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."

```
import faiss

from datetime import datetime, timedelta
from langchain.docstore import InMemoryDocstore
from langchain.embeddings import OpenAIEmbeddings
from langchain.retrievers import TimeWeightedVectorStoreRetriever
from langchain.schema import Document
from langchain.vectorstores import FAISS
```

Low Decay Rate

A low `decay rate` (in this, to be extreme, we will set 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.embed_query, index, InMemoryDocstore({}), {})
retriever = TimeWeightedVectorStoreRetriever(vectorstore=vectorstore, decay_rate=.0000000000000000000000001,
k=1)
```

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

```
    ['d7f85756-2371-4bdf-9140-052780a0f9b3']
```

```
# "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, 5, 13, 21, 0,
27, 678341), 'created_at': datetime.datetime(2023, 5, 13, 21, 0, 27, 279596), '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.embed_query, index, InMemoryDocstore({}), {})
retriever = TimeWeightedVectorStoreRetriever(vectorstore=vectorstore, decay_rate=.999, k=1)
```

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

```
    ['40011466-5bbe-4101-bfd1-e22e7f505de2']
```

```
# "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, 4, 16, 22, 9,
2, 494798), 'created_at': datetime.datetime(2023, 4, 16, 22, 9, 2, 178722), 'buffer_idx': 1})]
```

Virtual Time

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

```
from langchain.utils import mock_now
import datetime
```

```
# Notice the last access time is that date time
with mock_now(datetime.datetime(2011, 2, 3, 10, 11)):
    print(retriever.get_relevant_documents("hello world"))
```

```
    [Document(page_content='hello world', metadata={'last_accessed_at': MockDateTime(2011, 2, 3, 10, 11),
'created_at': datetime.datetime(2023, 5, 13, 21, 0, 27, 279596), 'buffer_idx': 0})]
```