Faiss 모듈

index를 기반으로 데이터를 검색하는 모듈(dense vector)

부성순강사 bakpak@empas.com

벡터검색이란?

벡터 검색은 **머신 러닝(ML)**을 활용하여 텍스트, 이미지 등 비정형 데이터의 의미와 컨텍스트를 캡처한 후 이를 숫자 표현으로 변환합니다. 시맨틱 검색에 자주 사용되는 벡터 검색은 근사 최근접 이웃(ANN) 알고리즘을 사용하여 유사한 데이터를 찾습니다. 기존 키워드 검색과 비교할 때 벡터 검색은 더 정확한 결과를 제공하고 더 빠르게 실행됩니다.

https://www.elastic.co/kr/what-is/vector-search

Elastic 솔루션중 elastic search는 전문검색 내용 전체를 색인해서 특정 단어가 포함된 문서를 검색하는 벡터검색에 특화된 엔진이며 유료임. 기업에서 이미 사용하고 있는 제품이며, AWS, 구글, MS에서도 오픈소스로 벡터검색 엔진을 점점더 제공하고 있는 실정임.

(참고) fassi: facebook research에서 개발한, dense vector들의 클러스터링과 유사도를 구할때 사용하는 라이브러리로 C++로 작성되었으며 python에서 지원된다. 그리고 GPU 상에서도 효율적으로 동작하도록 개발 되었다. Page3에서의 3번 참고(개념은) 실습은 page 4에서 실행

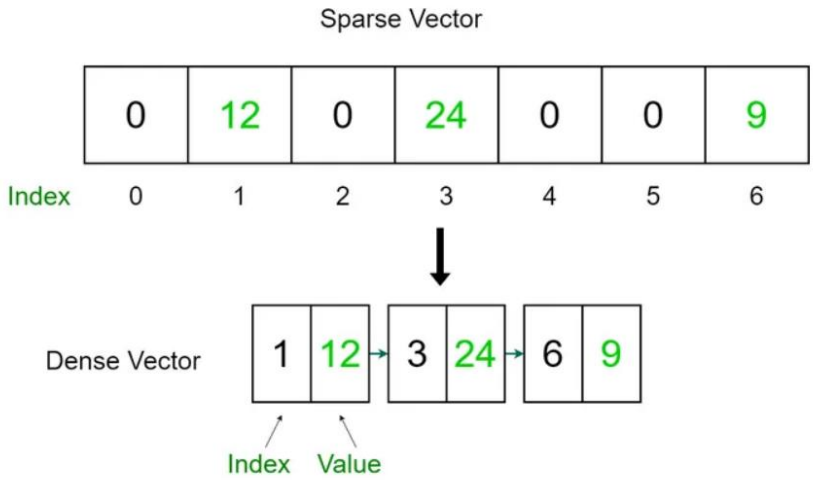
(참고) 엘라스틱 요금제

구글검색

'faiss 구인'

Dense_Vector : index와 value 로 값을 표현하는 vector

아래 사이트 필독



희소벡터(Sparse Vector)와 Dense Vector의 차이점등

https://medium.com/@yasindusanjeewa8/dense-vectors-in-natural-language-processing-06818dff5cd7

1

---

*밀집 벡터를 사용하는 이유*와 현재 사용할 수 있는 밀집 벡터를 구축하는 몇 가지 최선의 접근 방식을 살펴보겠습니다 .

https://www.pinecone.io/learn/series/nlp/dense-vector-embeddings-nlp/

2

---

vector database의 종류 및

https://hotorch.tistory.com/406

3

다양한 벡터 저장소 사용법은 사이트 참조 → https://wikidocs.net/234013

FAISS 소개 및 코드 : 추후 랭체인에서 vector DB에 사용됨

아래사이트 요약한 내용임으로 아래 url만 확인해도 가능함.

https://dajeblog.co.kr/16-faiss%EC%97%90-%EB%8C%80%ED%95%9C-%EB%AA%A8%EB%93%A0-%EA%B2%83/

→ **유사성 검색:** 파시스는 벡터 공간에서 가장 가까운 이웃 벡터를 효과적으로 찾아내는 기능을 제공합니다. 이는 예를 들어 이미지, 텍스트, 사운드 등 다양한 유형의 데이터에 대한 유사성 기반 검색에 사용될 수 있습니다.

→ **벡터 양자화:** 파시스는 벡터 데이터를 압축하고 저장하는 데 사용되는 벡터 양자화 기능도 제공합니다. 이 기능은 대량의 데이터를 효율적으로 저장하고, 메모리 사용량을 줄이는 데 도움이 됩니다.

파시스의 핵심 개념은 인덱스(index) 생성입니다. 인덱스는 벡터 데이터의 구조를 나타내는 메타데이터로 볼 수 있으며, 이를 통해 효율적인 검색이 가능해집니다. 파시스는 먼저 데이터를 양자화하여 인덱스를 생성하고, 이후 이 인덱스를 사용하여 유사성 검색을 수행합니다.
이 과정에서 파시스는 복잡한 벡터 공간을 작은 '클러스터'로 분할하는데, 각 클러스터는 그 안의 벡터들이 서로 유사하다는 점에서 차별성을 가집니다. 클러스터링은 원본 벡터 공간을 더 작고, 이해하기 쉽고, 계산하기 편한 공간으로 변환하는 역할을 합니다. 이렇게 생성된 클러스터는 원래의 대량 데이터 대신 사용될 수 있으며, 각 클러스터는 원본 데이터의 '대표' 또는 '중심'을 나타내게 됩니다.

FAISS를 사용하려면 먼저 필요한 벡터 데이터를 로드하고, 이를 파시스 인덱스에 추가하는 과정을 거쳐야 합니다. 그 다음으로는 인덱스를 학습시키고, 이를 사용하여 유사성 검색을 수행할 수 있습니다.

(1) FAISS 기본 원리 – 직접 norm 값 계산하여 보기

faiss 모듈의 유사도분석 이해

```python
import numpy
import pandas as pd

query_vector = np.array([[3]], dtype=np.float32) # 3을 기준으로 L2 norm으로 가장 가까운 값을 찾고자함
db_vectors = np.array([[1.],
                       [2.],
                       [3.],
                       [4.],
                       [5.]], dtype=np.float32)
df=pd.DataFrame(db_vectors[:,0],columns=['db_vectors'])

df['query']=query_vector[0][0]
df['L2norm']=(df['query']-df['db_vectors'])**2

display(df)

print('▶L2 sort------------')
df.sort_values(by='L2norm')

```

	db_vectors	query	L2norm
0	1.0	3.0	4.0
1	2.0	3.0	1.0
2	3.0	3.0	0.0
3	4.0	3.0	1.0
4	5.0	3.0	4.0

▶L2 sort------------

	db_vectors	query	L2norm
2	3.0	3.0	0.0
1	2.0	3.0	1.0
3	4.0	3.0	1.0
0	1.0	3.0	4.0
4	5.0	3.0	4.0

index번호

거리값

faiss 모듈에서 return하는 값(6page)

```
(array([[0., 1., 1., 4., 4.]], dtype=float32)
 array([[2, 1, 3, 0, 4]], dtype=int64))
```

(1) FAISS 기본 원리 – faiss 모듈을 이용하여 계산 거리 확인하기

faiss 모듈의 유사도분석 이해

```python
import faiss
import numpy

query_vector = np.array([[3]], dtype=np.float32) # 3을 기준으로 L2 norm으로 가장 가까운 값을 찾고자함
db_vectors = np.array([[1.],
                       [2.],
                       [3.],
                       [4.],
                       [5.]], dtype=np.float32)
database_size,dimension = db_vectors.shape[0],db_vectors.shape[1]# 데이터베이스 크기 및 벡터 차원
index.reset()   # Faiss index reset()
index = faiss.IndexFlatL2(dimension)  #  index를 생성, 이때 x변수단위로 생성함. 열의갯수가 1개
index.add(db_vectors)   #index를 부착
print(index.ntotal, database_size,dimension)

num_nearest_neighbors=5  #index 1개의 열에서 5개의 행중 L2 norm 값으로 찾을수 있는 가장 가까운 값은 5개가 최대임
distances, indices = index.search(query_vector, num_nearest_neighbors)  # 검색 수행
distances, indices
```

