

universität freiburg

Databases and Information Systems WS 25/26

Lecture 2: Ranking and Evaluation

October 21, 2025

Prof. Dr. Hannah Bast
Department of Computer Science
University of Freiburg

Overview of this lecture

■ Organizational

- Your experiences with ES1 Inverted index
- Your tutor Feedback and timeslots

■ Contents

- Ranking tf.idf and BM25, other tricks
- Evaluation Ground truth, Precision, AP, MAP, nDCG, BPref, Overfitting

- **ES2:** Implement BM25, tune your search engine using a training benchmark, then evaluate on a test benchmark

There will be a small competition for your enjoyment and motivation (in the form of a table on the Wiki)

Experiences with ES1 1/2

■ Excerpts from your feedback [somewhat representative]

"I really liked the lecture and the exercise sheet"

"Nice, gentle introduction back into study mode after the break"

"The first lecture was easy for me, nothing really complicated"

"Ich finde es toll, dass die Übungen optional sind"

"I like the lecture and especially that it is uploaded on YouTube"

"I only want the points and a Haiku rating my performance"

"The coding part was a bit boring to follow"

"War für mich recht schwierig wieder reinzukommen"

"Sometimes I wished the first exercise sheet was more difficult so that I would not start every semester illusionized and only realized at the end that I am stupid or whatever"

Experiences with ES1 2/2

■ Queries from your feedback

– Queries that worked well:

oppenheimer	specific word, popular movie
indiana jones	specific enough
gothic horror burton	specific words and combination

– Queries that didn't work as expected:

bomb	expected Oppenheimer, got Dr. Strangelove
toy story	first Finding Nemo, which mentions Toy Story
british western	matches Westerns that won British awards
iron man	first Avengers → makes sense but unexpected
12 years a slave	"12" ignored, rest unspecific, order ignored

Your tutor

■ Feedback

- If you have asked for feedback in your [experiences.md](#), you will find a file [feedback-tutor.md](#) in your repository by Friday

You need to do **svn update** to get it

■ Individual help

- If you have problems or issues that cannot be solved via the forum, you can book a timeslot with your tutor

See link when you log into Daphne

You can see who your tutor is by the end of today (Oct 21) or tomorrow (Oct 22) at the very latest + note that your tutor may change over the course of the semester

A glimpse of ES2

- Hands-on practice with rankings and evaluations

- **Exercise 1:** BM25 scores for better ranking

- BM25 is a formula for word-in-record scores → slides 14 – 17

- Computing these scores is an extension of ES1

- **Exercise 2:** Write an automatic evaluation script

- Automatically test how well your little "search engine" performs on a given benchmark (queries with their results)

- **Exercise 3:** Set good parameters for your BM25 scores

- Use the training set we give you for parameter tuning

- Then evaluate once on the test set

- That's also how you do it in "real" research later

■ Motivation

- A keyword query is a formulation of a search desire
- Not all documents that contain all or some of the keywords are also "relevant" for the search desire

You already saw this in ES1, and more examples in ES2

- We want the most "relevant" documents first

For web search, where we can have millions of hits, this is absolutely crucial for the usefulness of the engine

But even for the [Uni Freiburg web pages](#), it's non-trivial

- **Problem:** how to measure how "relevant" a document is?

Ranking 2/14

■ Basic Idea

- In the inverted lists, for each **doc id** also have a **score**, which captures how much the document is "about" that word

university 17 0.5 , 53 0.2 , 97 0.3 , 127 0.8

freiburg 23 0.1 , 34 0.8 , 53 0.1 , 127 0.7

- While merging, **aggregate** the scores, then **sort** by score

MERGED 17 0.5 , 23 0.1 , 34 0.8 , 53 0.3 , 97 0.3 , 127 1.5

SORTED 127 1.5 , 34 0.8 , 17 0.5 , 53 0.3 , 97 0.3 , 23 0.1

- The entries in the list are referred to as **postings**

Above, it's only doc id and score, but a posting can also contain more information, e.g. the position of a word

for now: just SUM

■ Getting the top-k results

- A full sort of the result list takes time $\Theta(n \cdot \log n)$, where n is the number of postings in the list
- Typically only the top- k hits need to be displayed
- Then a **partial sort** is sufficient: get the k largest elements, for a given k

Can be computed in time $\Theta(n + k \cdot \log k)$

k rounds of HeapSort yield time $\Theta(n + k \cdot \log n)$

For constant k these are both $\Theta(n)$

For ES2, you can ignore this issue

Ranking 4/14

■ Meaningful scores

- How do we precompute good **scores**

university 17 0.5 , 53 0.2 , 97 0.3 , 127 0.8

freiburg 23 0.1 , 34 0.8 , 53 0.1 , 127 0.7

- **Goal:** the score for the posting for doc D_i in the inverted list for word w should reflect the **relevance** of w in D_i

In particular, the larger the score, the more relevant

- **Problem:** relevance is somewhat subjective

But it has to be done somehow anyway !

