

**RAMAIAH INSTITUTE OF TECHNOLOGY**

(Autonomous Institute, Affiliated to VTU)

Bangalore – 560054

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**Course:** Information Retrieval **Course Code:** CSE13

**Lab Record**

**SUBMITTED BY**

|  |  |
| --- | --- |
| **USN** | **Name** |
| 1MS16CS044 | Sidharth Nambiar |
| 1MS16CS052 | Lakshya Sharma |

**Term:** Jan - May 2019

****

**Ramaiah Institute of Technology**

**(Autonomous Institute, Affiliated to VTU)**

**Department of CSE**

**Course:** Information Retrieval **Term:** Jan - May 2019

**Course Code:** CSE13

INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| **SL. NO** | **Lab Assignment/Project/Report** | **Mark/s Assigned** | **Mark/s**  **Obtained** |
|  | Implementation of string-matching algorithms: KMP and Rabin Karp. | 1 |  |
|  | Implementation of Pre-processing of a Text Document | 1 |  |
|  | Implementation of Inverted index: Construction and searching | 1 |  |
|  | Implementation of Vector Space Model.   1. Rank 10 documents for a given query. 2. Computing Similarity between any two documents. | 1 |  |
|  | Implementation of probabilistic Model.  Rank 10 documents for a given query. | 1 |  |
|  | Implementation of various evaluation measures.   1. Calculate recall and precision values for all relevant documents and draw precision VS Recall Curve. Also calculate R-precision. 2. Compare performance of two IR algorithms for the same query q. 3. Calculate harmonic mean and E- measure (All three cases b=1, b>1, b<1). | 1 |  |
|  | Implementation of a Web Crawler. | 2 |  |
|  | Lab Project: (Please mention your title) | 10 |  |
|  | Lab Record | 02 |  |
| **Total** | | 20 |  |

**Signature of Faculty**

1. Implementation of KMP ad Rabin Karp string matching algorithms

def KMPSearch(pat, txt):

M = len(pat)

N = len(txt)

lps = [0] \* M

j = 0

computeLPSArray(pat, M, lps)

i = 0

while i < N:

if pat[j] == txt[i]:

i += 1

j += 1

if j == M:

print("Found pattern at index " + str(i - j))

j = lps[j - 1]

elif i < N and pat[j] != txt[i]:

if j != 0:

j = lps[j - 1]

else:

i += 1

def computeLPSArray(pat, M, lps):

len = 0

lps[0]

i = 1

while i < M:

if pat[i] == pat[len]:

len += 1

lps[i] = len

i += 1

else:

if len != 0:

len = lps[len - 1]

else:

lps[i] = 0

i += 1

txt = "ABABDABACDABABCABAB"

pat = "ABABCABAB"

KMPSearch(pat, txt)

d=256;

def search(pat, txt, q):

M = len(pat)

N = len(txt)

i = 0

j = 0

p = 0

t = 0

h = 1

for i in range(M - 1):

h = (h \* d) % q

for i in range(M):

p = (d \* p + ord(pat[i])) % q

t = (d \* t + ord(txt[i])) % q

for i in range(N - M + 1):

if p == t:

for j in range(M):

if txt[i + j] != pat[j]:

break

j += 1

if j == M:

print ("Pattern found at index " + str(i))

if i < N - M:

t = (d \* (t - ord(txt[i]) \* h) + ord(txt[i + M])) % q

if t < 0:

t = t + q

txt = "GEEKS FOR GEEKS"

pat = "GEEK"

q = 101

search(pat, txt, q)

1. Implementation of Pre-processing of a Text Document

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

file = [None]\*10

def stem\_tokens(tokens, stemmer):

stemmed = []

for item in tokens:

stemmed.append(stemmer.stem(item))

return stemmed

stop\_words = set(stopwords.words('english'))

for i in range(10):

file[i] = open("T"+str(i+1)+".txt",encoding="ISO-8859-1")

line = file[i].read()

text = "".join([ch for ch in line if ch not in [",",".","(",")","-",":","?","#","[","]","\*",";","§","\"","`","/","\_","'",

"1","2","3","4","5","6","7","8","9","0","!","{","}"]])

tokens = word\_tokenize(text)

stems = stem\_tokens(tokens, stemmer)

for r in stems:

if not r in stop\_words:

appendFile = open('f-t'+str(i+1)+'.txt', 'a')

appendFile.write(" "+r)

appendFile.close()

def freq(str1):

str\_list = str1.split()

unique\_words = set(str\_list)

for words in unique\_words:

appendFile = open('freq-t'+str(i+1)+'.txt', 'a')

words = words.lower()

appendFile.write( words + ':'+str(str\_list.count(words)+1)+"\n" )

appendFile.close()

for i in range(10):

file[i] = open("f-t"+str(i+1)+".txt",encoding="ISO-8859-1")

line = file[i].read()

words = word\_tokenize(line)

freq(line)

