LangChain with FAISS Vector DB

Example by Joselin James. Example was adapted to use README.md as the source of documents in the DB.

```
import os

os.environ["TRULENS_OTEL_TRACING"] = "0"
```

Import packages

```
# !pip install trulens trulens-apps-langchain faiss-cpu unstructured
```

```
from typing import List

from langchain.callbacks.manager import CallbackManagerForRetrieverRun
from langchain.chains import ConversationalRetrievalChain
from langchain.chat_models import ChatOpenAI
from langchain.document_loaders import UnstructuredMarkdownLoader
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain_core.documents import Document
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import FAISS
from langchain.vectorstores.base import VectorStoreRetriever
import nltk
import numpy as np
from trulens.core import Feedback
from trulens.core import Select
from trulens.core import TruSession
from trulens.apps.langchain import TruChain
```

Set API keys

```
import os

if "OPENAI_API_KEY" not in os.environ:
    os.environ["OPENAI_API_KEY"] = "sk-..."
```

Create vector db

```
# Create a local FAISS Vector DB based on README.md .
loader = UnstructuredMarkdownLoader("README.md")
nltk.download("averaged_perceptron_tagger")
documents = loader.load()

text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=0)
docs = text_splitter.split_documents(documents)

embeddings = OpenAIEmbeddings()
db = FAISS.from_documents(docs, embeddings)

# Save it.
db.save_local("faiss_index")
```

Create retriever

```
class VectorStoreRetrieverWithScore(VectorStoreRetriever):
    def _get_relevant_documents(
        self, query: str, *, run_manager: CallbackManagerForRetrieverRun
    ) -> List[Document]:
        if self.search_type == "similarity":
            docs_and_scores = (
                self.vectorstore.similarity_search_with_relevance_scores(
                    query, **self.search_kwargs
                )
            )

            print("From relevant doc in vec store")
            docs = []
            for doc, score in docs_and_scores:
                if score > 0.6:
                    doc.metadata["score"] = score
                    docs.append(doc)
        elif self.search_type == "mmr":
            docs = self.vectorstore.max_marginal_relevance_search(
                query, **self.search_kwargs
            )
        else:
            raise ValueError(f"search_type of {self.search_type} not allowed.")
        return docs
```

Create app

```
# Create the example app.
class FAISSWithScore(FAISS):
    def as_retriever(self) -> VectorStoreRetrieverWithScore:
        return VectorStoreRetrieverWithScore(
            vectorstore=self,
            search_type="similarity",
            search_kwargs={"k": 4},
        )


class FAISSStore:
    @staticmethod
    def load_vector_store():
        embeddings = OpenAIEmbeddings()
        faiss_store = FAISSWithScore.load_local(
            "faiss_index", embeddings, allow_dangerous_deserialization=True
        )
        print("Faiss vector DB loaded")
        return faiss_store
```

Set up evals

```
from trulens.providers.openai import OpenAI

# Create a feedback function.
openai = OpenAI()

f_context_relevance = (
    Feedback(openai.context_relevance, name="Context Relevance")
    .on_input()
    .on(
        Select.Record.app.combine_docs_chain._call.args.inputs.input_documents[
            :
        ].page_content
    )
    .aggregate(np.min)
)
```

```
# Bring it all together.

def load_conversational_chain(vector_store):
    llm = ChatOpenAI(
        temperature=0,
        model_name="gpt-4",
    )
    retriever = vector_store.as_retriever()
    chain = ConversationalRetrievalChain.from_llm(
        llm, retriever, return_source_documents=True
    )

    # workaround to avoid hitting ValueError: run not supported when there is not
exactly one output key. Got ['answer', 'source_documents'] in langchain/chains/
base.py:546, in Chain._run_output_key(self)
    chain.return_source_documents = False
    truchain = TruChain(chain, feedbacks=[f_context_relevance], with_hugs=False)
    chain.return_source_documents = True
    return chain, truchain
```

```
# Run example:
vector_store = FAISSStore.load_vector_store()
chain, tru_chain_recorder = load_conversational_chain(vector_store)

with tru_chain_recorder as recording:
    ret = chain({"question": "What is trulens?", "chat_history": ""})
```

```
# Check result.
ret
```

```
# Check that components of the app have been instrumented despite various
# subclasses used.
tru_chain_recorder.print_instrumented()
```

```
# Start dashboard to inspect records.
TruSession().run_dashboard()
```