CE706 - Information Retrieval
Assignment 1: Indexing for Web Search

Dilpa Rao (1906319), Viraj Kumar Dewangan (1901181)

February 21, 2020

1 Introduction

The following report describes building a system that turns a web site into structured knowledge. This system takes HTML pages as input, process them using the following techniques: HTML Parsing, Sentence Splitting, Tokenization, Normalization, sentence breaking, script segmentation, encoding and language detection, natural language processing and outputs an index of terms identified in the documents. We used **Python notebook** environment to perform this system.

The work was organized as follows -

1. We read the web pages provided and Before the text could be analysed the HTML tags were removed without deleting the meta tag keywords. The result is plain text now.

2. Second, text was pre-processed (Sentence Splitting, Tokenization and Normalization) and

3. Third, Part of Speech tagging was done to identify entities.

4. Fourth, we removed words that are not useful and did TF-IDF computation.

5. Finally, we created .txt files for each step of our text analysis and indexing for web search.

2 System Development

The following URLs were used to develop the system:

URL1 = “http://www.multimediaeval.org/mediaeval2019/memorability/”

URL2 = “https://sites.google.com/view/siirh2020/”

2.1 *URL reading and processing the URL for parsing -*

All websites raw information are contained in HTML format. For that reason, the text is needed to be pre-processed to remove all the HTML tags present in the raw data. We took urllib.request and beautiful soup libraries to perform text reading and cleansing. Beautiful soup is a python library that pulls data out of HTML and XML files.[1] This library helps transform a complex HTML document into a complex tree of Python objects that aids to get insights about the text we analysed. In this case we used it to remove the HTML and tabulations and after that we extract the links that are in the URLs.

2.2 *Meta Data Extraction and storing it in the file -*

Metadata is present for each website, which is generally the summary information or the abstract information about each website. Some examples are author, date created, date modified, information about the files included, and for that reason when we filter through metadata the location of specific documents is much more easier. In this case, from each website we extract its own meta data information present. We have extracted the metadata information for the two links provided using Beautiful Soup and HTML Parser and wrote that into a text file "Meta_tags.txt"

2.3 *Sentence Splitting, Tokenization, Normalization and Part-of-Speech Tagging –*

On a website, most of the information is unique when we take a closer look into words rather than in phrases. In this section we wanted to perform tokenization, to do that we used the NLTK tool. With this tool we splitted the text into sentences and then the sentences were split into individual words. Also with this tool, we were able to follow the meaning of a sentence based on the words that were used by categorizing and tagging the words. According to the NLTK tool documentation, over 25 categories can be added to the words, in this system, we selected adverbs and nouns. Adverbs modify verbs to specify the time, manner, place or direction of the event described by the verb, moreover they can also modify adjectives. On the other hand, nouns generally refer to people, places, things, or concepts and can appear after determiners and adjectives that can be subject or object of the verb.[2] The above steps were performed and the input text was transformed into a normal form, which included the identification of sentences, bullet points and cells in tables. We created specific functions to perform each task which are properly commented in the source code of this assignment for proper readability and quick reference.

Then we assigned the selected taggers for all the text in the websites, by using parts of speech tagging. In this process, the sequences were split to sentences, then, the sentence tokenizer was applied and after that, we split into words and then applied the word tokenizer. At this point the words were tagged and the nouns and adverbs were extracted from each website.

2.5 *Chunking And Stemming or Morphological Analysis*-

Chunking is an important step when extracting information and identifying entities. Each website is different from each other because not only nouns and adverbs are different but also the topic and places, for that reason we seek to locate and classify named entities that are de ned within this pages such as: names of persons, organizations, locations, expression of items, quantities, monetary values, percentages, etc. For that, we used "chunkingg" tool in order to identify them. This tool segments and labels multi-token sequences meaning that onc̄e the word-level tokenization has happened the chunking unifies them and also classify them just like tokenization. Therefore, for each URL we extracted and stored these entities and relations into text files.

Writing word stems to the database rather than words allows to treat various inflected forms of a word in the same way is called stemming. This has been done using Stemmer function outlined very clearly in the source code.

2.6 *Selecting Keywords using TF-IDF computation* -

Term Frequency, Inverse Document Frequency is a way to score the importance of words or terms in a document based on how frequently they appear across multiple documents. This means that if a word appears frequently it's because it is important and after that a high score is given. However, if a word appears in other many documents, the identifier is not unique and a low score is given. On the other hand, this method is commonly used when the goal is to model each document into a vector space, ignoring the exact ordering of the words in the document while retaining information about the occurrences of each word.[3] It is composed by two terms: the first one computes the normalized term frequency, which is the number of times a word appears in a document divided by the total number of words in that document, and the second term is the inverse document frequency, which is computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the term appears.

