Contents

[Pre-requisites 2](#_Toc6394726)

[Hardware Requirements 2](#_Toc6394727)

[Software Requirements 2](#_Toc6394728)

[File Configuration 2](#_Toc6394729)

[Information for Program Execution 3](#_Toc6394730)

[Corpora setup 3](#_Toc6394731)

[Programs, Order of Run and runtime Inputs 3](#_Toc6394732)

[Post-Run 4](#_Toc6394733)

[Comparison of Design proposed for the product vs Features implemented in this project 4](#_Toc6394734)

[User interface 4](#_Toc6394735)

[Natural Language Interpreter 4](#_Toc6394736)

[Search, Ranking and Download 5](#_Toc6394737)

[Feedback 5](#_Toc6394738)

[Future Enhancements and ideas 5](#_Toc6394739)

[Implementation of the originally proposed features 5](#_Toc6394740)

[A feedback system for the CRA 6](#_Toc6394741)

[A cognizance measure for the system 6](#_Toc6394742)

[Building Temporal user-specific Context 6](#_Toc6394743)

[References and Citations 7](#_Toc6394744)

# Pre-requisites

### Hardware Requirements

The following minimum hardware requirements are to be met –

**Memory** : 2 GB RAM or more

**Disk Space**: 50 MB or more

### Software Requirements

The following software is required –

Windows 10 OS (Home or Pro)

Anaconda version 1.9.6 or later

Jupyter Notebook version 5.7.8 or later

Python version v3.7 or later

Python packages required to be installed (outside of what is shipped with Anaconda) –

NLTK

SpaCy

WordNet for NLTK

PyPDF2

### File Configuration

The following paths need to be set up at an accessible location on disk –

**Mandatory:**

Library for Temporary Files – C:\CRA\Temp or similar

Library for Corpus – C:\CRA\Corpus or similar

**Recommended:**

Library for Results – C:\CRA\Results or similar

# Information for Program Execution

Once the necessary software has been installed, please perform the following steps.

### Corpora setup

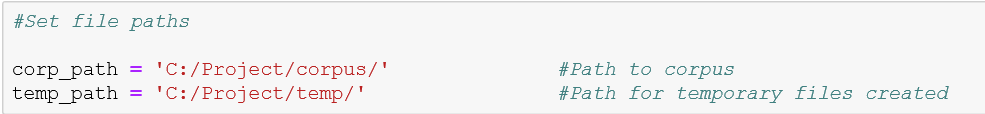
To setup the corpus which the program will be using as the search repository, please download/create PDF or TXT files on the topic of your choice. At this time, only these two file formats are supported. All documents need to be present in the Corpus library at the time of running.

### Programs, Order of Run and runtime Inputs

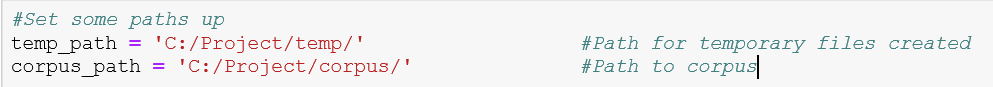
There are two Jupyter notebooks which were created as a part of this project –

* Preprocessed Document Library.ipynb
* The CRA.ipynb

The Preprocessed Document Library notebook should be run first as it would create a consolidate search table for the CRA to use. This notebook creates the search repository in the form of a .pkl file which is subsequently imported and read by the CRA to perform its search. There are two manual inputs for this notebook, the following paths must be changed to the paths which were set up earlier, before the notebook can be run. –



The CRA notebook should be run next. This notebook has 4 manual inputs - two static and two real-time. The following input needs to match the paths used in the Preprocessed Document Library notebook.

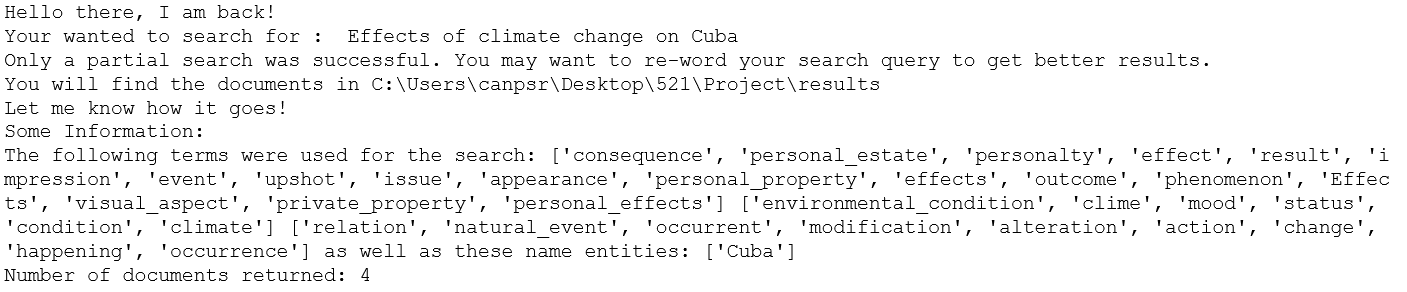


It would ask for two real-time inputs – 1. The search term and 2. The path where the results need to be output. If the path given doesn’t exist, the notebook will throw an error message and ask to re-enter path as many times as required until a valid path is entered. The results directory created earlier during configuration is recommended.

**NOTE:** After entering each input, please hit the Enter key.

# Post-Run

After the CRA notebook has run, a message will be displayed with the link to search results if any, the search terms used and the number of documents returned. You will find the search results in the results directory provided as input. A sample is provided below.



# Comparison of Design proposed for the product vs Features implemented in this project

### User interface

The User interface was implemented as a simple form-based input system using Jupyter Notebooks’ in-built real-time input integration feature. The backend is implemented in Python.

The request for input is presented as a box where the input needs to be typed followed by pressing the ‘Enter’ key.

The proposed product design in contrast is a dedicated web page. A full-featured product would have both a web interface as well as a Mobile UI. The backend will be in Python or a similar program with the UI interfacing with the backend using web server technologies.

### Natural Language Interpreter

The natural language interpreter used were the SpaCy and NLTK libraries available for Natural Language processing for Python. Python was the original choice of programming language for the backend. This was maintained and carried through.

### Search, Ranking and Download

The original proposition was to search the internet via web scraping and download content based on relevance. However due to last minute time constraints, the decision to use a locally stored document repository was made. The search was performed manually, the documents were downloaded into the folder designated to house the corpora. Similarities of the content within the documents were assessed with the terms in the search query provided by the user and documents containing most of the same terms or synonyms were determined to be the most likely candidates for download. No ranking of the documents was performed.

Instead of download, relevant documents identified during processing will be copied from the corpora to the results folder.

The full-featured product would perform a search on the internet, identify content, process it for relevance and similarity and download the content where allowed and applicable. Ranking of content will be performed after the content has been downloaded. Ranking will be relevant only when presenting results to the user, so setting the system up this way would save some time.

### Feedback

Owing to time constraints, no feedback was implemented in this project.

The feedback system is essential for the system’s performance but not a mandatory component to implement usable functionality. However, since this is the design of a Cognitive Assistant, this feature should be given equal importance as other components. A detailed description of the idea of a feedback system is presented in the section titled ‘Future Enhancements and ideas’.

# Future Enhancements and ideas

### Implementation of the originally proposed features

One of the first future enhancements to be made would be to implement all the originally proposed features under each module – building a well-designed UI and integrating it with Python as well as building the web search and relevance ranking modules.

In addition, a functionality to score the documents based on context can be included. This will be different from ranking since the scoring will be based on real-world semantic context whereas the ranking module is ranked based on relevance to the search initiated by the user.

### A feedback system for the CRA

The feedback system for the CRA would be accessible through the User Interface and its input will be completely user-provided.

The user will be asked to rank the documents returned in order of relevance to them.

The algorithm would have produced its own score for terms in the document when it initially produced its result. The terms will be ranked by score and the first-n highest ranking terms will stored along with the search term.

After the user gives feedback, the algorithm will produce scores for terms from the documents ranked in the feedback and compare it against what it initially produced. If they are very different, it will produce a score indicative of its own performance.

Based on whether its performance score is high or low, it will broaden its search and/or use alternative keyword searches to produce future search results. The single important goal for the algorithm is maximization of its performance score. It will also associate the search term with these new terms and scores. It will retain any terms in common. The structure thus built can act as a DB against which future questions from the user will be first checked, thus making the system more optimal with respect to performance.

### A cognizance measure for the system

The system can maintain a measure of its state of knowledge or cognizance of the world. It can start from some pre-determined low value. As it returns searches, builds semantic contexts and received feedback regarding performance, it can incorporate these as factors into the cognizance measure to indicate what its current level of knowledge (according to itself is). It is after all, how humans think of themselves!

If the system attains a high cognizance measure, it can consider itself an expert and ‘worthy’ enough to teach other systems what it knows. Let’s call this paradigm Machine teaching! (Because why not?!)

### Building Temporal user-specific Context

This would be a glorified version of Autofill. Based on the searches the user performs regularly, the system can build and drop temporary contexts of the user’s current environment, goals, preferences etc. This can feed into relevance ranking. For instance, if the user has been searching for destinations, flights and hotels for the past month, and if the current search is for ‘good’ cars, it is probably the user looking for a comfortable, rental car for a road trip. The system could infer this based on general experience (its cognizance) and the temporal context (searches in the past month) it has built. If the user doesn’t make any more searches for the next 10 weeks, the temporal context can be dropped.

# References and Citations

Cambria, E., & White, B. (2014). Jumping NLP curves: A review of natural language processing research. *IEEE Computational intelligence magazine*, *9*(2), 48-57.

Bird, Steven, Edward Loper and Ewan Klein (2009).Natural Language Processing with Python. *O'Reilly Media Inc.*

|  |
| --- |
| @ARTICLE{spacy2, |
|  | AUTHOR = {Honnibal, Matthew AND Montani, Ines}, |
|  | TITLE = {spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing}, |
|  | YEAR = {2017}, |
|  | JOURNAL = {To appear} |
|  | } |