Name : Devasy Patel

Roll No: 20BCE057

Title: Text Preprocessing using NLTK

AND , OR, NOT operations on boolean retrieval model

In [ ]:
```
import numpy as np

# Define the term-document incidence matrix
matrix = np.array([[1, 0, 1, 0, 1, 0, 0, 1, 0, 1],
                   [0, 1, 1, 0, 0, 1, 0, 1, 0, 1],
                   [0, 0, 0, 1, 0, 1, 0, 1, 1, 1],
                   [1, 1, 0, 1, 0, 0, 1, 0, 0, 0],
                   [0, 0, 0, 0, 1, 1, 1, 0, 0, 1]])

# Define the vocabulary of terms
vocabulary = ['term1', 'term2', 'term3', 'term4', 'term5']

# Define the documents
documents = ['doc1', 'doc2', 'doc3', 'doc4', 'doc5', 'doc6', 'doc7', 'doc8', 'doc9', 'doc10']



# Define the OR query
def OR_query(query):
    query_vector = np.zeros(matrix.shape[1], dtype=int)
    for term in query:
        if term in vocabulary:
            term_index = vocabulary.index(term)
            query_vector = np.logical_or(query_vector, matrix[term_index])
    return [documents[i] for i in range(len(query_vector)) if query_vector[i] == 1]

# Example usage
print(AND_query(['term1', 'term2']))
print(OR_query(['term1', 'term2']))


# Calculate sparsity value
sparsity = 1.0 - np.count_nonzero(matrix) / matrix.size
print("Sparsity:", sparsity)
```

```
[]
['doc1', 'doc2', 'doc3', 'doc5', 'doc6', 'doc8', 'doc10']
Sparsity: 0.54
```

In [ ]:
```
# now sentences are
sentences = ["Victory is ensured when there is learning involved",
"The youth of Rajasthan always come ahead of the rest when it comes to the security of the nation",
"The successful organisation of Jaipur Mahakhel is the next important link towards India's efforts",
"The country is forging new definitions and creating a new order in the Amrit Kaal"     ,
"The Sports Budget of the country has increased almost three times since 2014"  ,
"Sports universities are being set up in the country, and big events like Khel Mahakumbh are also being organised in a professional manner"     ,
"Our government is attentive that no youth should be left behind due to lack of money"  ,
"You will be fit, only then you will be superhit"       ,
"Rajasthan's Shree Anna-Bajra and Shree Anna-Jwala are the best examples of this"       ,
"The next gold and silver medalists for the country will emerge from among you"]
```

In [ ]:
```
def get_vocab(sentences):
    vocab = set()
    for sentence in sentences:
        for word in sentence.split():
            vocab.add(word)
    return list(vocab)

vocabulary = get_vocab(sentences)
vocabulary = vocab_preprocess(vocabulary)
print(vocabulary)
```

```
creating
come
definitions
will
almost
Anna-Jwala
and
Our
comes
Anna-Bajra
nation
Sports
events
The
medalists
2014
forging
for
You
Shree
Jaipur
are
of
from
Khel
since
lack
then
organisation
successful
should
next
country
a
left
to
rest
among
learning
also
always
due
new
emerge
money
Victory
towards
is
gold
in
India's
three
country,
universities
that
this
involved
silver
increased
times
security
has
Mahakumbh
being
youth
no
Mahakhel
efforts
big
like
when
Rajasthan's
link
up
best
manner
it
Kaal
ensured
organised
Rajasthan
only
there
be
the
order
important
behind
examples
set
ahead
attentive
fit,
you
professional
Budget
superhit
government
Amrit
['creating', 'come', 'definitions', 'will', 'almost', 'anna-jwala', 'and', 'our', 'comes', 'anna-bajra', 'nation', 'sports', 'events', 'the', 'medalists', '2014', 'forging', 'for', 'you', 'shree', 'jaipur', 'are', 'of', 'from', 'khel', 'since', 'lack', 'then', 'organisation', 'successful', 'should', 'next', 'country', 'a', 'left', 'to', 'rest', 'among', 'learning', 'also', 'always', 'due', 'new', 'emerge', 'money', 'victory', 'towards', 'is', 'gold', 'in', 'india', 'three', 'universities', 'that', 'this', 'involved', 'silver', 'increased', 'times', 'security', 'has', 'mahakumbh', 'being', 'youth', 'no', 'mahakhel', 'efforts', 'big', 'like', 'when', 'rajasthan', 'link', 'up', 'best', 'manner', 'it', 'kaal', 'ensured', 'organised', 'only', 'there', 'be', 'order', 'important', 'behind', 'examples', 'set', 'ahead', 'attentive', 'fit', 'professional', 'budget', 'superhit', 'government', 'amrit']
```

In [ ]:
```
def get_matrix(sentences, vocabulary):
    matrix = np.zeros((len(vocabulary), len(sentences)), dtype=int)
    for i, sentence in enumerate(sentences):
        for word in sentence.split():
            word = word.lower()
            word = word.replace("'s", "")
            word = word.replace(",", "")
            word = word.replace(".", "")
```

```
            matrix[vocabulary.index(word), i] = 1
    return matrix

matrix = get_matrix(sentences, vocabulary)
matrix.T
```

Out[ ]:
```
array([[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
         0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
         0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
         0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
         1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
         1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0],
        [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
         0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 1],
        [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
         1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
         0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
         0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
         0, 0, 1, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
         1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
         1, 0, 0, 0, 0, 1, 0],
        [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
         0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
         0, 1, 0, 0, 1, 0, 0],
        [0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
         1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
         0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
         0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0]]])
```

token, stemming, stopword removal, noramlization, lower-case, lametasization, etc. The results of the preprocessing are shown in Table 1.

In [ ]:
```
sparsity = 1.0 - np.count_nonzero(matrix) / matrix.size
print("Sparsity:", sparsity)
```

```
Sparsity: 0.8673684210526316
```

In [ ]:
```
# create vocabulary sprinter
def vocab_preprocess(vocabulary):
    vcb=[]
    for x in vocabulary:
        print(x)
        t = x.lower()
        t = t.replace(".", "")
        t = t.replace(",", "")
        t = t.replace("'s", "")
        t = t.replace("!", "")
        t = t.replace("?", "")
        if t not in vcb:
            vcb.append(t)
    return vcb

vocabulary = vocab_preprocess(vocabulary)
print(vocabulary)
```

```
creating
come
definitions
will
almost
anna-jwala
and
our
comes
anna-bajra
nation
sports
events
the
medalists
2014
forging
for
you
shree
jaipur
are
of
from
khel
since
lack
then
organisation
successful
should
next
country
a
left
to
rest
among
learning
also
always
due
new
emerge
money
victory
towards
is
gold
in
india
three
universities
that
this
involved
silver
increased
times
security
has
mahakumbh
being
youth
no
mahakhel
efforts
big
like
when
rajasthan
link
up
best
manner
it
kaal
ensured
organised
only
there
be
order
important
behind
examples
set
ahead
attentive
fit
professional
budget
superhit
government
amrit
['creating', 'come', 'definitions', 'will', 'almost', 'anna-jwala', 'and', 'our', 'comes', 'anna-bajra', 'nation', 'sports', 'events', 'the', 'medalists', '2014', 'forging', 'for', 'you', 'shree', 'jaipur', 'are', 'of', 'from', 'khel', 'since', 'lack', 'then', 'organisation', 'successful', 'should', 'next', 'country', 'a', 'left', 'to', 'rest', 'among', 'learning', 'also', 'always', 'due', 'new', 'emerge', 'money', 'victory', 'towards', 'is', 'gold', 'in', 'india', 'three', 'universities', 'that', 'this', 'involved', 'silver', 'increased', 'times', 'security', 'has', 'mahakumbh', 'being', 'youth', 'no', 'mahakhel', 'efforts', 'big', 'like', 'when', 'rajasthan', 'link', 'up', 'best', 'manner', 'it', 'kaal', 'ensured', 'organised', 'only', 'there', 'be', 'order', 'important', 'behind', 'examples', 'set', 'ahead', 'attentive', 'fit', 'professional', 'budget', 'superhit', 'government', 'amrit']
```

In [ ]:
```
# Define the AND query
def AND_query(query):
    query_vector = np.ones(matrix.shape[1], dtype=int)
    # print(query_vector)
    for term in query:
        if term in vocabulary:
            term_index = vocabulary.index(term)
            query_vector = np.logical_and(query_vector, matrix[term_index])
    return [documents[i] for i in range(len(query_vector)) if query_vector[i] == 1]
```

```
print(AND_query(['the', 'of']))
```

```
[1 1 1 1 1 1 1 1 1 1]
['doc2', 'doc3', 'doc5', 'doc9']
```

In [ ]:
```
def NOT_query(query):
    list = OR_query(query)
    xlist =[]
    for i in documents:
        if i not in list:
            xlist.append(i)
    return xlist
NOT_query(['of', 'the'])
```

Out[ ]:
```
['doc1', 'doc8']
```

INVERTED INDEX LABELS

In [ ]:
```

```