



Oque é BM25?

BM25 é um algoritmo de classificação de relevância usado em sistemas de recuperação de informações, como mecanismos de busca. BM25 é uma abreviação de "Best Match 25", ou seja, o melhor resultado de 25 resultados possíveis. Ele é projetado para avaliar a relevância de um documento para uma consulta de pesquisa específica.

O BM25 é uma variação aprimorada do modelo de espaço vetorial, que é um modelo clássico para classificação de relevância. Ele leva em consideração a frequência de termos nos documentos e nas consultas, bem como a frequência inversa de documentos (IDF) de cada termo. O modelo também incorpora um fator de normalização para lidar com a variação do comprimento do documento.

O BM25 é usado em muitos mecanismos de busca populares, como o Google, e é considerado um dos algoritmos de classificação de relevância mais eficazes disponíveis atualmente.



Como implementar um sistema de Information Retrival em python usando gpt-3 e BM25?

Para implementar um sistema de Information Retrieval (Recuperação de Informação) em Python usando o modelo GPT-3 e o algoritmo BM25, você precisaria seguir os seguintes passos:

1. Preparar os dados: Para criar um sistema de recuperação de informações, você precisará de um conjunto de dados contendo os documentos que deseja indexar. Esses documentos podem ser textos de sites da web, artigos, livros ou qualquer outra coisa que você queira indexar. Para este exemplo, vamos usar um conjunto de dados de artigos sobre tecnologia.
2. Indexar os documentos: Para indexar os documentos, você pode usar uma biblioteca de indexação, como o Whoosh ou o Elasticsearch. Essas bibliotecas são capazes de extrair informações dos documentos, como os termos que aparecem nos documentos e

a frequência com que aparecem. Eles também permitem que você armazene essas informações em um formato que permita consultas rápidas.

3. Implementar o algoritmo BM25: Para implementar o algoritmo BM25, você pode usar a biblioteca python-whoosh, que fornece uma implementação do BM25. Você precisará calcular o BM25 para cada termo da consulta em relação a cada documento indexado. A biblioteca python-whoosh permite que você calcule o BM25 com base nos termos da consulta e nos termos indexados nos documentos.
4. Usar o modelo GPT-3 para refinar a pesquisa: Para refinar a pesquisa, você pode usar o modelo GPT-3 da OpenAI. O modelo GPT-3 é capaz de gerar texto que é semanticamente semelhante ao texto fornecido como entrada. Você pode usar o modelo para gerar sugestões de consulta com base na consulta original. Por exemplo, se o usuário pesquisar "melhores smartphones", você pode usar o modelo GPT-3 para gerar sugestões de consulta, como "smartphones mais vendidos" ou "smartphones mais baratos".
5. Retornar os resultados: Depois de calcular o BM25 para cada documento e usar o modelo GPT-3 para refinar a pesquisa, você precisará retornar os resultados ao usuário. Os resultados podem ser classificados com base na pontuação BM25 ou em qualquer outra métrica que você preferir. Você pode retornar uma lista dos documentos mais relevantes ou exibir um snippet do texto em cada documento.

Observe que a implementação de um sistema de recuperação de informações é um processo complexo e requer conhecimento de várias tecnologias e algoritmos. O processo acima é apenas um exemplo simples para ilustrar o fluxo de trabalho básico.



quais bibliotecas voce recomenda que eu use pra isso?

Para implementar um sistema de Information Retrieval em Python usando GPT-3 e BM25, recomendo o uso das seguintes bibliotecas:

1. Hugging Face Transformers: É uma biblioteca de aprendizado profundo (deep learning) que inclui modelos de linguagem de última geração, incluindo o GPT-3. A biblioteca facilita o carregamento e uso do GPT-3 e outros modelos de linguagem em Python.
2. Whoosh: É uma biblioteca de indexação em Python que permite indexar e pesquisar documentos em Python. É fácil de usar e oferece uma implementação do algoritmo BM25 para classificar documentos.

3. NLTK (Natural Language Toolkit): É uma biblioteca de processamento de linguagem natural em Python que fornece recursos para pré-processamento de texto, tokenização, análise sintática e outras tarefas relacionadas à linguagem natural.
4. Pandas: É uma biblioteca de análise de dados em Python que permite manipular e analisar dados em Python. É útil para carregar e manipular dados de documentos e conjuntos de dados.
5. Flask: É um framework web em Python que permite construir aplicativos web com facilidade. É útil para criar uma interface do usuário para o sistema de recuperação de informações.

Essas bibliotecas são amplamente usadas na comunidade de ciência de dados e aprendizado de máquina em Python e têm documentação e recursos excelentes disponíveis online para ajudá-lo a começar.



É possível fazer um sistema desses em uma semana?

Implementar um sistema de Information Retrieval em uma semana usando GPT-3 e BM25 é um desafio, mas pode ser possível, dependendo do escopo e complexidade do projeto, e do seu nível de habilidade e experiência em Python e aprendizado de máquina.

Um fator importante para determinar se é possível concluir um projeto em uma semana é a quantidade de dados que precisam ser indexados e processados. Se você precisar processar e indexar uma grande quantidade de documentos, isso pode levar bastante tempo, e pode ser difícil concluir tudo em uma semana.

Outro fator importante é o seu nível de conhecimento e experiência em Python e aprendizado de máquina. Se você já tem experiência com as bibliotecas que mencionei anteriormente, isso pode tornar o processo mais fácil e rápido. No entanto, se você for novo nessas bibliotecas e no aprendizado de máquina em geral, pode levar mais tempo para aprender como usá-las efetivamente.

Dito isso, com um escopo bem definido, um conjunto de dados gerenciável e um nível razoável de experiência em Python e aprendizado de máquina, é possível criar um sistema de Information Retrieval simples em uma semana. É importante lembrar que a qualidade do sistema pode ser afetada pelo tempo limitado disponível para desenvolvê-lo, e você pode precisar priorizar certas funcionalidades em detrimento de outras.



traduza isso pra mim: Project Overview: The aim of this project is to introduce students to the basics of Information Retrieval (IR) and the BM25 ranking algorithm. The solution to this project will help us to select the students to attend the course on deep learning applied to search. The students will build a simple IR system that retrieves relevant documents from a pre-defined corpus based on the user's query. The students will also use the chatGPT to assist in all phases of the project. The coding will be done using Google Colab notebooks and the deliverables will be placed in github account.

Visão Geral do Projeto: O objetivo deste projeto é introduzir os alunos aos fundamentos da Recuperação de Informações (IR) e do algoritmo de classificação BM25. A solução para este projeto nos ajudará a selecionar os alunos para frequentar o curso de aprendizado profundo aplicado à pesquisa. Os alunos irão construir um sistema simples de IR que recupera documentos relevantes de um corpus pré-definido com base na consulta do usuário. Os alunos também utilizarão o chatGPT para auxiliar em todas as fases do projeto. A codificação será feita usando cadernos do Google Colab e os entregáveis serão colocados em uma conta do GitHub.



Me ensina a usar o python-whoosh para indexar os documentos, como no passo 2

Claro! Aqui está um exemplo simples de como usar o Whoosh para indexar documentos:

1. Primeiro, você precisa instalar a biblioteca Whoosh. Você pode fazer isso usando o gerenciador de pacotes pip. Abra o terminal e execute o seguinte comando:

```
pip install whoosh
```

2. Em seguida, crie um novo arquivo Python e importe a biblioteca Whoosh:

```
python
```

```
import os
from whoosh.index import create_in
from whoosh.fields import *
from whoosh.qparser import QueryParser
```

3. Agora, você precisa definir um esquema de indexação para seus documentos. Um esquema define como os documentos serão armazenados e indexados. Aqui está um exemplo simples de um esquema para documentos que contêm um título e um corpo de texto:

```
scss
schema = Schema(title=TEXT(stored=True), content=TEXT)
```

4. Em seguida, crie um índice vazio usando o esquema que você acabou de definir:

```
lua
if not os.path.exists("index"):
    os.mkdir("index")
ix = create_in("index", schema)
```

5. Agora você pode adicionar seus documentos ao índice. Aqui está um exemplo de como adicionar um único documento:

```
python
writer = ix.writer()
writer.add_document(title=u"My document", content=u"This is the content of my document")
writer.commit()
```

Você pode adicionar quantos documentos quiser dessa forma.

6. Por fim, você pode realizar uma pesquisa no índice para recuperar documentos relevantes. Aqui está um exemplo simples de como pesquisar o índice usando uma consulta:

```
scss
with ix.searcher() as searcher:
    query = QueryParser("content", ix.schema).parse("content")
    results = searcher.search(query)
    for r in results:
        print(r['title'])
```

Este exemplo procura por todos os documentos que contêm a palavra "content" no campo "content" e, em seguida, imprime o título de cada documento correspondente.

Este é apenas um exemplo simples de como usar o Whoosh para indexar e pesquisar documentos. Há muito mais que você pode fazer com o Whoosh, dependendo das suas necessidades. Certifique-se de ler a documentação oficial do Whoosh para obter mais informações.



me explica a classe Schema do whoosh-python

A classe Schema do Whoosh é usada para definir o esquema de indexação para seus documentos. Ela define quais campos devem ser armazenados e indexados para cada documento. Você pode criar vários campos para cada documento, como título, corpo do texto, autor, data de criação, entre outros.

Aqui estão alguns dos tipos de campos mais comuns que você pode usar com a classe Schema:

- **TEXT:** Este campo é usado para texto completo. O texto é dividido em palavras e indexado para pesquisa. É possível armazenar ou não o texto completo.
- **KEYWORD:** Este campo é usado para palavras-chave. As palavras-chave não são divididas em palavras e são indexadas para pesquisa. É possível armazenar ou não as palavras-chave.
- **ID:** Este campo é usado para o identificador exclusivo de cada documento. Este campo é indexado para que você possa procurar documentos por ID.
- **STORED:** Este campo é usado para armazenar dados que você deseja recuperar junto com o documento, mas que não serão indexados para pesquisa. Por exemplo, você pode usar este campo para armazenar o nome do arquivo ou a URL do documento.

Você pode criar um objeto Schema com a função Schema() e adicionar campos ao objeto usando os tipos de campos acima mencionados, como parâmetros. Aqui está um exemplo de como criar um objeto Schema para documentos com campos de título e conteúdo:

```
scss
```

```
from whoosh.fields import Schema, TEXT
```

```
my_schema = Schema(title=TEXT(stored=True), content=TEXT)
```

No exemplo acima, definimos um esquema com dois campos: "title" e "content". Ambos são campos de texto completo e o campo "title" é armazenado para que possamos recuperar o título do documento. Você pode adicionar mais campos, se desejar, para atender às suas necessidades de indexação.



me explica esse código: `import os from whoosh.index import create_in from whoosh.fields import Schema, TEXT, ID import sys def createSearchableData(root): "" Schema definition: title(name of file), path(as ID), content(indexed but not stored),textdata (stored text content) "" schema = Schema(title=TEXT(stored=True),path=ID(stored=True),\ content=TEXT,textdata=TEXT(stored=True)) if not os.path.exists("indexdir"): os.mkdir("indexdir") # Creating a index writer to add document as per schema ix = create_in("indexdir",schema) writer = ix.writer() filepaths = [os.path.join(root,i) for i in os.listdir(root)] for path in filepaths: fp = open(path,'r') print(path) text = fp.read() writer.add_document(title=path.split("\\")[1], path=path,\ content=text,textdata=text) fp.close() writer.commit() root = "corpus" createSearchableData(root)`

O código que você postou é um exemplo de como usar o Whoosh para criar um índice de pesquisa de um conjunto de documentos.

O objetivo deste código é indexar todos os arquivos de texto no diretório especificado e armazenar os dados indexados em um diretório chamado "indexdir". O esquema de indexação é definido como um título (o nome do arquivo), um caminho (o caminho completo para o arquivo), o conteúdo do arquivo (indexado, mas não armazenado) e o texto do arquivo (armazenado).

