Project Report

Information Retrieval Assignment-1

Group member information:-

Shashank Pratap Singh	2019B4A70956H
Satvik Omar	2019B4A70933H
Utkarsh Tiwari	2019B1A71147H
Gaurav Kumar	2019B3A71324H

Problem Statement

Retrieving a passage/paragraph from the sections like exclusions/inclusions for the documents (Any kind of documents like word/PDF/Image). Each policy document has multiple sections like inclusions, exclusions, conditions, definitions, extensions, covered sections, and so on. Each document must be extracted with its text information. From the policy documents, the section's (Already mentioned above) entire passage/paragraph needs to be retrieved depending on the query.

Introduction

The retrieval of specific information from documents can be a challenging task, particularly when dealing with complex documents such as policy documents. In these types of documents, there are often multiple sections that contain important information, and it can be time-consuming to search through them manually. This is where the need for an automated solution arises. The aim of our work is to present a solution for retrieving specific passages or paragraphs from documents based on user queries. The solution will involve extracting the text information from each document and then searching for the relevant sections based on user queries. This report will provide a detailed explanation of the solution's design, implementation, and testing, as well as its effectiveness in retrieving the desired information from policy documents.

Design and Implementation

The program is designed to extract data from multiple files and provide the paragraph and filename for any query executed. It also provides us with the number of results for the query, as well as the runtime for it. We used a variety of data structures to store and retrieve data efficiently, which will be later explained. The program starts by importing the necessary libraries, including os, ntlk, etc.

The program then proceeds to read the data from the files given, to store them for use later. It lists all the files in the given directory, and reads all of them one by one, extracting all the data from the files one by one. After extracting the data, it proceeds to split all of it by paragraphs, and stores them in a dictionary, which also consists of a unique id and the source file alongside each paragraph.

We then start the Tokenization process. The program iterates over the paragraph dictionary, and starts splitting each one of it by words. It also converts all of the words into lower case, so as to reduce the number of tokens we would have to store in the end. It then goes ahead and removes all the stop words from the words we just obtained. After that, we try to remove all of the duplicate words from the word list we currently have to get all of the unique words. After obtaining a list of unique words, we start with the stemming process. It is the process of producing morphological variants of a root/base word. To do so, we import the nltk library, to use the Portal Stemmer class.

Now, we move on to the query. Assume the user provided us with a query to be searched. The program goes on ahead and splits it into words, makes everything lower case, removes stop words and gets all of the unique words.Then, it spell checks the words, and does porter stemming on them. It then adds the connective ‘and’ between all of the query words, so as to do a bitwise AND operation on the results of all different words we obtained. Finally, it searches for the original paragraphs of the words, and prints them, along with the name of the original documents.

Data Structures

The following data structures were used in the program :-

1. **String:** Since we are working with text, string was a default choice. There are many inbuilt library methods in python which makes string manipulation easy and fast.
2. **Arrays:** We have used 2D array to store paragraphs and their corresponding document name. Because only purpose was to store paragraphs with a key, 2D array seemed simple and appropriate with row number being the key. At many other instances 2D and 1D vectors have been used as per convenience. Average insertion and access time is O(1).
3. **Dictionary:** We have used dictionary to map words to its corresponding list of paragraphs of which the word is a part. Using dictionary in this case makes it

faster to search for corresponding paragraphs. Average insertion time is O(n) and average access time is O(1).

Novelty/ Extra Features

We included a bunch of extra features, not mentioned in the problem statement, in our program. They are as follows:-

1. **Spelling Checker :** The program checks the spelling of the words in the query the user provides, and looks for the token that is the closest to the given word. It then provides the user with the passage/paragraph of origin the corrected spelling.
2. **Wild card queries :** The program can handle wild card queries as well, apart from exact word searches. When launching the program, the user can select whether they want to do a proper word search, or want it to work on a query with the wild card operator. The program handles both of them in different ways.
3. **Original document :** When providing the user with the passage/paragraphs where the given words originated from, the program also gives the name of the file where the passage/paragraph is originally from.