5개

3을 기준으로

5개 가까운값 찾기

✓ 0.0s

```
5 5 1

(array([[0., 1., 1., 4., 4.]], dtype=float32),
 array([[2, 1, 3, 0, 4]], dtype=int64))
```

1 L2 norm 작은값부터

2 L2 norm 작은값부터 index 위치

	db_vectors	query	L2norm
2	3.0	3.0	0.0
1	2.0	3.0	1.0
3	4.0	3.0	1.0
0	1.0	3.0	4.0
4	5.0	3.0	4.0

2

1

(1) FAISS 기본 원리 – faiss 모듈에서 index 갯수가 자료갯수를 초과할때 faiss 모듈의 유사도분석 이해

```python
import faiss
import numpy

query_vector = np.array([[3]], dtype=np.float32) # 3을 기준으로 L2 norm으로 가장 가까운 값을 찾고자함
db_vectors = np.array([[1.],
                       [2.],
                       [3.],
                       [4.],
                       [5.]],dtype=np.float32)
database_size,dimension = db_vectors.shape[0],db_vectors.shape[1]# 데이터베이스 크기 및 벡터 차원
index.reset()   # Faiss index reset()
index = faiss.IndexFlatL2(dimension)  #  index를 생성, 이때 x변수단위로 생성함. 열의갯수가 1개
index.add(db_vectors)   #index를 부착
print(index.ntotal, database_size,dimension)

num_nearest_neighbors=7 #index 1개의 열에서 5개의 행중 L2 norm 값으로 찾을수 있는 가장 가까운 값은 5개가 최대임
distances, indices = index.search(query_vector, num_nearest_neighbors)  # 검색 수행
distances, indices
```

5개

✓ 0.0s

```
5 5 1
```

5개 이상의 자료를 찾을때는 -1의 위치값이 출력됨

```
(array([[0.0000000e+00, 1.0000000e+00, 1.0000000e+00, 4.0000000e+00,
         4.0000000e+00, 3.4028235e+38, 3.4028235e+38]], dtype=float32),
 array([[ 2,  1,  3,  0,  4, -1, -1]], dtype=int64))
```

(1) FAISS 기본 원리 – x2데이터 pandas와 faiss 모듈로 구하기

faiss 모듈의 유사도분석 이해

Pandads 모듈

```python
####################
## x2데이터 pandas 로
####################
import numpy
import pandas as pd



query_vector = np.array([[2]], dtype=np.float32)
db_vectors = np.array([[2.],
                       [8.],
                       [9.],
                       [3.],
                       [1.]], dtype=np.float32)
df=pd.DataFrame(db_vectors[:,0],columns=['db_vectors'])

df['query']=query_vector[0][0]
df['L2norm']=(df['query']-df['db_vectors'])**2

display(df)

print('▶L2 sort------------')
df.sort_values(by='L2norm')
```

faiss 모듈

```python
####################################
## faiss 모듈로 구하기
####################################
query_vector = np.array([[2]], dtype=np.float32)
db_vectors = np.array([[2.],
                       [8.],
                       [9.],
                       [3.],
                       [1.]], dtype=np.float32)

database_size,dimension = db_vectors.shape[0],db_vectors.shape[1]# 데이터베이스 크기 및 벡터
index.reset()   # Faiss index reset()
index = faiss.IndexFlatL2(dimension)  #  index를 생성, 이때 x변수단위로 생성함. 열의갯수가 1개
index.add(db_vectors)   #index를 부착

print(index.ntotal, database_size,dimension)



################################33
## 가까운 거리 구하기
num_nearest_neighbors=5
distances, indices = index.search(query_vector, num_nearest_neighbors)
distances, indices
```

0.0s

```
5 5 1
```

```
(array([[ 0.,  1.,  1., 36., 49.]], dtype=float32),
 array([[0, 3, 4, 1, 2]], dtype=int64))
```

	db_vectors	query	L2norm
0	2.0	2.0	0.0
1	8.0	2.0	36.0
2	9.0	2.0	49.0
3	3.0	2.0	1.0
4	1.0	2.0	1.0

▶L2 sort------------

	db_vectors	query	L2norm
0	2.0	2.0	0.0
3	3.0	2.0	1.0
4	1.0	2.0	1.0
1	8.0	2.0	36.0
2	9.0	2.0	49.0

