	Title : CS4611 Study
	Student Name : Brian O Regan
	Student Number : 110707163
	Module : CS4611
	Exam Date: Friday 16th May @ 14:00

What is IR?

Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need within large collections (usually stored on computers).

Effectiveness of an IR System

Precision : Fraction of retrieved documents that are relevant to users information need.
Recall : Fraction of relevant documents in collection that are retrieved.

To build IR system we need index the documents in advance.

Term-document incidence matrix

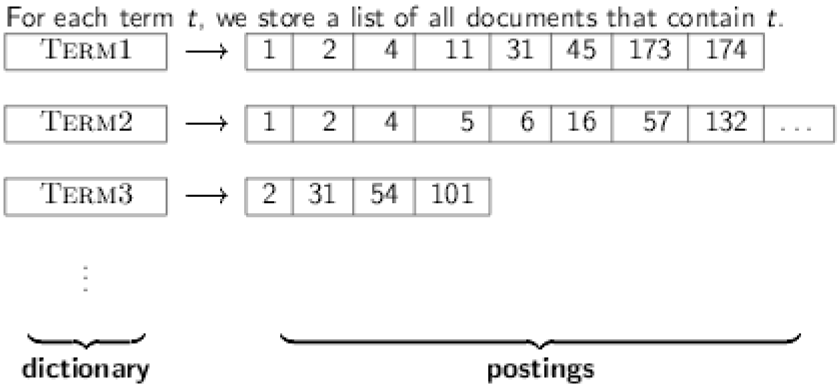
- Terms are the indexed units (usual words).
- Column: a vector for each document, showing the terms that occur in it.
- Row: a vector for each term, which shows the documents it appears in.
- Query: Answer Boolean expression of terms, do bitwise AND OR and NOT on vectors e.g.: 110100 and 110111 and 101111 = 100100.

	Doc1	Doc2	Doc3	Doc4	Doc5
TERM1	1	1	0	0	0
TERM2	1	1	0	1	0
TERM3	1	1	0	1	1
TERM4	0	1	0	0	0
TERM5	1	0	0	0	0

. . .
Entry is 1 if term occurs

Inverted Index

For each term t, we store a list of all documents that contain t.



UCC Coláiste na hOllscoile Corcaigh, Éire University College Cork, Ireland	Title : CS4611 Study
	Student Name : Brian O Regan
	Student Number : 110707163
	Module : CS4611
	Exam Date: Friday 16th May @ 14:00

Inverted Index Construction

1. Collect the documents to be indexed:
   Friends, Romans, countrymen. | So let it be with Caesar | ...
2. Tokenize the text, turning each document into a list of tokens:
   Friends | Romans | countrymen | So | ...
3. Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms: friend | roman | countryman | so | ...
4. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

Intersecting Two Posting Lists

TERM1 ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

TERM2 ⟶ 2 → 31 → 54 → 101

Intersection ⟹ 2 → 31

- This is linear in the length of the postings lists.
- Note: This only works if postings lists are sorted.

```
INTERSECT(p1, p2)
  answer ← ⟨ ⟩
  while p1 ≠ NIL and p2 ≠ NIL
  do if docID(p1) = docID(p2)
       then ADD(answer, docID(p1))
            p1 ← next(p1)
            p2 ← next(p2)
       else if docID(p1) < docID(p2)
              then p1 ← next(p1)
              else p2 ← next(p2)
  return answer
```

What is a Token?

- A token is a sequence of characters in a document.
- Example Friends / Romans / Countrymen are tokens in an input.

What is a Term?

- A term is a (normalised) word type, which is an entry in our IR system dictionary.
- We need normalise words in indexed text as well as query words into the same form

	Title : CS4611 Study
	Student Name : Brian O Regan
	Student Number : 110707163
	Module : CS4611
	Exam Date: Friday 16th May @ 14:00

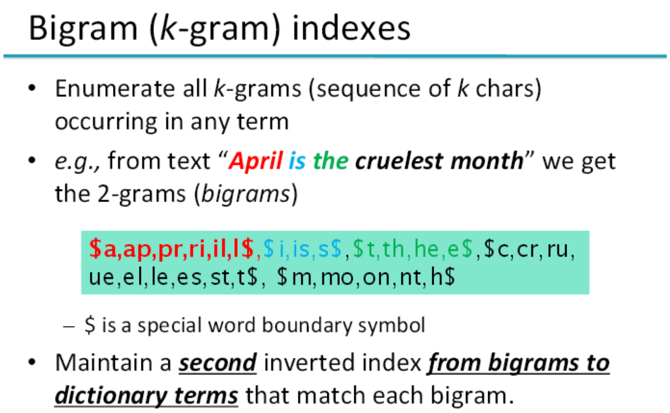
What is Lemmatization?

