

COL 764 - Info Retrieval and Web Search
Assignment 1 - Vector-space Retrieval

Submitted By : Om Prakash (2019MCS2567)

1 Special Terms

1.1 Vector Space Model

1. Term-weighting improves quality of the answer set
2. Partial Matching
3. Rank based results

1.2 Inverted Index

1. **Vocabulary** - All indexable terms in the collection
2. **Posting List** - List of document IDs with term frequencies. A term is counted only once in a document.

2 Construction on Inverted Index

We have been given hundreds of files, where each file contains multiple documents and further each document may have multiple texts. Text can be either **simple text** or **tagged text**. I have used **beautifulsoup** (a python library) to parse the text.

2.1 Implementation

I used a dictionary which has the following format:

```
IndexFile =
{
  term1
:[document frequency, [(DocId1, term frequency),(DocId2, term frequency),
(DocId3, term frequency)]]
[term2:[document frequency, [(DocId1, term frequency),(DocId2, term frequency)]]
} and so on.
```

2.2 Writing Dictionary to Disk

As we have limited primary memory and can't keep everything the RAM. So, it's better to keep the vocabulary in the memory and posting lists can be kept in the secondary storage.

When I stored the offset for each posting list in the main dictionary where all the terms are kept. Because next time we need the posting list of a particular term then we can pass a term and the offset and fetch the specific list instead of bringing everything in the memory.

Following are two index files written on the disk:

1. **indexfile.dict** it contains vocabulary and size and offset of a posting list.
2. **indexfile.idx** it contains the posting lists which can be fetched with an offset. There is no need to bring the whole posting lists in the memory.

2.3 Programs

- **invidx_cons.py** It constructs the index files and stores to the disk. Before construction of the index and posting lists,
 - stop-words are removed from the document
 - simple text tokens are converted to lower letters
 - Tagged entities have not been converted to lower letters, they have remained the same.
- **printdict.py** This small python program works as a helper tool and prints the dictionary. Following is the format in which the dictionary will be printed:

< term > : < df > : < offset to the posting list >

We can print either all the terms or we can pass an **optional parameter n** then it will print only first n terms only.

3 Processing of Queries

3.1 Implementation

3.1.1 Program

- **vecsearch.py** There are mainly three parts of this program.
 - Loads the vocabulary in the memory
 - Parses the query file and fetches required information
 - Process the query and fetches top k-ranked documents

3.1.2 Approach

There are mainly three types of queries:

- **Simple Queries** These are simple text queries.
- **Tagged Queries** Queries which can have tagged entities like Person, Organization and Location.
 - **P:Person** will look into those documents only where this tagged person is found.
 - **O:Organization** will look into those documents only where this tagged organization is found.
 - **L:Location** will look into those documents only where this tagged location is found.
 - **N:Noun** will look into all the documents where this noun is tagged as a person or location or organization
- **Prefix Queries** It will find all the possible combinations available in the vocabulary and check for all the combinations. Following is an example:

```

query : sou*
all_prefix_terms : ['soul', 'southern', 'sources', 'southeast', 'sound', 'southwestern', 'sought', 's',
'southampton', 'southernmost', 'souk', 'southeast', 'southwestward', 'sounding', 'soulful', 'southw',
ouces', 'southhead', 'southbound', 'southend', 'soundman', 'soundly', 'sounders', 'southland', 'south',
'soucy', 'southwesterly', 'southside', 'sourani', 'southen', 'souza', 'southdown', 'southeastern', 's',
dings', 'soukous', 'sours', 'souda', 'souless', 'southington', 'soundtracks', 'southmark', 'southle',
rton', 'sourrouille', 'soutendijk', 'soumayya', 'soupers', 'soupsthat', 'southold', 'southtown', 'sou',
'southworth', 'sousuki', 'soundproofing', 'soundest', 'souks', 'souffle', 'southpark', 'sous', 'souer',
'southgate', 'sourced', 'souping', 'southlake', 'southbridge', 'soursock', 'southwark', 'southerland',
'soule', 'soun', 'soulshakers', 'soumerai', 'sourmash', 'soundgarden', 'sough', 'southboro', 'souss',
'oofed', 'southwick', 'souding', 'southwell', 'sourl', 'souad', 'soulmate', 'southhaven', 'sounces', 's',
'soures', 'southwest', 'southeastern', 'sourdough', 'southerns', 'soulmates', 'southeby', 'soupless',
'southan', 'soundscape', 'southpaw', 'soudiere', 'soucre', 'soubti', 'soundgress', 'soumnar', 'soutwe',
pmaker', 'soubanh', 'sousaphone', 'souless', 'sou', 'soukup', 'soubeyran', 'soundabout', 'soutine',

```

Figure 1: Example of A Prefix Query

3.2 Algorithmic Details

3.2.1 Deriving term wights

$$tf_{ij} = f_{ij}$$

Where, tf_{ij} = term frequency of ith term in the jth document.

log is better than just the simple raw count, therefore we log normalized as follows,

$$tf_{ij} = 1 + \log_2 f_{ij} \text{ where } f_{ij} > 0, \text{ otherwise } 0.$$

3.2.2 Document Frequency (df_t)

Given a term t, its document frequency is the number of documents in the collection that contain the term. Not all query terms are not equally important, some words occur much more frequently than others.

$$idf_i = \log_2 \frac{N}{df_i}$$

Where, idf_i is inverse document frequency, N is the number of all the documents in the collection, and df_i is no. of documents in which that term occurs note that it's not counting number of times.

As mentioned in the assignment, we will use smoothed inversion frequency:

$$idf_i = \log_2(1 + \frac{N}{df_i})$$

3.2.3 Score calculation

$$w_{ij} = tf * idf > 0, \text{ otherwise } 0.$$

Where, tf and idf are for a query term.

3.2.4 Programming Details

For each term in the query, the score is found for each document as mentioned above.

Documents are sorted in the descending order based on their scores, and best k score documents are returned. Following is an output of the a query search:

```
51 0 AP880316-0292 1 63.53761980680801 STANDARD
51 0 AP880731-0085 2 60.517272880425615 STANDARD
51 0 AP880406-0267 3 60.041107489074896 STANDARD
51 0 AP880325-0293 4 56.06438942909777 STANDARD
51 0 AP880412-0268 5 50.98359130456502 STANDARD
51 0 AP880301-0271 6 34.614190451013215 STANDARD
51 0 AP880703-0076 7 34.614190451013215 STANDARD
51 0 AP880703-0077 8 34.614190451013215 STANDARD
51 0 AP891011-0255 9 34.614190451013215 STANDARD
51 0 AP880503-0244 10 34.614190451013215 STANDARD
52 0 AP891028-0117 1 70.52691227205305 STANDARD
52 0 AP891019-0140 2 62.01293750568279 STANDARD
52 0 AP891003-0173 3 61.151032552680746 STANDARD
52 0 AP880504-0025 4 59.55858657393395 STANDARD
52 0 AP890825-0238 5 56.182170117451896 STANDARD
52 0 AP891019-0092 6 56.182170117451896 STANDARD
52 0 AP880527-0206 7 55.21072149727618 STANDARD
52 0 AP891004-0090 8 55.181325546858346 STANDARD
```

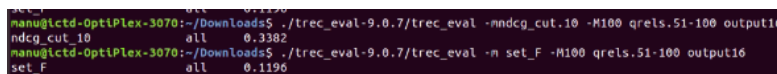
Figure 2: Output of a Search Query

4 How to Run?

- **Build Dictionary:**
\$ python3 invidx_cons.py assignment1_data/TaggedTrainingAP/ indexfile
- **Print Dictionary :**
last parameter (n) is optional
\$ python3 printdict.py indexfile.dict 10
- **Process Queries:**
\$ python3 vecsearch.py assignment1_data/topics.51-100 10 output2 indexfile.dict indexfile.idx
- **Caclulate nDCG score:**
\$./trec_eval-9.0.7/trec_eval -mndcg_cut.10 -M100 qrels.51-100 output
- **Calculate F1 score:**
\$./trec_eval-9.0.7/trec_eval -m set_F -M100 qrels.51-100 output

5 Scores

- nDCG - 0.3382
- F1 Score - 0.1196



```
manu@icld-OptiPlex-3070:~/Downloads$ ./trec_eval-9.0.7/trec_eval -mndcg_cut.10 -M100 qrels.51-100 output16
ndcg_cut_10      all      0.3382
manu@icld-OptiPlex-3070:~/Downloads$ ./trec_eval-9.0.7/trec_eval -m set_F -M100 qrels.51-100 output16
set_F           all      0.1196
```

Figure 3: Scores