(1) FAISS 기본 원리 – 2개이상의 x 데이터를 이용한 norm 값 구하기

faiss 모듈의 유사도분석 이해

```
############################
## pandas 로
## x1,x2값 모두 사용해보기
############################

query_vector = np.array([[3,2]], dtype=np.float32)
db_vectors = np.array([[1,2],
                       [2,8],
                       [3,9],
                       [4,3],
                       [5,1]], dtype=np.float32)


df=pd.DataFrame(db_vectors,columns=['x1','x2'])

df['query_x1']=query_vector[0][0]
df['query_x2']=query_vector[0][1]
df['x1_L2norm']=(df['query_x1']-df['x1'])**2
df['x2_L2norm']=(df['query_x2']-df['x2'])**2

df['L2norm']=df['x1_L2norm']+df['x2_L2norm']

display(df)

print('▶L2 sort------------')
df.sort_values(by='L2norm')
```

	x1	x2	query_x1	query_x2	x1_L2norm	x2_L2norm	L2norm
0	1.0	2.0	3.0	2.0	4.0	0.0	4.0
1	2.0	8.0	3.0	2.0	1.0	36.0	37.0
2	3.0	9.0	3.0	2.0	0.0	49.0	49.0
3	4.0	3.0	3.0	2.0	1.0	1.0	2.0
4	5.0	1.0	3.0	2.0	4.0	1.0	5.0

2 sort------------ 1

	x1	x2	query_x1	query_x2	x1_L2norm	x2_L2norm	L2norm
3	4.0	3.0	3.0	2.0	1.0	1.0	2.0
0	1.0	2.0	3.0	2.0	4.0	0.0	4.0
4	5.0	1.0	3.0	2.0	4.0	1.0	5.0
1	2.0	8.0	3.0	2.0	1.0	36.0	37.0
2	3.0	9.0	3.0	2.0	0.0	49.0	49.0

```
database_size,dimension = db_vectors.shape[0],db_vectors.shape[1]
index.reset()   # Faiss index reset()
index = faiss.IndexFlatL2(dimension)  #  index를 생성, 이때 x변수
index.add(db_vectors)   #index를 부착

print(index.ntotal, database_size,dimension)
num_nearest_neighbors=5
distances, indices = index.search(query_vector, num_nearest_neigh
distances, indices
```

136] ✓ 0.0s

Faiss 모듈

```
5 5 2

1 (array([[ 2.,  4.,  5., 37., 49.]], dtype=float32),
2  array([[3, 0, 4, 1, 2]], dtype=int64))
```

(2) text를 숫자화 하는 - word

text 자료의 vector(숫자)화

https://www.syncly.kr/blog/what-is-embedding-and-how-to-use → **한번 쓱읽기만 함.**

Word (token) embedding vs. Sentence/Document embedding

Embedding을 추출할 원본 텍스트의 형태를 기준으로 embedding을 구분하자면, 크게 Word (token) embedding과 Sentence/Document embedding으로 구분할 수 있습니다.

원본 텍스트는 AI 모델에 입력되기 전에 더 작은 조각들로 쪼개지는 과정을 반드시 먼저 거칩니다. 이 때의 "조각"를 "token", 쪼개는 일을 하는 모델을 "tokenizer"라고 지칭하며, 어떤 tokenizer를 사용하느냐에 따라 하나의 token이 곧 "word"(단어) 하나가 될 수도 있고 "subword"(단어의 일부 조각)가 될 수도 있습니다. 이런 token 하나로부터 추출한 embedding을 흔히 "Word (token) embedding"이라고 부릅니다.