- To reduce inflectional / variant forms to base form
- Example : am, are, is -> be
- Example : car, cars, car's, cars' -> car

What is Stemming

- Reduce terms to their "roots" before indexing

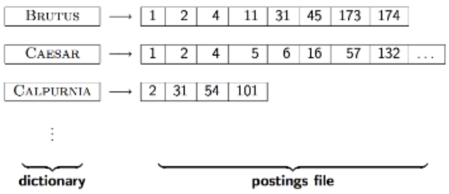
Bigram (k-gram) Indexes



Edit Distance

- Given two strings $S_1$and S_2, the minimum number of operations to convert one to the other
- Operations : Insert, Delete, Replace (Transposition)
- Example : dof -> dog = 1
- Example : cat -> act = 2 (1 transposition)
- Example : cat -> dog = 3

For each term t, we store a list of all documents that contain t.



- Motivation for compression in information retrieval systems
- How can we compress the dictionary component of the inverted index?
- How can we compress the postings component of the inverted index?
- Term statistics: how are terms distributed in document collections?

Why Compression?

- Use less disk space (saves money)
- Keep more stuff in memory (increases speed)
- Increase speed of transferring data from disk to memory (again, increases speed)
  - [read compressed data and decompress in memory] is faster than [read uncompressed data]
- Premise: Decompression algorithms are fast.
- This is true of the decompression algorithms we will use.

Why compression in IR?

- First, we will consider space for dictionary
  - Main motivation for dictionary compression: make it small enough to keep in main memory
- Then for the postings file
  - Motivation: reduce disk space needed, decrease time needed to read from disk
  - Note: Large search engines keep significant part of postings in memory
- We will devise various compression schemes for dictionary and postings.

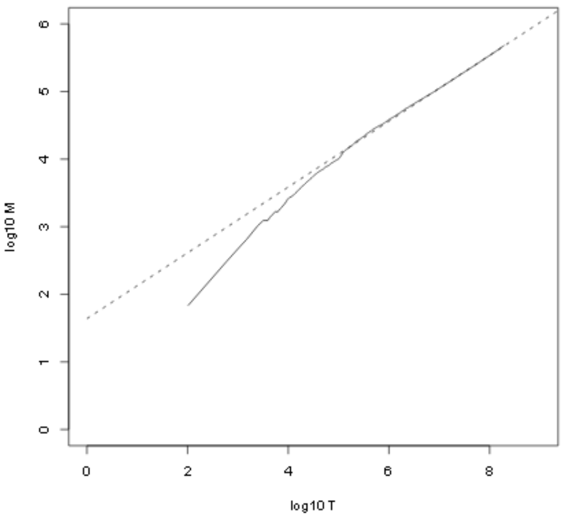
Lossy vs Lossless Compression

- Lossy compression: Discard some information
- Several of the preprocessing steps we frequently use can be viewed as lossy compression:
  - downcasing, stop words, porter, number elimination
- Lossless compression: All information is preserved.
  - What we mostly do in index compression

How big is the term vocabulary?

- That is, how many distinct words are there?
- In practice,the vocabulary will keep growing with collection size.(eg: names of new people)
- Heaps' law: $M = kT^b$
- M is the size of the vocabulary, T is the number of tokens in the collection.
- Typical values for the parameters k and b are: $30 \leq k \leq 100$ and $b \approx 0.5$. Thus $M \approx k\sqrt{T}$
- Notice $logM = logk + blogT (y = c + bx)$
- Heaps' law is linear in log-log space.
  - It is the simplest possible relationship between collection size and vocabulary size in log-log space.
  - An empirical finding(Empirical law).

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 $b = 0.49$.

Empirical fit for Reuters

- Good, as we just saw in the graph.
- Example: for the first 1,000,020 tokens Heaps' law predicts 38,323 terms:

$$44 \times 1{,}000{,}020^{0.49} \approx 38{,}323$$

- The actual number is 38,365 terms, very close to the prediction.
- Empirical observation: fit is good in general.

Basic Knowledge to Remember

To binary represent an integer n, number of bits need = $\lfloor log_2(n) \rfloor + 1$
$n = \{2\}_{10} = \{10\}_2$
$n = \{3\}_{10} = \{11\}_2$
$n = \{4\}_{10} = \{100\}_2$

Dictionary Compression

- The dictionary is small compared to the postings file.
- But we want to keep it in memory.
- Also: competition with other applications, cell phones, onboard computers, fast startup time
- So compressing the dictionary is important.