Aqui está uma explicação do que cada parte do código faz:

- `import os`: Importa o módulo OS para trabalhar com o sistema operacional.
- `from whoosh.index import create_in`: Importa a função `create_in` do módulo Whoosh index para criar um novo índice.
- `from whoosh.fields import Schema, TEXT, ID`: Importa a classe Schema e os tipos de campos TEXT e ID do módulo Whoosh fields.
- `import sys`: Importa o módulo sys para acessar argumentos da linha de comando.
- `def createSearchableData(root)::` Define uma função chamada `createSearchableData` que aceita um argumento de raiz.

- `schema = Schema(title=TEXT(stored=True),path=ID(stored=True),content=TEXT,textdata=TEXT(stored=True))`: Define um objeto Schema com quatro campos: título, caminho, conteúdo e dados de texto. O campo de título é armazenado e todos os outros são indexados.
- `if not os.path.exists("indexdir")::` Verifica se o diretório "indexdir" já existe. Se não existir, ele será criado.
- `ix = create_in("indexdir",schema)`: Cria um novo índice no diretório "indexdir" com o esquema especificado.
- `writer = ix.writer()`: Cria um objeto de escritor para o índice.
- `filepaths = [os.path.join(root,i) for i in os.listdir(root)]`: Cria uma lista de caminhos de arquivo para todos os arquivos no diretório especificado.
- `for path in filepaths::` Itera por cada caminho de arquivo.
- `fp = open(path,'r')`: Abre o arquivo para leitura.
- `text = fp.read()`: Lê o conteúdo do arquivo em uma variável.
- `writer.add_document(title=path.split("\\")[1], path=path,content=text,textdata=text)`: Adiciona um novo documento ao índice com o título (o nome do arquivo), o caminho (o caminho completo para o arquivo), o conteúdo do arquivo e o texto do arquivo.
- `fp.close()`: Fecha o arquivo.
- `writer.commit()`: Salva as alterações no índice.
- `root = "corpus"`: Define o diretório raiz que contém os arquivos de texto a serem indexados.
- `createSearchableData(root)`: Chama a função `createSearchableData` com o diretório raiz especificado.



me explica esse código `ix = open_dir("indexdir")` # query_str is query string `query_str = sys.argv[1]` # Top 'n' documents as result `topN = sys.argv[2]`

Este código tem como objetivo recuperar resultados para uma consulta em um índice criado anteriormente com o Whoosh.

A linha `ix = open_dir("indexdir")` abre o índice previamente criado no diretório "indexdir" e armazena uma referência a ele na variável `ix`.

A seguir, duas variáveis são definidas com os argumentos passados via linha de comando: `query_str` é a string de consulta e `topN` é o número máximo de resultados para retornar.

Isso indica que o código foi projetado para ser executado a partir da linha de comando, recebendo argumentos como entrada.

Com o índice aberto e as consultas definidas, o próximo passo seria criar um objeto searcher a partir do índice e executar a consulta usando a sintaxe do Whoosh. Como não há código adicional nesse exemplo, não é possível saber como isso foi feito especificamente.



O que é "CISI collections" para testar information retrieval ?

A CISI (Computer and Information Science Index) é uma coleção de documentos de texto usada para avaliar sistemas de recuperação de informações (IR). A coleção contém cerca de 1500 resumos de artigos de periódicos, extraídos principalmente do campo da ciência da computação, e é frequentemente usada como um padrão de referência para testar algoritmos e técnicas de IR.

A CISI foi criada em meados da década de 1960 pela Aslib Cranfield Research Project, que foi responsável por coletar e categorizar os documentos. A coleção tem sido usada em uma série de experimentos de IR ao longo dos anos, e muitos estudos têm relatado resultados de recuperação de documentos usando a CISI como coleção de teste.