Ranking 5/14

■ Term frequency (**tf**)

- The number of times a word occurs in a document
- **Problem:** some words are frequent in many documents, regardless of the content

university	...	57	5	,	123	2	, ...
of	...	57	14	,	123	23	, ...
freiburg	...	57	3	,	123	1	, ...
SCORE SUM	...	57	22	,	123	26	, ...

- A word like "of" should not count much for relevance

Some of you observed that already while trying out queries for ES1

Ranking 6/14

■ Document frequency (**df**)

- The number of documents containing a particular word

$$\mathbf{df}_{\text{university}} = 16.384, \mathbf{df}_{\text{of}} = 524.288, \mathbf{df}_{\text{freiburg}} = 1.024$$

(Handwritten: $= 2^{14}$, $= 2^{19}$, $= 2^{10}$)

For simplicity, numbers are powers of 2, see below why

- Inverse document frequency (**idf**)

$$\mathbf{idf} = \log_2 (N / \mathbf{df}) \quad N = \text{total number of documents}$$

For the example **df** scores above and $N = 1.048.576 = 2^{20}$

(Handwritten: $= \log_2 (2^{20} / 2^{14})$...)

$$\mathbf{idf}_{\text{university}} = 6, \mathbf{idf}_{\text{of}} = 1, \mathbf{idf}_{\text{freiburg}} = 10$$

Note: The more frequent a word, the smaller the **idf**

Without the **log₂** small differences in the value of **df** would have too much of an effect

