

## 09 | 语义检索，利用Embedding优化你的搜索功能

2023-04-03 徐文浩 来自北京

《AI大模型之美》




你好，我是徐文浩。

在过去的 8 讲里面，相信你已经对 Embedding 和 Completion 接口非常熟悉了。Embedding 向量适合作为一个中间结果，用于传统的机器学习场景，比如分类、聚类。而 Completion 接口，一方面可以直接拿来作为一个聊天机器人，另一方面，你只要善用提示词，就能完成合理的文案撰写、文本摘要、机器翻译等一系列的工作。

不过，很多同学可能会说，这个和我的日常工作又没有什么关系。的确，日常我们的需求里面，最常使用自然语言处理（NLP）技术的，是搜索、广告、推荐这样的业务。那么，今天我们就来看看，怎么利用 OpenAI 提供的接口来为这些需求提供些帮助。

### 让 AI 生成实验数据


在实际演示代码之前，我们需要一些可以拿来实验的数据。之前，我们都是在网上找一些数据集，或者直接使用一些机器学习软件包里面自带的数据集。但是，并不是所有时候我们都能很快找到合适的数据集。这个时候，我们也可以利用 AI，我们直接让 AI 帮我们生成一些数据不就好了吗？

 复制代码

```
1 import openai, os
2
3 openai.api_key = os.environ.get("OPENAI_API_KEY")
4 COMPLETION_MODEL = "text-davinci-003"
5
6 def generate_data_by_prompt(prompt):
7     response = openai.Completion.create(
8         engine=COMPLETION_MODEL,
9         prompt=prompt,
10        temperature=0.5,
11        max_tokens=2048,
12        top_p=1,
13    )
14    return response.choices[0].text
15
16 prompt = """请你生成50条淘宝网里的商品的标题，每条在30个字左右，品类是3C数码产品，标题里往往也
17 data = generate_data_by_prompt(prompt)
```

为了让数据和真实情况更加接近一点，我们可以好好设计一下我们的提示语。比如，我这里就指明了是淘宝的商品，品类是 3C，并且标题里要包含一些促销信息。

我们把拿到的返回结果，按行分割，加载到一个 DataFrame 里面，看看结果会是怎么样的。

 复制代码

```
1 import pandas as pd
2
3 product_names = data.strip().split('\n')
4 df = pd.DataFrame({'product_name': product_names})
5 df.head()
```

输出结果：

[复制代码](#)

```
1      product_name
2 0  1. 【限时特惠】Apple/苹果 iPhone 11 Pro Max 手机
3 1  2. 【超值折扣】Huawei/华为 P30 Pro 智能手机
4 2  3. 【热销爆款】OPPO Reno 10X Zoom 全网通手机
5 3  4. 【限量特价】Xiaomi/小米 9 Pro 5G 手机
6 4  5. 【限时促销】Apple/苹果 iPad Pro 11寸 平板
```

可以看到，AI 为我们生成了 50 条商品信息，并且每一个都带上了一些促销相关的标签。不过，在返回的结果里面，每一行都带上了一个标号，所以我们需要简单处理一下，去掉这个标号拿到一些干净的数据。

[复制代码](#)

```
1 df.product_name = df.product_name.apply(lambda x: x.split('.')[1].strip())
2 df.head()
```

输出结果：

[复制代码](#)

```
1      product_name
2 0  【限时特惠】Apple/苹果 iPhone 11 Pro Max 手机
3 1  【超值折扣】Huawei/华为 P30 Pro 智能手机
4 2  【热销爆款】OPPO Reno 10X Zoom 全网通手机
5 3  【限量特价】Xiaomi/小米 9 Pro 5G 手机
6 4  【限时促销】Apple/苹果 iPad Pro 11寸 平板
```

我们可以如法炮制，再生成一些女装的商品名称，覆盖不同的品类，这样后面我们演示搜索效果的时候就会方便一点。

[📄 复制代码](#)

```
1 clothes_prompt = """请你生成50条淘宝网里的商品的标题，每条在30个字左右，品类是女性的服饰箱包
2 clothes_data = generate_data_by_prompt(clothes_prompt)
3 clothes_product_names = clothes_data.strip().split('\n')
4 clothes_df = pd.DataFrame({'product_name': clothes_product_names})
5 clothes_df.product_name = clothes_df.product_name.apply(lambda x: x.split('.')[1])
6 clothes_df.head()
```