A CISI é uma das várias coleções de testes disponíveis publicamente para avaliar a eficácia de sistemas de IR. Outras coleções comumente usadas incluem a coleção de documentos TREC (Text Retrieval Conference) e a coleção de documentos CACM (Communications of the ACM).



como fazer um sistema de information Retrieval usando whoosh e a coleção da CISI

Para fazer um sistema de recuperação de informações usando Whoosh e a coleção da CISI, você pode seguir estes passos:

1. Baixe a coleção da CISI: A coleção pode ser baixada de várias fontes, como o site da Universidade de Glasgow ou o site da Cranfield University. Ela inclui um conjunto de documentos de texto e uma lista de consultas para usar como entrada no sistema de IR.
2. Instale o Whoosh: O Whoosh é uma biblioteca Python para construir e pesquisar índices de texto. Para instalá-lo, use o gerenciador de pacotes pip do Python com o seguinte comando: `pip install whoosh`
3. Crie um índice Whoosh: Use a biblioteca Whoosh para criar um índice de busca com base nos documentos da coleção CISI. O esquema do índice deve incluir campos que correspondam aos dados dos documentos, como o título, o conteúdo do texto e o identificador do documento. Para cada documento na coleção, adicione um registro ao índice usando o método `writer.add_document()`.
4. Execute consultas no índice: Use o índice criado anteriormente para executar consultas, utilizando a classe `searcher` do Whoosh. Para cada consulta na lista de consultas da CISI, execute a consulta no índice usando o método `searcher.search()` e retorne uma lista de resultados correspondentes. Os resultados devem ser classificados pelo BM25 (ou outro algoritmo de classificação) para retornar os documentos mais relevantes.
5. Avalie o desempenho do sistema: Para avaliar o desempenho do sistema de IR, compare os resultados retornados pelo sistema com a lista de resultados esperados da lista de consultas da CISI. Isso pode ser feito usando medidas como precisão, recall e F-measure.
6. Aperfeiçoe o sistema: Com base nos resultados da avaliação de desempenho, aperfeiçoe o sistema de IR para melhorar seu desempenho em futuras consultas.

Lembre-se de que a implementação de um sistema de IR pode ser um processo complexo e que este é apenas um resumo geral dos passos envolvidos. Para obter mais informações e detalhes específicos sobre como implementar um sistema de IR usando a coleção CISI e o Whoosh, consulte a documentação do Whoosh e os recursos disponíveis online.



como indexar uma coleção CISI usando whoosh

Para indexar a coleção CISI usando o Whoosh, você precisará seguir os seguintes passos:

1. Baixe a coleção CISI em seu computador. Você pode encontrar a coleção no seguinte link: http://ir.dcs.gla.ac.uk/resources/test_collections/cisi/
2. Extraia o conteúdo do arquivo zip baixado.
3. Instale o Whoosh em seu ambiente Python, caso ainda não o tenha feito. Você pode fazer isso usando o pip, o gerenciador de pacotes do Python, com o seguinte comando:

pip install whoosh

4. Crie um novo arquivo Python e adicione o seguinte código para criar um índice:

```
python
import os
from whoosh.index import create_in
from whoosh.fields import Schema, TEXT, ID
from whoosh.analysis import StemmingAnalyzer

# Define o schema para os campos do índice
schema = Schema(id=ID(unique=True, stored=True),
                title=TEXT(stored=True),
                author=TEXT(stored=True),
                source=TEXT(stored=True),
                abstract=TEXT(analyzer=StemmingAnalyzer()),
                content=TEXT(analyzer=StemmingAnalyzer()))

# Cria o diretório do índice (se ele não existir)
if not os.path.exists("cisi_index"):
    os.mkdir("cisi_index")

# Cria um indexador para a pasta do índice
ix = create_in("cisi_index", schema)

# Abre o arquivo de texto da coleção CISI
with open("cisi/CISI.ALL") as f:
    docs = f.read().split(".I ")

# Itera sobre cada documento na coleção
for doc in docs[1:]:
    doc_id, doc_content = doc.split(".W\n")

    # Remove quebras de linha do final do documento
    doc_content = doc_content.strip()

    # Adiciona o documento ao índice
    ix.writer().add_document(id=doc_id,
                             title="",
                             author="",
                             source="",
                             abstract="",
                             content=doc_content)
```

```
# Finaliza a indexação
ix.writer().commit()
```

5. Salve o arquivo como "index_cisi.py" e execute-o. Ele irá criar um índice em um diretório chamado "cisi_index", com os documentos da coleção CISI indexados.

