Retrieval

Retrieval is the centerpiece of our retrieval augmented generation (RAG) flow.

Let's get our vectorDB from before.

Vectorstore retrieval

In [ ]:
```
import os
import openai
import sys
sys.path.append('../..')

from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv()) # read local .env file

openai.api_key  = os.environ['OPENAI_API_KEY']
```

In [ ]:
```
#!pip install lark
```

Similarity Search

In [ ]:
```
from langchain.vectorstores import Chroma
from langchain.embeddings.openai import OpenAIEmbeddings
persist_directory = 'docs/chroma/'
```

In [ ]:
```
embedding = OpenAIEmbeddings()
vectordb = Chroma(
    persist_directory=persist_directory,
    embedding_function=embedding
)
```

In [ ]:
```
print(vectordb._collection.count())
```

```
209
```

In [ ]:
```
texts = [
    """The Amanita phalloides has a large and imposing epigeous (aboveground) fruiting body (basidiocarp).""",
    """A mushroom with a large fruiting body is the Amanita phalloides. Some varieties are all-white.""",
    """A. phalloides, a.k.a Death Cap, is one of the most poisonous of all known mushrooms.""",
]
```

In [ ]:
```
smalldb = Chroma.from_texts(texts, embedding=embedding)
```

In [ ]:
```
question = "Tell me about all-white mushrooms with large fruiting bodies"
```

In [ ]:
```
smalldb.similarity_search(question, k=2)
```

Out[ ]:
```
[Document(page_content='A mushroom with a large fruiting body is the Amanita phalloides. Some varieties are all-white.', metadata={}),
 Document(page_content='The Amanita phalloides has a large and imposing epigeous (aboveground) fruiting body (basidiocarp).', metadata={})]
```

In [ ]:
```
smalldb.max_marginal_relevance_search(question,k=2, fetch_k=3)
```

Out[ ]:
```
[Document(page_content='A mushroom with a large fruiting body is the Amanita phalloides. Some varieties are all-white.', metadata={}),
 Document(page_content='A. phalloides, a.k.a Death Cap, is one of the most poisonous of all known mushrooms.', metadata={})]
```

Addressing Diversity: Maximum marginal relevance

Last class we introduced one problem: how to enforce diversity in the search results.

`Maximum marginal relevance` strives to achieve both relevance to the query *and diversity* among the results.

In [ ]:
```
question = "what did they say about matlab?"
docs_ss = vectordb.similarity_search(question,k=3)
```

In [ ]:
```
docs_ss[0].page_content[:100]
```

Out[ ]:
```
'those homeworks will be done in either MATLA B or in Octave, which is sort of — I \nknow some people '
```

In [ ]:
```
docs_ss[1].page_content[:100]
```

Out[ ]:
```
'those homeworks will be done in either MATLA B or in Octave, which is sort of — I \nknow some people '
```

Note the difference in results with `MMR`.

In [ ]:
```
docs_mmr = vectordb.max_marginal_relevance_search(question,k=3)
```

In [ ]:
```
docs_mmr[0].page_content[:100]
```

Out[ ]:
```
'those homeworks will be done in either MATLA B or in Octave, which is sort of — I \nknow some people '
```

In [ ]:
```
docs_mmr[1].page_content[:100]
```

Out[ ]:
```
'algorithm then? So what's different? How come  I was making all that noise earlier about \nleast squa'
```

Addressing Specificity: working with metadata

In last lecture, we showed that a question about the third lecture can include results from other lectures as well.

To address this, many vectorstores support operations on `metadata`.

`metadata` provides context for each embedded chunk.

In [ ]:
```
question = "what did they say about regression in the third lecture?"
```

In [ ]:
```
docs = vectordb.similarity_search(
    question,
    k=3,
    filter={"source":"docs/cs229_lectures/MachineLearning-Lecture03.pdf"}
)
```

In [ ]:
```
for d in docs:
    print(d.metadata)
```

```
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 0}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 14}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 4}
```

In [ ]:
```

```

Addressing Specificity: working with metadata using self-query retriever

But we have an interesting challenge: we often want to infer the metadata from the query itself.

To address this, we can use `SelfQueryRetriever`, which uses an LLM to extract:

1. The `query` string to use for vector search
2. A metadata filter to pass in as well

Most vector databases support metadata filters, so this doesn't require any new databases or indexes.

In [ ]:
```
from langchain.llms import OpenAI
from langchain.retrievers.self_query.base import SelfQueryRetriever
from langchain.chains.query_constructor.base import AttributeInfo
```

In [ ]:
```
metadata_field_info = [
    AttributeInfo(
        name="source",
        description="The lecture the chunk is from, should be one of `docs/cs229_lectures/MachineLearning-Lecture01.pdf`,
        type="string",
    ),
    AttributeInfo(
        name="page",
        description="The page from the lecture",
        type="integer",
    ),
]
```

Note: The default model for `OpenAI` ("from langchain.llms import OpenAI") is `text-davinci-003`. Due to the deprication of OpenAI's model `text-davinci-003` on 4 January 2024, you'll be using OpenAI's recommended replacement model `gpt-3.5-turbo-instruct` instead.

In [ ]:
```
document_content_description = "Lecture notes"
llm = OpenAI(model='gpt-3.5-turbo-instruct', temperature=0)
retriever = SelfQueryRetriever.from_llm(
    llm,
    vectordb,
    document_content_description,
    metadata_field_info,
    verbose=True
)
```

In [ ]:
```
question = "what did they say about regression in the third lecture?"
```

You will receive a warning about predict_and_parse being deprecated the first time you executing the next line. This can be safely ignored.

In [ ]:
```
docs = retriever.get_relevant_documents(question)
```

```
/usr/local/lib/python3.9/site-packages/langchain/chains/llm.py:275: UserWarning: The predict_and_parse method is deprecated
, instead pass an output parser directly to LLMChain.
  warnings.warn(
query='regression' filter=Comparison(comparator=<Comparator.EQ: 'eq'>, attribute='source', value='docs/cs229_lectures/Machi
neLearning-Lecture03.pdf') limit=None
```

In [ ]:
```
for d in docs:
    print(d.metadata)
```

```
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 14}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 0}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 10}
{'source': 'docs/cs229_lectures/MachineLearning-Lecture03.pdf', 'page': 10}
```

Additional tricks: compression

Another approach for improving the quality of retrieved docs is compression.

Information most relevant to a query may be buried in a document with a lot of irrelevant text.

Passing that full document through your application can lead to more expensive LLM calls and poorer responses.

Contextual compression is meant to fix this.

In [ ]:
```
from langchain.retrievers import ContextualCompressionRetriever
from langchain.retrievers.document_compressors import LLMChainExtractor
```

In [ ]:
```
def pretty_print_docs(docs):
    print(f"\n{'-' * 100}\n".join([f"Document {i+1}:\n\n" + d.page_content for i, d in enumerate(docs)]))
```

In [ ]:
```
# Wrap our vectorstore
llm = OpenAI(temperature=0, model="gpt-3.5-turbo-instruct")
compressor = LLMChainExtractor.from_llm(llm)
```

In [ ]:
```
compression_retriever = ContextualCompressionRetriever(
    base_compressor=compressor,
    base_retriever=vectordb.as_retriever()
)
```

In [ ]:
```
question = "what did they say about matlab?"
compressed_docs = compression_retriever.get_relevant_documents(question)
pretty_print_docs(compressed_docs)
```

```
Document 1:

- "those homeworks will be done in either MATLA B or in Octave"
- "I know some people call it a free ve rsion of MATLAB"
- "MATLAB is I guess part of the programming language that makes it very easy to write codes using matrices, to write code
for numerical routines, to move data around, to plot data."
- "there's also a software package called Octave that you can download for free off the Internet."
- "it has somewhat fewer features than MATLAB, but it's free, and for the purposes of this class, it will work for just abo
ut everything."
- "once a colleague of mine at a different university, not at Stanford, actually teaches another machine learning course."
----------------------------------------------------------------------------------------------------
Document 2:

- "those homeworks will be done in either MATLA B or in Octave"
- "I know some people call it a free ve rsion of MATLAB"
- "MATLAB is I guess part of the programming language that makes it very easy to write codes using matrices, to write code
for numerical routines, to move data around, to plot data."
- "there's also a software package called Octave that you can download for free off the Internet."
- "it has somewhat fewer features than MATLAB, but it's free, and for the purposes of this class, it will work for just abo
ut everything."
- "once a colleague of mine at a different university, not at Stanford, actually teaches another machine learning course."
----------------------------------------------------------------------------------------------------
Document 3:

"Oh, it was the MATLAB."
----------------------------------------------------------------------------------------------------
Document 4:

"Oh, it was the MATLAB."
```

