

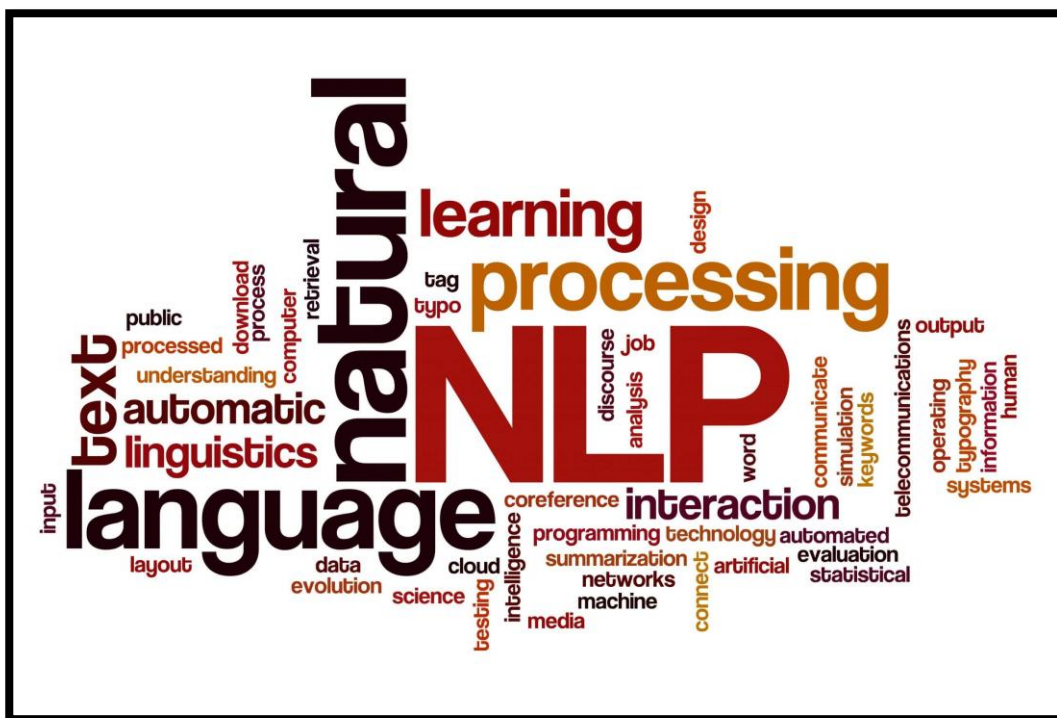


GLA UNIVERSITY MATHURA

Session: 2022-23

Subject - NATURAL LANGUAGE PROCESSING

ASSIGNMENT



Department of Computer Science & Applications
Institute of Engineering & Technology

Submitted to: -

Dr Swati Srivastava

Submitted by: -

Himanshu Sharma (N-59)

(201550062)

V Semester

1. Consider an example to find the similarity between two vectors – ‘x’ and ‘y’, using Cosine Similarity.

The ‘x’ vector has values, $x = \{ 3, 2, 0, 5 \}$

The ‘y’ vector has values, $y = \{ 1, 0, 0, 0 \}$

Design a program to compute the similarity between x and y.

```
[1] import math
```

```
▶ x=[3,2,0,5]
  y=[1,0,0,0]

  # Cosine Similarity

  def SOS(x):
      res=0
      for i in x:
          res+=(i*i)
      return res

  def dotProd(x,y):
      res=0
      for i in range(len(x)):
          res+=(x[i]*y[i])
      return res

  def CS(x,y):
      return dotProd(x,y)/math.sqrt(SOS(x)*SOS(y))

  print("Cosine Similarity between x & y is",CS(x,y))
```

```
📄 Cosine Similarity between x & y is 0.48666426339228763
```

2. Consider two sets $A = (0,1,2,5,6)$ and $B = (0, 2,3, 5, 7,9)$. How similar are A and B? Design a program to compute the similarity using Jaccard similarity.

```
▶ A=[0,1,2,5,6]
  B=[0,2,3,5,7,9]

#Jaccard Similarity

def JS(a,b):
    setA=set(a)
    setB=set(b)
    return len(setA.intersection(setB))/len(setA.union(setB))

print("Similarity between A & B is",JS(A,B))
```

☞ Similarity between A & B is 0.375

3. Consider Three documents:

$d1$: "Jack London traveled to Oakland"

$d2$: "Jack London traveled to the city of Oakland"

$d3$: "Jack traveled from Oakland to London"

Based on shingles of size 2 (2-grams or bigrams), what are the Jaccard coefficients $J(d1, d2)$ and $J(d1, d3)$?

```
from nltk.tokenize import word_tokenize
from nltk.util import bigrams

d1="Jack London traveled to Oakland"
d2="Jack London traveled to the city of Oakland"
d3="Jack traveled from Oakland to London"

# doc to Bigrams
def d2bigram(d):
    return set(bigrams(word_tokenize(d)))

# Jaccard Coefficient
def JC(d1,d2):
    bgram_d1=d2bigram(d1)
    bgram_d2=d2bigram(d2)
    return len(bgram_d1.intersection(bgram_d2))/len(bgram_d1.union(bgram_d2))

print("J(d1, d2) =",JC(d1,d2))
print("J(d1, d3) =",JC(d1,d3))

J(d1, d2) = 0.375
J(d1, d3) = 0.0
```

4. Consider set of documents as

D1 : I am Sam.

D2 : Sam I am.

D3 : I do not like green eggs and ham.

D4 : I do not like them, Sam I am.

- Design a program for (k = 1)-shingles of D1 U D2 U D3 U D4 : U is UNION
- Design a program for (k = 2)-shingles of D1 U D2 U D3 U D4 : U is UNION
- Design a program for (k = 3)-Character shingles of D1 U D2
- Design a program for (k = 4)-Character shingles of D1 U D2

```
from nltk.tokenize import word_tokenize
from nltk.util import ngrams

d1="I am Sam."
d2="Sam I am."
d3="I do not like green eggs and ham."
d4="I do not like them, Sam I am."

# doc to ngrams
def d2ngram(d,n):
    return set(ngrams(word_tokenize(d),n))

# doc to k character shingles
def d2k_shingle(d,k):
    return set(ngrams(d,k))

# union of results
def d_s2union(docs):
    res=set()
    for i in docs:
        res=res.union(i)
    return res

print('a')
l=[]
n=1
l.append(d2ngram(d1,n))
l.append(d2ngram(d2,n))
l.append(d2ngram(d3,n))
l.append(d2ngram(d4,n))
print(d_s2union(l))

print('\nb')
l=[]
n=2
l.append(d2ngram(d1,n))
l.append(d2ngram(d2,n))
l.append(d2ngram(d3,n))
l.append(d2ngram(d4,n))
print(d_s2union(l))

print('\nc')
l=[]
k=3
l.append(d2k_shingle(d1,k))
l.append(d2k_shingle(d2,k))
print(d_s2union(l))

print('\nd')
l=[]
k=4
l.append(d2k_shingle(d1,k))
l.append(d2k_shingle(d2,k))
print(d_s2union(l))
```

Output:

- {('not',), ('and',), ('.',), (','), ('like',), ('I',), ('eggs',), ('am',), ('them',), ('green',), ('Sam',), ('ham',), ('do',))}
- {('I', 'am'), ('I', 'do'), ('and', 'ham'), ('am', '.'), ('do', 'not'), ('not', 'like'), ('eggs', 'and'), ('Sam', '.'), ('like', 'green'), (',', 'Sam'), ('like', 'them'), ('am', 'Sam'), ('ham', '.'), ('Sam', 'I'), ('them', ','), ('green', 'eggs')}
- {('I', ' ', 'a'), (' ', 'S', 'a'), ('a', 'm', '.'), (' ', 'a', 'm'), ('S', 'a', 'm'), ('a', 'm', ' '), ('m', ' ', 'S')}
- {('I', ' ', 'a', 'm'), (' ', 'a', 'm', ' '), (' ', 'S', 'a', 'm'), ('a', 'm', ' ', 'S'), ('m', ' ', 'S', 'a'), ('S', 'a', 'm', ' ')}

5. Suppose our document D is the string abcdabd, and we pick $k = 2$. Then the set of 2-shingles for D is {ab, bc, cd, da, bd}. Note that the substring ab appears twice within D, but appears only once as a shingle. A variation of shingling produces a bag, rather than a set, so each shingle would appear in the result as many times as it appears in the document. Design a program to model the above scenario.

```
from nltk.util import ngrams

d="abcdabd"

# doc to k character shingles with repetition
def d2k_shingle(d,k):
    return list(ngrams(d,k))

print(d2k_shingle(d,2))
```

↳ [('a', 'b'), ('b', 'c'), ('c', 'd'), ('d', 'a'), ('a', 'b'), ('b', 'd')]

6. A database contains 80 records on a particular topic. A search was conducted on that topic and 60 records were retrieved. Of the 60 records retrieved, 45 were relevant. Design a Program to compute the precision and recall scores for the search.

```
#precision
def precision(retrieved,relevant):
    return (relevant/retrieved)*100

#recall
def recall(total,relevant):
    return (relevant/total)*100

#func
def fun(total,retrieved,relevant):
    print("Total Records =",total)
    print("Retrieved Records =",retrieved)
    print("Relevant Records Retrieved =",relevant)
    print()
    print("Precision =",precision(retrieved,relevant),"%")
    print("Recall =",recall(total,relevant),"%")

total=80
retrieved=60
relevant=45

fun(total,retrieved,relevant)
```

↳ Total Records = 80
Retrieved Records = 60
Relevant Records Retrieved = 45

Precision = 75.0 %
Recall = 56.25 %

7. Consider this sentence: “a rose is a rose is a rose”. And represent this document as set of shingles (Word n gram). Design a program for extracting shingles for $n = 3$ and $n = 4$ for above sentence.

```
from nltk.tokenize import word_tokenize
from nltk.util import ngrams

d="a rose is a rose is a rose"

# doc to ngrams
def d2ngram(d,n):
    return set(ngrams(word_tokenize(d),n))

print("Word ngram for n=3:")
print(d2ngram(d,3))
print()
print("Word ngram for n=4:")
print(d2ngram(d,4))
```

```
Word ngram for n=3:
{('a', 'rose', 'is'), ('is', 'a', 'rose'), ('rose', 'is', 'a')}

Word ngram for n=4:
{('is', 'a', 'rose', 'is'), ('rose', 'is', 'a', 'rose'), ('a', 'rose', 'is', 'a')}
```

8. Compute the similarity between:

- a) night and nacht
- b) Ashish and Aasheesh

```
[11] from nltk.util import ngrams
```

```
# doc to k character shingles
def d2k_shingle(d,k):
    return set(ngrams(d,k))

# Jaccard Similarity
def JS(d1,d2,n):
    nshingle_d1=d2k_shingle(d1,n)
    nshingle_d2=d2k_shingle(d2,n)
    return len(nshingle_d1.intersection(nshingle_d2))/len(nshingle_d1.union(nshingle_d2))

print("a")
print("Similarity between night and nacht is",JS("night","nacht",2))

print("\nb")
print("Similarity between Ashish and Aasheesh is",JS("Ashish","Aasheesh",2))
```

```
a)
Similarity between night and nacht is 0.14285714285714285

b)
Similarity between Ashish and Aasheesh is 0.11111111111111111
```

9. Consider a case insensitive query and document collection with a query Q and a document collection consisting of the following three documents:

Q: "gold silver truck"

D1: "Shipment of gold damaged in a fire"

D2: "Delivery of silver arrived in a silver truck"

D3: "Shipment of gold arrived in a truck"

Design a program to compute the similarity between

a) Q & D1

b) Q & D2

c) Q & D3

Find which document (D) matches best with the query (Q)

```
from nltk.tokenize import word_tokenize
from nltk.util import bigrams

q="gold silver truck"
d1="Shipment of gold damaged in a fire"
d2="Delivery of silver arrived in a silver truck"
d3="Shipment of gold arrived in a truck"

# doc to Bigrams
def d2bigram(d):
    return set(bigrams(word_tokenize(d)))

# Jaccard Coefficient
def JS(d1,d2):
    bgram_d1=d2bigram(d1)
    bgram_d2=d2bigram(d2)
    return len(bgram_d1.intersection(bgram_d2))/len(bgram_d1.union(bgram_d2))

#find best match
def findBestMatch(q,*d):
    res=0
    ans=""
    d_num=0
    for i in range(len(d)):
        doc=d[i]
        tmp=JS(q,doc)
        print("For JS(Q, D{}): {}".format(i+1,tmp))
        if tmp>res:
            res=tmp
            ans=doc
            d_num=i+1
    print("Document D{} matches best with the given Query.".format(d_num))

findBestMatch(q,d1,d2,d3)
```

```
For JS(Q, D1): 0.0
For JS(Q, D2): 0.125
For JS(Q, D3): 0.0
Document D2 matches best with the given Query.
```

Thank You!