1. Implementation of Inverted index: Construction and searching

keywords=[]

for i in range(10):

searchfile = open("freq-t"+str(i+1)+".txt", "r")

for line in searchfile:

line=line.split(":")

if line[0] not in keywords:

keywords.append(line[0])

searchfile.close()

#creation of inverted index

ii=open("invertedindex.txt", "a")

for word in keywords:

line=word+ " : "

for i in range(10):

searchfile = open("freq-t"+str(i+1)+".txt", "r")

for line1 in searchfile:

if word in line1:

line=line+" T"+str(i+1)+","

break

searchfile.close()

ii.write(line+"\n")

ii.close()

#search function

def search\_word(word):

file=open("invertedindex.txt","r")

for line in file:

line = line.split(":")

if word in line[0]:

print(line[1])

break

file.close()

word=input("Enter a word : ")

search\_word(word)

1. Implementation of Vector Space Model.
2. Rank 10 documents for a given query.
3. Computing Similarity between any two documents.

import math

import operator

n=10

documents=[]

keywords=[]

for i in range(10):

searchfile = open("freq-t"+str(i+1)+".txt", "r")

doci=[]

for line in searchfile:

line=line.split(":")

doci.append(line[0])

documents.append(doci)

searchfile.close()

print(documents)

def freqs(str1):

#str\_list = str1.split()

dict1=dict()

unique\_words = set(str1)

for words in unique\_words:

words = words.lower()

dict1[words]=str1.count(words)

return dict1

for i in documents:

keywords=keywords+i

print(keywords)

freq=freqs(keywords)

print(freq)

idf=dict()

#for x in freq:

# idf[x]=math.log(n/freq[x],2)

tf\_idf=dict()

for x in freq:

if freq[x]==0:

continue

tf\_idf\_x=[]

idf[x] = math.log(n / freq[x], 2)

for i in range(n):

if x in documents[i]:

tf\_idf\_x.append(1\*idf[x])

else:

tf\_idf\_x.append(0)

tf\_idf[x]=tf\_idf\_x

print("tf-idf :")

print(tf\_idf)

query=input("Enter query :").split(" ")

freq\_q=freqs(query)

tf\_idf\_q=dict()

for y in freq:

if y in query:

tf\_idf\_q[y]=((freq\_q[y]/max(freq\_q.values()))\*idf[y])

else:

tf\_idf\_q[y]=0

print(tf\_idf\_q)

def Length(param):

sum=0

for i in param:

sum=sum+(i\*i)

return math.sqrt(sum)

doclength=[]

docidf=[]

for i in range(n):

temp=[]

for x in tf\_idf:

temp.append(tf\_idf[x][i])

docidf.append(temp)

doclength.append(Length(docidf[i]))

q\_length=Length(list(tf\_idf\_q.values()))

print("Document and query lengths :")

print(doclength)

print(q\_length)

sim={}

for i in range(n):

sum=0

for x in tf\_idf:

sum=sum+(tf\_idf[x][i]\*tf\_idf\_q[x])

sum=sum/(doclength[i]\*q\_length)

sim[i+1]=sum

print("Ranking of documents : ")

print(sorted(sim.items(),key=operator.itemgetter(1),reverse=True))

sim\_matrix=[]

for i in range(n):

sim\_i = []

for j in range(n):

sum=0

if i==j:

sum=1

sim\_i.append(sum)

continue

for x in tf\_idf:

sum = sum + (tf\_idf[x][i] \* tf\_idf[x][j])

sum = sum / (doclength[i] \* doclength[j])

sim\_i.append(sum)

sim\_matrix.append(sim\_i)

print ("Similarity matrix : ")

for i in sim\_matrix:

print(i)

1. Implementation of probabilistic Model. Rank 10 documents for a given query.

import operator

N=10

documents=[]

for i in range(10):

searchfile = open("freq-t"+str(i+1)+".txt", "r")

doci=[]

for line in searchfile:

line=line.split(":")

doci.append(line[0])

documents.append(doci)

searchfile.close()

print(documents)