(Handwritten notes: ~~$\log_2 \frac{N}{2} \Rightarrow \log_2 1 = 0$~~ , $\log_2 N \Rightarrow \log_2 2 = 1$, $\log_2 1 = 0$, $\log_2 2 = 1$, $\log_2 4 = 2$, $\log_2 8 = 3$, $\log_2 16 = 4$, $\log_2 32 = 5$, $\log_2 64 = 6$, $\log_2 128 = 7$, $\log_2 256 = 8$, $\log_2 512 = 9$, $\log_2 1024 = 10$, $\log_2 2048 = 11$, $\log_2 4096 = 12$, $\log_2 8192 = 13$, $\log_2 16384 = 14$, $\log_2 32768 = 15$, $\log_2 65536 = 16$, $\log_2 131072 = 17$, $\log_2 262144 = 18$, $\log_2 524288 = 19$, $\log_2 1048576 = 20$, $\log_2 2097152 = 21$, $\log_2 4194304 = 22$, $\log_2 8388608 = 23$, $\log_2 16777216 = 24$, $\log_2 33554432 = 25$, $\log_2 67108864 = 26$, $\log_2 134217728 = 27$, $\log_2 268435456 = 28$, $\log_2 536870912 = 29$, $\log_2 1073741824 = 30$, $\log_2 2147483648 = 31$, $\log_2 4294967296 = 32$, $\log_2 8589934592 = 33$, $\log_2 17179869184 = 34$, $\log_2 34359738368 = 35$, $\log_2 68719476736 = 36$, $\log_2 137438953472 = 37$, $\log_2 274877906944 = 38$, $\log_2 549755813888 = 39$, $\log_2 1099511627776 = 40$, $\log_2 2199023255552 = 41$, $\log_2 4398046511104 = 42$, $\log_2 8796093022208 = 43$, $\log_2 17592186044416 = 44$, $\log_2 35184372088832 = 45$, $\log_2 70368744177664 = 46$, $\log_2 140737488355328 = 47$, $\log_2 281474976710656 = 48$, $\log_2 562949953421312 = 49$, $\log_2 1125899906842624 = 50$, $\log_2 2251799813685248 = 51$, $\log_2 4503599627370496 = 52$, $\log_2 9007199254740992 = 53$, $\log_2 18014398509481984 = 54$, $\log_2 36028797018963968 = 55$, $\log_2 72057594037927936 = 56$, $\log_2 144115188075855872 = 57$, $\log_2 288230376151711744 = 58$, $\log_2 576460752303423488 = 59$, $\log_2 1152921504606846976 = 60$, $\log_2 2305843009213693952 = 61$, $\log_2 4611686018427387904 = 62$, $\log_2 9223372036854775808 = 63$, $\log_2 18446744073709551616 = 64$, $\log_2 36893488147419103232 = 65$, $\log_2 73786976294838206464 = 66$, $\log_2 147573952589676412928 = 67$, $\log_2 295147905179352825856 = 68$, $\log_2 590295810358705651712 = 69$, $\log_2 1180591620717411303424 = 70$, $\log_2 2361183241434822606848 = 71$, $\log_2 4722366482869645213696 = 72$, $\log_2 9444732965739290427392 = 73$, $\log_2 18889465931478580854784 = 74$, $\log_2 37778931862957161709568 = 75$, $\log_2 75557863725914323419136 = 76$, $\log_2 151115727451828646838272 = 77$, $\log_2 302231454903657293676544 = 78$, $\log_2 604462909807314587353088 = 79$, $\log_2 1208925819614629174706176 = 80$, $\log_2 2417851639229258349412352 = 81$, $\log_2 4835703278458516698824704 = 82$, $\log_2 9671406556917033397649408 = 83$, $\log_2 19342813113834066795298816 = 84$, $\log_2 38685626227668133590597632 = 85$, $\log_2 77371252455336267181195264 = 86$, $\log_2 154742504910672534362390528 = 87$, $\log_2 309485009821345068724781056 = 88$, $\log_2 618970019642690137449562112 = 89$, $\log_2 1237940039285380274899124224 = 90$, $\log_2 2475880078570760549798248448 = 91$, $\log_2 4951760157141521099596496896 = 92$, $\log_2 9903520314283042199192993792 = 93$, $\log_2 19807040628566084398385987584 = 94$, $\log_2 39614081257132168796771975168 = 95$, $\log_2 79228162514264337593543950336 = 96$, $\log_2 158456325028528675187087900672 = 97$, $\log_2 316912650057057350374175801344 = 98$, $\log_2 633825300114114700748351602688 = 99$, $\log_2 1267650600228229401496703205376 = 100$, $\log_2 2535301200456458802993406410752 = 101$, $\log_2 5070602400912917605986812821504 = 102$, $\log_2 10141204801825835211973625643008 = 103$, $\log_2 20282409603651670423947251286016 = 104$, $\log_2 40564819207303340847894502572032 = 105$, $\log_2 81129638414606681695789005144064 = 106$, $\log_2 162259276829213363391578010288128 = 107$, $\log_2 324518553658426726783156020576256 = 108$, $\log_2 649037107316853453566312041152512 = 109$, $\log_2 1298074214633706907132624082305024 = 110$, $\log_2 2596148429267413814265248164610048 = 111$, $\log_2 5192296858534827628530496329220096 = 112$, $\log_2 10384593717069655257060992658440192 = 113$, $\log_2 20769187434139310514121985316880384 = 114$, $\log_2 41538374868278621028243970633760768 = 115$, $\log_2 83076749736557242056487941267521536 = 116$, $\log_2 166153499473114484112975882535043072 = 117$, $\log_2 332306998946228968225951765070086144 = 118$, $\log_2 664613997892457936451903530140172288 = 119$, $\log_2 1329227995784915872903807060280344576 = 120$, $\log_2 2658455991569831745807614120560689152 = 121$, $\log_2 5316911983139663491615228241121378304 = 122$, $\log_2 10633823966279326983230456482242756608 = 123$, $\log_2 21267647932558653966460912964485513216 = 124$, $\log_2 42535295865117307932921825928971026432 = 125$, $\log_2 85070591730234615865843651857942052864 = 126$, $\log_2 170141183460469231731687303715884105728 = 127$, $\log_2 340282366920938463463374607431768211456 = 128$, $\log_2 680564733841876926926749214863536422912 = 129$, $\log_2 1361129467683753853853498429727072845824 = 130$, $\log_2 2722258935367507707706996859454145691648 = 131$, $\log_2 5444517870735015415413993718908291383296 = 132$, $\log_2 10889035741470030830827987437816582766592 = 133$, $\log_2 21778071482940061661655974875633165533184 = 134$, $\log_2 43556142965880123323311949751266331066368 = 135$, $\log_2 87112285931760246646623899502532662132736 = 136$, $\log_2 174224571863520493293247799005065324265472 = 137$, $\log_2 348449143727040986586495598010130648530944 = 138$, $\log_2 696898287454081973172991196020261297061888 = 139$, $\log_2 1393796574908163946345982392040522594123776 = 140$, $\log_2 2787593149816327892691964784081045188247552 = 141$, $\log_2 5575186299632655785383929568162090376495104 = 142$, $\log_2 11150372599265311570767859136324180752990208 = 143$, $\log_2 22300745198530623141535718272648361505980416 = 144$, $\log_2 44601490397061246283071436545296723011960832 = 145$, $\log_2 89202980794122492566142873090593446023921664 = 146$, $\log_2 178405961588244985132285746181186892047843328 = 147$, $\log_2 356811923176489970264571492362373784095686656 = 148$, $\log_2 713623846352979940529142984724747568191373312 = 149$, $\log_2 1427247692705959881058285969449495136382746624 = 150$, $\log_2 