Search Engine

Information Retrieval

Submitted to
Dr. Syed Khaldoon Khurshid

Submitted By

Shahzaib Irfan 2021-CS-07

University of Engineering and Technology
Lahore, Pakistan

Table of Contents

1 Introduction

This document provides an overview and explanation of the code that implements a Search Engine. The system includes functionalities to add, list, and search documents by title or content, with an index-based search that includes token expansion using synonyms. The code utilizes `NLTK` (Natural Language Toolkit) for natural language processing, including tokenization, part-of-speech tagging, and lemmatization.

2 Imports and Setup

```
import os
import math
from collections import defaultdict, Counter
from nltk import word_tokenize, pos_tag
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
```

The code imports required libraries for file handling, mathematical calculations, and NLP (Natural Language Processing). `NLTK` provides tools for tokenization, lemmatization, synonym extraction, and tagging.

3 Helper Functions

3.1 `get_synonyms(word)`

This function retrieves a set of synonyms for a given word using the WordNet lexical database.

```
def get_synonyms(word):
    synonyms = set()
    for syn in wordnet.synsets(word):
        for lemma in syn.lemmas():
            synonyms.add(lemma.name().lower())
    return synonyms
```

3.2 `expand_query_with_synonyms(query_tokens)`

This function expands the user's query by including synonyms of each word in the query.

```
def expand_query_with_synonyms(query_tokens):
    expanded_query = set(query_tokens)
    for word in query_tokens:
        expanded_query.update(get_synonyms(word))
    return list(expanded_query)
```

3.3 extract_nouns_and_entities(content)

This function extracts nouns from the content, which are used as keywords for indexing.

```
def extract_nouns_and_entities(content):
    words = word_tokenize(content.lower())
    pos_tags = pos_tag(words)
    nouns = [word for word, pos in pos_tags if pos in ('NN', 'NNS')]
    lemmatizer = WordNetLemmatizer()
    lemmatized_nouns = [lemmatizer.lemmatize(noun) for noun in nouns]
    return lemmatized_nouns
```

4 TF-IDF Calculation Functions

4.1 calculate_tf(doc_words)

Calculates the term frequency (TF) for each word in a document.

```
def calculate_tf(doc_words):
    tf = {}
    total_words = len(doc_words)
    word_counts = Counter(doc_words)
    for word, count in word_counts.items():
        tf[word] = count / total_words
    return tf
```

4.2 calculate_idf(documents)

Calculates the inverse document frequency (IDF) for each unique word across all documents.

```
def calculate_idf(documents):
    idf = {}
    total_docs = len(documents)
    word_doc_counts = defaultdict(int)
    for doc_words in documents:
        unique_words = set(doc_words)
        for word in unique_words:
            word_doc_counts[word] += 1
    for word, doc_count in word_doc_counts.items():
        idf[word] = math.log(total_docs / (1 + doc_count))
    return idf
```

4.3 calculate_tfidf(doc_words, tf, idf)

Combines TF and IDF values to compute the TF-IDF score for each word.

```
def calculate_tfidf(doc_words, tf, idf):
    tfidf = {}
    for word in doc_words:
        if word in idf:
            tfidf[word] = tf[word] * idf[word]
    return tfidf
```

5 Phrase Matching Function

5.1 phrase_match(content, query)

This function checks if a query phrase matches any portion of the content.

```
def phrase_match(content, query):
    content_words = word_tokenize(content.lower())
    query_words = word_tokenize(query.lower())
    for i in range(len(content_words) - len(query_words) + 1):
        if content_words[i:i+len(query_words)] == query_words:
            return True
    return False
```

6 Search Engine

6.1 SearchEngine Class

This class is responsible for managing document loading, indexing, and searching.

6.1.1 load_documents()

Loads documents from the specified directory.

```
def load_documents(self):
    """Load all documents from the documents directory."""
        print("\nLoading existing documents...")
        loaded_count = 0

        for filename in os.listdir(self.docs_directory): # lists all files in th
            if filename.endswith(".txt"):
                try:
                    with open(os.path.join(self.docs_directory, filename), 'r',
                        title = filename
```

```
                    content = file.read().strip() # read file and strip (rem
                    self.documents[title] = content
                    loaded_count += 1
                except Exception as e:
                    print(f"Error loading document {filename}: {str(e)}")

        if loaded_count > 0:
            print(f"Successfully loaded {loaded_count} documents.")
            self.update_index()
        else:
            print("No existing documents found.")
```

6.1.2 update_index()

Indexes each document based on important terms using TF-IDF values.

```
def update_index(self):
    doc_word_list = []

        for doc_id, content in self.documents.items():
            nouns_and_entities = extract_nouns_and_entities(content) # extracts
            doc_word_list.append(nouns_and_entities)

        idf = calculate_idf(doc_word_list)

        self.index.clear()

        for doc_id, content in self.documents.items():
            doc_words = extract_nouns_and_entities(content)
            tf = calculate_tf(doc_words)
            tfidf = calculate_tfidf(doc_words, tf, idf)

            important_terms = {term for term, score in tfidf.items() if score >
            for term in important_terms:
                self.index[term].append(doc_id) # important terms are made index
```

6.2 search_by_title(query)

Searches documents by title matching.

```
def search_by_title(self, query):
    matching_docs = []
        for doc_id in self.documents:
            if query.lower() in doc_id.lower():
                matching_docs.append(doc_id)
        return matching_docs
```

6.3 search_by_content(query)

Searches documents by content and includes synonym expansion for the query.

```
def search_by_content(self, query):
        matching_docs = set()

        # First try exact phrase matching
        for doc_id, content in self.documents.items():
            if phrase_match(content, query):
                matching_docs.add(doc_id)

        # If no exact matches found, try semantic search
        if not matching_docs:
            query_tokens = word_tokenize(query.lower())
            # Extract only nouns from the query for index-based search
            query_pos_tags = pos_tag(query_tokens)
            query_nouns = [word for word, pos in query_pos_tags
                           if pos in ('NN', 'NNS', 'NNP', 'NNPS')]

            # Get expanded tokens only for nouns
            expanded_noun_tokens = []
            for noun in query_nouns:
                expanded_noun_tokens.extend(expand_query_with_synonyms([noun]))

            # Find documents that match the nouns using the index
            noun_matching_docs = set()
            if expanded_noun_tokens:
                noun_matching_docs = set(self.index.get(expanded_noun_tokens[0],
                for token in expanded_noun_tokens[1:]:
                    if token in self.index:
                        noun_matching_docs &= set(self.index[token])

            # If we found documents matching nouns, filter them further using no
            if noun_matching_docs:
                non_noun_tokens = [word for word, pos in query_pos_tags
                                   if pos not in ('NN', 'NNS', 'NNP', 'NNPS')]

                # If there are no non-noun tokens, return the noun matches
                if not non_noun_tokens:
                    return noun_matching_docs

                # For each document that matched nouns, check if it contains the
                for doc_id in noun_matching_docs:
                    content = self.documents[doc_id].lower()
                    if all(token.lower() in content for token in non_noun_tokens
                        matching_docs.add(doc_id)
```

```
            else:
                # If no noun matches found, fall back to basic content search
                for doc_id, content in self.documents.items():
                    content_lower = content.lower()
                    if all(token.lower() in content_lower for token in query_tok
                        matching_docs.add(doc_id)

        return matching_docs
```

7 User Interface Functions

7.1 `display_results(docs)`

Displays search results to the user.

```
def display_results(docs):
    if docs:
        print("\nFound documents:")
        for idx, doc in enumerate(docs, 1):
            print(f"{idx}. {doc}")
    else:
        print("\nNo matching documents found.")
```

7.2 `get_multiline_input(prompt)`

Prompts the user for multiline input.

```
def get_multiline_input(prompt):
    print(prompt)
    print("(Enter an empty line to finish)")
    lines = []
    while True:
        line = input()
        if line.strip() == "":
            break
        lines.append(line)
    return "\n".join(lines)
```

8 Main Program Execution

The main function starts the program and handles the user menu.

```
def main():
    search_engine = SearchEngine(docs_directory="./archive/business")
```

```
while True:
    print("\n=== Search Engine ===")
    print("1. Add new document")
    print("2. List all documents")
    print("3. Search by title")
    print("4. Search by content")
    print("5. Exit")

    choice = input("\nEnter your choice (1-5): ").strip()

    if choice == "1":
        title = get_multiline_input("\nEnter document title:")
        if not title:
            print("Title cannot be empty!")
            continue

        content = get_multiline_input("\nEnter document content:")
        if not content:
            print("Content cannot be empty!")
            continue

        search_engine.add_document(title, content)

    elif choice == "2":
        search_engine.list_documents()

    elif choice == "3":
        query = input("\nEnter title to search: ")
        results = search_engine.search_by_title(query)
        display_results(results)

    elif choice == "4":
        query = input("\nEnter content to search: ")
        results = search_engine.search_by_content(query)
        display_results(results)

    elif choice == "5":
        print("\nThank you for using the Search Engine!")
        break

    else:
        print("\nInvalid choice! Please try again.")
```