Midterm Solutions Exercise 06

Nguyen Thi Phuong Thao - N21DCCN078
Nguyen Thi Minh Thu - N21DCCN082
Du Trong Nhan - N21DCCN060
Trinh Khanh Quan - N20DCCN057

May 17, 2025

Contents

1 Problem Statement

Implement a program that creates and uses inverted indexes with the following functionalities:

(a) **CreateIndex(Dir, StopList)**: Takes a directory name and a file called StopList (in that directory) as input. It returns an inverted index as output. The DocTable includes all files in the directory `Dir`, except for the StopList file. The TermTable includes only all words occurring in the directory that start with the letter "c" (lower- or uppercase).

(b) **Find(Word, Weight, N)**: Finds the top N documents in the index associated with the specified word, considering the given weight.

(c) **FindFromFile(WordFile, N)**: Similar to the above, but instead of taking a single word as input, it takes a file called WordFile. This file has, on each line, a word (string) and a weight (float). It then attempts to find, using the inverted index, the top N matches for this query.

2 Implementation Details

The implementation consists of a Python program (`inverted_index.py`) that creates and searches an inverted index structure for text documents.

2.1 Data Structures

The program uses two main data structures:

1. **DocTable**: A dictionary mapping document IDs (integers) to document names (filenames)

```
DocTable = {
    0: "document1.txt",
    1: "document2.txt",
    # ...
}
```

2. **TermTable**: An inverted index mapping terms to dictionaries that map document IDs to term frequencies

```
TermTable = {
    "computer": {0: 5, 2: 3},  # "computer" appears 5 times in doc 0, 3 times in doc 2
    "cloud": {1: 7, 4: 2},     # "cloud" appears 7 times in doc 1, 2 times in doc 4
    # ...
}
```

2.2 Key Functions

2.2.1 CreateIndex(Dir, StopList)

This function:

- Takes a directory path and a stop words file name as input
- Loads stop words from the specified file
- Processes each file in the directory (excluding the stop list file)
- Extracts words from each document
- Filters to include only words starting with 'c' (case insensitive)
- Removes stop words
- Counts word frequencies
- Builds the DocTable and TermTable
- Returns both tables

```
def CreateIndex(Dir: str, StopList: str) -> Tuple[Dict[int, str], Dict[str, Dict[int, int]]]:
    # Initialize data structures
    DocTable = {}  # Maps document IDs to filenames
    TermTable = defaultdict(lambda: defaultdict(int))  # Maps terms to {doc_id: frequency}

    # Load stop words
    stop_words = set()
    stop_list_path = os.path.join(Dir, StopList)
    if os.path.exists(stop_list_path):
        with open(stop_list_path, 'r', encoding='utf-8', errors='ignore') as f:
            stop_words = {word.strip().lower() for word in f}

    # List of files to process (excluding the StopList file)
    files_to_process = [f for f in os.listdir(Dir)
                        if os.path.isfile(os.path.join(Dir, f)) and f != StopList]

    # Process each file
    for doc_id, filename in enumerate(files_to_process):
        file_path = os.path.join(Dir, filename)

        # Add file to DocTable
        DocTable[doc_id] = filename

        try:
            # Process file contents
            with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
                content = f.read().lower()

                # Extract words (simple tokenization)
```

```
                words = re.findall(r'\b\w+\b', content)

                # Count words that start with 'c' and not in stop words
                word_counts = Counter(word for word in words
                                      if word.startswith('c') and word not in
    stop_words)

                # Update TermTable with word frequencies
                for word, count in word_counts.items():
                    TermTable[word][doc_id] = count
        except Exception as e:
            print(f"Error processing file {filename}: {e}")

    # Convert nested defaultdicts to regular dicts for return
    return DocTable, {term: dict(doc_freqs) for term, doc_freqs in
    TermTable.items()}
```

2.2.2 Find(Word, Weight, N)

This function:

- Searches for a specific word in the TermTable
- Calculates document scores based on term frequency * weight
- Returns the top N documents sorted by score

```
def Find(Word: str, Weight: float, N: int, DocTable: Dict[int, str],
         TermTable: Dict[str, Dict[int, int]]) -> List[Tuple[str, float
    ]]:
    word = Word.lower()
    results = []

    # Check if word exists in the index
    if word not in TermTable:
        print(f"Word '{word}' not found in the index.")
        return []

    # Get document IDs and their frequencies for the word
    doc_freqs = TermTable[word]

    # Calculate scores based on term frequency and term weight
    for doc_id, freq in doc_freqs.items():
        doc_name = DocTable[doc_id]
        # Score is the product of term frequency and the term weight
        score = freq * Weight
        results.append((doc_name, score))

    # Sort by score in descending order
    results.sort(key=lambda x: x[1], reverse=True)

    # Return top N results
    return results[:N]
```

2.2.3 FindFromFile(WordFile, N)

This function:

- Reads words and their weights from a file
- Accumulates scores for documents across all query terms
- Returns the top N documents sorted by total score

```
def FindFromFile(WordFile: str, N: int, DocTable: Dict[int, str],
                 TermTable: Dict[str, Dict[int, int]]) -> List[Tuple[str, float]]:
    # Initialize a dictionary to accumulate scores for each document
    document_scores = defaultdict(float)

    try:
        # Read words and weights from the file
        with open(WordFile, 'r', encoding='utf-8', errors='ignore') as f:
            for line in f:
                line = line.strip()
                if not line:
                    continue

                # Parse word and weight from the line
                parts = line.split()
                if len(parts) >= 2:
                    word = parts[0].lower()
                    try:
                        weight = float(parts[1])
                    except ValueError:
                        print(f"Invalid weight for word '{word}', using default weight 1.0")
                        weight = 1.0

                    # Check if word exists in the index
                    if word not in TermTable:
                        print(f"Word '{word}' not found in the index, skipping.")
                        continue

                    # Get document IDs and frequencies for the word
                    doc_freqs = TermTable[word]

                    # Calculate and accumulate scores
                    for doc_id, freq in doc_freqs.items():
                        doc_name = DocTable[doc_id]
                        # Score is product of term frequency and query term weight
                        score = freq * weight
                        document_scores[doc_name] += score

                else:
                    print(f"Ignoring invalid line: {line}")

        # Convert scores to list of tuples
```

```
        results = [(doc_name, score) for doc_name, score in
    document_scores.items()]

        # Sort by score in descending order
        results.sort(key=lambda x: x[1], reverse=True)

        # Return top N results
        return results[:N]

    except Exception as e:
        print(f"Error processing word file '{WordFile}': {e}")
        return []
```

2.3 Helper Functions

The program includes several helper functions:

- **display_index()**: Displays the contents of DocTable and TermTable
- **save_to_csv()**: Saves DocTable and TermTable to CSV files
- **main()**: The main program loop that handles user interaction

3 Installation Guide

3.1 Prerequisites

- Python 3.6 or higher

3.2 Installation Steps

1. Ensure Python is installed on your system:

```
python --version
```

2. Download the `inverted_index.py` file to your preferred location

4 Usage Instructions

4.1 Running the Program

1. Open a terminal or command prompt
2. Navigate to the directory containing the `inverted_index.py` file
3. Run the program:

```
python inverted_index.py
```

4.2 Using the Program

After running the program, follow these steps:

1. **Specify Document Directory**:
   - When prompted, enter the full path to the directory containing your text documents
2. **Specify StopList File**:
   - Enter the name of the file containing stop words (words to be excluded from indexing)
3. **View the Created Index**:
   - The program will display the created DocTable and TermTable
   - The results will also be saved to CSV files (`doc_table.csv` and `term_table.csv`)
4. **Search Options**:
   - Option 1: Search by a single word
     - Enter the word, its weight, and the number of results to retrieve
   - Option 2: Search using a word file
     - Enter the path to a file containing words and weights
     - Enter the number of results to retrieve
   - Option 3: Exit the program

4.3 Word File Format

The word file should have the following format:

```
word1 weight1
word2 weight2
...
```

Example:

```
computer 1.5
coding 2.0
cloud 0.8
```

5 Examples

5.1 Example 1: Creating an Index

Input:

- Directory: `/documents`
- StopList: `stopwords.txt`

Contents of Directory:

- `article1.txt`: "Computer science is a fascinating field. Computing power has increased."
- `article2.txt`: "Cloud computing is revolutionizing IT. Companies are moving to the cloud."
- `stopwords.txt`: "is a the to are"

Output:

```
Document Table:
---------------
ID: 0 -> article1.txt
ID: 1 -> article2.txt

Term Table:
-----------
'computer' appears in: [0] (article1.txt (freq: 1))
'computing' appears in: [0, 1] (article1.txt (freq: 1), article2.txt (freq: 1))
'cloud' appears in: [1] (article2.txt (freq: 2))
'companies' appears in: [1] (article2.txt (freq: 1))
```

5.2 Example 2: Single Word Search

Input:

- Word: "computing"
- Weight: 1.5
- N: 2

Output:

```
Top 2 documents for 'computing' (weight: 1.5):
----------------------------------
Document: article1.txt, Score: 1.5
Document: article2.txt, Score: 1.5
```

5.3 Example 3: Word File Search

Input:

- WordFile: `query.txt` containing:

```
computing 1.5
cloud 2.0
```

- N: 2

Output:

```
Top 2 documents for query in 'query.txt':
-----------------------------------------
Document: article2.txt, Score: 5.5
Document: article1.txt, Score: 1.5
```

6 Notes and Limitations

1. The current implementation only indexes words that start with the letter 'c' (case insensitive)
2. The program uses a simple tokenization approach (words separated by spaces, punctuation removed)
3. All words are converted to lowercase for indexing and searching