Mostrar código

Gemini API: Embeddings Quickstart


Run in Google Colab

The Gemini API generates state-of-the-art text embeddings. An embedding is a list of floating point numbers that represent the meaning of a word, sentence, or paragraph. You can use embeddings in many downstream applications like document search.

This notebook provides quick code examples that show you how to get started generating embeddings.

```
!pip install -q -U google-generativeai
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 146.8/146.8 kB 994.5 kB/s eta 0:00:
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.5/664.5 kB 5.9 MB/s eta 0:00:00
```

```
import google.generativeai as genai
```

Configure your API key

To run the following cell, your API key must be stored it in a Colab Secret named `GOOGLE_API_KEY`. If you don't already have an API key, or you're not sure how to create a Colab Secret, see Authentication for an example.

```
from google.colab import userdata
GOOGLE_API_KEY=userdata.get('GOOGLE_API_KEY')
genai.configure(api_key=GOOGLE_API_KEY)
```

Embed content

Call the `embed_content` method with the `models/text-embedding-004` model to generate text embeddings.

```
text = "Hello world"
result = genai.embed_content(model="models/text-embedding-004", content=text)

# Print just a part of the embedding to keep the output manageable
print(str(result['embedding'])[:50], '... TRIMMED]')
```

```
     [0.013168523, -0.008711934, -0.046782676, 0.000699 ... TRIMMED]
```

```
print(len(result['embedding'])) # The embeddings have 768 dimensions
```

```
     768
```

Batch embed content

You can embed a list of multiple prompts with one API call for efficiency.

```
result = genai.embed_content(
    model="models/text-embedding-004",
    content=[
      'What is the meaning of life?',
      'How much wood would a woodchuck chuck?',
      'How does the brain work?'])

for embedding in result['embedding']:
  print(str(embedding)[:50], '... TRIMMED]')
```

```
     [-0.010632277, 0.019375855, 0.0209652, 0.000770642 ... TRIMMED]
     [0.018467998, 0.0054281196, -0.017658804, 0.013859 ... TRIMMED]
     [0.05808907, 0.020941721, -0.108728774, -0.0403925 ... TRIMMED]
```

Truncating embeddings

The `text-embedding-004` model also supports lower embedding dimensions. Specify `output_dimensionality` to truncate the output.

```
# Not truncated
result1 = genai.embed_content(
    model="models/text-embedding-004",
    content="Hello world")


# Truncated
result2 = genai.embed_content(
    model="models/text-embedding-004",
    content="Hello world",
    output_dimensionality=10)


(len(result1['embedding']), len(result2['embedding']))
```

```
(768, 10)
```

Specify `task_type`

Let's look at all the parameters the `embed_content` method takes. There are five:

- `model`: Required. Must be `models/text-embedding-004` or `models/embedding-001`.
- `content`: Required. The content that you would like to embed.
- `task_type`: Optional. The task type for which the embeddings will be used.
- `title`: Optional. You should only set this parameter if your task type is `retrieval_document` (or `document`).
- `output_dimensionality`: Optional. Reduced dimension for the output embedding. If set, excessive values in the output embedding are truncated from the end. This is supported by `models/text-embedding-004`, but cannot be specified in `models/embedding-001`.

`task_type` is an optional parameter that provides a hint to the API about how you intend to use the embeddings in your application.

The following task_type parameters are accepted:

- `unspecified`: If you do not set the value, it will default to `retrieval_query`.
- `retrieval_query` (or `query`): The given text is a query in a search/retrieval setting.
- `retrieval_document` (or `document`): The given text is a document from a corpus being searched. Optionally, also set the `title` parameter with the title of the document.
- `semantic_similarity` (or `similarity`): The given text will be used for Semantic Textual Similarity (STS).
- `classification`: The given text will be classified.
- `clustering`: The embeddings will be used for clustering.
- `question_answering`: The given text will be used for question answering.
- `fact_verification`: The given text will be used for fact verification.

```
# Notice the API returns different embeddings depending on `task_type`
result1 = genai.embed_content(
    model="models/text-embedding-004",
    content="Hello world")

result2 = genai.embed_content(
    model="models/text-embedding-004",
    content="Hello world",
    task_type="document")

print(str(result1['embedding'])[:50], '... TRIMMED]')
print(str(result2['embedding'])[:50], '... TRIMMED]')
```

```
 [0.013168523, -0.008711934, -0.046782676, 0.000699 ... TRIMMED]
 [0.023399517, -0.00854715, -0.052534223, -0.012143 ... TRIMMED]
```

Learning more

Check out these examples in the Cookbook to learn more about what you can do with embeddings:

- Search Reranking: Use embeddings from the Gemini API to rerank search results from Wikipedia.
- Anomaly detection with embeddings: Use embeddings from the Gemini API to detect potential outliers in your dataset.
- Train a text classifier: Use embeddings from the Gemini API to train a model that can classify different types of newsgroup posts based on the topic.
- Embeddings have many applications in Vector Databases, too. Check out this example with Chroma DB.

You can learn more about embeddings in general on ai.google.dev in the embeddings guide

- You can find additional code examples with the Python SDK here.
- You can also find more details in the API Reference for embedContent and batchEmbedContents.