输出结果：

[📄 复制代码](#)

```
1      product_name
2 0  【新款】时尚百搭羊绒毛衣，暖洋洋温暖你的冬天
3 1  【热卖】复古气质翻领毛衣，穿出时尚感
4 2  【特价】轻薄百搭棉衣，舒适温暖，冬季必备
5 3  【限时】经典百搭牛仔外套，轻松搭配，时尚范
6 4  【全新】简约大气羊绒连衣裙，温柔优雅
```

然后我们把这两个 DataFrame 拼接在一起，就是我们接下来用于做搜索实验的数据。

[📄 复制代码](#)

```
1 df = pd.concat([df, clothes_df], axis=0)
2 df = df.reset_index(drop=True)
3 display(df)
```

输出结果：

[📄 复制代码](#)

```
1      product_name
2 0  【新款】Apple/苹果 iPhone 11 Pro Max 智能手机
3 1  【特惠】华为P30 Pro 8GB+256GB 全网通版
4 2  【限量】OnePlus 7T Pro 8GB+256GB 全网通
5 3  【新品】小米CC9 Pro 8GB+256GB 全网通版
6 4  【热销】三星Galaxy Note10+ 8GB+256GB 全网通
7  ...  ...
8 92  【优惠】气质小清新拼接百搭双肩斜挎包
9 93  【热卖】活力色彩精致小巧百搭女士单肩斜挎包
10 94  【特价】简约可爱原宿风时尚双肩斜挎包
```

- 11 95 【折扣】潮流小清新拼接百搭女士单肩斜挎包
- 12 96 【特惠】百搭潮流活力色彩拼色双肩斜挎

不过如果你多试几次，你会发现 AI 有时候返回的条数没有 50 条，不过没有关系，这个基本不影响我们使用这个数据源。你完全可以多运行几次，获得足够你需要的数据。

## 通过 Embedding 进行语义搜索

那对于搜索问题，我们可以用 GPT 模型干些什么呢？像百度、阿里这样的巨头公司自然有很多内部复杂的策略和模型，但是对于大部分中小公司，特别是刚开始提供搜索功能的时候，往往是使用 Elasticsearch 这个开源项目。而 Elasticsearch 背后的搜索原理，则是先分词，然后再使用倒排索引。

简单来说，就是把上面的“气质小清新拼接百搭双肩斜挎包”这样的商品名称，拆分成“气质”“小清新”“拼接”“百搭”“双肩”“斜挎包”。每个标题都是这样切分。然后，建立一个索引，比如“气质”这个词，出现过的标题的编号，都按编号顺序跟在气质后面。其他的词也类似。

然后，当用户搜索的时候，比如用户搜索“气质背包”，也会拆分成“气质”和“背包”两个词。然后就根据这两个词，找到包含这些词的标题，根据出现的词的数量、权重等等找出一些商品。

但是，这个策略有一个缺点，就是如果我们有同义词，那么这么简单地搜索是搜不到的。比如，我们如果搜“自然淡雅背包”，虽然语义上很接近，但是因为“自然”“淡雅”“背包”这三个词在这个商品标题里都没有出现，所以就没有办法匹配上了。为了提升搜索效果，你就得做更多的工程研发工作，比如找一个同义词表，把标题里出现的同义词也算上等等。

不过，有了 OpenAI 的 Embedding 接口，我们就可以把一段文本的语义表示成一段向量。而向量之间是可以计算距离的，这个我们在之前的情感分析的零样本分类里就演示过了。那如果我们把用户的搜索，也通过 Embedding 接口变成向量。然后把它和所有的商品的标题计算一下余弦距离，找出离我们搜索词最近的几个向量。那最近的几个向量，其实就是语义和这个商品相似的，而并不一定需要相同的关键词。

那根据这个思路，我们不妨用代码来试一试。

首先，我们还是要将随机生成出来的所有商品标题，都计算出来它们的 Embedding，然后存下来。这里的代码，基本和之前通过 Embedding 进行分类和聚类一致，就不再详细讲解了。我们还是利用 backoff 和 batch 处理，让代码能够容错，并且快速处理完这些商品标题。


 复制代码

```
1 from openai.embeddings_utils import get_embeddings
2 import openai, os, backoff
3
4 openai.api_key = os.environ.get("OPENAI_API_KEY")
5 embedding_model = "text-embedding-ada-002"
6
7 batch_size = 100
8
9 @backoff.on_exception(backoff.expo, openai.error.RateLimitError)
10 def get_embeddings_with_backoff(prompts, engine):
11     embeddings = []
12     for i in range(0, len(prompts), batch_size):
13         batch = prompts[i:i+batch_size]
14         embeddings += get_embeddings(list_of_text=batch, engine=engine)
15     return embeddings
16
17 prompts = df.product_name.tolist()
18 prompt_batches = [prompts[i:i+batch_size] for i in range(0, len(prompts), batch_s
19
20 embeddings = []
21 for batch in prompt_batches:
22     batch_embeddings = get_embeddings_with_backoff(prompts=batch, engine=embeddin
23     embeddings += batch_embeddings
24
25 df["embedding"] = embeddings
26 df.to_parquet("data/taobao_product_title.parquet", index=False)
```

然后，我们就可以定义一个 search\_product 的搜索函数，接受三个参数，一个 df 代表用于搜索的数据源，一个 query 代表用于搜索的搜索词，然后一个 n 代表搜索返回多少条记录。而这个函数就干了这样三件事情。

1. 调用 OpenAI 的 API 将搜索词也转换成 Embedding。
2. 将这个 Embedding 和 DataFrame 里的每一个 Embedding 都计算一下余弦距离。


### 3. 根据余弦相似度去排序，返回距离最近的 n 个标题。

 复制代码

```
1 from openai.embeddings_utils import get_embedding, cosine_similarity
2
3 # search through the reviews for a specific product
4 def search_product(df, query, n=3, pprint=True):
5     product_embedding = get_embedding(
6         query,
7         engine=embedding_model
8     )
9     df["similarity"] = df.embedding.apply(lambda x: cosine_similarity(x, product_
10
11     results = (
12         df.sort_values("similarity", ascending=False)
13         .head(n)
14         .product_name
15     )
16     if pprint:
17         for r in results:
18             print(r)
19     return results
20
21 results = search_product(df, "自然淡雅背包", n=3)
```

我们就拿刚才举的那个例子，使用“自然淡雅背包”作为搜索词，调用这个 search\_product 函数，然后拿前 3 个返回结果。可以看到，尽管在关键词上完全不同，但是返回的结果里，的确包含了“小清新百搭拼色女士单肩斜挎包”这个商品。

输出结果：

 复制代码


```
1 【新品】潮流简约可爱时尚双肩斜挎包
2 【优惠】精致小巧可爱双肩斜挎包
3 【新品】小清新百搭拼色女士单肩斜挎包
```

注意，因为我们的商品标题都是随机生成的，所以你得到的数据集和搜索结果可能都和我不一样，你根据实际情况测试你想要的搜索词即可。

## 利用 Embedding 信息进行商品推荐的冷启动


Embedding 的向量距离，不仅可以用于搜索，也可以用于**商品推荐里的冷启动**。主流的推荐算法，主要是依托于用户“看了又看”这样的行为信息。也就是如果有很多用户看了 OPPO 的手机，又去看了 vivo 的手机，那么在用户看 OPPO 手机的时候，我们就可以向他推荐 vivo 手机。但是，往往一个新的商品，或者新的平台，没有那么多相关的行为数据。这个时候，我们同样可以根据商品名称在语义上的相似度，来进行商品推荐。

我们这里的代码实现和上面的搜索例子基本一致，唯一的差别就是商品名称的 Embedding 是根据输入的商品名称，从 DataFrame 里找到的，而不是通过调用 OpenAI 的 Embedding API 获取。因为这个 Embedding 我们之前已经计算过一遍了，没有必要浪费成本再请求一次。

 复制代码

```
1 def recommend_product(df, product_name, n=3, pprint=True):
2     product_embedding = df[df['product_name'] == product_name].iloc[0].embedding
3     df["similarity"] = df.embedding.apply(lambda x: cosine_similarity(x, product_
4
5     results = (
6         df.sort_values("similarity", ascending=False)
7         .head(n)
8         .product_name
9     )
10    if pprint:
11        for r in results:
12            print(r)
13    return results
14
15 results = recommend_product(df, "【限量】OnePlus 7T Pro 8GB+256GB 全网通", n=3)
```

输出结果：

 复制代码

```
1 【限量】OnePlus 7T Pro 8GB+256GB 全网通
2 【新品】OnePlus 7T Pro 8GB+256GB 全网通
3 【限量】荣耀V30 Pro 8GB+256GB 全网通版
```



## 通过 FAISS 加速搜索过程


不过，上面的示例代码里面，还有一个问题。那就是每次我们进行搜索或者推荐的时候，我们都要把输入的 Embedding 和我们要检索的数据的所有 Embedding 都计算一次余弦相似度。例子里，我们检索的数据只有 100 条，但是在实际的应用中，即使不是百度、谷歌那样的搜索引擎，搜索对应的内容条数在几百万上千万的情况也不在少数。如果每次搜索都要计算几百万次余弦距离，那速度肯定慢得不行。

这个问题也有解决办法。我们可以利用一些向量数据库，或者能够快速搜索相似性的软件包就好了。比如，我比较推荐你使用 Facebook 开源的 Faiss 这个 Python 包，它的全称就是 Facebook AI Similarity Search，也就是快速进行高维向量的相似性搜索。

我们把上面的代码改写一下，先把 DataFrame 里面计算好的 Embedding 向量都加载到 Faiss 的索引里，然后让 Faiss 帮我们快速找到最相似的向量，来看看效果怎么样。

当然，按照惯例我们还是需要先安装 Faiss 这个 Python 库。

```
1 conda install -c conda-forge faiss
```

 复制代码

把索引加载到 Faiss 里面非常容易，我们直接把整个的 Embedding 变成一个二维矩阵，整个加载到 Faiss 里面就好了。在加载之前，我们先要定义好 Faiss 索引的维度数，也就是我们 Embedding 向量的维度数。

```

1 import faiss
2 import numpy as np
3
4 def load_embeddings_to_faiss(df):
5     embeddings = np.array(df['embedding'].tolist()).astype('float32')
6     index = faiss.IndexFlatL2(embeddings.shape[1])
7     index.add(embeddings)
8     return index
9
10 index = load_embeddings_to_faiss(df)

```

而搜索 Faiss 也非常容易，我们把查询变成 Embedding，然后再把 Embedding 转换成一个 numpy 的 array 向量，然后直接对刚才生成的索引 index 调用 search 方法，并且指定返回的结果数量就可以了。

返回拿到的，只有索引的 index，也就是加载在 Faiss 里面的第几个索引。我们还是要根据这个，在 DataFrame 里面，反查到对应的是 DataFrame 里面的第几行，以及这一行商品的标题是什么，就能获得搜索的结果。

```

1 def search_index(index, df, query, k=5):
2     query_vector = np.array(get_embedding(query, engine=embedding_model)).reshape
3     distances, indexes = index.search(query_vector, k)
4
5     results = []
6     for i in range(len(indexes)):
7         product_names = df.iloc[indexes[i]]['product_name'].values.tolist()
8         results.append((distances[i], product_names))
9     return results
10
11 products = search_index(index, df, "自然淡雅背包", k=3)
12
13 for distances, product_names in products:
14     for i in range(len(distances)):
15         print(product_names[i], distances[i])

```

输出结果：

- 1 【新品】潮流简约可爱时尚双肩斜挎包 [0.22473168](#)
- 2 【优惠】精致小巧可爱双肩斜挎包 [0.22881898](#)
- 3 【新品】小清新百搭拼色女士单肩斜挎包 [0.22901852](#)

我们同样用“自然淡雅背包”这个搜索词，可以看到搜索结果和之前我们自己计算余弦距离排序的结果是一样的。

Faiss 的原理，是通过 ANN 这样的近似最近邻的算法，快速实现相似性的搜索。如果你想进一步了解 Faiss 的原理，你也可以去问问 ChatGPT。



是否可以简单介绍一下 faiss 这个库的原理



Faiss是Facebook AI Research开发的一个快速相似性搜索库，其全称为“Facebook AI Similarity Search”。该库主要用于高维向量的相似性搜索，可以处理百万甚至亿级别的数据集。



Faiss的原理是基于近似最近邻算法（ANN）来实现高效的相似性搜索。ANN是一种在大数据集中快速查找与给定查询向量最相似的向量的算法。Faiss支持多种ANN算法，包括k-means聚类、倒排索引、局部敏感哈希（LSH）等。

其中，局部敏感哈希是Faiss的核心算法之一。LSH是一种将高维数据映射到低维空间的方法，通过在低维空间中比较相似性来近似地找到高维空间中的最近邻。Faiss中实现了多种LSH方法，如MinHash LSH和Multi-Probe LSH，这些方法可以用于在不同场景下进行高效的相似性搜索。

Faiss还实现了多种向量压缩算法，可以将高维向量压缩为低维向量以提高搜索效率和降低存储开销。这些算法包括PCA、Product Quantization（PQ）、哈希方法等。

总的来说，Faiss利用了多种高效的ANN算法和向量压缩算法来实现高维向量的快速相似性搜索，因此在很多实际应用中都得到了广泛的应用。

Faiss 这个库能够加载的数据量，受限于我们的内存大小。如果你的数据量进一步增长，就需要选用一些向量数据库来进行搜索。比如 OpenAI 就推荐了 [🔗 Pinecone](#) 和 [🔗 Weaviate](#)，周围也有不少团队使用的是 [🔗 Milvus](#) 这个国人开源的产品。

当然，无论是搜索还是推荐，使用 Embedding 的相似度都只是一种快速启动的方式。需要真正做到更好的效果，一定也需要投入更复杂的策略。比如根据用户行为的反馈，更好地排序搜索和推荐结果。但是，对于提供一个简单的搜索或者推荐功能来说，通过文本的 Embedding 的相似度，是一个很好的快速启动的方式。

## 小结

好了，相信经过这一讲，你已经有了快速优化你们现有业务里的推荐和搜索功能的思路了。这一讲里，我主要想教会你三件事情。

第一，是在没有合适的测试数据的时候，我们完全可以让 AI 给我们生成一些数据。既节约了在网上找数据的时间，也能根据自己的要求生成特定特征的数据。比如，我们就可以要求在商品标题里面有些促销相关的信息。

第二，是如何利用 Embedding 之间的余弦相似度作为语义上的相似度，优化搜索。通过 Embedding 的相似度，我们不要求搜索词和查询的内容之间有完全相同的关键字，而只要它们的语义信息接近就好了。

第三，是如何通过 Faiss 这样的 Python 库，或者其他的向量数据库来快速进行向量之间的检索。而不是必须每一次搜索都和整个数据库计算一遍余弦相似度。

通过对于我们自己的数据计算 Embedding，然后索引起来，我们可以将外部的知识和信息引入到使用 GPT 模型的应用里来。后面，我们还会进一步学习如何利用这些外部知识，开发更加复杂的 AI 应用。

## 课后练习

搜索里面经常会遇到这样一个问题，同样的关键词有歧义。比如，我们搜索“小米手机”，返回结果里也许应该有“荣耀 V30 Pro”，但是不应该返回“黑龙江优质小米”。你可以试一试

用 Embedding 进行语义搜索，看看还会不会遇上这样的问题？

期待能在评论区看到你的分享，也欢迎你把这节课分享给感兴趣的朋友，我们下一讲再见。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

## 精选留言 (20)



**廉烨**

2023-04-11 来自上海

老师，请问是否有开源的embedding组件，能够达到或接近openai embedding能力的？能够用于中文问答搜索

作者回复：我们之前用过开源的t5-base，embedding也还不错

你可以选择 flan-t5 系列，或者后面介绍开源平替的 sentence\_transformers

只是embedding的话，可以选的还是不少的



👍 5



**金口**

2023-04-03 来自广东

这门课程主要讲nlp吗？

作者回复：主要是如何运用现在的AI工具做应用开发。自然语言是其中的一部分，核心也不是自然语言处理知识，而是怎么利用好大语言模型直接开发应用。



👍 4



**eagle**

2023-04-05 来自江苏

过几天openAI的模型版本升级了，这些保存的embedding会失效吗？

作者回复：类似 text-ada-embedding 之类的小模型不会改变

升级往往是提供一个新模型

特定模型也有带日期的快照版本，选取那些快照版本就好了。



👍 2



**Joe Black**

2023-05-11 来自北京

embedding的实现是否也需要基础模型的知识沉淀呢？比如文字上虽然不相同，但是含义相近的句子生成的向量是相似的，这个是依靠模型之前学习的知识是吗？那这样自然就是越大的模型embedding的效果越好？可以这样理解吗？

作者回复: embedding就是基础模型啊，只是不是拿来生成文本，只是用了最后一层Transformer里的输出向量。

所以的确是越大的模型，embedding按道理应该越好。



👍 1



**aoe**

2023-04-18 来自浙江

原来商品搜索、推荐系统都已经使用上了 AI，确实强大



👍 1



**Yezhiwei**

2023-04-12 来自北京

请问老师可以把关系型数据库的结构化数据embedding 到向量数据库吗？比如财务报表的数据，然后通过自然语言的方式查询数据，谢谢

作者回复: 可以，但是我会建议反过来操作。把数据库的表信息和需求提供给大语言模型的上下文，然后让大语言模型自动生成查询的SQL，也许效果更好一些。

共 2 条评论 >

👍 1



**渔樵耕读**

2023-04-11 来自北京

请问有遇到安装faiss时提示: PackagesNotFoundError的没？改为pip install faiss 报错：  
ERROR: Could not find a version that satisfies the requirement faiss (from versions: none)  
ERROR: No matching distribution found for faiss

作者回复: `pip install faiss-cpu` 或者 `pip install faiss-gpu`



👍 1



**xbc**

2023-04-04 来自海南

`cosine_similarity` 也可以传入多个embeddings.

```
scores = cosine_similarity(list[list[float]], list[float])
indices = np.argsort(scores)[::-1]
```

作者回复: 👍



👍 1



**Oli张帆**

2023-04-03 来自北京

考虑到我的服务器硬件资源有限（300MB的软限制，500MB的硬限制，现在已经占到了450MB），而且我已经在进行其他任务时使用了很多资源，当我在OpenAI之上构建AI机器人时，我考虑使用这个策略。请老师看看是否能行得通。

每当用户发送请求时，我首先检测他的意图。我可以使用小的embeddings来帮助意图检测。甚至可以有其他方式，如缓存和预先一些意图让用户来点击，来加速意图检测。

在我检测到用户意图后，我可以调用不同的embeddings库。例如，我的客服embeddings库将仅有50个项目，这对于余弦相似性来说非常快速和高效。我还可以将新闻embeddings库限制为最新的100篇文章，以便它可以轻松地通过余弦相似性处理。

尽管这种方法可能不像搜索单个大型嵌入数据库那样准确，但如果我为用户提供足够的指导，您认为它能产生足够好的用户体验吗？例如，在我的界面中，我可以向用户显示他们当前正在讨论“最新的AI发展”或“客户服务”。然后，也可以允许用户快速地改变当前的话题。

请老师看看，这个办法能不能跑通。还没有什么我没有想到的地方？

作者回复: 这个办法当然不是不可以。但是为什么要限制自己用那么小的内存呢？毕竟现在服务器的成本并不高啊。

不用faiss，直接搞个云托管的向量数据库好了？大部分都有免费档位都比你自己的100条资源要多啊？

共 3 条评论 >

👍 1



**stg609**

2023-05-28 来自浙江

老师，请问下，利用 Embedding 进行语义搜索是不是仅限于上下文与提问的关键字存在相关性，比如文中的例子。如果问题是一个需要进行聚合计算之类的问题，比如提问："一共有多少个气质背包？"。这种问题还能利用向量相似性来给出答案吗？



**农民园丁**

2023-05-26 来自内蒙古

运行## 通过Embedding进行语义搜索代码出现如下错误：

InvalidRequestError: Too many inputs. The max number of inputs is 1. We hope to increase the number of inputs per request soon. Please contact us through an Azure support request at: <https://go.microsoft.com/fwlink/?linkid=2213926> for further questions

-----

用的是azure openapi，请问可能是什么原因？



**农民园丁**

2023-05-26 来自立陶宛

运行## 通过Embedding进行语义搜索代码出现如下错误：

InvalidRequestError: Too many inputs. The max number of inputs is 1. We hope to increase the number of inputs per request soon. Please contact us through an Azure support request at: <https://go.microsoft.com/fwlink/?linkid=2213926> for further questions

-----

用的是azure openapi，请问可能是什么原因？



**黄智荣**

2023-05-07 来自福建



这个挺有意思的。不过这种性能应该会降低很多，就算有这种通过faiss计算。原来通过倒排索引，用很低的资源就可以实现，用这faiss 数据量一大，估计都要用显卡才可以，量大对显存要求也很高

作者回复: faiss其实还是很快的，百万量级的faiss similarity通过CPU计算也只需要10ms以内



**Devon**

2023-05-04 来自上海

老师，对于几十万个精准的中文公司名称或者单位地址进行向量化之后，预存进DataFrame作为之后的搜索源，使用时根据较为“脏”的公司名称或者单位地址进行匹配，在这样的操作中对于中文公司名称或者单位地址的向量化预处理，有什么推荐的方式或者模型吗？用OpenAI embedding的话，费用和时间会不会不太经济？

作者回复: 我觉得这个没有必要用Embedding吧，因为公司名称里面的“语义”信息很少，模糊搜索更多是基于字或者音的

按照单个字，或者拼音，做个Trie Tree可能更合适？或者算一下按照字或者字的拼音的字符编辑距离来做搜索排序就好了。



**Allan**

2023-04-25 来自中国香港

授人以鱼不如授人以渔



**Jelly**

2023-04-23 来自广东

当使用Llama Index导入一篇产品介绍的时候，问：本产品的特征是什么，向量匹配不准确。使用XXX的特征就没问题。当导入年报的时候问：2022年营收是多少？向量匹配也不准确，直接问：营收是多少就可以。请问怎么让用户问的问题更智能？

作者回复: 有几种做法

1. 不只是用embedding来做文本召回，llama还提供了更多的数据结构，可以深入看一遍文档，试一

下其他的indices

2. 对于文本，通过 self-ask，先让OpenAI生成很多针对内容的问题。然后和问题做匹配。



1



**Geek\_4ec46c**

2023-04-09 来自福建

老师 在最后一个例子中

```
def search_index(index, df, query, k=5):
```

这里的 query 不用先进行向量化吗? 那么这个时候不是要调用到Openai?

作者回复: 多久一个例子是推荐已经算过embedding的商品，所代码是先根据商品的index在faiss里找到它的向量呀

这个向量是前面调用OpenAI已经计算过了的

共 2 条评论 >



**Geek\_4ec46c**

2023-04-09 来自福建

老师,这样的搜索效果的效率如何? 如果我是一个话术类搜索的,1万多个标题直接用内容来进行搜索的话可以?

作者回复: 用faiss或者上milvus我试过百万级商品毫无压力

共 3 条评论 >



**幼儿编程教学**

2023-04-07 来自浙江

请教老师， get\_embedding函数会请求openai。那么cosine\_similarity函数是否会请求openai接口?

作者回复: 不会，这个就是在你本机上计算两个向量的相似度而已

共 2 条评论 >





**Ethan New**

2023-04-04 来自浙江

打卡学习