2854495385411919762116571938898990272765493248 = 151$, $\log_2 5708990770823839524233143877797980545530986496 = 152$, $\log_2 11417981541647679048466287755595961091061972992 = 153$, $\log_2 22835963083295358096932575511191922182123945984 = 154$, $\log_2 45671926166590716193865151022383844364247891968 = 155$, $\log_2 91343852333181432387730302044767688728495783936 = 156$, $\log_2 182687704666362864775460604089535377456991567872 = 157$, $\log_2 365375409332725729550921208179070754913983135744 = 158$, $\log_2 730750818665451459101842416358141509827966271488 = 159$, $\log_2 1461501637330902918203684832716283019655932542976 = 160$, $\log_2 2923003274661805836407369665432566039311865085952 = 161$, $\log_2 5846006549323611672814739330865132078623730171904 = 162$, $\log_2 11692013098647223345629478661730264157247460343808 = 163$, $\log_2 23384026197294446691258957323460528314494920687616 = 164$, $\log_2 46768052394588893382517914646921056628989841375232 = 165$, $\log_2 93536104789177786765035829293842113257979682750464 = 166$, $\log_2 187072209578355573530071658587684226515959365500928 = 167$, $\log_2 374144419156711147060143317175368453031918731001856 = 168$, $\log_2 748288838313422294120286634350736906063837462003712 = 169$, $\log_2 1496577676626844588240573268701473812127674924007424 = 170$, $\log_2 2993155353253689176481146537402947624255349848014848 = 171$, $\log_2 5986310706507378352962293074805895248510699696029696 = 172$, $\log_2 11972621413014756705924586149611790497021399392059392 = 173$, $\log_2 23945242826029513411849172299223580994042798784118784 = 174$, $\log_2 47890485652059026823698344598447161988085597568237568 = 175$, $\log_2 95780971304118053647396689196894323976171195136475136 = 176$, $\log_2 191561942608236107294793378393788647952342390272950272 = 177$, $\log_2 383123885216472214589586756787577295904684780545900544 = 178$, $\log_2 766247770432944429179173513575154591809369561091801088 = 179$, $\log_2 1532495540865888858358347027150309183618739122183602176 = 180$, $\log_2 3064991081731777716716694054300618367237478244367204352 = 181$, $\log_2 6129982163463555433433388108601236734474956488734408704 = 182$, $\log_2 12259964326927110866866776217202473468949912977468817408 = 183$, $\log_2 24519928653854221733733552434404946937899825954937634816 = 184$, $\log_2 49039857307708443467467104868809893875799651909875269632 = 185$, $\log_2 98079714615416886934934209737619787751599303819750539264 = 186$, $\log_2 196159429230833773869868419475239575503198607639501078528 = 187$, $\log_2 392318858461667547739736838950479151006397215279002157056 = 188$, $\log_2 784637716923335095479473677900958302012794430558004314112 = 189$, $\log_2 1569275433846670190958947355801916604025588861116008628224 = 190$, $\log_2 3138550867693340381917894711603833208051177722232017256448 = 191$, $\log_2 6277101735386680763835789423207666416102355444464034512896 = 192$, $\log_2 12554203470773361527671578846415332832204710888928069025792 = 193$, $\log_2 25108406941546723055343157692830665664409421777856138051584 = 194$, $\log_2 50216813883093446110686315385661331328818843555712276103168 = 195$, $\log_2 100433627766186892221372630771322662657637687111424552206336 = 196$, $\log_2 200867255532373784442745261542645325315275374222849104412672 = 197$, $\log_2 401734511064747568885490523085290650630550748445698208825344 = 198$, $\log_2 803469022129495137770981046170581301261101496891396417650688 = 199$, $\log_2 1606938044258990275541962092341162602522202993782792835301376 = 200$, $\log_2 3213876088517980551083924184682325205044405987565585670602752 = 201$, $\log_2 6427752177035961102167848369364650410088811975131171341205504 = 202$, $\log_2 12855504354071922204335696738729300820177623950262342682411008 = 203$, $\log_2 25711008708143844408671393477458601640355247900524685364822016 = 204$, $\log_2 51422017416287688817342786954917203280710495801049370729644032 = 205$, $\log_2 102844034832575377634685573909834406561420991602098741459288064 = 206$, $\log_2 205688069665150755269371147819668813122841983204197482918576128 = 207$, $\log_2 411376139330301510538742295639337626245683966408394965837152256 = 208$, $\log_2 822752278660603021077484591278675252491367932816789931674304512 = 209$, $\log_2 1645504557321206042154969182557350504982735865633579863348609024 = 210$, $\log_2 3291009114642412084309938365114701009965471731267159726697218048 = 211$, $\log_2 6582018229284824168619876730229402019930943462534319453394436096 = 212$, $\log_2 13164036458569648337239753460458804039861886925068638906788872192 = 213$, $\log_2 26328072917139296674479506920917608079723773850137277813577744384 = 214$, $\log_2 52656145834278593348959013841835216159447547700274555627155488768 = 215$, $\log_2 105312291668557186697918027683670432318895095400549111254310977536 = 216$, $\log_2 210624583337114373395836055367340864637790190801098222508621955072 = 217$, $\log_2 421249166674228746791672110734681729275580381602196445017243910144 = 218$, $\log_2 842498333348457493583344221469363458551160763204392890034487820288 = 219$, $\log_2 1684996666696914987166688442938726917102321526408785780068975640576 = 220$, $\log_2 3369993333393829974333376885877453834204643052817571560137951281152 = 221$, $\log_2 6739986666787659948666753771754907668409286105635143120275902562304 = 222$, $\log_2 13479973333575319897333507543509815336818572211270286240551805124608 = 223$, $\log_2 26959946667150639794667015087019630673637144422540572481103610249216 = 224$, $\log_2 53919893334301279589334030174039261347274288845081144962207220498432 = 225$, $\log_2 107839786668602559178668060348078522694548577690162289924414440996864 = 226$, $\log_2 215679573337205118357336120696157045389097155380324579848828881993728 = 227$, $\log_2 431359146674410236714672241392314090778194310760649159697657763987456 = 228$, $\log_2 862718293348820473429344482784628181556388621521298319395315527974912 = 229$, $\log_2 1725436586697640946858688965569256363112777243042596638790631055949824 = 230$, $\log_2 34508731733952818937173779311385127262255544860851932775812621118996$