Após seguir esses passos, você poderá usar o índice criado para realizar buscas na coleção CISI.



```

estou fazendo um projeto em python de um sistema de informacao retrieval usando
whoosh para indexar a pasta CISI, bm25 para pegar os melhores hits e preciso
implementar o gpt-3 para refinar a busca. Até agora eu tenho esse codigo para indexar o
CISI
import os
import re
from whoosh import index
from whoosh.index import create_in
from whoosh.fields import Schema, TEXT, ID
from whoosh.analysis import StemmingAnalyzer

def index_cisi_collection():
    if index.exists_in("cisi_index"):
        try:
            ix = index.open_dir("cisi_index")
            writer = ix.writer()
        except index.LockError:
            ix = index.open_dir("cisi_index", readonly=True)
            ix._unlock()
            os.remove(os.path.join("cisi_index", "write.lock"))
            ix = index.open_dir("cisi_index")
            writer = ix.writer()
        else:
            # Define o schema para os campos do índice
            schema = Schema(
                id=ID(unique=True, stored=True),
                title=TEXT(stored=True),
                author=TEXT(stored=True),
                source=TEXT(stored=True),
                abstract=TEXT(analyzer=StemmingAnalyzer()),
                content=TEXT(analyzer=StemmingAnalyzer())
            )
            # Cria o diretório do índice (se ele não existir)
            if not os.path.exists("cisi_index"):
                os.mkdir("cisi_index")
            # Cria um indexador para a pasta do índice
            ix = create_in("cisi_index", schema)
            writer = ix.writer()
            # Abre o arquivo de texto da coleção CISI
            with open("cisi/CISI.ALL", "r") as f:
                # Lê o arquivo inteiro de uma vez
                text = f.read()
                # Usa expressões regulares para separar os documentos e extrair os campos relevantes
                pattern = r"\.l(\d+)\.l\.T\.l(.*?)\.l\.A\.l(.*?)\.l\.B\.l(.*?)\.l\.W\.l(.*?)\.l\.X\.l"
                docs = re.findall(pattern, text, re.DOTALL)
                # Itera sobre cada documento na coleção
                for doc in docs:
                    doc_id, title, author, source, content = doc
                    # Adiciona o documento ao índice
                    writer.add_document(
                        id=doc_id,
                        title=title,
                        author=author,
                        source=source,
                        abstract="",
                        content=content.strip()
                    )
            # Finaliza a indexação
            writer.commit()
            # esse codigo para fazer a busca
            from whoosh import qparser
            from whoosh.index import open_dir
            from whoosh.qparser import QueryParser
            from indexing import index_cisi_collection
            from

```

```

whoosh import scoring def parser(search_string): ix = open_dir("cisi_index") # Define o
esquema de consulta qp = QueryParser("title", schema=ix.schema) # Define o algoritmo
BM25 como o algoritmo de pontuação searcher =
ix.searcher(weighting=scoring.BM25F()) # Executa a consulta query =
qp.parse(search_string) results = searcher.search(query, limit=None) # Itera sobre os
resultados for hit in results: print(f"ID: {hit['id']}") print(f"Title: {hit['title']}")
print(f"Author: {hit['author']}") print(f"Source: {hit['source']}") #print(f"Abstract:
{hit['abstract']}") #print(f"Content: {hit['content']}") print(f"Score: {hit.score}") print()
como implemento o gpt-3 em um novo arquivo python para refinar a busca?

```

Para implementar o GPT-3 para refinar a busca, você precisará de uma API key do OpenAI para acessar o modelo GPT-3. Se você não tiver uma chave API, precisará se inscrever no programa de acesso antecipado do OpenAI.

Após ter acesso à API key, você pode usar a biblioteca OpenAI para Python para enviar consultas ao GPT-3. Você pode usar as respostas geradas pelo modelo para refinar a consulta original e, em seguida, executar a busca novamente usando Whoosh e BM25F para obter os melhores resultados.

Aqui está um exemplo básico de como usar o GPT-3 para refinar uma consulta:

```

python
import openai
from whoosh import scoring
from whoosh.index import open_dir
from whoosh.qparser import QueryParser

# Configuração da API key
openai.api_key = "SUA_API_KEY_DO_OPENAI"

# Define a consulta original
query_string = "informação retrieval"

# Chama a API do GPT-3 para refinar a consulta
response = openai.Completion.create(
    engine="davinci",
    prompt=f"Refine the search query: {query_string}.",
    max_tokens=64,
    n=1,
    stop=None,
    temperature=0.5
)

# Extrai a resposta gerada pelo modelo
refined_query_string = response.choices[0].text.strip()

```

```
# Imprime a consulta refinada
print(f'Refined query: {refined_query_string}')

# Executa a consulta refinada no índice Whoosh
ix = open_dir("cisi_index")
qp = QueryParser("title", schema=ix.schema)
searcher = ix.searcher(weighting=scoring.BM25F())
query = qp.parse(refined_query_string)
results = searcher.search(query, limit=None)

# Imprime os resultados da busca
for hit in results:
    print(f'ID: {hit["id"]}')
    print(f'Title: {hit["title"]}')
    print(f'Author: {hit["author"]}')
    print(f'Source: {hit["source"]}')
    print(f'Score: {hit.score}')
    print()
```

Neste exemplo, a consulta original é definida como "informação retrieval" e é passada para a API do GPT-3 para refinar a consulta. A resposta gerada pelo modelo é armazenada em `refined_query_string` e, em seguida, a consulta refinada é executada no índice Whoosh. Os resultados da busca são impressos na tela. Observe que você pode ajustar os parâmetros de chamada da API do GPT-3, como o modelo a ser usado, o número máximo de tokens a serem gerados e a temperatura de amostragem.



como eu faria isso usando o Hugging Face Transformers/

Para usar o GPT-3 com o Hugging Face Transformers em Python, você precisará seguir alguns passos:

1. Instale o pacote transformers usando pip:

```
pip install transformers
```

2. Importe a classe GPT3Tokenizer e GPT3ForConditionalGeneration do módulo transformers:

```
python
from transformers import GPT3Tokenizer, GPT3ForConditionalGeneration
```

3. Crie uma instância do tokenizador GPT3Tokenizer e do modelo GPT3ForConditionalGeneration. Você precisará especificar o nome do modelo (por exemplo, davinci), bem como a chave da API do OpenAI:

```
python
model_name = "openai/gpt-3"
api_key = "sua_chave_api"

tokenizer = GPT3Tokenizer.from_pretrained(model_name, api_key=api_key)
model = GPT3ForConditionalGeneration.from_pretrained(model_name, api_key=api_key)
```

4. Defina uma frase ou pergunta que você deseja passar para o modelo para gerar um texto refinado. Você precisará preparar a entrada para o modelo usando o tokenizador GPT3Tokenizer. Por exemplo:

```
python
input_text = "Refine search results for 'machine learning'"
input_ids = tokenizer.encode(input_text, return_tensors="pt")
```

5. Use o modelo GPT3ForConditionalGeneration para gerar o texto refinado. Você precisará passar os IDs de entrada para o modelo e especificar a quantidade de texto que deseja gerar (em tokens).

```
python
max_length = 100
generated_ids = model.generate(input_ids, max_length=max_length)
generated_text = tokenizer.decode(generated_ids[0], skip_special_tokens=True)
```

O texto gerado contém sugestões para refinar a busca. Você pode exibi-lo na saída ou usá-lo para ajustar sua consulta Whoosh e pesquisar novamente. Lembre-se de que a API do OpenAI GPT-3 é paga, portanto, certifique-se de estar ciente dos custos antes de usá-la.





tem como usar a api do gpt-3 de graça?

Atualmente, a API do GPT-3 não está disponível gratuitamente. Para usar a API, é necessário ter uma conta no OpenAI e pagar pelo acesso.

A OpenAI oferece um plano de uso gratuito chamado "OpenAI API Beta", que oferece um número limitado de solicitações gratuitas por mês para testar a API. No entanto, essa oferta é limitada e pode não ser suficiente para uso em projetos em produção ou com grande volume de solicitações.

Além disso, a OpenAI também oferece um programa de créditos para pesquisadores, educadores e organizações sem fins lucrativos, que permite o acesso gratuito ou com desconto à API do GPT-3. Porém, para se qualificar para esse programa, é necessário enviar uma solicitação para a OpenAI e aguardar aprovação.

Em resumo, atualmente não é possível usar a API do GPT-3 gratuitamente sem restrições.



preciso usar o CISI para testar um sistema de information retrieval

O CISI (Collected and Integrated Set of Indexes) é um conjunto de dados comumente usado em pesquisas e avaliações de sistemas de recuperação de informações. Ele contém um conjunto de documentos indexados e uma série de consultas correspondentes para testar a eficácia do sistema em recuperar informações relevantes.

Se você estiver desenvolvendo um sistema de recuperação de informações, o CISI pode ser uma boa escolha para testar e avaliar o desempenho do seu sistema. Ele fornece uma coleção padronizada de dados e consultas, o que pode ajudar a comparar o desempenho do seu sistema com outros sistemas de recuperação de informações.

Para usar o CISI, você precisará baixar a coleção de dados e as consultas correspondentes. Existem várias fontes onde você pode baixar o CISI, incluindo o site oficial do conjunto de dados e repositórios online de dados de pesquisa, como o Kaggle.

Uma vez que você tenha a coleção de dados e consultas, você pode usá-los para testar e avaliar seu sistema de recuperação de informações. Existem várias métricas que podem ser usadas para avaliar o desempenho do seu sistema, incluindo precisão, recall e F1-score. Você pode usar essas métricas para comparar o desempenho do seu sistema com outros sistemas de recuperação de informações e para ajustar e melhorar o seu próprio sistema.



como extrair os campos relevantes do CISI na hora de indexar?

Na hora de indexar os documentos da coleção CISI, é necessário extrair os campos relevantes de cada documento e adicioná-los ao índice. Os campos relevantes na coleção CISI são:

- Identificador do documento (campo .I no arquivo de texto da coleção)
- Título do documento (campo .T)
- Autor(es) do documento (campo .A)
- Fonte do documento (campo .B)
- Resumo do documento (campo .W)
- Texto completo do documento (após o campo .X)

Para extrair esses campos, é possível utilizar expressões regulares para fazer a correspondência com o formato dos campos na coleção. Aqui está um exemplo de como fazer isso em Python:

```
python
import re
```

```
with open("CISI.ALL", "r") as f:
```

```
    # Lê o arquivo inteiro de uma vez
    text = f.read()
```

```
    # Usa expressões regulares para separar os documentos e extrair os campos relevantes
```

```
    pattern = r".I (\d+)\n.T\n(?:\n.A\n(?:\n.B\n(?:\n.W\n(?:\n.X\n"
```

```
docs = re.findall(pattern, text, re.DOTALL)
```

```
    # Itera sobre cada documento na coleção
```

```
    for doc in docs:
```

```
        doc_id, title, author, source, abstract, content = doc
```

```
# Adiciona o documento ao índice
writer.add_document(id=doc_id,
                    title=title,
                    author=author,
                    source=source,
                    abstract=abstract,
                    content=content.strip())
```

O código acima usa a expressão regular `r"\.l`
`(\d+)\n\.T\n(.*?)\n\.A\n(.*?)\n\.B\n(.*?)\n\.W\n(.*?)\n\.X\n"` para encontrar todos os documentos na coleção CISI. Cada campo do documento é representado por um grupo de captura na expressão regular. Depois que a correspondência é feita, cada grupo de captura é usado para extrair os campos relevantes do documento, que são então adicionados ao índice. Note que o campo `abstract` é opcional na coleção CISI, portanto, em alguns casos pode estar vazio.