Libraries used

1. **os :** The OS module is a part of the standard library of Python. When imported, it lets the user interact with the native OS Python is currently running on. In simple terms, it provides an easy way for the user to interact with several os functions that come in handy in day to day programming.
2. **re :** The re module provides a set of powerful regular expression facilities, which allows us to quickly check whether a given string matches a given pattern, or contains such a pattern. It is used here to help with the matching of wild card queries.
3. **ntlk :** NLTK, or Natural Language Toolkit, is a Python package that we can use for NLP. A lot of the data that we could be analyzing is unstructured data and contains human-readable text. Before we can analyze that data programmatically, we first need to preprocess it, which we can do with the help of this library. The program uses this library to remove stop words, and do porter stemming on the tokens/queries.

4. **glob :** The glob module finds all the pathnames matching a specified pattern. It is used to list all of the files available for reading in the specified path.
5. **chardet :** It is the Universal Character Encoding Detector. It's used to help with the data extraction of the files, as they can contain special characters that cannot be read without the specific encoding.

Recall and Precision

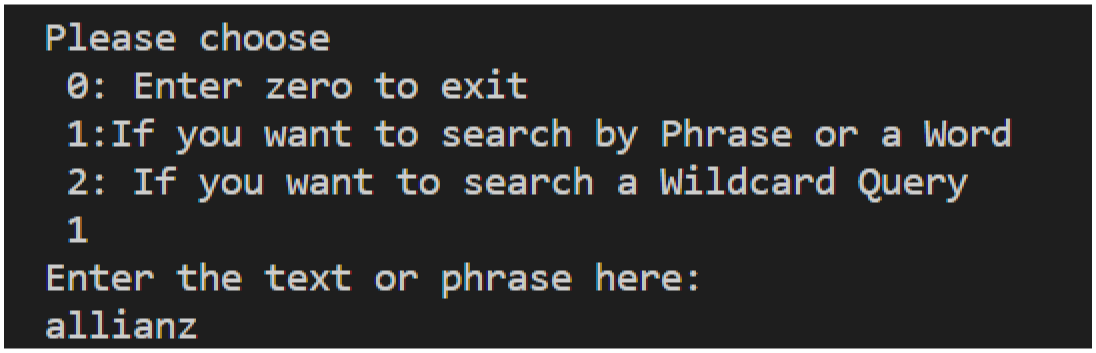
Precision and recall are the measures used to measure how well an information retrieval system retrieves the relevant documents requested by a user. The measures are defined as follows:

Precision = Total number of documents retrieved that are relevant/Total number of documents that are retrieved.

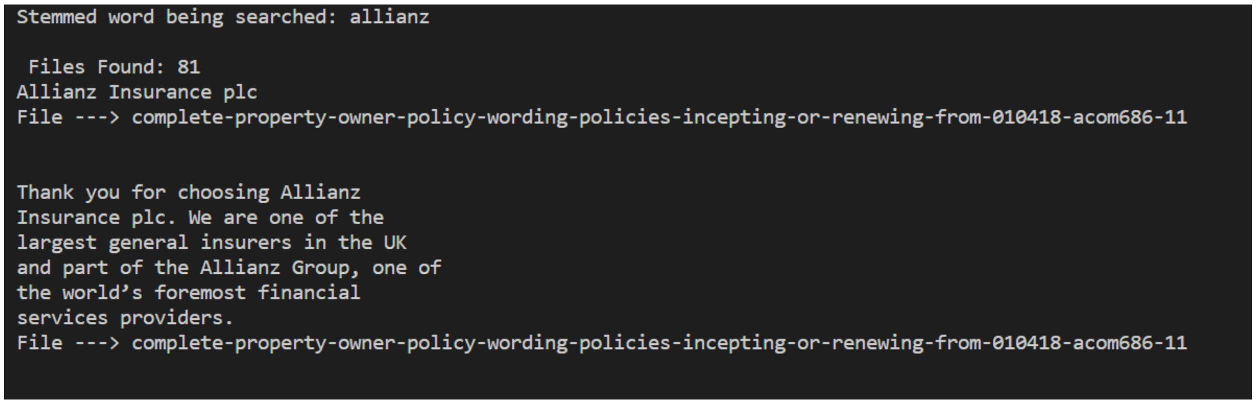
Recall = Total number of documents retrieved that are relevant/Total number of relevant documents in the database.

Results

1. Example for a simple word search:

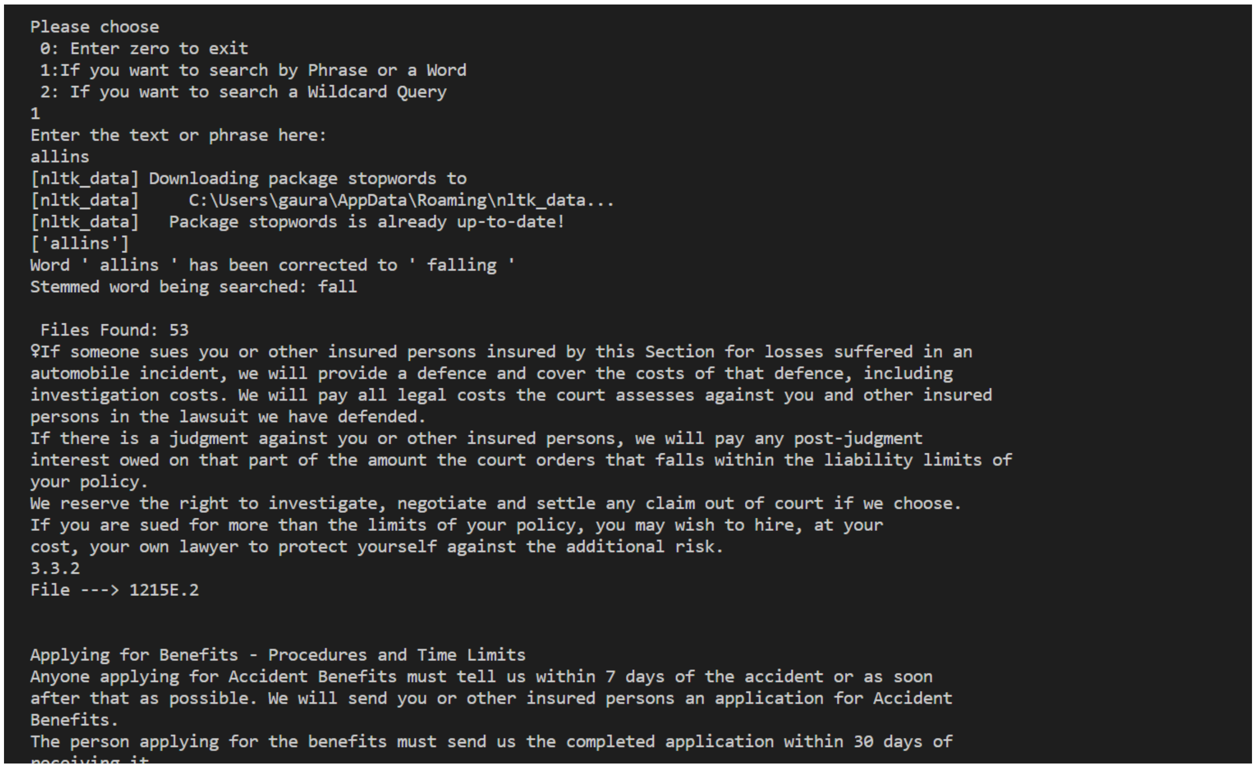


```
Please choose
 0: Enter zero to exit
 1:If you want to search by Phrase or a Word
 2: If you want to search a Wildcard Query
 1
Enter the text or phrase here:
allianz
```



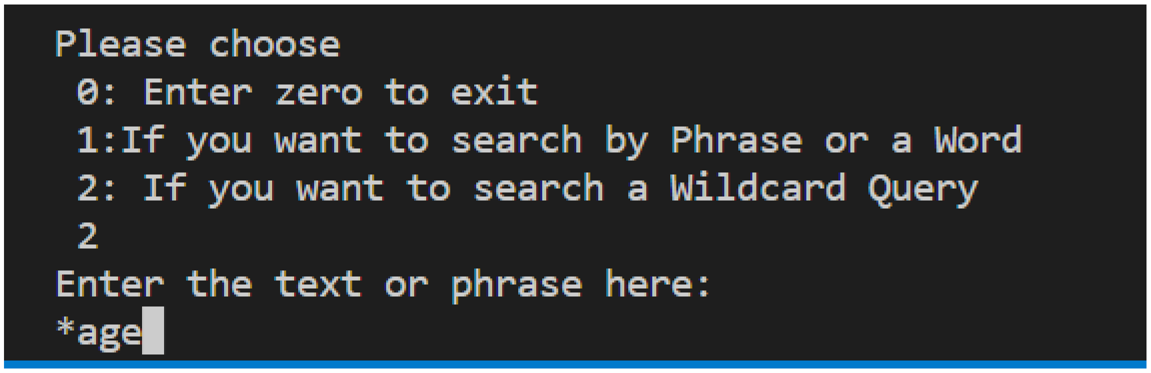
```
Stemmed word being searched: allianz

 Files Found: 81
Allianz Insurance plc
File ---> complete-property-owner-policy-wording-policies-incepting-or-renewing-from-010418-acom686-11


Thank you for choosing Allianz
Insurance plc. We are one of the
largest general insurers in the UK
and part of the Allianz Group, one of
the world’s foremost financial
services providers.
File ---> complete-property-owner-policy-wording-policies-incepting-or-renewing-from-010418-acom686-11
```

2. Example of spell-check feature with an incorrect word which gets corrected to a similar word: In this example case the word “allins” has been corrected to “falling”.

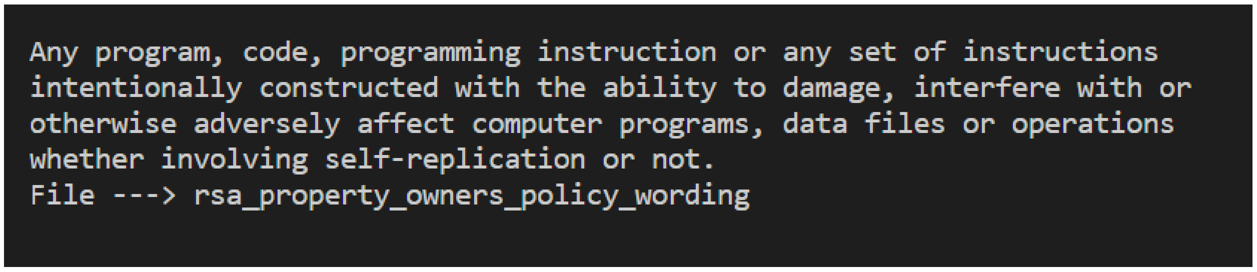


```
Please choose
 0: Enter zero to exit
 1:If you want to search by Phrase or a Word
 2: If you want to search a Wildcard Query
1
Enter the text or phrase here:
allins
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\gaura\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
['allins']
Word ' allins ' has been corrected to ' falling '
Stemmed word being searched: fall

 Files Found: 53
♀If someone sues you or other insured persons insured by this Section for losses suffered in an
automobile incident, we will provide a defence and cover the costs of that defence, including
investigation costs. We will pay all legal costs the court assesses against you and other insured
persons in the lawsuit we have defended.
If there is a judgment against you or other insured persons, we will pay any post-judgment
interest owed on that part of the amount the court orders that falls within the liability limits of
your policy.
We reserve the right to investigate, negotiate and settle any claim out of court if we choose.
If you are sued for more than the limits of your policy, you may wish to hire, at your
cost, your own lawyer to protect yourself against the additional risk.
3.3.2
File ---> 1215E.2


Applying for Benefits - Procedures and Time Limits
Anyone applying for Accident Benefits must tell us within 7 days of the accident or as soon
after that as possible. We will send you or other insured persons an application for Accident
Benefits.
The person applying for the benefits must send us the completed application within 30 days of
```

3. Example with wildcard expression: In this example case, a sample paragraph is shown which contains the word “damage” which comes out from the wildcard input “*age”. We can also use symbols like ‘?’ to enhance our search query this way.



```
Please choose
 0: Enter zero to exit
 1:If you want to search by Phrase or a Word
 2: If you want to search a Wildcard Query
 2
Enter the text or phrase here:
*age
```



```
Any program, code, programming instruction or any set of instructions
intentionally constructed with the ability to damage, interfere with or
otherwise adversely affect computer programs, data files or operations
whether involving self-replication or not.
File ---> rsa_property_owners_policy_wording
```