🏠   Modules   Data connection   Retrievers   Self-querying

Self-querying with Pinecone

Self-querying with Pinecone

In the walkthrough we'll demo the `SelfQueryRetriever` with a `Pinecone` vector store.

Creating a Pinecone index

First we'll want to create a `Pinecone` VectorStore and seed it with some data. We've created a small demo set of documents that contain summaries of movies.

To use Pinecone, you have to have `pinecone` package installed and you must have an API key and an Environment. Here are the installation instructions.

NOTE: The self-query retriever requires you to have `lark` package installed.

```
# !pip install lark
```

```
#!pip install pinecone-client
```

```
import os

import pinecone


pinecone.init(
    api_key=os.environ["PINECONE_API_KEY"],
environment=os.environ["PINECONE_ENV"]
)
```

```
/Users/harrisonchase/.pyenv/versions/3.9.1/envs/langchain/lib/python3.9/si
packages/pinecone/index.py:4: TqdmExperimentalWarning: Using
`tqdm.autonotebook.tqdm` in notebook mode. Use `tqdm.tqdm` instead to for
console mode (e.g. in jupyter console)
    from tqdm.autonotebook import tqdm
```

```python
from langchain.schema import Document
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.vectorstores import Pinecone

embeddings = OpenAIEmbeddings()
# create new index
pinecone.create_index("langchain-self-retriever-demo", dimension=1536)
```

API Reference:

- Document from `langchain.schema`

- OpenAIEmbeddings from `langchain.embeddings.openai`

- Pinecone from `langchain.vectorstores`

```python
docs = [
    Document(
        page_content="A bunch of scientists bring back dinosaurs and
mayhem breaks loose",
        metadata={"year": 1993, "rating": 7.7, "genre": ["action",
"science fiction"]},
    ),
    Document(
        page_content="Leo DiCaprio gets lost in a dream within a dream
within a dream within a ...",
        metadata={"year": 2010, "director": "Christopher Nolan",
"rating": 8.2},
    ),
    Document(
        page_content="A psychologist / detective gets lost in a series
of dreams within dreams within dreams and Inception reused the idea",
        metadata={"year": 2006, "director": "Satoshi Kon", "rating":
8.6},
    ),
    Document(
        page_content="A bunch of normal-sized women are supremely
wholesome and some men pine after them",
        metadata={"year": 2019, "director": "Greta Gerwig", "rating":
8.3},
    ),
    Document(
        page_content="Toys come alive and have a blast doing so",
        metadata={"year": 1995, "genre": "animated"},
    ),
    Document(
```

```python
        page_content="Three men walk into the Zone, three men walk out
of the Zone",
        metadata={
            "year": 1979,
            "rating": 9.9,
            "director": "Andrei Tarkovsky",
            "genre": ["science fiction", "thriller"],
            "rating": 9.9,
        },
    ),
]
vectorstore = Pinecone.from_documents(
    docs, embeddings, index_name="langchain-self-retriever-demo"
)
```

Creating our self-querying retriever

Now we can instantiate our retriever. To do this we'll need to provide some information upfront about the metadata fields that our documents support and a short description of the document contents.

```python
from langchain.llms import OpenAI
from langchain.retrievers.self_query.base import SelfQueryRetriever
from langchain.chains.query_constructor.base import AttributeInfo

metadata_field_info = [
    AttributeInfo(
        name="genre",
        description="The genre of the movie",
        type="string or list[string]",
    ),
    AttributeInfo(
        name="year",
        description="The year the movie was released",
        type="integer",
    ),
    AttributeInfo(
        name="director",
        description="The name of the movie director",
        type="string",
    ),
    AttributeInfo(
        name="rating", description="A 1-10 rating for the movie",
        type="float"
```

```
        ),
    ]
document_content_description = "Brief summary of a movie"
llm = OpenAI(temperature=0)
retriever = SelfQueryRetriever.from_llm(
    llm, vectorstore, document_content_description,
metadata_field_info, verbose=True
)
```

API Reference:

- OpenAI from `langchain.llms`
- SelfQueryRetriever from `langchain.retrievers.self_query.base`
- AttributeInfo from `langchain.chains.query_constructor.base`

Testing it out

And now we can try actually using our retriever!

```
# This example only specifies a relevant query
retriever.get_relevant_documents("What are some movies about
dinosaurs")
```

```
    query='dinosaur' filter=None




    [Document(page_content='A bunch of scientists bring back dinosaurs
and mayhem breaks loose', metadata={'genre': ['action', 'science
fiction'], 'rating': 7.7, 'year': 1993.0}),
     Document(page_content='Toys come alive and have a blast doing so',
metadata={'genre': 'animated', 'year': 1995.0}),
     Document(page_content='A psychologist / detective gets lost in a
series of dreams within dreams within dreams and Inception reused the
idea', metadata={'director': 'Satoshi Kon', 'rating': 8.6, 'year':
2006.0}),
     Document(page_content='Leo DiCaprio gets lost in a dream within a
dream within a dream within a ...', metadata={'director': 'Christopher
Nolan', 'rating': 8.2, 'year': 2010.0})]
```

```python
# This example only specifies a filter
retriever.get_relevant_documents("I want to watch a movie rated higher than 8.5")
```

```
    query=' ' filter=Comparison(comparator=<Comparator.GT: 'gt'>, attribute='rating', value=8.5)
```

```
    [Document(page_content='A psychologist / detective gets lost in a series of dreams within dreams within dreams and Inception reused the idea', metadata={'director': 'Satoshi Kon', 'rating': 8.6, 'year': 2006.0}),
     Document(page_content='Three men walk into the Zone, three men walk out of the Zone', metadata={'director': 'Andrei Tarkovsky', 'genre': ['science fiction', 'thriller'], 'rating': 9.9, 'year': 1979.0})]
```

```python
# This example specifies a query and a filter
retriever.get_relevant_documents("Has Greta Gerwig directed any movies about women")
```

```
    query='women' filter=Comparison(comparator=<Comparator.EQ: 'eq'>, attribute='director', value='Greta Gerwig')
```

```
    [Document(page_content='A bunch of normal-sized women are supremely wholesome and some men pine after them', metadata={'director': 'Greta Gerwig', 'rating': 8.3, 'year': 2019.0})]
```

```python
# This example specifies a composite filter
retriever.get_relevant_documents(
    "What's a highly rated (above 8.5) science fiction film?"
)
```

```
    query=' ' filter=Operation(operator=<Operator.AND: 'and'>,
arguments=[Comparison(comparator=<Comparator.EQ: 'eq'>,
attribute='genre', value='science fiction'), Comparison(comparator=
<Comparator.GT: 'gt'>, attribute='rating', value=8.5)])
```

```
    [Document(page_content='Three men walk into the Zone, three men
walk out of the Zone', metadata={'director': 'Andrei Tarkovsky',
'genre': ['science fiction', 'thriller'], 'rating': 9.9, 'year':
1979.0})]
```

```python
# This example specifies a query and composite filter
retriever.get_relevant_documents(
    "What's a movie after 1990 but before 2005 that's all about toys,
and preferably is animated"
)
```

```
    query='toys' filter=Operation(operator=<Operator.AND: 'and'>,
arguments=[Comparison(comparator=<Comparator.GT: 'gt'>,
attribute='year', value=1990.0), Comparison(comparator=<Comparator.LT:
'lt'>, attribute='year', value=2005.0), Comparison(comparator=
<Comparator.EQ: 'eq'>, attribute='genre', value='animated')])
```

```
    [Document(page_content='Toys come alive and have a blast doing so',
metadata={'genre': 'animated', 'year': 1995.0})]
```

Filter k

We can also use the self query retriever to specify `k`: the number of documents to fetch.

We can do this by passing `enable_limit=True` to the constructor.

```python
retriever = SelfQueryRetriever.from_llm(
    llm,
```

```python
    vectorstore,
    document_content_description,
    metadata_field_info,
    enable_limit=True,
    verbose=True,
)
```

```python
# This example only specifies a relevant query
retriever.get_relevant_documents("What are two movies about dinosaurs")
```