Also, the TF-IDF gives how important is a word to a document in a collection since it takes into consideration not only the isolated term but also the term within the document collection.[3] Because most of the text that is present in the website are words that don't give any new information, at this step, we store the relevant information about each website by performing TF-IDF. For each of the URLs we analysed the terms that occur frequently and scale them to know if they're important or not.

Till now, our system has identified the words and phrases in the text that are most useful for indexing purposes and removed words which are not useful, such as very frequent words or stopwords, and identify phrases suitable as index terms.

3 **Holistic Discussion of the Whole Web Indexing System Engineered in the Task -**

This system processes textual data and extracts interesting and significant patterns to explore knowledge from the sources given. The system performs the following steps: collecting unstructured data from URLs, pre-processing and cleansing to detect and remove

anomalies, and by doing this, we make sure we capture the real essence of the text included on the sites. We remove stop words, process and control tokenization to classify and extract relevant information. Then with the use of Natural Language Processing we performed entity recognition that allows to identify relationships and other information within the text and, we perform TF-IDF to scale the words frequency and perform web search trhough cosine similarity. Finally, 10 documents in .txt format pertaining to each step were generated that summarize all the system outputs.

However, as the size of data is increasing at exponential rates day by day this tool may one day be incapable to handle a lot of textual data, since it requires time to extract information. On the other hand, some issues arise when letting the tool interpret the sense of a document, although it is true that we can have an idea of what is the site or the text on the site about with all the ltered words, the problem is when some words have same spelling but give diverse meaning, these tools keep considering these words as similar that's why we have to be careful with grammatical rules according to the nature and context so the insights that we gained from the text are the closest as possible with the reality or the truly meaning of the text.

References

[1] Richardson, Leonard, "Beautiful Soup Documentation". Online resource: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[2] NLTK toolkit documentation, "Categorizing and tagging words". Online resource: https://www.nltk.org/book/ch05.html

[3] Stevenloria, "Tutorial: Finding important words in text using TF{IDF". Online resource: https://stevenloria.com/tf-idf/

4. Appendix –

SOURCE CODE WITH COMMENTS -

```
# import pacakages
import urllib.parse
import urllib.request
from bs4 import BeautifulSoup
from nltk import re
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk import pos_tag
from nltk import RegexpParser
from nltk import sent_tokenize
from math import log
```

```
# This function gets the URL
def get_page(my_url):
    my_req = urllib.request.Request(my_url)
    my_resp = urllib.request.urlopen(my_req)  # opens url as request object
    my_data = my_resp.read()
    return my_data  # returns URL data


# This function processes the URL for parsing
def process_webpage(my_url):
    my_html = get_page(my_url)
    my_text = parse_html(my_html)
    my_meta = parse_meta(my_html)
    return tokenize1(my_text)

# This function returns the meta tags and stores it in the file
def parse_meta(my_meta):
    my_list=[]
    page=BeautifulSoup(my_meta,'html.parser')
    for link in page.find_all('meta'):
        my_list.append(link.get('name'))
        my_list.append(link.get('content'))

    with open("Meta_Tags.txt", "w",encoding="utf-8") as file0:  # opens a file to
write to
         file0.write("Meta Tags: \n%s" % my_list)
    file0.close()

# This function parses the html page
def parse_html(my_html):
    soup = BeautifulSoup(my_html, 'html.parser')  # BeautifulSoup to parse webpage
    #print(soup)
    # Prints the meta tags if any

    for i in soup(['script', 'style']):  # remove script
        i.decompose()
    for tag in soup.findAll(True):  # removes html tags
        tag.unwrap()
    parsed_text = soup.get_text().strip().lower()
    sentence_splitting(parsed_text)
    with open("Html_Parsing.txt", "w",encoding="utf-8") as file1:  # opens a file to
write to
        file1.write("Html Parsing: \n%s" % parsed_text)
    file1.close()

    return parsed_text

# This function performs sentence splitting
def sentence_splitting(my_sentence):

    with open("Sentence_Splitting.txt", "w",encoding="utf-8") as file2:
        file2.write("Sentence Splitting: \n%s" % sent_tokenize(my_sentence))  #
nltk.sent.tokenize
    file2.close()
    return sent_tokenize(my_sentence)


# This function tokenises the parsed html file
def tokenize1(my_text):
```

```
    my_tokens = word_tokenize(my_text)  # nltk tokenization

    #stemming_function(tokens)

    normalise(my_tokens)
    with open("Tokenisation.txt", "w",encoding="utf-8") as file3:
        file3.write("Tokenisation: \n%s" % my_tokens)
    file3.close()
    return my_tokens


# This function normalises the tokens
def normalise(my_tokens):
    tokens_nopunct = [word.lower() for word in my_tokens
                      if re.search("\w", word)]  # removes punctuation and makes all
lower case
    stopwordss(tokens_nopunct)
    with open("Normalising.txt", "w",encoding="utf-8") as file4:
        file4.write("Normalising : \n%s" % tokens_nopunct)
    file4.close()
    return tokens_nopunct


# This function removes stop words and only Select keywords are left
def stopwordss(tokens_nopunct):
    stop_wordss = set(stopwords.words('English'))
    filtered_words = [word for word in tokens_nopunct if word not in stop_wordss]
    stemming_function(filtered_words)  # removes stopwords
    pos_tag_words(filtered_words)
    with open("Select_Keywords.txt", "w",encoding="utf-8") as file5:
        file5.write("Stop words and Select Keywords: \n%s" % filtered_words)
    file5.close()
    return filtered_words


# This function assigns speech tokens to each word
def pos_tag_words(my_tokens):
    my_pos = pos_tag(my_tokens) # PoS tags the words using nltk pos_tag
    with open("Pos_tag.txt", "w",encoding="utf-8") as file6:
        file6.write("Part of speech tagging : \n%s" % my_pos)
    file6.close()
    chunkingg(my_pos)



# This function chunks to follows POS Tagging and adds more structure to the sentence
def chunkingg(pos_tagged):
    # grammar chosen for chunking
    grammar = """NP:
                    {<NN><NN>}
                    {<DT>?<JJ>*<NN>}

                     """
    chunkParser = RegexpParser(grammar)  # uses grammar
    chunking_text = chunkParser.parse(pos_tagged)
    #stemming_function(chunkingg)
    with open("Chunking.txt", "w",encoding="utf-8") as file7:
        file7.write("Chunking : \n%s" % chunking_text)
    file7.close()
```

```
    #print(chunking_text)
    #stemming_function(chunking_text)
    # return chunking


# This function performs stemming to reduce a word to its word stem that affixes to
suffixes and prefixes
# #or to the roots of words
def stemming_function(my_chunks):
    my_list = []

    ps = PorterStemmer()  # nltk PorterStemmer to stem words
    for i in my_chunks:
        p_words = (ps.stem(i))
        #print(p_words)
        my_list.append(p_words)

    # puts a list together of words
    with open("Stemming.txt", "w",encoding="utf-8") as file8:
        file8.write("Stemming : \n%s" % my_list)
    file8.close()
    return my_list

def token_stemmer(my_tokens):
    ps = PorterStemmer()  # nltk PorterStemmer to stem words
    my_stemmed = [ps.stem(token) for token in my_tokens]
    return my_stemmed

def count(tokens):
    counts = dict()
    for token in tokens:
        if token in counts:
            counts[token] += 1
        else:
            counts[token] = 1
    return counts


def computeTF(tf_dict, total_tokens):
    result = dict()
    for key,value in tf_dict.items():
        result[key] = value/total_tokens
    return result


def idf_word_occurance(vocab_set, vocab_arr, num_of_urls):
    idf_occurance = dict.fromkeys(vocab_set)
    for key in idf_occurance:
        idf_occurance[key]=0
    for key in idf_occurance:
        for i in range(0, num_of_urls):
            if key in vocab_arr[i]:
                idf_occurance[key]+=1
    for key, value in idf_occurance.items():
        idf_occurance[key]=log(num_of_urls/value, 10)
    return idf_occurance

def tf_idfcalculation(tf_dicts, idf_dict):
    tfidf_arr = []
```

```
    for i in range (0, len(tf_dicts)):
        temp_dict = tf_dicts[i]
        for key, value in temp_dict.items():
            temp_dict[key] = temp_dict[key] * idf_dict[key]
        tfidf_arr.append(tf_dicts[i])
    return tfidf_arr


def main():

    tf_dicts = []
    tf_idf_dicts = []
    combined_vocab_set = set()
    separate_pages_vocab = []

    urls = ["http://www.multimediaeval.org/mediaeval2019/memorability/",
"https://sites.google.com/view/siirh2020/"]

    for url in urls:
        processed = process_webpage(url)
        normalised_tokens = normalise(processed)
        stemmed_tokens = token_stemmer(normalised_tokens)
        tf_dict = count(stemmed_tokens)
        tf_values = computeTF(tf_dict, len(stemmed_tokens))
        tf_dicts.append(tf_values)
        #print(tf_values.keys())
        vocab_set = set(tf_values.keys())
        #print(vocab_set)
        combined_vocab_set = combined_vocab_set.union(vocab_set)
        #print(combined_vocab_set)
        separate_pages_vocab.append(vocab_set)


    # Needs to run through urls again after gathering a set of all words from all
documents
    idf_occ = idf_word_occurance(combined_vocab_set, separate_pages_vocab, len(urls))
    tf_idf_dicts = tf_idfcalculation(tf_dicts, idf_occ)
    with open("TF-IDF_dictionaries.txt", "w",encoding="utf-8") as file9:
        file9.write("TF-IDF dictionaries for each website: \n")
        for dict in tf_idf_dicts:
            file9.write(str(dict) + "\n")
        file9.close()

if __name__ == '__main__':
    main()
```