Introdução

Este tutorial fornecerá uma base sólida para você explorar e implementar soluções de IA generativa, utilizando as ferramentas e técnicas mais avançadas disponíveis. Ao final deste tutorial, você será capaz de:

- **Compreender** o que são Grandes Modelos de Linguagem (LLM) e suas aplicações práticas.
- **Entender** o funcionamento do Retrieval-Augmented Generation (RAG).
- **Aplicar** os conceitos aprendidos para implementar um sistema de RAG usando R e um pouco de Python.
- **Criar** um chatbot capaz de interagir com o conteúdo de um arquivo PDF.

Todo trecho de código apresentado neste tutorial terá uma marcação no título indicando se é R (cor azul) ou Python (cor amarela).

Conceitos Básicos

Esses são os conceitos fundamentais para o entendimento do tutorial:

- **Grandes Modelos de Linguagem**: Modelos de inteligência artificial treinados em grandes volumes de texto para entender e gerar linguagem natural de forma semelhante aos humanos. Exemplos incluem GPT-3 e GPT-4.
- **ChatGPT**: Um chatbot avançado baseado no modelo de linguagem GPT da OpenAI, capaz de realizar conversas complexas, responder perguntas e executar tarefas de linguagem natural.
- **Prompt**: Uma instrução ou conjunto de instruções fornecidas a um modelo de linguagem para orientar a geração de respostas ou execução de tarefas específicas.
- **Embedding**: Representação numérica de dados, como palavras ou frases, em um espaço vetorial multidimensional, permitindo que modelos de IA compreendam relações semânticas entre eles.
- **RAG**: Retrieval-Augmented Generation, uma técnica que combina recuperação de informações e geração de texto, onde um modelo de linguagem é usado para gerar respostas com base em documentos recuperados de uma base de dados.
- **API**: Interface de Programação de Aplicações, um conjunto de definições e protocolos que permite a integração entre diferentes softwares, permitindo que eles se comuniquem e compartilhem funcionalidades.

Requisitos

- **Conta na OpenAI**: para usar a API do ChatGPT, você deve se cadastrar. Além disso você deve adicionar créditos para conseguir utilizar a API da OpenAI. Entre nesta página e adicione. Sugiro adicionar US$ 1, que será mais do que suficiente para execução deste tutorial.
- **R e RStudio Instalados**: Para instalação, faça o download do R em http://www.r-project.org. Em seguida, instale a IDE (Integrated Development Environment) R Studio.
- **Python Instalado**: Caso não tenha o Python instalado na sua máquina, você pode instalá-lo através do pacote `reticulate` usando o comando `reticulate::install_miniconda()`. Detalhes podem ser encontrados aqui.

Para configurar o Python dentro do RStudio, siga os passos abaixo:

1. Abra o RStudio.
2. Vá para o menu `Tools` e selecione `Global Options`.
3. No painel esquerdo, selecione `Python`.
4. Clique no botão `Select...`.
5. Navegue até o executável do Python que você deseja usar e selecione-o. Este caminho é geralmente algo como `C:\Users\SeuUsuario\AppData\Local\Programs\Python\PythonXX\python.exe` no Windows, ou `/usr/bin/python3` no Linux/Mac.
6. Clique em `Apply` e depois em `OK` para salvar as configurações.

Conexão com a API da OpenAI

Na área de API Keys, crie uma nova chave usando o botão *"Create new secret key"*. Salve essa chave, que será usada neste tutorial.

Use o código abaixo para salvar a chave como uma variável de ambiente do RStudio.

Listagem 1 `R`

```r
Sys.setenv(`OPENAI_API_KEY`= "COLE SUA CHAVE AQUI")
```

Configuração de ambiente para rodar Python no RStudio

O pacote `reticulate` do R facilita a integração entre R e Python, permitindo que pessoas programadoras de R utilizem funcionalidades avançadas disponíveis em Python diretamente no RStudio. Isso é particularmente útil para trabalhar com grandes modelos linguagens (LLM), que possuem implementações mais robustas e atualizadas em Python.

Embora `reticulate` torne possível essa integração, é importante ter um conhecimento básico da sintaxe de Python para aproveitar ao máximo as capacidades dos modelos de IA generativa, como o ChatGPT, em suas análises.

No código abaixo, vamos criar um ambiente virtual Python chamado `"langchain_rag_pdf"` e instalar os pacotes necessários para a nossa análise usando o `reticulate`. Um ambiente no Python é um espaço isolado onde você pode instalar pacotes e dependências específicas para um projeto, sem interferir em outros projetos. Isso garante que cada projeto possa ter suas próprias versões de pacotes, evitando conflitos e problemas de compatibilidade. No R, normalmente não é necessário criar ambientes separados, pois o gerenciamento de pacotes é feito de forma global. No entanto, a criação de ambientes no Python oferece vantagens significativas, como isolamento, reprodutibilidade e melhor gestão de dependências.

Listagem 2 `R`

```r
library(reticulate)

virtualenv_create(envname = "langchain_rag_pdf",
                  packages = c( "langchain", "openai", "pypdf", "bs4",
                                "python-dotenv", "chromadb", "tiktoken",
                                "langchain-openai", "langchain-community"))

reticulate::use_virtualenv("langchain_rag_pdf")
```

Como exemplo de funcionamento da integração entre R e Python usando o pacote reticulate, podemos usar o código abaixo:

Listagem 3 `R`

```r
reticulate::py_run_string('
print("Estou no SINAPE em Fortaleza!")
')
```

Para finalizar a configuração, vamos definir uma variável de ambiente com chave da API da OpenAI que foi gerada anteriormente. Altere o código abaixo conforme necessário, substituindo “COLE SUA CHAVE AQUI” pela sua chave de API.

Com isso, a chave da API estará configurada corretamente e você estará pronto para utilizar a OpenAI API no seu ambiente de desenvolvimento.

Listagem 4 R

```r
chave_api <- Sys.getenv("OPENAI_API_KEY")
```

Download e importação do arquivo em PDF

Vamos usar como arquivo de exemplo a documentação do pacote ggplot2, que é amplamente utilizado para criar gráficos no R usando a gramática de gráficos.

Baixe o arquivo PDF que será utilizado no tutorial usando o código abaixo.

Listagem 5 R

```r
if(!(dir.exists("docs"))) {
  dir.create("docs")
}

download.file("https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf",
              destfile = "docs/ggplot2.pdf", mode = "wb")
```

Importe o arquivo como um objeto no LangChain e carregue seu conteúdo para processamento.

Listagem 6 Python

```python
from langchain_community.document_loaders import PyPDFLoader

carregar_pdf = PyPDFLoader('docs/ggplot2.pdf')
print(type(carregar_pdf))

paginas = carregar_pdf.load()

print(type(paginas))

print(len(paginas))
```

Você pode examinar o objeto `paginas` do Python no R usando o objeto `py` do `reticulate`. O código R a seguir armazena o objeto Python paginas em uma variável R chamada

`paginas_em_r`. Você pode então trabalhar com o objeto como qualquer outro objeto R. Neste caso, é uma lista.

Listagem 7 `R`

```r
paginas_em_r <- py$paginas

# metadatos da primeira pagina
paginas_em_r[[1]]$metadata

# quantidade de caracteres da centésima pagina
nchar(paginas_em_r[[100]]$page_content)
```

Divisão do documento em pedaços

Divida o documento PDF em partes menores para facilitar o processamento pelos modelos de IA.

O código abaixo importa a chave da API da OpenAI da variável `chave_api` do R utilizando o objeto `r` do `reticulate` dentro do Python. Ele também carrega o pacote `openai` do Python e o divisor de caracteres recursivo do LangChain, cria uma instância da classe `RecursiveCharacterTextSplitter` e executa o método `split_documents()` dessa instância nos pedaços do documento `paginas`.

Listagem 8 `Python`

```python
import openai
openai.api_key = r.chave_api
from langchain.text_splitter import RecursiveCharacterTextSplitter
divisor_documento = RecursiveCharacterTextSplitter()
partes_pdf = divisor_documento.split_documents(paginas)
```

Avaliação do custo da aplicação

Calcule o custo da aplicação com base no número de tokens gerados. the TheOpenAIR R package has a count_tokens() function (make sure to install both that and purrr to use the R code below):

Vamos calcular o custo da aplicação com base no número de tokens gerados. O pacote `TheOpenAIR` do R possui uma função `count_tokens()` (certifique-se de instalar tanto este pacote quanto o `purrr` para usar o código R abaixo):

Listagem 9 `R`

```
partes_pdf <- py$partes_pdf
length(partes_pdf)

total_tokens <- purrr::map_int(partes_pdf,
                               ~ TheOpenAIR::count_tokens(.x$page_content)) |>
    sum()
```

O código acima mostra que há 153.172 tokens.

Atualmente (agosto de 2024), o custo do modelo **ada v2** usado para criação dos *embeddings* da OpenAI é de US$ 0,10 por 1 milhão de tokens. Como temos 153.172 tokens, o custo dessa etapa será de aproximadamente US$ 0,0153172.

Geração de *embeddings*

Vamos criar um diretório para armazenar a base de dados de embeddings e gerar os embeddings dos pedaços do PDF. O LangChain possui componentes pré-fabricados para criar embeddings a partir dos pedaços de texto e armazená-los. Utilizaremos uma das opções mais simples disponíveis no LangChain: Chroma, uma base de dados de embeddings de código aberto que pode ser usada localmente.

Primeiro, criaremos um subdiretório no diretório docs para armazenar a base de dados, conforme sugerido, para evitar que qualquer coisa além da base de dados esteja no diretório Chroma. O código R a seguir cria este subdiretório:

Listagem 10 `R`

```
if(!dir.exists("docs/chroma_db")) {
  dir.create("docs/chroma_db")
}
```

O código acima gera os embeddings utilizando o OpenAIEmbeddings do LangChain, que atualmente utiliza o modelo ada-2 da OpenAI por padrão. O LangChain suporta vários outros LLMs com sua classe Embeddings, incluindo Hugging Face Hub, Cohere, Llama-cpp e Spacy.

Listagem 11 `Python`

```
import os
os.environ["OPENAI_API_KEY"] = r.chave_api

from langchain_openai import OpenAIEmbeddings
embed = OpenAIEmbeddings()

from langchain_community.vectorstores import Chroma
diretorio_chroma_db = "docs/chroma_db"

vetor_db = Chroma.from_documents(
  documents = partes_pdf,
  embedding = embed,
  persist_directory = diretorio_chroma_db
)

# numero de embeddings criados
print(vetor_db._collection.count())
```

Geração de respostas únicas

Agora vamos trabalhar na construção da integração com o ChatGPT para obter respostas a perguntas. Nessa etapa, a aplicação não terá memória, ou seja, não será possível se referir a respostas anteriores. Usaremos a função `RetrievalQA` do LangChain para perguntar a um LLM, como o GPT-3.5, e gerar uma resposta escrita baseada nos documentos relevantes.

Carregar embeddings

Carregue os embeddings gerados aneriormente para realizar consultas ao documento.

Definir qual modelo de linguagem será usado

Configure o modelo de linguagem a ser usado para responder às perguntas.

Listagem 12 Python

```python
import openai
openai.api_key = r.chave_api

from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma

diretorio_chroma_db = "docs/chroma_db"
vetor_db = Chroma(persist_directory = diretorio_chroma_db,
                  embedding_function = OpenAIEmbeddings())
```

Listagem 13 Python

```python
from langchain_community.chat_models import ChatOpenAI
from langchain_openai import ChatOpenAI
modelo_llm = ChatOpenAI(model_name = "gpt-3.5-turbo", temperature = 0)
```

Crie a cadeia de obtenção de respostas usando o componente RetrievalQA.

Listagem 14 Python

```python
from langchain.chains import RetrievalQA
```

```python
cadeia = RetrievalQA.from_chain_type(modelo_llm, retriever = vetor_db.as_retriever())
```

Obtenção da resposta

Faça uma pergunta ao modelo e obtenha a resposta baseada no conteúdo do PDF.

Obtemos a seguinte resposta:

```
Para rotacionar o texto no eixo x de um gráfico, você pode
usar a função theme() no ggplot2. Especificamente, o argumento
axis.text.x é utilizado para modificar a aparência do texto
do eixo x. Aqui está um exemplo:

library(ggplot2)
```

Listagem 15 `Python`

```
pergunta = "Como rotacionar o texto no eixo x de um gráfico?"
print(cadeia.invoke(pergunta))
```

```
# Create a basic scatter plot
p <- ggplot(mtcars, aes(x = mpg, y = wt)) +
  geom_point()

# Rotate x-axis text by 45 degrees
p + theme(axis.text.x = element_text(angle = 45, hjust = 1))

Neste exemplo, o argumento angle é definido como 45, o que
rotaciona o texto do eixo x em 45 graus. O argumento hjust é
definido como 1, alinhando o texto à direita. Você pode ajustar
os valores de angle e hjust para obter a rotação e o alinhamento
desejados do texto no eixo x.
```

Agora que a cadeia está configurada, você pode executá-la em outras perguntas com apenas um comando usando um script R:

Listagem 16 `R`

```
py_run_string('
print(cadeia.run("Como posso fazer um gráfico de barras onde
as barras são de cor azul usando ggplot?"))
')
```

Resposta:

```
library(ggplot2)

# Criar um gráfico de barras com barras em azul aço
p <- ggplot(mtcars, aes(factor(cyl)))
p + geom_bar(fill = "blue")

Neste exemplo, usamos a estética fill para especificar a cor
das barras como "blue". Você pode ajustar a cor conforme sua
preferência, alterando o nome da cor ou utilizando um código
```

```
de cor hexadecimal.
```

Você também pode querer confirmar que as respostas não estão sendo extraídas da base de conhecimento geral do ChatGPT, mas realmente do documento que você enviou. Para descobrir, você pode fazer uma pergunta completamente não relacionada ao `ggplot2` e que não estaria na documentação:

Listagem 17 `R`

```
py_run_string('
print(cadeia.run("Qual a capital do Ceará?"))
')
```

A resposta deve ser:

```
Eu não sei.
```

Referências

- Documentação da Plataforma OpenAI
- Documentação LangChain Memory
- Curso LangChain Chat with Your Data - DeepLearning.AI Short
- Call ChatGPT (or really any other API) from R
- Building a Portuguese Language RAG Pipeline using Sabia-7B, Qdrant, and LangChain