# Da Teoria à Prática: Modelos de IA Generativa com R e Python

### Consultando e Extraindo Informações de PDFs com IA

Magno T. F. Severino

2024-08-08

## Índice



code-block-bg: true

Organização e Apoio



Patrocínio


## Introdução

### Conceitos Básicos

### Requisitos

R e RStudio Instalado (link para instalação)

Python Instalado (inserir tutorial de instalação através do r)

ChatGTP API: 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.

## Conexão com a API da OpenAI

Na área de API Keys você deve criar 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

**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")

reticulate::py_run_string('
print("Hello, world!")
')

api_key_for_py <- r_to_py(Sys.getenv("OPENAI_API_KEY"))
```

## Download e importação do arquivo em PDF

**Listagem 3** `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")
```

## Divisão do documento em pedaços

## Avaliação do custo da aplicação

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

## Geração de *embeddings*

## Geração de respostas únicas

### Carregar embeddings

### Definir qual modelo de linguagem será usado

### Obtenção da resposta

## Geração de respostas como em uma conversa

### Junção da consulta com os pedaços do documento

**Listagem 4** Python

```python
from langchain.document_loaders import PyPDFLoader

my_loader = PyPDFLoader('docs/ggplot2.pdf')

print(type(my_loader))

all_pages = my_loader.load()

print(type(all_pages))

print(len(all_pages))
```

**Listagem 5** R

```r
# carregar documento em R

all_pages_in_r <- py$all_pages

all_pages_in_r[[1]]$metadata # metadatos do primeiro item

nchar(all_pages_in_r[[100]]$page_content) # conta a quantidade de caracteres do centésimo
```

### Construção da resposta

### Geração da resposta através de uma conversa

### Usando o chat via Python

**Listagem 6** Python

```
import openai
openai.api_key = r.api_key_for_py
from langchain.text_splitter import RecursiveCharacterTextSplitter
my_doc_splitter_recursive = RecursiveCharacterTextSplitter()
my_split_docs = my_doc_splitter_recursive.split_documents(all_pages)
```

**Listagem 7** R

```
my_split_docs <- py$my_split_docs
```

### Usando o chat via R

```
py_run_string('
print(qa_chain.run("How can I make a bar chart where the bars are steel blue? Answer in pt
')

py_run_string('
print(qa_chain.run("What is the capital of Australia?"))
')
```

**Listagem 8** R

```r
my_split_docs <- py$my_split_docs
length(my_split_docs)

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

**Listagem 9** R

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

**Listagem 10** Python

```python
import os
os.environ["OPENAI_API_KEY"] = r.api_key_for_py

from langchain_openai import OpenAIEmbeddings
embed_object = OpenAIEmbeddings()

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

vectordb = Chroma.from_documents(
    documents=my_split_docs,
    embedding=embed_object,
    persist_directory=chroma_store_directory
)

# numero de embeddings criados

print(vectordb._collection.count())
```

**Listagem 11** Python

```python
# carregar embeddings

import openai
openai.api_key = r.api_key_for_py
from langchain_community.embeddings import OpenAIEmbeddings
embed_object = OpenAIEmbeddings()

from langchain_community.vectorstores import Chroma
chroma_store_directory = "docs/chroma_db"
vectordb = Chroma(persist_directory=chroma_store_directory,
                  embedding_function=embed_object)
```

**Listagem 12** Python

```python
# Set up the LLM you want to use, in this example OpenAI's gpt-3.5-turbo
# from langchain.chat_models import ChatOpenAI
from langchain_community.chat_models import ChatOpenAI
from langchain_openai import ChatOpenAI
the_llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)

# Create a chain using the RetrievalQA component
from langchain.chains import RetrievalQA

qa_chain = RetrievalQA.from_chain_type(the_llm, retriever=vectordb.as_retriever())
```

**Listagem 13** Python

```python
my_question = "How do you rotate text on the x-axis of a graph?"
print(qa_chain.invoke(my_question))
```

**Listagem 14** Python

```python
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain.chains import create_retrieval_chain
from langchain.chains.combine_documents import create_stuff_documents_chain
from langchain_core.runnables.history import RunnableWithMessageHistory
from langchain_core.chat_history import BaseChatMessageHistory
from langchain_community.chat_message_histories import ChatMessageHistory
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder

retriever = vectordb.as_retriever()

llm = ChatOpenAI(model="gpt-3.5-turbo", temperature=0)

### Contextualize question ###
contextualize_q_system_prompt = (
    "Given a chat history and the latest user question "
    "which might reference context in the chat history, "
    "formulate a standalone question which can be understood "
    "without the chat history. Do NOT answer the question, "
    "just reformulate it if needed and otherwise return it as is."
)

contextualize_q_prompt = ChatPromptTemplate.from_messages(
    [
        ("system", contextualize_q_system_prompt),
        MessagesPlaceholder("chat_history"),
        ("human", "{input}"),
    ]
)

history_aware_retriever = create_history_aware_retriever(
    llm, retriever, contextualize_q_prompt
)
```

**Listagem 15** Pyhton

```
system_prompt = (
    "You are an assistant for question-answering tasks. "
    "Use the following pieces of retrieved context to answer "
    "the question. If you don't know the answer, say that you "
    "don't know. If the question is out of context of the "
    "retrieved context, do not answerr and just say it is out of context.\n\n"
    "{context}"
)

qa_prompt = ChatPromptTemplate.from_messages(
    [
        ("system", system_prompt),
        MessagesPlaceholder("chat_history"),
        ("human", "{input}"),
    ]
)

question_answer_chain = create_stuff_documents_chain(llm, qa_prompt)
```



```
rag_chain = create_retrieval_chain(history_aware_retriever, question_answer_chain)
```

**Listagem 16** Python

```python
store = {}

def get_session_history(session_id: str) -> BaseChatMessageHistory:
    if session_id not in store:
        store[session_id] = ChatMessageHistory()
    return store[session_id]


conversational_rag_chain = RunnableWithMessageHistory(
    rag_chain,
    get_session_history,
    input_messages_key="input",
    history_messages_key="chat_history",
    output_messages_key="answer",
)

def get_answer(question, session_id = "abc123"):
  result = conversational_rag_chain.invoke(
    {"input": question},
    config={"configurable": {"session_id": session_id}},
  )

  return result
```

**Listagem 17** Python

```python
while True:
  my_question = input("Diga: ")

  if my_question.lower() == 'sair':

    print("Encerrando o chatbot.")
    break

  result = get_answer(my_question, "abc123")


  print(result["answer"])
```