zt1rfbyyf

March 29, 2025

[ ]:
```
#2.Algorithm for Creating representation of document by calculating TFIDF Step :
```

[22]:
```
#Import the necessary libraries :
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

[23]:
```
#Initialize the Documents:
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'
```

[24]:
```
# Create BagofWords (BoW) for Document A and B:
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

[25]:
```
# Create Collection of Unique words from Document A and B:
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

[28]:
```
#Create a dictionary of words and their occurrence for each document in the␣
 ↪corpus :
numOfWordsA = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsA:
 numOfWordsA[word] += 1
 numOfWordsB = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsB:
 numOfWordsB[word] += 1
```

[29]:
```
# Compute the term frequency for each of our documents:
def computeTF(wordDict, bagOfWords):
    tfDict = {}
    bagOfWordsCount = len(bagOfWords)

    for word, count in wordDict.items():
        tfDict[word] = count / float(bagOfWordsCount)

    return
tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)
```

[30]:
```
#Compute the term Inverse Document Frequency:
import math

def computeIDF(documents):
    N = len(documents)
    idfDict = dict.fromkeys(documents[0].keys(), 0)

    for document in documents:
        for word, val in document.items():
            if val > 0:
                idfDict[word] += 1

    for word, val in idfDict.items():
        idfDict[word] = math.log(N / float(val)) if val > 0 else 0  # To avoid␣
 ↪division by zero

    return idfDict

# Assuming numOfWordsA and numOfWordsB are predefined dictionaries
idfs = computeIDF([numOfWordsA, numOfWordsB])
print(idfs)
```

```
{'Jupiter': 0.6931471805599453, 'Planet': 0.6931471805599453, 'fourth':
0.6931471805599453, 'largest': 0.6931471805599453, 'planet': 0.6931471805599453,
'Mars': 0.6931471805599453, 'Sun': 0.6931471805599453, 'the': 0.0, 'is': 0.0,
'from': 0.6931471805599453}
```

[39]:
```
#Compute the term TF/IDF for all words:

import math
import pandas as pd

# Define the documents
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'

# Tokenize the documents and compute term frequencies (TF)
def computeTF(document):
    words = document.lower().split()  # Convert to lowercase and split into␣
 ↪words
    tfDict = {}
    for word in words:
        if word in tfDict:
            tfDict[word] += 1
        else:
            tfDict[word] = 1
    # Normalize the term frequencies by dividing by the total number of words
```

```
    total_words = len(words)
    for word in tfDict:
        tfDict[word] = tfDict[word] / total_words
    return tfDict

# Compute Term Frequencies for both documents
tfA = computeTF(documentA)
tfB = computeTF(documentB)

# Check if TF values are calculated properly
print("TF for Document A:", tfA)
print("TF for Document B:", tfB)

# Compute the IDF for each word
def computeIDF(documents):
    N = len(documents)
    idfDict = {}
    all_words = set()  # To get the unique words across all documents
    for document in documents:
        all_words.update(document.split())

    for word in all_words:
        count = 0
        for document in documents:
            if word in document.split():
                count += 1
        idfDict[word] = math.log(N / (1 + count))  # +1 to avoid division by␣
 ↪zero
    return idfDict

# Compute the IDF values for the words in both documents
idfs = computeIDF([documentA, documentB])

# Check if IDF values are calculated properly
print("IDF values:", idfs)

# Compute the TF-IDF values
def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs.get(word, 0)  # Use idfs.get(word, 0) to avoid␣
 ↪missing key errors
    return tfidf

# Compute TF-IDF for both documents
tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)
```

```
# Check if TF-IDF values are computed properly
print("TF-IDF for Document A:", tfidfA)
print("TF-IDF for Document B:", tfidfB)

# Create a DataFrame from the computed TF-IDF values
df = pd.DataFrame([tfidfA, tfidfB], index=['Document A', 'Document B'])

# Display the DataFrame
print(df)
```

```
TF for Document A: {'jupiter': 0.2, 'is': 0.2, 'the': 0.2, 'largest': 0.2,
'planet': 0.2}
TF for Document B: {'mars': 0.125, 'is': 0.125, 'the': 0.25, 'fourth': 0.125,
'planet': 0.125, 'from': 0.125, 'sun': 0.125}
IDF values: {'Jupiter': 0.0, 'Planet': 0.0, 'fourth': 0.0, 'largest': 0.0,
'planet': 0.0, 'Mars': 0.0, 'Sun': 0.0, 'the': -0.40546510810816444, 'is':
-0.40546510810816444, 'from': 0.0}
TF-IDF for Document A: {'jupiter': 0.0, 'is': -0.0810930216216329, 'the':
-0.0810930216216329, 'largest': 0.0, 'planet': 0.0}
TF-IDF for Document B: {'mars': 0.0, 'is': -0.050683138513520555, 'the':
-0.10136627702704111, 'fourth': 0.0, 'planet': 0.0, 'from': 0.0, 'sun': 0.0}
            jupiter        is       the  largest  planet  mars  fourth  from  \
Document A      0.0 -0.081093 -0.081093      0.0     0.0   NaN     NaN   NaN
Document B      NaN -0.050683 -0.101366      NaN     0.0   0.0     0.0   0.0

            sun
Document A  NaN
Document B  0.0
```

[ ]: