Document Retrieval using TF-IDF & Ranking using Matching Score and Cosine Similarity

...

Submitted by: Group 1

Khushwant Kaswan - 195030
Monu Kumar - 195083
Mohd Salmaan - 195088

Submitted to:

Dr. Arun Kumar Yadav

Content

TF - IDF

- "Term Frequency - Inverse Document Frequency"
- **Term Frequency**: frequency of a word in a document.
- **Document Frequency**: frequency of documents in which the word is present.
- TF-IDF = Term Frequency (TF) * Inverse Document Frequency (IDF)
- tf-idf(t, d) = tf(t, d) * log(N/(df))
- The goal of tf-idf is to quantify how uniquely important a word of interest is to a given document within a collection of documents

Workflow

- Data scraping from local fs
- Collecting the file paths, titles and creating dataset
- Data Preprocessing
- Calculating DF for all word
- Calculating TF-IDF for body text
- Calculating TF-IDF for Title text
- Merging the TF-IDF of title and body
- Ranking
    - Matching Score
    - Cosine Similarity

Dataset Used

The dataset we have used are collection of files stored in different directories.

This means we have documents in different formats i.e. html, txt, hac, hum, key....

Dataset : http://archives.textfiles.com/

Files are present in different folders. There's an index.html in each folder (including the root), which contains all the document names and their titles.

Total files in dataset : 467

After Preprocessing, Total Vocab Size : 32350

Dataset at a glance

Filename	Size	Description of the Textfile
FARNON		The Stories of Tristan Farnon
SRE		The Solar Realms Elite, by Josh Renaud
100west.txt	20839	Going 100 West by 53 North by Jim Prentice (1990)
13chil.txt	8457	The Story of the Sly Fox
14.lws	5261	A Smart Bomb with a Language Parser
16.lws	15294	Two Guys in a Garage, by M. Pshota
17.lws	10853	The Early Days of a High-Tech Start-up are Magic (November 18, 1991) by M. Peshota
18.lws	26624	The Couch, the File Cabinet, and the Calendar, by M. Peshota (December 9, 1991)
19.lws	17902	Engineering the Future of American Technology by M. Peshota (January 5, 1992)
20.lws	13588	What Research and Development Was Always Meant to Be, by M. Peshota
3gables.txt	33985	The Adventure of the Three Gables
3lpigs.txt	5403	The Story of the 3 Little Pigs
3sonnets.vrs	2433	A Collection of Sonnets by Staeorra Rokraven (March 9, 1989)

Filename and Description are in the index.html, we need to extract those names and titles.

Preprocessing

- convert_lower_case(data) : The One With Unagi => the one with unagi
- remove_punctuation(data) : !\"#$%&()*+-./:;<=>?@[\]^_`{|}~\n => <space>
- remove_apostrophe(data) : ‘ => <empty>
- remove_stop_words(data) : the one with unagi => one unagi
- convert_numbers(data) : 100 => hundred
- stemming(data)
    - Porter Stemmer

Calculating DF for all words

Measures the frequency of documents in which the word is present.

We will use a dictionary and we can use the word as the key and a set of documents as the value.

But for DF we don’t actually need the list of docs, we just need the count. so we are going to replace the list with its count.

```
{'sharewar': 1,
 'trial': 1,
 'project': 4,
 'freewar': 1,
 'need': 6,
 'support': 2,
 'continu': 4,
 'one': 10,
 'hundr': 8,
```

```python
DF = {}

for i in range(N):
    tokens = processed_text[i]
    for w in tokens:
        try:
            DF[w].add(i)
        except:
            DF[w] = {i}

    tokens = processed_title[i]
    for w in tokens:
        try:
            DF[w].add(i)
        except:
            DF[w] = {i}
for i in DF:
    DF[i] = len(DF[i])
```

TF - IDF for body and title

We need to iterate over all the documents, we can calculate the frequency of the tokens, tf-idf and finally store as a (doc, token) pair in tf_idf.

```python
tf_idf = {}

for i in range(N):
    tokens = processed_text[i]
    counter = Counter(tokens + processed_title[i])
    words_count = len(tokens + processed_title[i])

    for token in np.unique(tokens):
        tf = counter[token]/words_count
        df = doc_freq(token)
        idf = np.log((N+1)/(df+1))
        tf_idf[doc, token] = tf*idf
    doc += 1
```

```
{(0, 'fifti'): 0.005434960598980563,
 (0, 'go'): 0.0002906893990853149,
 (0, 'hundr'): 0.002570392381970895,
 (0, 'jim'): 0.005269857144642146,
 (0, 'nine'): 0.0008420698058556812,
 (0, 'nineti'): 0.0013127162788344334,
 (0, 'north'): 0.021919902239379185,
 (0, 'one'): 0.0003992734536048051,
 (0, 'prentic'): 0.008085184948722846,
 (0, 'thousand'): 0.0008961984314476824,
 (0, 'three'): 0.0015785688576535318,
 (0, 'west'): 0.0033256596840258424,
 (1, 'fox'): 0.11198195635330804,
 (1, 'sli'): 0.11239056533822733,
 (1, 'stori'): 0.0007682063585522353,
```

Merging the TF-IDF of title and body

document = body + title

TF-IDF(document) = TF-IDF(body) * (alpha) + TF-IDF(title) * (1-alpha)

alpha = 0.3

Flow :

- Calculate DF
- Calculate TF-IDF for Body for all docs
- Calculate TF-IDF for title for all docs
- Merging the TF-IDF of title and body

So, finally, we have a dictionary tf_idf which has the values as a (doc, token) pair

Ranking: Matching Score

We need to check in every document if these query_tokens exist and if the query_tokens exists, then the tf_idf value is added to the matching score of that particular doc_id. Format of tf_idf is (doc, token).

```
query_weights = {}
for key in tf_idf:
    if key[1] in query_tokens:
        try:
            query_weights[key[0]] += tf_idf[key]
        except:
            query_weights[key[0]] = tf_idf[key]
query_weights = sorted(query_weights.items(), key=lambda x: x[1], reverse=True)
```

```
Query: One day, he noticed that one of the peacocks had dropped a feather. When the

Tokens: 36

Matching Score


429 c:\Users\kk910\OneDrive\Desktop\IR/data/vaincrow.txt | Matching Score  0.4
186 c:\Users\kk910\OneDrive\Desktop\IR/data/foxncrow.txt | Matching Score  0.21
405 c:\Users\kk910\OneDrive\Desktop\IR/data/tailbear.txt | Matching Score  0.1
339 c:\Users\kk910\OneDrive\Desktop\IR/data/quarter.c11 | Matching Score  0.08
351 c:\Users\kk910\OneDrive\Desktop\IR/data/quarter.c5 | Matching Score  0.05
416 c:\Users\kk910\OneDrive\Desktop\IR/data/the-tree.txt | Matching Score  0.03
291 c:\Users\kk910\OneDrive\Desktop\IR/data/musibrem.txt | Matching Score  0.02
176 c:\Users\kk910\OneDrive\Desktop\IR/data/fish.txt | Matching Score  0.02
294 c:\Users\kk910\OneDrive\Desktop\IR/data/narciss.txt | Matching Score  0.02
264 c:\Users\kk910\OneDrive\Desktop\IR/data/lionbird | Matching Score  0.01
```

Vectorising Documents

Used total_vocab (list of unique tokens) to generate an index for each token, and we will use numpy array of size (docs, total_vocab) to store the document vectors.

```python
D = np.zeros((N, len(total_vocab)))
for i in tf_idf:
    try:
        ind = total_vocab.index(i[1])
        D[i[0]][ind] = tf_idf[i]
    except:
        pass

row, col = D.shape
print(row, col)
```

Ranking: Cosine Similarity

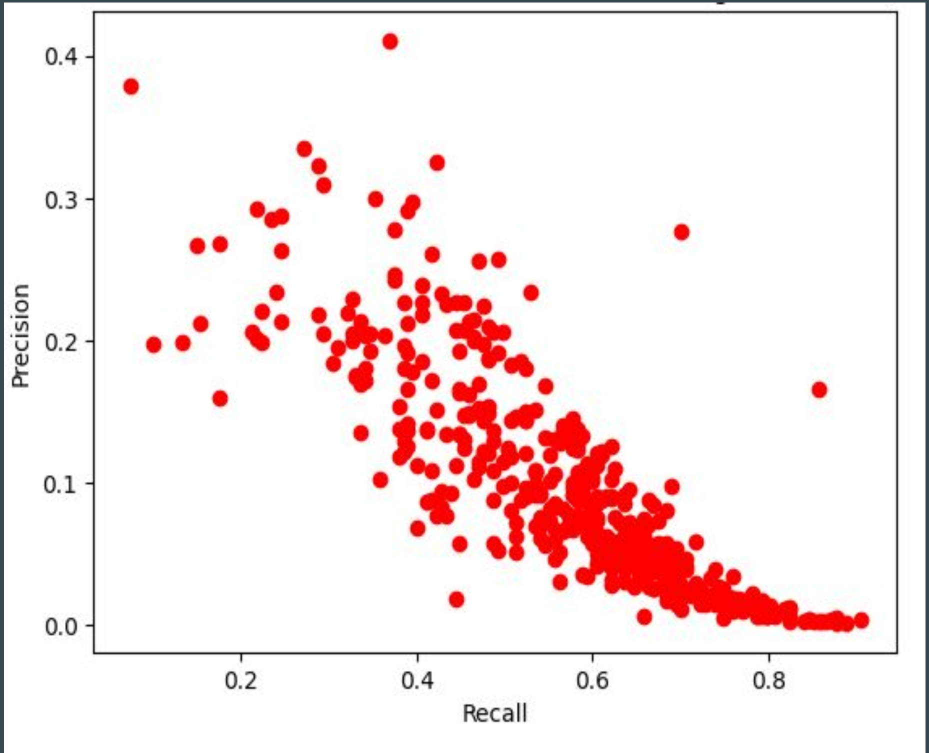
Measure of similarity between two non-zero vectors in a multi-dimensional space. It measures the cosine of the angle between the two vectors, which ranges from -1 to 1.

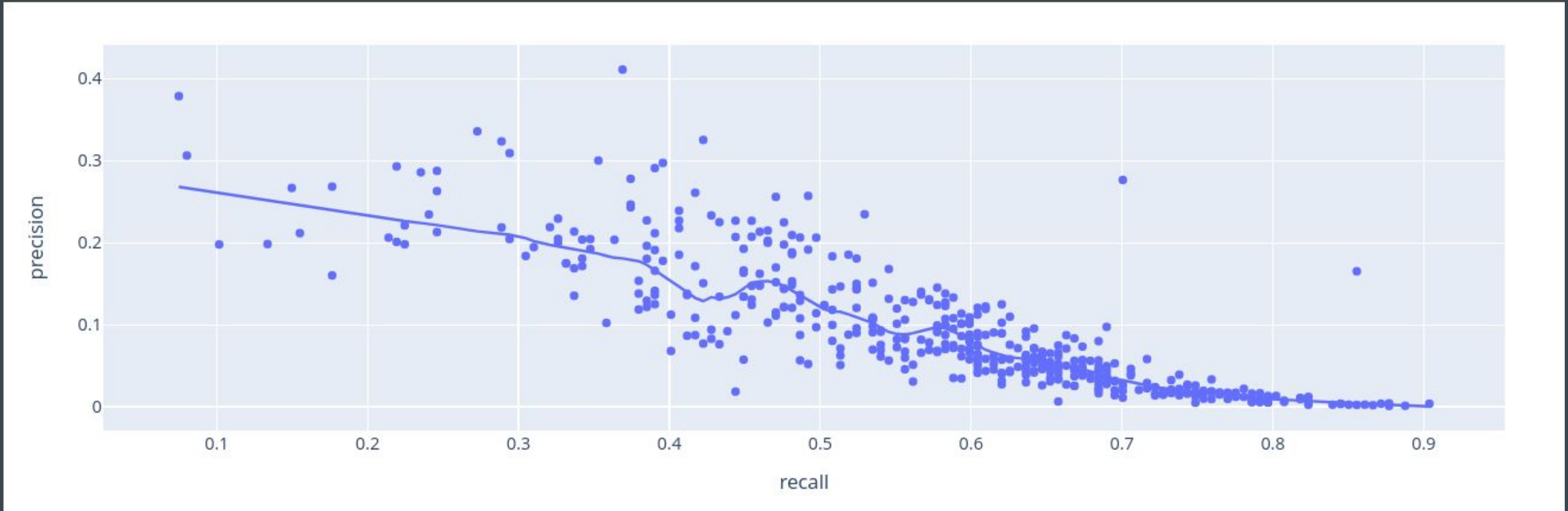
```
d_cosines = []

query_vector = gen_vector(query_tokens)

for d in D:
    d_cosines.append(cosine_sim(query_vector, d))

d_cosines_sorted=sorted(d_cosines)[-k:][::-1]
```

```
def gen_vector(tokens):

    Q = np.zeros((len(total_vocab)))

    counter = Counter(tokens)
    words_count = len(tokens)

    query_weights = {}

    for token in np.unique(tokens):

        tf = counter[token]/words_count
        df = doc_freq(token)
        idf = math.log((N+1)/(df+1))

        try:
            ind = total_vocab.index(token)
            Q[ind] = tf*idf
        except:
            pass
    return Q
```

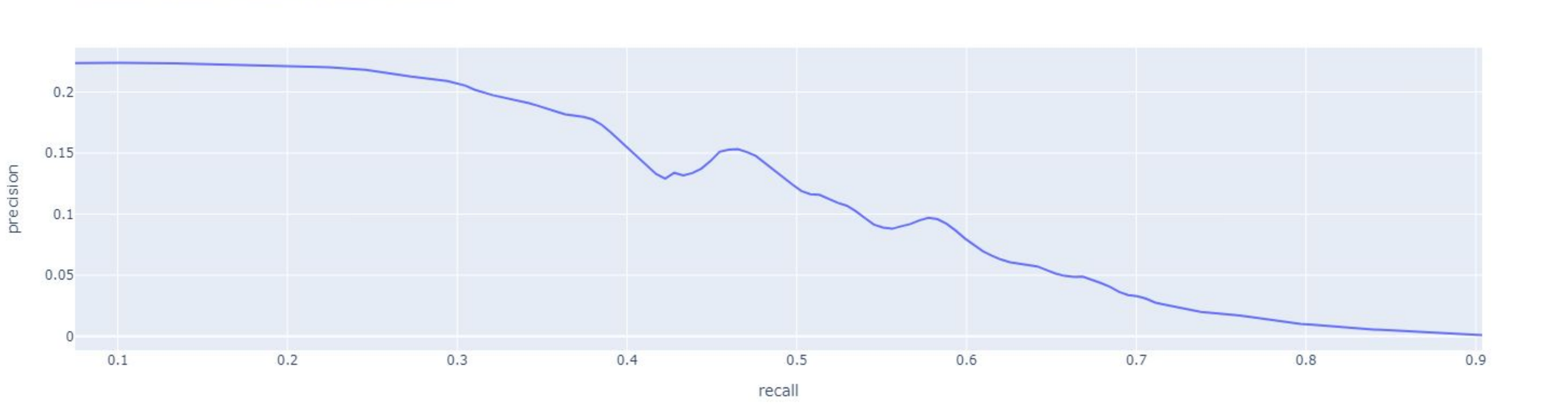
```
Cosine Similarity

Query: One day, he noticed that one of the peacocks had dropped a feather. When the sun

Tokens: 44


429 c:\Users\kk910\OneDrive\Desktop\IR/data/vaincrow.txt | Cosine Similarity 0.49
186 c:\Users\kk910\OneDrive\Desktop\IR/data/foxncrow.txt | Cosine Similarity 0.13
27 c:\Users\kk910\OneDrive\Desktop\IR/data/aesopa10.txt | Cosine Similarity 0.1
229 c:\Users\kk910\OneDrive\Desktop\IR/data/home.fil | Cosine Similarity 0.07
405 c:\Users\kk910\OneDrive\Desktop\IR/data/tailbear.txt | Cosine Similarity 0.06
416 c:\Users\kk910\OneDrive\Desktop\IR/data/the-tree.txt | Cosine Similarity 0.06
339 c:\Users\kk910\OneDrive\Desktop\IR/data/quarter.c11 | Cosine Similarity 0.04
194 c:\Users\kk910\OneDrive\Desktop\IR/data/frogp.txt | Cosine Similarity 0.04
26 c:\Users\kk910\OneDrive\Desktop\IR/data/aesop11.txt | Cosine Similarity 0.04
450 c:\Users\kk910\OneDrive\Desktop\IR/data/yukon.txt | Cosine Similarity 0.03
```

Comparison: Mixed Query from doc(429) + doc(439)





precision
0.4
0.3
0.2
0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
recall



precision
0.2
0.15
0.1
0.05
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
recall

Matching Score

```
Query: One day, he noticed that one of the peacocks had dropped a feather. When the sun went down,
Tokens: 98
429 c:\Users\kk910\OneDrive\Desktop\IR/data/vaincrow.txt | Matching Score  0.44
186 c:\Users\kk910\OneDrive\Desktop\IR/data/foxncrow.txt | Matching Score  0.22
439 c:\Users\kk910\OneDrive\Desktop\IR/data/weeprncs.txt | Matching Score  0.18
400 c:\Users\kk910\OneDrive\Desktop\IR/data/sucker.txt | Matching Score  0.11
405 c:\Users\kk910\OneDrive\Desktop\IR/data/tailbear.txt | Matching Score  0.11
309 c:\Users\kk910\OneDrive\Desktop\IR/data/omarsheh.txt | Matching Score  0.1
161 c:\Users\kk910\OneDrive\Desktop\IR/data/fantas.hum | Matching Score  0.08
339 c:\Users\kk910\OneDrive\Desktop\IR/data/quarter.c11 | Matching Score  0.08
382 c:\Users\kk910\OneDrive\Desktop\IR/data/sleprncs.txt | Matching Score  0.07
293 c:\Users\kk910\OneDrive\Desktop\IR/data/myeyes | Matching Score  0.05
```

Cosine Similarity

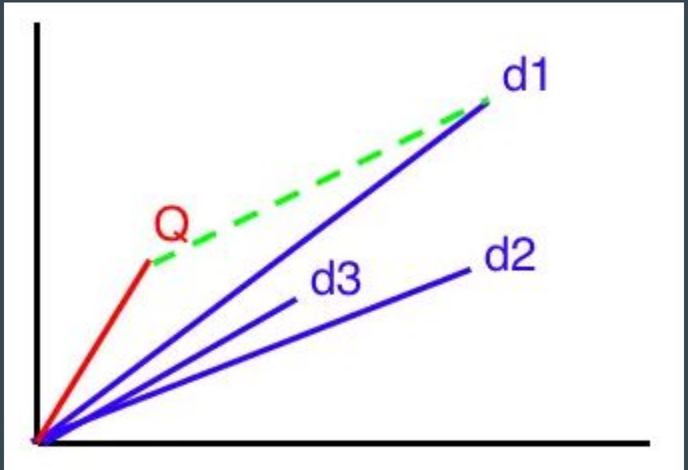
```
Query: One day, he noticed that one of the peacocks had dropped a feather. When the sun w
Tokens: 98
429 c:\Users\kk910\OneDrive\Desktop\IR/data/vaincrow.txt | Cosine Similarity 0.37
439 c:\Users\kk910\OneDrive\Desktop\IR/data/weeprncs.txt | Cosine Similarity 0.24
382 c:\Users\kk910\OneDrive\Desktop\IR/data/sleprncs.txt | Cosine Similarity 0.1
186 c:\Users\kk910\OneDrive\Desktop\IR/data/foxncrow.txt | Cosine Similarity 0.1
27 c:\Users\kk910\OneDrive\Desktop\IR/data/aesopa10.txt | Cosine Similarity 0.08
161 c:\Users\kk910\OneDrive\Desktop\IR/data/fantas.hum | Cosine Similarity 0.06
229 c:\Users\kk910\OneDrive\Desktop\IR/data/home.fil | Cosine Similarity 0.06
405 c:\Users\kk910\OneDrive\Desktop\IR/data/tailbear.txt | Cosine Similarity 0.05
450 c:\Users\kk910\OneDrive\Desktop\IR/data/yukon.txt | Cosine Similarity 0.05
416 c:\Users\kk910\OneDrive\Desktop\IR/data/the-tree.txt | Cosine Similarity 0.04
```

Conclusion

- Mixed Query : doc(429) + doc(439)
  - Matching Score shows 429 , 186 , 439 , 400 ,..
  - Cosine Similarity shows 429, 439, 382, 186 ,..
- Matching Score gives relevant documents but it quite fails when we give long queries, it will not be able to rank them properly.
- Vector POV : Matching Score computes Manhattan distance (straight line from tips) while Cosine score considers the angle of the vectors.

Conclusion (contd..)

- Matching Score will return document d3 but that is not very closely related.
- Cosine Similarity will return document d1



Cosine similarity learns the context more. In every type of query (small , medium , large) cosine similarity ranking is better than matching score ranking.

Thank you