COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

NATURAL LANGUAGE PROCESSING

(UML602)

Project Report On

DOCUMENT RETRIEVEL USING TF-IDF

Submitted to

Dr. Jasmeet Singh

Submitted by

Akshay Rathee 101783045

Vikas Airan 101783067

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Introduction

Document retrieval is defined as the matching of some stated user query against a set of free-text records. These records could be any type of mainly unstructured text, such as newspaper articles, real estate records or paragraphs in a manual. User queries can range from multi-sentence full descriptions of an information need to a few words.

Document retrieval is sometimes referred to as, or as a branch of, text retrieval. Text retrieval is a branch of information retrieval where the information is stored primarily in the form of text. Text retrieval is a critical area of study today, since it is the fundamental basis of all internet search engines.

Main Task of Document Retrieval is to

- Find relevant documents to user queries
- Evaluate the matching results and sort them according to relevance, using algorithms.

Steps/Methodology

- **Tokenization:** This step extracts individual terms from the document, converts them to lower case and removes punctuation marks.
- **Stop-words removal:** Stop words are high frequency words which have little semantic weight and are thus not useful in retrieval. These words play useful grammatical role such as formation of phrases but do not contribute in keyword-based representation. Eliminating stop words considerably reduce the index terms size.
- **Stemming:** After stop word removal, the morphological variant word forms are stemmed to their root word. Stemming helps in solving the problem of vocabulary mismatch and reduction in size of index terms. In some cases, stemming reduces the performance of IR systems by conflating incorrect morphological variant variants. But if the stemming rules are correct then it helps in increasing the performance of the system.
- **Term Weighting:** Each term that is selected as an indexing feature for a document, acts as a discriminator between the document and all other documents in corpus.
    1. Term Frequency (tf): It is based on the fact that more a document contains a given word, the more the document is about a concept represented by that word. It is generally computed as the raw frequency of the term in the document.
    2. Inverse Document Frequency (idf): The inverse document frequency is a measure of how much information the word provides, i.e., if it's common or rare across all documents
- **Similarity Measures:** Different similarity measures such as Pearson correlation, Jaccard similarity and cosine similarity are used to find the similarity scores between the sentence and different documents according to each similarity measure and then a final rank is assigned to each document giving some weight to scores of each similarity measure.

Code

```
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import *
from math import *
import os

path=os.getcwd()
docs=os.listdir(path)

keywords=[]
doc_keywords={}
porter=PorterStemmer()

for doc in docs:
    f=open(path+"\\"+doc,"r")
    tokens=nltk.word_tokenize(f.read())
    raw=[w.lower() for w in tokens if not w in stopwords.words('english')]
    doc_keywords[doc]=[porter.stem(t) for t in raw]
    keywords=keywords+doc_keywords[doc]
keywords=sorted(set(keywords))
tfreq={}

for doc in docs:
    freq={}
    for word in keywords:
        count=0
        for w in doc_keywords[doc]:
            if w==word:
             count+=1
```

```
            if count==0:
                freq[word]=count
            else:
                freq[word]=1+log(count)
        tfreq[doc]=freq
idfreq={}

for word in keywords:
        count=0
        for doc in docs:
            if word in doc_keywords[doc]:
                count+=1
        idfreq[word]=log(len(docs)/count)

tfidf={}

for doc in docs:
        freq={}
        for word in keywords:
            freq[word]=tfreq[doc][word]*idfreq[word]
        tfidf[doc]=freq

def pearson_correlation(x, y):
    mean_x = sum(x)/len(x)
    mean_y = sum(y)/len(y)
    subtracted_mean_x = [i - mean_x for i in x]
    subtracted_mean_y = [i - mean_y for i in y]
    x_times_y = [a * b for a, b in list(zip(subtracted_mean_x, subtracted_mean_y))]
    x_squared = [i * i for i in x]
    y_squared = [i * i for i in y]
    return sum(x_times_y) / sqrt(sum(x_squared) * sum(y_squared))
```

```
def square_rooted(x):
    return round(sqrt(sum([a*a for a in x])),3)

def cosine_similarity(x,y):
   numerator = sum(a*b for a,b in zip(x,y))
  denominator = square_rooted(x)*square_rooted(y)
  return round(numerator/float(denominator),3)

def jaccard_similarity(x,y):
    intersection_cardinality = len(set.intersection(*[set(x), set(y)]))
   union_cardinality = len(set.union(*[set(x), set(y)]))
   return intersection_cardinality/float(union_cardinality)

search_sentence=input("Enter text to Search ")

tokens=nltk.word_tokenize(search_sentence)
raw=[w.lower() for w in tokens if not w in stopwords.words('english')]
search_sentence=[porter.stem(t) for t in raw]

tf_sent={}
for word in keywords:
        count=0
        for w in search_sentence:
                if w==word:
                        count+=1
        if count==0:
                tf_sent[word]=count
        else:
                tf_sent[word]=1+log(count)

tfidf_sent={}
```

```
for word in keywords:
    tfidf_sent[word]=tf_sent[word]*idfreq[word]

cosine_sim={}
pearson_corr={}
jaccard_sim = {}
try:
    for doc in docs:
        cosine_sim[doc]=cosine_similarity(tfidf[doc].values(),tfidf_sent.values())
        jaccard_sim[doc]=jaccard_similarity(doc_keywords[doc],search_sentence)
        pearson_corr[doc]=pearson_correlation(tfidf[doc].values(),tfidf_sent.values())
    points={}
    cosine_sim_points={}
    jaccard_sim_points={}
    pearson_corr_points={}

    count=1
    for key, value in sorted(cosine_sim.items(), key=lambda item: item[1]):
        cosine_sim_points[key]=count
        count+=1

    print("\nCosine Similarity Score:\n")
    print(cosine_sim_points)

    count=1
    for key, value in sorted(pearson_corr.items(), key=lambda item: item[1]):
        pearson_corr_points[key]=count
        count+=1

    print("\nPearson Correlation Score\n")
    print(pearson_corr_points)
```

```
        count=1
        for key, value in sorted(jaccard_sim.items(), key=lambda item: item[1]):
                jaccard_sim_points[key]=count
                count+=1

        print("\nJaccard Similarity Score\n")
        print(jaccard_sim_points)

        for doc in docs:
           points[doc]=0.4*cosine_sim_points[doc]+0.3*pearson_corr_points[doc]+0.3*jaccard_si
m_points[doc]
        print("\nFinal Rankings of Documents\n")
        rank=1
        for key, value in sorted(points.items(), key=lambda item: item[1],reverse=True):
                print("%s: %s" % (key, rank))
                rank+=1
except:
        print("\nNo document related to search query found\n")
```

Result

```
C:\Users\Vikas Airan\Desktop\docs_retrieval>document_retrieval.py
Enter text to Search Iron Man is the godfather of Marvel Cinematic Universe

Cosine Similarity Score:

{'document_retrieval.py': 1, 'iot.txt': 2, 'sport3.txt': 3, 'virtual reality.txt': 4, 'fogcomputing.txt': 5, 'sport 4.txt': 6, 'sport1.txt': 7, 'sport2.txt'
: 8, 'mcu1.txt': 9, 'mcu2.txt': 10}

Pearson Correlation Score

{'iot.txt': 1, 'sport3.txt': 2, 'virtual reality.txt': 3, 'document_retrieval.py': 4, 'fogcomputing.txt': 5, 'sport 4.txt': 6, 'sport1.txt': 7, 'sport2.txt'
: 8, 'mcu1.txt': 9, 'mcu2.txt': 10}

Jaccard Similarity Score

{'document_retrieval.py': 1, 'iot.txt': 2, 'sport3.txt': 3, 'sport2.txt': 4, 'fogcomputing.txt': 5, 'virtual reality.txt': 6, 'sport1.txt': 7, 'mcu1.txt': 8
, 'sport 4.txt': 9, 'mcu2.txt': 10}

Final Rankings of Documents

mcu2.txt: 1
mcu1.txt: 2
sport1.txt: 3
sport 4.txt: 4
sport2.txt: 5
fogcomputing.txt: 6
virtual reality.txt: 7
sport3.txt: 8
document_retrieval.py: 9
iot.txt: 10

C:\Users\Vikas Airan\Desktop\docs_retrieval>
```

```
C:\Users\Vikas Airan\Desktop\docs_retrieval>document_retrieval.py
Enter text to Search Michael Jordan is a professional basketball player

Cosine Similarity Score:

{'document_retrieval.py': 1, 'fogcomputing.txt': 2, 'iot.txt': 3, 'mcu2.txt': 4, 'sport1.txt': 5, 'sport 4.txt': 6, 'mcu1.txt': 7, 'virtual reality.txt': 8,
 'sport2.txt': 9, 'sport3.txt': 10}

Pearson Correlation Score

{'iot.txt': 1, 'mcu2.txt': 2, 'fogcomputing.txt': 3, 'mcu1.txt': 4, 'document_retrieval.py': 5, 'sport1.txt': 6, 'sport 4.txt': 7, 'virtual reality.txt': 8,
 'sport2.txt': 9, 'sport3.txt': 10}

Jaccard Similarity Score

{'document_retrieval.py': 1, 'fogcomputing.txt': 2, 'iot.txt': 3, 'mcu2.txt': 4, 'mcu1.txt': 5, 'sport1.txt': 6, 'virtual reality.txt': 7, 'sport 4.txt': 8,
 'sport2.txt': 9, 'sport3.txt': 10}

Final Rankings of Documents

sport3.txt: 1
sport2.txt: 2
virtual reality.txt: 3
sport 4.txt: 4
sport1.txt: 5
mcu1.txt: 6
mcu2.txt: 7
iot.txt: 8
fogcomputing.txt: 9
document_retrieval.py: 10

C:\Users\Vikas Airan\Desktop\docs_retrieval>
```

```
C:\Users\Vikas Airan\Desktop\docs_retrieval>document_retrieval.py
Enter text to Search IOT is a hot topic to research

Cosine Similarity Score:

{'document_retrieval.py': 1, 'mcu1.txt': 2, 'sport 4.txt': 3, 'sport2.txt': 4, 'sport3.txt': 5, 'mcu2.txt': 6, 'virtual reality.txt': 7, 'sport1.txt': 8, 'f
ogcomputing.txt': 9, 'iot.txt': 10}

Pearson Correlation Score

{'mcu1.txt': 1, 'sport2.txt': 2, 'sport3.txt': 3, 'sport 4.txt': 4, 'document_retrieval.py': 5, 'mcu2.txt': 6, 'virtual reality.txt': 7, 'iot.txt': 8, 'spor
t1.txt': 9, 'fogcomputing.txt': 10}

Jaccard Similarity Score

{'document_retrieval.py': 1, 'mcu1.txt': 2, 'sport 4.txt': 3, 'sport2.txt': 4, 'sport3.txt': 5, 'mcu2.txt': 6, 'virtual reality.txt': 7, 'iot.txt': 8, 'spor
t1.txt': 9, 'fogcomputing.txt': 10}

Final Rankings of Documents

fogcomputing.txt: 1
iot.txt: 2
sport1.txt: 3
virtual reality.txt: 4
mcu2.txt: 5
sport3.txt: 6
sport2.txt: 7
sport 4.txt: 8
document_retrieval.py: 9
mcu1.txt: 10
```

```
C:\Users\Vikas Airan\Desktop\docs_retrieval>document_retrieval.py
Enter text to Search PorterStemmer

No document related to search query found
```

Applications

The main application of document retrieval is its usage in search engines. As we know the data present on the internet is huge and whenever we search a query in any search engine it instantly returns with related documents/links ranked in order, this is done using document retrieval.

Future Works

The work presented is an implementation of tf-idf algorithm extended to using various similarity measures to search for a query in given set of documents and rank them based on their similarity to the search text. Its implementation can be extended to construct a search engine based collection of documents like in library monitoring system where various documents can be classified based on the used algorithm and user can easily search for a document based on certain set of keywords. It can further be extended to be used with audio recognition systems to create a personal assistant that can recognize our commands and based on above algorithm and pre set of defined instructions, it can be used to generate results as per users query.