Vector Space Model for Ranking Documents

Nishant Sahoo(150953244)
Samarth Agarwal(150953200)
Vaibhav Mehrotra()
Srishti Goyal()

Problem Statement:

Develop a simple Information Retrieval System using vector space model for ranking the document. Following are the functionalities to be implemented.

- Read the content of the document (minimum 5 documents, 20 terms)
- Compute tf, idf, tf-idf for document as well as query. Represent document and query as a vector where tf-idf is the component of the vector.
- Rank the document using vector space model.
- Create appropriate interfaces for inputting the query, displaying the query results according to the rank along with the score.

Explanation:

Vector Space model for ranking documents follows the following steps:

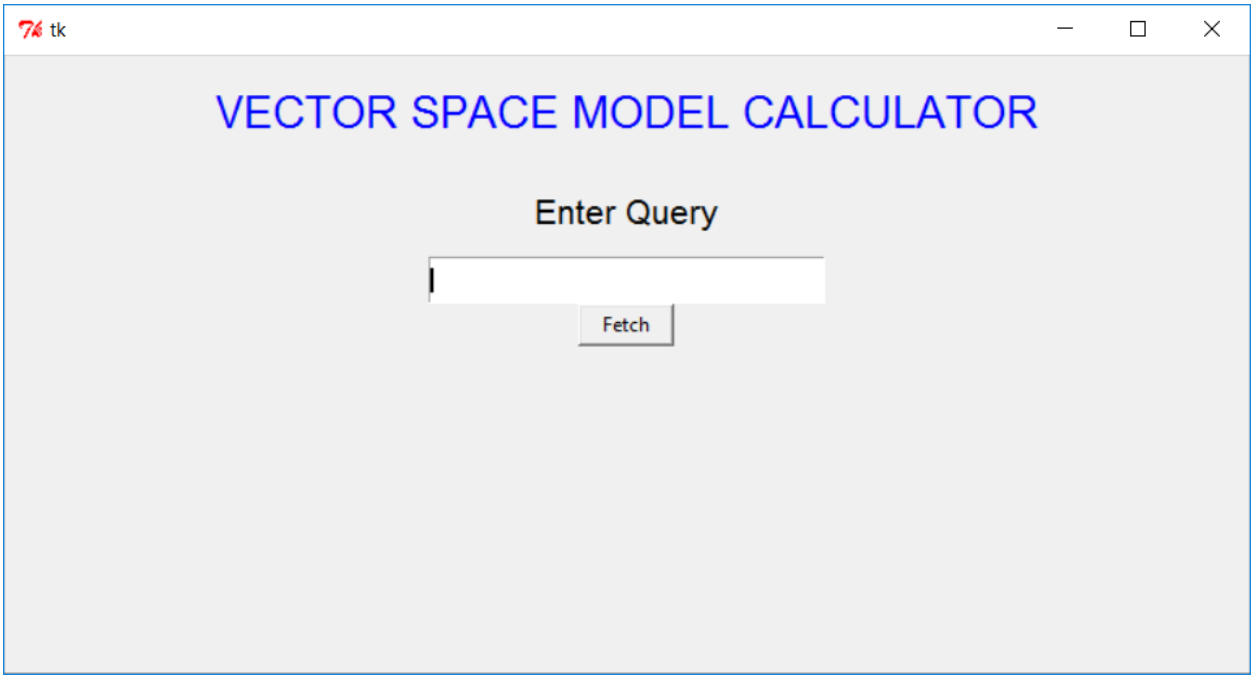
1. The documents and query are made into vectors.
2. Term frequency which is the number of occurrences of a word in a document is calculated for each term in the query.
3. Document frequency which is the number of documents in the collection in which the term t occurs is calculated for each term in the query.
4. Inverse Document Frequency is calculated for each term in the query by using the formula log(N/df).
5. Tf-idf value is calculated for each term in the query.
6. Cosine Score of each document is calculated by multiplying the tf-idf of various documents with query tf-idf dividing it by the euclidean lengths.
7. The ranking of the documents is done by sorting these scores.

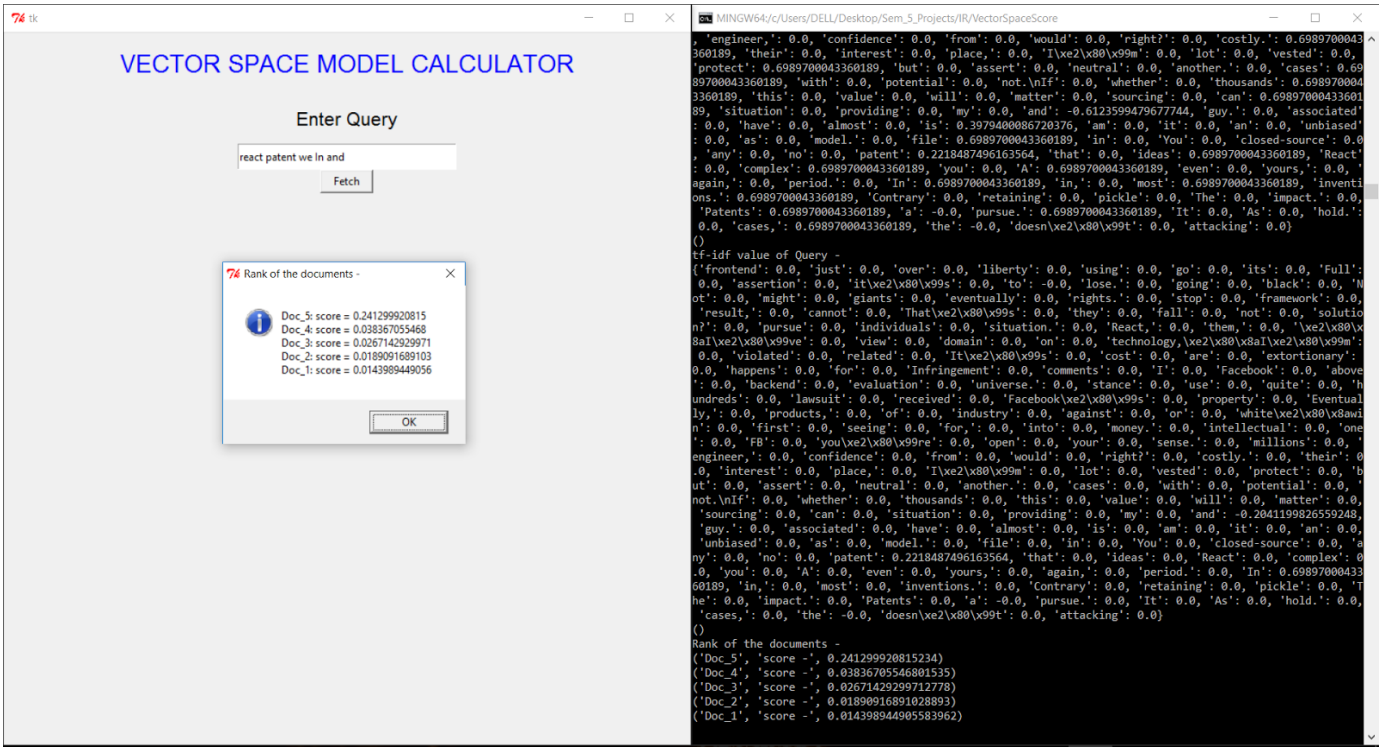
Implementation:

The code was implemented using Python 3.0 and the interface of the system was implemented using PythonGUI Tkinter library.

1. The interface is seen at first where the query can be entered.
2. On pressing fetch the scoring and ranking is displayed in a message box.
3. First the documents are read from a directory. The sentences are taken from the documents and split into words and stored into a list thus making a vector of each document.
4. The query is also split and made into a vector.
5. Term frequency of each query term is calculated and stored into doc_tf.
6. Document frequency of each term is calculated and stored into doc_all_df.
7. Then inverse document frequency is calculated for each term and fetched into idf[].
8. Tf-idf value is calculated and fetched into doc_no._tf_idf.
9. In the next step the partial scores of each document is calculated by multiplying the tf_idf value of the query with the tf_idf value of the document.
10. Euclidean lengths of both the query and document is calculated and multiplied and stored into score_doc_no.
11. A sorting function is run on these scores and the documents are ranked in the ascending order.

Snapshots of Output:





Code:

```
#!python3

from __future__ import division # to obtain force division to be floating point
import math
import platform
from tkinter import *
from tkinter import messagebox
import math
import os

print('Python version ', platform.python_version())


# GUI Window
root = Tk()
root.geometry("800x400") # Size of the window

# Heading
w = Label(root, text="VECTOR SPACE MODEL CALCULATOR",
              fg="blue",
              font=("Franklic Gothic Book", 20),
              justify= CENTER, height=2)
w.pack()

# Taking Input
Label(root, text="Enter Query",
          font=("Franklic Gothic Book", 16),
          justify= CENTER, height=2).pack()


# Storing into a variable s on click of a button fetch
def fetch():
   print ('Input =>',e.get())
   s=''
   s=e.get()
   print(s)

   ####################################
   # Critical section #

   print('Vector Space Scoring')

   # Initializing all variables ----------------------------------------------------

   documents = []
   for cwd, sub_folder, files in os.walk("documents"):
           for file in files:
                   print("File name:", file)
```

```
            f = open(str(cwd + "/" + file), "r")
            doc_text = f.read()
            print(doc_text)
            documents += [doc_text]


print(documents)
doc_1 = documents[0]
doc_2 = documents[1]
doc_3 = documents[2]
doc_4 = documents[3]
doc_5 = documents[4]


query = s

doc_1_list = doc_1.split(' ')
doc_2_list = doc_2.split(' ')
doc_3_list = doc_3.split(' ')
doc_4_list = doc_4.split(' ')
doc_5_list = doc_5.split(' ')
query_list = query.split(' ')

all_words  = list(set(doc_1_list + doc_2_list + doc_3_list + doc_4_list + doc_5_list))
number_of_documents = 5

doc_1_tf   = {} # term frequency of doc_1
doc_2_tf   = {} # term frequency of doc_1
doc_3_tf   = {} # term frequency of doc_1
doc_4_tf   = {}
doc_5_tf   = {}
query_tf   = {} # term frequency of query
doc_all_df = {} # document frequency

idf = {} # Inverse Document Frequency

doc_1_tf_idf = {} # tf-df of doc_1
doc_2_tf_idf = {} # tf-df of doc_2
doc_3_tf_idf = {} # tf-df of doc_3
doc_4_tf_idf = {}
doc_5_tf_idf = {}
query_tf_idf = {} # tf-df of query

# Initializing scores as 0
score_doc_1  = 0
score_doc_2  = 0
score_doc_3  = 0
score_doc_4  = 0
score_doc_5  = 0
```

```
    # Initializing all tf lists to be empty
    for each in all_words:
        doc_1_tf[each] = 0
        doc_2_tf[each] = 0
        doc_3_tf[each] = 0
        doc_4_tf[each] = 0
        doc_5_tf[each] = 0

    # Variable initialization complete ------------------------------------------

    # Display data collection
    print('List of all words -')
    print(all_words)
    print()
    print('Query -', query)
    print()

    # Term Frequency calculation for document 1
    for each in all_words:
        doc_1_tf.update({each: doc_1_list.count(each)})

    # Term Frequency calculation for document 2
    for each in all_words:
        doc_2_tf.update({each: doc_2_list.count(each)})

    # Term Frequency calculation for document 3
    for each in all_words:
        doc_3_tf.update({each: doc_3_list.count(each)})

    # Term Frequency calculation for document 4
    for each in all_words:
        doc_4_tf.update({each: doc_4_list.count(each)})

    # Term Frequency calculation for document 5
    for each in all_words:
        doc_5_tf.update({each: doc_5_list.count(each)})

    # Term Frequency calculation for query
    for each in all_words:
        query_tf.update({each: query_list.count(each)})

    # Document Frequency Calculation
    for each in all_words:
        doc_all_df[each] = (doc_1_tf[each] + doc_2_tf[each] + doc_3_tf[each] + doc_4_tf[each] +
doc_5_tf[each])

    print('Term frequency of Document 1 -')
    print(doc_1_tf)
    print()
    print('Term frequency of Document 2 -')
```

```
print(doc_2_tf)
print()
print('Term frequency of Document 3 -')
print(doc_3_tf)
print()
print('Term frequency of Document 3 -')
print(doc_4_tf)
print()
print('Term frequency of Document 3 -')
print(doc_5_tf)
print()
print('Term frequency of Query -')
print(query_tf)
print()
print('Document Frequency -')
print(doc_all_df)

sum = 0
for each in all_words:
        sum += query_tf[each]

if sum == 0:
        messagebox.showinfo("ERROR!", "No document found for the given query")
        e.delete(0, END)

# Calculation of Inverse Document Frequency (idf)
for each in all_words:
    idf[each] = math.log10((number_of_documents/doc_all_df[each]))

print('Inverted Document Frequency -')
print(idf)

# tf-idf calculation

for each in all_words:
    doc_1_tf_idf[each] = idf[each]*doc_1_tf[each]
    doc_2_tf_idf[each] = idf[each]*doc_2_tf[each]
    doc_3_tf_idf[each] = idf[each]*doc_3_tf[each]
    doc_4_tf_idf[each] = idf[each]*doc_4_tf[each]
    doc_5_tf_idf[each] = idf[each]*doc_5_tf[each]
    query_tf_idf[each] = idf[each]*query_tf[each]

print()
print('tf-idf value of Document 1 -')
print(doc_1_tf_idf)
print()
print('tf-idf value of Document 2 -')
print(doc_2_tf_idf)
print()
print('tf-idf value of Document 3 -')
```

```
print(doc_3_tf_idf)
print()
print('tf-idf value of Document 3 -')
print(doc_4_tf_idf)
print()
print('tf-idf value of Document 3 -')
print(doc_5_tf_idf)
print()
print('tf-idf value of Query -')
print(query_tf_idf)
print()

# Calculation of partial scores of documents for the given query
for each in doc_1_tf_idf.keys():
    score_doc_1 += doc_1_tf_idf[each]*query_tf_idf[each]

for each in doc_2_tf_idf.keys():
    score_doc_2 += doc_2_tf_idf[each]*query_tf_idf[each]

for each in doc_3_tf_idf.keys():
    score_doc_3 += doc_3_tf_idf[each]*query_tf_idf[each]

for each in doc_4_tf_idf.keys():
    score_doc_4 += doc_4_tf_idf[each]*query_tf_idf[each]

for each in doc_5_tf_idf.keys():
    score_doc_5 += doc_5_tf_idf[each]*query_tf_idf[each]

# Calculation of euclidean_length
def euclidean_length(vector):
    t_sum = length = 0
    for each in vector.values():
        t_sum += each**2

    length = math.sqrt(t_sum)
    return length

# Calculation of scores of documents for the given query
score_doc_1 /= (euclidean_length(doc_1_tf_idf)*euclidean_length(query_tf_idf))
score_doc_2 /= (euclidean_length(doc_2_tf_idf)*euclidean_length(query_tf_idf))
score_doc_3 /= (euclidean_length(doc_3_tf_idf)*euclidean_length(query_tf_idf))
score_doc_4 /= (euclidean_length(doc_4_tf_idf)*euclidean_length(query_tf_idf))
score_doc_5 /= (euclidean_length(doc_5_tf_idf)*euclidean_length(query_tf_idf))

sorted_list_score = [("Doc_1", score_doc_1), ("Doc_2", score_doc_2), ("Doc_3",
score_doc_3), ("Doc_4", score_doc_4), ("Doc_5", score_doc_5)]

# Custom function for sorting (Sorts according to the score)
def score_cmp(list_item):
    return list_item[1]
```

```
    # Sorting the document scores in descending order
    sorted_list_score.sort(key=score_cmp, reverse=True)

    print('Rank of the documents -')
    for each in sorted_list_score:
        print(each[0], 'score -' , each[1])

    output_string = ""
    for each in sorted_list_score:
        output_string += str(each[0]) + ': score = ' + str(each[1]) + '\n'

    messagebox.showinfo("Rank of the documents -", output_string)
    e.delete(0, END)
    # Critical section ends here #
    ####################################

# Data is fetched into s

e=Entry(root, width=40)
e.pack(ipady=5)
e.focus()
e.bind('<Return>', (lambda event: fetch()))
btn = Button(root, text='Fetch', command=fetch, height=1, width=7)
btn.pack(side=TOP)

root.mainloop()
```