Faiss

Open in Colab

Open on GitHub

> Facebook AI Similarity Search (FAISS) is a library for efficient similarity search and clustering of dense vectors. It contains algorithms that search in sets of vectors of any size, up to ones that possibly do not fit in RAM. It also includes supporting code for evaluation and parameter tuning.
>
> See The FAISS Library paper.

You can find the FAISS documentation at this page.

This notebook shows how to use functionality related to the `FAISS` vector database. It will show functionality specific to this integration. After going through, it may be useful to explore relevant use-case pages to learn how to use this vectorstore as part of a larger chain.

Setup

The integration lives in the `langchain-community` package. We also need to install the `faiss` package itself. We can install these with:

Note that you can also install `faiss-gpu` if you want to use the GPU enabled version

```
pip install -qU langchain-community faiss-cpu
```

If you want to get best in-class automated tracing of your model calls you can also set your LangSmith API key by uncommenting below:

```
# os.environ["LANGSMITH_TRACING"] = "true"
# os.environ["LANGSMITH_API_KEY"] = getpass.getpass()
```

Initialization

Select **embeddings model**: OpenAI ▾

```
pip install -qU langchain-openai
```

```
import getpass
import os

if not os.environ.get("OPENAI_API_KEY"):
  os.environ["OPENAI_API_KEY"] = getpass.getpass("Enter API key for OpenAI:
")

from langchain_openai import OpenAIEmbeddings

embeddings = OpenAIEmbeddings(model="text-embedding-3-large")
```

```
import faiss
from langchain_community.docstore.in_memory import InMemoryDocstore
from langchain_community.vectorstores import FAISS

index = faiss.IndexFlatL2(len(embeddings.embed_query("hello world")))

vector_store = FAISS(
    embedding_function=embeddings,
    index=index,
    docstore=InMemoryDocstore(),
    index_to_docstore_id={},
)
```

Manage vector store

Add items to vector store

```
from uuid import uuid4

from langchain_core.documents import Document
```

```
document_1 = Document(
    page_content="I had chocolate chip pancakes and scrambled eggs for
breakfast this morning.",
    metadata={"source": "tweet"},
)

document_2 = Document(
    page_content="The weather forecast for tomorrow is cloudy and overcast,
with a high of 62 degrees.",
    metadata={"source": "news"},
)

document_3 = Document(
    page_content="Building an exciting new project with LangChain - come
check it out!",
    metadata={"source": "tweet"},
)

document_4 = Document(
    page_content="Robbers broke into the city bank and stole $1 million in
cash.",
    metadata={"source": "news"},
)

document_5 = Document(
    page_content="Wow! That was an amazing movie. I can't wait to see it
again.",
    metadata={"source": "tweet"},
)

document_6 = Document(
    page_content="Is the new iPhone worth the price? Read this review to find
out.",
    metadata={"source": "website"},
)

document_7 = Document(
    page_content="The top 10 soccer players in the world right now.",
    metadata={"source": "website"},
)

document_8 = Document(
    page_content="LangGraph is the best framework for building stateful,
agentic applications!",
    metadata={"source": "tweet"},
)
```

```
document_9 = Document(
    page_content="The stock market is down 500 points today due to fears of a
recession.",
    metadata={"source": "news"},
)

document_10 = Document(
    page_content="I have a bad feeling I am going to get deleted :(",
    metadata={"source": "tweet"},
)

documents = [
    document_1,
    document_2,
    document_3,
    document_4,
    document_5,
    document_6,
    document_7,
    document_8,
    document_9,
    document_10,
]
uuids = [str(uuid4()) for _ in range(len(documents))]

vector_store.add_documents(documents=documents, ids=uuids)
```

API Reference: Document

```
['22f5ce99-cd6f-4e0c-8dab-664128307c72',
 'dc3f061b-5f88-4fa1-a966-413550c51891',
 'd33d890b-baad-47f7-b7c1-175f5f7b4e59',
 '6e6c01d2-6020-4a7b-95da-ef43d43f01b5',
 'e677223d-ad75-4c1a-bef6-b5912bd1de03',
 '47e2a168-6462-4ed2-b1d9-d9edfd7391d6',
 '1e4d66d6-e155-4891-9212-f7be97f36c6a',
 'c0663096-e1a5-4665-b245-1c2e6c4fb653',
 '8297474a-7f7c-4006-9865-398c1781b1bc',
 '44e4be03-0a8d-4316-b3c4-f35f4bb2b532']
```

Delete items from vector store

```
vector_store.delete(ids=[uuids[-1]])
```

```
True
```

Query vector store

Once your vector store has been created and the relevant documents have been added you will most likely wish to query it during the running of your chain or agent.

