

Introduction to Information Retrieval

<http://informationretrieval.org>

IIR 6: Scoring, Term Weighting, The Vector Space Model

Hinrich Schütze & Wiltrud Kessler

Institute for Natural Language Processing, University of Stuttgart

2012-11-13

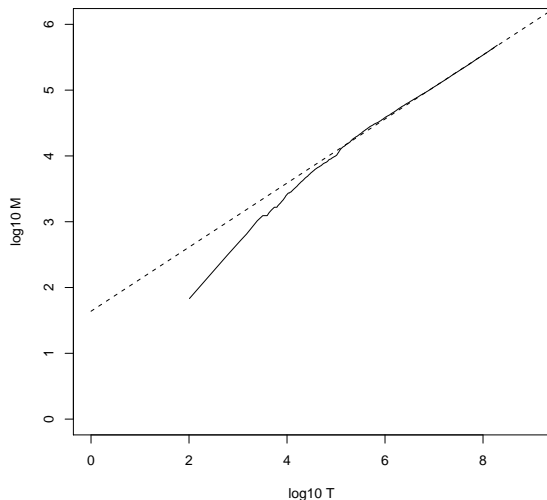
Overview

- 1 Recap
- 2 Why ranked retrieval?
- 3 Term frequency
- 4 tf-idf weighting
- 5 The vector space model

Outline

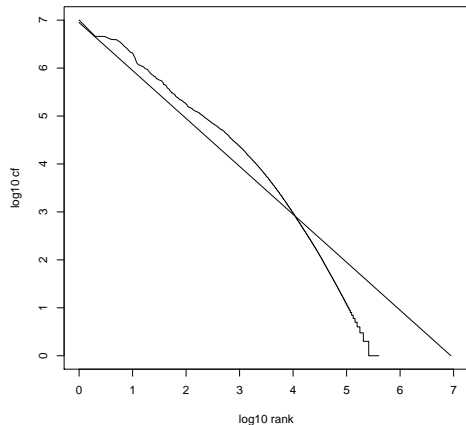
- 1 Recap
- 2 Why ranked retrieval?
- 3 Term frequency
- 4 tf-idf weighting
- 5 The vector space model

Heaps' law for Reuters



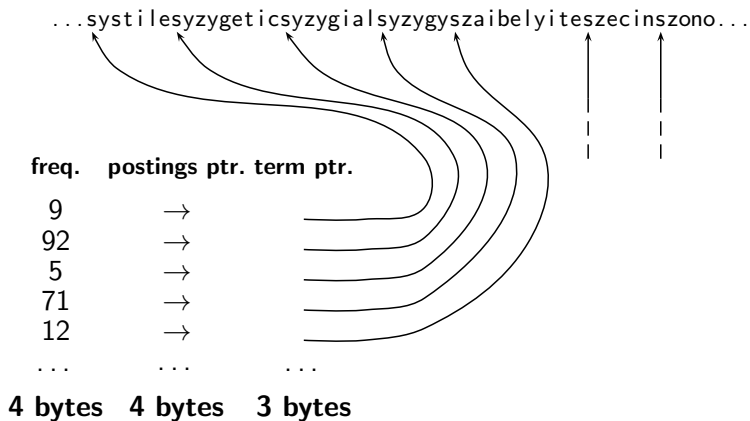
Vocabulary size M as a function of collection size T (number of tokens) for Reuters-RCV1. For these data, the dashed line $\log_{10} M = 0.49 * \log_{10} T + 1.64$ is the best least squares fit. Thus, $M = 10^{1.64} T^{0.49}$ and $k = 10^{1.64} \approx 44$ and $b = 0.49$.

Zipf's law for Reuters



Fit is not great. What is important is the key insight: Few frequent terms, many rare terms.

Compressing the dictionary



Compressing the postings

| | encoding | postings list | | | | | |
|----------------|----------|---------------|--------|--------|--------|--------|-----|
| THE | docIDs | ... | 283042 | 283043 | 283044 | 283045 | ... |
| | gaps | | | 1 | 1 | 1 | ... |
| COMPUTER | docIDs | ... | 283047 | 283154 | 283159 | 283202 | ... |
| | gaps | | | 107 | 5 | 43 | ... |
| ARACHNOCENTRIC | docIDs | 252000 | 500100 | | | | |
| | gaps | 252000 | 248100 | | | | |

- Key idea: Store gaps instead of docIDs
- Thus: We can encode small gaps with fewer than 20 bits.
- In order to implement this, we need to devise some form of [variable length encoding](#).
- Variable byte (VB) code.
- Gamma code

Take-away today

- **Ranking** search results: why it is important (as opposed to just presenting a set of unordered Boolean results)
- **Term frequency**: This is a key ingredient for ranking.
- **Tf-idf ranking**: best known traditional ranking scheme
- **Vector space model**: One of the most important formal models for information retrieval (along with Boolean and probabilistic models)

Outline

- 1 Recap
- 2 Why ranked retrieval?
- 3 Term frequency
- 4 tf-idf weighting
- 5 The vector space model

Ranked retrieval

- Thus far, our queries have been **Boolean**.
 - Documents either match or don't.
- **Good for expert users** with precise understanding of their needs and of the collection.
- Also **good for applications**: Applications can easily consume 1000s of results.
- **Not good for the majority of users**
- Most users are not capable 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 (boolean conjunction): [standard user dlink 650]
 - → 200,000 hits – **feast**
- Query 2 (boolean conjunction): [standard user dlink 650 no card found]
 - → 0 hits – **famine**
- In Boolean retrieval, it takes a lot of skill to come up with a query that produces a manageable number of hits.

Feast or famine: No problem in ranked retrieval

- With ranking, large result sets are not an issue.
- Just show the top 10 results
- Doesn't overwhelm the user
- Premise: The ranking algorithm works: More relevant results are ranked higher than less relevant results.

Scoring as the basis of ranked retrieval

- We wish to rank documents that are more relevant higher than documents that are less relevant.
- How can we accomplish such a ranking of the documents in the collection with respect to a query?
- Assign a score to each query-document pair, say in $[0, 1]$.
- This score measures how well document and query “match”.

Query-document matching scores

- How do we compute the score of 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.
- We will look at a number of alternatives for doing this.

Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets
- Let A and B be two sets
- Jaccard coefficient:

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

($A \neq \emptyset$ or $B \neq \emptyset$)

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

Jaccard coefficient: Example

- What is the query-document match score that the Jaccard coefficient computes for:
 - Query: “ides of March”
 - Document “Caesar died in March”
 - $JACCARD(q, d) = 1/6$

What's wrong with Jaccard?

- It doesn't consider term frequency (how many occurrences a term has).
- Rare terms are more informative than frequent terms. Jaccard does not consider this information.
- We need a more sophisticated way of normalizing for the length of a document.
- Later in this lecture, we'll use $|A \cap B| / \sqrt{|A \cup B|}$ (cosine) ...
- ... instead of $|A \cap B| / |A \cup B|$ (Jaccard) for length normalization.

Outline

- 1 Recap
- 2 Why ranked retrieval?
- 3 Term frequency
- 4 tf-idf weighting
- 5 The vector space model

Binary incidence matrix

| | 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 | |
| ... | | | | | | | |

Each document is represented as a **binary vector** $\in \{0, 1\}^{|V|}$.

Count matrix

| | 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 | |
| ... | | | | | | | |

Each document is now represented as a **count vector** $\in \mathbb{N}^{|V|}$.

Bag of words model

- 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 course.
- For now: bag of words model

Term frequency tf

- The term frequency $tf_{t,d}$ 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 because:
- A document with $tf = 10$ occurrences of the term is more relevant than a document with $tf = 1$ occurrence of the term.
- But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Instead of raw frequency: Log frequency weighting

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

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

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

Exercise

- Compute the Jaccard matching score and the tf matching score for the following query-document pairs.
- q: [information on cars] d: “all you’ve ever wanted to know about cars”
- q: [information on cars] d: “information on trucks, information on planes, information on trains”
- q: [red cars and red trucks] d: “cops stop red cars more often”

Outline

- 1 Recap
- 2 Why ranked retrieval?
- 3 Term frequency
- 4 tf-idf weighting**
- 5 The vector space model

Frequency in document vs. frequency in collection

- In addition, to term frequency (the frequency of the term in the document) ...
- ... we also want to use the frequency of the term **in the collection** for weighting and ranking.

Desired weight for rare terms

- Rare terms are more informative than frequent terms.
- 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.
- → We want **high weights for rare terms** like ARACHNOCENTRIC.

Desired weight for frequent terms

- Frequent terms are less informative than rare terms.
- Consider a term in the query that is **frequent** in the collection (e.g., GOOD, INCREASE, LINE).
- A document containing this term is more likely to be relevant than a document that doesn't ...
- ...but words like GOOD, INCREASE and LINE are not sure indicators of relevance.
- → **For frequent terms** like GOOD, INCREASE, and LINE, we want positive weights ...
- ...but **lower weights** than for rare terms.

Document frequency

- We want **high weights for rare terms** like ARACHNOCENTRIC.
- We want **low (positive) weights for frequent words** like GOOD, INCREASE, and LINE.
- We will use **document frequency** to factor this into computing the matching score.
- The document frequency is **the number of documents in the collection that the term occurs in.**

idf weight

- df_t is the document frequency, the number of documents that t occurs in.
- df_t is an inverse measure of the **informativeness** of term t .
- We define the **idf weight** of term t as follows:

$$\text{idf}_t = \log_{10} \frac{N}{df_t}$$

(N is the number of documents in the collection.)

- idf_t is a measure of the **informativeness** of the term.
- $[\log N/df_t]$ instead of $[N/df_t]$ to “dampen” the effect of idf
- Note that we use the log transformation for both term frequency and document frequency.

Examples for idf

Compute idf_t using the formula: $\text{idf}_t = \log_{10} \frac{1,000,000}{\text{df}_t}$

| term | df_t | idf_t |
|-----------|---------------|----------------|
| calpurnia | 1 | 6 |
| animal | 100 | 4 |
| sunday | 1000 | 3 |
| fly | 10,000 | 2 |
| under | 100,000 | 1 |
| the | 1,000,000 | 0 |

Effect of idf on ranking

- idf affects the ranking of documents for **queries with 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.
- idf has **little effect** on ranking for **one-term queries**.

Collection frequency vs. Document frequency

| word | collection frequency | document frequency |
|-----------|----------------------|--------------------|
| INSURANCE | 10440 | 3997 |
| TRY | 10422 | 8760 |

- Collection frequency of t : number of tokens of t in the collection
- Document frequency of t : number of documents t occurs in
- Why these numbers?
- Which word is a better search term (and should get a higher weight)?
- This example suggests that df (and idf) is better for weighting than cf (and “icf”).

tf-idf weighting

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



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

- **tf-weight**
- **idf-weight**
- Best known weighting scheme in information retrieval
- Note: the “-” in tf-idf is a hyphen, not a minus sign!
- Alternative names: tf.idf, tf x idf

Summary: tf-idf

- Assign a tf-idf weight for each term t in each document d :
$$w_{t,d} = (1 + \log \text{tf}_{t,d}) \cdot \log \frac{N}{\text{df}_t}$$
- The tf-idf weight ...
 - ... increases with the number of occurrences within a document. (term frequency)
 - ... increases with the rarity of the term in the collection. (inverse document frequency)

Exercise: Term, collection and document frequency

| Quantity | Symbol | Definition |
|----------------------|------------|--|
| term frequency | $tf_{t,d}$ | number of occurrences of t in d |
| document frequency | df_t | number of documents in the collection that t occurs in |
| collection frequency | cf_t | total number of occurrences of t in the collection |

- Relationship between df and cf ?
- Relationship between tf and cf ?
- Relationship between tf and df ?

Outline

- 1 Recap
- 2 Why ranked retrieval?
- 3 Term frequency
- 4 tf-idf weighting
- 5 The vector space model

Binary incidence matrix

| | 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 | |
| ... | | | | | | | |

Each document is represented as a **binary vector** $\in \{0, 1\}^{|V|}$.

Count matrix

| | 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 | |
| ... | | | | | | | |

Each document is now represented as a **count vector** $\in \mathbb{N}^{|V|}$.

Binary \rightarrow count \rightarrow weight matrix

| | 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 | |
| ... | | | | | | | |

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

Documents as vectors

- Each document is now represented as a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.
- 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 engines
- Each vector is very sparse - most entries are zero.

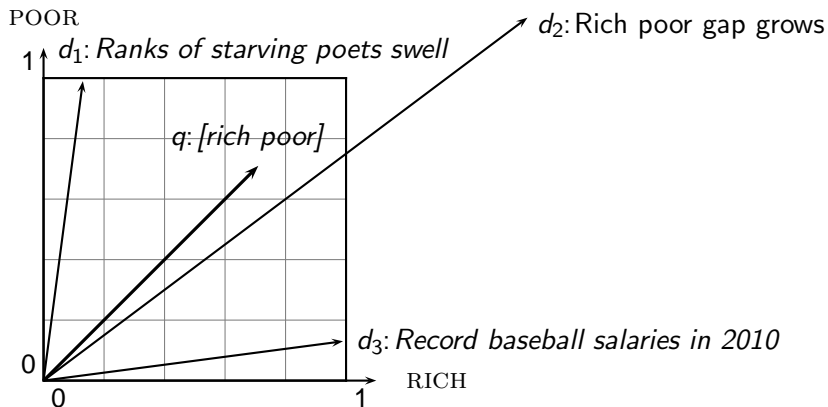
Queries as vectors

- Key idea 1: do the same for queries: represent them as vectors in the high-dimensional space
- Key idea 2: Rank documents according to their proximity to the query
- proximity = similarity
- proximity \approx negative distance
- Recall: We're doing this because we want to get away from the you're-either-in-or-out, feast-or-famine Boolean model.
- Instead: rank relevant documents higher than nonrelevant documents

How do we formalize vector space similarity?

- First cut: (negative) 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 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.

Questions about basic vector space setup?

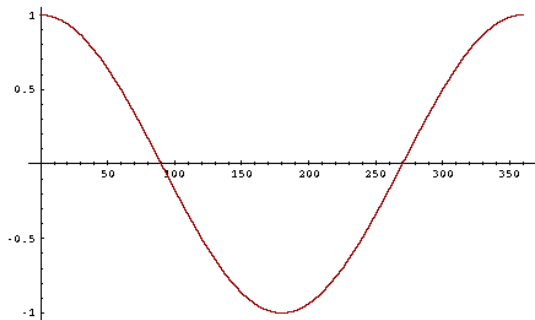
Use angle instead of distance

- Rank documents according to angle with query
- Thought experiment: take a document d and append it to itself. Call this document d' . d' is twice as long as d .
- “Semantically” d and d' have the same content.
- The angle between the two documents is 0, corresponding to maximal similarity . . .
- . . . even though the Euclidean distance between the two documents can be quite large.

From angles to cosines

- The following two notions are equivalent.
 - Rank documents according to the **angle** between query and document in decreasing order
 - Rank documents according to **cosine**(query,document) in increasing order
- Cosine is a monotonically decreasing function of the angle for the interval $[0^\circ, 180^\circ]$

Cosine



Length normalization

- How do we compute the cosine?
- A vector can be (length-) normalized by dividing each of its components by its length – here we use the L_2 norm:

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

- This maps vectors onto the unit sphere ...
- ... since after normalization: $\|x\|_2 = \sqrt{\sum_i x_i^2} = 1.0$
- As a result, longer documents and shorter documents have weights of the same order of magnitude.
- Effect on the two documents d and d' (d appended to itself) from earlier slide: they have **identical vectors** after length-normalization.

Cosine similarity between query and document

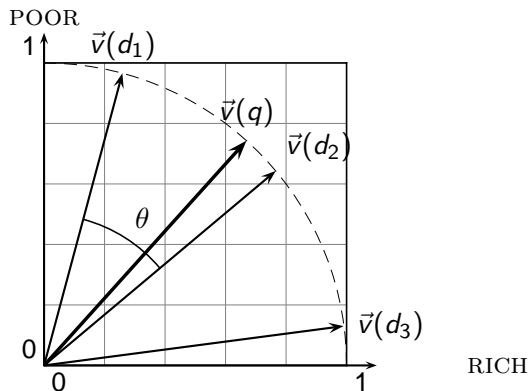
$$\cos(\vec{q}, \vec{d}) = \text{SIM}(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}| |\vec{d}|} = \frac{\sum_{i=1}^{|\mathcal{V}|} q_i d_i}{\sqrt{\sum_{i=1}^{|\mathcal{V}|} q_i^2} \sqrt{\sum_{i=1}^{|\mathcal{V}|} d_i^2}}$$

- 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} .

Cosine for normalized vectors

- For normalized vectors, the cosine is equivalent to the dot product or scalar product.
- $\cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d} = \sum_i q_i \cdot d_i$
 - (if \vec{q} and \vec{d} are length-normalized).

Cosine similarity illustrated



Cosine: Example

How similar are these novels?

SaS: Sense and Sensibility

PaP: Pride and Prejudice

WH: Wuthering Heights

term frequencies (counts)

| term | SaS | PaP | WH |
|-----------|-----|-----|----|
| AFFECTION | 115 | 58 | 20 |
| JEALOUS | 10 | 7 | 11 |
| GOSSIP | 2 | 0 | 6 |
| WUTHERING | 0 | 0 | 38 |

Cosine: Example

term frequencies (counts)

| term | SaS | PaP | WH |
|-----------|-----|-----|----|
| AFFECTION | 115 | 58 | 20 |
| JEALOUS | 10 | 7 | 11 |
| GOSSIP | 2 | 0 | 6 |
| WUTHERING | 0 | 0 | 38 |

log frequency weighting

| term | SaS | PaP | WH |
|-----------|------|------|------|
| AFFECTION | 3.06 | 2.76 | 2.30 |
| JEALOUS | 2.0 | 1.85 | 2.04 |
| GOSSIP | 1.30 | 0 | 1.78 |
| WUTHERING | 0 | 0 | 2.58 |

(To simplify this example, we don't do idf weighting.)

Cosine: Example

log frequency weighting

| term | SaS | PaP | WH |
|-----------|------|------|------|
| AFFECTION | 3.06 | 2.76 | 2.30 |
| JEALOUS | 2.0 | 1.85 | 2.04 |
| GOSSIP | 1.30 | 0 | 1.78 |
| WUTHERING | 0 | 0 | 2.58 |

log frequency weighting
& cosine normalization

| term | SaS | PaP | WH |
|-----------|-------|-------|-------|
| AFFECTION | 0.789 | 0.832 | 0.524 |
| JEALOUS | 0.515 | 0.555 | 0.465 |
| GOSSIP | 0.335 | 0.0 | 0.405 |
| WUTHERING | 0.0 | 0.0 | 0.588 |

- $\cos(\text{SaS}, \text{PaP}) \approx 0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0 \approx 0.94$.
- $\cos(\text{SaS}, \text{WH}) \approx 0.79$
- $\cos(\text{PaP}, \text{WH}) \approx 0.69$
- Why do we have $\cos(\text{SaS}, \text{PaP}) > \cos(\text{SaS}, \text{WH})$?

Computing the cosine score

COSINESCORE(q)

```
1  float Scores[N] = 0
2  float 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  $Scores[d] + = w_{t,d} \times w_{t,q}$ 
7  Read the array  $Length$ 
8  for each  $d$ 
9  do  $Scores[d] = Scores[d] / Length[d]$ 
10 return Top  $K$  components of  $Scores[]$ 
```

Components of tf-idf weighting

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

Best known combination of weighting options

Default: no weighting

tf-idf example

- We often use **different weightings** for queries and documents.
- Notation: ddd.qqq
- Example: Inc.ltn
- document: logarithmic tf, no df weighting, cosine normalization
- query: logarithmic tf, idf, no normalization
- **Isn't it bad to not idf-weight the document?**
- Example query: “best car insurance”
- Example document: “car insurance auto insurance”

tf-idf example: Inc.ltn

Query: "best car insurance". Document: "car insurance auto insurance".

| word | query | | | | | document | | | | product |
|-----------|--------|---------|-------|-----|--------|----------|---------|--------|---------|---------|
| | tf-raw | tf-wght | df | idf | weight | tf-raw | tf-wght | weight | n'lized | |
| auto | 0 | 0 | 5000 | 2.3 | 0 | 1 | 1 | 1 | 0.52 | 0 |
| best | 1 | 1 | 50000 | 1.3 | 1.3 | 0 | 0 | 0 | 0 | 0 |
| car | 1 | 1 | 10000 | 2.0 | 2.0 | 1 | 1 | 1 | 0.52 | 1.04 |
| insurance | 1 | 1 | 1000 | 3.0 | 3.0 | 2 | 1.3 | 1.3 | 0.68 | 2.04 |

Key to columns: **tf-raw**: raw (unweighted) term frequency, **tf-wght**: logarithmically weighted term frequency, **df**: document frequency, **idf**: inverse document frequency, **weight**: the final weight of the term in the query or document, **n'lized**: document weights after cosine normalization, **product**: the product of final query weight and final document weight

$$\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92$$

$$1/1.92 \approx 0.52$$

$$1.3/1.92 \approx 0.68$$

Final similarity score between query and document: $\sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08$

Questions?

Summary: Ranked retrieval in the vector space model

- 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., $K = 10$) to the user

Take-away today

- **Ranking** search results: why it is important (as opposed to just presenting a set of unordered Boolean results)
- **Term frequency**: This is a key ingredient for ranking.
- **Tf-idf ranking**: best known traditional ranking scheme
- **Vector space model**: One of the most important formal models for information retrieval (along with Boolean and probabilistic models)

Resources

- Chapter 6 of IIR
- Resources at <http://ifnlp.org/ir>
 - Vector space for dummies
 - Exploring the similarity space (Moffat and Zobel, 2005)
 - Okapi BM25 (a state-of-the-art weighting method, 11.4.3 of IIR)