Recall : Dictionary as Array of Fixed-Width Entries

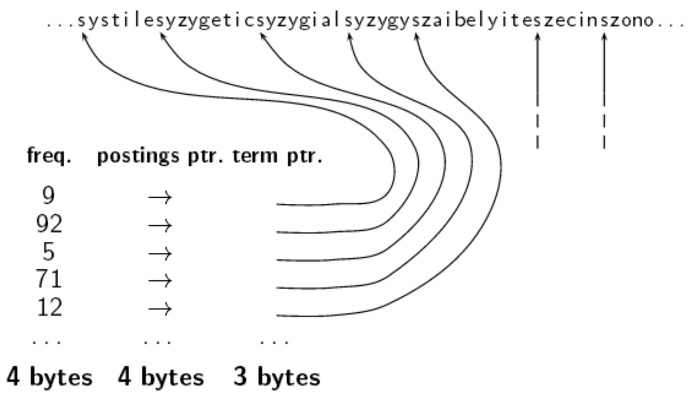
	term	document frequency	pointer to postings list
	a	656,265	$\longrightarrow$
	aachen	65	$\longrightarrow$
	...	...	...
	zulu	221	$\longrightarrow$
space needed:	20 bytes	4 bytes	4 bytes

Space for Reuters: (20+4+4)*400,000 = 11.2 MB

Fixed-Width Entries are Bad !

- Most of the bytes in the term column are wasted.
  - We allot 20 bytes for terms of length 1.
- We can't handle HYDROCHLOROFLUOROCARBONS and SUPERCALIFRAGILISTICEXPIALIDOCIOUS
- Average length of a term in English: 8 characters
- How can we use on average 8 characters per term?

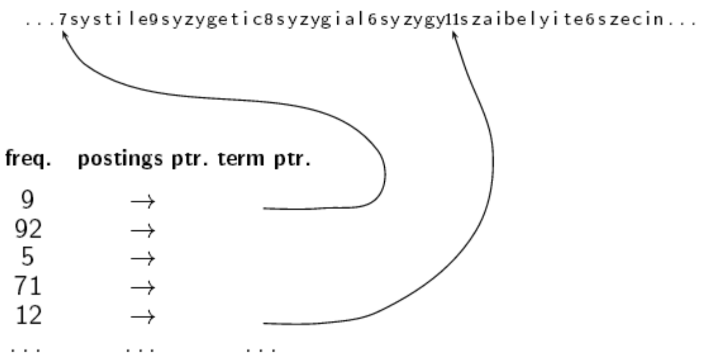
Dictionary as a String



Space for Dictionary as a String

- 4 bytes per term for frequency
- 4 bytes per term for pointer to postings list
- 8 bytes (on average) for term in string
- 3 bytes per pointer into string (need $\log_2 8 \cdot 400000 < 24$ bits to resolve $8 \cdot 400{,}000$ positions)
- Space: $400{,}000 \times (4 + 4 + 3 + 8) = 7.6$MB (compared to 11.2 MB for fixed-width array)

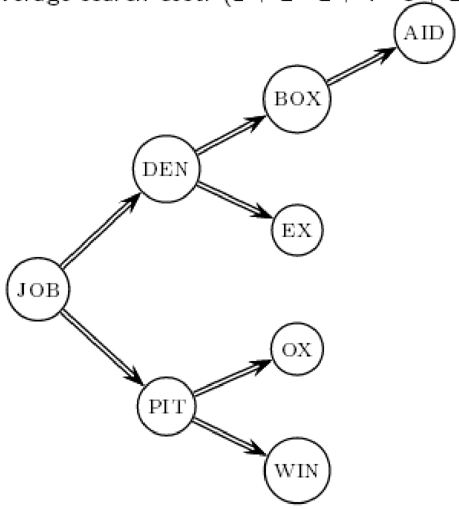
Dictionary as a String with Blocking



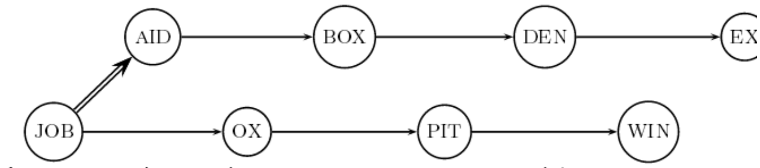
- Example block size $k = 4$
- Where we used 4×3 bytes for term pointers without blocking ...
- ...we now use 3 bytes for one pointer plus 4 bytes for indicating the length of each term.
- We save $12 - (3 + 4) = 5$ bytes per block.
- Total savings: $400{,}000/4 * 5 = 0.5$ MB
- This reduces the size of the dictionary from 7.6 MB to 7.1 MB.

Lookup of a term Without Blocking

Average search cost: $(1 + 2 * 2 + 4 * 3 + 1 * 4)/8 \approx 2.6$ steps



Lookup of a term with Blocking (slightly) slower



Average search cost: $(2+3+4+5+1+2+3+4)/8 \approx 3$ steps.

Postings Compression

- The postings file is much larger than the dictionary, factor of at least 10.
- Key desideratum: store each posting compactly
- A posting for our purposes is a docID.
- For Reuters (800,000 documents), we would use 32 bits per docID when using 4-byte integers.
- Alternatively, we can use $\log_2 800{,}000 \approx 19.6 < 20$ bits per docID.
- Our goal: use a lot less than 20 bits per docID.

Key Idea : Store Gaps instead of docIDs

- Each postings list is ordered in increasing order of docID.
- Example postings list: COMPUTER: 283154, 283159, 283202, ...
- It suffices to store gaps: 283159-283154=5, 283202-283154=43
- Example postings list using gaps : COMPUTER: 283154, 5, 43, ...
- Gaps for frequent terms are small.
- Thus: We can encode small gaps with fewer than 20 bits.

	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								

Variable Length Encoding

- Aim:
  - For ARACHNOCENTRIC and other rare terms, we will use about 20 bits per gap (= posting).
  - For THE and other very frequent terms, we will use only a few bits per gap (= posting).
- In order to implement this, we need to devise some form of variable length encoding.
- Variable length encoding uses few bits for small gaps and many bits for large gaps.

Variable Byte (BV) Code

- Used by many commercial/research systems
- Good low-tech blend of variable-length coding and sensitivity to alignment matches (bit-level codes, see later).
- Dedicate 1 bit (high bit) to be a continuation bit c.
- If the gap G fits within 7 bits, binary-encode it in the 7 available bits and set $c = 1$.
- Else: encode lower-order 7 bits and then use one or more additional bytes to encode the higher order bits using the same algorithm.
- At the end set the continuation bit of the last byte to 1 ($c = 1$) and of the other bytes to 0 ($c = 0$).

Examples

docIDs	824	829	215406
gaps		5	214577
VB code	00000110 10111000	10000101	00001101 00001100 10110001

Gamma Codes for Gap Encoding

- You can get even more compression with another type of variable length encoding: bitlevel code.
- Gamma code is the best known of these.
- First, we need unary code to be able to introduce gamma code.
- Unary code
  - Represent n as n 1s with a final 0.
  - Unary code for 3 is 1110
  - Unary code for 40 is 11111111111111111111111111111111111111110
  - Unary code for 70 is:

11111111111111111111111111111111111111111111111111111111111111111111110

Gamma Code

- Represent a gap G as a pair of length and offset.
- Offset is the gap in binary, with the leading bit chopped off.
- For example $13 \rightarrow 1101 \rightarrow 101 =$ offset
- Length is the length of offset.
- For 13 (offset 101), this is 3.
- Encode length in unary code: 1110.
- Gamma code of 13 is the concatenation of length and offset: 1110101.

Length of Gamma Code

- The length of *offset* is $\lfloor \log_2 G \rfloor$ bits.
- The length of *length* is $\lfloor \log_2 G \rfloor + 1$ bits,
- So the length of the entire code is $2 \times \lfloor \log_2 G \rfloor + 1$ bits.
- γ codes are always of odd length.
- Gamma codes are within a factor of 2 of the optimal encoding length $\log_2 G$.

UCC Coláiste na hOllscoile Corcaigh, Éire University College Cork, Ireland	Title : CS4611 Study
	Student Name : Brian O Regan
	Student Number : 110707163
	Module : CS4611
	Exam Date: Friday 16th May @ 14:00

Simple Boolean vs Ranking of Result

- 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

- 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.
- 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 a 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".

UCC Coláiste na hOllscoile Corcaigh, Éire University College Cork, Ireland	*Title : CS4611 Study*
	Student Name : Brian O Regan
	Student Number : 110707163
	Module : CS4611
	Exam Date: Friday 16th May @ 14:00

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 $A \cap B = 0$
- A and B don't have to be the same size.
- Always assigns a number between 0 and 1.

Example

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

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

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

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

- Best known weighting scheme in information retrieval
- 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.
- 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 : 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 $\in \{0, 1\}^{|V|}$.

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

UCC Coláiste na hOllscoile Corcaigh, Éire University College Cork, Ireland	*Title : CS4611 Study*
	Student Name : Brian O Regan
	Student Number : 110707163
	Module : CS4611
	Exam Date: Friday 16th May @ 14:00

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

- 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

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

- 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

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

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

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

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

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

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

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

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

Precision and Recall

- Precision (P) is the fraction of retrieved documents that are relevant

$$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = P(\text{relevant}|\text{retrieved})$$

- Recall (R) is the fraction of relevant documents that are retrieved

$$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = P(\text{retrieved}|\text{relevant})$$

	Relevant	Nonrelevant
Retrieved	true positives (TP)	false positives (FP)
Not retrieved	false negatives (FN)	true negatives (TN)

$$P = TP/(TP + FP)$$
$$R = TP/(TP + FN)$$

accuracy $= (TP + TN)/(TP + FP + FN + TN)$.

Accuracy

- Accuracy is the fraction of decisions (relevant/nonrelevant) that are correct.
- In terms of the contingency table above, accuracy $= (TP + TN)/(TP + FP + FN + TN)$.
- Why is accuracy not a useful measure for web information retrieval?
- Ans: In IR system normaly only a small fraction of documents in the collection are relevance, as a result $TN >> TP$, even we have a good IR system which only retrieve relevant documents, the accuracy between this good IR system with a poor system(such as always return nothing) is small, thus this measurement can't help us evaluate IR system.

Exercise

- The snoogle search engine below always returns 0 results ("0 matching results found"), regardless of the query. Why does snoogle demonstrate that accuracy is not a useful measure in IR?

- Simple trick to maximize accuracy in IR: always say no and return nothing
- You then get 99.99% accuracy on most queries.
- Searchers on the web (and in IR in general) want to find something and have a certain tolerance for junk.
- It's better to return some bad hits as long as you return something.
- $\rightarrow$ We use precision, recall, and F for evaluation, not accuracy.

Precision / Recall Tradeoff

- You can increase recall by returning more docs.
- Recall is a non-decreasing function of the number of docs retrieved.
- A system that returns all docs has 100% recall!
- The converse is also true (usually): It's easy to get high precision for very low recall.
- Which is better: IR sytem1 P: 63% R: 57%, IR system2 P: 69% R:60%

A Combined Measure : F

- F allows us to trade off precision against recall.
- $$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \quad \text{where} \quad \beta^2 = \frac{1-\alpha}{\alpha}$$
- $\alpha \in [0,1]$ and thus $\beta^2 \in [0,\infty]$
- Most frequently used: balanced F with $\beta = 1$ or $\alpha = 0.5$
  - This is the harmonic mean of P and R: $\frac{1}{F} = \frac{1}{2}(\frac{1}{P} + \frac{1}{R})$
  - $F = \frac{2PR}{P+R}$

F : Exercise

	relevant	not relevant	
retrieved	20	40	60
not retrieved	60	1,000,000	1,000,060
	80	1,000,040	1,000,120

- $P = 20/(20+40) = 1/3$
- $R = 20/(20+60) = 1/4$
- $F_1 = 2\frac{1}{\frac{1}{\frac{1}{3}}+\frac{1}{\frac{1}{4}}} = 2/7$

F: Why Harmonic Mean?

- Why don't we use a different mean of P and R as a measure?
  - e.g., the arithmetic mean $\frac{P+R}{2}$
- The simple (arithmetic) mean is 50% for "return-everything" search engine, which is too high.
- Desideratum: Punish really bad performance on either precision or recall.
- Taking the minimum achieves this.
- But minimum is not smooth and hard to weight.
- F (harmonic mean) is a kind of smooth minimum.

Difficulties in using Precision, Recall and F

- We need relevance judgments for information-need-document pairs – but they are expensive to produce.
- For alternatives to using precision/recall and having to produce relevance judgments

Framework for the Evaluation of an IR System

- *test collection* consisting of (i) a document collection, (ii) a test suite of information needs and (iii) a set of relevance judgements for each *doc-query* pair

- *gold-standard* judgement of relevance
  → classification of a document either as relevant or as irrelevant wrt an information need

Assessing Relevance

- How good is an IR system at satisfaying an information need ?
- Needs an agreement between judges

  → computable via the **kappa** statistic:

$$kappa = \frac{P(A) - P(E)}{1 - P(E)}$$

where:
$P(A)$: the proportion of agreements within the judgements
$P(E)$: what agreement would we get by chance

Example:

Consider the following judgements (from Manning et al., 2008):

		Judge 2		
		Yes	No	Total
Judge 1	Yes	300	20	320
	No	10	70	80
	Total	310	90	400

- $$P(A) = \frac{370}{400} \qquad P(E) = P(rel)^2 + P(notrel)^2$$
  $$P(rel) = \frac{1}{2}\frac{320}{400} + \frac{1}{2}\frac{310}{400} = \frac{320+310}{800} \qquad P(notrel) = \frac{80+90}{800}$$
- $$kappa = \frac{P(A) - P(E)}{1 - P(E)} \qquad k = 0.776$$

$P(A)$ is the proportion of agreements within the judgements
$P(E)$ is the proportion of expected agreements

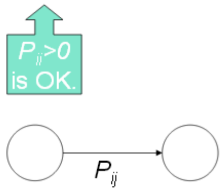
Relevance Continued

- Interpretation of the kappa statistic k:
  - Values of k in the interval [2/3, 1.0] are seen as acceptable.
  - With smaller values: need to redesign relevance assessment methodology used etc.
- Note that the kappa statistic can be negative if the agreements between judgements are worse than random
- In case of large variations between judgements, one can choose an assessor as a gold-standard

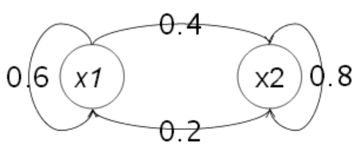
  → considerable impact on the *absolute* assessment
  → little impact on the *relative* assessment

Markov Chains

- A Markov chain consists of n <u>states</u>, plus an $n \times n$ <u>transition probability matrix</u> **P**.
- At each step, we are in exactly one of the states.
- For $1 \leq i,j \leq n$, the matrix entry P_{ij} tells us the probability of j being the next state, given we are currently in state i.



Example



	x1	x2
x1	**0.6**	**0.4**
x2	0.2	0.8

$P_0(x1)=1 \quad P_0(x2)=0$

What is $P_1(x1)$ and $P_1(x2)$

$$P_1(x1) = P_0(x1) \cdot P_{x1x1} + P_0(x2) \cdot P_{x2x1} = 1*0.6+0*0.2 = 0.6$$

$$P_1(x2) = P_0(x1) \cdot P_{x1x2} + P_0(x2) \cdot P_{x2x2} = 1*0.4+0*0.8 = 0.4$$

$$P_1(x2) = 1 - P_1(x1)$$

$$P_t(x_1) = P_{t-1}(x_1) \cdot P_{x1x1} + P_{t-1}(x_2) \cdot P_{x2x1}$$

$P_1(x1)=0.6 \quad P_1(x2)=0.4$

What is $P_2(x1)$ and $P_3(x1)$?

$P_2(x1) = P_1(x1) * P_{x1x1} + P_1(x2) * P_{x2x1} = 0.6*0.6+0.4*0.2 = 0.44$

$P_3(x1) = P_2(x1) * P_{x1x1} + P_2(x2) * P_{x2x1} = 0.44*0.6+0.56*0.2 = 0.376$

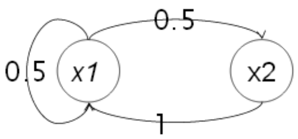
How to calculate $P_\infty(x1)$?

$P_t(x_1) = P_{t-1}(x_1) * P_{x1x1} + P_{t-1}(x_2) * P_{x2x1}$

When t goes to ∞ *notice* $P_t(x_1) = P_{t-1}(x_1)$!

$P_t(x_1) = P_t(x_1) * 0.6 + (1 - P_t(x_1)) * 0.2$

$P_t(x_1) = \frac{1}{3}$ when t -> ∞ steady state probability



	x1	x2
x1	**0.5**	**0.5**
x2	1	0

Calculate steady state probability for x1 and x2

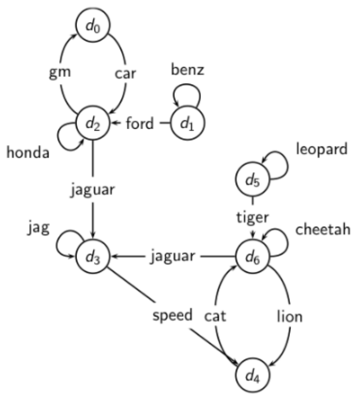
$P_t(x_1) = \frac{2}{3}$ when t -> ∞

$P_t(x_2) = \frac{1}{3}$ when t -> ∞

Model Behind PageRank : Random Walk

- Imagine a web surfer doing a random walk on the web
  - Start at a random page
  - At each step, go out of the current page along one of the links on that page, equiprobably
- In the steady state, each page has a long-term visit rate.
- This long-term visit rate is the page's PageRank.
- PageRank = long-term visit rate = steady state probability.

Example Web Graph



Link Matrix for Web Graph

	d_0	d_1	d_2	d_3	d_4	d_5	d_6
d_0	0	0	1	0	0	0	0
d_1	0	1	1	0	0	0	0
d_2	1	0	1	1	0	0	0
d_3	0	0	0	1	1	0	0
d_4	0	0	0	0	0	0	1
d_5	0	0	0	0	0	1	1
d_6	0	0	0	1	1	0	1

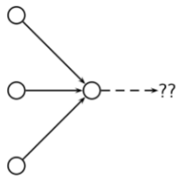
Transistion Probability Matrix P for Web Graph

	d_0	d_1	d_2	d_3	d_4	d_5	d_6
d_0	0.00	0.00	1.00	0.00	0.00	0.00	0.00
d_1	0.00	0.50	0.50	0.00	0.00	0.00	0.00
d_2	0.33	0.00	0.33	0.33	0.00	0.00	0.00
d_3	0.00	0.00	0.00	0.50	0.50	0.00	0.00
d_4	0.00	0.00	0.00	0.00	0.00	0.00	1.00
d_5	0.00	0.00	0.00	0.00	0.00	0.50	0.50
d_6	0.00	0.00	0.00	0.33	0.33	0.00	0.33

Long Term Visit Rate

- Recall: PageRank = long-term visit rate.
- Long-term visit rate of page *d* is the probability that a web surfer is at page *d* at a given point in time.
- Next: what properties must hold of the web graph for the long-term visit rate to be well defined?
- The web graph must correspond to an ergodic Markov chain.
- First a special case: The web graph must not contain dead ends.

Dead Ends



- The web is full of dead ends.
- Random walk can get stuck in dead ends.
- If there are dead ends, long-term visit rates are not well-defined (or non-sensical).

Teleporting – to get us of dead ends

- At a dead end, jump to a random web page with prob. 0.1/ *N* .
- At a non-dead end, with probability 10%, jump to a random web page (to each with a probability of 0.1/*N*).
- With remaining probability (90%), go out on a random hyperlink.
  - For example, if the page has 4 outgoing links: randomly choose one with probability (1-0.10)/4=0.225
- 10% is a parameter, the teleportation rate.
- Note: "jumping" from dead end is independent of teleportation rate.

Transition matrix with teleporting

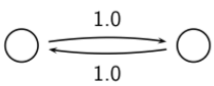
	d_0	d_1	d_2	d_3	d_4	d_5	d_6
d_0	0.02	0.02	0.88	0.02	0.02	0.02	0.02
d_1	0.02	0.45	0.45	0.02	0.02	0.02	0.02
d_2	0.31	0.02	0.31	0.31	0.02	0.02	0.02
d_3	0.02	0.02	0.02	0.45	0.45	0.02	0.02
d_4	0.02	0.02	0.02	0.02	0.02	0.02	0.88
d_5	0.02	0.02	0.02	0.02	0.02	0.45	0.45
d_6	0.02	0.02	0.02	0.31	0.31	0.02	0.31

Result of Teleporting

- With teleporting, we cannot get stuck in a dead end.
- But even without dead ends, a graph may not have well-defined long-term visit rates.
- More generally, we require that the Markov chain be ergodic.

Ergodic Markov Chains

- A Markov chain is ergodic if it is irreducible and aperiodic.
- Irreducibility. Roughly: there is a path from any other page.
- Aperiodicity. Roughly: The pages cannot be partitioned such that the random walker visits the partitions sequentially.
- A non-ergodic Markov chain:



- Theorem: For any ergodic Markov chain, there is a unique long-term visit rate for each state.
- This is the steady-state probability distribution.
- Over a long time period, we visit each state in proportion to this rate.
- It doesn't matter where we start.
- Teleporting makes the web graph ergodic.
- $\Longrightarrow$Web-graph+teleporting has a steady-state probability distribution.
- $\Longrightarrow$Each page in the web-graph+teleporting has a PageRank.

Formalisation of "visit" : Probability Vector

- A probability (row) vector $\vec{x} = (x_1, \ldots, x_N)$ tells us where the random walk is at any point.
- Example: (0 0 0 … 1 … 0 0 0)

0	0	0	…	1	…	0	0	0
1	2	3	…	*i*	…	N-2	N-1	N

- More generally: the random walk is on the page *i* with probability x_i.
- Example:

0.05	0.01	0.0	…	0.2	…	0.01	0.05	0.03
1	2	3	…	*i*	…	N-2	N-1	N

- $\Sigma x_i = 1$

Change in Probability Vector

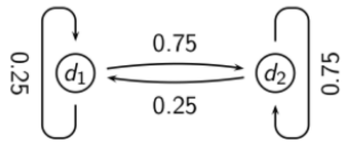
- If the probability vector is $\vec{x} = (x_1, \ldots, x_N)$, at this step, what is it at the next step?
- Recall that row *i* of the transition probability matrix *P* tells us where we go next from state *i*.
- So from $\vec{x}$, our next state is distributed as $\vec{x}P$.

Steady State in Vector Notation

- The steady state in vector notation is simply a vector $\vec{\pi} = (\pi_1, \pi_2, \ldots, \pi_N)$ of probabilities.
- (We use $\vec{\pi}$ to distinguish it from the notation for the probability vector $\vec{x}$.)
- π is the long-term visit rate (or PageRank) of page *i*.
- So we can think of PageRank as a very long vector – one entry per page.

Steady-State Distribution : Example

- What is the PageRank / steady state in this example?



One way of Computing the PageRank $\vec{\pi}$

- Start with any distribution $\vec{x}$, e.g., uniform distribution
- After one step, we're at $\vec{x}P$.
- After two steps, we're at $\vec{x}P^2$.
- After k steps, we're at $\vec{x}P^k$.
- Algorithm: multiply $\vec{x}$ by increasing powers of P until convergence.
- This is called the power method.
- Recall: regardless of where we start, we eventually reach the steady state $\vec{\pi}$.
- Thus: we will eventually (in asymptotia) reach the steady state.

Computing PageRank: Power Example

	x_1 $P_t(d_1)$	x_2 $P_t(d_2)$			
			$P_{11} = 0.1$ $P_{21} = 0.3$	$P_{12} = 0.9$ $P_{22} = 0.7$	
t_0	0	1	0.3	0.7	$= \vec{x}P$
t_1	0.3	0.7	0.24	0.76	$= \vec{x}P^2$
t_2	0.24	0.76	0.252	0.748	$= \vec{x}P^3$
t_3	0.252	0.748	0.2496	0.7504	$= \vec{x}P^4$
				. . .	. . .
t_∞	0.25	0.75	0.25	0.75	$= \vec{x}P^\infty$

PageRank vector $= \vec{\pi} = (\pi_1, \pi_2) = (0.25, 0.75)$

$P_t(d_1) = P_{t-1}(d_1) * P_{11} + P_{t-1}(d_2) * P_{21}$

$P_t(d_2) = P_{t-1}(d_1) * P_{12} + P_{t-1}(d_2) * P_{22}$

PageRank Summary

- Preprocessing
  - Given graph of links, build matrix *P*
  - Apply teleportation
  - From modified matrix, compute $\vec{\pi}$
  - $\vec{\pi}_i$ is the PageRank of page *i*.
- Query processing
  - Retrieve pages satisfying the query
  - Rank them by their PageRank
  - Return reranked list to the user

PageRank Issues

- Real surfers are not random surfers.
  - Examples of nonrandom surfing: back button, short vs. long paths, bookmarks, directories – and search!
  - → Markov model is not a good model of surfing.
  - But it's good enough as a model for our purposes.
- Simple PageRank ranking produces bad results for many pages.
  - Consider the query [video service].
  - The Yahoo home page (i) has a very high PageRank and (ii) contains both *video* and *service*.
  - If we rank all Boolean hits according to PageRank, then the Yahoo home page would be top-ranked.
  - Clearly not desireble.
- In practice: rank according to weighted combination of raw text match, anchor text match, PageRank & other factors.
- need more lecture on Learning to Rank.

How Important is PageRank?

- Frequent claim: PageRank is the most important component of web ranking.
- The reality:
  - There are several components that are at least as important: e.g., anchor text, phrases, proximity, tiered indexes ...
  - Rumor has it that PageRank in his original form (as presented here) now has a negligible impact on ranking!
  - However, variants of a page's PageRank are still an essential part of ranking.
  - Addressing link spam is difficult and crucial.