Query directly

Similarity search

Performing a simple similarity search with filtering on metadata can be done as follows:

```
results = vector_store.similarity_search(
    "LangChain provides abstractions to make working with LLMs easy",
    k=2,
    filter={"source": "tweet"},
)
for res in results:
    print(f"* {res.page_content} [{res.metadata}]")
```

```
* Building an exciting new project with LangChain - come check it out!
[{'source': 'tweet'}]
* LangGraph is the best framework for building stateful, agentic applications!
[{'source': 'tweet'}]
```

Some MongoDB query and projection operators are supported for more advanced metadata filtering. The current list of supported operators are as follows:

- `$eq` (equals)
- `$neq` (not equals)
- `$gt` (greater than)
- `$lt` (less than)

- `$gte` (greater than or equal)
- `$lte` (less than or equal)
- `$in` (membership in list)
- `$nin` (not in list)
- `$and` (all conditions must match)
- `$or` (any condition must match)
- `$not` (negation of condition)

Performing the same above similarity search with advanced metadata filtering can be done as follows:

```
results = vector_store.similarity_search(
    "LangChain provides abstractions to make working with LLMs easy",
    k=2,
    filter={"source": {"$eq": "tweet"}},
)
for res in results:
    print(f"* {res.page_content} [{res.metadata}]")
```

```
* Building an exciting new project with LangChain - come check it out!
[{'source': 'tweet'}]
* LangGraph is the best framework for building stateful, agentic applications!
[{'source': 'tweet'}]
```

Similarity search with score

You can also search with score:

```
results = vector_store.similarity_search_with_score(
    "Will it be hot tomorrow?", k=1, filter={"source": "news"}
)
for res, score in results:
    print(f"* [SIM={score:3f}] {res.page_content} [{res.metadata}]")
```

```
* [SIM=0.893688] The weather forecast for tomorrow is cloudy and overcast,
with a high of 62 degrees. [{'source': 'news'}]
```

Other search methods

There are a variety of other ways to search a FAISS vector store. For a complete list of those methods, please refer to the API Reference

Query by turning into retriever

You can also transform the vector store into a retriever for easier usage in your chains.

```
retriever = vector_store.as_retriever(search_type="mmr", search_kwargs={"k": 1})
retriever.invoke("Stealing from the bank is a crime", filter={"source": "news"})
```

```
[Document(metadata={'source': 'news'}, page_content='Robbers broke into the city bank and stole $1 million in cash.')]
```

Usage for retrieval-augmented generation

For guides on how to use this vector store for retrieval-augmented generation (RAG), see the following sections:

- Tutorials
- How-to: Question and answer with RAG
- Retrieval conceptual docs

Saving and loading

You can also save and load a FAISS index. This is useful so you don't have to recreate it everytime you use it.

```
vector_store.save_local("faiss_index")

new_vector_store = FAISS.load_local(
    "faiss_index", embeddings, allow_dangerous_deserialization=True
)
```

```
docs = new_vector_store.similarity_search("qux")
```

```
docs[0]
```

```
Document(metadata={'source': 'tweet'}, page_content='Building an exciting new
project with LangChain - come check it out!')
```

Merging

You can also merge two FAISS vectorstores

```
db1 = FAISS.from_texts(["foo"], embeddings)
db2 = FAISS.from_texts(["bar"], embeddings)

db1.docstore._dict
```

```
{'b752e805-350e-4cf5-ba54-0883d46a3a44': Document(page_content='foo')}
```

```
db2.docstore._dict
```

```
{'08192d92-746d-4cd1-b681-bdfba411f459': Document(page_content='bar')}
```

```
db1.merge_from(db2)
```

```
db1.docstore._dict
```

```
{'b752e805-350e-4cf5-ba54-0883d46a3a44': Document(page_content='foo'),
 '08192d92-746d-4cd1-b681-bdfba411f459': Document(page_content='bar')}
```

API reference

For detailed documentation of all `FAISS` vector store features and configurations head to the API reference: https://python.langchain.com/api_reference/community/vectorstores/langchain_community.vectorstores.faiss.FAISS.html

Related

- Vector store conceptual guide
- Vector store how-to guides

✏ Edit this page