OpenAI에서 제공하는 GPT-3 Tokenizer는, OpenAI에서 제공하는 GPT 계열의 모델에서 사용하는 tokenizer가 어떻게 동작하는지를 이해하기 쉽게 보여 주는 툴입니다. 예를 들어 아래 그림과 같이 해당 툴의 입력란에 원하는 영문 텍스트를 입력하면, 해당 텍스트가 어떠한 (subword) tokens로 쪼개져서 AI 모델에 들어가게 되는지 한 눈에 확인할 수 있습니다.

https://platform.openai.com/tokenizer

한편, 사용자가 입력으로 넣기를 원하는 것은 보통 단어 하나가 아니라 하나의 문장 또는 여러 문장으로 구성된 하나의 문서일 텐데, 이렇게 문장 또는 문서 전체로부터 추출한 embedding을 "Sentence embedding" 또는 "Document embedding"이라고 부릅니다. Sentence/Document embedding의 경우, 이를 구성하는 (sub)word들로부터 계산된 여러 개의 word (token) embedding들을, averaging(평균) 연산 등을 통해 하나의 embedding으로 집계하는 방식으로 얻어집니다.

https://platform.openai.com/docs/guides/embeddings/use-cases → 개념만, 모듈에러임

OpenAI 측에서 제공한 **가이드 문서**에 의거하면, OpenAI Embeddings에서는 embedding 추출을 위해 **GPT-3** 계열의 LLM을 사용합니다. OpenAI Embeddings로부터 추출한 embedding vector는 (text-embedding-ada-002 모델 사용 시) 1,536차원이기 때문에, 길이가 긴 텍스트에 담긴 의미적 정보 또한 충실하게 담기에 적절한 구성을 가지고 있다고 할 수 있습니다.

현재는 **Assistants API에서 사용해야함**
아래와 같이 사용해야함.

```
from langchain_openai import OpenAIEmbeddings
```

(2) text를 숫자화 하는 - word

text 자료의 vector(숫자)화

```python
from langchain_openai import OpenAIEmbeddings
embeddings_model = OpenAIEmbeddings(
    model= "text-embedding-ada-002" ,
    openai_api_key="본인키")


embeddings = embeddings_model.embed_documents(
    [
        "a",
        "b",
        "a",
        "a",
        "b",
        'c r'
    ], chunk_size=7
)

embeddings=np.array(embeddings)
len(embeddings), np.shape(embeddings), embeddings[:,:5]
```

6개

✓ 2.6s

```
(6,
 (6, 1536),
 array([[ 0.00477949, -0.01501446,  0.00756838, -0.0127521 , -0.02970282],
        [-0.01128527, -0.02000359,  0.00639077, -0.00099669, -0.02875515],
        [ 0.00474026, -0.01500573,  0.0076116 , -0.01274943, -0.02968524],
        [ 0.00474026, -0.01500573,  0.0076116 , -0.01274943, -0.02968524],
        [-0.01128527, -0.02000359,  0.00639077, -0.00099669, -0.02875515],
        [ 0.01481909, -0.00950524, -0.01673718, -0.03296287, -0.02772007]]))
```

1 2 3 4 5

6개

(3) 실전 (문서 유사도)

*** 살펴보기만함**

https://velog.io/@gmlwlswldbs/NLP2-%EC%9B%8C%EB%93%9C-%EC%9E%84%EB%B2%A0%EB%94%A9-1

*** 질의응답 챗봇 제작**
도식화 및 전체적인 proces보기: 텍스트자료 –> 숫자화(word embedding) ->랭체인 챗봇까지 (Lessons01~06까지 다살펴봄)

https://pythonmldaily.com/lesson/python-chatbot-langchain/intro-tools-plan-of-action

*** 이미지도 faiss로 작업가능함.**

https://www.gpters.org/c/llm/8-vectordb-faiss-faiss-vectordb

진짜 결론

VectorDB 중 Faiss는 vector 연산에 특화된 DB 입니다.

Langchain을 관통하는 기본 데이터는 text입니다. 그래서 Faiss는 text 데이터 저장 및 서칭에 손쉽게 사용될 수 있습니다. 반면, 그렇기 때문에 이미지와 같은 데이터를 저장하거나 서칭하는 과정은 내장 코드만으로는 힘듭니다.

하지만 서비스 종류에 따라 이종간 데이터 저장 및 서칭이 필요할 수 있겠죠. 그를 위해 Custom Faiss VectorDB를 구현해봤고 필수 요소(index와 docstore)의 역할을 공부해봤습니다.

제 삽질이 비슷한 문제를 고민하는 분들에게 혹은 Langchain을 공부하시는 분들에게 도움이 되었길 희망합니다 ^^

*** (필수) 실습함 (Faiss 객체 저장 및 사용법)**

https://wikidocs.net/234014