Nw=dict()

docs=[]

for i in range(10):

docs=docs+documents[i]

docs=list(set(docs))

def compute(Nw):

return (N-Nw+0.5)/(Nw+0.5)

table=dict()

for x in docs:

nw=0

list1=[]

for i in documents:

if x in i:

nw+=1

list1.append(nw)

calc=compute(nw)

list1.append(calc)

table[x]=list1

print(table)

query=input("Enter query :").split(" ")

pd=dict()

for i in range(N):

pdi=1

for x in query:

if x in documents[i]:

pdi=pdi\*table[x][1]

pd[i+1]=pdi

sorted\_x = sorted(pd.items(), key=operator.itemgetter(1),reverse=True)

print(sorted\_x)

1. Implementation of various evaluation measures.
2. Calculate recall and precision values for all relevant documents and draw precision VS Recall Curve. Also calculate R-precision.
3. Compare performance of two IR algorithms for the same query q.
4. Calculate harmonic mean and E- measure (All three cases b=1, b>1, b<1).

rQ = [3, 5, 9, 25, 39, 44, 56, 71, 89, 94, 105, 119, 124, 136, 144]

aQ = [123, 84, 56, 6, 8, 777, 511, 129, 187, 25, 38, 48, 250, 113, 44, 99, 95, 214, 136, 39, 128, 71, 14, 5]

p = {}

r = {}

relevant = 0

for i in range(len(aQ)):

doc = aQ[i]

if (doc in rQ):

relevant += 1;

p[doc] = relevant / (i + 1) \* 100

r[doc] = relevant / len(rQ) \* 100

print("Precision of relevant docs : \n", p)

print("Recall of relevant docs \n: ", r)

keysR = r.keys()

keys30 = []

keys60 = []

keys90 = []

values = []

for i in keysR:

if (r[i] <= 30.0):

keys30.append(i)

if (r[i] > 30.0 and r[i] <= 60.0):

keys60.append(i)

if (r[i] > 60.0 and r[i] <= 90.0):

keys90.append(i)

for i in keys30:

values.append(p[i])

if not values:

print("Interpolated precision at 30% is : 0")

else:

print("Interpolated precision at 30% is : ", max(values))

values.clear()

for i in keys60:

values.append(p[i])

if not values:

print("Interpolated precision at 60% is : 0")

else:

print("Interpolated precision at 60% is : ", max(values))

values.clear()

for i in keys90:

values.append(p[i])

if not values:

print("Interpolated precision at 90% is : 0")

else:

print("Interpolated precision at 90% is : ", max(values))

count=0

for i in range(len(rQ)):

if aQ[i] in rQ:

count+=1

print("R Precision : " + str(count\*100/(len(rQ))))

def harmonic\_mean(a,b):

return 2/((100/a)+(100/b))

hm={}

for doc in p:

hm[doc]=harmonic\_mean(p[doc],r[doc])

print("Harmonic Mean :")

print(hm)

b=[0.5,1,2]

def Emeasure(r,p,b):

Em=[]

for i in b:

if i<1:

Em.append(1-((1+i\*i)/((i\*i\*100)/r)+(100/p)))

if i==1:

Em.append(1-harmonic\_mean(r,p))

if i>1:

Em.append(1 - ((1 + i \* i) / ((i \* i) / r) + (1 / p)))

return Em

Em={}

for doc in p:

Em[doc]=Emeasure(r[doc],p[doc],b)

print("E measure : ")

print(Em)

1. Implementation of a Web Crawler.

import requests

from bs4 import BeautifulSoup

import csv

URL = "http://www.passiton.com/inspirational-quotes"

r = requests.get(URL)

soup = BeautifulSoup(r.content, 'html5lib')

quotes=[] # a list to store quotes

table = soup.find('div', attrs = {'id':'wrapper'})

# print(table)

for row in table.findAll('div', attrs = {'class':'portfolio-image'}):

quote = {}

# quote['theme'] = row.h5.text

quote['url'] = row.a['href']

quote['img'] = row.img['alt'].split('#')[0]

print("Quote : ",quote['img'])

# quote['lines'] = row.h6.text

# quote['author'] = row.p.text

quotes.append(quote)

for item in quotes:

print(item)

1. Lab Project: Develop an information extraction system that learns a particular kind of fact from constructed documents (e.g. crimes: perpetrator, victim, date, officers involved)

**import** spacy

**import** pickle

**from** gensim **import** corpora

**import** gensim

**import** warnings

warnings.filterwarnings('ignore')

spacy.load('en')

**from** spacy.lang.en **import** English

parser = English()

#Text Cleaning

**def** tokenize(text):

lda\_tokens = []

tokens = parser(text)

**for** token **in** tokens:

**if** token.orth\_.isspace():

**continue**

**elif** token.like\_url:

lda\_tokens.append('URL')

**elif** token.orth\_.startswith('@'):

lda\_tokens.append('SCREEN\_NAME')

**else**:

lda\_tokens.append(token.lower\_)

**return** lda\_tokens

**import** nltk

**from** nltk.corpus **import** wordnet **as** wn

**def** get\_lemma(word):

lemma = wn.morphy(word)

**if** lemma **is** None:

**return** word

**else**:

**return** lemma

**from** nltk.stem.wordnet **import** WordNetLemmatizer

**def** get\_lemma2(word):

**return** WordNetLemmatizer().lemmatize(word)

en\_stop = set(nltk.corpus.stopwords.words('english'))

**def** prepare\_text\_for\_lda(text):

tokens = tokenize(text)

tokens = [token **for** token **in** tokens **if** len(token) > 4]

tokens = [token **for** token **in** tokens **if** token **not** **in** en\_stop]

tokens = [get\_lemma(token) **for** token **in** tokens]

**return** tokens

**import** random

text\_data = []

**with** open('D:\\Documents\\DataSets\\MillionNewsHeadlines\\abcnews-date-text.csv') **as** f:

**for** line **in** f:

tokens = prepare\_text\_for\_lda(line)

**if** random.random() > .99:

**print**(tokens)

text\_data.append(tokens)

#LDA

**from** gensim **import** corpora

dictionary = corpora.Dictionary(text\_data)

corpus = [dictionary.doc2bow(text) **for** text **in** text\_data]

pickle.dump(corpus, open('corpus.pkl', 'wb'))

dictionary.save('dictionary.gensim')

**with** open('D:\\Documents\\Programs\\Python\\corpus.pkl', 'rb') **as** f:

corpus = pickle.load(f)

dictionary = gensim.corpora.Dictionary.load('D:\\Documents\\Programs\\Python\\dictionary.gensim')

NUM\_TOPICS = 20

ldamodel = gensim.models.ldamodel.LdaModel(corpus, num\_topics = NUM\_TOPICS, id2word=dictionary, passes=30)

ldamodel.save('D:\\Documents\\Programs\\Python\\model50.gensim')

topics = ldamodel.print\_topics(num\_words=4)

**for** topic **in** topics:

**print**(topic)

ldamodel = gensim.models.ldamodel.LdaModel(corpus, num\_topics = 20, id2word=dictionary, passes=35)

ldamodel.save('D:\\Documents\\Programs\\Python\\model20.gensim')

topics = ldamodel.print\_topics(num\_words=4)

**for** topic **in** topics:

**print**(topic)

new\_doc = 'After a 30 day long trial at the supreme court of india, the suspected rapist were released from jail on bail for a bail amount of 50000. The victims family'

new\_doc = prepare\_text\_for\_lda(new\_doc)

new\_doc\_bow = dictionary.doc2bow(new\_doc)

**print**(new\_doc\_bow)

**print**(ldamodel.get\_document\_topics(new\_doc\_bow))

**import** pyLDAvis.gensim

lda20 = gensim.models.ldamodel.LdaModel.load('D:\\Documents\\Programs\\Python\\model20.gensim')

lda\_display10 = pyLDAvis.gensim.prepare(lda20, corpus, dictionary, sort\_topics=False)

pyLDAvis.display(lda\_display20)

**Explanation**

Topic modelling is a type of statistical modelling for discovering the abstract “topics” that occur in a collection of documents. Latent Dirichlet Allocation (LDA) is an example of topic model and is used to classify text in a document to a particular topic. It builds a topic per document model and words per topic model, modelled as Dirichlet distributions.

The following is done:

1. The data used is a list of over one million news headlines published over a period of 15 years downloaded from Kaggle.
2. Data Pre Processing
3. Tokenization: Split the text into sentences and the sentences into words.
4. Lowercase the words and remove punctuation.
5. Words that have fewer than 3 characters are removed.
6. All stopwords are removed.
7. Words are lemmatized — words in third person are changed to first person and verbs in past and future tenses are changed into present.
8. Words are stemmed — words are reduced to their root form.
9. We filter out tokens that appear in
   * + - 1. Less than 15 documents (absolute number) or
         2. more than 0.5 documents (fraction of total corpus size, not absolute number).
         3. after the above two steps, we keep only the first 100000 most frequent tokens.
10. For each document, we create a dictionary reporting how many words and how many times those words appear. Save this to ‘bow\_corpus’, then check our selected document earlier.
11. We create tf-idf model object using models. TfidfModel on ‘bow\_corpus’ and save it to ‘tfidf’, then apply transformation to the entire corpus and call it ‘corpus\_tfidf’.
12. We train our lda model using gensim and save it to ‘lda\_model’
13. We use pyLDAvis to extract information from a fitted LDA topic model to inform an interactive web-based visualization.