Embeddings

Select a language

Python JavaScript

Embeddings are the A.I-native way to represent any kind of data, making them the perfect fit for working with all kinds of A.I-powered tools and algorithms. They can represent text, images, and soon audio and video. There are many options for creating embeddings, whether locally using an installed library, or by calling an API.

Chroma provides lightweight wrappers around popular embedding providers, making it easy to use them in your apps. You can set an embedding function when you create a Chroma collection, which will be used automatically, or you can call them directly yourself.

To get Chroma's embedding functions, import the `chromadb.utils.embedding_functions` module.

```
from chromadb.utils import embedding_functions
```

Default: all-MiniLM-L6-v2

By default, Chroma uses the Sentence Transformers `all-MiniLM-L6-v2` model to create embeddings. This embedding model can create sentence and document embeddings that can be used for a wide variety of tasks. This embedding function runs locally on your machine, and may require you download the model files (this will happen automatically).

```
default_ef = embedding_functions.DefaultEmbeddingFunction()
```

TIP

Embedding functions can linked to a collection, which are used whenever you call `add`, `update`, `upsert` or `query`. You can also be use them directly which can be handy for debugging.

```
val = default_ef(["foo"])
```

```
→ [[0.05035809800028801, 0.0626462921500206, -0.061827320605516434...]]
```

Sentence Transformers

Chroma can also use any **Sentence Transformers** model to create embeddings.

```
sentence_transformer_ef =
embedding_functions.SentenceTransformerEmbeddingFunction(model_name="all-MiniLM-
L6-v2")
```

You can pass in an optional `model_name` argument, which lets you choose which Sentence Transformers model to use. By default, Chroma uses `all-MiniLM-L6-v2`. You can see a list of all available models **here**.

OpenAI

Chroma provides a convenient wrapper around OpenAI's embedding API. This embedding function runs remotely on OpenAI's servers, and requires an API key. You can get an API key by signing up for an account at **OpenAI**.

This embedding function relies on the `openai` python package, which you can install with `pip install openai`.

```
openai_ef = embedding_functions.OpenAIEmbeddingFunction(
                api_key="YOUR_API_KEY",
                model_name="text-embedding-ada-002"
            )
```

To use the OpenAI embedding models on other platforms such as Azure, you can use the `api_base` and `api_type` parameters:

```
openai_ef = embedding_functions.OpenAIEmbeddingFunction(
                api_key="YOUR_API_KEY",
                api_base="YOUR_API_BASE_PATH",
                api_type="azure",
                api_version="YOUR_API_VERSION",
                model_name="text-embedding-ada-002"
            )
```

You can pass in an optional `model_name` argument, which lets you choose which OpenAI embeddings model to use. By default, Chroma uses `text-embedding-ada-002`. You can see a list of all available models **here**.

Cohere

Chroma also provides a convenient wrapper around Cohere's embedding API. This embedding function runs remotely on Cohere's servers, and requires an API key. You can get an API key by signing up for an account at **Cohere**.

This embedding function relies on the `cohere` python package, which you can install with `pip install cohere`.

```
cohere_ef  = embedding_functions.CohereEmbeddingFunction(api_key="YOUR_API_KEY",
model_name="large")
cohere_ef(texts=["document1","document2"])
```

You can pass in an optional `model_name` argument, which lets you choose which Cohere embeddings model to use. By default, Chroma uses `large` model. You can see the available models under `Get embeddings` section **here**.

Multilingual model example

```
cohere_ef  = embedding_functions.CohereEmbeddingFunction(
        api_key="YOUR_API_KEY",
        model_name="multilingual-22-12")

multilingual_texts  = [ 'Hello from Cohere!', 'مرحبًا من كوهير!',
        'Hallo von Cohere!', 'Bonjour de Cohere!',
        '¡Hola desde Cohere!', 'Olá do Cohere!',
        'Ciao da Cohere!', '您好，来自 Cohere! ',
        'कोहेरे से नमस्ते!'  ]

cohere_ef(texts=multilingual_texts)
```

For more information on multilingual model you can read **here**.

Instructor models

The **instructor-embeddings** library is another option, especially when running on a machine with a cuda-capable GPU. They are a good local alternative to OpenAI (see the **Massive Text Embedding Benchmark** rankings). The embedding function requires the InstructorEmbedding package. To install it, run `pip install InstructorEmbedding` .

There are three models available. The default is `hkunlp/instructor-base` , and for better performance you can use `hkunlp/instructor-large` or `hkunlp/instructor-xl` . You can also specify whether to use `cpu` (default) or `cuda` . For example:

```
#uses base model and cpu
ef = embedding_functions.InstructorEmbeddingFunction()
```

or

```
ef = embedding_functions.InstructorEmbeddingFunction(
model_name="hkunlp/instructor-xl", device="cuda")
```

Keep in mind that the large and xl models are 1.5GB and 5GB respectively, and are best suited to running on a GPU.

Google PaLM API models

Google PaLM APIs are currently in private preview, but if you are part of this preview, you can use them with Chroma via the `GooglePalmEmbeddingFunction` .

To use the PaLM embedding API, you must have `google.generativeai` Python package installed and have the API key. To use:

```
palm_embedding = embedding_functions.GooglePalmEmbeddingFunction(
    api_key=api_key, model=model_name)
```

HuggingFace

Chroma also provides a convenient wrapper around HuggingFace's embedding API. This embedding function runs remotely on HuggingFace's servers, and requires an API key. You can get an API key by signing up for an account at **HuggingFace**.

This embedding function relies on the `requests` python package, which you can install with `pip install requests` .

```
huggingface_ef = embedding_functions.HuggingFaceEmbeddingFunction(
    api_key="YOUR_API_KEY",
    model_name="sentence-transformers/all-MiniLM-L6-v2"
)
```

You can pass in an optional `model_name` argument, which lets you choose which HuggingFace model to use. By default, Chroma uses `sentence-transformers/all-MiniLM-L6-v2` . You can see a list of all available models **here**.

Custom Embedding Functions

You can create your own embedding function to use with Chroma, it just needs to implement the `EmbeddingFunction` protocol.

```
from chromadb import Documents, EmbeddingFunction, Embeddings

class MyEmbeddingFunction(EmbeddingFunction):
    def __call__(self, texts: Documents) -> Embeddings:
        # embed the documents somehow
        return embeddings
```

We welcome contributions! If you create an embedding function that you think would be useful to others, please consider **submitting a pull request** to add it to Chroma's `embedding_functions` module.

We welcome pull requests to add new Embedding Functions to the community.

Edit this page