Lab 3

Jaydeep Dey - 20BCE1419

In [ ]:
```
from nltk.corpus import stopwords
from nltk import word_tokenize
from string import punctuation

stop_words = set(stopwords.words('english'))
```

In [ ]:
```
doc = [
    "Selenium is a portable framework for testing web applications",
    "Beautiful Soup is useful for web scraping",
    "It is a python package for parsing the pages",
    "Java programming can be used for web applications",
    "scraping web and crawling web is useful"
]
```

In [ ]:
```
def preprocessing(word):
    word = word.strip()
    word = word.lower()
    word = word_tokenize(word)
    new_word = list()
    for i in word:
        if i not in stop_words and i not in punctuation:
            new_word.append(i)
    return new_word
```

In [ ]:
```
new_doc = list()

for i in doc:
    new_doc.append(preprocessing(i))
new_doc
```

Out[ ]:
```
[['selenium', 'portable', 'framework', 'testing', 'web', 'applications'],
 ['beautiful', 'soup', 'useful', 'web', 'scraping'],
 ['python', 'package', 'parsing', 'pages'],
 ['java', 'programming', 'used', 'web', 'applications'],
 ['scraping', 'web', 'crawling', 'web', 'useful']]
```

In [ ]:
```
def WordOccurance(text, word):
    new_text = text.replace('/[^A-Za-z0-9]/g', '').lower().strip().split()
    word_pos = list()
    word_count = 0
    for i in range(len(new_text)):
        if word == new_text[i]:
            word_count += 1
            word_pos.append(i)
    return (word_count, word_pos)

# WordOccurance("selenium portable selenium selenium portable portable", 'portable')
```

In [ ]:
```
inverted_index = dict()

for (i,text) in enumerate(doc):
```

```
    words = preprocessing(text)
    for word in words:
        if word not in inverted_index.keys():
            inverted_index[word] = []
            wordcount, wordpos = WordOccurance(text, word)
            inverted_index[word].append((i+1, wordcount, wordpos))
inverted_index
```

Out[ ]:
```
{'selenium': [(1, 1, [0])],
 'portable': [(1, 1, [3])],
 'framework': [(1, 1, [4])],
 'testing': [(1, 1, [6])],
 'web': [(1, 1, [7])],
 'applications': [(1, 1, [8])],
 'beautiful': [(2, 1, [0])],
 'soup': [(2, 1, [1])],
 'useful': [(2, 1, [3])],
 'scraping': [(2, 1, [6])],
 'python': [(3, 1, [3])],
 'package': [(3, 1, [4])],
 'parsing': [(3, 1, [6])],
 'pages': [(3, 1, [8])],
 'java': [(4, 1, [0])],
 'programming': [(4, 1, [1])],
 'used': [(4, 1, [4])],
 'crawling': [(5, 1, [3])]}
```

Search Word in Inverted Index

In [ ]:
```
print("Selenium word occurs in the following position")
print("Word 'Selenium'")
print("Doc no  no.of times  offset number")
for word in inverted_index.keys():
    if word == "Selenium".lower():
        data = inverted_index[word][0]
        print('D',data[0], "\t\t", data[1],"\t" ,data[2])

print("Word 'Web'")
print("Doc no  no.of times  offset number")
for word in inverted_index.keys():
    if word=="web".lower():
        data = inverted_index[word][0]
        print('D',data[0], "\t\t", data[1],"\t" ,data[2])
```

```
Selenium word occurs in the following position
Word 'Selenium'
Doc no  no.of times  offset number
D 1              1       [0]
Word 'Web'
Doc no  no.of times  offset number
D 1              1       [7]
```

In [ ]:
```
print("Word 'soup'")
print("Doc no  no.of times  offset number")
for word in inverted_index.keys():
    if word=="soup".lower():
        data = inverted_index[word][0]
        print(data[0], "\t\t", data[1],"\t" ,data[2])
```

```
Word 'soup'
Doc no  no.of times  offset number
2                1       [1]
```

In [ ]:
```
print("Word 'Python' and 'Java' ")
print("Doc no  no.of times  offset number")
for word in inverted_index.keys():
    if word=="python".lower() or word=="java".lower() :
        data = inverted_index[word][0]
        print('D', data[0], "\t\t", data[1],"\t" ,data[2])
```

```
Word 'Python' and 'Java'
Doc no  no.of times  offset number
D 3             1       [3]
D 4             1       [0]
```

