**PRACTICAL NO 2**

**AIM :** Retrieval Models

Implement the Boolean retrieval model and process queries.

Implement the vector space model with TF-IDF weighting and cosine similarity.

**SOLUTION:**

1) Implement the Boolean retrieval model and process queries

**INPUT:**

documents = {

1: "apple banana orange",

2: "apple banana",

3: "banana orange",

4: "apple",

}

def build\_index(docs):

index = {}

for doc\_id, text in docs.items():

for term in set(text.split()):

index.setdefault(term, set()).add(doc\_id)

return index

inverted\_index = build\_index(documents)

def boolean\_and(operands, index):

result = index.get(operands[0], set())

for term in operands[1:]:

result &= index.get(term, set())

return list(result)

def boolean\_or(operands, index, total\_docs):

result = set()

for term in operands:

result |= index.get(term, set())

return list(result | set(range(1, total\_docs + 1)))

def boolean\_not(operand, index, total\_docs):

return list(set(range(1, total\_docs + 1)) - index.get(operand, set()))

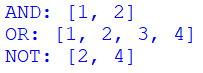
query1, query2, query3 = ["apple", "banana"], ["apple", "orange"], "orange"

print("AND:", boolean\_and(query1, inverted\_index))

print("OR:", boolean\_or(query2, inverted\_index, len(documents)))

print("NOT:", boolean\_not(query3, inverted\_index, len(documents)))

**OUTPUT:**



2) Implement the vector space model with TF-IDF weighting and cosine similarity.

**INPUT:**

from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer

import nltk

from nltk.corpus import stopwords

import numpy as np

from numpy.linalg import norm

train\_set = ["The sky is blue.", "The sun is bright."]

test\_set = ["The sun in the sky is bright."]

nltk.download('stopwords')

stopWords = stopwords.words('english')

vectorizer = CountVectorizer(stop\_words=stopWords)

trainVectorizerArray = vectorizer.fit\_transform(train\_set).toarray()

testVectorizerArray = vectorizer.transform(test\_set).toarray()

print('Fit Vectorizer to train set', trainVectorizerArray)

print('Transform Vectorizer to test set', testVectorizerArray)

cx = lambda a, b: round(np.inner(a, b) / (norm(a) \* norm(b)), 3)

for vector in trainVectorizerArray:

print(vector)

for testV in testVectorizerArray:

print(testV)

cosine = cx(vector, testV)

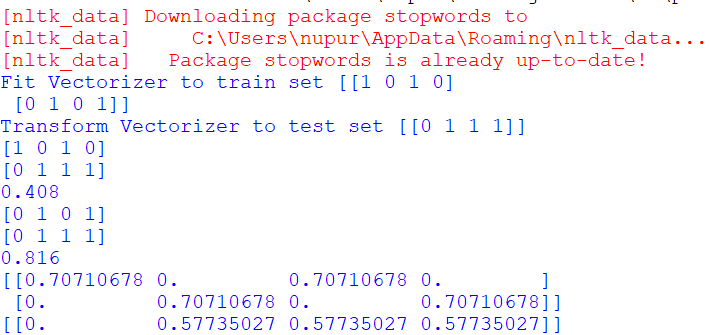
print(cosine)

transformer = TfidfTransformer()

print(transformer.fit\_transform(trainVectorizerArray).toarray())

print(transformer.fit\_transform(testVectorizerArray).todense())

**OUTPUT:**

****