Report

1. Introduction

In recent times, the capability to search and retrieve relevant information has become an integral part of various software applications. Search engines, at their core, are designed to find the most pertinent content based on user input. In this project, we aim to build a basic search engine in Python.

[Project Purpose and Background]

This project was undertaken to apply the knowledge learned in the seven weeks.

- The objective was to practice practical implementation based on the lessons covered.

File Input/Output: The code utilizes the `open(file_name, "r", encoding="utf8").readlines()` method to read files. Through this, a functionality to read and process datasets was implemented.

String Processing and Tokenization: The functions `strip()`, `split(" ")`, and `lower()` were used for sentence preprocessing. The crux of the search lies in appropriately processing the text data.

Utilization of Data Structures: The `set` data structure was employed to formulate a non-duplicate token set for similarity calculation. The `set` was harnessed to perform union and intersection operations, which in turn helped in calculating similarity.

Functionization and Modularization: Various functionalities in the code were compartmentalized and implemented as functions. Functions like `preprocess`, `indexing`, and `calc_similarity` were used to enhance the modularity and reusability of the code.

Goal: To develop a basic search engine that retrieves sentences similar to the user's query.

---

2. Requirements

1) User requirements:

- The system should be capable of searching for sentences similar to the user's query.

2) Functional Requirements:

- **Input**: The user provides an English query.
- **Processing**: The system tokenizes the query and the sentences from the dataset and calculates the similarity.
- **Output**: The system outputs the top 10 similar sentences to the query based on similarity scores.

---

3. Design and Implementation

Implementation Details:

1. **Preprocessing**:
    - This function preprocesses the given sentence, converts it into lowercase, strips it, and splits it into tokens.

```
def preprocess(sentence):
    preprocessed_sentence = sentence.strip().lower().split(" ")
    return preprocessed_sentence # 전처리된 토큰 리스트.
```

    - **Input**:
        - `sentence`: The original sentence that needs preprocessing.

- **Output**:
  - **Return Value**: A list of tokens converted to lowercase.
  - The sentence is converted to lowercase and tokenized for return.
- **Description**:
  - The input sentence is converted to lowercase.
  - The sentence is tokenized based on spaces.

2. **Indexing**:

- It indexes the given file into token pairs for each line.

```
# 파일 인덱싱 함수.
def indexing(file_name):
    #주어진 파일의 각 라인을 토큰화하고, 그 결과를 리스트로 반환

    file_tokens_pairs = []
    lines = open(file_name, "r", encoding="utf8").readlines()

    for line in lines:
        tokens = preprocess(line)
        file_tokens_pairs.append(tokens)
    return file_tokens_pairs
```

- **Input**:
  - `file_name`: The name of the file from which token pairs need to be extracted.
- **Output**:
  - **Return Value**: A list containing lists of tokens for each line.
  - Each line of the file is preprocessed and tokenized for storage.
- **Description**:
  - The file is opened and read line by line using the provided file name.
  - Each line is tokenized using the preprocessing function.
  - The tokenized result is added to a list.

3. **Similarity** :

- Calculates the similarity between the preprocessed query and sentences in the dataset.

```python
# 유사도 계산 함수.
def calc_similarity(preprocessed_query, preprocessed_sentences):
    score_dict = {}

    # 대소문자 구분 없이 쿼리 처리
    query_str = ' '.join(preprocessed_query).lower()
    preprocessed_query = set(preprocess(query_str))

    for i, sentence in enumerate(preprocessed_sentences):

        # 대소문자 구분 없이 문장처리
        sentence_str = ' '.join(sentence).lower()
        preprocessed_sentence = set(preprocess(sentence_str))

        # 유사도 계산
        all_tokens = preprocessed_query | preprocessed_sentence
        same_tokens = preprocessed_query & preprocessed_sentence
        similarity = len(same_tokens) / len(all_tokens)

        score_dict[i] = similarity

    return score_dict
```

- **Input**:
  - `preprocessed_query`: A list of tokens from the preprocessed query.
  - `preprocessed_sentences`: A list containing lists of tokens for each sentence.
- **Output**:
  - **Return Value**: A dictionary storing similarity scores for each sentence.
  - The similarity between the query and each sentence is calculated.
- **Description**:
  - The preprocessed query is joined into a string and then converted into lowercase. It is subsequently tokenized and transformed into a set of tokens.
  - For each sentence, it is joined into a string, converted to lowercase, tokenized, and then transformed into a set of tokens.

- The similarity is computed by comparing the tokens of the preprocessed query with the tokens of each preprocessed sentence.
- The similarity is represented as the ratio of the size of the intersection (common tokens between query and sentence) to the size of the union (total unique tokens between query and sentence).

---

4. Testing

- Testing involved verifying if the system provided the expected results based on user input.

Test Results for Each Functionality:

Preprocess:

```
def preprocess(sentence):
    preprocessed_sentence = sentence.strip().lower().split(" ")
    return preprocessed_sentence

# 함수 테스트
test_sentence = " This is a Sample Sentence. "
result = preprocess(test_sentence)
print("Original Sentence:", test_sentence)
print("Preprocessed Tokens:", result)
```

```
Original Sentence:  This is a Sample Sentence.
Preprocessed Tokens: ['this', 'is', 'a', 'sample', 'sentence.']
```

indexing:

```python
# 함수 테스트
test_file_name = "jhe-koen-dev.en"
result = indexing(test_file_name)
print("\nTokenized Lines from File:")
for idx, line_tokens in enumerate(result):
    print(f"Line {idx + 1}: {line_tokens}")
```

```
["you'll", 'be', 'picking', 'fruit', 'and', 'generally', 'helping', 'us', 'do', 'all', 'the', 'usual', 'farm',
'work.']
['in', 'the', 'middle', 'ages,', 'cities', 'were', 'not', 'very', 'clean,', 'and', 'the', 'streets', 'were',
'filled', 'with', 'garbage.']
['for', 'the', 'moment', 'they', 'may', 'yet', 'be', 'hiding', 'behind', 'their', 'apron', 'strings,', 'but',
'sooner', 'or', 'later', 'their', 'society', 'will', 'catch', 'up', 'with', 'the', 'progressive', 'world.']
['do', 'you', 'know', 'what', 'the', 'cow', 'answered?"', 'said', 'the', 'minister.']
['poland', 'and', 'italy', 'may', 'seem', 'like', 'very', 'different', 'countries.']
['mr.', 'smith', 'and', 'i', 'stayed', 'the', 'whole', 'day', 'in', 'oxford.']
['the', 'sight', 'of', 'a', 'red', 'traffic', 'signal', 'gave', 'him', 'an', 'idea.']
['so', 'they', 'used', 'pumpkins', 'instead.']
```

- The function was tested using various queries to ensure the correct sentences were retrieved.
- (Insert screenshots of each test case here)

Final Test Screenshot:

In case there are no similar sentences)

```python
    for i, score  in sorted_score_list:
        print(rank, i, score, ' '.join(file_tokens_
        if rank == 10:
            break
        rank = rank + 1
```

```
영어 쿼리를 입력하세요.Hello
There is no similar sentence.
```

In case there are similar sentences)

```
rank = rank + 1
영어 쿼리를 입력하세요.Hello My name is Subin Ahn
rank    Index   score   sentence
1       679     0.42857142857142855     My name is Mike.
2       526     0.25    Bob is my brother.
3       538     0.25    My hobby is traveling.
4       453     0.2222222222222222      My mother is sketching them.
5       241     0.2     My father is running with So-ra.
6       336     0.2     My family is at the park.
7       212     0.18181818181818182     My sister Betty is waiting for me.
8       505     0.16666666666666666     My little sister Annie is five years
9       610     0.14285714285714285     I would raise my voice and yell, "LUI
10      190     0.125   It is Sunday.
```

5. Results and Conclusion

Result:

1. A text preprocessing feature was developed, effectively tokenizing sentences and converting them to lowercase.
2. An indexing function was established, extracting token pairs for each sentence within a document.
3. A similarity calculation feature was implemented, enabling efficient search for relevant sentences based on a given query.

Reflection:

During lectures, the pacing is quite fast, which often causes me to miss out on some points even if I try to grasp just a bit. This aspect is regrettable. For Python beginners like myself, it poses some challenging moments to keep up and understand fully.

It felt as though we were given challenging assignments immediately after learning new material, without ample time to truly digest and practice what we had learned. Without having fully grasped the previous lessons, it became increasingly difficult to keep up in subsequent classes, especially given the rapid pace of the lectures.