MultiQueryRetriever

Distance-based vector database retrieval embeds (represents) queries in high-dimensional space and finds similar embedded documents based on “distance”. But, retrieval may produce different results with subtle changes in query wording or if the embeddings do not capture the semantics of the data well. Prompt engineering / tuning is sometimes done to manually address these problems, but can be tedious.

The `MultiQueryRetriever` automates the process of prompt tuning by using an LLM to generate multiple queries from different perspectives for a given user input query. For each query, it retrieves a set of relevant documents and takes the unique union across all queries to get a larger set of potentially relevant documents. By generating multiple perspectives on the same question, the `MultiQueryRetriever` might be able to overcome some of the limitations of the distance-based retrieval and get a richer set of results.

```
# Build a sample vectorDB
from langchain.text_splitter import
RecursiveCharacterTextSplitter
from langchain_community.document_loaders impor
WebBaseLoader
from langchain_community.vectorstores import Ch
from langchain_openai import OpenAIEmbeddings

# Load blog post
loader =
WebBaseLoader("https://lilianweng.github.io/pos
06-23-agent/")
data = loader.load()

# Split
text_splitter =
RecursiveCharacterTextSplitter(chunk_size=500,
chunk_overlap=0)
splits = text_splitter.split_documents(data)

# VectorDB
embedding = OpenAIEmbeddings()
vectordb = Chroma.from_documents(documents=spli
embedding=embedding)
```

Simple usage

Specify the LLM to use for query generation, and the retriever will do the rest.

```
from langchain.retrievers.multi_query import
MultiQueryRetriever
from langchain_openai import ChatOpenAI

question = "What are the approaches to Task
Decomposition?"
llm = ChatOpenAI(temperature=0)
retriever_from_llm =
MultiQueryRetriever.from_llm(
    retriever=vectordb.as_retriever(),
llm=llm
)
```

```
# Set logging for the queries
import logging

logging.basicConfig()
logging.getLogger("langchain.retrievers.multi_q
```

```
unique_docs =
retriever_from_llm.get_relevant_documents(query=
len(unique_docs)
```

```
INFO:langchain.retrievers.multi_query:Generated
queries: ['1. How can Task Decomposition be
approached?', '2. What are the different
```

```
methods for Task Decomposition?', '3. What are
the various approaches to decomposing tasks?']
```

```
5
```

Supplying your own prompt

You can also supply a prompt along with an output parser to split the results into a list of queries.

```
from typing import List

from langchain.chains import LLMChain
from langchain.output_parsers import
PydanticOutputParser
from langchain.prompts import PromptTemplate
from pydantic import BaseModel, Field


# Output parser will split the LLM result
into a list of queries
class LineList(BaseModel):
    # "lines" is the key (attribute name) of
the parsed output
    lines: List[str] =
Field(description="Lines of text")


class
```

```
LineListOutputParser(PydanticOutputParser):
    def __init__(self) -> None:

super().__init__(pydantic_object=LineList)

    def parse(self, text: str) -> LineList:
        lines = text.strip().split("\n")
        return LineList(lines=lines)


output_parser = LineListOutputParser()

QUERY_PROMPT = PromptTemplate(
    input_variables=["question"],
    template="""You are an AI language model
assistant. Your task is to generate five
    different versions of the given user
question to retrieve relevant documents from
a vector
    database. By generating multiple
perspectives on the user question, your goal
is to help
    the user overcome some of the limitations
of the distance-based similarity search.
    Provide these alternative questions
separated by newlines.
    Original question: {question}""",
)
llm = ChatOpenAI(temperature=0)

# Chain
```

```
llm_chain = LLMChain(llm=llm,
prompt=QUERY_PROMPT,
output_parser=output_parser)

# Other inputs
question = "What are the approaches to Task
Decomposition?"
```

```
# Run
retriever = MultiQueryRetriever(
    retriever=vectordb.as_retriever(),
llm_chain=llm_chain, parser_key="lines"
)  # "lines" is the key (attribute name) of
the parsed output

# Results
unique_docs =
retriever.get_relevant_documents(
    query="What does the course say about
regression?"
)
len(unique_docs)
```

```
INFO:langchain.retrievers.multi_query:Generated
queries: ["1. What is the course's perspective
on regression?", '2. Can you provide
information on regression as discussed in the
course?', '3. How does the course cover the
```

```
topic of regression?', "4. What are the
course's teachings on regression?", '5. In
relation to the course, what is mentioned about
regression?']
```

```
11
```