Information Retrieval Assignment

By

Mohammed Almahsery	2232783
Mohannad Assaf	2233114
Khaled Saleh	2232890

Instructor: Dr. Zaher

Introduction

In this assignment, we were required to build a simple text-based information retrieval system. We managed to complete the assignment using several steps, which we will describe throughout this documentation.

The program is divided into two phases. The first phase involves preparing the corpus to construct the inverted index. The second phase involves constructing a Boolean retrieval system using the inverted index we created earlier.

Corpus

We searched about the corpus that we used in our system on the internet. The corpus we used is called “20 newsgroups”, It is a around 13000 newsgroups posts from various Usenet newsgroups on 20 distinct topics. These documents cover a wide range of topics, such as politics, science, sports, and technology. Fortunately, The scikit-learn module already includes the dataset in its 'datasets' library.

PHASE 1

We applied the two major components, tokenization and normalization for the corpus. Also, we constructed the inverted index. This was achieved by following several steps:

1. Importing Libraries

```
from sklearn.datasets import fetch_20newsgroups
from collections import defaultdict
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
import tkinter as tk
from tkinter import messagebox
import tqdm
import warnings
from tkinter import scrolledtext
warnings.filterwarnings('ignore')
corpus = fetch_20newsgroups()
```

Here, we imported the libraries we used in our product. It also includes a library for GUI to accept user's query and display the results for him.

2. Downloading Necessary NLTK Data Files

```
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

These packages are important for working with text in natural language processing (NLP). Downloading them gives the program the tools it needs to break down text into smaller parts and understand word meanings. For example, 'punkt' helps split text into sentences or words, while 'wordnet' acts like a big dictionary that groups similar words, provides short definitions, and shows usage examples. The 'omw-1.4' package extends WordNet to multiple languages, making it useful for processing text in different languages.

3. Importing Corpus

```
corpus = fetch_20newsgroups()
```

We imported the corpus from sklearn library.

4. Removing Punctuations from Documents

```
def remove_punctuations(text):
    result = ''

    for i, letter in enumerate(text):
        if letter in '<>#!$%^&*()+-/*\n\t:,][}{"\'':
            result += ' '
        elif letter == '.' and ( i == len(text) - 1 or text[i+1] in '\n\t '):
            continue
        else:
            result += letter

    return result
```

We are supposed to remove the punctuation from the document because it helps make the text simpler. Without punctuation, the words are easier to process and match, which improves search results and makes it clearer to understand the content. Also, it reduces the size of the index.

5. Stemming Words in Documents

```
def stemming(corpus):
    result = []
    stemmer = PorterStemmer()

    for file in corpus:
        stemmed_file = ''
        file = remove_punctuations(file)
        stemmed_words = [stemmer.stem(word) for word in file.split()]
        result.append(' '.join(stemmed_words))

    return result
```

```
def lemmatizing(corpus):
    result = []
    lemmatizer = WordNetLemmatizer()

    for file in corpus:
        lemmatized_file = ''
        file = remove_punctuations(file)
        lemmatized_words = [lemmatizer.lemmatize(word) for word in file.split()]
        lemmatized_file = ' '.join(lemmatized_words)
        result.append(lemmatized_file)

    return result
```

Now, we need to define two functions, one for stemming and the other for Lemmatizing. Stemming is the process of removing suffixes and prefixes from words to reduce them to their root form, often without considering the word's context or correct linguistic form. Lemmatizing is to reduce words to their base or dictionary form (lemma) by considering the context and morphological analysis, resulting in valid words.

6. Constructing Inverted Index

```
def indexing(corpus):
    index = dict()

    for file_index, text in enumerate(corpus):
        for word in text.split():
            if word in index.keys():
                index[word].add(file_index)
            else:
                index[word] = {file_index}

    return dict(sorted(index.items()))
```

Inverted index is the core of the information retrieval system. It stores corpus's words and their documents' files. We used python data-structure 'dictionary' to store it as key-value pairs, where the key is the word, and the value is a list containing word's documents. We created an index from the corpus. For each document, it splits the text into words and adds each word to a dictionary. The dictionary maps each word to a set of document indices where the word appears. If a word is already in the dictionary, the function updates the set with the current document index; otherwise, it creates a new set for that word. Finally, the function returns the dictionary, sorted by word.

7. Applying Previous Steps

```
#stemmed_corpus = stemming(corpus.data)
#index = indexing(stemmed_corpus)
## ------------------------------------
lemmatizing_corpus = lemmatizing(corpus.data)
index = indexing(lemmatizing_corpus)
```

Now there are two choices: either we choose stemming, which reduces time complexity but also decreases model accuracy, or we choose lemmatizing, which takes more time but provides more accurate results. We preferred accuracy over speed, so we chose lemmatization.

PHASE 2

Phase 2 is the stage of constructing Boolean retrieval system, which works as follows: taking user input, finding relevant documents, and displaying them to the user. This phase depends on the existence of the inverted index. We divided this phase into four steps:

1. Pre-processing Query

```
def preprocess_query(query):

    query = app.user_query
    query = remove_punctuations(query)
    #query = stemming(query.split())
    query = lemmatizing(query.split())

    return query
```

Everything done on the documents must also be done on the query. We defined a function that removes punctuation from the query and lemmatizes it using the same functions used for document preprocessing, to ensure that the search is more accurate and efficient.

2. Search index

```
def search_index(query):

    resulted_docs = []

    for word in query:
        if word in index.keys():
            resulted_docs.append(index[word])
        else:
            return []

    return resulted_docs
```

After applying preprocessing to the query, it is now ready for searching. We search the inverted index by iterating over the query and storing the keys and the indices of the documents where each key appears.

3. Merging Algorithm

```
def merge(files):

    if files == []:
        return None

    files = [sorted(list(word_file)) for word_file in files]
    files = sorted(files, key=len)

    final_results = files[0]

    for i in files[1:]:
        final_results = [element for element in final_results if element in i]

    return final_results
```

If we find the IDs of the relevant documents for each word we want, how can we find the IDs that contain all the words? Merging algorithms come in handy! Merging involves finding the intersection of the resulting documents using the merge algorithm. It has already been discussed in the lectures, so there is no need to go in depth.

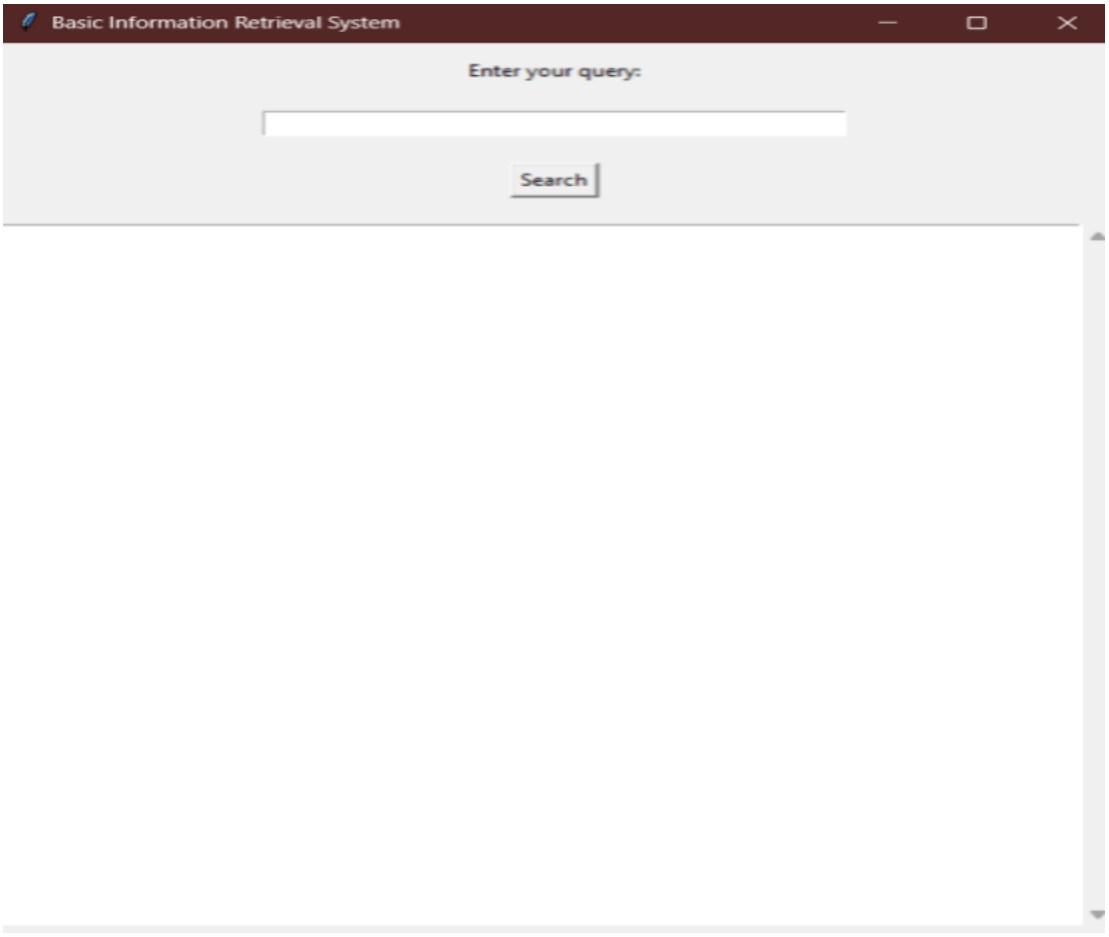
4. Graphical User Interface (GUI)

```
def GUI_result(docs):

    if docs == None or len(docs) == 0:
        return "No Relevant Documents.\n"

    string =  f'Document ID: '
    for i in docs:

        string += f'{i}, '

    string += '\nThank You For Using Our System.'

    return string
```

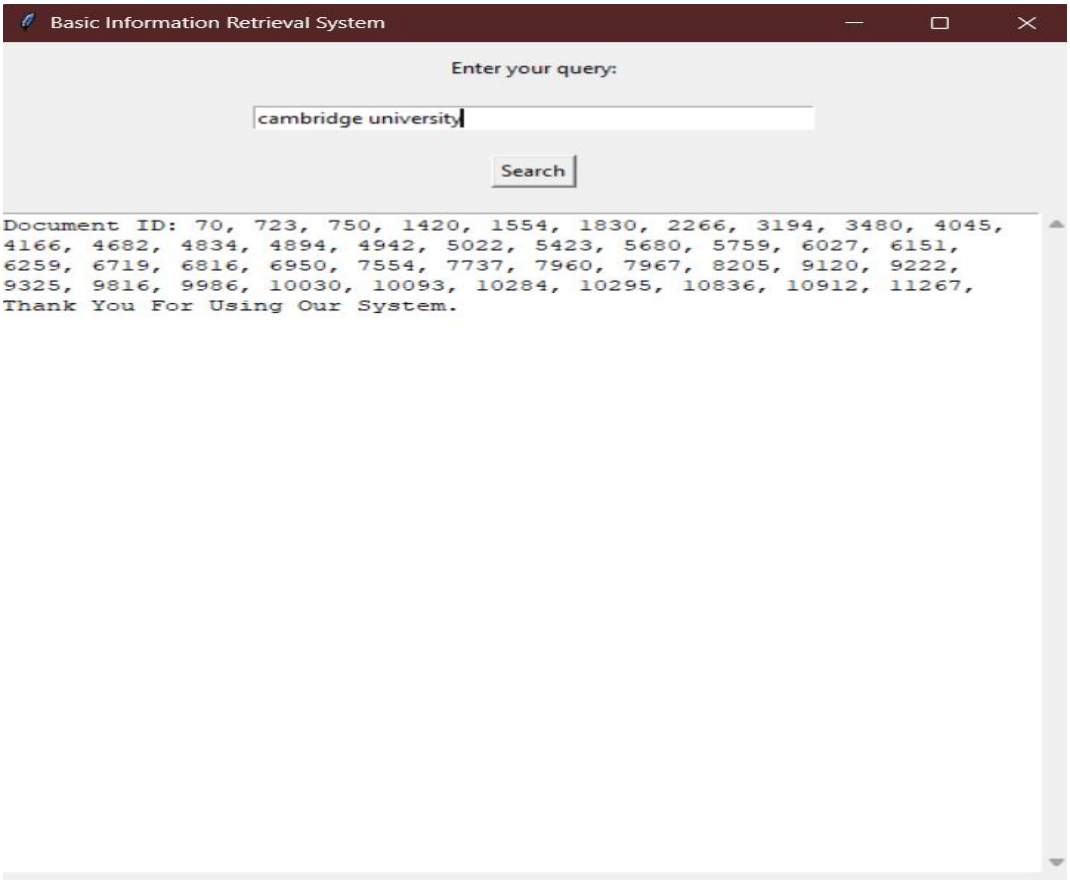
Before implementing the GUI, we must create a function that prepares the results for display to the user. This function checks if the query returned any relevant documents. If no relevant documents are found, the GUI displays a message informing the user that no relevant documents.

```
class BasicIRSystemGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Basic Information Retrieval System")

        self.label = tk.Label(root, text="Enter your query:")
        self.label.pack(pady=10)

        self.query_entry = tk.Entry(root, width=50)
        self.query_entry.pack(pady=10)

        self.search_button = tk.Button(root, text="Search", command=self.search)
        self.search_button.pack(pady=10)

        self.result_text = scrolledtext.ScrolledText(root, wrap=tk.WORD, width=70, height=40)
        self.result_text.pack(pady=10)


    def search(self):
        self.user_query = self.query_entry.get()

        self.result_text.delete(1.0, tk.END)  # Clear previous results
        self.result_text.insert(tk.INSERT, GUI_result(merge(search_index(preprocess_query(self.user_query)))))

root = tk.Tk()
app = BasicIRSystemGUI(root)
_ = root.mainloop()
```

Here we combine all the functionalities we created before into the GUI implementation of our IR system. When running the program, a window appears and asks the user to insert their query. The 'BasicIRSystemGUI' class includes the main parts: a window title, an entry box for typing queries, and a button to start the search. The GUI sends the query through the pre-processing step, the searching the index step, the merging results step, and finally the preparing the results step, all in order. After processing, the search results are shown in a scrollable text area, and the GUI displays "Thank You For Using Our System" at the end. The 'search' method gets the user’s query, processes it, searches for matches, and shows the results in the scrollable text area.



When you run the program, this window appears. You need to enter your query and press the ‘Search' button to see results.



For example, after entering a 'Cambridge university' the program displays the documents IDs that contain query words. If the results were too long, you can use the scroll bar on the right to scroll up and down. If you need to insert a new query, simply you can delete the old one, insert your new query, and hit the ‘Search' button.

Summary

First, we downloaded the necessary libraries and NLTK data files. Second, we started defining functions to apply to our query. We defined a function to remove punctuation and then applied lemmatization techniques to clean the data. Next, we began building the inverted index. After that, we defined a function to preprocess the queries using the functions we had already defined. We then created the search index function to search the inverted index and applied merging to find the intersecting documents. Lastly, we implemented the GUI to take queries from the user and display the results.

Thank You