**IMPLEMENTATION**

Implementation is the stage in the project where the theoretical design is turned into a working system. The most critical stage is achieving a successful system and in giving confidence on new system for the users, that it will work efficient and effectively.

It involves careful planning, investigating the current system, and its constraints on implementation, design of methods to achieve the changeover, an evaluation of changeover methods.

The implementation process started with preparing a plan for the implementation of the system. According to this plan, discussion has been made regarding the equipment, resources and test activities to be performed. Thus, a clear plan was prepared for activities.

**7.1 Module Implementation and related concepts**

**7.1.1 Python**

Python is a widely used high-level programming language for general-purpose programming. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly braces or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library. Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems. The version used here is 3.6.5.

**7.1.2 re**

The module re provides full support for Perl-like regular expressions in Python.

A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. Regular expressions are widely used in UNIX world. Both patterns and strings to be searched can be Unicode strings as well as 8-bit strings. A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let you check if a particular string matches a given regular expression (or if a given regular expression matches a particular string, which comes down to the same thing). The regular expression language is relatively small and restricted, so not all possible string processing tasks can be done using regular expressions. There are also tasks that can be done with regular expressions, but the expressions turn out to be very complicated. In these cases, it is better off writing Python code to do the processing. Even though Python code will be slower than an elaborate regular expression, it will also probably be more understandable. In the project, both approaches have been used.

**7.1.3 urllib.request**

A Python module for fetching URLs is urllib.request. It offers a very simple interface, in the form of the urlopen function. This is capable of fetching URLs using a variety of different protocols. It also offers a slightly more complex interface for handling common situations like basic authentication, cookies, and proxies and so on. These are provided by objects called handlers and openers. It supports fetching URLs for many schemes of URL using their associated network protocols like FTP or HTTP.

**7.1.4 urllib.error**

The urllib.error module defines the exception classes for exceptions raised by urllib.request. The base exception class is URLError. urlopen raises URLError when it cannot handle a response. HTTPError is the subclass of URLError raised in the specific case of HTTP URLs. The HTTPError instance raised will have an integer code attribute, which corresponds to the error sent by the server. The HTTPError instance can be used as a response on the page returned.

**7.1.5 NumPy**

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. Matrix and vector manipulations are extremely important for scientific computations.

NumPy is an acronym for "Numeric Python" or "Numerical Python". It is the fundamental package for scientific computing with Python. It has an intuitive syntax and high-performance matrix computation capabilities. NumPy is most suitable for performing basic numerical computations such as mean, median, range, etc. It can also be used to integrate C/C++ and FORTRAN code.

**7.1.6 operator**

The operator module exports a set of efficient functions corresponding to the intrinsic operators of Python. This means that the module provides a functional interface to the standard operators in Python. The functions in this module can be used instead of some lambda constructs, when processing data with functions like map and filter. The operator module defines functions that correspond to built-in operations for arithmetic and comparison, as well as sequence and dictionary operations.

**7.1.7 string**

The string module provides additional tools to manipulate strings. It contains a number of useful constants and classes, as well as some deprecated legacy functions that are also available as methods on strings. In addition, Python’s built-in string classes support the sequence type methods like str, bytes, bytearray, list, tuple, range section, and also the string-specific methods like capitalize, center and so on.

**7.2 Algorithms Used**

**7.2.1 TextRank**

TextRank algorithm is used for extractive summarization. It entails selecting the X most representative sentences that best cover the whole information expressed by the original text.

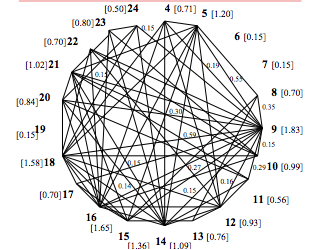
The basic idea of TextRank is to provide a score for each sentence in a text then you can take up the top-n sentences and sort them as they appear in the text to build an automatic summary.

The first step is to extract all the sentences from the text. This can be as simple as just splitting the text at full stops, newlines or more complex.

After this an undirected weighted graph is constructed with every node connected to one another via a weighted edge. The nodes of the graph consist of sentences. The weight of each edges derives their value from similarity, a measure of their content overlap. In this case, content overlap is the number of words that both sentences have in common divided by the sum of the logs of the total number of words in each sentence (so longer sentences don’t completely dominate shorter sentences). The similarity measure used most often is cosine similarity.

Similarity based on content overlap in general can be described by the following formula.

After this the weights of the nodes has to be calculated. Include random numbers as initial weights on the graph nodes. Any range of numbers could work, however the paper recommends 0–10. After this the graph between the sentences looks similar to this.



The graph is made up of sentences with random weights connected to one another by edges that have weights based off their similarity. The TextRank formula is run over this.

The weight (W) of the sentence we wish to re rank (Vi) is equal to one minus the dampening factor (d), .85. Dampening factor is a value that ensures that the weights of their nodes eventually settle down to a single value. The dampening factor can be anything from 0–1, however .85 is the commonly accepted value.

For two sentences (i, j) the weight between the two of them (wij) is divided by the sum of every weight(w) between the sentence(j) and every sentence that it connects to (wjk) this value is then multiplied by the weight of sentence (j). The previous process is then repeated for every sentence (Vj) that is connected into the sentence that we are re ranking (Vi). Those values are then summed and multiplied by the dampening factor (.85). This value is added to previous calculated value and new weight is calculated.

The calculation must then be repeated for every sentence. The weight of each sentence is now significantly closer to what their true weight should be, but to ensure that the rank has been recalculated enough times, TextRank utilizes an acceptable error rate of .0001. This error rate is calculated by simply subtracting the weights of a sentence before and after every sentence has been re ranked. Once the weight changes by less than .0001, the weights are now close enough to their true weights to stop the algorithm.

Inside the sentence class there’s the tokenized words, but that’s completely unreadable. So for each sentence that’s constructed the original sentence is added to the class, and now when the sentence class is sent to be output, the full original sentence is printed instead. Next the sentences are sorted and the top few sentences are output as the summary.

**7.3 Steps involved in Implementation**

**7.3.1 Getting links from the Web Page**

**7.3.1.1 Convert from keyword to link**

A simple code in Python written to convert the keyword input to a format which can be parsed by the scraping modules. It takes a string, which is the keyword entered. Any Wikipedia article is stored in the format https://www.wikipedia.org/keyword\_keyword/. This format is generated by the function for ease of use. The code splits the keyword into tokens and appends it to the required substrings to create a string which is the URL format, which is returned to main function. This is further sent to the main function so that the Wikipedia data can be extracted out of it.

**set** s **as** string '/wiki/'

**set** tokens **as** list of tokens in keyword

**loop through** all tokens

**if not** the first word

**append** word to s

**append** ‘\_’ to s

**return** s

**7.3.1.2 Get useful hyperlinks to get prerequisites**

This function is written in python to get all the hyperlinks in the current document from the corresponding Wikipedia page. It takes one parameter, the URL of the page in which it has to extract the data. It uses the urlopen and beautifulsoup modules for this purpose. urlopen acts as a socket and gets the data, while beautifulsoup parses the HTML content in the page and gets the required hyperlinks using regular expressions. Further cleaning up is done and a list containing the hyperlinks is returned.

**set** bsObj **as** BeautifulSoup Object

**set** newLinks **as** list of links

**loop through each** tag **in** bsObj content

**if** tag is ‘bodycontent’

**if** tag starts with ‘/wiki/’

**if** tag does not contain semicolons

clean the link of unwanted text

**append** link to newLinks

**return** newLinks

**7.3.2 Extractive Summarization**

**7.3.2.1 Extract the text data**

The data from the Wikipedia page is extracted using this Python function. It takes the keyword as the parameter. Using the summary function of Wikipedia API, it gets the required corpus from the web page and return it to the main function.

**set** text **as** summary of article

**return** text

**7.3.2.2 Summarization using TextRank**

The corpus obtained is to be summarized to obtain the prerequisites. This is done using TextRank algorithm. The main function used to obtain the summary uses many other sub functions as well. Initially the text is processed, as in, white spaces, invalid characters and any undesirable elements in the corpus are removed. Next a similarity matrix is constructed as a step in TextRank algorithm. Cosine similarity is used to do this. The number of words that both sentences have in common divided by the sum of the logs of the total number of words in each sentence can be used as a measure of similarity between the sentences.

Next, for every sentence, the relevance of the sentence with respect to the entire text is calculated. The logic behind this is that, if the sentence to be ranked has a high similarity to that of the entire text, then it is of higher relevance to the document. This is done for every sentence in the document. Once all the sentences are ranked and put into a list. Once all the sentences are ranked, they are sorted based on the ranking. The sorted list is returned. Top few sentences are the summary of the document.

**set** sentences **as**  processed corpus

**set** length **as**  length of sentences

**create** S **as** a Similarity Matrix

**set** summary **as** summary of document

**call** TextRank **on** sentences, S

**loop through** TextRank data

**append** sentences **to** summary

**return** summary