In [1]:
```
pip install num2words
```

```
Requirement already satisfied: num2words in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (0.5.12)
Requirement already satisfied: docopt>=0.6.2 in /Users/Raj/opt/anaconda3/lib/python3.9/site-packages (from num2words)
(0.6.2)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
## Library imports
import numpy as np
import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer

import os, glob, re, sys, random, unicodedata, collections
from tqdm import tqdm
from functools import reduce
import nltk
from collections import Counter

from nltk.corpus import stopwords
from nltk.stem import RSLPStemmer
from nltk.tokenize import sent_tokenize , word_tokenize
```

In [3]:
```
import os, pickle

import nltk, string, copy, re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from collections import Counter
from num2words import num2words

import numpy as np, pandas as pd
import math
```

In [4]:
```
### Processing documents
lines = ""
with open('./CISI.ALL') as f:
    for l in f.readlines():
        lines += "\n" + l.strip() if l.startswith(".") else " " + l.strip()
    lines = lines.lstrip("\n").split("\n")
```

In [5]:
```
doc_count = 0
doc_name_dict = {}
doc_author_dict = {}
doc_text_dict = {}
for l in lines:
    if l.startswith(".I"):
        doc_id = int(l.split(" ")[1].strip())-1
    elif l.startswith(".T"):
        doc_name_dict[doc_id] = l.strip()[3:]
    elif l.startswith(".A"):
        doc_author_dict[doc_id] = l.strip()[3:]
    elif l.startswith(".W"):
        doc_text_dict[doc_id] = l.strip()[3:]
    else:
        continue
```

In [6]:
```
len(doc_name_dict)
len(doc_author_dict)
len(doc_text_dict)
```

Out[6]:
```
1460
```

In [7]:
```
# Text Pre-processing

STOP_WORDS = set(stopwords.words('english'))
WORD_MIN_LENGTH = 2


def convert_lower_case(text):
    return np.char.lower(text)


def remove_punctuation(text):
    symbols = "!\"#$%&()*+-./:;<=>?@[\]^_`{|}~\n"
    for i in range(len(symbols)):
        text = np.char.replace(text, symbols[i], ' ')
        text = np.char.replace(text, ',', ' ')
        text = np.char.replace(text, "\s+", " ")
    return text

def remove_apostrophe(text):
    return np.char.replace(text, "'", "")

def stemming(text):
    stemmer= PorterStemmer()

    tokens = word_tokenize(str(text))
    new_text = ""
    for w in tokens:
        new_text = new_text + " " + stemmer.stem(w)
    return new_text

def convert_numbers(text):
    tokens = word_tokenize(str(text))
    new_text = ""
    for w in tokens:
        try:
            w = num2words(int(w))
        except:
            a = 0
        new_text = new_text + " " + w
    new_text = np.char.replace(new_text, "-", " ")
    return new_text

def preprocess(text):
    text = convert_numbers(text)
    text = convert_lower_case(text)
    text = remove_punctuation(text)
    text = remove_apostrophe(text)
    text = stemming(text)
    return text


def tokenize_text(text):
    text = preprocess(text)
    text = re.sub(re.compile('\n'),' ',text)
    words = word_tokenize(text)
    words = [word.lower() for word in words]
    words = [word for word in words if word not in STOP_WORDS and len(word) >= WORD_MIN_LENGTH]
    return words
```

In [8]:
```
def inverted_index(words):
    """Create a inverted index of words (tokens or terms) from a list of terms

    Parameters:
    words (list of str): tokenized document text

    Returns:
    Inverted index of document (dict)

   """
    inverted = {}
    for index, word in enumerate(words):
        locations = inverted.setdefault(word, [])
        locations.append(index)
    return inverted

def inverted_index_add(inverted, doc_id, doc_index):
    """Insert document id into Inverted Index

    Parameters:
    inverted (dict): Inverted Index
    doc_id (int): Id of document been added
    doc_index (dict): Inverted Index of a specific document.

    Returns:
    Inverted index of document (dict)

   """
    for word in doc_index.keys():
        locations = doc_index[word]
        indices = inverted.setdefault(word, {})
        indices[doc_id] = locations
    return inverted
```

In [9]:
```
inverted_doc_indexes = {}
files_with_index = []
files_with_tokens = {}

for doc_id, text in tqdm(doc_text_dict.items()):
    words = tokenize_text(text)
    #Store tokens
    files_with_tokens[doc_id] = words
    doc_index = inverted_index(words)
    inverted_index_add(inverted_doc_indexes, doc_id, doc_index)
    files_with_index.append(doc_name_dict[doc_id])
```

```
100%|██████████████████████████████████| 1460/1460 [00:06<00:00, 221.49it/s]
```

Ranked TF-IDF Retrieval Model

In [10]:
```
## Number of documents each term occurs
DF = {}
for word in inverted_doc_indexes.keys():
    DF[word] = len ([doc for doc in inverted_doc_indexes[word]])

total_vocab_size = len(DF)
print(total_vocab_size)
```

```
5767
```

In [11]:
```
vocab = list(DF.keys())
```

In [12]:
```
tf_idf = {} # Our data structure to store Tf-Idf weights

N = len(files_with_tokens)

for doc_id, tokens in tqdm(files_with_tokens.items()):

    counter = Counter(tokens)
    words_count = len(tokens)

    for token in np.unique(tokens):

        # Calculate Tf
        tf = counter[token] # Counter returns a tuple with each terms counts
        tf = 1+np.log(tf)

        # Calculate Idf
        if token in DF:
            df = DF[token]
        else:
            df = 0
        idf = np.log((N+1)/(df+1))

        # Calculate Tf-idf
        tf_idf[doc_id, token] = tf*idf
```

```
100%|██████████████████████████████████| 1460/1460 [00:00<00:00, 2967.33it/s]
```

In [13]:
```
def ranked_search(k, tf_idf_index, file_names, query):
    """Run ranked query search using tf-idf model.

    Parameters:
    k (int): number of results to return
    tf_idf_index (dict): Data Structure storing Tf-Idf weights to each
                         pair of (term,doc_id)
    file_names (list): List with names of files (books)
    query (txt): Query text

    Returns:
    Top-k names of books that matchs the query.

   """
    tokens = tokenize_text(query)
    query_weights = {}
    for doc_id, token in tf_idf:
        if token in tokens:
            query_weights[doc_id] = query_weights.get(doc_id, 0) + tf_idf_index[doc_id, token]

    query_weights = sorted(query_weights.items(), key=lambda x: x[1], reverse=True)
    results = []
    for i in query_weights[:k]:
        results.append(file_names[i[0]])

    return results
```

In [14]:
```
print(ranked_search(10, tf_idf, files_with_index,
                    "What is information science?  Give definitions where possible."))
```

```
['The Origins of the Information Crisis:  A Contribution to the Statement of the Problem', 'The Phenomena of Interest
to Information Science', 'Progress in Documentation', 'Analysis and Organization of Knowledge for Retrieval', 'Inform
ation Science: Toward the Development of a True Scientific Discipline', 'Topical Aspects of Informatics to-date', 'In
formation:  Methodology', 'Comments about Terminology in Documentation. II:  communication and Information', 'A Defin
ition of Relevance for Information Retrieval', 'Citation Indexes for Science']
```

In [15]:
```
print(ranked_search(10, tf_idf, files_with_index,
                    "Testing automated information systems."))
```

```
['The Annual Review of Information Science and Technology', 'Conceptual Design of an Automated National Library Syste
m', 'Guidelines for Library Automation; a Handbook for Federal and Other Libraries', 'Automated Information-Retrieval
Systems (IRS)', 'Linguistics and Information Science', 'Automated Acquisitions Procedures at the University of Michig
an Library', 'Adventures in Librarianship', 'Automation in Libraries', 'Library Automation:  Experience, Methodology,
and Technology of the Library as an Information System', 'Automation Activities in the Processing Department of the L
ibrary of Congress']
```

In [17]:
```
print(ranked_search(10, tf_idf, files_with_index,
                    "Methods of coding used in computerized index systems."))
```

```
['The Automatic Encoding of Chemical Structures', 'Indexing Language Structure for Automated Retrieval', 'A Notation
for Coding Organic Compounds', 'Future Developments in Telecommunications', 'A Computer_Aided Information Service for
Nuclear Science and Technology', 'Human Factors in the Design of an Interactive Library System', 'Some Aspects of Dev
eloping and Studying a Descriptor Information Language for General Technology', 'The Thesaurofacet: A Multipurpose Re
trieval Language Tool', 'A Study of General Categories Applicable to Classification and Coding in Documentation', 'Se
lected Results From An Inquiry Into Testing of Information Retrieval Systems']
```

BOOLEAN DOCUMENT RETRIEVAL

In [18]:
```
## Using AND as logical operator
def boolean_search(inverted, file_names, query):
    """Run a boolean search with AND operator between terms over
    the inverted index.

    Parameters:
    inverted (dict): Inverted Index
    file_names (list): List with names of files (books)
    query (txt): Query text

    Returns:
    Names of books that matchs the query.

   """
    # preprocess the user query using same function used to build Inverted Index
    words = [word for _, word in enumerate(tokenize_text(query)) if word in inverted]
    # list with a disctinct document match for each term from query
    results = [set(inverted[word].keys()) for word in words]
    # AND operator. Replace & for | to modify to OR behavior.
    docs = reduce(lambda x, y: x & y, results) if results else []
    return ([file_names[doc] for doc in docs])
```

In [20]:
```
print(boolean_search(inverted_doc_indexes, files_with_index,
                     "automated information systems."))
```

```
['A Decision Theoretic Foundation for Indexing', 'Design of Information Systems and Services', 'The Annual Review of
Information Science and Technology', 'Utility of Automatic Classification Systems for Information Storage and Retriev
al', 'Application of Computer Technology to Library Process: a syllabus', 'Adventures in Librarianship', 'Selective D
issemination and Indexing of Scientific Information', 'Library Automation:  Experience, Methodology, and Technology o
f the Library as an Information System', 'Guidelines for Library Automation; a Handbook for Federal and Other Librari
es', 'Automated Language Processing', 'Automation in Libraries', 'Improving Access to Library Resources', 'Automated
Information-Retrieval Systems (IRS)', 'Linguistics and Information Science', 'Psychology and Information', 'Fields of
Information on Library of Congress Catalog Cards: Analysis of a Random Sample, 1950-1964', 'Conceptual Design of an A
utomated National Library System', 'Analysis of Information Flows in Shipbuilding and the Allied Fields', 'MEDLARS:
A Summary Review and Evaluation of Three Reports', 'Automation Activities in the Processing Department of the Library
of Congress', 'A Grammatical Elements in a Descriptor Language for an Information Retrieval System', 'Scope: A Cost A
nalysis of an Automated Serials Record System', 'Standardization Requirements of a National Program for Information T
ransfer', 'Encyclopedia of Information Systems and Services']
```

BOOLEAN DOCUMENT RETRIEVAL BASED ON COSINE SIMILARITY

Ranking using Cosine Similarity

Matching score gives relevant documents but quite fails when we give **long query**.

Cosine similarity will rank all documents as vectors of **TFIDF** tokens and consider the **angle** between the two vectors.

In [22]:
```
def compute_cosine_similarity(a, b):
    '''compute consine similarity between two vectors'''
    return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))
```

In [24]:
```
D = np.zeros((N, total_vocab_size))
for k, v in tf_idf.items():
    ind = vocab.index(k[1])
    D[k[0]][ind] = v
```

In [32]:
```
import operator
import itertools

def cosine_similarity_ranked_search(k, query):
    Q = np.zeros((total_vocab_size))
    query_tokens = word_tokenize(str(preprocess(query)))
    counter = Counter(query_tokens)
    words_count = len(query_tokens)
    for i, token in enumerate(set(query_tokens)):
        tf = counter[token] / words_count
        df = DF[token] if token in vocab else 0
        idf = np.log((N+1) / (df + 1))
        Q[i] = tf * idf
    cosine_similarity = {i:0 for i in range(len(D))}
    for i, doc_vector in enumerate(D):
        cosine_similarity[i] = compute_cosine_similarity(Q, doc_vector)
    sorted_d = dict(sorted(cosine_similarity.items(), key=operator.itemgetter(1),reverse=True))
    top_k = dict(list(sorted_d.items())[0: k])
    result = []
    for i in top_k:
        result.append(files_with_index[i])
    return result
```

In [33]:
```
print(cosine_similarity_ranked_search(10, 'What lists of words useful for indexing or classifying material are availabl
```

```
['18 Editions of the Dewey Decimal Classifications', 'Personnel Administration in Libraries', 'Documentation', 'HDB o
f Data Processing for Libraries', 'Classifying Courses in the University Catalog', 'Non-book Materials: The Organizat
ion of Integrated Collections', 'The Subject Approach to Information', 'Dewey Decimal Classification', 'Correlation t
he Subjects of Books Taken Out Of and Books Used Within an Open-Stack Library', 'Classification Practice in Britain.
Report on a survey of classification opinion and practice in Great Britain, with particular reference to the Dewey De
cimal Classification']
```

In [ ]: