Python Report

행정학과

233973

김세희

[Introduction]

This report presents the results of a project that leveraged the knowledge acquired through the first seven weeks of the course to develop a simple text similarity search system. The project's objective was to create a basic search engine. The system calculates the similarity between a given query and pre-indexed text data, providing results based on this similarity.

[Requirements]

1. User Requirement:

- The system should output 10 sentences that are similar to the string (query) entered by the user.

Functional Requirements:

1. Data Preprocessing and Storage Requirement:

- Sentences within the target data are preprocessed and stored in a list.

2. User Input and Preprocessing Requirement:

- The system should accept English string input from the user and perform preprocessing.

3. Similarity Calculation Requirement:

- The system must calculate the similarity between the entered English string and the sentences within the search dataset.

4. Ranking Requirement:

- Sentences should be ranked based on their similarity scores.

5. Ranked Sentences Output Requirement:

- The system should present the top 10 ranked sentences to the user.

[Design and Implementation]

1. Preprocessing and Storing Sentences from the Target Data (indexing function):

- The **indexing(file_name)** function reads English sentences from the given file and preprocesses them. It reads each line from the file, splits it into words, and then stores each preprocessed sentence in a list. These preprocessed sentences are then added to the list file_tokens_pairs, which serves as the searchable dataset.

2. Receiving and Preprocessing User Input for an English String (input function):

- The **input()** function is responsible for receiving the user's English search query. This query needs to be preprocessed as well. Preprocessing involves splitting the query into words using spaces and converting all words to lowercase.

3. Calculating Similarity between the Input English String and Sentences in the Search Dataset (calc_similarity function):

- The **calc_similarity(preprocessed_query, preprocessed_sentences)** function calculates the similarity between the user's query and the sentences in the search dataset. This function employs the Jaccard similarity metric to perform the calculations. To compute similarity, it follows these steps:

- Preprocesses both the input query and each sentence by splitting them into words using spaces and converting them to lowercase.

- Calculates the number of common words between the query and each sentence.

- Combines all words from the query and the sentence, then calculates Jaccard similarity by dividing the number of common words by the total number of words.

- Stores the similarity scores for each sentence in a dictionary.

4. Ranking Sentences Based on Similarity and Displaying Results (Result Output Section):

- The section for ranking sentences based on similarity and displaying results is found towards the end of the code.

- **sorted_score_list** is a list of sentence indices and their corresponding similarity scores, sorted in descending order of similarity.

[Testing]

Results by Function: (Screenshots for Each Requirement)

1. Requirement: Data Preprocessing and Storage

```
#파일에서 문장을 읽어온 후 전처리 해서 저장시키는 함수 생성
def indexing(file_name):
    file_tokens_pairs = [] #전처리 된 문장을 저장할 리스트 생성
    lines = open(file_name, "r", encoding="utf8").readlines()  #파일 불러오기
    for line in lines:
        tokens = preprocess(line) #각 줄을 preproecess 함수를 통해 전처리 한후 tokesns 에 저장
        file_tokens_pairs.append(tokens) #  fill_tokens_pairs 리스트에 tokens 값 추가
    return file_tokens_pairs
```

1) 입력

1) Input

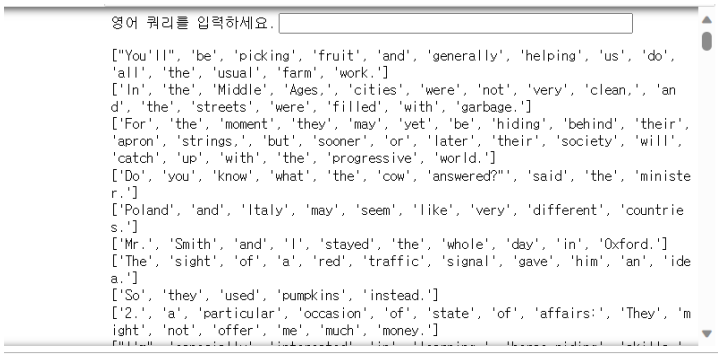
- indexing() : Input the file name.

