Web Information Retrieval

Textbook by
Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze
Notes Revised by X. Meng for SEU
May 2014

Vector Space Model Algorithms and Query Expansion

Recap

- IR model
  - Document representation
  - Query representation
  - Retrieval function
- Vector space model
  - Term-document matrix
  - Cosine similarity
- In this lecture, we'll discuss algorithms that implement these models

Document Collection

- A collection of *n* documents can be represented in the vector space model by a term-document matrix.
- An entry in the matrix corresponds to the "weight" of a term in the document; zero means the term has no significance in the document or it simply doesn't exist in the document.

$$\begin{array}{c|cccc} & T_1 & T_2 & .... & T_t \\ D_1 & w_{11} & w_{21} & ... & w_{t1} \\ D_2 & w_{12} & w_{22} & ... & w_{t2} \\ : & : & : & & : \\ : & : & : & & : \\ D_n & w_{1n} & w_{2n} & ... & w_{tn} \end{array}$$

Term-Document Weight

- Assign a *tf-idf* weight for each term *t* in each document *d*:

$$w_{t,d} = (1+\log(tf_{t,d}))*\log(\frac{N}{df_t})$$

- Alternatively, we can define

$$w_{t,d} = \frac{tf_{t,d}}{\max(tf_{*,d})}*\log(\frac{N}{df_t})$$

5

Cosine Similarity Between Query and Document

$$\cos(\vec{q},\vec{d}) = \text{SIM}(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

- q_i is the *tf-idf* weight of term *i* in the query.
- d_i is the *tf-idf* weight of term *i* in the document.
- $|\vec{q}|$ and $|\vec{d}|$ are the lengths of $\vec{q}$ and $\vec{d}$.
- This is the cosine similarity of $\vec{q}$ and $\vec{d}$ ..... or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.
- How to compute the cosine similarity?
- We will need to compute
  - Term weight tf-idf
  - Vector lengths of query and documents
  - Inner product of query and document vectors

6

Exercise

- Compute the cosine similarity score for the following queries for the given document collection.
- Document collection
  - d1: "all you have ever wanted to know about cars"
  - d2: "information on trucks, information on planes, information on trains"
  - d3: "cops stop red cars more often"
- Stop words: *all, you, have, ever, to, about, on, more, often*
- q1: [information on cars]
- q2: [red cars and red trucks]

7

The Term-Document Matrix

Term frequency matrix

	car	cops	information	know	plane	red	stop	train	truck	want
d1	1			1						1
d2			3		1			1	1	
d3	1	1				1	1			
q1	1		1							
q2	1					2			1	

The Term-Document Matrix

tf-idf weighted matrix, $w_{i,j} = (1+\log(t_{i,j})) * \log(N/d_f)$ N = 3

Note that (1+log(1))*log(3/2) = 0.176, (1+log(1))*log(3/1) = 0.477, (1+log(3))*log(3/1) = 0.704

	car	cops	information	know	plane	red	stop	train	truck	want
d1	0.176			0.477						0.477
d2			0.704		0.477			0.477	0.477	
d3	0.176	0.477				0.477	0.477			
q1	0.176		0.477							
q2	0.176					0.621			0.477	

Also note: for queries, idf ~= idf in the original collection, only the tf in query is calculated separately.

Document Length

- |d1| = sqrt(0.176^2 + 0.477^2 + 0.477^2) = 0.697
- |d2| = sqrt(0.704^2 + 3*(0.477)^2) = 1.085
- |d3| = sqrt(0.176^2 + 0.477^2 + 0.477^2) = 0.697
- |q1| = sqrt(0.176^2 + 0.477^2) = 0.508
- |q2| = sqrt(0.176^2 + 0.621^2 + 0.477^2) = 0.802

Cosine Similarity

$$\text{sim}(q1, d1) = \frac{q1 \bullet d1}{|q1||d1|}$$

= (0.176*0.176)/(|q1|*|d1|)

= 0.0309/0.354 = 0.0873

$$\text{sim}(q1,d2) = \frac{q1 \bullet d2}{|q1||d2|}$$

= (0.477*0.704)/(|q1|*|d2|)

= 0.335/(0.508*1.085) = 0.335/0.468 = 0.716

sim(q1,d3) = 0.176^2/(0.508*0.697) = 0.0873

Query 1 is most relevant to doc 2

Computing IDF

Let N be the total number of documents;

For each token, T, in I (term list or dictionary):

Determine the total number of documents, M,

in which T occurs (the length of T's posting list);

Set the IDF for T to log(N/M);

Note this requires a second pass through all the tokens after all documents have been indexed.

Document Vector Length

- Remember that the length of a document vector is the square-root of sum of the squares of the weights of its tokens.
- Remember the weight of a token is:
  TF * IDF
- Therefore, must wait until IDF's are known (and therefore until all documents are indexed) before document lengths can be determined.

13

Computing Document Vector Lengths

Assume the length of all document vectors (stored in the DocumentReference) are initialized to 0.0;
For each token T in I:
  Let, *idf*, be the IDF weight of T;
  For each TokenOccurence of T in document D
    Let, C, be the count of T in D;
    Increment the length of D by $(idf*C)^2$;
For each document D in the document set:
  Set the length of D to be the square-root of the current stored length;

Here we use the raw count C as the weight, we could use (1+log C) as the weight

14

Retrieval with an Inverted Index

- Tokens that are not in both the query and the document do not effect cosine similarity.
  - Product of token weights is zero and does not contribute to the dot product.
- Usually the query is fairly short, and therefore its vector is *extremely* sparse.
- Use inverted index to find the limited set of documents that contain at least one of the query words.

15

Processing the Query

- Incrementally compute cosine similarity of each indexed document as query words are processed one by one.
- To accumulate a total score for each retrieved document, store retrieved documents in a hashtable, where DocumentReference is the key and the partial accumulated score is the value.
- Remember that we are computing cosine similarity:

$$\cos(\vec{q},\vec{d}) = \text{SIM}(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

16

Inverted-Index Retrieval Algorithm

Create a vector, Q, for the query.
Create empty HashMap, R, to store retrieved documents with scores.
For each token, T, in Q:
  Let idf be the IDF of T, and K be the count of T in Q;
  Set the weight of T in Q: W = K * idf;
  Let L be the list of TokenOccurences of T from I (term list);
  For each TokenOccurence, O, in L:
    Let D be the document of O, and C be the count of O (tf of T in D);
    If D is not already in R (D was not previously retrieved)
      Then add D to R and initialize score to 0.0;
    Increment D's score by W * idf * C; (product of T-weight in Q and D)

17

Retrieval Algorithm (cont)

Compute the length, L, of the vector Q (square-root of the sum of the squares of its weights).
For each retrieved document D in R:
  Let S be the current accumulated score of D;
  (S is the dot-product of D and Q)
  Let Y be the length of D as stored in its DocumentReference;
  Normalize D's final score to S/(L * Y);
Sort retrieved documents in R by final score and return results in an array.

18

User Interface

Until user terminates with an empty query:
- Prompt user to type a query, Q.
- Compute the ranked array of retrievals R for Q;
- Print the name of top N documents in R;
- Until user terminates with an empty command:
  - Prompt user for a command for this query result:
    - 1) Show next N retrievals;
    - 2) Show the Mth retrieved document;

19

Query Reformulation

Revised from Professor Mooney's notes
for SEU
Spring 2014

20

Query Reformulation and Relevance Feedback

- When a query is submitted from the user, the IR system (search engine) may re-formulate the query in two ways
  - Expanding the query automatically based on the synonyms of the original query terms
  - Expanding the query based on the feedback to the initial set of search results from the user

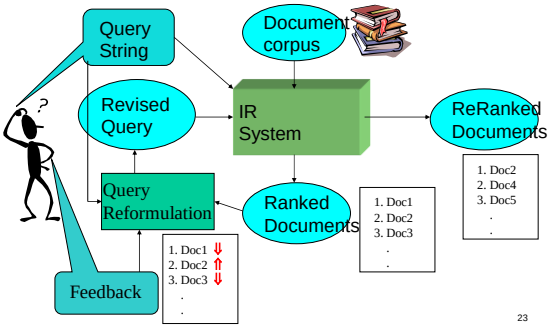
21

Relevance Feedback

- After initial retrieval results are presented, allow the user to provide feedback on the relevance of one or more of the retrieved documents.
- Use this feedback information to reformulate the query.
- Produce new results based on reformulated query.
- Allows more interactive, multi-pass process.

22

Relevance Feedback Architecture



23

Query Reformulation

- Revise query to account for feedback:
  - Query Expansion: Add new terms to query from relevant documents.
  - Term Reweighting: Increase weight of terms in relevant documents and decrease weight of terms in irrelevant documents.
- Several algorithms for query reformulation.

24

Query Reformulation

- Change query vector using vector algebra.
- **Add** the vectors for the **relevant** documents to the query vector.
- **Subtract** the vectors for the **irrelevant** docs from the query vector.
- This both adds both positive and negatively weighted terms to the query as well as reweighting the initial terms.

25

Optimal Query

- Assume that the relevant set of documents C_r are known.
- Then the best query that ranks all and only the relevant queries at the top is:

$$\vec{q}_{opt} = \frac{1}{|C_r|} \sum_{\forall \vec{d}_j \in C_r} \vec{d}_j - \frac{1}{N - |C_r|} \sum_{\forall \vec{d}_j \notin C_r} \vec{d}_j$$

Where N is the total number of documents.

26

Standard Rocchio Method

- Since all relevant documents unknown, just use the **known** relevant (D_r) and irrelevant (D_n) sets of documents and include the initial query q.

$$\vec{q}_m = \alpha \vec{q} + \frac{\beta}{|D_r|} \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \frac{\gamma}{|D_n|} \sum_{\forall \vec{d}_j \in D_n} \vec{d}_j$$

α: Tunable weight for initial query.
β: Tunable weight for relevant documents.
γ: Tunable weight for irrelevant documents.

27

Ide Regular Method

- Since more feedback should perhaps increase the degree of reformulation, do not normalize for amount of feedback:

$$\vec{q}_m = \alpha \vec{q} + \beta \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \gamma \sum_{\forall \vec{d}_j \in D_n} \vec{d}_j$$

α: Tunable weight for initial query.
β: Tunable weight for relevant documents.
γ: Tunable weight for irrelevant documents.

28

Ide "Dec Hi" Method

- Bias towards rejecting **just** the highest ranked of the irrelevant documents:

$$\vec{q}_m = \alpha \vec{q} + \beta \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \gamma \max_{non-relevant}(\vec{d}_j)$$

α: Tunable weight for initial query.
β: Tunable weight for relevant documents.
γ: Tunable weight for irrelevant document.

29

Comparison of Methods

- Overall, experimental results indicate no clear preference for any one of the specific methods.
- All methods generally improve retrieval performance (recall & precision) with feedback.
- Generally just let tunable constants (α, β, and γ) equal 1.

30

Evaluating Relevance Feedback

- By construction, reformulated query will rank explicitly-marked relevant documents higher and explicitly-marked irrelevant documents lower.
- Method should not get credit for improvement on *these* documents, since it was told their relevance.
- In machine learning, this error is called "testing on the training data."
- Evaluation should focus on generalizing to **other** un-rated documents.

31

Fair Evaluation of Relevance Feedback

- Remove from the corpus any documents for which feedback was provided.
- Measure recall/precision performance on the remaining *residual collection*.
- Compared to complete corpus, specific recall/precision numbers may decrease since relevant documents were removed.
- However, **relative** performance on the residual collection provides fair data on the effectiveness of relevance feedback.

32

Why is Feedback Not Widely Used

- Users sometimes reluctant to provide explicit feedback.
- Results in long queries that require more computation to retrieve, and search engines process lots of queries and allow little time for each one.
- Makes it harder to understand why a particular document was retrieved.

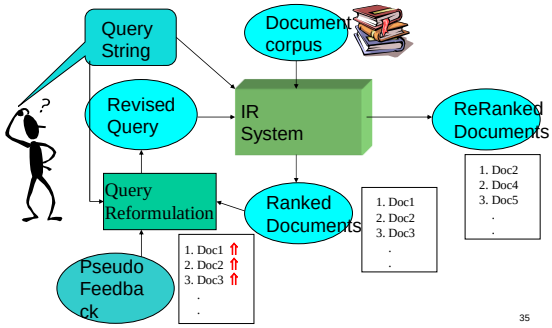
33

Pseudo Feedback

- Use relevance feedback methods without explicit user input.
- Just **assume** the top *m* retrieved documents are relevant, and use them to reformulate the query.
- Allows for query expansion that includes terms that are correlated with the query terms.

34

Pseudo Feedback Architecture



35

PseudoFeedback Results

- Found to improve performance on TREC competition ad-hoc retrieval task.
- Works even better if top documents must also satisfy additional boolean constraints in order to be used in feedback.

36

Thesaurus

- A thesaurus provides information on synonyms and semantically related words and phrases.
- Example:

```
physician
   syn: doc, doctor, MD, medical,
mediciner, medico
   rel: medic, general
practitioner, surgeon
```

37

Thesaurus-based Query Expansion

- For each term, *t*, in a query, expand the query with synonyms and related words of *t* from the thesaurus.
- May weight added terms less than original query terms.
- Generally increases recall.
- May significantly decrease precision, particularly with ambiguous terms.
  - "interest rate" → "interest rate fascinate evaluate"

38

WordNet

- A more detailed database of semantic relationships between English words.
- http://wordnet.princeton.edu/
- Developed by famous cognitive psychologist George Miller and a team at Princeton University.
- About 144,000 English words.
- Nouns, adjectives, verbs, and adverbs grouped into about 109,000 synonym sets called *synsets*.

39

WordNet Synset Relationships

- Antonym: front → back
- Attribute: benevolence → good (noun to adjective)
- Pertainym: alphabetical → alphabet (adjective to noun)
- Similar: unquestioning → absolute
- Cause: kill → die
- Entailment: breathe → inhale
- Holonym: chapter → text (part-of)
- Meronym: computer → cpu (whole-of)
- Hyponym: tree → plant (specialization)
- Hypernym: fruit → apple (generalization)

40

WordNet Query Expansion

- Add synonyms in the same synset.
- Add hyponyms to add specialized terms.
- Add hypernyms to generalize a query.
- Add other related terms to expand query.

41

Statistical Thesaurus

- Existing human-developed thesauri are not easily available in all languages.
- Human thesuari are limited in the type and range of synonymy and semantic relations they represent.
- Semantically related terms can be discovered from statistical analysis of corpora.

42

Automatic Global Analysis

- Determine term similarity through a pre-computed statistical analysis of the complete corpus.
- Compute association matrices which quantify term correlations in terms of how frequently they co-occur.
- Expand queries with statistically most similar terms.

Association Matrix

	w_1	w_2	w_3	.....	w_n
w_1	c_{11}	c_{12}	c_{13}	.....	c_{1n}
w_2	c_{21}				
w_3	c_{31}				
.	.				
.	.				
w_n	c_{n1}				

c_{ij}: Correlation factor between term i and term j

$$c_{ij} = \sum_{d_k \in D} f_{ik} \times f_{jk}$$

f_{ik} : Frequency of term i in document k

Normalized Association Matrix

- Frequency based correlation factor favors more frequent terms.
- Normalize association scores:

$$s_{ij} = \frac{c_{ij}}{c_{ii} + c_{jj} - c_{ij}}$$

- Normalized score is 1 if two terms have the same frequency in all documents.

Metric Correlation Matrix

- Association correlation does not account for the proximity of terms in documents, just co-occurrence frequencies within documents.
- Metric correlations account for term proximity.

$$c_{ij} = \sum_{k_u \in V_i} \sum_{k_v \in V_j} \frac{1}{r(k_u, k_v)}$$

V_i: Set of all occurrences of term i in any document.

$r(k_u,k_v)$: Distance in words between word occurrences k_u and k_v (∞ if k_u and k_v are occurrences in different documents).

Normalized Metric Correlation Matrix

- Normalize scores to account for term frequencies:

$$s_{ij} = \frac{c_{ij}}{|V_i| \times |V_j|}$$

Query Expansion with Correlation Matrix

- For each term i in query, expand query with the n terms, j, with the highest value of c_{ij} (s_{ij}).
- This adds semantically related terms in the "neighborhood" of the query terms.

Problems with Global Analysis

- Term ambiguity may introduce irrelevant statistically correlated terms.
  - "Apple computer" → "Apple red fruit computer"
- Since terms are highly correlated anyway, expansion may not retrieve many additional documents.

Automatic Local Analysis

- At query time, dynamically determine similar terms based on analysis of top-ranked retrieved documents.
- Base correlation analysis on only the "local" set of retrieved documents for a specific query.
- Avoids ambiguity by determining similar (correlated) terms only within relevant documents.
  - "Apple computer" → "Apple computer Powerbook laptop"

Global vs. Local Analysis

- Global analysis requires intensive term correlation computation only once at system development time.
- Local analysis requires intensive term correlation computation for every query at run time (although number of terms and documents is less than in global analysis).
- But local analysis gives better results.

Global Analysis Refinements

- Only expand query with terms that are similar to *all* terms in the query.

$$sim(k_i, Q) = \sum_{k_j \in Q} c_{ij}$$

  - "fruit" not added to "Apple computer" since it is far from "computer."
  - "fruit" added to "apple pie" since "fruit" close to both "apple" and "pie."
- Use more sophisticated term weights (instead of just frequency) when computing term correlations.

Query Expansion Conclusions

- Expansion of queries with related terms can improve performance, particularly recall.
- However, must select similar terms very carefully to avoid problems, such as loss of precision.