

CSE508 Winter2024 A1

Github Repo Link - https://github.com/NalishJain/CSE508_Winter2024_A1_2021543

Question 1

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
import pickle
from itertools import product

# Ensure NLTK resources are downloaded
nltk.download('punkt')
nltk.download('stopwords')
```

```
def preprocess_text(text):
    text = text.lower()
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))

    tokens = [word for word in tokens if word.lower() not in stop_words]
    tokens = [word for word in tokens if word not in string.punctuation]
    tokens = [word for word in tokens if word.strip() != '']
    preprocessed_text = ' '.join(tokens)

    return preprocessed_text
```

For preprocessing I have used nltk library, firstly I converted all the strings to lower case using `.lower()`, then I tokenised the words using `word_tokenize()`, then I removed the stop words and punctuation and lastly I removed all the empty spaces. Then I used `' '.join(tokens)` operation to join the pre-processed words and wrote it back to the processed file using the following code.

```

for id in range(1, 1000):
    input_file_path = "/Users/nalishjain/Documents/GitHub/CSE508_Winter2024_A1_2021543/text_files/file" + str(id) + ".txt"
    output_file_path = "/Users/nalishjain/Documents/GitHub/CSE508_Winter2024_A1_2021543/out_files/file" + str(id) + "processed" + ".txt"
    if id <= 5:
        with open(input_file_path, 'r', encoding='utf-8') as file:
            print("Text before pre-processing")
            text = file.read()
            print(text)
            preprocessed_text = preprocess_text(text)

            print("Text after pre-processing")
            print(preprocessed_text)
            print()
            with open(output_file_path, 'w', encoding='utf-8') as file:
                file.write(preprocessed_text)
    else:
        with open(input_file_path, 'r', encoding='utf-8') as file:
            text = file.read()

            preprocessed_text = preprocess_text(text)
            with open(output_file_path, 'w', encoding='utf-8') as file:
                file.write(preprocessed_text)

```

Output for first 5 samples in que1

1. Text before pre-processing

Loving these vintage springs on my vintage strat. They have a good tension and great stability. If you are floating your bridge and want the most out of your springs than these are the way to go.

Text after pre-processing

loving vintage springs vintage strat good tension great stability floating bridge want springs way go

2. Text before pre-processing

Works great as a guitar bench mat. Not rugged enough for abuse but if you take care of it, it will take care of you. Makes organization of workspace much easier because screws won't roll around. Color is good too.

Text after pre-processing

works great guitar bench mat rugged enough abuse take care take care makes organization workspace much easier screws wo n't roll around color good

3. Text before pre-processing

We use these for everything from our acoustic bass down to our ukuleles. I know there is a smaller model available for ukes, violins, etc.; we haven't yet ordered those, but these will work on smaller instruments if one doesn't extend the feet to their maximum width. They're gentle on the instruments, and the grippy material keeps them secure.

The greatest benefit has been when writing music at the computer and needing to set a guitar down to use the keyboard/mouse - just easier for me than a hanging stand.

We have several and gave one to a friend for Christmas as well. I've used mine on stage, and it folds up small enough to fit right in my gig bag.

Text after pre-processing

use everything acoustic bass ukuleles know smaller model available ukes violins etc n't yet ordered work smaller instruments one n't extend feet maximum width 're gentle instruments grippy material keeps secure greatest benefit writing music computer needing set guitar use keyboard/mouse easier hanging stand several gave one friend christmas well 've used mine stage folds small enough fit right gig bag

4. Text before pre-processing

Great price and good quality. It didn't quite match the radius of my sound hole but it was close enough.

Text after pre-processing

great price good quality n't quite match radius sound hole close enough

5. Text before pre-processing

I bought this bass to split time as my primary bass with my Dean Edge. This might be winning me over. The bass boost is outstanding. The active pickups really allow you to adjust to the sound you want. I recommend this for anyone. If you're a beginner like I was not too long ago, it's an excellent bass to start with. If you're on tour and/or music is making you money, this bass will be beautiful on stage. The color is a bit darker than in the picture. But, all around, this is a great buy.

Text after pre-processing

bought bass split time primary bass dean edge might winning bass boost outstanding active pickups really allow adjust sound want recommend anyone 're beginner like long ago 's excellent bass start 're tour and/or music making money bass beautiful stage color bit darker picture around great buy

Question 2

```
def create_index(file_path):
    index_dict = {}
    for id in range(1, 1000):
        processed_file_path = file_path + str(id) + "processed" + ".txt"
        with open(processed_file_path, 'r', encoding='utf-8') as file:
            text = file.read()
            words = text.split(" ")
            words = set(words)
            for word in words:
                if word not in index_dict:
                    index_dict[word] = set()
                    index_dict[word].add(id)
                else:
                    index_dict[word].add(id)
    return index_dict
```

I have used a dictionary to create index, with mapping from term to document number

```
index_dict = create_index(file_path="/Users/nalishjain/Documents/GitHub/CSE508_Winter2024_A1_2021543/out_files/file")
with open("index_dictionary.pkl", 'wb') as pickle_file:
    pickle.dump(index_dict, pickle_file)

file_path = 'index_dictionary.pkl'
with open(file_path, 'rb') as pickle_file:
    loaded_dict = pickle.load(pickle_file)
print("Loaded Dictionary:", loaded_dict)
```

I have saved and loaded the dictionary using pickle.dump() and pickle.load()

```
def take_input(n):
    terms_list = []
    operators_list = []
    for i in range(n):
        input_user = input("Enter sentence : ")
        # print(input_user)
        input_user = preprocess_text(input_user).split(" ")
        terms_list.append(input_user)

        input_user = input("Enter operator : ")
        input_user = input_user.split(",")
        operators_list.append(input_user)
        operators_list = [[word.strip() for word in sublist] for sublist in operators_list]
    return (terms_list, operators_list)
```

I have taken input from the user using the following code, and preprocessed using the process_text function.

I am handling the user query function in the following way

```
def execute_query(terms, operators, inverted_index, total_doc_num):
    result = None

    query_string = ""
    for i, word in enumerate(terms):
        query_string += word
        if i < len(terms) - 1:
            query_string += f' {operators[i % len(operators)]} '

    for i in range(0, len(terms)):
        # AND, OR, AND NOT, OR NOT
        term = terms[i]
        if result is None:
            result = inverted_index.get(term, set())
            operator = ""
        else:
            operator = operators[i-1]
            if operator == 'AND':
                result = result.intersection(inverted_index.get(term, set()))

            elif operator == 'OR':
                result = result.union(inverted_index.get(term, set()))

            elif operator == 'AND NOT':
                total_docs = set(range(1, total_doc_num))
                not_term = total_docs.difference(inverted_index.get(term, set()))
                result = result.intersection(not_term)

            elif operator == 'OR NOT':
                total_docs = set(range(1, total_doc_num))
                not_term = total_docs.difference(inverted_index.get(term, set()))
                result = result.union(not_term)

    # print(result)
    print("*"*20)
    print("Query is : ", query_string)
    print(f"Number of documents retrieved: {len(result)}")
    print(f"Names of the documents retrieved:", ["file" + str(id) + ".txt" for id in sorted(result)])
    print("*"*20)
```

I am using the set operations to perform the boolean logic operations. For ‘AND’ operation I am using `set.intersection()`, for ‘OR’ I am using `set.union()`, for ‘OR NOT’ I am using `set.difference()` first to perform the not operation and then doing `.union()` similarly for ‘AND NOT’ I use `.intersection()` performing `.difference()`

Question 3

```
def create_positional_index(file_path):
    position_index_dict = {}
    for id in range(1, 1000):
        processed_file_path = file_path + str(id) + "processed" + ".txt"
        with open(processed_file_path, 'r', encoding='utf-8') as file:
            text = file.read()
            words = text.split(" ")
            for position, word in enumerate(words):
                if word not in position_index_dict:
                    position_index_dict[word] = {}
                if id not in position_index_dict[word]:
                    position_index_dict[word][id] = []
                position_index_dict[word][id].append(position)

    return position_index_dict
```

The `create_positional_index` function processes a series of text files identified by numerical IDs, tokenizes each file into words, and builds a positional index dictionary. This dictionary stores the positions of words within each file, organized by word and file ID. The resulting `position_index_dict` provides a mapping of words to their occurrences and positions across multiple text files.

```
position_index_dict = create_positional_index("/Users/nalishjain/Documents/GitHub/CSE508_Winter2024_A1_2021543/out_files/file")
with open("position_index_dictionary.pkl", 'wb') as pickle_file:
    pickle.dump(position_index_dict, pickle_file)

file_path = 'position_index_dictionary.pkl'
with open(file_path, 'rb') as pickle_file:
    loaded_dict = pickle.load(pickle_file)
print("Loaded Dictionary:", loaded_dict)
```

I have saved and loaded my positional index in the form of dictionary

I have handled the query with the help of the following functions, I create all the possible combinations of the positional indices using `cartesian_product()` and out of those combinations I select only those where the difference between consecutive indices is 1.

```
def cartesian_product(*lists):
    if not lists:
        return [()]
    else:
        result = []
        for product_tail in cartesian_product(*lists[1:]):
            for item in lists[0]:
                result.append((item,) + product_tail)
        return result


def generate_combinations(lists):
    all_combinations = cartesian_product(*lists)
    valid_combinations = [comb for comb in all_combinations if all(comb[i] - comb[i-1] == 1 for i in range(1, len(comb)))]
    return valid_combinations

def execute_positional_query(terms, positional_index):
    result = None

    for term in terms:
        if result is None:
            result = set(positional_index.get(term, {}).keys())
        else:
            result = result.intersection(positional_index.get(term, {}).keys())

    for id in list(result):
        positions = [positional_index[term][id] for term in terms]
        combinations = generate_combinations(positions)
        if len(combinations) != 0:
            continue
        else:
            result.remove(id)

    print("*"*20)
    print("Query : ", terms)
    print(f"Number of documents retrieved: {len(result)}")
    print(f"Names of the documents retrieved:", ["file" + str(id) + ".txt" for id in sorted(result)])
    print("*"*20)
```

 Are you using a screen reader?