**Project A: Extracting drug side effects from Full English Wikipedia**

**Group members:**

Sai Avinash Chintala Cheruvu, Mithun Mistry, Lohitaksh Yogi,

Maryam Tavakoli Hosseinabadi, Sai Amrit Bulusu, Aabir Kumar Datta

**Introduction:**

Many user studies demonstrated that Wikipedia is one of the popular references for people to check their medical information. Based on this study [1] on 2009 of 1,900 physicians, about 50% of physicians going online for professional purposes, visiting Wikipedia for their health and medical information. Meanwhile, a study [2] from 2014 demonstrated that the overall completeness of drug information on English Wikipedia was 93.1%±0.01 in compare to text books, and the accuracy was 99.7%±0.2%, which based on their conclusion they assume it as an accurate and complete reference for drug information.

|  |  |
| --- | --- |
| https://lh6.googleusercontent.com/XRsKHkPCjRfvzkDaXlWtDNsZ7zlxDcAzwJQOk6BESbj5SuZqRNTSrNoBAuVLzY3BQWRMD6v2BqmOqZQBrruULrEay9Kyd9QVwXe_Ao9mmhGgvjoLwjzG9eCGfHr9qN66TlwfA3FyqvkvMQlC-Q | https://lh6.googleusercontent.com/CQdITqS0TmtXvP09fY7cli-JT8IIyJU9jUse7iTStqRX5DSmJCvyIGwbI4h9obvTptansREd5mrQ-Bg7K3xPACSMLHT-XrI6BeugLXcxuvxN1iEvwsv147VjHh949CNtHZF9S0ExpPNH54ZEkw |

***Diagram 1:*** Diagram of accuracy and completeness of drugs on English Wikipedia based on [2]

The goal of this project is to extract the drugs’ side effect from Wikimedia dump data. We had two main ideas to do this project:

-       Extract medical-related pages and looking for the name of drug inside the page and then finding the side effect part of that page

-       Extracting the drug page itself (correlation with the title not the page content), and then finding the side effect part

The first one would give us more results for many drugs, but not necessarily the better results. Our experiment showed that it would return many irrelevant results due to some references in other drugs’ pages. For example *Aspirin* came inside the *Salicin* page to show some similarity between these two, however, the side effects of these two drugs are not that similar. Therefore, finding the word inside any medical-related page doesn’t mean the side effect part of that page is relevant.

Our explorations showed that we might lose some drugs, not the popular ones and even most of the unpopulars, that do not have independent pages, however, we will save healthiness of our results from many irrelevant data. This will improve both recall and more than that precision, as it is critical in medical related data to have a high precision. For that reason, we have chosen the second method for this project to extract the side effect of drugs.

In order to implement the second method, we separated the task into these three main steps:

**Phase 1: Extracting the drug data from the whole Wikimedia dump data**

-       List of Drugs

There are some list pages in Wikimedia data that has been linked to all the drugs’ name that are available in wiki data (some of them have incomplete pages). The way that we could recognize these pages was via their categories. We found out that the category that these pages belonged to was “Lists\_of\_drugs”. Having the name of the category, there is an interface to export the content of the pages belonging to those categories:

https://en.wikipedia.org/wiki/Special:Export

-       Extraction of the drug names

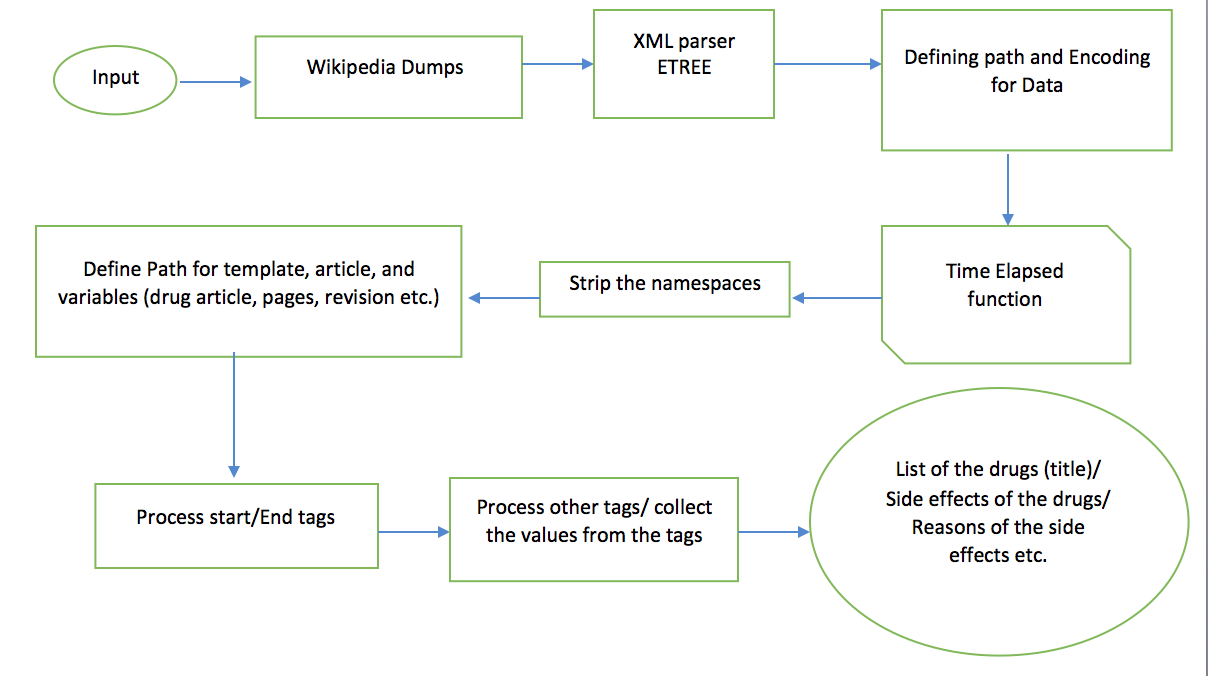
These pages consisted of the titles of drugs, and linked to the relevant pages via these titles. The format of hyperlinks in these pages is as [[\*]]. Inside the bracket there could be either one word which is the title of the page or multiple titles that are separated by “|”.  We extract all the titles and wrote them into another file.

-       Extraction of the XML of the drug data

Using the same interface for exporting Wiki data, we feed this list of titles, and got the XML export contained all the drug pages, the current contents only (not the whole history). This latter file is about 55 MB, which in compare to the whole Wikimedia dump, 50 GB, is very small!!

**Phase 2: Parse the XML file and return the title, and content of all drug pages**

The overall process is demonstrated in Diagram 2, which starts with the raw XML input coming from previous steps, parsed by ETREE parser and followed by some format and encoding modifications, and outputs the drugs’ titles and texts. The final output after doing the phase 3 will be id, drug title and side effects as mentioned in the circle. Following this part, we explained the details of important components of our work:



***Diagram 2:*** The flowchart of our method for extraction of drug side effects from Wikimedia Dump data (As we explained in the phase 1, instead of Wikipedia Dump, we have the filtered data, which contains only the drug pages)

**ElementTree (ET):**

Python offers us various options for the manipulation and parsing of XML data [3]. One such option, and a very highly recommended one, is ElementTree [4]. It gives us a python API which is very light to use and easy to handle. It is supported by a strong C implementation for parsing and creating XML data. ET is faster than a lot of other available options and also provided a feature by which we can parse data without having to load the entire dataset into memory. ET can be implemented in more ways than one using the Python standard library. *xml.etree.ElementTree* is the pure Python implementation and *xml.etree.cElementTree* is the C implementation. The latter has an accelerated implementation and takes up less memory space.

XML is a hierarchical data format and the best way to visualise this data would be in the form of a tree data structure. *ElementTree* represents the entire tree in the representation and *Element* depicts a single node.We use these objects to reach and connect with all the elements in the XML document using a simple recursive procedure, that is, for every element we recursively visit its children. The *iter* method can be used to do a depth first iteration over the sub elements below the parent element.

*Execution on Wikimedia articles:*

The file is very big so it is important not to read directly as the DOM parser would usually do. So, we use xml.etree.ElementTree class. We begin to stream the XML file. Wikimedia has several pages in a single document. Each page starts with a <page> tag and ends with </page> tag. So, our first step is separating the pages based on the tags. Next step after getting the page is iterating through different tags available on the page. Each page has many tags like comment, model, text etc. However, all tags are not useful. Only text tag is important to us as it contains side effects. When the iterator reaches to the text tag, we get the entire content of text tag and save it in a CSV file along with its title.

The content we got from the text has several other information like “Society and culture”, “Definition and Medical uses” etc. which are not relevant. We are interested in side effects and side effects are present in the file with different tags like “Side Effects”, “Adverse effects” etc. Now, from the entire content of text tag, we will only extract the side effects in which we are interested.

**Phase 3: Having the content of the drug pages, finding the side effect part / paragraph and save the results as our final document**

Having the final document of this step, we can just look into the titles, find the target drug and retrieve the side effect of that drug.

* **Searching for Side Effects**

For each drug page in wikipedia, the side effects are presented in a designated column called Effects or Side Effects or Adverse effects. We can search for these columns in the drug pages in a plethora of ways. Since the data is in xml format, we can certainly choose one of the xml parsers to search for headings that contain xmls tags named as side effects or adverse effects. But assuming that some of the headings don’t have xml tags, we used regular expression to search for headings related to side effects for each page. A regular expression, regex or regexp is, in theoretical computer science and formal language theory, a sequence of characters that define a search pattern. Usually this pattern is then used by string searching algorithms for "find" or "find and replace" operations on strings.

We used a regular expression to find the side effects/effects/adverse effects of a particular drug. In each drug page, the effects are either named side effects or effects or adverse effects. The regular expression that we used is of the form “**^==[a-zA-Z0-9 ]\*effects+==**”. The headings and subheadings in the xml document for each page are presented in the form ==”Heading”== or ===”Sub-Heading”===. So, the regular expression searches for sub-headings or headings starting and ending with two or more ‘=’ symbols, the headings can be effects or side effects or adverse effects and this expression deals with such anomalies - “**[a-zA-Z0-9 ]\*effects+**”.

* **Extracting Side Effects**

The code searches for the headings in the text, line by line and if the text in the line matches the regular expression, it extracts the text contained in the heading. The extraction is a line by line process using Python file read line operation. The extraction stops when the reader sees another heading. This is an iterative process which is performed for each drug page. The output contains the drug name and its side effects in text format.

* **Extracting Side Effects Names**

In wikipedia, almost all the side effects are hyperlinked and they have their own wikipedia pages. The text we obtain in the extraction phase in not processed and contains the hyperlink tags. We used another regular expression to find the hyperlink tags in the text we obtained. The regular expression is of the form “\[\[.\*?\]\]”. By using this we can get the names of all the words in the text which are listed as hyperlinks. The only downside by using such a process is that, we sometimes get words which are hyperlinked but not names of side effects.

**Numbers:**

* Total scrapped drugs from list of drugs page : 28000 (~ approx. which include synonyms, abbreviations, etc., )
* Filtered drugs from the list of drugs : 16,000
* Total drugs with articles on wikipedia : 9000
* Total drugs with a separate section for side effects : 1218
* Total drugs without a separate section for side effects: 7782
* Total drugs with something related to effects : 1511 (6271 articles do not )

**Regular Expressions:**

The text contained in these sections are extracted using regular expression -

* '^==[ ]\*[a-zA-Z0-9 ]\*effects+'
* '^==+[ ]\*[a-zA-Z0-9 ]\*effects+'
* '^=='
* '<!--[ ]\*[a-zA-Z0-9 ]\* effects+'
* '<!--'
* '[a-zA-Z0-9 ]\*effects'

**Results and Discussion:**

Bellow you can see a sample output for sample drug “*Oxycodone*”. We just demonstrate two first paragraphs of the output here to save our space for this report.

----*Serious side effects of oxycodone include reduced sensitivity to pain (beyond the pain the drug is taken to reduce), [[euphoria]], [[anxiolysis]], feelings of [[relaxation (psychology)|relaxation]], and [[respiratory depression]].&lt;ref name=&quot;FitzgibbonLoeser2012&quot; /&gt; Common side effects of oxycodone include [[constipation]] (23%), [[nausea]] (23%), [[vomiting]] (12%), [[somnolence]] (23%), [[dizziness]] (13%), [[pruritus|itching]] (13%), [[dry mouth]] (6%), and [[sweating]] (5%).&lt;ref name=&quot;FitzgibbonLoeser2012&quot; /&gt;&lt;ref name=drugscomSE&gt;{{cite web |title=Oxycodone Side Effects |url=http://www.drugs.com/sfx/oxycodone-side-effects.html |publisher=Drugs.com|accessdate=22 May 2013}}&lt;/ref&gt; Less common side effects (experienced by less than 5% of patients) include [[anorexia (symptom)|loss of appetite]], [[anxiety|nervousness]], [[abdominal pain]], [[diarrhea]], [[urinary retention|urine retention]], [[dyspnea]], and [[hiccups]].&lt;ref name=&quot;OxyContinInsertUS&quot; /&gt;*

*In high doses, [[overdose]]s, or in some persons not tolerant to opioids, oxycodone can cause [[shallow breathing]], [[bradycardia|slowed heart rate]], cold/clammy skin, [[apnea|pauses in breathing]], [[hypotension|low blood pressure]], [[miosis|constricted pupils]], [[circulatory collapse]], [[respiratory arrest]], and [[death]].&lt;ref name=&quot;OxyContinInsertUS&quot; /&gt;*

As we can see that the above output obtained from extracting the side effects has a lot of unprocessed text in it and the reader might find it intimidating to read the entire text to figure out what the side effects are. As explained earlier, the side effects are hyperlinked and can be extracted. Example: [[respiratory arrest]]. By using the regular expression in the ‘Extracting Side Effects Names’ phase, we extract the following list of side effects. The following output is processed and is easier to read, like previous part we just left few number of items to save the report space and removed the rest from here.

**Data Processing:**

Since the side effects paragraphs are in XML format and contain unwanted symbols, numbers etc. We performed some data preprocessing techniques such as:

* Used Beautiful Soup 4 to extract text and remove XML tags.
* Extracted only text and removed all other characters by using the regular expression - "[^a-zA-Z]"
* Lowercased all the data to regularize the data.
* Output - Drug name and side effects paragraphs in csv.

**Entity Analysis Extraction:**

Entity Analysis inspects the given text for known entities (proper nouns such as public figures, landmarks, etc.), and returns information about those entities. Entity analysis is performed with the analyzeEntities method. The assumption is that the side effects paragraphs have nouns which are the side effects names. Following are the steps we took:

* Entities broadly fall into two categories: proper nouns that map to unique entities (specific people, places, etc.) or common nouns (also called "nominals" in natural language processing).
* Used Google NL API’s analyze\_entities() method to extract entities.
* We assumed that the side effects names fall under the category either “EVENT” or “OTHER”
* Entities with “salience” > 0.001.

**List of side effects for Oxycodone:**

* euphoria
* anxiolysis
* relaxation (psychology)
* relaxation
* respiratory depression
* constipation
* nausea
* vomiting
* somnolence
* dizziness
* pruritus
* itching
* dry mouth
* sweating
* anorexia (symptom)
* loss of appetite

In order to evaluate these results, we can use the resulted either itemized/paragraphed side effects, and compare them with the ones in either reference books, or authentic medical websites such as drugs.com, or Medscape database, and find the average accuracy and precision for some random drugs.

**Evaluation Strategy:**

While in this project, we did not do the evaluation, we will provide our evaluation strategy that we couldn’t do it based on time-limit. For the evaluation we need three components to be defined: “Ground Truth” , “Drug-level Recall & Precision”, “Side-effect-level Recall & Precision”

**Ground Truth**

Ground truth will be our reference that we will compare our work with it. Anything that we take as ground truth will be our 100% performance and our results should ideally look like that.

For that the first suggestion we have will be Text Books like the  ones used in [2] and accumulate their knowledge [2]. In this way, our accuracy and recall will be compared to the traditional way of finding the drugs’ side effect for example for medical students.

The next suggestion will be online Authorized websites like webmd or drugs.com. This methods assume that these websites are good, and we want our methodology to be as informative as them. As a result we will evaluate our work in compare to the trusted online tools that people can use, and say if we can fulfill the same trust for our works.

The final suggestion will be manually extracting the side effects of random drug from wikipedia and compare our result with them. While the previous methods are evaluating Wikipedia as a reference for checking drugs’ side effect, the goal of this ground truth will be evaluation of our implementation, which means how accurate and complete is our output in compare to what human will get by manually searching Wikipedia.

**Drug-Level Recall & Precision**

By drug-level evaluation our goal is finding out how successful our methodology was in providing all the drug names. This evaluation definitely should be weighted toward more important/popular drugs than those that are rarely used/known.

*Recall* = (number of matched drug name with the ground-truth / number of all the drugs in ground-truth)

*Precision* = (number of matched drug name with the ground-truth / number of all the returned values by our methodology)

It is important to note that the precision could the result of retrieving other names, instead of drug names and/or the incompleteness of our ground-truth!!

**Side-Effect-Level Recall & Precision**

The side-effect evaluation on the other hand means to measure how successful our approach is in extracting all and accurate side-effect for each drug. In order to do this evaluation we will pick random drugs and get our output side effects for that. Then we will compare them with our selected ground-truth and based on those we can calculate Recall and Precision:

*Recall* = (number of matched side effects with the ground-truth / number of all the ground-truth side effect)

*Precision* = (number of matched side effects with the ground-truth / number of all the returned side effect by our methodology)

**Future works:**

The next step would be computing the recall and precision factor, and trying to extend the work toward other medical related articles of wikimedia, to compare and see if the result will be improved.

Also, the output obtained after extracting the side effects names might contain some words which are not names of side effects, we can, in the future write a filter that filters the non-side effects words and only extracts the words which are side effects.

Current version only contains all the drug names and their side-effect. As one of the future works we can consider getting a query, considering dictation errors and printing the side effect for the user. Another idea for working on improving this work is converting the side effect paragraph into beautiful bullet point format.

**Appendix (codes & details):**

***Note:*** You can also find all these source codes and data in this address:

https://github.com/mithunjmistry/mdswiki

**Appendix I (export drug name):**

Sample input records:

Sample output records:

|  |  |
| --- | --- |
| Accolate  AstraZenecazafirlukast  Zafirlukast  Accuneb  Accupril |  |
|  |  |

|  |
| --- |
|  |
| ### This code gets the content of “list\_of\_drugs” pages in wikipedia and returns the #name of the drugs. It will be fed to the next step (wikipedia exporter) to get the #drugs’ page contents ###  import re |
|  |
| def find\_drug\_pages(pattern,file\_content): |
| starts = [m.start() for m in re.finditer(pattern, file\_content)] |
| ends = [m.end() for m in re.finditer(pattern, file\_content)] |
| # retrieve [[\*]] |
| hyper\_links = [file\_content[start+2:end-2] for start,end in zip(starts,ends)] |
|  |
| # | |
|  |
| drug\_items = [] |
| for item in hyper\_links: |
| inner\_items = item.split("|") |
| drug\_items = drug\_items + inner\_items |
|  |
| # space |
| return drug\_items |
|  |
|  |
| #read file |
| f = open("Wikipedia-20171013010045.xml") |
| list\_of\_drugs\_pages = f.read() |
| #drug list |
| drug\_pages\_list = find\_drug\_pages("\[\[.\*?\]\]",list\_of\_drugs\_pages) |
|  |
| #print items |
| f\_write = open("drug\_page\_titles",'w') |
|  |
| page\_titles = "" |
|  |
| for page\_title in drug\_pages\_list: |
| page\_titles = page\_titles + page\_title + "\n" |
|  |
| f\_write.write(page\_titles) |
|  |
| #export wikipedia pages |

**Appendix II (XML Parser):**

**import** **xml.etree.ElementTree** **as** **etree**  
**import** **codecs**  
**import** **csv**  
**import** **time**  
**import** **os**  
  
PATH\_WIKI\_XML = r'C:\Users\lohit\Desktop'  
FILENAME\_WIKI = 'Wikipedia-20171015101942.xml'  
FILENAME\_ARTICLES = 'articles.csv'  
FILENAME\_REDIRECT = 'articles\_redirect.csv'  
FILENAME\_TEMPLATE = 'articles\_template.csv'  
ENCODING = "utf-8"  
  
  
*# Nicely formatted time string*  
**def** hms\_string(sec\_elapsed):  
    h = int(sec\_elapsed / (60 \* 60))  
    m = int((sec\_elapsed % (60 \* 60)) / 60)  
    s = sec\_elapsed % 60  
    **return** "**{}**:**{:>02}**:**{:>05.2f}**".format(h, m, s)  
  
  
**def** strip\_tag\_name(t):  
    t = elem.tag  
    idx = k = t.rfind("}")  
    **if** idx != -1:  
        t = t[idx + 1:]  
    **return** t  
  
  
pathWikiXML = os.path.join(PATH\_WIKI\_XML, FILENAME\_WIKI)  
pathArticles = os.path.join(PATH\_WIKI\_XML, FILENAME\_ARTICLES)  
pathArticlesRedirect = os.path.join(PATH\_WIKI\_XML, FILENAME\_REDIRECT)  
pathTemplateRedirect = os.path.join(PATH\_WIKI\_XML, FILENAME\_TEMPLATE)  
  
totalCount = 0  
articleCount = 0  
redirectCount = 0  
templateCount = 0  
title = **None**  
start\_time = time.time()  
  
**with** codecs.open(pathArticles, "w", ENCODING) **as** articlesFH, \  
        codecs.open(pathArticlesRedirect, "w", ENCODING) **as** redirectFH, \  
        codecs.open(pathTemplateRedirect, "w", ENCODING) **as** templateFH:  
    articlesWriter = csv.writer(articlesFH, quoting=csv.QUOTE\_MINIMAL)  
    redirectWriter = csv.writer(redirectFH, quoting=csv.QUOTE\_MINIMAL)  
    templateWriter = csv.writer(templateFH, quoting=csv.QUOTE\_MINIMAL)  
  
    articlesWriter.writerow(['id', 'title', 'redirect'])  
    redirectWriter.writerow(['id', 'title', 'redirect'])  
    templateWriter.writerow(['id', 'title'])  
  
    **for** event, elem **in** etree.iterparse(pathWikiXML, events=('start', 'end')):  
        tname = strip\_tag\_name(elem.tag)  
  
        **if** event == 'start':  
            **if** tname == 'page':  
                title = ''  
                id = -1  
                redirect = ''  
                inrevision = **False**  
                ns = 0  
            **elif** tname == 'revision':  
                *# Revision ID not needed*  
                inrevision = **True**  
        **else**:  
            **if** tname == 'title':  
                title = elem.text  
                print(title)  
            **elif** tname == 'id' **and** **not** inrevision:  
                id = int(elem.text)  
            **elif** tname == 'redirect':  
                redirect = elem.attrib['title']  
            **elif** tname == 'ns':  
                ns = int(elem.text)  
            **elif** tname == 'page':  
                totalCount += 1  
            **elif** tname == 'text':  
                textby=elem.text  
                print(textby)  
  
                **if** ns == 10:  
                    templateCount += 1  
                    templateWriter.writerow([id, title])  
                **elif** len(redirect) > 0:  
                    articleCount += 1  
                    articlesWriter.writerow([id, title, redirect])  
                **else**:  
                    redirectCount += 1  
                    redirectWriter.writerow([id, title, redirect])  
  
                *# if totalCount > 100000:*  
                *#  break*  
  
                **if** totalCount > 1 **and** (totalCount % 100000) == 0:  
                    print("**{:,}**".format(totalCount))  
  
            elem.clear()  
  
elapsed\_time = time.time() - start\_time  
  
print("Total pages: **{:,}**".format(totalCount))  
print("Template pages: **{:,}**".format(templateCount))  
print("Article pages: **{:,}**".format(articleCount))  
print("Redirect pages: **{:,}**".format(redirectCount))  
print("Elapsed time: **{}**".format(hms\_string(elapsed\_time))