2) Result

- Return value: None

- Preprocess the data and store it in a list.

```
for line in lines:
    tokens = preprocess(line) #각 줄을 preproecess 함수를 통해 전처리 한후 tokes
    file_tokens_pairs.append(tokens) #  fill_tokens_pairs 리스트에 tokens 값 추
    print(tokens)
return file_tokens_pairs
```



3) Description

- The 'indexing()' function takes the user-provided file name as input. It loads the file based on the user's input and preprocesses it using the 'preprocess' function, storing the results in the 'tokens' variable.

- The preprocessed data is saved in the 'file_tokens_pairs' list

2. Requirement: Query

```
#문장을 전처리 시키는 함수 생성
def preprocess(sentence):
    preprocessed_sentence = sentence.strip().split(" ") #공백을 기준으로 단어로 분리
    return preprocessed_sentence
```

1) Input

- The user inputs a query string.

2) Result

- The user's input query string is preprocessed accurately and displayed on the screen.

```
#문장을 전처리 시키는 함수 생성
def preprocess(sentence):
    preprocessed_sentence = sentence.strip().split(" ") #공백을 기준으로 단어로 분리
    print(preprocessed_sentence)
    return preprocessed_sentence
```

```
['name', 'my']
['this', 'makes', 'it', 'harder', 'for', 'female', 'polar', 'bears', 'to', 'have', 'babies.']
['name', 'my']
['this', 'puzzled', 'the', 'pig.']
['name', 'my']
['they', "didn't", 'do', 'their', 'homework', 'and', 'now', 'she', "can't", 'teach', 'the', 'lesson', 'she', 'planne
['name', 'my']
['how', 'are', 'you?']
['name', 'my']
['susie', 'was', 'a', 'young', 'gril.']
['name', 'my']
['what', 'are', 'you', 'going', 'to', 'watch', 'next?']
['name', 'my']
['you', 'can', 'go', 'into', 'the', 'world', 'of', 'the', 'past.']
['name', 'my']
['they', 'are', 'going', 'to', 'go', 'to', 'the', 'park', 'by', 'the', 'side', 'of', 'lake', "ch'unch'on."]
['name', 'my']
['so', 'i', "didn't", 'like', 'the', 'idea', 'of', 'my', 'son', 'becoming', 'a', 'cook.']
['name', 'my']
```

3) Description

- The function receives a string as input and splits it into words based on spaces using the 'sentence' function, returning the result.

- The preprocessed query is then displayed to the user.

3. Requirement: Similarity Calculation

```
# 쿼리와 문장의 유사도를 계산하는 함수
def calc_similarity(preprocessed_query, preprocessed_sentences):
    score_dict = {} #문장 인덱스와 유사도를 저장할 수 있는 딕셔너리 생성
    for i in range(len(preprocessed_sentences)):
        # 시작: 대소문자 구분 없는 토큰 셋을 만들기 위한 코드
        sentence = preprocessed_sentences[i] #파일에서 읽은 전처리 문장
        query_str = ' '.join(preprocessed_query).lower() #사용자에게 입력받은 쿼리를 공백으로 연결 후 소문자
        sentence_str = ' '.join(sentence).lower() #파일에서 읽은 전처리 문장을 공백으로 연결하고 소문자로 변
        preprocessed_query = set(preprocess(query_str)) #쿼리를 다시 전처리 후 토큰의 집합을 생성
        preprocessed_sentence = preprocess(sentence_str)  #문장을 다시 전처리
        # 끝: 대소문자 구분 없는 토큰 셋을 만들기 위한 코드

        file_token_set = set(preprocessed_sentence) #파일에서 읽은 문장의 토큰에 집합
        all_tokens = preprocessed_query | file_token_set #쿼리와 문장의 모든 토큰의 집합
        same_tokens = preprocessed_query & file_token_set # 쿼리와 문장에서 공통된 토큰 집합
        similarity = len(same_tokens) / len(all_tokens) #유사도 계산
        score_dict[i] = similarity #문장의 인덱스와 유사도를 딕셔너리에 저장
    return score_dict #딕셔너리 반환
```

1) Input

- Preprocessed query from the user and preprocessed sentences from the user-entered file are input to the 'calc_similarity' function.

