WARWICK

CS918: LECTURE 14

Efficiency and Evaluation of Information Retrieval

Arkaitz Zubiaga, 19th November, 2018

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

WARWICK

- 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: TF-IDF WEIGHTING HAS MANY VARIANTS

WARWICK

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

RECAP: COMPUTING COSINE SCORES

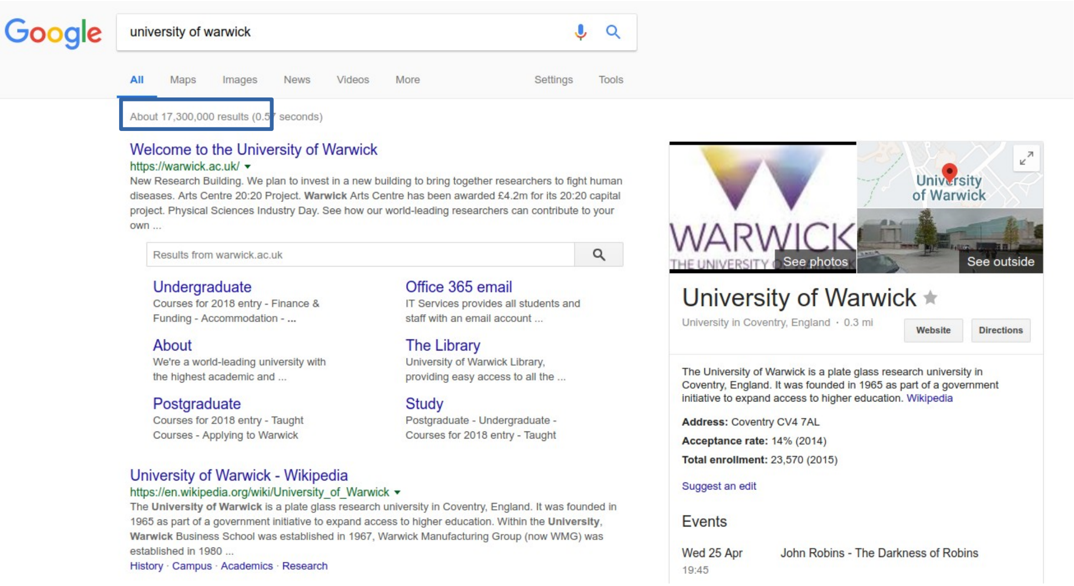
CosineScore(q)

float $Scores[N] = 0$
float $Length[N]$
for each query term t
do calculate $w_{t,q}$ and fetch postings list for t
  for each pair(d, $tf_{t,d}$) in postings list
  do $Scores[d] += w_{t,d} \times w_{t,q}$
Read the array $Length$
for each d
do $Scores[d] = Scores[d]/Length[d]$
return Top K components of $Scores[]$

RECAP: VECTOR SPACE RANKING

- OK, but… are we going to compute similarity for a query wrt 17 million documents?!?!

- We need to be more efficient…



QUESTION ABOUT COSINE-BASED RANKING

WARWICK

- Somebody asked:

  What if I create an HTML page with just “University of Warwick” in it?

  Will that rank 1st for a query “University of Warwick”?

QUESTION ABOUT COSINE-BASED RANKING

- If we only rely on cosine similarity:

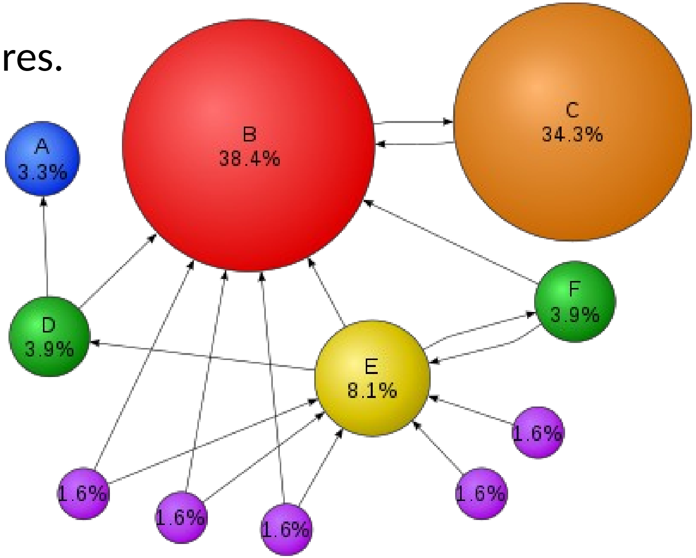
  doc and query = “University of Warwick”
  will lead to max similarity (cosine = 1)

