Information Retrieval
Tutorial 4: Vector Space Model

Professor: Michel Schellekens
TA: Ang Gao

University College Cork

2012-11-15

Outline

Simple Boolean vs. Ranking of result set

- Simple Boolean vs. Ranking of result set
  - Simple Boolean retrieval returns matching documents in no particular order.
  - Google (and most well designed Boolean engines) rank the result set – they rank good hits (according to some estimator of relevance) higher than bad hits.

Ranked retrieval

Ranked retrieval

- Thus far, our queries have been Boolean.

Ranked retrieval

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

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.

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.

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

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 . . .

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.

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.

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.

Problem with Boolean search: Feast or famine

- Boolean queries often result in either too few (=0) or too many (1000s) results.
- In Boolean retrieval, 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.
- In Boolean retrieval, 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

Scoring as the basis of ranked retrieval

Scoring as the basis of ranked retrieval

- We wish to rank documents that are more relevant higher than documents that are less relevant.

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?

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 $[0, 1]$.

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".

Take 1: Jaccard coefficient

Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets

Take 1: Jaccard coefficient

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

Take 1: Jaccard coefficient

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

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

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

Take 1: Jaccard coefficient

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

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

  ($A \neq \emptyset$ or $B \neq \emptyset$)
- $\textsc{jaccard}(A, A) = 1$

Take 1: Jaccard coefficient

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

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

  $(A \neq \emptyset$ or $B \neq \emptyset)$
- $\textsc{jaccard}(A, A) = 1$
- $\textsc{jaccard}(A, B) = 0$ if $A \cap B = 0$

Take 1: Jaccard coefficient

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

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

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

Take 1: Jaccard coefficient

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

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

  ($A \neq \emptyset$ or $B \neq \emptyset$)
- $\textsc{jaccard}(A, A) = 1$
- $\textsc{jaccard}(A, B) = 0$ 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

- Problem1: What is the query-document match score that the Jaccard coefficient computes for:

Jaccard coefficient: Example

- Problem1: What is the query-document match score that the Jaccard coefficient computes for:
  - Query: "University College Cork"

Jaccard coefficient: Example

- Problem1: What is the query-document match score that the Jaccard coefficient computes for:
  - Query: "University College Cork"
  - Document "Cork City Tourism guide"

Jaccard coefficient: Example

- Problem1: What is the query-document match score that the Jaccard coefficient computes for:
  - Query: "University College Cork"
  - Document "Cork City Tourism guide"
  - $\textsc{jaccard}(q, d) = 1/6$

What's wrong with Jaccard?

What's wrong with Jaccard?

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

What's wrong with Jaccard?

- It doesn't consider term frequency (how many occurrences a term has). (tf)
- Rare terms are more informative than frequent terms. Jaccard does not consider this information. (idf)

What's wrong with Jaccard?

- It doesn't consider term frequency (how many occurrences a term has). (tf)
- Rare terms are more informative than frequent terms. Jaccard does not consider this information. (idf)
- We need a more sophisticated way of normalizing for the length of a document.

tf-idf weighting

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

tf-idf weighting

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

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
- 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.

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
- 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.
- df_t is the document frequency, the number of documents that t occurs in.

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
- 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.
- 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.

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

Computing TF-IDF: An Example

- Problem2: Given a document containing terms with given frequencies:

Computing TF-IDF: An Example

- Problem2: Given a document containing terms with given frequencies:
  - $A(3), B(2), C(1)$

Computing TF-IDF: An Example

- Problem2: Given a document containing terms with given frequencies:
  - $A(3), B(2), C(1)$
  - Assume collection contains $10,000$ documents and document frequencies of these terms are:

Computing TF-IDF: An Example

- Problem2: Given a document containing terms with given frequencies:
  - $A(3), B(2), C(1)$
  - Assume collection contains $10,000$ documents and document frequencies of these terms are:
  - $A(50), B(1300), C(250)$

Computing TF-IDF: An Example

- Problem2: Given a document containing terms with given frequencies:
  - $A(3), B(2), C(1)$
  - Assume collection contains $10,000$ documents and document frequencies of these terms are:
  - $A(50), B(1300), C(250)$
  - Calculate tf-idf weight for A,B,C in this document.

Computing TF-IDF: An Example

- Problem2: Given a document containing terms with given frequencies:
  - $A(3), B(2), C(1)$
  - Assume collection contains $10,000$ documents and document frequencies of these terms are:
  - $A(50), B(1300), C(250)$
  - Calculate tf-idf weight for A,B,C in this document.
  - A: $(1 + log(3)) * log(\frac{10000}{50}) = 11.119$

Computing TF-IDF: An Example

- Problem2: Given a document containing terms with given frequencies:
  - $A(3), B(2), C(1)$
  - Assume collection contains $10,000$ documents and document frequencies of these terms are:
  - $A(50), B(1300), C(250)$
  - Calculate tf-idf weight for A,B,C in this document.
  - A: $(1 + log(3)) * log(\frac{10000}{50}) = 11.119$
  - B: $(1 + log(2)) * log(\frac{10000}{1300}) = 3.295$

Computing TF-IDF: An Example

- Problem2: Given a document containing terms with given frequencies:
  - $A(3), B(2), C(1)$
  - Assume collection contains $10,000$ documents and document frequencies of these terms are:
  - $A(50), B(1300), C(250)$
  - Calculate tf-idf weight for A,B,C in this document.
  - A: $(1 + log(3)) * log(\frac{10000}{50}) = 11.119$
  - B: $(1 + log(2)) * log(\frac{10000}{1300}) = 3.295$
  - C: $(1 + log(1)) * log(\frac{10000}{250}) = 3.689$

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 $\in \{0,1\}^{|V|}$.

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 $\in \mathbb{N}^{|V|}$.

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

Summary: Ranked retrieval in the vector space model

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
  - Euclidean distance is large for vectors of different lengths, long documents and short documents (or queries) will be positioned far apart.

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
  - Euclidean distance is large for vectors of different lengths, long documents and short documents (or queries) will be positioned far apart.
  - The angle between Semantically same documents is 0.

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
  - Euclidean distance is large for vectors of different lengths, long documents and short documents (or queries) will be positioned far apart.
  - The angle between Semantically same documents is 0.
- 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
  - Euclidean distance is large for vectors of different lengths, long documents and short documents (or queries) will be positioned far apart.
  - The angle between Semantically same documents is 0.
- Rank documents with respect to the query
- Return the top K (e.g., $K = 10$) to the user

Cosine similarity between query and 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.

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 $\vec{d}$.

Ranked retrieval in the vector space model example

Example

Consider these documents:

Doc1 Shipment of gold damaged in a fire

Doc2 Delivery of silver arrived in a silver truck

Doc3 Shipment of gold arrived in a truck

Ranked retrieval in the vector space model example

Example

Consider these documents:

Doc1 Shipment of gold damaged in a fire

Doc2 Delivery of silver arrived in a silver truck

Doc3 Shipment of gold arrived in a truck

- Compute the tf-idf weights for each terms in each document

Ranked retrieval in the vector space model example

Example

Consider these documents:

Doc1 Shipment of gold damaged in a fire

Doc2 Delivery of silver arrived in a silver truck

Doc3 Shipment of gold arrived in a truck

- Compute the tf-idf weights for each terms in each document
- Rank the three documents by computed score for the query 'gold silver truck'

First for each document and query, we compute all vector lengths (zero terms ignored)

TERM VECTOR MODEL BASED ON $w_i = tf_i * IDF_i$

Query, Q: "gold silver truck"
D_1: "Shipment of gold damaged in a fire"
D_2: "Delivery of silver arrived in a silver truck"
D_3: "Shipment of gold arrived in a truck"
D = 3; IDF = log(D/df_i)

		Counts, tf_i						Weights, $w_i = tf_i * IDF_i$			
Terms	**Q**	**D_1**	**D_2**	**D_3**	**df_i**	**D/df_i**	**IDF_i**	**Q**	**D_1**	**D_2**	**D_3**
a	0	1	1	1	3	3/3 = 1	0	0	0	0	0
arrived	0	0	1	1	2	3/2 = 1.5	0.1761	0	0	0.1761	0.1761
damaged	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
delivery	0	0	1	0	1	3/1 = 3	0.4771	0	0	0.4771	0
fire	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
gold	1	1	0	1	2	3/2 = 1.5	0.1761	0.1761	0.1761	0	0.1761
in	0	1	1	1	3	3/3 = 1	0	0	0	0	0
of	0	1	1	1	3	3/3 = 1	0	0	0	0	0
silver	1	0	2	0	1	3/1 = 3	0.4771	0.4771	0	0.9542	0
shipment	0	1	0	1	2	3/2 = 1.5	0.1761	0	0.1761	0	0.1761
truck	1	0	1	1	2	3/2 = 1.5	0.1761	0.1761	0	0.1761	0.1761

$$|D_1| = \sqrt{0.4771^2 + 0.4771^2 + 0.1761^2 + 0.1761^2} = \sqrt{0.5173} = 0.7192$$

$$|D_2| = \sqrt{0.1761^2 + 0.4771^2 + 0.9542^2 + 0.1761^2} = \sqrt{1.2001} = 1.0955$$

$$|D_3| = \sqrt{0.1761^2 + 0.1761^2 + 0.1761^2 + 0.1761^2} = \sqrt{0.1240} = 0.3522$$

$$|Q| = \sqrt{0.1761^2 + 0.4771^2 + 0.1761^2} = \sqrt{0.2896} = 0.5382$$

$$\therefore |Q| = \sqrt{\sum_i w_{Q,j}^2} \qquad \therefore |D_i| = \sqrt{\sum_i w_{i,j}^2}$$

Next, we compute all dot products (zero products ignored)

$$Q \bullet D_1 = 0.1761 * 0.1761 = 0.0310$$

$$Q \bullet D_2 = 0.4771 * 0.9542 + 0.1761 * 0.1761 = 0.4862$$

$$Q \bullet D_3 = 0.1761 * 0.1761 + 0.1761 * 0.1761 = 0.0620$$

$$\therefore \; Q \bullet D_i = \sum_i w_{Q,j} w_{i,j}$$

Now we calculate the similarity values

$$\text{Cosine } \theta_{D_1} = \frac{Q \bullet D_1}{|Q| * |D_1|} = \frac{0.0310}{0.5382 * 0.7192} = 0.0801$$

$$\text{Cosine } \theta_{D_2} = \frac{Q \bullet D_2}{|Q| * |D_2|} = \frac{0.4862}{0.5382 * 1.0955} = 0.8246$$

$$\text{Cosine } \theta_{D_3} = \frac{Q \bullet D_3}{|Q| * |D_3|} = \frac{0.0620}{0.5382 * 0.3522} = 0.3271$$

$$\therefore \; \text{Cosine } \theta_{D_i} = \text{Sim}(Q, D_i)$$

$$\therefore \; \text{Sim}(Q, D_i) = \frac{\sum_i w_{Q,j} w_{i,j}}{\sqrt{\sum_j w_{Q,j}^2} \sqrt{\sum_i w_{i,j}^2}}$$

Ranked retrieval in the vector space model example

Problem 3

Consider these documents:

Doc1 a a b e c

Doc2 b c a c c

Doc3 e b d

Ranked retrieval in the vector space model example

Problem 3

Consider these documents:
Doc1 a a b e c
Doc2 b c a c c
Doc3 e b d

- Compute the tf-idf weights for each terms in each document

Ranked retrieval in the vector space model example

Problem 3

Consider these documents:
Doc1 a a b e c
Doc2 b c a c c
Doc3 e b d

- Compute the tf-idf weights for each terms in each document
- Rank the three documents by computed score for the query 'a c d'

Query: a c d
D1: a a b e c
D2: b c a c c
D3: e b d

IDF = log(D/df)

W = (1+log(tf))*idf

Terms	Q	D1	D2	D3	df	D/df	IDF	Q	D1	D2	D3
a	1	2	1	0	2	1.5	0.1761	0.1761	0.2291	0.1761	0.0000
b	0	1	1	1	3	1	0.0000	0.0000	0.0000	0.0000	0.0000
c	1	1	3	0	2	1.5	0.1761	0.1761	0.1761	0.2601	0.0000
d	1	0	0	1	1	3	0.4771	0.4771	0.0000	0.0000	0.4771
e	0	1	0	1	2	1.5	0.1761	0.0000	0.1761	0.0000	0.1761

|D1|=	0.3384
|D2|=	0.3141
|D3|=	0.5086
|Q|=	0.5382

Sim(Q,D1)=	0.3918
Sim(Q,D2)=	0.4544
Sim(Q,D3)=	0.8317

Q*D1=	0.0714
Q*D2=	0.0768
Q*D3=	0.2276