Ranking 7/14

■ Combining the two (**tf.idf**)

$$\begin{aligned} \text{idf}_{\text{university}} &= 6 \\ \text{idf}_{\text{of}} &= 1 \\ \text{idf}_{\text{freiburg}} &= 10 \end{aligned}$$

- For comparison, here is our earlier **tf** only example

university	...	57	5	, ...	123	2	, ...
of	...	57	14	, ...	123	23	, ...
freiburg	...	57	3	, ...	123	1	, ...
SCORE SUM	...	57	22	, ...	123	26	, ...

- And here the same lists with **tf.idf** scores

university	...	57	^{=5.6} 30	, ...	123	^{=2.6} 12	, ...
of	...	57	^{=14.1} 14	, ...	123	^{=23.1} 23	, ...
freiburg	...	57	^{=3.10} 30	, ...	123	^{=1.10} 10	, ...
SCORE SUM	...	57	74	, ...	123	45	, ...

■ Problems with tf.idf in practice

- The **idf** part is fine, but the **tf** part has several problems
- Let **w** be a word, and **D₁** and **D₂** be two documents
- **Problem 1:** assume that **D₁** is longer than **D₂**

Then **tf** for **w** in **D₁** tends to be larger than **tf** for **w** in **D₂**, because **D₁** is longer, not because it's more "about" **w**

- **Problem 2:** assume that **D₁** and **D₂** have the same length, and that the **tf** of **w** in **D₁** is twice the **tf** of **w** in **D₂**

Then it is reasonable to assume that **D₁** is more "about" **w** than **D₂**, but just a little more, and not twice more

■ The **BM25** (best match) formula

- This **tf.idf** style formula became famous for outperforming other formulas in standard benchmarks over many years

BM25 score = $\text{tf}^* \cdot \log_2(N / \text{df})$, where $\log = 0 \Rightarrow \log^* := 0$
 $\text{tf}^* = \text{tf} \cdot (k + 1) / (k \cdot \alpha + \text{tf})$ $\alpha = (1 - b + b \cdot \text{DL} / \text{AVDL})$
 $b = 0 \Rightarrow \alpha = 1$
 $b \in [0, 1]$

tf = term frequency, **DL** = document length, **AVDL** = average document length (measured in number of words)

- Standard setting for **BM25**: $k = 1.75$ and $b = 0.75$

Binary: $k = 0, b = 0$; Normal tf.idf: $k = \infty, b = 0$

$$\log^* = \log \cdot \frac{0 + 1}{0 + \log} = 1$$

$$\log^* = \log \cdot \frac{\infty + 1}{\infty + \log} = \log \cdot \frac{1 + 1/\infty}{1 + \log/\infty} \xrightarrow{\infty \rightarrow \infty} = \log \cdot \frac{1}{1} = \log$$

Ranking 10/14

■ Plausibility argument for BM25, part 1 ($tf \rightarrow tf^*$)

– Start with the simple formula $tf \cdot idf$

$$\alpha = 1$$

– Replace tf by tf^* such that the following properties hold:

• $tf^* = 0$ if and only if $tf = 0$

• tf^* increases as tf increases

• $tf^* \rightarrow \text{fixed limit}$ as $tf \rightarrow \infty$

$$tf^* = \frac{\alpha + 1}{\alpha / tf + 1} > 0 \text{ if } tf > 0$$

$$tf^* = \frac{\alpha + 1}{\alpha / tf + 1}$$



$$tf^* = \frac{\alpha + 1}{\alpha / tf + 1} \xrightarrow{tf \rightarrow \infty} \alpha + 1$$

– The "simplest" formula with these properties is

• $tf^* = tf \cdot (k + 1) / (k \cdot \alpha + tf)$ for any α (see next slide)

$$tf^* = tf \cdot \frac{\alpha + 1}{\alpha + tf} = \frac{tf \cdot \alpha + tf}{\alpha + tf} = \frac{\alpha + 1}{\alpha / tf + 1}$$

■ Plausibility argument for BM25, part 2 (choice of α)

- The α can be understood as a "normalization" of tf

$$tf^* = tf \cdot (k + 1) / (k \cdot \alpha + tf) = tf/\alpha \cdot (k + 1) / (k + tf/\alpha)$$

- BM25 normalizes by the length of the document

- Full normalization: $\alpha = DL / AVDL$... too extreme
- Some normalization: $\alpha = (1 - b) + b \cdot DL / AVDL$

In the literature, you find much more "theory" behind the BM25 formula. To me, it is **not** more convincing than the simple plausibility arguments from this slide and the last

■ BM25 implementation advice

- First compute the inverted lists with **tf** scores

We already did that (implicitly) in Lecture 1 (not in ES1)

- Along with that compute the document length (DL) for each document, and the average document length (AVDL)

You can measure DL (and AVDL) via the number of words

- Make a second pass over the inverted lists and replace the **tf** scores by **tf* · idf** scores

$$\text{tf} \cdot (k + 1) / (k \cdot (1 - b + b \cdot \text{DL} / \text{AVDL}) + \text{tf}) \cdot \log_2 (N / \text{df})$$

Note that the **df** of a word is just the length (number of postings) in its inverted list

■ Further refinements

- For ES2, play around with the BM25 parameters **k** and **b**
- Boost results that match each query word at least once

Warning: when you **restrict** your results to such matches, you miss relevant results, see your bad queries from ES1

- Somehow take the popularity of a movie into account

In the file on the Wiki, movies are sorted by popularity

Note: some scores are already normalized (e.g. IMDb rating), others can't be taken directly as scores (e.g. IMDb #votes)

A common heuristic to normalize scores with huge differences is to apply the **logarithm** (like we did for computing **idf**)

- Anything else that comes to your mind and might help ...

■ Other methods

- There is a multitude of other sources / approaches for improving the quality of the ranking

- **Example 1:** Using query logs and click-through data

Who searches what and clicked on what ... important signal in the ranking of big search engines like Google

- **Example 2:** Learning to rank

1. For a given query, compute a feature vector (of many different "scores") for a subset of candidate documents
2. Use machine learning to rank these feature vectors, and thus the documents

■ Ground truth

- For a set of queries, for each query from that set, determine the set of ids of **all** documents relevant for that query, e.g.

Query: matrix movies

Relevant: 10, 582, 877, 10003

- This set is also called the **ground truth** for that query, and a set of queries + their ground truth is called a **benchmark**
- For ES2, we have built a benchmark with **56** queries

Building high-quality benchmarks of sufficient size is an important (and time-consuming) part of research



- On the next slides we will show how to use queries with a ground truth to measure the quality of a search engine

■ Precision (**P@k**)

- The P@k for a given result list for a given query is the percentage of relevant documents among the top-k

Query: matrix movies

Relevant: 10, 582, 877, 10003

Result list: 582, 17, 5666, 10003, 10, 37, ...

REL
877
not in
the list

P@1:	1 / 1 = 100%	= 1
P@2:	1 / 2 = 50%	= 0.5
P@3:	1 / 3 = 33%	= 0.33
P@4:	2 / 4 = 50%	= 0.5
P@5:	3 / 5 = 60%	= 0.6

- **P@R** = P@k, where k is the number of relevant documents
 $P@R = P@4 = 50\% = 0.5$

■ Average Precision (**AP**)

- Let R_1, \dots, R_k be the sorted list of positions of the relevant documents in the result list of a given query
- Then AP is the average of the k $P@R_i$ values

Query: matrix movies

Relevant: 10, 582, 877, 10003

Result list: 582, 17, 5666, 10003, 10, ..., 877

R_1, \dots, R_4 : 1, 4, 5, 40

$P@R_1, \dots, P@R_4$: $P@1 = 100\%$, $P@4 = 50\%$, $P@5 = 60\%$, $P@40 = \frac{4}{40} = 10\%$
 $(100\% + 50\% + 60\% + 10\%) / 4 = 55\%$

AP:

Note: AP must also consider documents that do not appear in the result list ... for those, we take $P@R_i = 0$

■ Mean Precisions (**MP@k**, **MP@R**, **MAP**)

- Given a benchmark with several queries + ground truth
- Then one can capture the quality of a system by taking the **mean** (average) of a given measure over all queries

MP@k = mean of the $P@k$ values over all queries

MP@R = mean of the $P@R$ values over all queries

MAP = mean of the AP values over all queries

These are very common measures, which you will find in many research papers concerned with ranking objects

■ Discounted Cumulative Gain (DCG, nDCG)

- Sometimes relevance comes in more than one shade, e.g.

0 = not relevant, 1 = somewhat rel, 2 = very relevant

- There should be a bonus if very relevant documents are ranked higher than only somewhat relevant documents

- **Cumulative gain:** $CG@k = \sum_{i=1..k} rel_i$ rel_i = relevance of i-th doc in result list

- **Discounted CG:** $DCG@k = \sum_{i=1..k} rel_i / \log_2 (i + 1)$

- Problem: CG and DCG are not normalized to [0,1]

- Solution: normalize by maximally achievable value

- **Ideal DCG:** $iDCG@k = DCG@k$ of ideal ranking

- **Normalized DCG:** $nDCG@k = DCG@k / iDCG@k$

■ Discounted Cumulative Gain (DCG, nDCG), example

- Consider the following result list and relevances, assuming only 3 relevant documents overall (for this query)

Hit #1: relevant 1

Hit #2: not relevant 0

Hit #3: very relevant 2

Hit #4: relevant 1

Hit #5: not relevant 0

$$\text{DCG}@2 = \sum_{i=1}^2 \text{rel}_i / \log_2(i+1)$$

$$= 1 + 1 + \frac{1}{\log_2 5} = 2.42\dots$$

– Then $\text{DCG}@5 = \frac{1}{\log_2 2} + \frac{0}{\log_2 3} + \frac{2}{\log_2 4} + \frac{1}{\log_2 5} + \frac{0}{\log_2 6}$

$\log_2 2 = 1$, $\log_2 4 = 2$, $\log_2 5 = 2.34\dots$

– And $\text{iDCG}@5 = \frac{2}{\log_2 2} + \frac{1}{\log_2 3} + \frac{1}{\log_2 4} + \frac{0}{\log_2 5} + \frac{0}{\log_2 6}$

– Hence $\text{nDGC}@5 = \frac{2.42\dots}{3.13\dots} \approx 77\%$

$\text{iDCG}@5 = 2 + \frac{1}{\log_2 3} + 0.5 = 3.13$

$\log_2 3 = 1.58\dots$, $\frac{1}{\log_2 3} = 0.63$

A common approach in competitions is **pooling**: judge only the top-k results from each participant

■ Binary preference (bpref)

- Sometimes we have relevance judgements only for a subset of all the documents

Typically for very large datasets, where it is infeasible to check all the documents for relevance

- Then for each judged relevant doc, compute a score based on how many judged non-relevant docs come before it, and take **bpref** as the average of these scores, formally:

$$\text{bpref} = 1/|R| \cdot \sum_{d \in RR} (1 - |NR(d)| / \min(|R|, |N|)) \text{ where}$$

R = docs judged relevant

N = docs judged non-relevant

RR = docs judged relevant in result list

$NR(d)$ = docs from the $|R|$ top-ranked non-relevant before d

Evaluation 8/12

■ Binary preference (bpref), example

- Consider the following result list

#1: ???

#2: not relevant

#3: relevant

#4: not relevant

#5: ???

#6: ???

#7: not relevant

#8: relevant

#9: ???

Not in list:

One more relevant document

Ten more non-relevant documents

$$R = \{ \#3, \#8 \}$$

$$|R| = 2$$

$$|N| = 10$$

$$\min\{|R|, |N|\} = 2$$

$$NR(\#3) = \cancel{1} \{ \#2 \}$$

$$NR(\#8) = \cancel{3} \{ \#2, \#4, \#7 \}$$

$$1 - \frac{|NR(\#3)|}{\min\{|R|, |N|\}} = 1 - \frac{1}{2} = 1/2$$

$$1 - \frac{|NR(\#8)|}{\min\{|R|, |N|\}} = 1 - \frac{3}{2} = 0$$

$$BPREF =$$

$$\frac{1/2 + 0 \times \cancel{10}}{2} = \frac{1/2}{2} = 1/4 = 25\%$$

■ Binary preference (bpref), finer points

- Why $NR(d) / \min(|R|, |N|)$ and not $NR(d) / |N|$?

First verify that both $NR(d) \leq |R|$ and $NR(d) \leq |N|$, so that $NR(d) / \min(|R|, |N|)$ is always in the range $[0, 1]$

Then note that $1 - NR(d) / \min(|R|, |N|)$ becomes **zero** already when only $|R|$ non-relevant docs come before d

With $NR(d) / \min(|R|, |N|)$ instead of $NR(d) / |N|$ it thus becomes harder to achieve a high **bpref** score

- Why $1 / |R|$ and not $1 / |RR|$?

It should be punished if a relevant doc is not in the result list

It's the same reason that for AP (average precision) on slide 23, we took $P@R_i = 0$ for docs that are not in the result list

■ Overfitting

- Tuning parameters / methods to achieve good results on a given benchmark is called **overfitting**

In an extreme case: for each query from the benchmark, output the list of relevant docs from the ground truth

- In a realistic environment (real search engine or competition), one is given a **training** set for development

The actual evaluation is on a **test** set, which must not be used or is not available during development

For ES2, we explicitly provide a training benchmark for development, and a test benchmark to measure the quality of your system once you are done with tuning

■ Set vs. ranked list

- On the previous slides, the ground truth for each query was a **set** and the results we evaluated were **ranked lists**

That looks strange, why didn't we evaluate it in one of the following three ways instead?

- **1. Why** don't we let our search engine also output a set and measure how similar it is to the ground truth?

Because there is no natural cut-off value

- **2. Why** isn't our ground truth also a ranked list and we measure the similarity of the two rankings?

Because the set of relevant documents usually doesn't have a full order (though there might be levels of relevance)

■ Set vs. ranked list, continued

- **3. Why** don't we evaluate the accuracy of the scores themselves, instead of only the order that is implied by these scores?

The absolute values of our scores don't mean much, we only compute them as a means to rank the documents

Note: there are indeed problems, where the goal is to predict an absolute score as accurately as possible

These are called regression problems, and we will encounter them in a later lecture

References

- In the Manning/Raghavan/Schütze textbook

 - Section 6: Scoring and Term Weighting

 - Section 8: Evaluation in Information Retrieval

- Relevant Papers

 - Probabilistic Relevance: BM25 and Beyond

 - FnTIR 2009**

 - Test Collection Based Evaluation of IR Systems

 - FnTIR 2010**

- Relevant Wikipedia articles

 - http://en.wikipedia.org/wiki/Okapi_BM25

 - https://en.wikipedia.org/wiki/Information_retrieval#Performance_and_correctness_measures