 8.2 Acepções da palavra trabalho conforme dicionário Emprego da força física ou intelectual para realizar alguma coisa Aplicação dessas forças como ocupação profissional: Seu trabalho é de gari. Local onde isso se realiza: Mora longe do trabalho. Esmero, cuidado que se emprega na confecção ou elaboração de uma obra A confecção, elaboração ou composição de uma obra Obra realizada: Essa cômoda é um belo trabalho de marcenaria. Grande esforço; TRABALHÃO; TRABALHEIRA

O Trecho 1 com o score 0.6075577139854431 com o seguinte conteúdo:

Contudo, o senhor Palomar não perde o ânimo e a cada momento acredita haver cons O vento estaria mudando? É pena que a imagem que o senhor Palomar havia conseguido Bastaria não perder a paciência, coisa que não tarda a acontecer. O senhor Palomar CALVINO, Ítalo. Palomar. São Paulo: Companhia das Letras, 1994. p.7-11. O que vemos, por trás da simplória tarefa de observação de uma única onda, é a di

O Trecho 0 com o score 0.551091194152832 com o seguinte conteúdo:

O fato de representações distribuídas terem levado a resultados positivos no PLM Os próximos capítulos aprofundam cada uma dessas maneiras de trabalhar com o sigr BAKER, C.; FELLBAUM, C.; PASSONNEAU, R. Semantic Annotation of MASC. Em: Handbook BREWSTER, C.; WILKS, Y. Ontologies, taxonomies, thesauri: learning from texts. (M FREITAS, C. Linguística Computacional. [s.l.] Editora Parábola, 2022. ILARI, R.; GERALDI, J. W. Semântica. [s.l.] Ética, 1985. JURAFSKY, D.; MARTIN, J. H. Speech and Language Processing: An Introduction to Na KILGARRIFF, A. I Don't Believe in Word Senses. Computers and the Humanities, 1997 KILGARRIFF, A. Thesauruses for Natural Language Processing. Proceedings of Natural MARTINS, H. Sobre a estabilidade do significado em Wittgenstein. Veredas, v. 4, 1 MARTINS, H. Três Caminhos na Filosofia da Linguagem. Em: Introdução à Linguística MITKOV, R. The Oxford handbook of Computational Linguistics. [s.l.] Oxford Univer


```
# Definição da função para dividir um texto em chunks e obter embeddings utilizando OpenAI
def splitter_OpenAIEmbeddings(text):
    # Configuração do text splitter para separar o texto por quebras de linha.
    text_splitter = CharacterTextSplitter(
        separator="\n",
        chunk_size=2500,
        chunk_overlap=200,
        length_function=len
    )

    # Divisão do texto em chunks.
    chunks = text_splitter.split_text(text)

    # Configuração e criação de embeddings utilizando OpenAIEmbeddings.
    embeddings = OpenAIEmbeddings(
        request_timeout=180,
        deployment='text-embedding-ada-002',
        chunk_size=10,
        openai_api_key=openai.api_key,
        show_progress_bar=True
    )

    return chunks, embeddings
```

```
from requests.models import Response
```

```
def requestOpenAI(query_user, chunks, embeddings, knowledge_base, mensagens):  
    pergunta = query_user  
    query = embeddings.embed_query(query_user)  
    docs = knowledge_base.similarity_search_with_score_by_vector(query, k=4)  
    json_data = []  
    context = ''
```

```
    # Itera sobre os documentos retornados.
```

```
    for index, document in enumerate(docs):
```

```
        data = {  
            "indice": index,  
            "page_content": document[0].page_content,  
            "score": float(document[1])  
        }
```

```
        json_data.append(data)
```

```
        context += f'\nTrecho {index}:\n {document[0].page_content}'
```

```
    # Template para criar o prompt com o contexto e a pergunta.
```

```
    prompt_template = "Use as seguintes partes dos trechos para responder à pergunta. Con
```

```
    prompt = prompt_template.replace("{Context}", context).replace("{Pergunta}", pergunta
```

```
# Concatenando os textos normalizados dos Capítulos 8 e 18 para criar o texto base
```