# A Web Opinion Visualiser

**Course**: B. Sc. Software Development – Artificial Intelligence (2020)
**Submitted by** Justin Servis

## Specification Outline

Develop a multithreaded AI search application that can generate a word cloud from the top 20 words associated with an internet search term.

- Use the JSoup API to parse the results returned from the DuckDuckGo search engine for a given term and goal threshold and identify a set of candidate child nodes.
- Use an AI search or searches to follow the hyperlinks contained in the HTML page of search results, parse these pages and index accordingly.
- Generate a word cloud from the most frequently occurring words in the index. The maximum number of words to display should be selectable by the user.
- The application must be threaded.

## User Interface

From the web interface, the user has several options to choose from:

- Choose to search either DuckDuckGo or Wikipedia. The Wikipedia search is not refined and so will result in some odd results. This is as a result of the initial search page not being parsed in an optimal way and it would be best to create a specific implementation of IDocumentSelector to parse correctly, as has been done with DuckDuckGoDocumentSelector.
- Choose to use either Best First or Beam search.
- Max Results – the maximum number of results to display in the word cloud.
- Max Depth – the maximum depth for the search algorithm to navigate to.
- Beam Width – only used for beam search, the width of the beam to use when selecting best results.
- Threshold – the maximum, inclusive score to accept for a scored document.
- Polling Time – number of seconds between each page refresh until a result is received.
- Enter Text – enter the query text here and click Search and Visualise to run the search.

## Architecture

There is extensive use of interfaces throughout which make the application very adaptable to new types of searches, support for parsing search results from different sites and types of documents. With a small bit more abstraction, it is possible to remove the reliance on JSoup as the parser and switch between different parsers at runtime but I feel that this was beyond the scope of the project.

Additionally, I have created a couple of Map interfaces (IFrequencyMap, IMergeableMap) to allow merging of frequency maps together. Interfaces have also been used to define the Indexable and Scorable behaviours required by the heuristic scoring and result generation systems.

## Submitting a Query

Once the user has submitted a query via the web interface, a Runnable WebSearchTaskWorker instance is created to handle the query. Each instance handles a single query and dies thereafter.

## Search Implementations

Two search algorithms can be used which are implementations of the ISearch interface - BestFirstSearch and BeamSearch. The default search used to generate the word cloud is the Best First Search, but the user may select Beam Search from the dropdown before submitting the query.

Each of the two search algorithms use a FrequencyMap to hold the search results. FrequencyMap uses a ConcurrentSkipListMap as its backing data structure and implements the IMergeableMap<E> interface, which allows for the merging of one FrequencyMap to another - combining the results of each map together. In this way, many threads can simply merge their results into this central data structure and the final contents are placed in a list, sorted and trimmed to the desired size (maximum results).

### *Best First Search*

A standard Best First Search implemented using a PriorityQueue and comparator to prioritise each returned IScorableDocument instance by its getCombinedScore() method. A depth variable is user selectable which determines how deep to go with each search.

### *Beam Search*

A modified Beam Search algorithm which is implemented using a ConcurrentLinkedQueue. This search is slower but a lot more selective than the Best First Search since it only accepts the highest scoring links from every other link, until the maximum depth is reached.

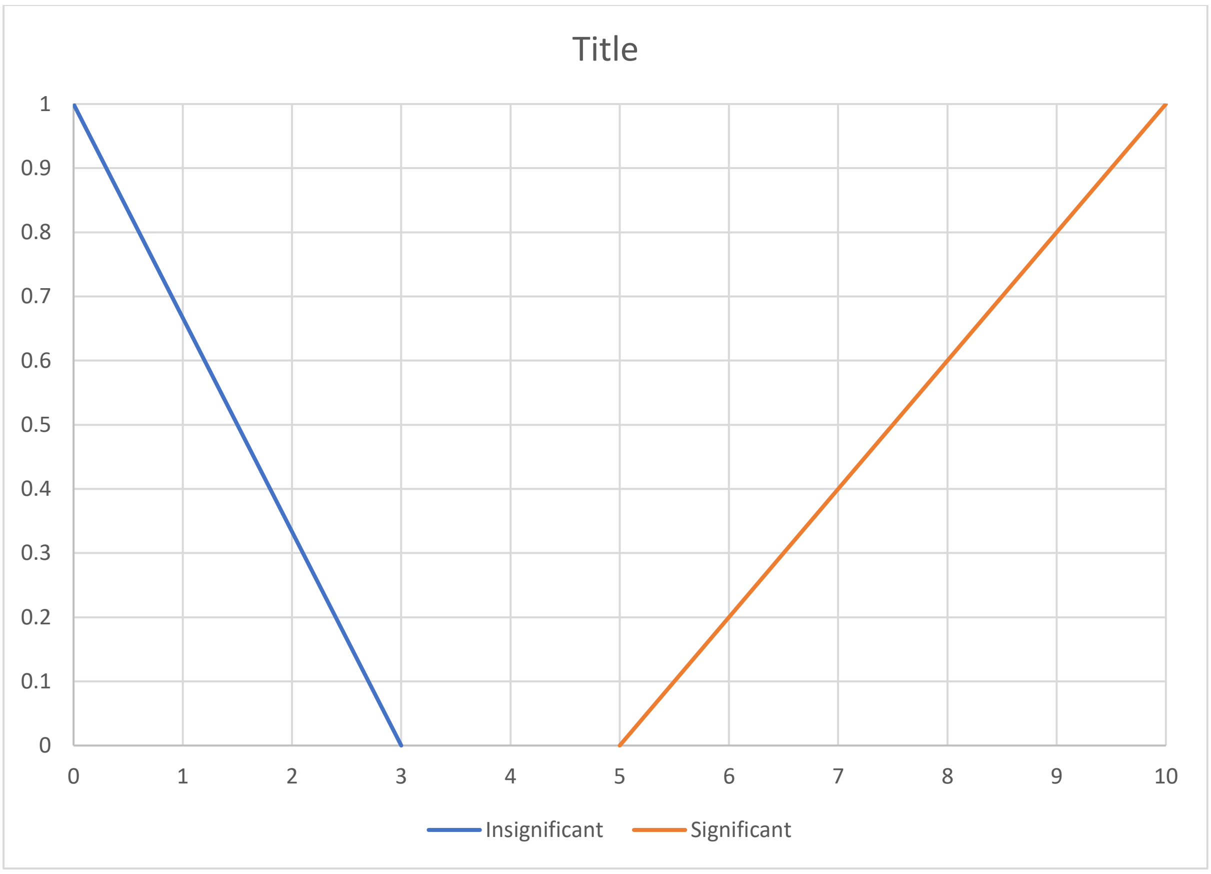
## Fuzzy Logic

Several different classifications have been used to implement the fuzzy logic within the application.

It would have been possible to create more classifications of input variables, for example, for H3 and H4 tags; tables; lists; etc. From testing the application with the final set, I found that the results were quite satisfactory and so kept it the final three.

### Title Input Variable

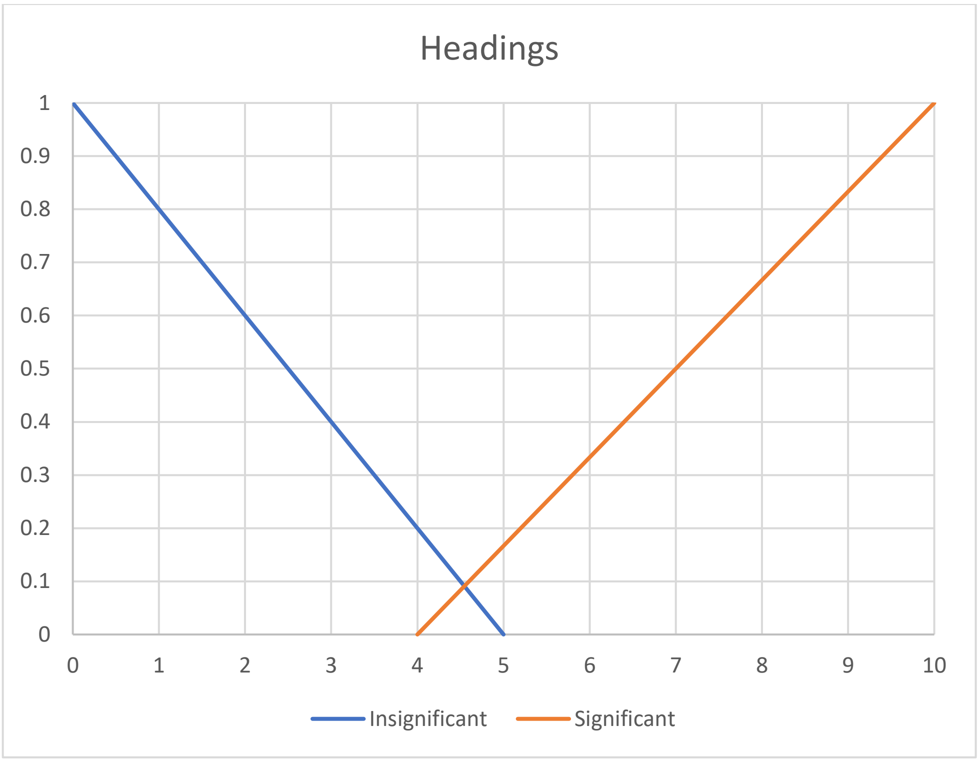
Only the upper end is used to classify titles since the existing search only returns whether the query was present in the title. The original intent was to introduce a ‘relevant’ term (like the body variable) in addition to ‘significant’ and ‘insignificant’ and use a distance measure to score the title, e.g. if search term was Ireland but title contained Ireland’s, or similar.



### *Headings Input Variable*

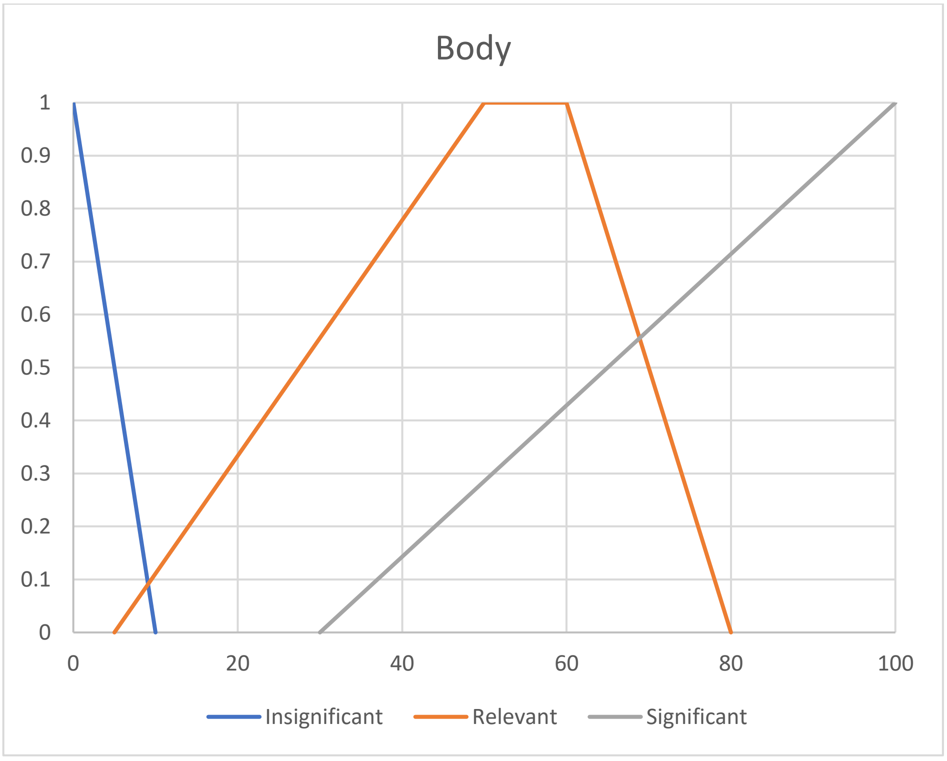
Like the title variable, I had also intended on using a ‘relevant’ term here. I had planned on scoring each heading tag related to its value, e.g. H1 scoring more than H2 and so on. I found from testing that headings are not that entirely a great of a marker for the relevance of a document and so I omitted this functionality from the application in the hope that it would not become overly complex.

I settled on using the count and a simple membership function as depicted below.



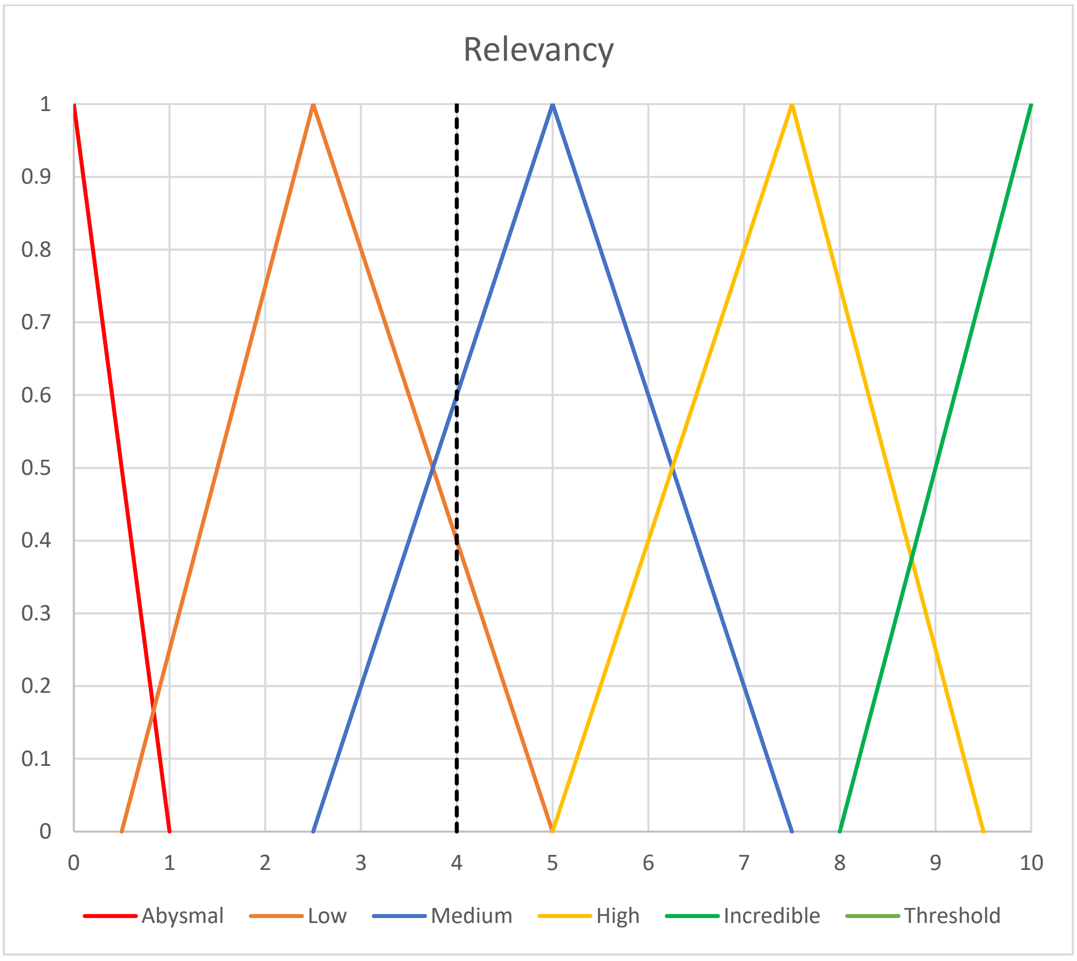
*Body Input Variable*

For the body variable, I used a more complex membership function and introduced a 'relevant' term to increase the precision of the expected results of the fuzzy logic. I found that the values depicted in the membership chart below worked well enough that I settled on them for the final application.



### *Relevancy Output Variable*

For the output variable which I have called 'relevancy', I expanded the terms to 5 as depicted below. From testing, I found that 3 was simply too narrow with the results that it returned, and I also wanted to expand the ruleset to remedy this. The following is the final membership chart for relevancy which is used in the application.



### *Ruleset*

Following is the set of final rules used in the application

- RULE 1 : IF title IS significant AND headings IS significant AND body IS significant THEN relevancy IS incredible;
- RULE 2 : IF title IS significant OR headings IS significant AND body IS significant THEN relevancy IS incredible;
- RULE 3 : IF title IS significant AND headings IS significant OR body is significant THEN relevancy IS high;
- RULE 4 : IF title IS significant OR headings IS significant AND body IS relevant THEN relevancy IS high;
- RULE 5 : IF title IS significant OR headings IS significant AND body IS relevant THEN relevancy IS medium;
- RULE 6 : IF title IS insignificant AND headings IS significant AND body is relevant THEN relevancy IS medium;

- RULE 7 : IF title is insignificant OR headings IS insignificant AND body is relevant THEN relevancy IS low;
- RULE 8 : IF title is significant AND headings IS insignificant AND body is insignificant THEN relevancy IS low;
- RULE 9: IF title is insignificant AND headings IS insignificant AND body IS insignificant THEN relevancy IS abysmal;