Time-weighted vector store retriever

This retriever uses a combination of semantic similarity and a time decay.

The algorithm for scoring them is:

```
semantic_similarity + (1.0 - decay_rate) ^
hours_passed
```

Notably, `hours_passed` refers to the hours passed since the object in the retriever **was last accessed**, not since it was created. This means that frequently accessed objects remain "fresh".

```
from datetime import datetime, timedelta

import faiss
from langchain.docstore import
InMemoryDocstore
from langchain.retrievers import
```

```
TimeWeightedVectorStoreRetriever
from langchain.schema import Document
from langchain_community.vectorstores import
FAISS
from langchain_openai import OpenAIEmbeddings
```

Low decay rate

A low `decay rate` (in this, to be extreme, we will set it close to 0) means memories will be "remembered" for longer. A `decay rate` of 0 means memories never be forgotten, making this retriever equivalent to the vector lookup.

```
# Define your embedding model
embeddings_model = OpenAIEmbeddings()
# Initialize the vectorstore as empty
embedding_size = 1536
index = faiss.IndexFlatL2(embedding_size)
vectorstore = FAISS(embeddings_model, index,
InMemoryDocstore({}), {})
retriever = TimeWeightedVectorStoreRetriever(
    vectorstore=vectorstore,
decay_rate=0.0000000000000000000000001, k=1
)
```

```
yesterday = datetime.now() - timedelta(days=1)
retriever.add_documents(
    [Document(page_content="hello world", metad
{"last_accessed_at": yesterday})]
)
retriever.add_documents([Document(page_content=
foo")])
```

```
['c3dcf671-3c0a-4273-9334-c4a913076bfa']
```

```
# "Hello World" is returned first because it
is most salient, and the decay rate is close
to 0., meaning it's still recent enough
retriever.get_relevant_documents("hello
world")
```

```
[Document(page_content='hello world',
metadata={'last_accessed_at':
datetime.datetime(2023, 12, 27, 15, 30, 18,
457125), 'created_at':
datetime.datetime(2023, 12, 27, 15, 30, 8,
442662), 'buffer_idx': 0})]
```

High decay rate

With a high `decay rate` (e.g., several 9's), the `recency score` quickly goes to 0! If you set this all the way to 1, `recency` is 0 for all objects, once again making this equivalent to a vector lookup.

```
# Define your embedding model
embeddings_model = OpenAIEmbeddings()
# Initialize the vectorstore as empty
embedding_size = 1536
index = faiss.IndexFlatL2(embedding_size)
vectorstore = FAISS(embeddings_model, index,
InMemoryDocstore({}), {})
retriever = TimeWeightedVectorStoreRetriever(
    vectorstore=vectorstore,
decay_rate=0.999, k=1
)
```

```
yesterday = datetime.now() - timedelta(days=1)
retriever.add_documents(
    [Document(page_content="hello world", metad
{"last_accessed_at": yesterday})]
)
retriever.add_documents([Document(page_content=
foo")])
```

```
['eb1c4c86-01a8-40e3-8393-9a927295a950']
```

```
# "Hello Foo" is returned first because
"hello world" is mostly forgotten
retriever.get_relevant_documents("hello
world")
```

```
[Document(page_content='hello foo', metadata=
{'last_accessed_at': datetime.datetime(2023,
12, 27, 15, 30, 50, 57185), 'created_at':
datetime.datetime(2023, 12, 27, 15, 30, 44,
720490), 'buffer_idx': 1})]
```

Virtual time

Using some utils in LangChain, you can mock out the time component.

```
import datetime

from langchain.utils import mock_now
```

```
# Notice the last access time is that date
time
with mock_now(datetime.datetime(2024, 2, 3,
10, 11)):
```

```
print(retriever.get_relevant_documents("hello world"))
```

```
[Document(page_content='hello world',
metadata={'last_accessed_at':
MockDateTime(2024, 2, 3, 10, 11),
'created_at': datetime.datetime(2023, 12, 27,
15, 30, 44, 532941), 'buffer_idx': 0})]
```