Information Retrieval

Part 2
Indexing and TF-IDF

Qianni Zhang

Roadmap of this lecture

1. Generating Document Representations
2. Boolean v.s. ranked retrieval
3. Indexing and TF-IDF
   a. Index Term Weighting
   b. TF-IDF
   c. Inverted file (index, list)
4. Weighting schemes
5. Vector space scoring
6. Other term weighting techniques

Documents in IR

Document types

- **Monomedia document:** text document, etc.
- **Multimedia document:** document containing parts of different media
- **Hypertext document:** document with links; referred to as non-linear document
- **Hypermedia document:** multimedia + hypertext

- **User generated (content) document:** blogs, comments, tweets

Bag of Words

DOCUMENT

This is a document in
information retrieval

INDEX

document
information
retrieval
Is
this

Tokenization

- Treat text as sequence of discrete tokens or words
- Filter irrelevant words: Simplest approach is to ignore all numbers and punctuation and use only case-insensitive unbroken strings of alphabetic characters as tokens
- However, sometimes punctuation is meaningful. Examples: Emails, URLs
- More elaborated technique:
  - Analyse ? ! ; : “ ‘ [ ] ( ) < >
  - Extract question words: what? why? when?

Stopword Removal

- Very frequent words are not good discriminators
- list of stopwords
  - *a, about, above, according, across, after, afterwards, again, against, albeit, all, almost, alone, already, also, although, always, among, as, at*
- Stopwords are language dependent
- For efficiency, they are stored in a lookup table

Tokenization & stopwords (exercise)

Huge amounts of digital visual content are currently available in unstructured, non-indexed form. The availability of a wide range of digital recorders accessible to anyone, from cheap digital cameras to complex professional movie capturing devices, is making possible that the stocks of digital libraries already packed with visual content continue rising rapidly. This trend has generated an acute need for full automatic annotation systems able to index images at the same speed the population of digital libraries is growing. The lack of proper indexing and retrieval systems is rendering useless significant portions of available information. Actually, generating digital content is easy and cheap, managing and structuring it to produce effective services is not. This applies to the whole range of content owners, from professional digital libraries with their terabytes of visual content to the private collector of digital pictures stored in the disks of conventional personal computers.

(147 words)

Filtered result

huge amounts digital visual content are currently available unstructured nonindexed form availability wide range digital recorders accessible anyone cheap digital cameras complex professional movie capturing devices is making possible stocks digital libraries already packed visual content continue rising rapidly trend has generated acute need full automatic annotation systems able index images same speed population digital libraries is growing lack proper indexing retrieval systems is rendering useless significant portions available information actually generating digital content is easy cheap managing structuring produce effective services is this applies whole range content owners professional digital libraries terabytes visual content private collector digital pictures stored disks conventional personal computers

(104 words)

Lemmatization

- Reduce inflectional or variations of a word/verb to the base word
  - *am, are, is* $\rightarrow$ *be*
  - *dog, dogs, dog's* $\rightarrow$ *dog*
- *the dogs are playing in the garden* $\rightarrow$ *the dog is play in the garden*
- Reduce vocabulary size
- Generally improve recall but hurt precision
- A list of grammatical rules is needed
- A list of irregular words

Stemming

- Simplest: suffix stripping
  - *develop → develop*
  - *developing → develop*
  - *development → develop*
  - *developments → develop*

- It is language specific and can become complex.
- Stemming “blindly” strips off known affixes (prefixes and suffixes) in an iterative fashion.
  - *sses → ss*
  - *ies → i*
  - *ational → ate*
  - *tional → tion*

- The effectiveness of stemming:
  - For English: increase in recall doesn’t compensate loss in precision

Statistical Properties of Text

Important questions:

- How is the frequency words in a language (English)?
- How fast does vocabulary size grow wrt. the size of the database?
- This can be used to select appropriate term weights and other strategies of IR systems.
- In English few words are very common, e.g. “the”, “of” can account for about 10% of words in a text
- Some words are very rare: half the words in a text appear only once
- “Heavy tailed” distribution, since most of the probability mass is in the “tail”

Bag of words

- Tokenization
- Stopwords removal
- Lemmatization
- Stemming

- What next?

The Boolean Model

- A classical information retrieval (IR) model
- The first and most adopted one
- Used by many IR systems to this day

- Simple model based on set theory
- Queries specified as boolean expressions
- Terms are either present or absent (boolean)

The Boolean Model

- Record whether a document contains a word: document is binary vector in {0,1}
- Idea: Query satisfaction = **overlap measure** = $|Q \cap D|$

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	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

- Formal definition of Boolean model will be given in the next lecture

The Boolean Model

Drawbacks

- It does not allow context or partial matching
- No ranking of the documents is provided
- The Boolean queries are often too simplistic
- The Boolean model frequently returns either too few or too many documents (empty answer and many answer problem)
- Imposes a binary criterion for deciding relevance

The Boolean Model

- 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.
  - Also good for applications: Applications can easily consume 1000s of results.
- Not good for the majority of users.
  - Most users incapable of writing Boolean queries (or they are, but they think it’s too much work).
  - Most users don’t want to wade through 1000s of results.
  - This is particularly true of web search.

Problem with Boolean search: feast or famine

- Boolean queries often result in either too few (=0) or too many (1000s) results.
  - Query 1: "*standard user dlink 650*" → 200,000 hits
  - Query 2: "*standard user dlink 650 no card found*": 0 hits
- It takes a lot of skill to come up with a query that produces a manageable number of hits.
  - AND gives too few; OR gives too many

Query Engine

- Process query
- Look-up the index
- Retrieve list of documents
- Order documents
  - Content relevance
  - Link analysis
  - Popularity
- Prepare results page

Today's question: Given a large list of documents that match a query, how to order them according to their relevance?

Answer: Scoring Documents

- Given document *d*
- Given query *q*
- Calculate *score(q,d)*
- Rank documents in decreasing order of *score(q,d)*

- Generic Model: Documents = bag of [unordered] words (in set theory a bag is a multiset)
- A document is composed of terms
- A query is composed of terms
- *score(q,d)* will depend on terms

Ranked retrieval models

- Rather than a set of documents satisfying a query expression, in ranked retrieval, the system returns an ordering over the (top) documents in the collection for a query
- Free text queries: Rather than a query language of operators and expressions, the user’s query is just one or more words in a human language
- In principle, there are two separate choices here, but in practice, ranked retrieval has normally been associated with free text queries and vice versa

Feast or famine: not a problem in ranked retrieval

- When a system produces a ranked result set, large result sets are not an issue
  - Indeed, the size of the result set is not an issue
  - We just show the top k (≈ 10) results
  - We don’t overwhelm the user
  - Premise: the ranking algorithm works

Scoring as the basis of ranked retrieval

- 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?
- Assign a score – say in [0, 1] – to each document
- This score measures how well document and query "match".

Query-document matching scores

- We need a way of assigning a score to a query/document pair
- Let’s start with a one-term query
- If the query term does not occur in the document: score should be 0
- The more frequent the query term in the document, the higher the score (should be)
- We will look at a number of alternatives for this.

Binary term-document matrix

Recall slide 14

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	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

Each document is represented by a binary vector ∈ {0,1}

Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector in $\mathbb{N}^{v}$: a column below

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

Bag of words model

- Vector representations do not consider the ordering of words in a document
- Thus the documents
  - *John is quicker than Mary*
  - *Mary is quicker than John*
- have the same vectors
- are indistinguishable
- This is called the bag of words model.

Index Term Weighting

Effectiveness of an indexing language:

- Exhaustivity
  - number of different topics indexed
  - high exhaustivity: high recall and low precision
- Specificity
  - ability of the indexing language to describe topics precisely high specificity:
  - high precision and low recall

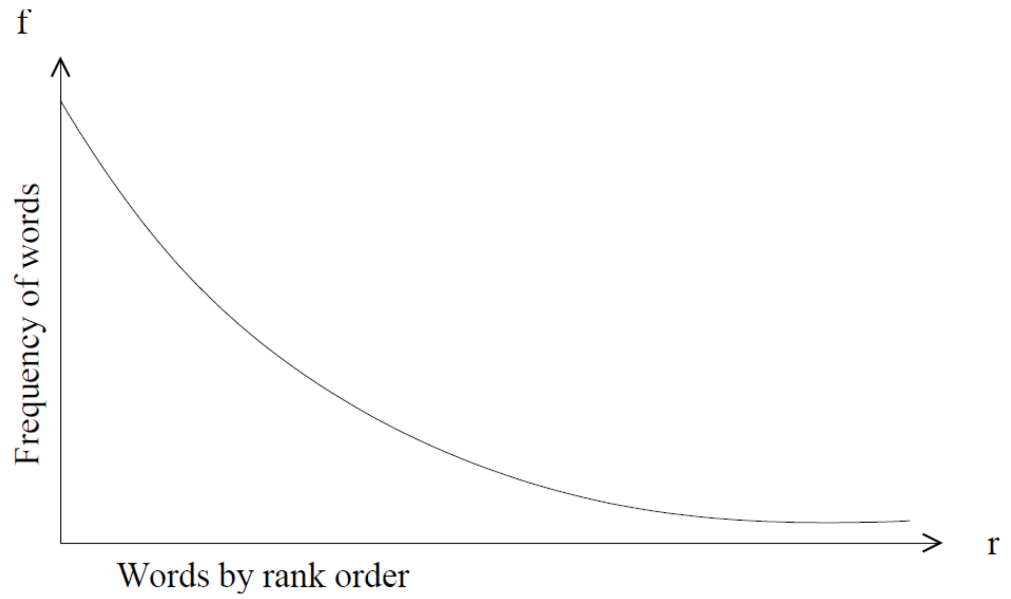
Index Term Weighting

- Term frequency (tf): frequency of a term in document (number of “locations” at which a term occurs)
- Document frequency (df): number of documents in which a term occurs

Index Term Weighting

Zipf's law [1949]

Distribution of word frequencies is similar for different texts (natural language) of significantly large size



•given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table

Index Term Weighting

Zipf’s law [1949]

- The most frequent word will occur approximately twice as often as the second most frequent word
- Example: “the” accounts for nearly 7% and “of” accounts for slightly over 3.5%
- Zipf’s law holds for several languages

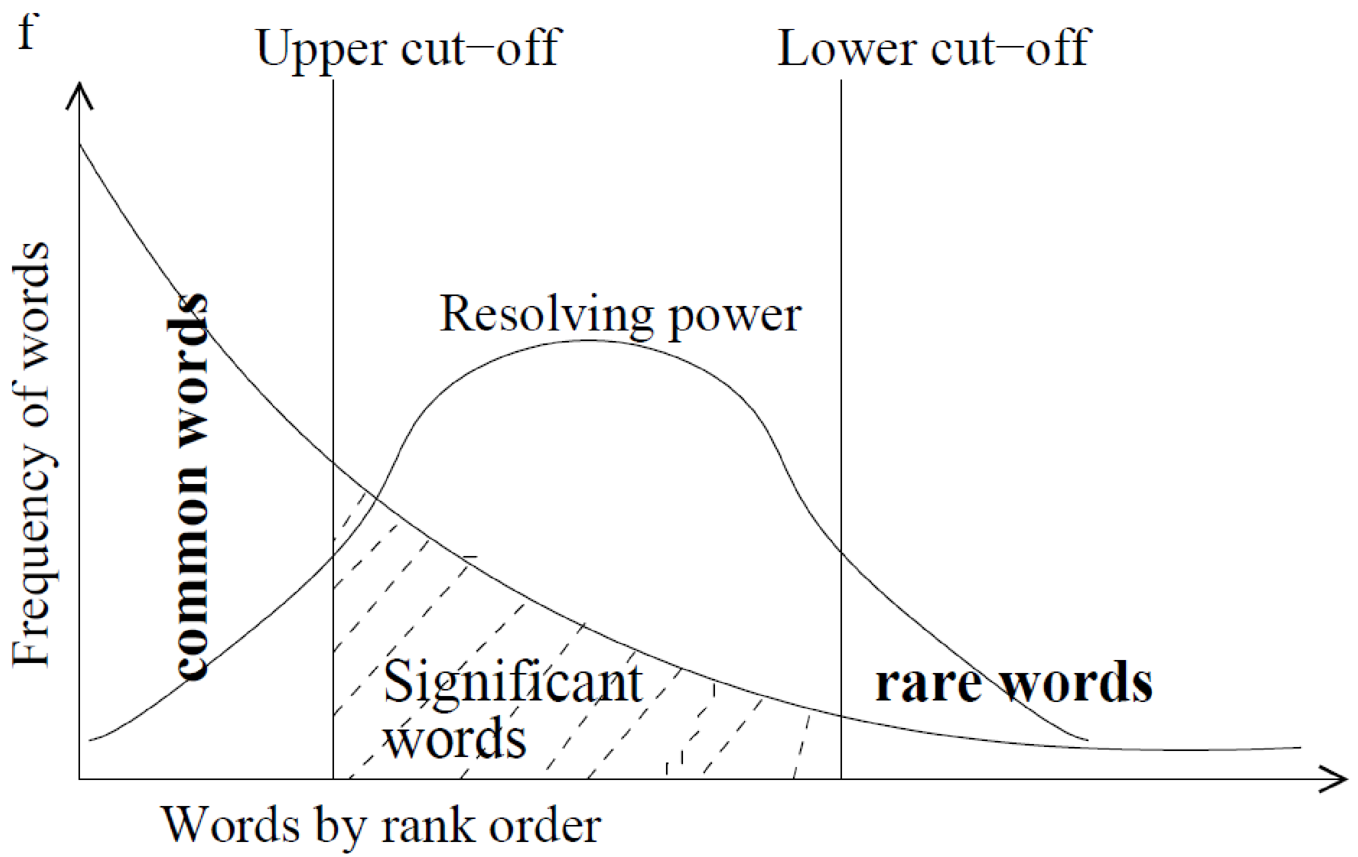
Index Term Weighting

Luhn’s analysis – Hypothesis

[1957]
Hypothesis: Frequency of words is a measurement of word significance.
... a measurement of the power of a word to discriminate documents by their content ...

Index Term Weighting

Luhn's analysis – Observation



Index Term Weighting

Luhn’s analysis – Explanation

Resolving/Discriminating power of words
Optimal power half way between the cut-offs

Method 1: Assign weights to terms: TF only

- Assign to each term a weight *$tf_{t,d}$* - term frequency (how often term *t* occurs in document *d*)

$$score(q,d) = \sum_{t \in q} tf_{t,d}$$

- query = 'who wrote wild boys'
- doc1 = 'Duran Duran sang Wild Boys in 1984.'
- doc2 = 'Wild boys don't remain forever wild.'
- doc3 = 'Who brought wild flowers?'
- doc4 = 'It was John Krakauer who wrote *In to the wild.'*

query = {boys: 1, who: 1, wild: 1, wrote: 1}

doc1 = {1984: 1, boys: 1, duran: 2, in: 1, sang: 1, wild: 1}

doc2 = {boys: 1, don't: 1, forever: 1, remain: 1, wild: 2} …

score(q, doc1) = 1 + 1 = 2   *score(q, doc2) = 1 + 2 = 3*

score(q,doc3) = 1 + 1 = 2   *score(q, doc4) = 1 + 1 + 1 = 3*

Term Frequency (TF)

- The term frequency $\mathrm{tf}_{t,d}$ 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. But how?
- Raw term frequency is 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.

NB: frequency = count in IR

Log-frequency weighting

- The log frequency weight of term t in d is

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

- 0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.
- Score for a document-query pair: sum over terms *t* in both *q* and *d*:
- 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.

Why is Method 1 (TF-only) not good?

- All terms have equal importance.
- Bigger documents have more terms, thus the score is larger.
- It ignores term order.

Postulate: If a word appears in every document, probably it is not that important (it has no discriminatory power).

Method 2: New weights: IDF

Document frequency

- Rare terms are more informative than frequent terms
  - Recall stop words
- 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 to the query *arachnocentric*
- → We want a high weight for rare terms like *arachnocentric*.

Method 2: New weights: IDF

Document frequency (cont.)

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the collection (e.g., *high, increase, line*)
- A document containing such a term is more likely to be relevant than a document that doesn't
- But it's not a sure indicator of relevance.
- → For frequent terms, we want high positive weights for words like *high, increase, and line*
- But lower weights than for rare terms.
- We will use document frequency (df) to capture this.

Method 2: New weights: IDF

IDF weight

- df_t is the document frequency of *t* : the number of documents that contain *t*
  - df_t is an inverse measure of the informativeness of *t*
  - $\mathrm{df}_t \leq N$
- We define the idf (inverse document frequency) of *t* by

$$\mathrm{idf}_t = \log_{10}\,(N/\mathrm{df}_t)$$

- We use log (N/ df_t) instead of N/ df_t to "dampen" the effect of idf.
- Will turn out the base of the log is immaterial.

Effect of IDF on ranking

- Does idf have an effect on ranking for one-term queries, like
  - iPhone?
  - Why not?
- idf has no effect on ranking one term queries
  - idf affects the ranking of documents for queries with at least two terms
  - For the query capricious person, idf weighting makes occurrences of capricious count for much more in the final document ranking than occurrences of person.

TF-IDF

- df_t - document frequency for term t
- idf_t - inverse document frequency for term t

$$idf_t = \log \frac{N}{df_t}$$

N - total number of documents

- tf-idf_{td} - a combined weight for term t in document d

$$tf\text{-}idf_{t,d} = tf_{t,d} * idf_t$$
$$score(q,d) = \sum_{t \in q} tf\ idf_{t,d}$$

- Increases with the number of occurrences *within* a doc
- Increases with the rarity of the term *across* the whole corpus
- tf-idf became popular as a weighting scheme in the vector-space model (VSM), SMART retrieval system, Salton [1971]

Example: idf values

terms	df	idf
1984	1	0.602
boys	2	0.301
brought	1	0.602
don’t	1	0.602
duran	1	0.602
flowers	1	0.602
forever	1	0.602
in	2	0.301
it	1	0.602
john	1	0.602

terms	df	idf
krakauer	1	0.602
remain	1	0.602
sang	1	0.602
the	1	0.602
to	1	0.602
was	1	0.602
who	2	0.301
wild	4	0.0
wrote	1	0.602

D1	duran duran sang wild boys in 1984
D2	wild boys don't remain forever wild
D3	who brought wild flowers
D4	it was john krakauer who wrote in to the wild

Example: calculating scores (1)

query = ‘who wrote wild boys’

documents	S: tf-idf	S: tf
duran duran sang wild boys in 1984		
wild boys don't remain forever wild		
who brought wild flowers		
it was john krakauer who wrote in to the wild		

terms	df	idf
who	2	0.301
wild	4	0.0
wrote	1	0.602
boys	2	0.301

Example: calculating scores (1)

query = ‘who wrote wild boys’

documents	S: tf-idf	S: tf
duran duran sang wild boys in 1984	0.301	2
wild boys don't remain forever wild	0.301	3
who brought wild flowers	0.301	2
it was john krakauer who wrote in to the wild	**0.903**	**3**

terms	df	idf
who	2	0.301
wild	4	0.0
wrote	1	0.602
boys	2	0.301

Example: calculating scores (1)

query = ‘who wrote wild boys’

documents	S: tf-idf	S: tf
duran duran sang wild boys in 1984		
wild boys don't remain forever wild		
who brought ~~wild~~ flowers		
it was john krakauer who wrote in to the wild		

terms	df	idf
who	2	0.301
wild	**3**	**0.125**
wrote	1	0.602
boys	2	0.301

Example: calculating scores (1)

query = ‘who wrote wild boys’

documents	**S: tf-idf**	**S: tf**
duran duran sang wild boys in 1984	0.426	2
wild boys don't remain forever wild	0.551	3
who brought ~~wild~~ flowers	0.301	1
it was john krakauer who wrote in to the wild	**1.028**	**3**

terms	df	idf
who	2	0.301
wild	**3**	**0.125**
wrote	1	0.602
boys	2	0.301

Example: calculating scores (2)

query = ‘who wrote wild boys’

documents	S: tf-idf	S: tf
duran duran who sang wild boys in 1984		
wild boys don't remain forever wild		
who brought ~~wild~~ flowers		
it was john krakauer who wrote in to the wild		

terms	df	idf
who	**3**	**0.125**
wild	**3**	**0.125**
wrote	1	0.602
boys	2	0.301

documents	S: tf-idf	S: tf
duran duran ~~sang~~ wrote wild boys in 1984		
wild boys don't remain forever wild		
who brought ~~wild~~ flowers		
it was john krakauer who wrote in to the wild		

terms	df	idf
who	2	0.301
wild	**3**	**0.125**
wrote	**2**	**0.301**
boys	2	0.301

Example: calculating scores (2)

query = 'who wrote wild boys'

documents	S: tf-idf	S: tf
duran duran who sang wild boys in 1984	0.551	3
wild boys don't remain forever wild	0.551	3
who brought ~~wild~~ flowers	0.125	1
it was john krakauer who wrote in to the wild	**0.852**	**3**

terms	df	idf
who	**3**	**0.125**
wild	**3**	**0.125**
wrote	1	0.602
boys	2	0.301

documents	S: tf-idf	S: tf
duran duran ~~sang~~ wrote wild boys in 1984	**0.727**	**3**
wild boys don't remain forever wild	0.551	3
who brought ~~wild~~ flowers	0.301	1
it was john krakauer who wrote in to the wild	**0.727**	**3**

terms	df	idf
who	2	0.301
wild	**3**	**0.125**
wrote	**2**	**0.301**
boys	2	0.301

Binary → count → weight matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

Each document is now represented by a real-valued vector of tf-idf weights $\in R^{|V|}$

Documents as vectors: Vector Space Model

- So we have a |V|-dimensional 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 a web search engine
- These are very sparse vectors - most entries are zero.

Queries as vectors

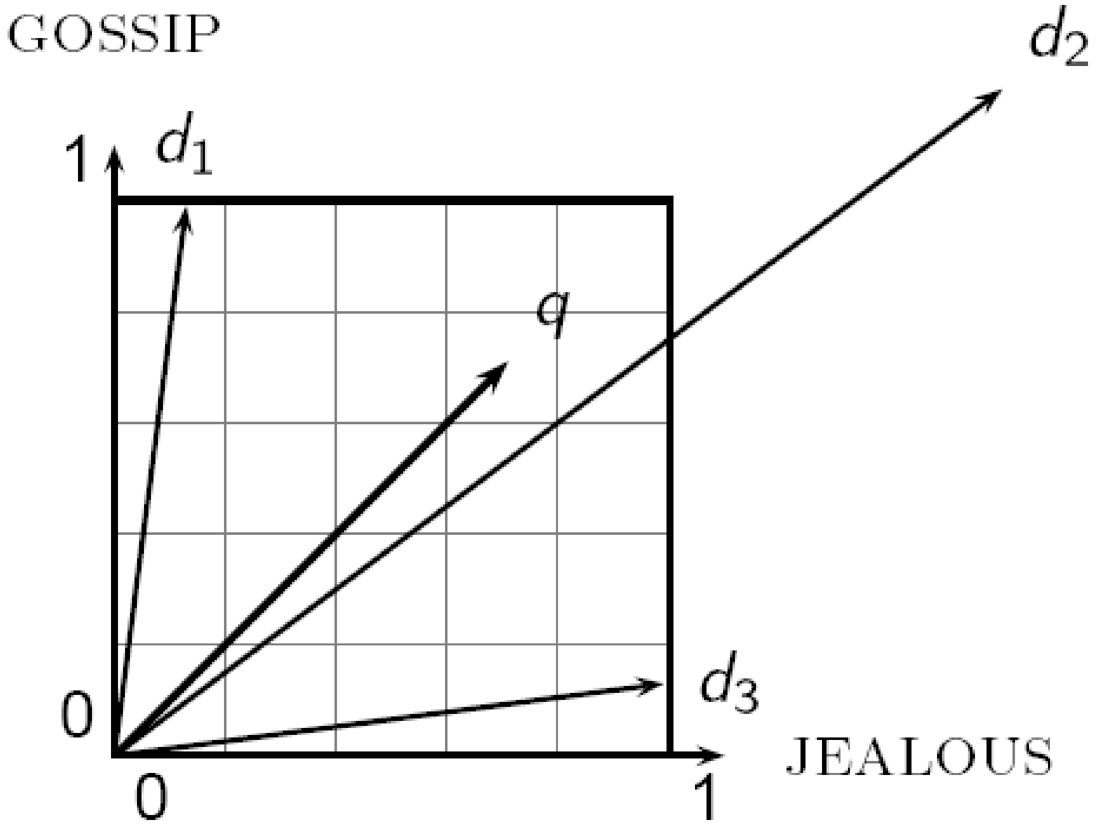
- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance
- Recall: We do this because we want to get away from the you're-either-in-or-out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

Formalizing vector space proximity

- First cut: distance between two points
  - (= distance between 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 between $\vec{q}$ and $\vec{d_2}$ is large even though the distribution of terms in the query $\vec{q}$ and the distribution of terms in the document $\vec{d_2}$ are very similar.



Use angle instead of distance

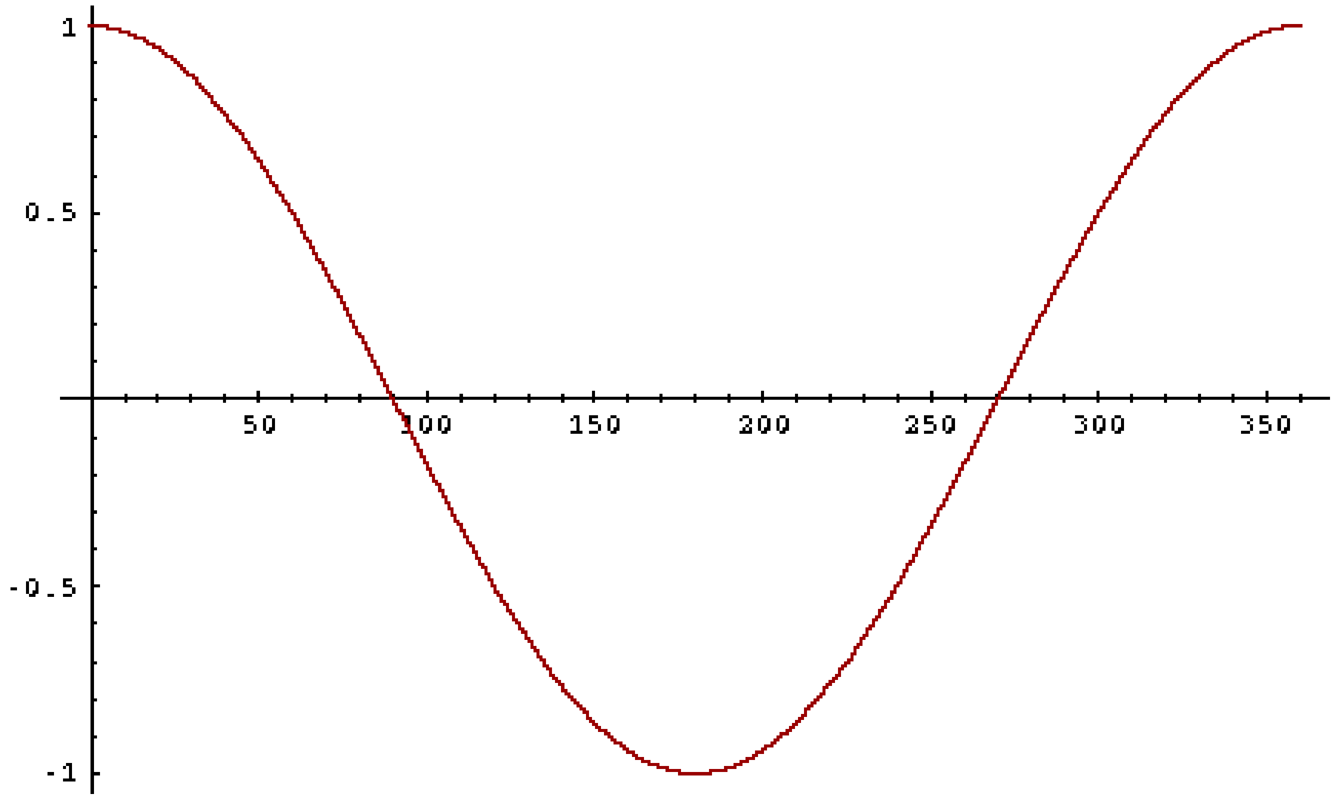
- Thought experiment: take a document *d* and append it to itself. Call this document d_2.
- "Semantically" *d* and d_2 have the same content
- The Euclidean distance between the two documents can be quite large
- The angle between the two documents is 0, corresponding to maximal similarity.

- Key idea: Rank documents according to angle with query.

From angles to cosines

- The following two notions are equivalent.
  - Rank documents in decreasing order of the angle between query and document
  - Rank documents in increasing order of cosine(query,document)
- Cosine is a monotonically decreasing function for the interval [0^o, 180^o]
- All the vectors built by TFIDF are in the [0^o,90^o] range in all dimensions. **Why?**

From angles to cosines



- But how - *and why* - should we be computing cosines?

Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

- Dividing a vector by its L_2 norm makes it a unit (length) vector (on surface of unit hypersphere)
- Effect on the two documents *d* and d_2 (*d* appended to itself) from earlier slide: they have identical vectors after length-normalization.
  - Long and short documents now have comparable weights

cosine(query,document)

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q} \overset{\text{Dot product}}{\bullet} \vec{d}}{|\vec{q}||\vec{d}|} = \overbrace{\frac{\vec{q}}{|\vec{q}|} \bullet \frac{\vec{d}}{|\vec{d}|}}^{\text{Unit vectors}} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

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

$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

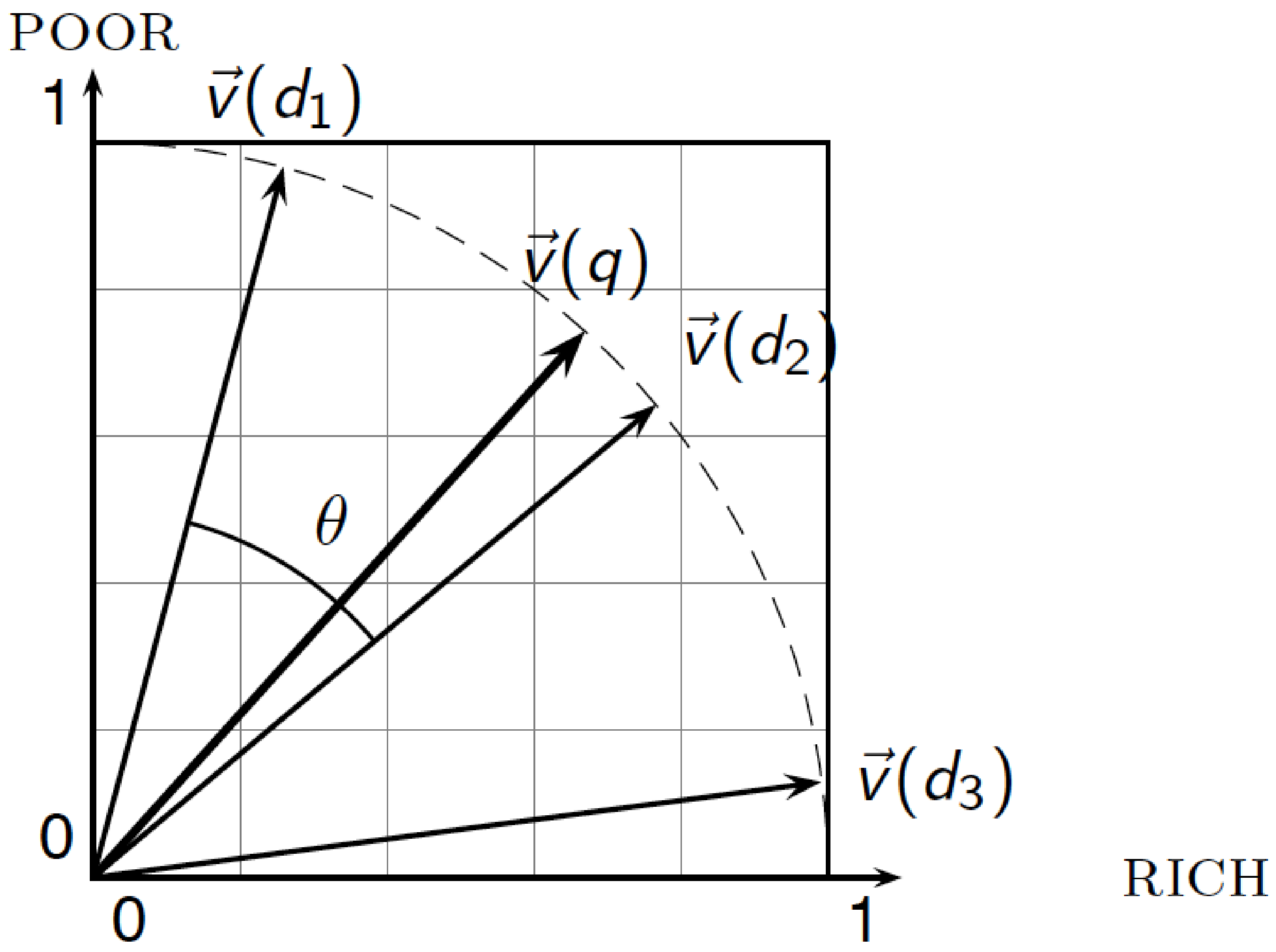
Cosine for length-normalized vectors

- For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

$$\cos(\vec{q},\vec{d}) = \vec{q} \bullet \vec{d} = \sum_{i=1}^{|V|} q_i d_i$$

for q, d length-normalized.

Cosine similarity illustrated



Cosine similarity amongst 3 documents

How similar are the novels

SaS: *Sense and Sensibility*

PaP: *Pride and Prejudice*, and

WH: *Wuthering Heights*

?

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts)

Note: To simplify this example, we only do TF weighting.

3 documents example contd.

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) ≈
$0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0$
≈ 0.94
cos(SaS,WH) ≈ 0.79
cos(PaP,WH) ≈ 0.69

Why do we have cos(SaS,PaP) > cos(SaS,WH)?

Other forms of term frequency

- Simplest form:

  $tf_{t,d} = f_{t,d}$ (raw count of a term in a document)

- Boolean "frequencies":

  $tf_{t,d} = 1$ if t occurs in d and 0 otherwise;

- Term frequency adjusted for document length :

  $tf_{t,d} = f_{t,d} \div (number\ of\ words\ in\ d)$

- Logarithmically scaled frequency:

$$tf_{t,d} = \begin{cases} 1 + \log(f_{t,d}), when\ f_{t,d} \neq 0 \\ 0, when\ f_{t,d} = 0 \end{cases}$$

- Augmented frequency:

$$tf_{t,d} = 0.5 + 0.5 \times \frac{f_{t,d}}{\max\{f_{t',d}: t' \in d\}}$$

Other Term Weighting Techniques

- **Dictionary Based Methods**
  - Look up each query term in a dictionary and use dictionary data to determine the term weight. For example:
    - Weight adverbs and adjectives lower than nouns
    - Weight nouns over verbs
    - Weight articles (such as “the”, “a”, “an”, etc.) low
    - Weight terms with many dictionary meanings (i.e. highly ambiguous terms) lower than terms with fewer dictionary meanings

Other Term Weighting Techniques

- **Stop Words**

  Much smaller than a full dictionary, “stop words” are frequently occurring words that may simply be removed from relevancy scoring completely

- **Term length**

  Words that are longer (have more characters), such as “computability” are considered to be more valuable than small words such as “is”, “go”, etc.

Other Term Weighting Techniques

- **Term Cohesiveness**

  By analysing the context around each occurrence of a term, one can compute its "cohesiveness", or in other words, how often the term is used within a similar context.

  - For example, terms that are used with the same set of neighbouring terms are "more cohesive" and may therefore be considered to be more useful (less ambiguous) search terms

- **Position In Query**

  Terms at the start of the query may be considered to be more important than terms at the end of the query

Other Term Weighting Techniques

- **Frequency of Occurrence in Query**

  When queries are very large, for example when cutting-and-pasting whole paragraphs into the query box or when using an entire document as the "query" for similarity searching, then the number of occurrences of a term in the query can be used to help determine the appropriate term weight
- **Capitalization**

  Some early search engines (Alta Vista was one of them) would add additional weight to capitalized terms entered by the user. It was assumed that these terms represented proper nouns that would be more important as search terms.

Summary

- Boolean Retrieval vs Ranked Retrieval (TF-IDF)
- Index Term Weighting Theory (Zipf, Luhn)
- VSM:
  - Represent the query as a weighted tf-idf vector
  - Represent each document as a weighted tf-idf vector
  - Compute the cosine similarity score for the query vector and each document vector
  - Rank documents with respect to the query by score
  - Return the top *K* (e.g., *K* = 10) to the user