**Appendix III (Side-Effect Extraction):**

|  |  |
| --- | --- |
| # -\*- coding: utf-8 -\*- |  |
|  | """ |
|  | Created on Fri Oct 13 20:35:36 2017 |
|  |  |
|  | @author: saiav |
|  | """ |
|  | from xml.dom import minidom |
|  | import re |
|  | ls = list() |
|  | f = open('drugs\_full\_xml.xml', encoding = "utf8") |
|  |  |
|  |  |
|  | for line in f: |
|  | #ls = line.rstrip() |
|  | ls = re.findall('^==[a-zA-Z0-9 ]\*Effects+==',line) |
|  | if len(ls) > 0: |
|  | while True: |
|  | nline = f.readline() |
|  | if len(re.findall('^==+[a-zA-Z0-9 ]\*effects+==+',nline))>0: |
|  | continue |
|  | elif len(re.findall('^==',nline))>0: |
|  | break |
|  | else: |
|  | print(nline) |
|  |  |
|  |  |
|  | #print([f.readline() for i in range(4)]) |

**APPENDIX IV(Data Preprocessing and Entity Extraction):**

import argparse

import json

import sys

import os

import csv

from bs4 import BeautifulSoup

import re

from nltk.corpus import stopwords

from googleapiclient import discovery

import httplib2

from oauth2client.client import GoogleCredentials

# words = lower\_case.split()

# words = [w for w in words if not w in stopwords.words("english")]

# print(lower\_case)

def get\_service():

   credentials = GoogleCredentials.get\_application\_default()

   scoped\_credentials = credentials.create\_scoped(

       ['https://www.googleapis.com/auth/cloud-platform'])

   http = httplib2.Http()

   scoped\_credentials.authorize(http)

   return discovery.build('language', 'v1beta1', http=http)

def get\_native\_encoding\_type():

   """Returns the encoding type that matches Python's native strings."""

   if sys.maxunicode == 65535:

       return 'UTF16'

   else:

       return 'UTF32'

def analyze\_entities(text, encoding='UTF32'):

   body = {

       'document': {

           'type': 'PLAIN\_TEXT',

           'content': text,

       },

       'encoding\_type': encoding,

   }

   service = get\_service()

   request = service.documents().analyzeEntities(body=body)

   response = request.execute()

   entities = []

   for i in range(len(response['entities'])):

    if(response['entities'][i]['salience']>0.001 and (response['entities'][i]['type']=="OTHER" or response['entities'][i]['type']=="EVENT")):

    entities.append(response['entities'][i]['name'])

   return(set(entities))

side\_effects = {}

for filename in os.listdir(r'C:\Users\amrit\OneDrive\Documents\Projects\wikidrugs\Final\_Side\_effects\Side\_effects\_Final'):

if filename.endswith('.txt'):

pathname = os.path.join(r'C:\Users\amrit\OneDrive\Documents\Projects\wikidrugs\Final\_Side\_effects\Side\_effects\_Final', filename)

textfile = open(pathname,encoding = 'utf8')

buffer = []

for line in textfile:

buffer.append(line)

text = ' '.join(map(str, buffer))

else:

soup = BeautifulSoup(text)

letters\_only = re.sub("[^a-zA-Z]", " ", soup.get\_text())

lower\_case = letters\_only.lower()

# entities = analyze\_entities(lower\_case)

side\_effects[filename.replace(".txt", "")] = lower\_case

textfile.close()

with open('Side\_effects\_final\_paragraph.csv', 'w', encoding='utf-8') as csv\_file:

   writer = csv.writer(csv\_file)

   for key, value in side\_effects.items():

      writer.writerow([key, value])

**References:**

[1] Comer B. Docs look to Wikipedia for condition info: Manhattan Research [Internet] New York, NY: MM&M Medical Marketing and Media; 21 Apr 2009 [cited 4 Nov 2010]. < http://www.mmm-online.com/docs-look-to-wikipedia-for-condition-info-manhattan-research/article/131038/>

[2] Kräenbring, J., Penza, T. M., Gutmann, J., Muehlich, S., Zolk, O., Wojnowski, L., ... & Sarikas, A. (2014). Accuracy and completeness of drug information in Wikipedia: a comparison with standard textbooks of pharmacology.

[3]  http://www.heatonresearch.com/2017/03/03/python-basic-wikipedia-parsing.html

[4] https://pypi.python.org/pypi/elementtree/