Combining various techniques

In [ ]:
```
compression_retriever = ContextualCompressionRetriever(
    base_compressor=compressor,
    base_retriever=vectordb.as_retriever(search_type = "mmr")
)
```

In [ ]:
```
question = "what did they say about matlab?"
compressed_docs = compression_retriever.get_relevant_documents(question)
pretty_print_docs(compressed_docs)
```

```
Document 1:

- "those homeworks will be done in either MATLA B or in Octave"
- "I know some people call it a free ve rsion of MATLAB"
- "MATLAB is I guess part of the programming language that makes it very easy to write codes using matrices, to write code
for numerical routines, to move data around, to plot data."
- "there's also a software package called Octave that you can download for free off the Internet."
- "it has somewhat fewer features than MATLAB, but it's free, and for the purposes of this class, it will work for just abo
ut everything."
- "once a colleague of mine at a different university, not at Stanford, actually teaches another machine learning course."
----------------------------------------------------------------------------------------------------
Document 2:

"Oh, it was the MATLAB."
----------------------------------------------------------------------------------------------------
Document 3:

- learning algorithms to teach a car how to drive at reasonably high speeds off roads avoiding obstacles.
- that's a robot program med by PhD student Eva Roshen to teach a sort of somewhat strangely configured robot how to get on
top of an obstacle, how to get over an obstacle.
- So I think all of these are robots that I think are very difficult to hand-code a controller for by learning these sorts
of learning algorithms.
- Just a couple more last things, but let me just check what questions you have right now.
- So if there are no questions, I'll just close with two reminders, which are after class today or as you start to talk wit
h other people in this class, I just encourage you again to start to form project partners, to try to find project partners
to do your project with.
- And also, this is a good time to start forming study groups, so either talk to your friends or post in the newsgroup, but
we just encourage you to try to start to do both of those today, okay? Form study groups, and try to find two other project
partners.
```

Other types of retrieval

It's worth noting that vectordb as not the only kind of tool to retrieve documents.

The `LangChain` retriever abstraction includes other ways to retrieve documents, such as TF-IDF or SVM.

In [ ]:
```
from langchain.retrievers import SVMRetriever
from langchain.retrievers import TFIDFRetriever
from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
```

In [ ]:
```
# Load PDF
loader = PyPDFLoader("docs/cs229_lectures/MachineLearning-Lecture01.pdf")
pages = loader.load()
all_page_text=[p.page_content for p in pages]
joined_page_text=" ".join(all_page_text)

# Split
text_splitter = RecursiveCharacterTextSplitter(chunk_size = 1500,chunk_overlap = 150)
splits = text_splitter.split_text(joined_page_text)
```

In [ ]:
```
# Retrieve
svm_retriever = SVMRetriever.from_texts(splits,embedding)
tfidf_retriever = TFIDFRetriever.from_texts(splits)
```

In [ ]:
```
question = "What are major topics for this class?"
docs_svm=svm_retriever.get_relevant_documents(question)
docs_svm[0]
```

```
/usr/local/lib/python3.9/site-packages/sklearn/svm/_classes.py:32: FutureWarning: The default value of `dual` will change f
rom `True` to `'auto'` in 1.5. Set the value of `dual` explicitly to suppress the warning.
  warnings.warn(
```

Out[ ]:
```
Document(page_content="let me just check what questions you have righ t now. So if there are no questions, I'll just \ncl
ose with two reminders, which are after class today or as you start to talk with other \npeople in this class, I just enc
ourage you again to start to form project partners, to try to \nfind project partners to do your project with. And also,
this is a good time to start forming \nstudy groups, so either talk to your friends  or post in the newsgroup, but we jus
t \nencourage you to try to star t to do both of those today, okay? Form study groups, and try \nto find two other projec
t partners.  \nSo thank you. I'm looking forward to teaching this class, and I'll see you in a couple of \ndays.  [End o
f Audio]  \nDuration: 69 minutes", metadata={})
```

In [ ]:
```
question = "what did they say about matlab?"
docs_tfidf=tfidf_retriever.get_relevant_documents(question)
docs_tfidf[0]
```

In [ ]: