

CS918: LECTURE 13

Ranking in Information Retrieval

Arkaitz Zubiaga, 14th November, 2018

LECTURE 13: CONTENTS

- Why Rank Results in Information Retrieval.
- Scoring with the Jaccard coefficient.
- TF, IDF and TF-IDF weighing schemes.
- The vector space ranking model.

RECAP: INDEXING WITH POSITIONAL INDICES

- In the postings, **store for each term the position(s)** in which tokens of it appear:
- <term, number of docs containing term;
doc1: position1, position2 ... ;
doc2: position1, position2 ... ;
etc.>

RECAP: INDEXING WITH POSITIONAL INDICES

- With positional indices:
 - We can search for queries of any length, e.g. **“to be or not be”**
 - We can issue proximity queries, e.g. **“Student” AROUND(5) “Warwick”**

RECAP: INDEXING WITH POSITIONAL INDICES

- So far, however, we have limited to **boolean search results**.
 - **Documents** in our database **match or don't match a query**.
- This **isn't always ideal**, particularly for large collections.
 - In web search, imagine getting 1000s of results matching our query, randomly ordered.
 - We **need to rank results**, e.g. by relevance.

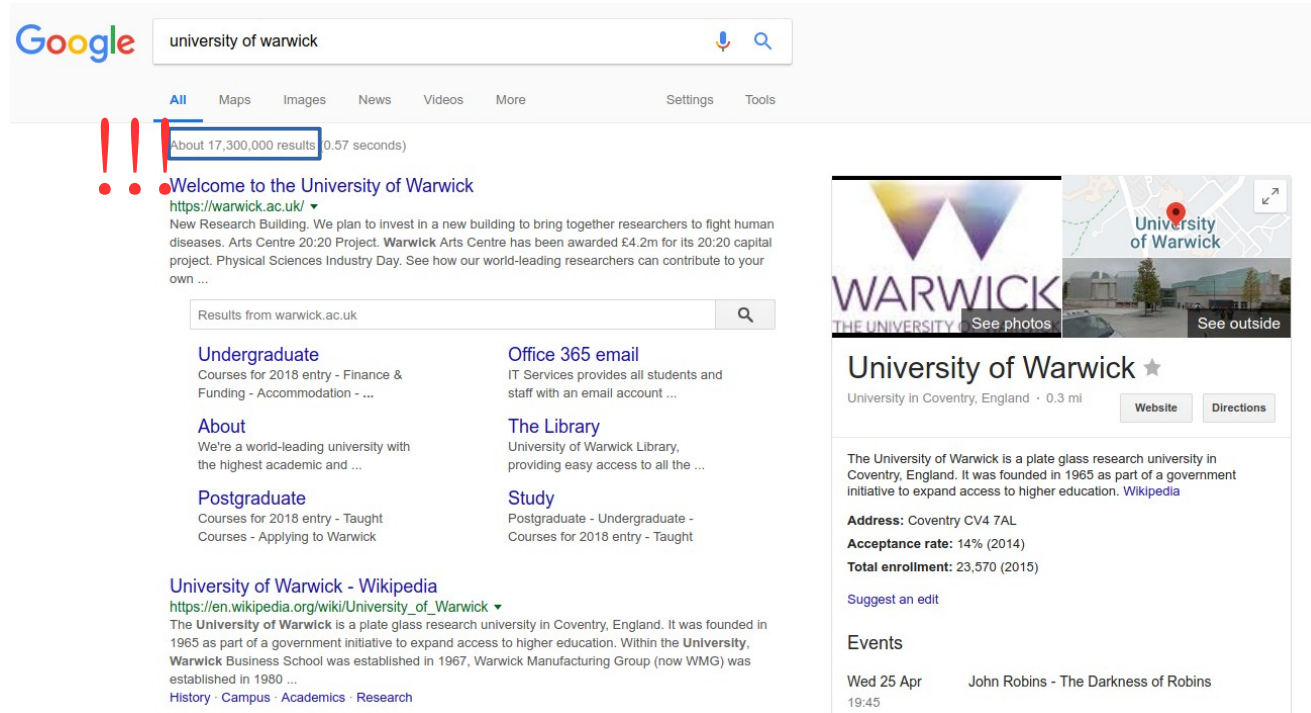
RANKED INFORMATION RETRIEVAL

- **Boolean retrieval:**
 - Unordered list of documents matching query.
 - Query often with operators (e.g. +sports -basketball).
- **Ranked retrieval:**
 - System returns an ordering over the top documents.
 - Queries generally free text (e.g. sporting events in Coventry).

LARGE RESULTS ARE NOT AN ISSUE (IF THEY ARE RANKED!)

- When a system produces a **ranked result set**:
 - **Size** of the result set is **not an issue**.
 - We **just show the top k** (≈ 10) results.
 - We **don't overwhelm** the user.

LARGE RESULTS ARE NOT AN ISSUE (IF THEY ARE RANKED!)



Google university of warwick

All Maps Images News Videos More Settings Tools

!!! About 17,300,000 results (0.57 seconds)

Welcome to the University of Warwick
<https://warwick.ac.uk/>
 New Research Building. We plan to invest in a new building to bring together researchers to fight human diseases. Arts Centre 20:20 Project. Warwick Arts Centre has been awarded £4.2m for its 20:20 capital project. Physical Sciences Industry Day. See how our world-leading researchers can contribute to your own ...

Results from warwick.ac.uk

Undergraduate
 Courses for 2018 entry - Finance & Funding - Accommodation - ...

About
 We're a world-leading university with the highest academic and ...

Postgraduate
 Courses for 2018 entry - Taught Courses - Applying to Warwick

Office 365 email
 IT Services provides all students and staff with an email account ...

The Library
 University of Warwick Library, providing easy access to all the ...

Study
 Postgraduate - Undergraduate - Courses for 2018 entry - Taught

University of Warwick - Wikipedia
https://en.wikipedia.org/wiki/University_of_Warwick
 The University of Warwick is a plate glass research university in Coventry, England. It was founded in 1965 as part of a government initiative to expand access to higher education. Within the University, Warwick Business School was established in 1967, Warwick Manufacturing Group (now WMG) was established in 1980 ...
[History](#) · [Campus](#) · [Academics](#) · [Research](#)

University of Warwick ★
 University in Coventry, England · 0.3 mi
[Website](#) [Directions](#)

The University of Warwick is a plate glass research university in Coventry, England. It was founded in 1965 as part of a government initiative to expand access to higher education. [Wikipedia](#)

Address: Coventry CV4 7AL
Acceptance rate: 14% (2014)
Total enrollment: 23,570 (2015)
[Suggest an edit](#)

Events
 Wed 25 Apr 19:45 John Robins - The Darkness of Robins

SCORING AS THE BASIS OF RANKED RETRIEVAL

- We wish to **return in order the documents most likely to be useful** to the searcher.
- **How can we rank-order** the documents in the collection **with respect to a query?**
- **Assign a score – say in $[0, 1]$ – to each document.**
- This score measures how well document and query “match”.

QUERY-DOCUMENT MATCHING SCORES

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



SCORING WITH THE JACCARD COEFFICIENT

JACCARD COEFFICIENT

- **Measures overlap of 2 sets A & B** (which can have different sizes).
 - $\text{jaccard}(A,B) = |A \cap B| / |A \cup B|$
 - 1 if $A=B$, 0 if $A \cap B = 0$ (always [0-1])
- User query (A) = University of Warwick
- Web page (B) = University of California Berkeley
- $|A \cap B| = |\text{University, of}| = 2$
- $|A \cup B| = |\text{University, of, Warwick, California, Berkeley}| = 5$

————→ $2/5 = 0.4$

JACCARD COEFFICIENT: SCORING EXAMPLE

- What is the query-document match score that the Jaccard coefficient computes for each of the three documents below?
 - Query: Netflix subscription UK.
 - Document 1: Watch TV with **Netflix** → $1/6 = 0.17$
 - Document 2: List of **UK** cities → $1/6 = 0.17$
 - Document 3: **UK** government → $1/4 = 0.25$ → we would rank it 1st!?

JACCARD COEFFICIENT: ISSUES FOR SCORING

- **Issues:**
 - We need a more sophisticated way of **normalising for length**.
 - It **doesn't consider term frequency** (number of occurrences of term).
 - How many times does Netflix appear in each document?
 - **Nor does it consider popularity of terms** (i.e. we could expect that rare terms are more informative).
 - Are Netflix, subscription and UK all equally important in my query?



WARWICK

TERM FREQUENCY WEIGHTING

RECAP: BINARY TERM-DOCUMENT MATRIX

- Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

TERM-DOCUMENT COUNT MATRICES

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

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

BAG OF WORDS MODEL

- The **bag of words** model **doesn't consider the ordering of words** in a document.
- **John is quicker than Mary** and **Mary is quicker than John** have the same vectors.
- 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 on.

TERM FREQUENCY: TF

- Term frequency $tf_{t,d}$ is the number of times that t occurs in d .
 - Can we use it to compute query-document match scores?

TERM FREQUENCY: TF

- $tf_{t,d}$ is a useful and meaningful **weight** for query-document **relevance scoring**.
- But:
 - Doc1: $tf_{t,d1} = 100$
Doc2: $tf_{t,d2} = 1$ } Doc1 certainly more relevant than Doc2,
but really 100 times more relevant?
 - Relevance should not increase proportionally with term frequency.

LOG-FREQUENCY WEIGHTING

- The log frequency weight of term t in d is:

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

- **Score for document-query pair: sum over terms t in both q & d :**

$$\text{score} = \sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})$$

- The score is 0 if none of the query terms is present in the document.

LOG-FREQUENCY WEIGHTING

$$\text{score} = \sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})$$

- Query = University of Warwick

$$\text{score} = (1 + \log \text{tf}_{\text{University},d}) + (1 + \log \text{tf}_{\text{of},d}) + (1 + \log \text{tf}_{\text{Warwick},d})$$

- Doc1: The University of Warwick is a university in Coventry

$$\text{score} = (1 + \log 2) + (1 + \log 1) + (1 + \log 1) = 3.301$$

- Doc2: University College London

$$\text{score} = (1 + \log 1) + 0 + 0 = 1$$

- Doc3: University university university university university university

$$\text{score} = (1 + \log 7) + 0 + 0 = 1.845$$



INVERSE DOCUMENT FREQUENCY WEIGHTING



WARWICK

IMPORTANCE OF QUERY TERMS

- So far we're considering that all query terms are equally important.
- i.e. in the query “University of Warwick”, keyword matches of:
 - University
 - of
 - Warwick

have all the same importance, but is it really fair?

DOCUMENT FREQUENCY

- **Intuition: Rare terms are more informative** than frequent terms.
 - e.g. “Warwick” (rare) more informative than “of” or “University” (both are more frequent)
- For the **query “University of Warwick”**: document containing **“Warwick” should be more relevant** than document containing **“of”**.
 - With term frequencies, we weren’t considering this.
 - We want **“Warwick” to have a higher weight**.
 - We can do this with DF, **document frequency**.

DOCUMENT FREQUENCY

- **df_t is the number of documents that contain t .**
- If we analyse our entire collection, we'd find that:
 - “of” occurs in almost every document (very high df)
 - “University” occurs in several documents (high df)
 - “Warwick” occurs in just a few (low df)

IDF WEIGHT

- df_t is the number of documents that contain t .
 - We need the inverse, i.e. low df , higher score = important keyword.
- We define the **idf (inverse document frequency)** of t by

$$idf_t = \log_{10} (N/df_t)$$

- N = number of documents in collection.
- df_t = number of documents in collection that contain t .
- We use $\log(N/df_t)$ instead of N/df_t to “dampen” the effect of idf .

IDF EXAMPLE

- Suppose $N = 1$ million
- Each term t in a collection has its own idf value

term	df_t	idf_t
cs918	1	6
nlp	100	4
computer	1,000	3
university	10,000	2
under	100,000	1
the	1,000,000	0

$$idf_t = \log_{10} (N/df_t)$$



WARWICK

TF-IDF WEIGHTING

TF-IDF WEIGHTING

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

$$w_{t,d} = (1 + \log \text{tf}_{t,d}) \times \log_{10}(N / \text{df}_t)$$

- **Best known weighting scheme** in information retrieval.
 - Increases with the number of occurrences within a document.
 - Increases with the rarity of the term in the collection.

FINAL RANKING OF DOCUMENTS FOR A QUERY

$$\text{Score}(q, d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

BINARY → COUNT → WEIGHT MATRIX

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

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



THE VECTOR SPACE RANKING MODEL

DOCUMENTS AS VECTORS

- Now we have a **$|V|$ -dimensional vector space**.
- **Documents** are points or **vectors in this space**.
- **Very high-dimensional**: tens of millions of dimensions when you apply this to a web search engine.
- These are **very sparse vectors** – most entries are zero.

QUERIES AS VECTORS

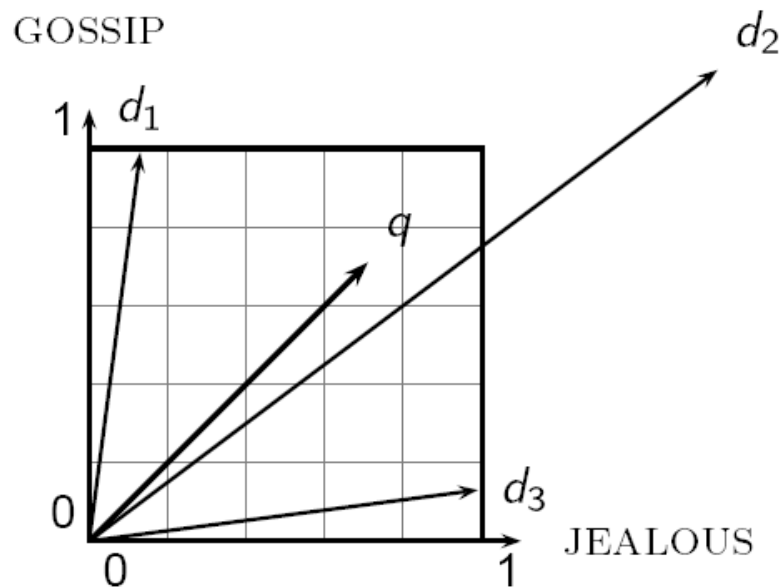
- **Key idea 1:** Do the same for queries: represent them as vectors in the space.
- **Key idea 2:** Rank documents according to their proximity to the query in this space.
- proximity = similarity of vectors \approx inverse of distance.

FORMALISING VECTOR SPACE PROXIMITY

- First cut: distance between two points
 - (= **distance between the end points of the two vectors**)
- **Euclidean distance is a bad idea**, because it is large for vectors of different lengths.
 - Remember document can be long, queries generally short.

WHY DISTANCE IS A BAD IDEA

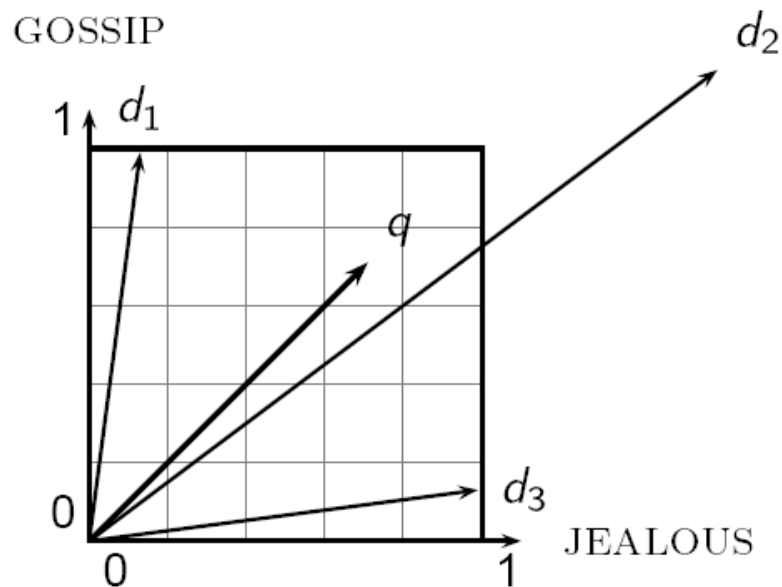
The Euclidean distance between \vec{q} and \vec{d}_2 is large even though the distribution of terms in the query \vec{q} and the distribution of terms in the document \vec{d}_2 are very similar.



SOLUTION: USE ANGLES

Use angles instead:

Similar documents, despite having different lengths, will be near-parallel, i.e. low angle between them.



USE ANGLE INSTEAD OF DISTANCE

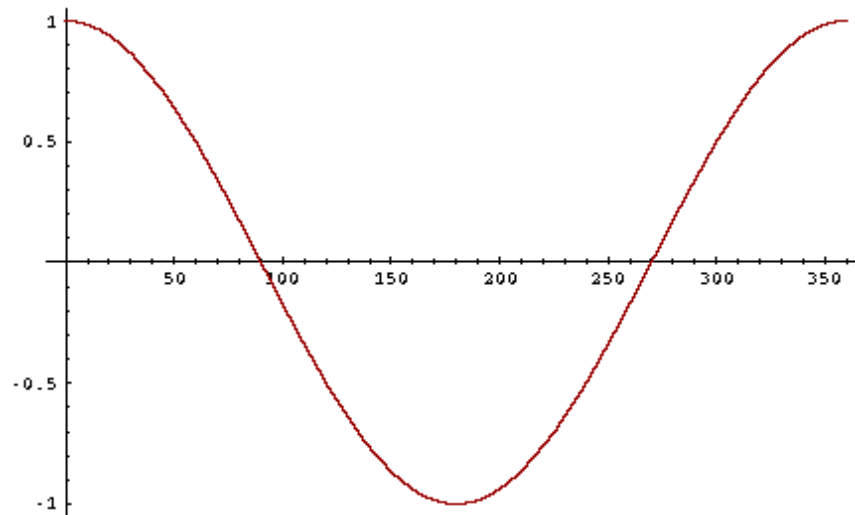
- Thought experiment: **take a document d and append it to itself**. Call this document d' .
- “Semantically” d and d' have the **same content, but the Euclidean distance** between them **can be large**.
- Instead, the **angle between the two documents is 0**, corresponding to maximal similarity.
- **Key idea:** Rank documents according to angle with query.

FROM ANGLES TO COSINES

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

FROM ANGLES TO COSINES

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



LENGTH NORMALISATION

- A **vector can be (length-) normalised by dividing each of its components by its length** – for this we use the L2 norm:

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

- Dividing a vector by its L2 norm makes it a unit (length) vector (on surface of unit hypersphere).
- Effect on the two documents d and d' (d appended to itself) from earlier slide: they have identical vectors after length-normalisation.
- Long and short documents now have comparable weights.

COSINE(QUERY, DOCUMENT)

Dot product

Unit vectors

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

- 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
- $\cos(q, d)$ is the cosine similarity of q and d ... or,
- equivalently, the cosine of the angle between q and d .

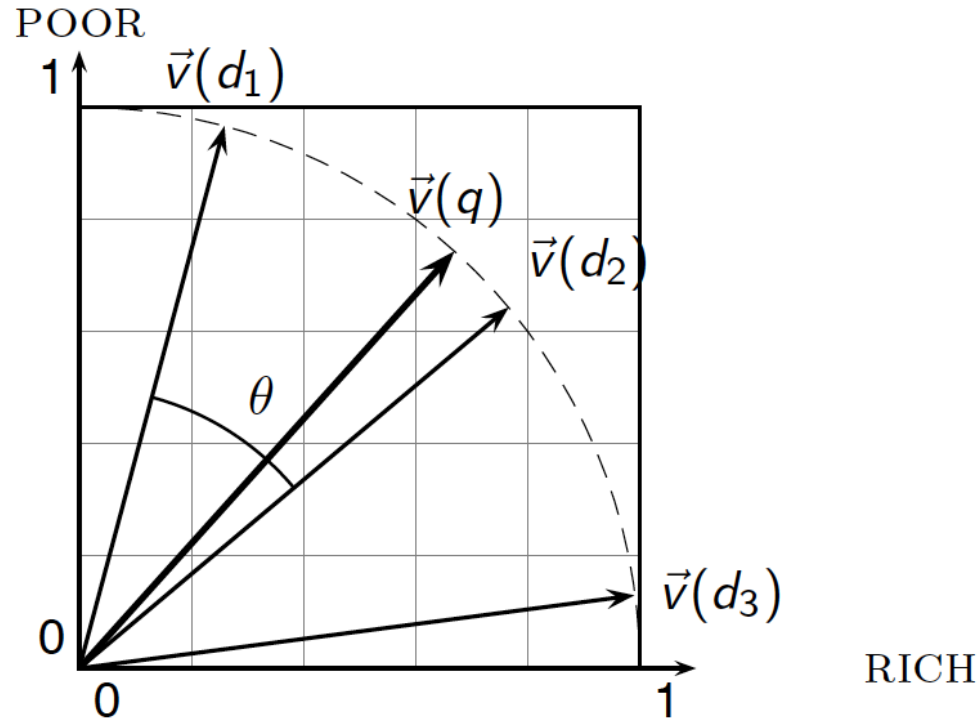
COSINE FOR LENGTH-NORMALISED VECTORS

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

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

for q, d length-normalised.

COSINE SIMILARITY ILLUSTRATED



COSINE SIMILARITY AMONG 3 DOCUMENTS

How similar are
the novels

SaS: *Sense and
Sensibility*

PaP: *Pride and
Prejudice*, and

WH: *Wuthering
Heights*?

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

Term frequencies (counts)

COSINE SIMILARITY AMONG 3 DOCUMENTS

Log frequency weighting

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

After length normalisation

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

$$\cos(\text{SaS}, \text{PaP}) \approx 0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0 \approx 0.94$$

$$\cos(\text{SaS}, \text{WH}) \approx 0.79$$

$$\cos(\text{PaP}, \text{WH}) \approx 0.69$$



WARWICK

CALCULATING TF-IDF COSINE SCORES IN AN IR SYSTEM

TF-IDF WEIGHTING HAS MANY VARIANTS

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

WEIGHTING MAY DIFFER IN QUERIES AND DOCS

- Many search engines use **different weightings for queries vs. documents**.
- **SMART Notation:** denotes the **combination in use** in an engine, with the notation **ddd.qqq**, using the acronyms from the previous table.
- A very standard weighting scheme is: **Inc.ltc**
 - Doc: logarithmic tf (l), no idf (n) and cosine normalisation (c)
 - Query: logarithmic tf (l), idf (t), cosine normalisation (c)

COMPUTING COSINE SCORES

COSINESCORE(q)

```
1  float Scores[ $N$ ] = 0
2  float Length[ $N$ ]
3  for each query term  $t$ 
4  do calculate  $w_{t,q}$  and fetch postings list for  $t$ 
5      for each pair( $d, tf_{t,d}$ ) in postings list
6      do Scores[ $d$ ] +=  $w_{t,d} \times w_{t,q}$ 
7  Read the array Length
8  for each  $d$ 
9  do Scores[ $d$ ] = Scores[ $d$ ]/Length[ $d$ ]
10 return Top  $K$  components of Scores[]
```

LEARNING TO RANK: DATASETS

- The ranking problem is often referred to as “**Learning to Rank**” in Information Retrieval.
i.e. use machine learning to rank documents.
- To tackle the task, we need datasets **not only with binary relevance judgements**.
Recall: each document annotated as relevant (R) or non-relevant (NR) for a particular query.
- Instead, **we need scores**, e.g. 0 (irrelevant) to 4 (perfectly relevant).

LEARNING TO RANK: DATASETS

- Microsoft Learning to Rank Datasets:
<https://www.microsoft.com/en-us/research/project/mslr/>

We released two large scale datasets for research on learning to rank: MSLR-WEB30k with more than 30,000 queries and a random sampling of it MSLR-WEB10K with 10,000 queries.

Dataset Descriptions

The datasets are machine learning data, in which queries and urls are represented by IDs. The datasets consist of feature vectors extracted from query-url pairs along with relevance judgment labels:

- (1) The relevance judgments are obtained from a retired labeling set of a commercial web search engine ([Microsoft Bing](#)), which take 5 values from 0 (irrelevant) to 4 (perfectly relevant).
- (2) The features are basically extracted by us, and are those widely used in the research community.

In the data files, each row corresponds to a query-url pair. The first column is relevance label of the pair, the second column is query id, and the following columns are features. The larger value the relevance label has, the more relevant the query-url pair is. A query-url pair is represented by a 136-dimensional feature vector.

Below are two rows from MSLR-WEB10K dataset:

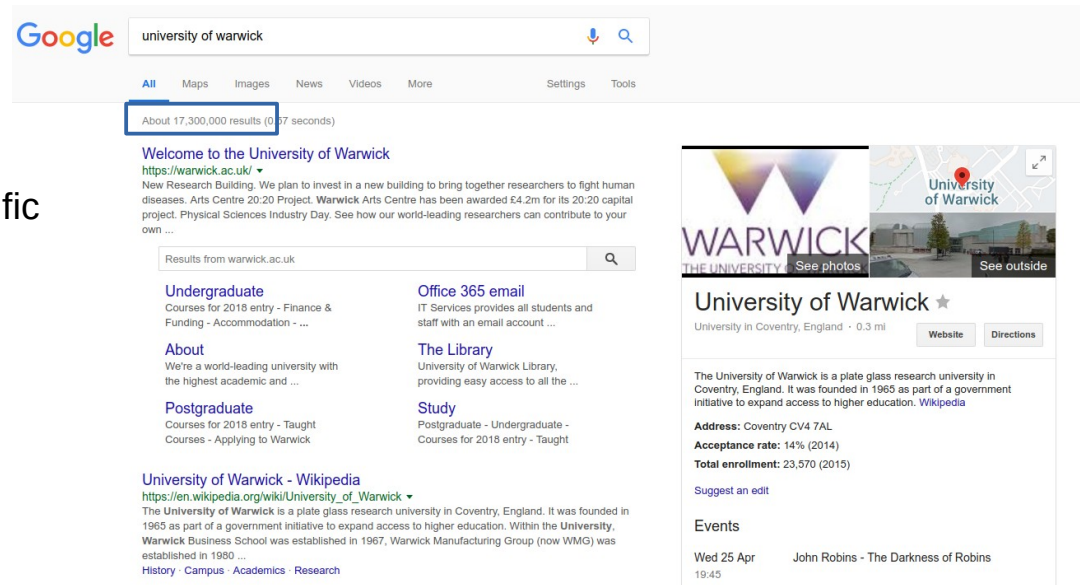
```
=====
0 qid:1 1:3 2:0 3:2 4:2 ... 135:0 136:0
2 qid:1 1:3 2:3 3:0 4:0 ... 135:0 136:0
=====
```

SUMMARY: VECTOR SPACE RANKING

- Represent **query as a weighted tf-idf vector**.
- Use **positional index to retrieve documents matching query**:
 - <term, number of docs containing term;
 - doc1: position1, position2 ... ;
 - doc2: position1, position2 ... ;
 - etc.
- Represent each **document as a weighted tf-idf vector**.
- Compute the **cosine similarity** score for the **query vector and each document vector**.
- **Rank** documents with respect to the query **by score**.
- **Return the top K** (e.g., $K = 10$) to the user.

SUMMARY: VECTOR SPACE RANKING

- OK, but:
 - We're assuming the user query is perfect, what if it wasn't specific enough?
 - Also, are we going to compute similarity for a query wrt 17 million documents?!?!?
 - We need to do better... (in the next two lectures!)



ASSOCIATED READING

- Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to information retrieval (Vol. 1, No. 1, p. 496). Cambridge: Cambridge university press. **Chapter 6.**

<https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf>