Scoring, term weighting, the vector space model [1]

December, 2009

[1] Vorlage: Folien von M. Schütze

Overview

1 Term frequency

2 tf-idf weighting

3 The vector space

Outline

1 Term frequency

2 tf-idf weighting

3 The vector space

Ranked retrieval

- Thus far, our queries have all been Boolean.

Ranked retrieval

- Thus far, our queries have all been Boolean.
  - Documents either match or don't.

Ranked retrieval

- 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.

Ranked retrieval

- 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.

Ranked retrieval

- 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.

Ranked retrieval

- 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 are not capable of writing Boolean queries (or they are, but they think it's too much work).

Ranked retrieval

- 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 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.

Ranked retrieval

- 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 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.

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

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" $\rightarrow$ 200,000 hits
- Query 2: "standard user dlink 650 no card found": 0 hits

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" $\rightarrow$ 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.

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" $\rightarrow$ 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.
- With a ranked list of documents it does not matter how large the retrieved set is.

Scoring as the basis of ranked retrieval

- We wish to return in order the documents most likely to be useful to the searcher.

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?

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

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.

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.

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.

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

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
- We will look at a number of alternatives for doing this.

Take 1: Jaccard coefficient

- Recall from IIR 3: A commonly used measure of overlap of two sets

Take 1: Jaccard coefficient

- Recall from IIR 3: A commonly used measure of overlap of two sets
- Let A and B be two sets

Take 1: Jaccard coefficient

- Recall from IIR 3: A commonly used measure of overlap of two sets
- Let A and B be two sets
- Jaccard coefficient:

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

Take 1: Jaccard coefficient

- Recall from IIR 3: A commonly used measure of overlap of two sets
- Let A and B be two sets
- Jaccard coefficient:

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

- $\text{jaccard}(A, A) = 1$

Take 1: Jaccard coefficient

- Recall from IIR 3: A commonly used measure of overlap of two sets
- Let A and B be two sets
- Jaccard coefficient:

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

- $\text{jaccard}(A, A) = 1$
- $\text{jaccard}(A, B) = 0$ if $A \cap B = 0$

Take 1: Jaccard coefficient

- Recall from IIR 3: A commonly used measure of overlap of two sets
- Let A and B be two sets
- Jaccard coefficient:

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

- $\text{jaccard}(A, A) = 1$
- $\text{jaccard}(A, B) = 0$ if $A \cap B = 0$
- A and B don't have to be the same size.

Take 1: Jaccard coefficient

- Recall from IIR 3: A commonly used measure of overlap of two sets
- Let A and B be two sets
- Jaccard coefficient:

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

- $\text{jaccard}(A, A) = 1$
- $\text{jaccard}(A, B) = 0$ if $A \cap B = 0$
- 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:

Jaccard coefficient: Example

- What is the query-document match score that the Jaccard coefficient computes for:
  - Query: "ides of March"

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 coefficient: Example

- What is the query-document match score that the Jaccard coefficient computes for:
  - Query: “ides of March”
  - Document “Caesar died in March”
  - ?

What's wrong with Jaccard?

- It doesn't consider term frequency (how many occurrences a term has).

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 doesn't consider this information.

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 doesn't consider this information.
- We need a more sophisticated way of normalizing for length.

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 doesn't consider this information.
- We need a more sophisticated way of normalizing for length.
- Later in this lecture, we'll use $|A \cap B|/\sqrt{|A \cup B|}$ (cosine) ...

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 doesn't consider this information.
- We need a more sophisticated way of normalizing for length.
- 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.

Recall: 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 by a binary vector $\in \{0, 1\}^{|V|}$.

Recall: 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 by a binary vector $\in \{0,1\}^{|V|}$.

From now on, we will use the frequencies of terms

	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 represented by a count vector $\in \mathbb{N}^{|V|}$.

From now on, we will use the frequencies of terms

	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 represented by a count vector $\in \mathbb{N}^{|V|}$.

Bag of words model

- We do not consider the order of words in a document.

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.

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.

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.

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.

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 $\mathrm{tf}_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.

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.

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.
- But how?

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.

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 one occurrence of the term.

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 one occurrence of the term.
- But not 10 times more relevant.

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 one occurrence of the term.
- But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

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 one occurrence of the term.
- But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

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 one occurrence of the term.
- But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Log frequency weighting

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

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

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.

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.
- Score for a document-query pair: sum over terms t in both q and d:
  matching-score $= \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$

Log frequency weighting

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

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

- $0 \rightarrow 0$, $1 \rightarrow 1$, $2 \rightarrow 1.3$, $10 \rightarrow 2$, $1000 \rightarrow 4$, etc.
- 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.

Outline

1 Term frequency

2 tf-idf weighting

3 The vector space

Document frequency

- Rare terms are more informative than frequent terms.

Document frequency

- Rare terms are more informative than frequent terms.
- Consider a term in the query that is rare in the collection (e.g., arachnocentric)

Document frequency

- 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.

Document frequency

- 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 a high weight for rare terms like arachnocentric.

Document frequency

- 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 a high weight for rare terms like arachnocentric.
- Consider a term in the query that is frequent in the collection (e.g., high, increase, line)

Document frequency

- 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 a high weight for rare terms like arachnocentric.
- Consider a term in the query that is frequent in the collection (e.g., high, increase, line)
  - A document containing this term is more likely to be relevant than a document that doesn't, but it's not a sure indicator of relevance.

Document frequency

- 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 a high weight for rare terms like arachnocentric.
- Consider a term in the query that is frequent in the collection (e.g., high, increase, line)
  - A document containing this 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 positive weights for words like high, increase, and line, but lower weights than for rare terms.

Document frequency

- 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 a high weight for rare terms like arachnocentric.
- Consider a term in the query that is frequent in the collection (e.g., high, increase, line)
  - A document containing this 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 positive weights for words like high, increase, and line, but lower weights than for rare terms.
- We will use document frequency to factor this into computing the matching score.

Document frequency

- 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 a high weight for rare terms like arachnocentric.
- Consider a term in the query that is frequent in the collection (e.g., high, increase, line)
  - A document containing this 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 positive weights for words like high, increase, and line, but lower weights than for rare terms.
- 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.

idf weight

- df_t is the document frequency, the number of documents that t occurs in.
- df is an inverse measure of the informativeness of the term.

idf weight

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

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

idf weight

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

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

- idf is a measure of the informativeness of the term.

idf weight

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

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

- idf is a measure of the informativeness of the term.
- We use $\log N/\mathrm{df}_t$ instead of N/df_t to "dampen" the effect of idf.

idf weight

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

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

- idf is a measure of the informativeness of the term.
- We use $\log N/\text{df}_t$ instead of N/df_t to "dampen" the effect of idf.
- So we use the log transformation for both term frequency and document frequency.

Examples for idf

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

term	df_t	idf_t
calpurnia	1	
animal	100	
sunday	1000	
fly	10,000	
under	100,000	
the	1,000,000	

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 only if the query has at least two terms.

Effect of idf on ranking

- 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.

Effect of idf on ranking

- 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.
- idf has no effect on ranking for one-term queries.

Effect of idf on ranking

- 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.
- idf has no effect on ranking for one-term queries.
- Questions about idf?

Collection frequency vs. Document frequency

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

- The collection frequency of t is the number of tokens of t in the collection where we count multiple occurrences.

Collection frequency vs. Document frequency

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

- The collection frequency of t is the number of tokens of t in the collection where we count multiple occurrences.
- Why these numbers?

Collection frequency vs. Document frequency

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

- The collection frequency of t is the number of tokens of t in the collection where we count multiple occurrences.
- Why these numbers?
- Which word is a better search term (and should get a higher weight)?

Collection frequency vs. Document frequency

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

- The collection frequency of t is the number of tokens of t in the collection where we count multiple occurrences.
- Why these numbers?
- Which word is a better search term (and should get a higher weight)?
- This example suggests that df is better for weighting that cf.

tf-idf weighting

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

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.
- $$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.
-
$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$
- Best known weighting scheme in information retrieval

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.
- $$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$
- Best known weighting scheme in information retrieval
- Note: the "-" in tf-idf is a hyphen, not a minus sign!

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.
- $$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$
- 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: $w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$

Summary: tf-idf

- Assign a tf-idf weight for each term t in each document d: $w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$
- N: total number of documents

Summary: tf-idf

- Assign a tf-idf weight for each term t in each document d: $w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$
- N: total number of documents
- Increases with the number of occurrences within a document

Summary: tf-idf

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

Term, collection and document frequency

Quantity	Symbol	Definition
term frequency	$\mathrm{tf}_{t,d}$	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

Term, collection and document frequency

Quantity	Symbol	Definition
term frequency	$\text{tf}_{t,d}$	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?

Term, collection and document frequency

Quantity	Symbol	Definition
term frequency	$\mathrm{tf}_{t,d}$	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 tf and cf?

Outline

1 Term frequency

2 tf-idf weighting

3 The vector space

Binary → count → 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 by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.

Binary → count → 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 by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.

Documents as vectors

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

Documents as vectors

- Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.
- So we have a $|V|$-dimensional real-valued vector space.

Documents as vectors

- Each document is now represented by 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 as vectors

- Each document is now represented by 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.

Documents as vectors

- Each document is now represented by 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 a web search engine

Documents as vectors

- Each document is now represented by 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 a web search engine
- This is a 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

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

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

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
- proximity $\approx$ negative distance

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
- proximity $\approx$ negative distance
- Recall: We're doing this because we want to get away from the you're-either-in-or-out Boolean model.

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
- proximity $\approx$ negative distance
- 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

How do we formalize vector space similarity?

- First cut: distance between two points

How do we formalize vector space similarity?

- First cut: distance between two points
- (= distance between the end points of the two vectors)

How do we formalize vector space similarity?

- First cut: distance between two points
- (= distance between the end points of the two vectors)
- Euclidean distance?

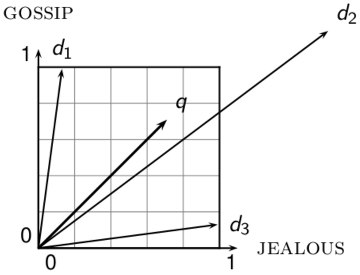
How do we formalize vector space similarity?

- First cut: distance between two points
- (= distance between the end points of the two vectors)
- Euclidean distance?
- Euclidean distance is a bad idea . . .

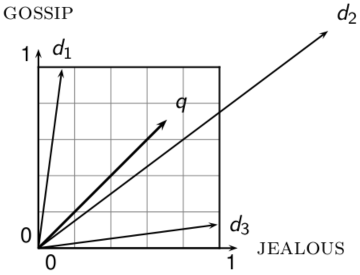
How do we formalize vector space similarity?

- 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

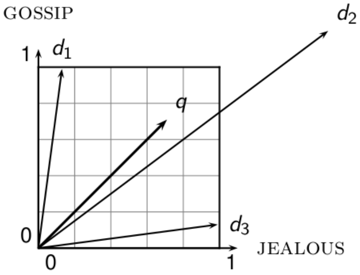


Why distance is a bad idea



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.

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

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'.

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'.
- "Semantically" d and d' have the same content.

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'.
- "Semantically" d and d' have the same content.
- The angle between the two documents is 0, corresponding to maximal similarity.

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'.
- "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.

From angles to cosines

- The following two notions are equivalent.

From angles to cosines

- The following two notions are equivalent.
  - Rank documents according to the angle between query and document in decreasing order

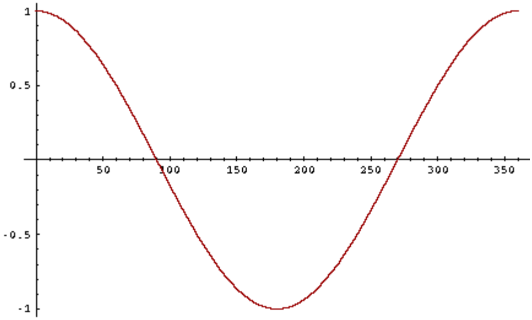
From angles to cosines

- 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

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



What about angles $> 180°$?

Length normalization

- How do we compute the 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: $||x||_2 = \sqrt{\sum_i x_i^2}$

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 . . .

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: $||x||_2 = \sqrt{\sum_i x_i^2} = 1.0$

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.

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

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

- q_i is the tf-idf weight of term i in the query.

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.

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 $\vec{d}$.

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 $\vec{d}$.

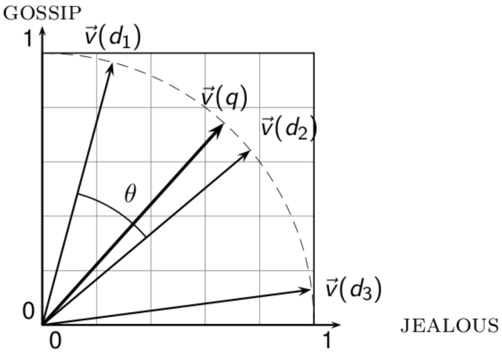
Cosine for normalized vectors

- For normalized vectors, the cosine is equivalent to the dot product or scalar product.

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 the novels? SaS: Sense and Sensibility, PaP: Pride and Prejudice, and WH: Wuthering Heights?

Cosine: Example

How similar are the novels? SaS: Sense and Sensibility, PaP: Pride and Prejudice, and 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

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

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

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

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(SaS,PaP) $\approx$ $0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0 \approx 0.94$.

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(SaS,PaP) $\approx$
  $0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0 \approx 0.94$.
- cos(SaS,WH) ≈ 0.79

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

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

Components of tf-idf weighting

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

Components of tf-idf weighting

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)	$1/u$
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)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

Best known combination of weighting options

Components of tf-idf weighting

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)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/\mathit{CharLength}^\alpha$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

Default: no weighting

tf-idf example

- We often use different weightings for queries and documents.

tf-idf example

- We often use different weightings for queries and documents.
- Notation: qqq.ddd

tf-idf example

- We often use different weightings for queries and documents.
- Notation: qqq.ddd
- Example: ltn.lnc

tf-idf example

- We often use different weightings for queries and documents.
- Notation: qqq.ddd
- Example: ltn.lnc
- query: logarithmic tf, idf, no normalization

tf-idf example

- 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

tf-idf example

- 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
- Isn't it bad to not idf-weight the document?

tf-idf example

- 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
- Isn't it bad to not idf-weight the document?
- Example query: "best car insurance"

tf-idf example

- 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
- Isn't it bad to not idf-weight the document?
- Example query: "best car insurance"
- Example document: "car insurance auto insurance"

tf-idf example: ltn.lnc

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										
best										
car										
insurance										

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

tf-idf example: ltn.lnc

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									
best	1									
car	1									
insurance	1									

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

tf-idf example: ltn.lnc

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					1				
best	1					0				
car	1					1				
insurance	1					2				

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

tf-idf example: ltn.lnc

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				1				
best	1	1				0				
car	1	1				1				
insurance	1	1				2				

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

tf-idf example: ltn.lnc

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				1	1			
best	1	1				0	0			
car	1	1				1	1			
insurance	1	1				2	1.3			

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

tf-idf example: ltn.lnc

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			1	1			
best	1	1	50000			0	0			
car	1	1	10000			1	1			
insurance	1	1	1000			2	1.3			

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

tf-idf example: ltn.lnc

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		1	1			
best	1	1	50000	1.3		0	0			
car	1	1	10000	2.0		1	1			
insurance	1	1	1000	3.0		2	1.3			

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

tf-idf example: ltn.lnc

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			
best	1	1	50000	1.3	1.3	0	0			
car	1	1	10000	2.0	2.0	1	1			
insurance	1	1	1000	3.0	3.0	2	1.3			

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

tf-idf example: ltn.lnc

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			
best	1	1	50000	1.3	1.3	0	0			
car	1	1	10000	2.0	2.0	1	1			
insurance	1	1	1000	3.0	3.0	2	1.3			

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

tf-idf example: ltn.lnc

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		
best	1	1	50000	1.3	1.3	0	0	0		
car	1	1	10000	2.0	2.0	1	1	1		
insurance	1	1	1000	3.0	3.0	2	1.3	1.3		

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

tf-idf example: ltn.lnc

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	
best	1	1	50000	1.3	1.3	0	0	0	0	
car	1	1	10000	2.0	2.0	1	1	1	0.52	
insurance	1	1	1000	3.0	3.0	2	1.3	1.3	0.68	

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

$\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92$
$1/1.92 \approx 0.52$
$1.3/1.92 \approx 0.68$

tf-idf example: ltn.lnc

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

tf-idf example: ltn.lnc

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

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

tf-idf example: ltn.lnc

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

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

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

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

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

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

Resources

- Chapters 6 and 7 of IIR

Resources

- Chapters 6 and 7 of IIR
- Resources at http://ifnlp.org/ir

Resources

- Chapters 6 and 7 of IIR
- Resources at http://ifnlp.org/ir
- Vector space for dummies

Resources

- Chapters 6 and 7 of IIR
- Resources at http://ifnlp.org/ir
- Vector space for dummies
- Exploring the similarity space (Moffat and Zobel, 2005)

Resources

- Chapters 6 and 7 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)