Python Programming and Lab Report

Search Engine Implementation

221186 정승현

introduction

Purpose and background of the project: It was conducted to review and conduct simple practical training based on what has been learned through Python programming and practical training.

Practice Objective: Create a search engine to check the similarity between typed sentences and those in text files

Requirements

User Requirements: Added a system that searches text and documents for similar sentences when users type, and a case-insensitive search function when searching text and documents.

Design and Implementation

```
def preprocess(sentence):
    preprocess_sentence = sentence.strip().split(" ") # 검색쿼리를 토큰화 한다.
    return preprocess_sentence
```

- Specify a function called preprocess and receive a parameter called sentence.
- A function called preprocess removes the margin of the sentence received by the parameter and divides it into spaces. (Tokenization)
- - Returns tokenized preprocess_sentence.

```
def indexing(file_name):
    # 검색 대상 파일을 불러오고 preprocess함수를 통해 토큰화를 한다음 반복문을 통하
    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
```

- Specify a function called indexing and receive a parameter called file_name.
- - Create an empty list to store the tokens of the read text file.
- - A function called indexing reads the file_name received by the parameter, saves it in a list with readlines, and then saves it in lines.
- - The lines are stored in the line through repeated sentences, tokenized through the preprocess function, and stored in tokens.
- - Store tokenized tokens in a pre-created empty list called file_tokens_pairs.

```
def calc_similarity(preprocessed_query, preprocessed_sentences):
    score_dict = {} # 검색 쿼리와 검색 대상 문장들에서 모든 토큰과 공통 토큰을 찾
    for i in range(len(preprocessed_sentences)):

        # sentence에 prepreocessed_sentences의 리스트형태를 저장한다.
        sentence = preprocessed_sentences[i]
        # sentence는 리스트형이므로 띄어쓰기를 기준으로 나눠 문자열로 바꿔준 뒤 소
        sentence_str = ' '.join(sentence).lower()
        # 소문자로 바뀐 검색 파일을 토큰화 해준다.
        preprocess_sentence = preprocess(sentence_str)
        # set의 합집합, 교집합 기호를 사용하기 위해 set으로 바꿔준다.

        file_token_set = set(preprocess_sentence)
        all_tokens = query_token_set | file_token_set
        same_tokens = query_token_set & file_token_set
        similarity = len(same_tokens) / len(all_tokens)
        score_dict[i] = similarity
    return score_dict
```

- A function called calc_similarity is specified, and preprocessed_query and preprocessed_sentences are received as parameters.
- Repeat only each search target sentence in the tokenized preprocessed_sentences list.
- Sentence stores the result of indexing preprocessed_sentences as a list.
- In sentence_str, words in the sentence list are distributed based on spacing, and words changed into strings are changed to lowercase letters.
- It tokens all lowercase search files through the preprocess function.
- pre to use the set's sum and interset symbols to see similarityChange process_sentence to set type.
- All_tokens is a collection of all tokens in search queries and sentences.
- The same_tokens represents tokens included in common between the search query and the token of the search target sentence.
- Similarity is the value of all_tokens divided by same_tokens, and the higher the value, the higher the similarity.
- A similarity value indicating similarity between the relevant sentence and the

search query is stored as a value by using an index of the search object sentence as a key in a score_dict.

```
# 1. Indexing
file_name = "jhe-koen-dev.en"
file_tokens_pairs = indexing(file_name)
```

- Read the file whose file_name is "jhe-koen-dev.en" through the indexing function, tokenize it, and save it to the file_tokens_pairs list.

```
# 2. Input the query
query = input("영어 쿼리를 입력하세요.").lower() # 검색 쿼리를 모두 소문자로 바꿔
preprocessed_query = preprocess(query)
query_token_set = set(preprocessed_query)
```

- - A user is made to input a search query, and all characters of the query are changed into lowercase letters and stored in the query.
- - query as preprocess function Tokenize by .
- - Change preprocessd_query to set type so that you can use the set's sum and cross-set symbols to see similarities

```
# 3. Calculate similarities based on a same token set
score_dict = calc_similarity(query_token_set, file_tokens_pairs)
```

- A query_token_set and file_token_pairs tokenized in a calc_similarity function are defined as respective parameters.
- - The calc_similarity function measures the similarity between query_token_set and file_token_pairs.

```
# 4. Sort the similarity list
sorted_score_list = sorted(score_dict.items(), key = operator.itemgetter(1), rever
```

- The sorted_score_list is a list that contains sentence indexes arranged in descending order of similarity scores and similarity scores of the 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
```

- If the score of the first item is zero in the sorted list, it is determined that there is no similar sentence and a message is outputted.
- - If not, the order from 1st to 10th place, sentence number, and sentence are printed.

Testing

1) When there is no similar sentence

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

2) When there is similar sentence

```
영어 쿼리를 입력하세요.Hello my name is jsh
rank    Index   score   sentence
1       679     0.5     My name is Mike.
2       526     0.2857142857142857      Bob is my brother.
3       538     0.2857142857142857      My hobby is traveling.
4       453     0.25    My mother is sketching them.
5       241     0.2222222222222222      My father is running with So-ra.
6       336     0.2222222222222222      My family is at the park.
7       212     0.2     My sister Betty is waiting for me.
8       505     0.18181818181818182     My little sister Annie is five years old.
9       610     0.15384615384615385     I would raise my voice and yell, "LUNCH IS
READY!"
10      190     0.14285714285714285     It is Sunday.
```

3) User entered all lowercase queries

```
영어 쿼리를 입력하세요.life is too short
rank    Index   score   sentence
1       190     0.16666666666666666     It is Sunday.
2       314     0.16666666666666666     This is Washington.
3       438     0.16666666666666666     Short of cash?
4       675     0.16666666666666666     Se-na: It's too hot today. When is your E
nglish test?
5       710     0.16666666666666666     Travel is exciting.
6       526     0.14285714285714285     Bob is my brother.
7       538     0.14285714285714285     My hobby is traveling.
8       679     0.14285714285714285     My name is Mike.
9       45      0.125   This method is called *acupuncture.
10      107     0.125   But this is very interesting.
```

4) When the user enters an input query in all uppercase letters

```
영어 쿼리를 입력하세요.LIFE IS TOO SHORT
rank    Index   score   sentence
1       190     0.16666666666666666     It is Sunday.
2       314     0.16666666666666666     This is Washington.
3       438     0.16666666666666666     Short of cash?
4       675     0.16666666666666666     Se-na: It's too hot today. When is your E
nglish test?
5       710     0.16666666666666666     Travel is exciting.
6       526     0.14285714285714285     Bob is my brother.
7       538     0.14285714285714285     My hobby is traveling.
8       679     0.14285714285714285     My name is Mike.
9       45      0.125   This method is called *acupuncture.
10      107     0.125   But this is very interesting.
```

Conclusion) Similarity is measured case-insensitive when a user enters an input query

5) search engine full code

```python
import operator

def preprocess(sentence):
    preprocess_sentence = sentence.strip().split(" ") # 토큰화 한다.
    return preprocess_sentence

def indexing(file_name):
    # 검색 대상 파일을 불러오고 preprocess함수를 통해 토큰화를 한다음 반복문을 통해 리스트에 저장한다.
    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

def calc_similarity(preprocessed_query, preprocessed_sentences):
    score_dict = {} # 검색 쿼리와 검색 대상 문장들에서 모든 토큰과 공통 토큰을 찾아보고 유사 토큰을 score_dict에 저장한다.
    for i in range(len(preprocessed_sentences)):

        # sentence에 prepreocessed_sentences의 리스트형태를 저장한다.
        sentence = preprocessed_sentences[i]
        # sentence는 리스트형이므로 띄어쓰기를 기준으로 나눠 문자열로 바꿔준 뒤 소문자로 변환한다.
        sentence_str = ' '.join(sentence).lower()
        # 소문자로 바뀐 검색 파일을 토큰화 해준다.
        preprocess_sentence = preprocess(sentence_str)
        # set의 합집합, 교집합 기호를 사용하기 위해 set으로 바꿔준다.
        file_token_set = set(preprocess_sentence)

        all_tokens = query_token_set | file_token_set
        same_tokens = query_token_set & file_token_set
        similarity = len(same_tokens) / len(all_tokens)
        score_dict[i] = similarity
    return score_dict


# 1. Indexing
file_name = "jhe-koen-dev.en"
file_tokens_pairs = indexing(file_name)

# 2. Input the query
query = input("영어 쿼리를 입력하세요.").lower() # 검색 쿼리를 모두 소문자로 바꿔준다.
preprocessed_query = preprocess(query)
query_token_set = set(preprocessed_query)

```

```python
# 3. Calculate similarities based on a same token set
score_dict = calc_similarity(query_token_set, file_tokens_pairs)

# 4. Sort the similarity list
sorted_score_list = sorted(score_dict.items(), key = operator.itemgetter(1), reverse=True)

# 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
```

Conclusion

Project Results: We created a search engine that selects sentences from the top 1 to 10 with high similarities between given text documents and user-entered queries.

point of feeling: I think the challenge that goes on every time is not a simple problem, but a challenge itself. I always feel like I'm climbing an insurmountable wall while solving the challenge, and the problem I'm solving seems to be going up a mountain. Sometimes, if you look at the code and solve the problem as I want,

the challenge will be solved. And it's hard to improve your skills only when you solve difficult problems, but if you continue to do so, it seems that your skills will improve, so you get a lot every time.