Jacob Orshalick

https://linkedin.com/in/jorshalick

How to setup a

VECTOR SEARCH WITH YOUR DATA

Using JavaScript express

AI for Developers, not Data Scientists | Part 3

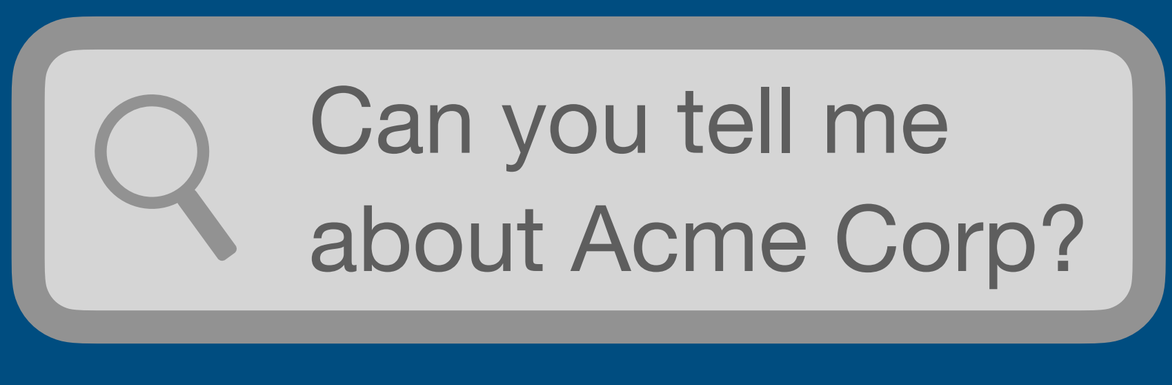
LangChain

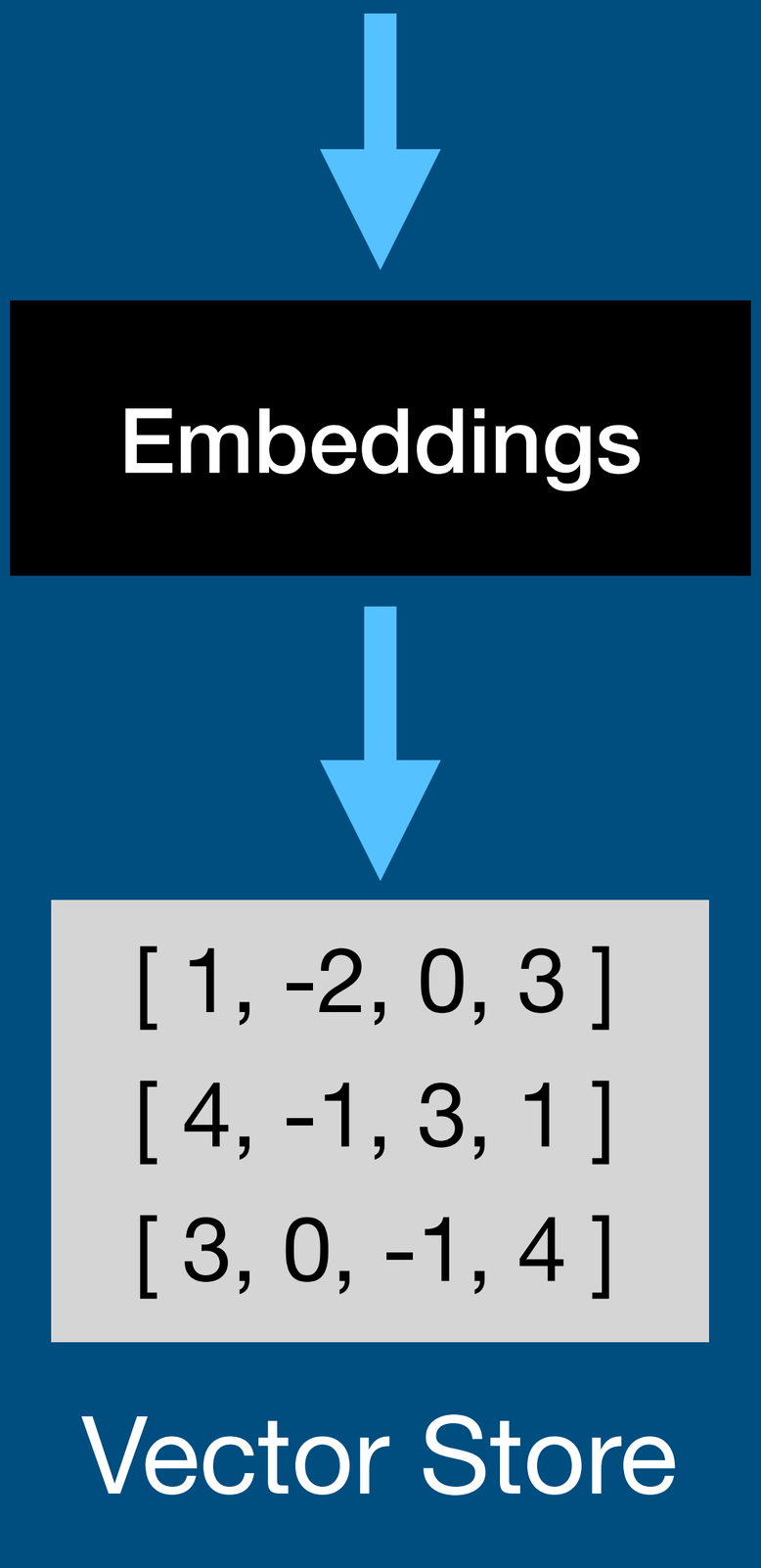
node JS

OpenAI

Why Vector Search?

- Allows you to search your data based on the meaning of text instead of keywords
- Your documents and search queries get converted to numeric vectors by an embedding provider
- The Vector Store applies an algorithm to find the most similar vector during search





1

Install Node dependencies

- We'll be using LangChain.js with `HNSWLib` to build our Vector Search:
  - `HNSWLib` provides a Vector Store that can be saved and loaded from disk
  - LangChain is a framework for building applications powered by LLMs
  - Install the dependencies in a Node project by running the following in your terminal:

```
~/ai-for-developers/part3 % npm install \
  hnswlib-node @langchain/community
```

https://github.com/jorshali/ai-for-developers/part3

2 Load documents to search with LangChain

- LangChain provides a variety of ways to load your documents:
  - `DirectoryLoader`: loads documents in a directory and additional loaders are specified to handle file types
  - `TextLoader`: loads a single text document from a disk path

```
import { DirectoryLoader } from
  "langchain/document_loaders/fs/directory";
import { TextLoader } from
  "langchain/document_loaders/fs/text";

const loader = new DirectoryLoader(
  "companies",
  {
    ".txt": (path) => new TextLoader(path)
  }
);

const docs = await loader.load();
```

3 Create a Vector Store with `HNSWLib` and Open AI

- We simply call the `fromDocuments` method on `HNSWLib` with our loaded documents
- Notice we also pass `OpenAIEmbeddings`
- `OpenAIEmbeddings` is used to generate the vectors from our documents using Open AI APIs

```
import { HNSWLib } from
  "@langchain/community/vectorstores/hnswlib";
import { OpenAIEmbeddings } from
  "@langchain/openai";

const loader = new DirectoryLoader(
  "companies",
  {
    ".txt": (path) => new TextLoader(path)
  }
);

const docs = await loader.load();

const vectorStore = await HNSWLib.fromDocuments(
  docs, new OpenAIEmbeddings());
```

4 Search the Vector Store

- Now we can ask the `vectorStore` a question in natural language
- The `similaritySearch` is passed our question and a *k* value (*k*-nearest neighbor or said in a simpler way, the number of results we want)
- The `vectorStore` will once again use `OpenAIEmbeddings` to create a vector from our question via the Open AI APIs
- That vector will be compared to the document vectors to find the closest match (our result)

```
const vectorStore = await HNSWLib.fromDocuments(
  docs, new OpenAIEmbeddings());

const resultOne = await vectorStore.similaritySearch(
  "Can you tell me about Acme Corp?", 1);
```

5 Save and load the Vector Store with `HNSWLib`

- Recreating the vectors every time you start your Vector Store would be costly
- The `save` method from `HNSWLib` allows to provide a directory to save your Vector Store on disk
- You can then `load` your Vector Store from that directory

```
const embeddings = new OpenAIEmbeddings();

const vectorStore = await HNSWLib.fromDocuments(
  docs, embeddings);

const dataDirectory = 'data';

await vectorStore.save(dataDirectory);

const loadedVectorStore = await HNSWLib.load(
  dataDirectory, embeddings);
```

6 Create a file that loads the Vector Store

- Now we can create a `loadData.mjs` file that simply loads and saves the Vector Store using `HNSWLib`

```
import { DirectoryLoader } from
  "langchain/document_loaders/fs/directory";
import { TextLoader } from
  "langchain/document_loaders/fs/text";
import { HNSWLib } from
  "@langchain/community/vectorstores/hnswlib";
import { OpenAIEmbeddings } from
  "@langchain/openai";

const loader = new DirectoryLoader(
  "companies",
  {
    ".txt": (path) => new TextLoader(path)
  }
);

const docs = await loader.load();
const embeddings = new OpenAIEmbeddings();

const vectorStore = await HNSWLib.fromDocuments(
  docs, embeddings);

await vectorStore.save('data');
```

7

Create a server file

- We can create a `server.mjs` file that simply loads the Vector Store from disk at startup

```
import { HNSWLib } from
  "@langchain/community/vectorstores/hnswlib";
import { OpenAIEmbeddings } from
  "@langchain/openai";

import express from "express";
import cors from 'cors';

const app = express();

app.use(cors());
app.use(express.json());

const embeddings = new OpenAIEmbeddings();

const loadedVectorStore = await HNSWLib.load(
  'data', embeddings);

app.get('/', async (request, response) => {
  const resultOne = await loadedVectorStore.similaritySearch(
    "Can you tell me about Acme Corp?", 1);

  response.send(resultOne[0].pageContent);
});

app.listen(3000, () => {
  console.log(`Server is running on port 3000`);
});
```

8

Launching your server

- Back in the terminal, make sure you have your API key set:

```
~/ai-for-devs/part3 % export \
  OPENAI_API_KEY=<YOUR_KEY_VALUE>
```

- Run the following terminal command to load the Vector Store:

```
~/ai-for-devs/part3 % node loadData.mjs
```

- Now start the server:

```
~/ai-for-devs/part3 % node server.mjs
```

- Open your web browser and visit: http://localhost:3000
- You'll see the search result in your browser!

Congratulations!

- You've created your own Vector Search!
- What's next?
  - You can find the source code for this tutorial on GitHub:

https://github.com/jorshali/ai-for-developers/part3

- The `README` file provides the instructions to get the example up and running
- **In the next post**, we'll use our Vector Search to implement RAG. **Stay tuned!**

Jacob Orshalick

https://linkedin.com/in/jorshalick

focus.dev

IT PRO search