Information Retrieval and Organisation

Dell Zhang

Birkbeck, University of London

Scoring, Term Weighting, and the Vector Space Model

Problems with Boolean Queries

- Thus far, our queries have all been Boolean.
  - Documents either match or don't.
- Good
  - for expert users with precise understanding of their needs and the collection; and
  - for software applications which can easily consume 1000s of results.
- Not good
  - for the majority of users, as
    (1) they are unable or unwilling to write Boolean queries; and
    (2) they don't want to wade through 1000s of results, which is is particularly true of web search.

Ranked Retrieval

- Boolean queries often result in either too few (=0) or too many (1000s) results.
  - Query 1: "standard user dlink 650"
    $\rightarrow$ 200,000 hits
  - Query 2: "standard user dlink 650 no card found"
    $\rightarrow$ 0 hits
- It takes a lot of skill to come up with a query that produces a manageable number of hits.
- With a ranked list of documents, it does not matter how large the retrieved set is.

Scoring Documents

- We wish to return in order the documents most likely to be useful to the searcher.
- How can we rank-order the documents in the collection with respect to a query?
- We need a way of assigning a score to a query/document pair.
- This score measures how well document and query "match".

Query-Document Matching Scores

- Let's start with a simple approach
- Count how many of the query terms appear in a document:

$$\text{score}(Q, D) = |Q \cap D|$$

- It can be computed easily
- However, it is very biased towards large documents
  - Large documents have a greater chance of getting a higher score (they just contain more terms)
  - Bigger is not always better ...

Jaccard Coefficient

- We need some way of normalizing the score
- Why not use Jaccard coefficient?

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

  - A and B don't have to be the same size.
  - Always assigns a number between 0 and 1.
    - $\text{Jaccard}(A, B) = 1$ if $A = B$
    - $\text{Jaccard}(A, B) = 0$ if $A \cap B = 0$

Jaccard Coefficient

- What's wrong with Jaccard coefficient?
  - Having a higher term frequency makes a document more relevant
    - How many occurrences does a term have in a document?
  - Rare terms are more informative than frequent terms
    - How often does a term occur in a document collection?
- Jaccard coefficient doesn't consider such information.
- We need a more sophisticated way of normalizing for length.

Binary Incidence Matrix

- Up to now, we used a binary incidence matrix
  - Each document represented by binary vector $\in \{0,1\}^{|V|}$.

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Anthony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

Term Frequency Matrix

- We will now use a matrix containing the term frequencies:
  - Each document represented by count vector $\in \mathbb{N}^{|V|}$

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Anthony	157	73	0	0	0	1	
Brutus	4	157	0	2	0	0	
Caesar	232	227	0	2	1	0	
Calpurnia	0	10	0	0	0	0	
Cleopatra	57	0	0	0	0	0	
mercy	2	0	3	8	5	8	
worser	2	0	1	1	1	5	
...							

Bag of Words Model

- For now, we do not consider the order of words in a document.
  - "John is quicker than Mary" and
    "Mary is quicker than John"
    are represented the same way.
- This is called a *bag of words model*.
  - In a sense, this is a step back: The positional index was able to distinguish these two documents.
  - We will look at recovering positional information later in this module.

Term Frequency TF

- The term frequency $\text{tf}_{t,d}$ of term t in document d is defined as: the number of times that t occurs in d.
- We want to use tf when computing query-document match scores.
- However, raw term frequency is often not what we want.
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term, but not 10 times more relevant.
  - Relevance does not increase proportionally with term frequency.

Term Frequency Weighting

- The effect of non-proportional increases can be seen in other areas as well
  - Economics: The law of diminishing returns (e.g., sowing)
  - Biology: Human senses operate logarithmically (e.g., 10 times increase in sound volume is perceived as being twice as loud)
- The term frequencies can be weighted in a similar way

Log Frequency Weighting

- The log frequency weight of term t in d is defined as follows

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

  - $0 \rightarrow 0$, $1 \rightarrow 1$, $2 \rightarrow 1.3$, $10 \rightarrow 2$, $1000 \rightarrow 4$, etc.
  - The score for a document-query pair: sum over terms t in both q and d:
    matching-score $= \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$
  - The score is 0 if none of the query terms is present in the document.

Document Frequency

- A document containing a query term is more likely to be relevant than a document that doesn't, but that's not the whole story
- Rare terms are more informative than frequent terms.
  - For instance, a collection of documents on the auto industry is likely to have the term `auto` in almost every document:
    a document containing the term `auto` is not very relevant for a query containing the term `auto`
  - Now, consider a term in the query that is rare in the collection (e.g., `arachnocentric`):
    a document containing this term is very likely to be relevant.

Document Frequency

- We want to have high weights for rare terms; and low weights (but still larger than 0) for common terms.
- We will use document frequency to factor this into computing the matching score.
- The higher the document frequency, the lower the weight (and vice versa)

Inverse Document Frequency

- The document frequency df_t of term t is the number of documents that t occurs in (with N documents in the collection).
  - df_t is an inverse measure of t's *informativeness*.
- We define the idf weight of term t as follows (note the logarithmic weighting):

$$\mathrm{idf}_t = \log_{10} \frac{N}{\mathrm{df}_t}$$

  - idf_t is a measure of of t's *informativeness*.

Effect on Ranking

- The idf affects the ranking of documents only if the query has at least two terms.
  - For example, in the query "`arachnocentric line`", idf weighting increases the relative weight of '`arachnocentric`' and decreases the relative weight of '`line`'.
- The idf has no effect on ranking for one-term queries.

TF-IDF Weighting

- The tf-idf weight of a term is the *product* of its tf weight and its idf weight:

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$

  - One of the best known weighting scheme in information retrieval
  - Note: the "-" in tf-idf is a hyphen, not a minus sign!
  - Alternative names: tf.idf, tfxidf

Weight Matrix

- We will now use a matrix containing the tf-idf weights:
  - Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Anthony	5.25	3.18	0.0	0.0	0.0	0.35	
Brutus	1.21	6.10	0.0	1.0	0.0	0.0	
Caesar	8.59	2.54	0.0	1.51	0.25	0.0	
Calpurnia	0.0	1.54	0.0	0.0	0.0	0.0	
Cleopatra	2.85	0.0	0.0	0.0	0.0	0.0	
mercy	1.51	0.0	1.90	0.12	5.25	0.88	
worser	1.37	0.0	0.11	4.15	0.25	1.95	
...							

Documents as Vectors

- So we have a $|V|$-dimensional real-valued **vector space**.
- Terms are *axes* of the space.
- Documents are *points* or *vectors* in this space.
  - Very high-dimensional: tens of millions of dimensions when you apply this to web search.
  - Very sparse vector — most entries are zero.

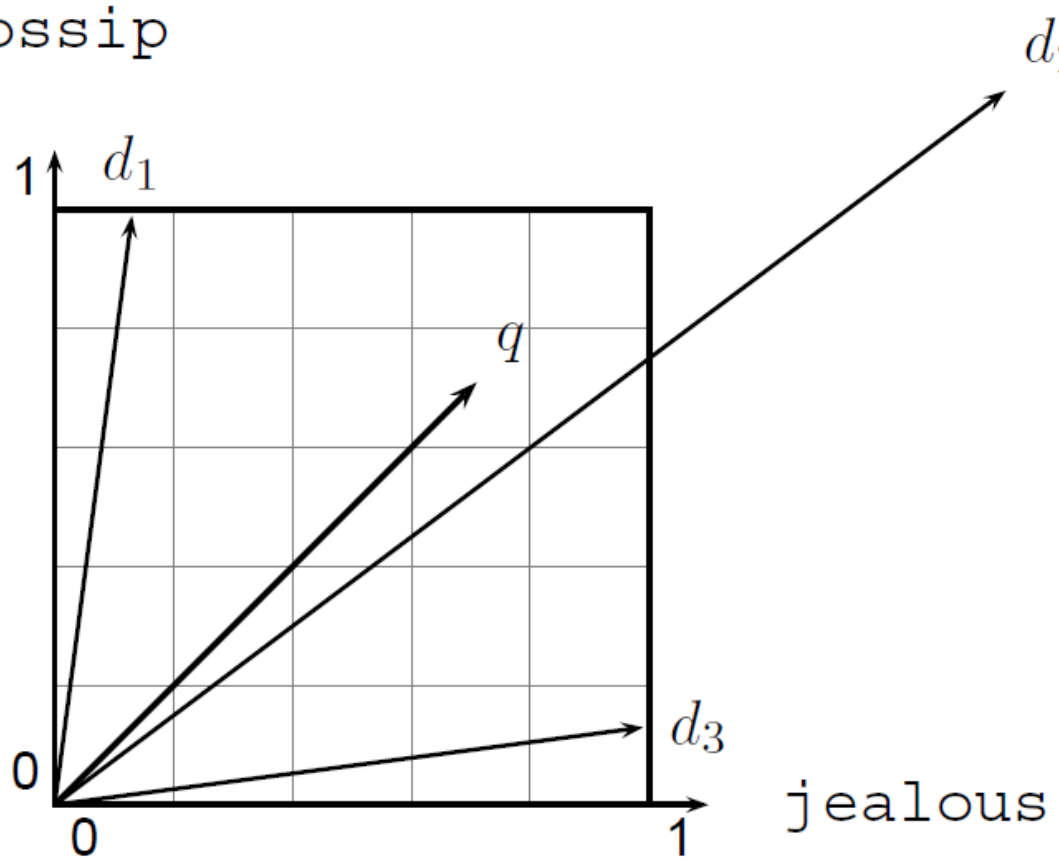
Queries as Vectors

- Key idea 1: do the same for queries: represent them as vectors in the space
- Key idea 2: rank documents according to their proximity to the query (proximity = similarity)
  - Recall: we're doing this because we want to get away from the you're-either-in-or-out Boolean model.
  - Instead: rank more relevant documents higher than less relevant documents

Formalizing Vector Space Similarity

- First cut: the distance between two points (i.e., the end points of the two vectors)
- Euclidean distance?
  - Euclidean distance is a bad idea . . .
  - . . . because Euclidean distance is large for vectors of different lengths.

Why Distance is a Bad Idea

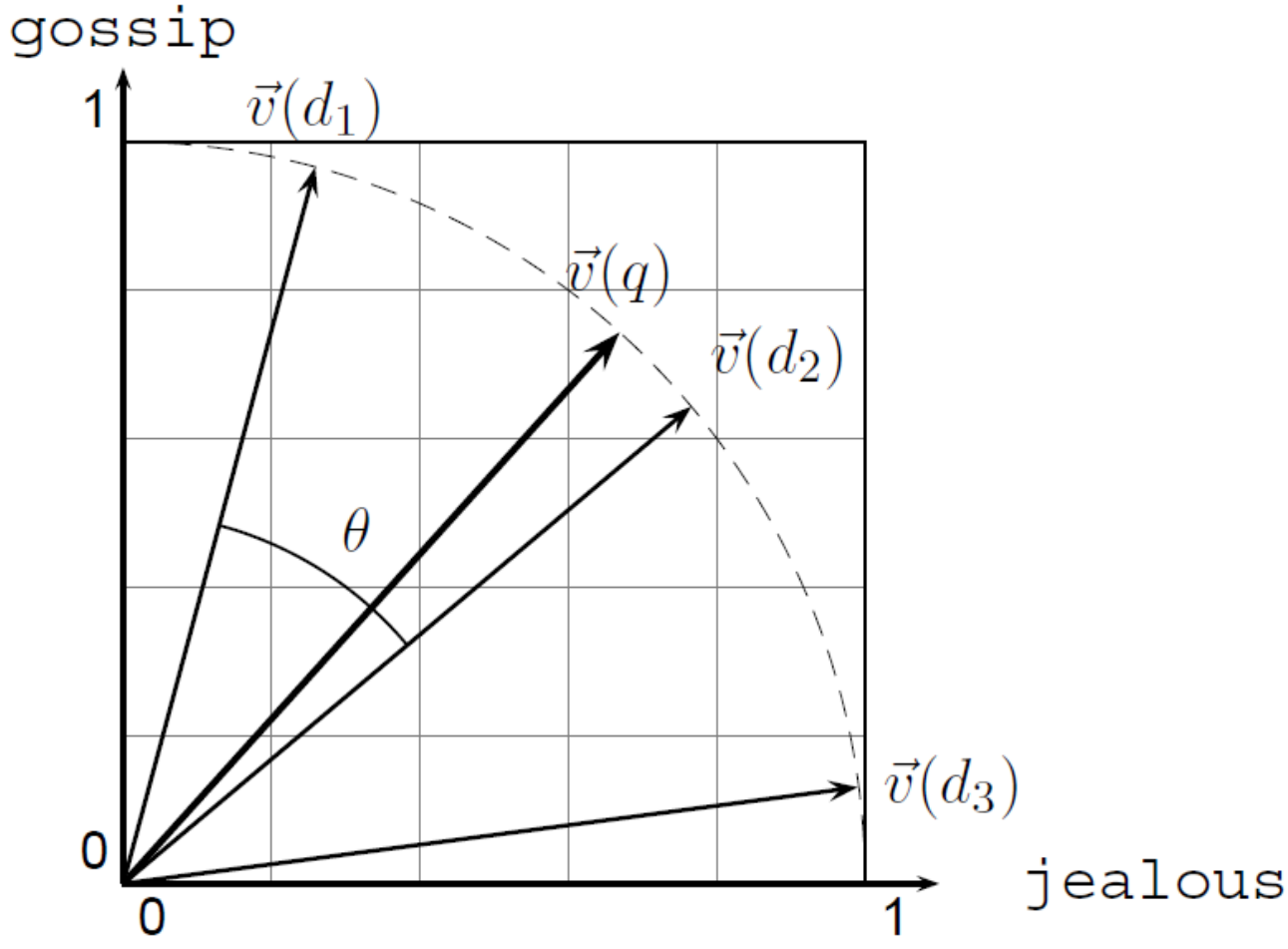


The Euclidean distance of $\vec{q}$ and $\vec{d_2}$ is large although the distribution of terms in the query q and the distribution of terms in the document d_2 are very similar

Use Angle Instead of Distance

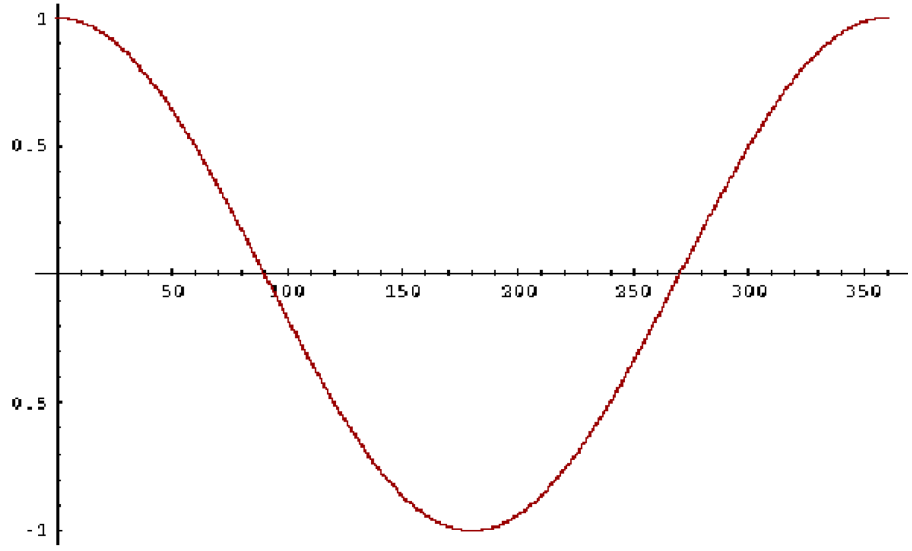
- Rank the documents according to their *angles* with the query
  - Thought experiment: take a document d and append it to itself, call this document d'
  - "Semantically" d and d' have the same content.
  - The angle between the two documents is 0, corresponding to maximal similarity.
  - The Euclidean distance between the two documents can be quite large.
  - Thus, measuring the angle θ between the query vector and a document vector is much better.

Illustration



From Angles to Cosines

- As all vector components are ≥ 0, all vectors are in the same quadrant
- We only have angles between 0° and 90°
- The cosine is a monotonically decreasing function of the angle for the interval $[0^\circ, 90^\circ]$
  - The larger the angle θ, the smaller the cosine of θ
  - The smaller the angle θ, the larger the cosine of θ



From Angles to Cosines

- The following two notions are equivalent.
  - Rank documents according to the *angle* between query and document in increasing order
  - Rank documents according to the *cosine* of the angle(query,document) in decreasing order
- The cosine of an angle can be computed more easily than the angle itself

Computing the Cosine

- The cosine between a vector $\vec{x}$ and a vector $\vec{y}$ is computed as follows:

$$\cos\theta = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| \cdot |\vec{y}|}$$

where $\cdot$ is the *dot product* (or *inner product*) of vectors $\vec{x} \cdot \vec{y} = \sum_{i=1}^{k} x_i y_i$ and $|\vec{x}| = \sqrt{\sum_{i=1}^{k} x_i^2}$ is the length of a vector.

Computing the Cosine

- So the matching-score of a document d_j with regard to a query q is

$$\frac{\vec{q} \cdot \vec{d_j}}{|\vec{q}| \cdot |\vec{d_j}|}$$

  - The vectors $\vec{q}$ and $\vec{d_j}$ are made up of tf-idf weights
  - The length is used for normalization purposes (every matching-score is between 0 and 1)

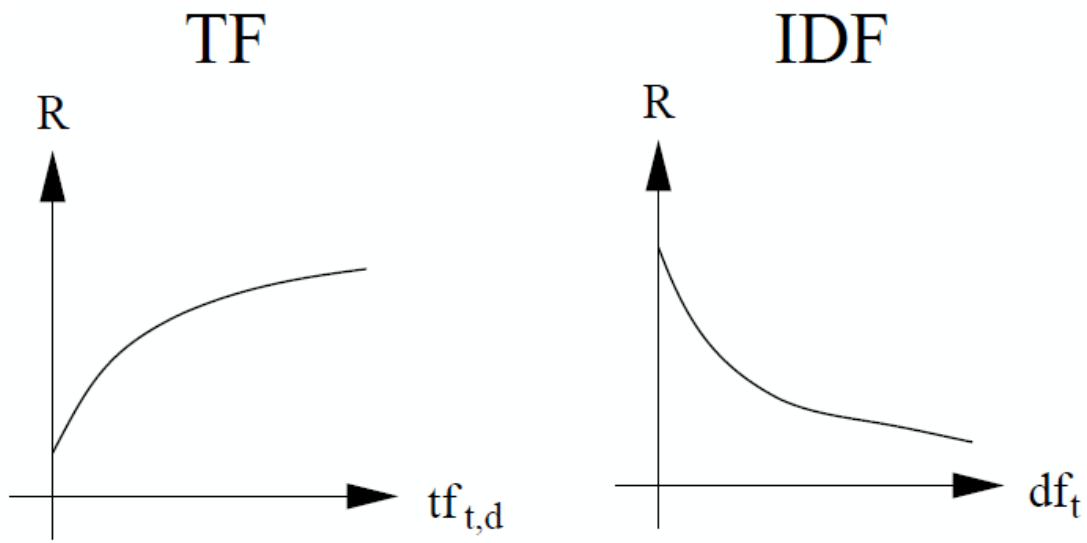
Algorithm

```
CosineScore(q)
 1  float Scores[N] = 0
 2  Initialize Length[N]
 3  for each query term t
 4  do calculate w_{t,q} and fetch postings list for t
 5      for each pair(d, tf_{t,d}) in postings list
 6      do Scores[d] += wf_{t,d} × w_{t,q}
 7  Read the array Length[d]
 8  for each d
 9  do Scores[d] = Scores[d]/Length[d]
10  return Top K components of Scores[]
```

- The array *Length* contains the lengths of each document (used for normalization)
- We don't need to divide by the query length (as this is just a constant factor)

Variants

- There are variants for tf-idf factors: a ranking is called a tf-idf ranking, when the importance of a document
  - increases with the number of occurrences within a document
  - decreases with the number of occurrences of the term in the collection



Variants

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1+\log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5+\frac{0.5\times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$ (Section 6.4.4)
b (boolean)	$\begin{cases}1 & \text{if } \mathrm{tf}_{t,d}>0\\ 0 & \text{otherwise}\end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha<1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

- The logarithmic one is most popular
- According to Zobel and Moffat, there is no big difference in terms of quality for most tf-idf heuristics

Variants

- We often use *different weightings* for queries and documents.
- Notation: qqq.ddd
  - Example: ltn.lnc
    - query: logarithmic tf, idf, no normalization
    - document: logarithmic tf, no df weighting, cosine normalization
  - bnn.ltc can be computed quite efficiently
    - Only multiplication with 0 or 1 in line 6 of the algorithm

Summary

- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the cosine similarity between the query vector and each document vector
- Rank documents with respect to the query
- Return the top k (e.g., $k = 20$) to the user