In [ ]:
```
print("Word 'Web and Craw'")
print("Doc no  no.of times  offset number")
for word in inverted_index.keys():
    if word=="web".lower() and word=="craw".lower():
        data = inverted_index[word][0]
        print('D', data[0], "\t\t", data[1],"\t" ,data[2])
```

```
Word 'Web and Craw'
Doc no  no.of times  offset number
```

PART B

Boolean and Vector Model, TF-IDF, Similarity Measures

In [ ]:
```
doc1 = [
    "Information Retrieval Systems is used with database systems",
    "Information is in Storage",
    "Digital Speech can be used in Synthesis and Systems",
    "Speech Filtering, Speech Retrieval systems are applications of Information Retr
    "Database Management system is used for storage"
]
```

In [ ]:
```
import numpy as np
import pandas as pd
```

In [ ]:
```
key = set()

for i in doc1:
    vocab_each = set(preprocessing(i))
    for j in vocab_each:
        key.add(j)
key = list(key)
print(key)
```

```
['synthesis', 'information', 'system', 'database', 'management', 'digital', 'storag
e', 'speech', 'retrieval', 'filtering', 'applications', 'systems', 'used']
```

In [ ]:
```
boolean_table = dict()
isthere1 = list()
for i in key:
    isthere = [0]*len(doc1)
    for j in range(len(doc1)):
        if i in doc1[j]:
            isthere[j] = 1
    isthere1.append(isthere)
```

```
for i in range(len(key)):
    boolean_table[key[i]] = isthere1[i]

df = pd.DataFrame(boolean_table)
df
```

Out[ ]:

	synthesis	information	system	database	management	digital	storage	speech	retrieval	filteri
0	0	0	1	1	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	
3	0	0	1	0	0	0	0	0	0	
4	0	0	1	0	0	0	1	0	0	

a. Retrieve the documents for the Boolean query "Information Retrieval Synthesis" using simple match. (Rank the documents in the order of relevance)

In [ ]:

```
query = "Information Retrieval Synthesis".lower().split()

res = df[(df[query[0]] == 1) & (df[query[1]] & (df[query[2]]))]


if(len(res) == 0):
    print("No match")
else:
    for i in list(res.index):
        print(f"Match found at Doc: {i}")
```

```
No match
```

b.Retrieve the documents for the Boolean query "Database Retrieval Storage" using weighted match. (Rank the documents in the order of relevance)

In [ ]:

```
isthere1 = list()
query = "Database Retrieval Storage"
query = preprocessing(query)

doc1 = [
    "Information Retrieval Systems is used with database systems",
    "Information is in Storage",
    "Digital Speech can be used in Synthesis and Systems",
    "Speech Filtering, Speech Retrieval systems are applications of Information Retr
    "Database Management system is used for storage"
]

s = set(' '.join(doc1).split(" "))
p = set(stopwords.words("english"))
l = []
for i in s:
    if i not in p:
        l.append(i)

d=dict()
for i in range(len(doc1)):
    for j in range(len(l)):
        if i not in d.keys():
            d[i] = [0] * len(l)
```

```
        if l[j] in doc1[i]:
            d[i][j] = 1

query = "Database Retrieval Storage"
p = [0] * len(l)
q =query.split(" ")
for i in range(len(l)):
    if l[i] in q:
        p[i] = 1

ans = []
for i in d.keys():
    m = []
    for j in range(len(p)):
        m.append(d[i][j] and p[j])
    ans.append(m.count(l))

import numpy as np
t = np.array(ans)
ans= list(t.argsort()[::-1])
ans
```

Out[ ]: [4, 3, 2, 1, 0]

Vector Model

iii. Construct a vector space model to build the term weights. Compute the TF-IDF and identify the most important terms across the documents.

In [ ]:

```
import math
import copy
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
nltk.download("punkt")
nltk.download('stopwords')
docs=["Information Retrieval Systems is used with database systems",
"Information is in Storage",
"Digital Speech can be used in Synthesis and Systems",
"Speech Filtering, Speech Retrieval systems are applications of Information Retrieva
"Database Management system is used for storage"
]
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\jayde\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\jayde\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [ ]:

```
def preprocess(doc,indx):
    doc=doc.lower()
    word_tokens = word_tokenize(doc)
    stop_words = set(stopwords.words('english'))
    l=list()
    for i in range(0,len(word_tokens)):
        if(word_tokens[i]==',' or word_tokens[i].lower() in stop_words or word_token
            continue
        else:
            l.append(word_tokens[i].lower())
    return (" ").join(l)
```

In [ ]:
```
for i in range(0,len(doc1)):
    doc1[i]=preprocess(doc1[i],i)
```

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()

response = tfidf.fit_transform(doc1)
tdidfTable = pd.DataFrame(response.toarray())

tdidfTable.columns = [keys for keys in tfidf.vocabulary_.keys()]

tdidfTable
```

Out[ ]:

	information	retrieval	systems	used	database	storage	digital	speech	synthesis	filter
0	0.00000	0.403755	0.000000	0.00000	0.335153	0.000000	0.403755	0.000000	0.000000	0.000
1	0.00000	0.000000	0.000000	0.00000	0.638711	0.000000	0.000000	0.000000	0.769447	0.000
2	0.00000	0.000000	0.530899	0.00000	0.000000	0.000000	0.000000	0.428326	0.000000	0.530
3	0.35127	0.000000	0.000000	0.35127	0.235249	0.000000	0.566804	0.566804	0.000000	0.000
4	0.00000	0.416607	0.000000	0.00000	0.000000	0.516374	0.000000	0.000000	0.416607	0.000

In [ ]:
```
# identify the most important terms across the documents
feature_names = tfidf.get_feature_names_out()
feature_names.tolist()
```

Out[ ]:
```
['applications',
 'database',
 'digital',
 'filtering',
 'information',
 'management',
 'retrieval',
 'speech',
 'storage',
 'synthesis',
 'system',
 'systems',
 'used']
```

In [ ]:
```
# Rank all the documents in the collection for the query “Speech Systems”?
query = preprocessing("Speech Systems")
query = " ".join(query)
query_vector = tfidf.transform([query]).toarray()
```

In [ ]:
```
from sklearn.metrics.pairwise import cosine_similarity
doc1_vector = tfidf.transform(doc1)
cosineSimilarity = cosine_similarity(doc1_vector, query_vector).flatten()

print(cosineSimilarity.argsort()[:-10:-1])
for i in cosineSimilarity.argsort()[:-10:-1]:
    print(doc1[i])
```

```
[3 2 0 4 1]
speech filtering speech retrieval systems applications information retrieval
digital speech used synthesis systems
information retrieval systems used database systems
database management system used storage
information storage
```

Compute the cosine similarities between docs 1 and docs 2

In [ ]:
```
import numpy as np
def cosine_similarity(x, y):
    if len(x) != len(y):
        return None
    dot_product = np.dot(x,y)
    mag_x = np.sqrt(np.sum(x**2))
    mag_y = np.sqrt(np.sum(y**2))
    np.sqrt(np.sum(x**2))
    np.sqrt(np.sum(y**2))
    cosine_similarity = dot_product / (mag_x * mag_y)
    return cosine_similarity
```

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer

Docs = [
    "Information Retrieval Systems is used with database systems",
    "Information is in Storage",
    "Digital Speech can be used in Synthesis and Systems",
    "Speech Filtering, Speech Retrieval systems are applications of Information Retr
    "Database Management system is used for storage"
]

x = CountVectorizer().fit_transform(Docs).toarray()


cos_sim = cosine_similarity(x[0,: ], x[1,: ])

print(cos_sim)
```

```
0.31622776601683794
```

Compute Dice Co-efficient between docs 3 and docs 4.

In [ ]:
```
def dice(x, y):
    x = x.astype(bool)
    y = y.astype(bool)
    sum_= x.sum() + y.sum()
    if sum_== 0:
        return 1
    intersection= np.logical_and(x,y)
    return 2. *intersection.sum()/ sum_
```

In [ ]:
```
d = dice(x[2,: ], x[3,: ])
d
```

Out[ ]:
```
0.23529411764705882
```

Compute the Jaccard co-efficient between docs 4 and docs 5.

In [ ]:
```
def jaccard(x, y):
    intersection= np.logical_and(x,y)
```

```
    union= np.logical_or(x,y)
    similarity= intersection.sum()/ float(union.sum())
    return similarity
```

In [ ]:

```
j = jaccard(x[3,: ], x[ 4,: ])
j
```

Out[ ]: 0.0