Chatting with my notes

Akash Garg

Version 1.0 - May 28, 2023

There's been a lot of chatter around langchain, so I decided to do a quick experiment with leveraging that against the openai api. It was surprisingly easy and quick to get everything setup and working. I was able to summarize and "chat" with my notes with a few hundred lines of code. Pretty incredible. I'm sure this will just keep getting better as context windows grow and more efficient representations are discovered. Pretty exciting future ahead.

```
import os
from pprint import pprint
from langchain.llms import OpenAI
from langchain.chains.summarize import load_summarize_chain
from langchain.text_splitter import RecursiveCharacterTextSplitter

# The vectorstore we'll be using
from langchain.vectorstores import Chroma

# The LangChain component we'll use to get the documents
from langchain.chains import RetrievalQA

# The easy document loader for text
from langchain.document_loaders import TextLoader
from langchain.document_loaders import UnstructuredMarkdownLoader

# The embedding engine that will convert our text to vectors
from langchain.embeddings.openai import OpenAIEmbeddings

OPENAI_API_KEY=os.environ.get('OPENAI_API_KEY')

llm = OpenAI(temperature=0, openai_api_key=OPENAI_API_KEY)

def print_color(text, color):
    print("\033[38;5;{}m{}\033[0m".format(color, text))

def load_documents(folder):
    def filetree(folder):
        return [os.path.join(dp, f) for dp, dn, fn in os.walk(os.path.expanduser(folder)) for f in fn]

    texts = []
    for file in filetree(folder):
        if file.endswith(".md"):
            print_color("loading " + file, 196)
            loader = UnstructuredMarkdownLoader(file)
            documents = loader.load_and_split()
            text_splitter = RecursiveCharacterTextSplitter(chunk_size=1500, chunk_overlap=10)
            texts.extend(text_splitter.split_documents(documents))
    return texts

def main():
    embeddings = OpenAIEmbeddings(openai_api_key=OPENAI_API_KEY)
    # Embed your documents and combine with the raw text in a pseudo db. Note: This will make an API call to OpenAI
```

```
    db = Chroma.from_documents(load_documents("data"), embeddings, persist_directory="docs.db")
    qa = RetrievalQA.from_chain_type(llm=llm, chain_type="stuff", retriever=db.as_retriever(), return_source_documents=True

    # Ask a question
    while True:
        question = input("Question: ")
        result = qa({"query": question})
        print_color("Answer: " + result["result"], 46)
        if os.environ.get("DEBUG", False):
            print_color("Source documents:", 46)
            pprint(result["source_documents"])

if __name__ == '__main__':
    main()
```

Revision	Date	Description
1.0	May 28, 2023	Creation