2) Result

- After finding the common token set between the query and the sentences, the function calculates similarity and stores it in a dictionary, which is returned.

```
# 쿼리와 문장의 유사도를 계산하는 함수
def calc_similarity(preprocessed_query, preprocessed_sentences):
    score_dict = {} #문장 인덱스와 유사도를 저장할 수 있는 딕셔너리 생성
    for i in range(len(preprocessed_sentences)):
        # 시작: 대소문자 구분 없는 토큰 셋을 만들기 위한 코드
        sentence = preprocessed_sentences[i] #파일에서 읽은 전처리 문장
        query_str = ' '.join(preprocessed_query).lower() #사용자에게 입력받은 쿼리를
        sentence_str = ' '.join(sentence).lower() #파일에서 읽은 전처리 문장을 공백으
        preprocessed_query = set(preprocess(query_str)) #쿼리를 다시 전처리 후 토큰으
        preprocessed_sentence = preprocess(sentence_str)  #문장을 다시 전처리
        # 끝: 대소문자 구분 없는 토큰 셋을 만들기 위한 코드

        file_token_set = set(preprocessed_sentence) #파일에서 읽은 문장의 토큰에 집합
        all_tokens = preprocessed_query | file_token_set #쿼리와 문장의 모든 토큰의
        same_tokens = preprocessed_query & file_token_set # 쿼리와 문장에서 공통된 토
        similarity = len(same_tokens) / len(all_tokens) #유사도 계산
        score_dict[i] = similarity #문장의 인덱스와 유사도를 딕셔너리에 저장
        print(score_dict)
    return score_dict #딕셔너리 반환
```

```
{0: 0.0}
{0: 0.0, 1: 0.0}
{0: 0.0, 1: 0.0, 2: 0.0}
{0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0}
{0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0}
{0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0}
{0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0}
{0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0}
{0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0}
{0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0, 9: 0.0}
{0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0, 9: 0.0,
10: 0.0}
{0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0, 9: 0.0,
10: 0.0, 11: 0.0}
{0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0, 9: 0.0,
10: 0.0, 11: 0.0, 12: 0.0}
{0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0, 9: 0.0,
10: 0.0, 11: 0.0, 12: 0.0, 13: 0.0}
```

3) Description

- After creating a case-insenstive token set, the file and the query provided by the user are concatenated with spaces and coverted to lowercase.

- After reprocessing the sentences, it calculates the similarity by finding the set of common tokens between the query and the sentences through the combination of tokens, file, and query.

4. Requirement: Ranking of Sentences

```
# 5. Print the result
if sorted_score_list[0][1] == 0.0:
    print("There is no similar sentence.")
else:
    print("rank", "Index", "score", "sentence", sep = "₩t")
    rank = 1
    for i, score  in sorted_score_list:
        print(rank, i, score, ' '.join(file_tokens_pairs[i]), sep = "₩t")
        if rank == 10:
            break
        rank = rank + 1
```

1) Input

- A list where sentences are sorted by their similarity scores.

2) Result

- Return value: None

- Output the sentences with the top 10 similarity scores and their scores.

- if there are no similar sentences, it prints that there are none.

3) Description

- If the score of the first item in the sorted list is 0, it is determined that there are no similar sentences, and a message is displayed.

- Otherwise, it prints the rankings from 1st to 10th, along with the sentence number and the sentences.

Final Test Screenshots

1. When There Are No Similar Sentences

```
영어 쿼리를 입력하세요.dbtkskg answkd
There is no similar sentence.
```

2. When There Are Similar Sentences

```
영어 쿼리를 입력하세요.my name
rank    Index   score   sentence
1       679     0.5     My name is Mike.
2       526     0.2     Bob is my brother.
3       538     0.2     My hobby is traveling.
4       453     0.16666666666666666     My mother is sketching them.
5       667     0.16666666666666666     Mike: I'm cleaning my room.
6       195     0.14285714285714285     Hi, my fans. My name's Diana Rich.
7       241     0.14285714285714285     My father is running with So-ra.
8       336     0.14285714285714285     My family is at the park.
9       388     0.14285714285714285     I even have my feet pickled!
10      564     0.14285714285714285     But my grandfather wasn't so kind.
```

[Results and Conclusion]

1. Project Result: We created a search engine.
2. Reflections: It still seems that Python is a complex and challenging process. However, while performing the project, I was able to gain a better understanding of coding, and although it was challenging to do it alongside the semester, it was a rewarding assignment nonetheless.