임베딩 벡터와 유사도 측정

1. 벡터(Vector)

벡터의 개념

- **벡터**: 수학에서 벡터는 여러 숫자들이 모인 리스트.
  - 예를 들어, 2차원 벡터는 평면에서 한 점을 나타낼 수 있음. `(x, y)` 형태로 표현.
  - 벡터는 여러 차원으로 확장될 수 있음. 3차원에서는 `(x, y, z)`로 표현.
  - n차원에서는 더 많은 숫자로 데이터를 표현할 수 있음.
- **벡터의 예시**:
  - 2차원 벡터: `[2, 3]`
  - 3차원 벡터: `[1, -2, 3]`
  - 10차원 벡터: `[1, 0, -3, -1, 5, 10, 2, 8, 6, 1]`

```
import numpy as np

# 2차원 벡터
vector_1 = np.array([2, 3])
vector_2 = np.array([1, 4])

print("벡터 1:", vector_1)
print("벡터 2:", vector_2)
```

단위 벡터

- **크기**: 벡터의 길이를 말하며, 벡터가 얼마나 큰지를 나타냄.
  - 벡터의 크기는 피타고라스의 정리를 사용해 구할 수 있음.
  - 예: `[3, 4]`벡터의 크기는 `√(3² + 4²) = 5`
- **방향**: 벡터가 어느 방향으로 가는지를 나타냄.
- **단위 벡터**: 길이가 1인 벡터의 방향을 나타내기 위해 사용.
  - 벡터를 단위 벡터로 만들기 위해서는 해당 벡터를 그 크기로 나누면 됨.
  - $\hat{v} = \frac{v}{|v|}$
  - 예: `[3, 4]`벡터를 크기 `5`로 나누면 단위 벡터는 `[3/5, 4/5] = [0.6, 0.8]`

```
import numpy as np

# 벡터 정의
vector = np.array([3, 4])

# 벡터 크기 구하기 (노름)
magnitude = np.linalg.norm(vector)
```

```
print(f"벡터의 크기: {magnitude}")

# 단위 벡터 구하기
unit_vec = vector / magnitude
print(f"단위 벡터: {unit_vec}")
```

백터의 연산

1. **더하기**: 두 벡터의 각 요소를 더함.
    - 예: `[2, 3] + [1, 4] = [3, 7]`
2. **빼기**: 두 벡터의 각 요소를 뺌.
    - 예: `[2, 3] - [1, 4] = [1, -1]`
3. **스칼라 곱하기**: 벡터에 스칼라 값을 곱함.
    - 예: `2 * [2, 3] = [4, 6]`
4. **스칼라 나누기**: 벡터에 스칼라 값을 나눔.
    - 예: `[2, 4] / 2 = [1, 2]`
5. **내적(Dot production)**: 두 벡터의 대응하는 요소들의 곱을 더한 값.
    - 내적은 두 벡터의 크기와 두 벡터 사이 각도를 코사인(cos) 함수에 넣은 값을 모두 곱한 값과 같음.
    - 예: `[2, 3] · [1, 4] = (2*1) + (3*4) = 2 + 12 = 14`

- 벡터의 연산은 서로 같은 차원의 벡터 사이에서만 가능.

```
# 벡터 연산 예시
import numpy as np

vector_1 = np.array([5, 7])
vector_2 = np.array([3, 2])

# 벡터 더하기
add_result = vector_1 + vector_2
print("벡터 더하기:", add_result)

# 벡터 빼기
subtract_result = vector_1 - vector_2
print("벡터 빼기:", subtract_result)

# 스칼라 곱하기
scalar_multiplication = 2 * vector_1
print("스칼라 곱하기:", scalar_multiplication)

# 스칼라 나누기
scalar_division = vector_1 / 2
print("스칼라 나누기:", scalar_division)

# 내적(점곱)
dot_product = np.dot(vector_1, vector_2)
print("내적:", dot_product)
```

2. 임베딩 벡터 (Embedding Vector)

- **임베딩**: 데이터(텍스트, 이미지 등)를 벡터로 변환하는 과정.
    - 예: 텍스트 데이터를 숫자로 변환한 후 벡터로 표현한 것.

BERT

- Bidirectional Encoder Representations from Transformers
- 자연어 처리에서 자주 사용되는 임베딩 모델로, 텍스트 데이터를 임베딩 벡터로 변환.
- 텍스트를 임베딩 벡터로 변환한 후에 다양한 어플리케이션에 활용 가능.
    - 텍스트 분류: 뉴스 기사, 리뷰 등 텍스트 데이터를 벡터로 변환하여 긍정/부정 감성 분석.
    - 문서 간 유사도 측정: 두 문장을 임베딩한 후, 벡터 간의 거리를 계산하여 문장 간의 유사성을 비교.

```
from transformers import BertTokenizer, BertModel
import torch

# BERT 토크나이저와 모델 불러오기
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# 문장 정의
sentence = "I love reading books."

# 문장을 토큰으로 변환 및 BERT 입력 형식으로 변환
inputs = tokenizer(sentence, return_tensors='pt')

# BERT 모델을 통해 임베딩 벡터 얻기
with torch.no_grad():
    outputs = model(**inputs)

# 임베딩 벡터 추출
embedding_vector = outputs.last_hidden_state.mean(dim=1)
print("임베딩 벡터:", embedding_vector)
```

3. 벡터 간 거리와 유사도

- 벡터 사이의 거리를 계산하여 데이터 간의 차이를 측정.
- 벡터 간 거리를 통해서 유사도를 측정할 수 있음.

유클리드 거리(Euclidean Distance): 벡터 사이의 직선 거리.

- 피타고라스의 정리를 통해서 계산할 수 있음.
- $d(x, y) = \sqrt{\sum(x_i - y_i)^2}$

```
# 유클리드 거리 계산
def euclidean_distance(vec1, vec2):
    return np.linalg.norm(vec1 - vec2)
```

```
distance = euclidean_distance(vector_1, vector_2)
print(f"유클리드 거리: {distance}")
```

코사인 유사도(Cosine Similarity): 벡터 간의 각도로 유사도를 측정.

- 벡터의 크기(길이)는 고려하지 않고, **벡터의 방향**만을 이용해 유사도를 계산합니다.
- 두 벡터의 방향이 얼마나 비슷한지를 확인.
- **값의 범위**: 코사인 유사도는 -1에서 1 사이의 값을 가짐.
  - **1**: 두 벡터가 완전히 같은 방향을 가리킬 때.
  - **0**: 두 벡터가 직교(90도, 서로 관련 없음)할 때.
  - **-1**: 두 벡터가 완전히 반대 방향을 가리킬 때.
- 즉, 코사인 유사도 값이 1에 가까울수록 두 벡터가 비슷한 방향을 가진다는 것을 의미하며, -1에 가까울수록 반대 방향을 가리킨다는 것을 의미.

코사인 유사도 계산 공식

- 코사인 유사도는 두 벡터의 내적(dot product)을 각 벡터의 크기(norm)로 나눈 값.
- $$\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}||\mathbf{B}|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$
  - `x · y`: 두 벡터의 내적
  - `|x|`: 첫 번째 벡터의 크기
  - `|y|`: 두 번째 벡터의 크기

```
# 코사인 유사도 계산
def cosine_similarity(vec1, vec2):
    dot_product = np.dot(vec1, vec2)
    norm_vec1 = np.linalg.norm(vec1)
    norm_vec2 = np.linalg.norm(vec2)
    return dot_product / (norm_vec1 * norm_vec2)

similarity = cosine_similarity(vector_1, vector_2)
print(f"코사인 유사도: {similarity}")
```

퀴즈 1: 벡터 연산 구현하기

- 아래 코드를 완성하여 벡터 간 코사인 유사도를 계산하시오.
- 직접 계산해보고, 프로그램을 돌린 결과와 일치하는지 확인하시오.

```
import numpy as np

# 벡터 정의
vector_1 = np.array([1, 2, 3, 5])
vector_2 = np.array([4, 5, 6, 8])

# 코사인 유사도 함수
```

```
def cosine_similarity(vec1, vec2):
    # TODO: 여기에 코드를 작성하세요
    pass

# 함수 실행
similarity = cosine_similarity(vector_1, vector_2)

print(f"코사인 유사도: {similarity}")
```

퀴즈 2: 임베딩 벡터 만들기

- 아래 코드를 완성하여 `categories`에 저장된 단어들을 각각 임베딩 벡터로 변환하여 리스트로 저장하시오.

```
from transformers import BertTokenizer, BertModel
import torch

# 임베딩할 단어 리스트
categories = ['fruit', 'vehicle', 'animal', 'tool', 'electronic device',
'sport', 'profession', 'emotion', 'planet', 'musical instrument']

# BERT 모델 및 토크나이저 불러오기
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# 단어 임베딩 벡터를 저장할 리스트
embedding_vectors = []

# 각 단어를 임베딩 벡터로 변환하고 embedding_vectors에 저장
for word in categories:
    # TODO: 여기에 코드를 작성하세요

# 각 단어의 임베딩 벡터 출력
for word, vec in zip(categories, embedding_vectors):
    print(f"{word} 임베딩 벡터: {vec}")
```

퀴즈 3: 텍스트 분류 프로그램 만들기

- 코사인 유사도를 통해서 사용자가 입력한 텍스트의 카테고리를 분류하는 프로그램을 작성하시오.

```
import numpy as np
from transformers import BertTokenizer, BertModel
import torch

# 코사인 유사도 함수 정의
def cosine_similarity(vec1, vec2):
    # TODO: 퀴즈 1 정답 입력
    pass

# 임베딩할 단어 리스트
```

```
categories = ['fruit', 'vehicle', 'animal', 'tool', 'electronic device',
'sport', 'profession', 'emotion', 'planet', 'musical_instrument']

# BERT 모델 및 토크나이저 불러오기
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# 단어 임베딩 벡터를 저장할 리스트
embedding_vectors = []

# 각 단어를 임베딩 벡터로 변환하고 embedding_vectors에 저장
for word in categories:
    # TODO: 퀴즈 2 정답 입력

# 사용자 입력 받기
user_input = input("텍스트를 입력하세요: ")

# 입력된 단어(user_input)를 임베딩 벡터로 변환
# TODO: 여기에 코드를 작성하게요.

# 코사인 유사도를 통해 입력된 단어와 각 카테고리의 유사도 계산
similarities = []
# TODO: 여기에 코드를 작성하세요.

# 유사도가 가장 높은 카테고리 선택
max_index = np.argmax(similarities)
predicted_category = categories[max_index]

# 결과 출력
print(f"입력한 단어 '{user_input}'는 '{predicted_category}' 카테고리와 가장 유사합니
다.")
print('유사도 결과')
for i in range(len(categories)):
    print(f'- {categories[i]}: {similarities[i]}' )
```

퀴즈 4: 오류 분석하기

- `apple`를 입력하였을 때 `fruit`이 나오는 지 확인하시오.
- `Apple is red and delicious`를 입력했을 때 어떤 카테고리가 나오는 지 확인하시오.
- `Powerful CPU is embedded in Apple`를 입력했을 때 어떤 카테고리가 나오는 지 확인하시오.
- 위와 같이 결과가 나오는 이유를 적어보시오.