- Maybe fine for corporate search engine (we have a control of the docs).
- A (web) search engine needs more:
  - Adversarial Information Retrieval (classify spam, those trying to deceive us).
  - We can use more features, beyond text (e.g. PageRank).

INTUITION OF PAGERANK

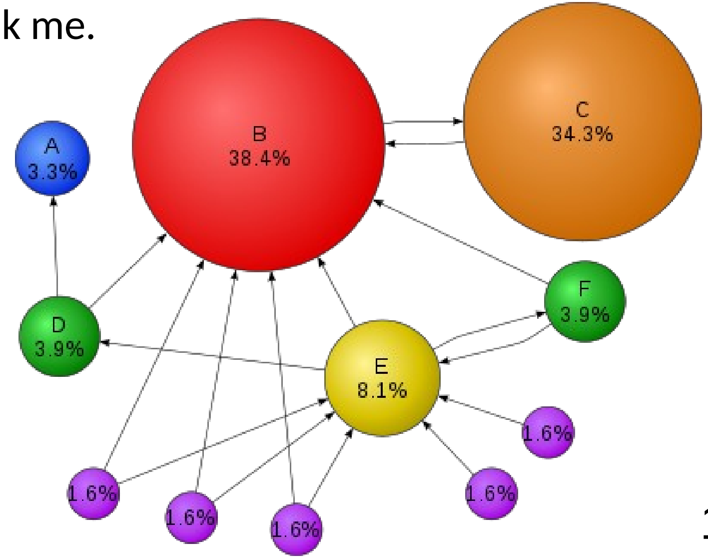
- **Recursively weight web pages based on links**:
  site is important if other important sites have links to it.

- I can then combine these weights with cosine scores.



INTUITION OF PAGERANK

- If I create a page with just "University of Warwick":
  - I will most likely be one of those purple sites.
  - Red and orange (reputable sites) will not link me.



- NB: more on PageRank on Data Analytics and/or Data Mining, here we focus on textual IR.

LECTURE 14: CONTENTS

- How to speed up vector speed ranking.
- Putting together a complete search system.
- Evaluation of Information Retrieval.
    - MAP and NDCG.

SPEEDING UP VECTOR SPACE RANKING

EFFICIENT COSINE RANKING

WARWICK

- Find **k documents** in the collection with **largest query-document cosines**.
  - e.g. k=10

- We need two things to **achieve efficient ranking**:
  - Compute **cosine** for each query-doc pair **efficiently**.
  - Choose **K largest** cosines **efficiently.**
    - ideally without computing cosines for all docs in collection!

EFFICIENT COSINE RANKING

- Basically, we're trying to solve the **K-nearest neighbour problem** for a query vector.

- This is **challenging for high-dimensional spaces** with many data points (documents).
  - **Brevity of queries** helps make the task **easier**.

BOTTLENECK

- Primary **bottleneck** is in **cosine computation**.

- Can we avoid all this computation?
  - Yes, but may sometimes get it wrong.
  - a doc not in top K may creep into the list of K output docs.
  - Is this such a bad thing?

COSINE SIMILARITY IS ONLY A PROXY

- Cosine matches docs to query.

- Thus cosine is anyway a proxy for similarity.

- Let’s try to **get a list of K docs “close” to the actual top K** by cosine measure, **not necessarily the closest K**.

GENERIC APPROACH

- Find a **set A of contenders**, with K < |A| << N
- A does **not necessarily contain the top K**, but has **many** docs from **among the top K**.
- Return the **top K docs in A**.

- Will look next at several schemes following this approach.

INDEX ELIMINATION

- Basic algorithm; cosine computation algorithm **only** considers docs **containing at least one query term**.

- Take this further:
  - **Only** consider **high-idf query terms**.
  - **Only** consider docs containing **many query terms.**
  - **Champion lists.**

- And of course, **for frequent queries, cache results.**

HIGH-IDF QUERY TERMS ONLY

WARWICK

- For a query such as: **weather in the uk**.
  - Only accumulate scores from **weather** and **uk**
- **Intuition:** “in” and “the” contribute little to the scores and so don’t alter rank-ordering much.
  - Postings of low-idf terms (in, the) have many docs, hence we eliminate them from set A of contenders.

DOCS CONTAINING MANY QUERY TERMS

- For multi-term queries, only compute scores for **docs containing several of the query terms.**
  - e.g. at least 3 out of 4.
- Used by some web search engines (e.g. early Google).
- Easy to implement.

EXAMPLE: 3 OR MORE OF 4 QUERY TERMS

- Query: **Antony Brutus Caesar Calpurnia**

Antony	3	4	8	16	32	64	128	
Brutus	2	4	8	16	32	64	128	
Caesar	1	2	3	5	8	13	21	34
Calpurnia	13	16	32					

- We will **only compute** cosine scores for docs **8, 16 and 32**.

CHAMPION LISTS

WARWICK

- For each dictionary term ***t***, precompute the ***r*** docs of highest weight in ***t***'s postings.
  - This is the **champion list for t**.
  - When user searches for **t**, consider **only docs in champion list**.

- Note that ***r*** has to be chosen at **index building time**.

CHAMPION LISTS: EXAMPLE AND LIMITATION

- Query: “university london”
  - Both terms are frequent: university (5M results), london (10M results).
  - Precompute (say r = 500):
    - 500 top results for “university”.
    - 500 top results for “london”.

  - At query time, only look at those 1,000 results.

- **Limitation:** top K results for query “t_1 t_2” may not be in champion lists of t_1 or t_2.
  - We need to choose **r** carefully.

CACHE RESULTS FOR FREQUENT QUERIES

WARWICK

- If I have users issuing the search query “London” every 10 seconds.
  - I can store a pregenerated ranking for “London”.
  - Update cached results every X hours.

- I only need to bother computing cosine scores for less frequent queries.

STATIC QUALITY SCORES

- We want **top-ranked documents** to be both **relevant** and **authoritative**.
  - **Relevance** is being captured by **cosine scores**.

  - **Authority** is typically a **query-independent property**, e.g.:
    - Document is linked to from top newspapers.
    - Scientific paper with many citations.
    - Many mentions in social media.
    - PageRank.

MODELLING AUTHORITY

- Assign a **query-independent quality score** in **[0,1]** to each **document d**.
    - We call this g(d).

- Basically, we will **normalise** {citation count, link count, social media mentions, pagerank,…}

NET SCORE

- The **net score** is a simple, total score combining **relevance and authority.**
  - **net-score(q,d) = g(d) + cosine(q,d)**

- Now we seek the top K docs by net score

PARAMETRIC AND ZONE INDICES

- So far, index = sequence of terms.
- We **may also want to index metadata**, such as:
  - Author
  - Title
  - Language
  - Format (e.g. query: **warwick filetype:pdf**)
  - URL (e.g. query: **NLP site:warwick.ac.uk**)

ZONE

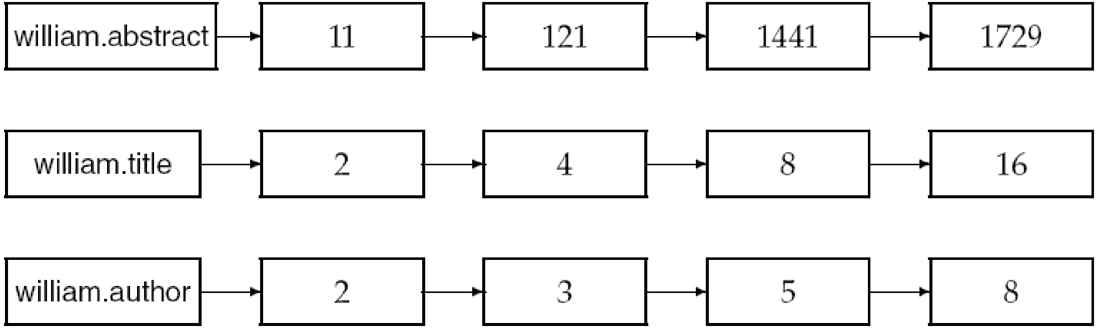
- A **zone** is a region of the doc that can contain an arbitrary amount of text, e.g. title, abstract, author, publication date, URL.

- We can also build **inverted indices on zones**.
  - e.g. search for docs with **UK in the title**, hosted in **theguardian.com** and **published in 2016**.

EXAMPLE OF ZONE INDICES

WARWICK

- We can encode zones in separate dictionaries:



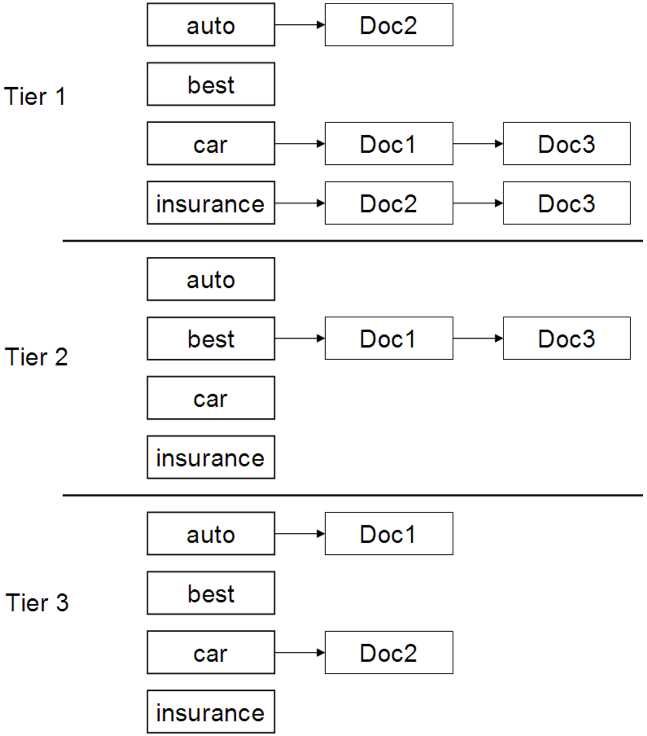
- Or as part of the postings:

TIERED INDICES

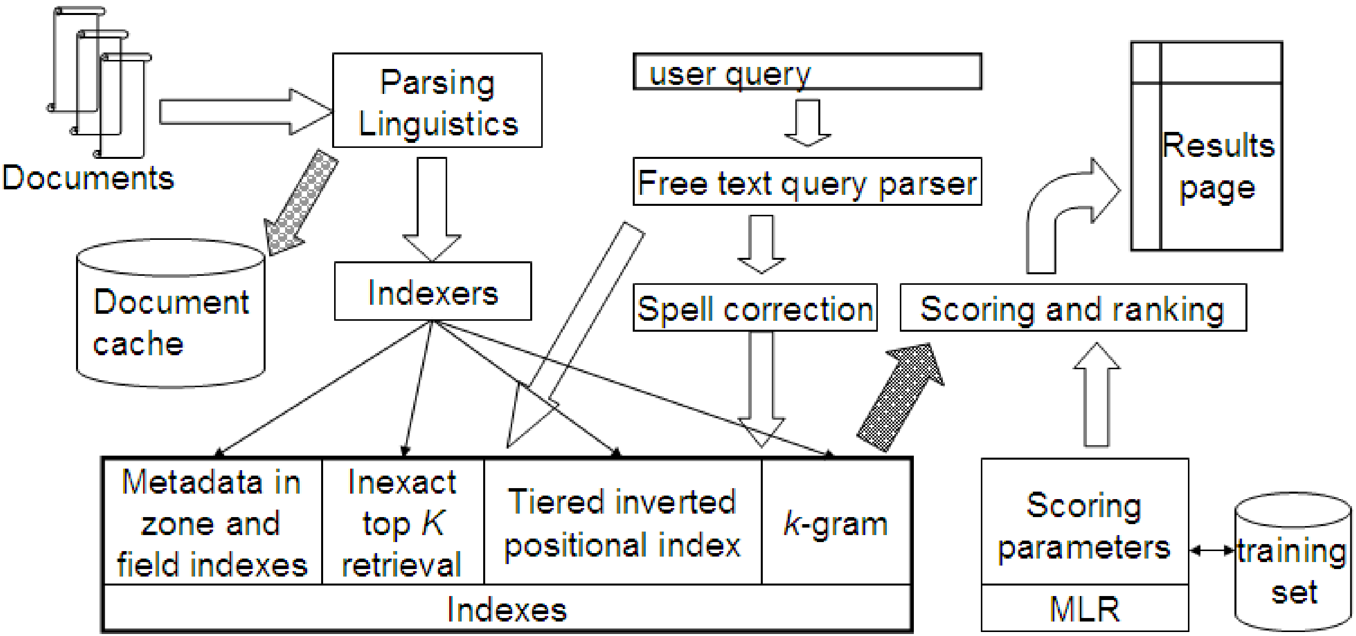
- Break postings up into a **hierarchy of lists**:
  - **Tier 1:** Most important
  - …
  - **Tier n:** Least important
- Can be done by **g(d) or another measure**.

- At **query time use top tier**. If it yields **fewer than K docs, use lower tier**.

EXAMPLE OF TIERED INDEX



BUILDING AN ENTIRE SEARCH ENGINE



EVALUATING SEARCH ENGINES

EVALUATING AN IR SYSTEM

- An **information need is translated into a query.**
- **Relevance of results is assessed relative to the information need** not the query, e.g.:
  - **Information need:** I'm looking for information on whether eating chocolate is an effective way to sleep better.
  - **Query:** chocolate better sleep effective
- We want to **evaluate whether the results address the information need**, not whether it has the words in the query.

EVALUATING RANKED RESULTS

- **Evaluation** of a result set, having:
    - a **benchmark document collection.**
    - a **benchmark set of queries**.
    - **manual judgements/annotations** of whether **documents are relevant (R) to queries** (or **not relevant, NR**).

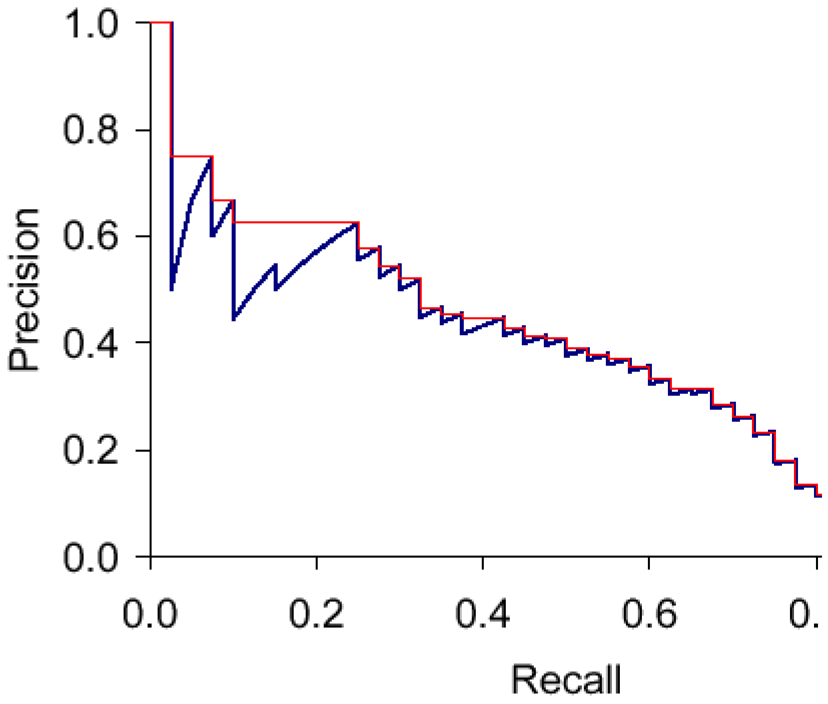
- Note: manual judgements of all query-doc pairs as R/NR is often unfeasible (e.g. 1 million docs, 1000 queries → 1 billion pairs!).
    - **Judgements usually done only for a sample** (e.g. subset containing query words)

RANKED RESULTS: VISUALISING PERFORMANCE

WARWICK

- The system can return any number of results.
- **For every rank k, we can compute precision** (how many of those k are relevant?) **and recall** (how many of those actually relevant did we output?).
- By taking those precision/recall scores **over different k positions**, we can **produce a precision-recall curve**.

A PRECISION-RECALL CURVE



PRECISION-RECALL CURVE

- Why does the P-R curve look like this?
  - Say we return 20 results, our collection has 5 relevant documents.

    1: R, 2: NR, 3: NR, 4: R, 5: NR, 6: NR, 7: NR, 8: R,…

  - As we go down in the ranking:
    - Precision decreases (we include mistakes).
      - But occasionally goes up (we included a good result).
    - Recall/coverage increases (higher % of actual R docs included).

QUANTIFYING EVALUATION

- Precision-Recall (P-R) curves are good for visualisation, but we often want a single score to quantify performance.

PRECISION AT K

- A **simple approach** is precision@k, i.e. ratio of relevant items within the top k results.
  - **Good if ranking is not important**, however these have the same precision@5 of 0.4:
    (regardless of relevant items being the top 2 or the bottom 2 items)
    - **1: R, 2: R, 3: NR, 4: NR, 5: NR**
    - **1: NR, 2: NR, 3: NR, 4: R, 5: R**

- **Better metrics** when **ranking** is important: **MAP, NDCG**.

MAP: MEAN AVERAGE PRECISION

- I issue Q queries in my system.
  - I get an AP (average precision) for each query.
  - I get the mean of all Q AP’s.

- Average precision: up to position k, compute the **average Precision@K for all positions {1..k}**, then get the **mean of all these k scores**.

- It **weighs performance on top positions higher** (precision on top 1 item is being counted in every iteration {1..k}).

MAP: EXAMPLE

- Running example, 1 IR system, 2 different queries:
  - **1: R, 2: R, 3: NR, 4: NR, 5: NR**
  - **1: NR, 2: NR, 3: NR, 4: R, 5: R**

MAP: EXAMPLE

WARWICK

- Running example, 1 IR system, 2 different queries:
  - **1: R, 2: R, 3: NR, 4: NR, 5: NR**
  - **1: NR, 2: NR, 3: NR, 4: R, 5: R**

- Query 1:
  - Precision@1: 1/1 = 1
  - Precision@2: 2/2 = 1
  - Precision@3: 2/3 = 0.67
  - Precision@4: 2/4 = 0.5
  - Precision@5: 2/5 = 0.4

AP_{Q1} = (1+1+0.67+0.5+0.4)/5 = 0.713

MAP: EXAMPLE

WARWICK

- Running example, 1 IR system, 2 different queries:
  - **1: R, 2: R, 3: NR, 4: NR, 5: NR**
  - **1: NR, 2: NR, 3: NR, 4: R, 5: R**

- Query 2:
  - Precision@1: 0/1 = 0
  - Precision@2: 0/2 = 0
  - Precision@3: 0/3 = 0
  - Precision@4: 1/4 = 0.25
  - Precision@5: 2/5 = 0.4

AP_{Q2} = (0+0+0+0.25+0.4)/5 = 0.13

MAP: EXAMPLE

WARWICK

- Running example, 1 IR system, 2 different queries:
  - **1: R, 2: R, 3: NR, 4: NR, 5: NR**
  - **1: NR, 2: NR, 3: NR, 4: R, 5: R**

- $AP_{Q1} = 0.713$
- $AP_{Q2} = 0.13$

- $MAP@5 = (AP_{Q1} + AP_{Q2}) / 2 = (0.713 + 0.13) / 2 = 0.422$

NDCG: NORMALISED DISCOUNTED CUMULATIVE GAIN

- NDCG: **graded degrees of relevance, normalised wrt ideal/perfect output**.
- We first compute DCG_k: $$\mathrm{DCG_k} = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)}$$
- We then normalise it to get $NDCG_k$: $$\mathrm{nDCG_k} = \frac{DCG_k}{IDCG_k}$$
- where $IDCG_k = DCG_k$ for the ideal system (all k elements are relevant)

NDCG

- Running example, 1 IR system, 2 different queries:
  - **1: R, 2: R, 3: NR, 4: NR, 5: NR**
  - **1: NR, 2: NR, 3: NR, 4: R, 5: R**

- Query 1:
  - $DCG_{5,Q1} = 1 / \log_2(2) + 1 / \log_2(3) + 0 + 0 + 0 = 1.631$
- Query 2:
  - $DCG_{5,Q2} = 0 + 0 + 0 + 1 / \log_2(5) + 1 / \log_2(6) = 0.818$
- $IDCG_5 = 1 / \log_2(2) + 1 / \log_2(3) + 1 / \log_2(4) + 1 / \log_2(5) + 1 / \log_2(6) = 2.948$

NDCG

- Running example, 1 IR system, 2 different queries:
  - **1: R, 2: R, 3: NR, 4: NR, 5: NR**
  - **1: NR, 2: NR, 3: NR, 4: R, 5: R**

- Query 1:
  - $NDCG_{5,Q1} = DCG_{5,Q1} / IDCG_5 = 1.631 / 2.948 = 0.553$
- Query 2:
  - $NDCG_{5,Q2} = DCG_{5,Q2} / IDCG_5 = 0.818 / 2.948 = 0.277$
- **Global $NDCG_5$** = (0.553 + 0.277) / 2 = **0.415**

NDCG

- What's good about NDCG is it can consider different levels of relevance (e.g. 0-5), not just R vs NR:
  - **1: 4, 2: 5, 3: 2, 4: 0, 5: 2**
  - **1: 2, 2: 4, 3: 0, 4: 1, 5: 5**

NDCG

- Weighted example:
  - **1: 4, 2: 5, 3: 2, 4: 0, 5: 2**
  - **1: 2, 2: 4, 3: 0, 4: 1, 5: 5**

- Query 1:
  - $DCG_{5,Q1} = 4 / \log_2(2) + 5 / \log_2(3) + 2 / \log_2(4) + 0 + 2 / \log_2(6) = 8.928$
- Query 2:
  - $DCG_{5,Q2} = 2 / \log_2(2) + 4 / \log_2(3) + 0 + 1 / \log_2(5) + 5 / \log_2(6) = 6.889$
- $ICDG_5 = 5 / \log_2(2) + 5 / \log_2(3) + 5 / \log_2(4) + 5 / \log_2(5) + 5 / \log_2(6) = 15.472$

NDCG

- Weighted example:
  - **1: 4, 2: 5, 3: 2, 4: 0, 5: 2**
  - **1: 2, 2: 4, 3: 0, 4: 1, 5: 5**

- Query 1:
  - $NDCG_{5,Q1} = DCG_{5,Q1} / IDCG_5 = 8.928 / 15.472 = 0.577$
- Query 2:
  - $NDCG_{5,Q2} = DCG_{5,Q2} / IDCG_5 = 6.889 / 15.472 = 0.445$
- **Global NCDG$_5$** $= (0.577 + 0.445) / 2 = 0.511$

MAP vs NDCG

- Both consider **top results are more important**.
- **MAP is simple**, easy to understand, **widely used**.
- NDCG can consider different levels of relevance.
    - Instead of **just relevant (1) vs non-relevant (0)**.
    - NDCG can take **relevance scores from e.g. 0 to 5**.
        - Especially used in these cases.
    - **MAP can only handle 0's and 1's**.

MAP and NDCG

WARWICK

- Note: MAP and NDCG **not only** used in **information retrieval**.
    - Useful for **any ranking problem**.
        - e.g. to evaluate system that ranks universities.
        - e.g. to evaluate system that predicts Premier League table.
        - e.g. to evaluate recommender systems (forthcoming lecture!)

RESOURCES

- Text REtrieval Conference (TREC) datasets: http://trec.nist.gov/data.html
  - Including:
    - Relevance judgements for web search: http://trec.nist.gov/data/webmain.html
    - Relevance judgements for Twitter search: http://trec.nist.gov/data/microblog.html

ASSOCIATED READING

WARWICK

- Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to information retrieval (Vol. 1, No. 1, p. 496). Cambridge: Cambridge university press. **Chapters 6-8**. https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf