POLITECNICO DI MILANO

Natural Language Processing

Searching Text

Natural Language Processing

Some slide content based on textbook:
An Introduction to Information Retrieval by Christopher D. Manning, Prabhakar Raghavan, & Hinrich Schütze

Mark Carman

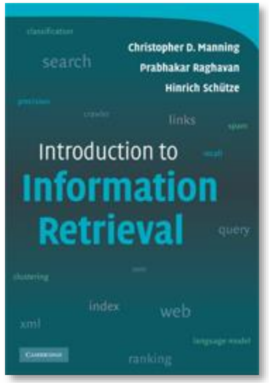
Contents

- What is Information Retrieval?
- Term weighting
- Building Indices
- Crawling
- Training a Reranker
- Evaluating search

Source https://www.pexels.com/photo/google-internet-online-search-search-48123/

Textbook:

- *An Introduction to Information Retrieval*
  - by Christopher D. Manning, Prabhakar Raghavan, & Hinrich Schütze
  - Free online: https://nlp.stanford.edu/IR-book/information-retrieval-book.ht

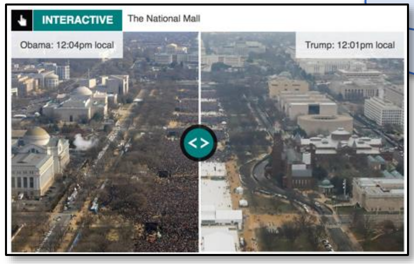


Information Retrieval

What is Information Retrieval?

Task of **finding content**, which could be documents, images, video, etc.

- that is useful (i.e. relevant) to user's **information need**



Source : https://www.bbc.com/news/world-us-canada-38682574

My inauguration crowd was the biggest ever. Period!
Somebody find me some pictures!

Source: https://www.flickr.com/photos/gageskidmore/29381357345

My fingers look pretty big to me.
They can't be shorter than the average, can they?

Gender	Right hand (RH) (mm)			Left hand (LH) (mm)		
	Rt2D	Rt3D	Rt4D	Lt2D	Lt3D	Lt4D
Male	74.278	80.172	75.715	74.61	80.421	75.697
Female	65.894	71.807	67.541	66.192	71.987	67.344

Source : https://www.researchgate.net/publication/281750703_Digit_ratio_2D4D_Index_finger_Ring_finger_in_the_right_and_left_hand_of_males_and_females_in_Malaysia

Hmm. Those banana and apple pancakes look so good! I wonder how you make them …

Source : https://food.ndtv.com/recipe-banana-pancakes-with-caramelized-apples-588999

Information needs to query keywords

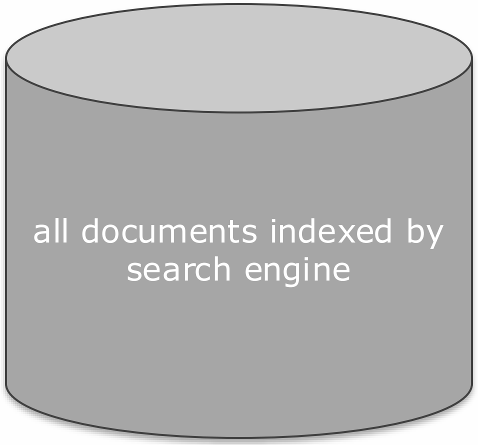
From **information needs** we extract **query keywords**

- and look for documents containing those keywords

Hmm. Those banana and apple pancakes look so good! I wonder how you make them …

Source: https://www.flickr.com/photos/gageskidmore/29381357345

large collection containing many **billions** of documents:



millions of docs contain word: banana

thousands of docs contain words: banana + apple

hundreds of docs contain words: banana + apple + pancakes

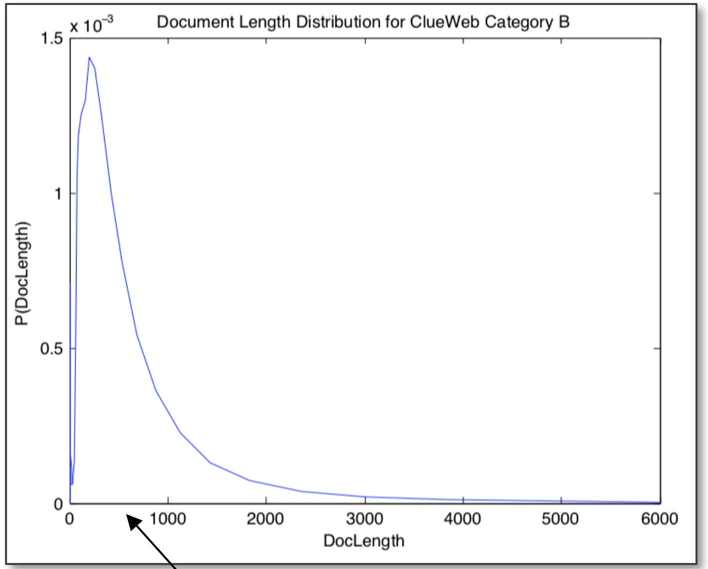
tens of docs contain words: banana + apple + pancakes + make

mostly only relevant documents

So web search isn't hard?

Not really:

- **typical adult vocabulary** lies in range 20 to 35 thousand words:
  see: https://www.economist.com/johnson/2013/05/29/lexical-facts
- **and typical document length** is less than one thousand words
- so **vocabulary of each document** is **small**



Average document length for web crawl (ClueWeb09) is only 756
https://www.sciencedirect.com/science/article/pii/S1532046414000677

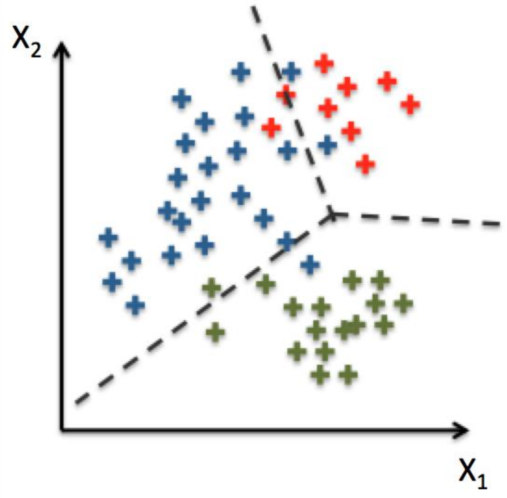
Heavy lifting in text retrieval can be done by **vocabulary matching**!

- providing vocabulary is well distributed over different documents
- just need **fast indexes** to find all documents containing selected keywords

But what if **no document** contains all query terms or **many documents** contain all of them?

- assign score to keywords based on how **discriminative** they are
- expand **document representation** to include more information (e.g. page importance) and train **ML model**

Is retrieval just text classification?



Why not just train a classifier?

- **concatenate** query and document bag-of-words into single feature vector
- **predict** probability that user finds document relevant to query
  - obviously can't use linear classifier since interactions between query & document terms wouldn't be taken into account
  - could train **non-linear model** that includes **pairwise interactions** between query & document terms
- problems:
  - for vocabulary of 100 thousand, might require up to 10 billion parameters to cover all pairwise interactions!
  - need huge amounts of training data in form: (query, document, relevance_label)
  - retrieval would be very slow if need to iterate over all documents to calculate score for each

We will return to idea of treating retrieval as a classification/regression problem later

Term Weighting

Some terms are **more discriminative** than others,
making them **more useful** for identifying relevant documents

Term weighting – query term subsets

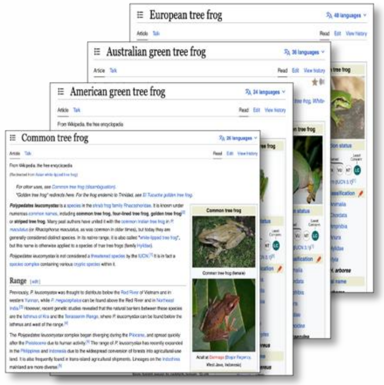
Source:
https://en.wikipedia.org/wiki/Agalychnis_callidryas#/media/File:Red-eyed_Tree_Frog_(Agalychnis_callidryas)_1.png

Imagine searching for name of cool frog you saw in forest

- you enter query "**giant tree frogs**" but **no document** in Wikipedia contains all 3 keywords
- so you drop one keyword at a time:

many documents contain:
(**giant, tree**)

a couple contain:
(**giant, frog**)



and a few contain:
(**tree, frog**)

Which set of documents is likely the most relevant?

- in general, the **smallest set** will be the most specific and **most on topic**
- so rank documents by size of returned document set (smaller the better) for query term subset
- is there are principled way to extend this idea into an efficient ranking algorithm?

Motivating IDF weighting

We can **estimate** how many documents be returned for given query term subset:

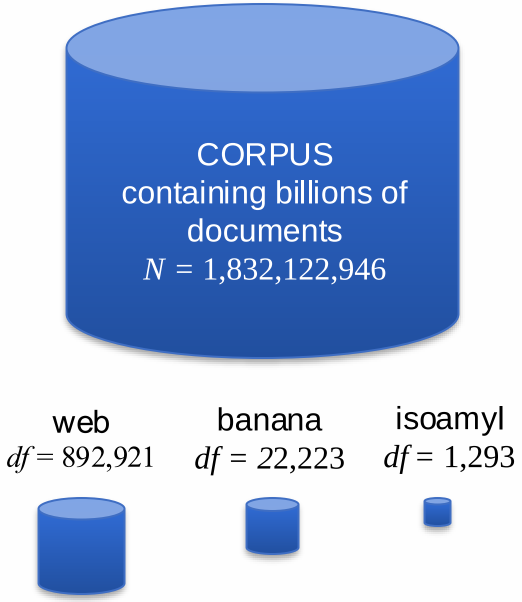
- using **probability** that **random document would** contain those terms
- then rank documents by how **unlikely** it was see **so many query terms** in it

Probability that randomly chosen document contains keywords: $t_1, \ldots, t_{|q|} \in q$

- assuming terms are independent we have,

$$P(q \subseteq d') = \prod_{t \in q} P(t \in d') = \prod_{t \in q} \frac{\mathrm{df}_t}{N}$$

- where **document frequency**, df_t = number of docs in corpus that contain t
- and corpus size, N = number of docs in corpus



Rank documents by how **unlikely** they are:

- so score by **one on the probability** $(1/P)$ resulting in an **inverse document frequency (IDF)** weighting
- and take logarithm to make **score additive** (rather than multiplicative) over query terms

$$score(d) = -\log \prod_{t \in q \cap d} P(t \in d') = \sum_{t \in q \cap d} \log \frac{N}{\mathrm{df}_t}$$

Information Theory & odds variant

Inverse Document Frequency (IDF)

- weights each term by logarithm of inverse probability of finding term in a document: $$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t}$$
- standard **Information Theory** measure for information gained from observing term:

  $\mathrm{info}(t) = -log\ P(t)$

  - amount of information = surprise at observing term
  - same formula used to determine how many bits to use to represent words when compressing text files

Source: https://commons.wikimedia.org/wiki/File:SURPRISE.jpg

Common variant of Inverse Document Frequency (IDF)

- uses **odds** of observing term: $odds(t) = P(t)/[1-P(t)]$
- resulting in document score: $$score(d) = \sum_{t\in q} \log \frac{N - \mathrm{df}_t + \frac{1}{2}}{\mathrm{df}_t + \frac{1}{2}}$$
  - smoothing of 0.5 added to all counts to prevent terms with small frequencies (1, 2, etc.) from dominating ranking
  - little difference between two formulations unless term is very common ($\mathrm{df}_t > N/2$)

Term Weighting – TF-IDF

Term weighting – TF-IDF

> Isolated-term correction would fail to correct typographical errors such as flew form Heathrow, where all three query terms are correctly spelled. When a phrase such as this retrieves few documents, a search engine may like to offer the corrected query flew from Heathrow. The simplest way to do this is to enumerate corrections of each of the three query terms (using the methods leading up to Section 3.3.4) even though each query term is correctly spelled, then try substitutions of each correction in the phrase. For the example flew form Heathrow, we enumerate such phrases as fled form Heathrow and flew fore Heathrow. For each such substitute phrase, the search engine runs the query and determines the number of matching results.

IDF weights vocabulary terms

- but there is **more information** in a document than just its **vocabulary**!
- some documents contain the **same query term many times** and are more likely to be relevant to the query
- simplest option to include **term count** information is to directly weight score by it:

$$score(q,d) = \sum_{t \in q} \mathrm{tf}_{t,d} \log \frac{N}{\mathrm{df}_t}$$

  - where $\mathrm{tf}_{t,d}$ = # of occurrences of term t in document d (a.k.a. *term frequency*)

Can be otivated as follows:

- instead of calculating probability that random document contains the term
- calculate probability it contains the term exactly k times: $P(t,k) \cong P(next\text{-}token=t)^k$
- estimate next token probability using term occurrences over collection: $P(next\ token = t) = \mathrm{ctf}_t / \sum_{t'} \mathrm{ctf}_{t'}$
  - where **collection term frequency**, ctf_t = number of occurrences of t in collection
- thus probability that term occurs k times : $P(t,k) \cong (\mathrm{ctf}_t / \sum_{t'} \mathrm{ctf}_{t'})^{tf}$
- logarithm of product over all query terms gives: $score(q,d) = - \sum_{t \in q} \mathrm{tf}_{t,d} \log(\mathrm{ctf}_t / \sum_{t'} \mathrm{ctf}_{t'})$
- not quite the formula above, but close …
  - replacing collection frequency (ctf_t) with document frequency (df_t) doesn't drastically change formula and may make it more robust

Variants of TF-IDF

The **TF-IDF score** performs well in practice

- But assumes a linear relationship between term frequency and document score

Researchers have questioned this linear assumption:

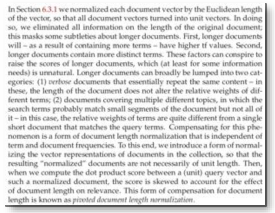
- should, all else being equal, doubling occurrences of term, double score for document?
- answer: probably not
  - score should improve with increases in the term count, but not linearly
  - already expected to see term multiple times in doc after see it for first time

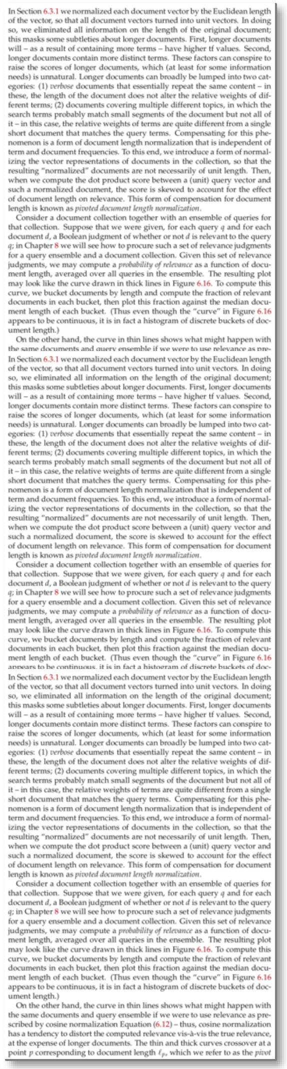
Common alternative (with little theoretical justification):

- increase score with logarithm of term count: $\log(1 + \mathrm{tf}_{t,d})$ *or* $\max(0, 1 + \log(\mathrm{tf}_{t,d}))$

Length Normalization

Term weighting – length normalisation





Need to normalise for the **length** of the document!

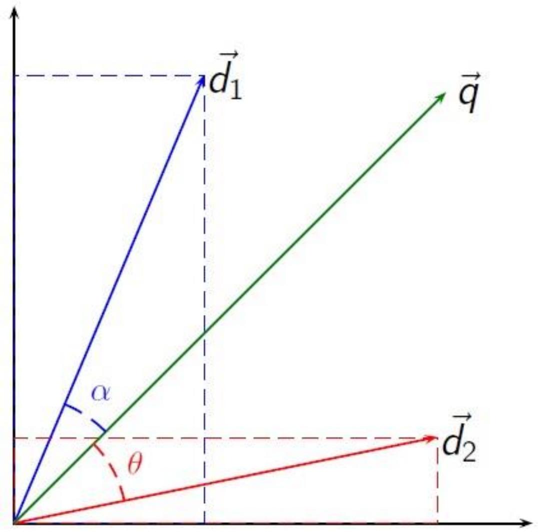
- longer documents have a larger vocabulary
  - so more likely to contain the query terms
  - but not necessarily more likely to be useful to the searcher
- so **shorter documents** with **same term count** should be **preferred**

How to normalise for length?

- could just divide by length of the document
  - is done in some "Language Modeling" based retrieval functions:

    E.g. http://sifaka.cs.uiuc.edu/czhai/pub/tois-smooth.pdf
- but most common normalization uses L_2 rather than L_1 norm …

Cosine similarity between TFIDF vectors



Source:
https://en.wikipedia.org/wiki/Vector_space_model#/media/File:Vector_space_model.jpg

Vector Space Model treats tf-idf values for all terms in document as a vector representation:

$$\mathbf{d} = (\text{tf}_{1,d}.\text{idf}_1, ..., \text{tf}_{1,d}.\text{idf}_n)$$

Similarity between query & document computed based on **angle between vectors**

- actually cosine of angle is used (since it gives similarity values in range [0,1])

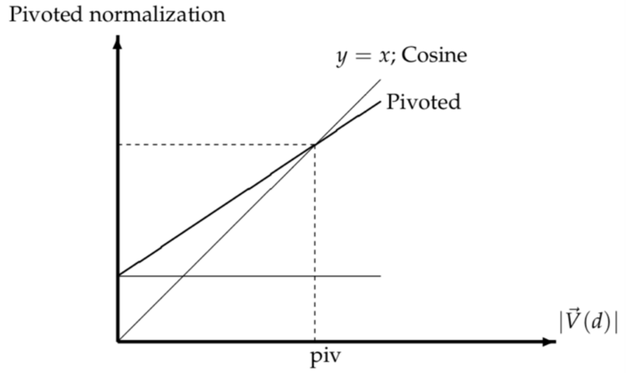
$$sim(\mathbf{d}_1, \mathbf{d}_2) = \frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{||\mathbf{d}_1|| ||\mathbf{d}_2||}$$

  - $\mathbf{d}_1$ = query, $\mathbf{d}_2$ = document
- using cosine similarity involves normalising by Euclidean length (a.k.a. L_2 norm) of vectors:

$$||\mathbf{x}|| = \sqrt{\sum_{i=1}^{n} x_i^2}$$

  - rather than the length of the document in tokens (a.k.a. the L_1 norm)

Alternative Length Normalization



There have been many studies over the years into other types of normalization

Such as Pivoted Length Normalisation (PLN):

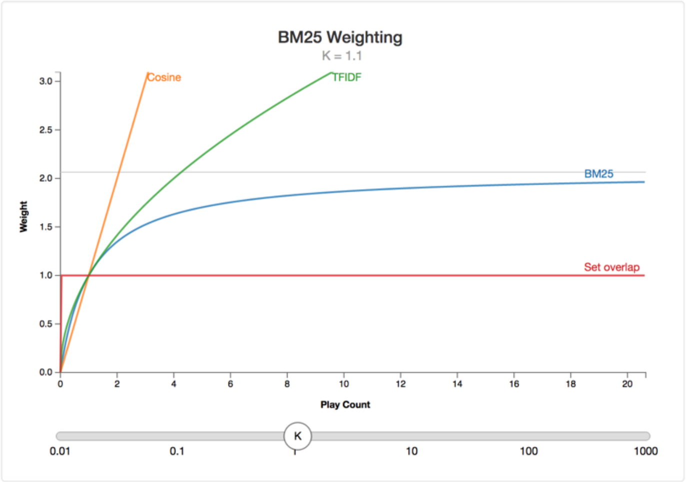
- idea: **generally longer documents do contain more information** than shorter ones, but normalizing loses all length information
- so instead parameterise L_1 normalisation around average document length:

$$\frac{\mathrm{tf}_{t,d}}{L_d} \quad \rightarrow \quad \frac{\mathrm{tf}_{t,d}}{bL_d + (1-b)L_{ave}}$$

where: $L_d = \sum_t \mathrm{tf}_{tid} \qquad L_{ave} = \frac{1}{N}\sum_d L_d$

$0 \leq b \leq 1$

Term Weighting – BM25



Source: https://www.benfrederickson.com/distance-metrics/

Pivoted length normalization leads to venerable (it's been around a while) **Okapi BM25 ranking formula**:

$$RSV_d = \sum_{t \in q} \log \left[\frac{N}{\mathrm{df}_t} \right] \cdot \frac{(k_1 + 1)\mathrm{tf}_{td}}{k_1((1-b) + b \times (L_d / L_{\mathrm{ave}})) + \mathrm{tf}_{td}}$$

k_1 , b = parameters to be set

Why the weird name BM25?

- BM stands for Best Match, and it was literally the 25th formula they tried ;-)

Was **GOTO method** for **term-based** text retrieval

- formula has stood test of time
- has nice properties:
  - term importance asymptotes to maximum value so document containing a query term repeated a massive number of times won't always rise to the top of the ranking
- Parameters control dependence on document length
  - default values for parameters (k_1 between 1.2 & 2, b=.75) are ok, but usually improved on some validation set

Index Structures

Under the Hood of a Search Engine

Retrieval measures must be calculated fast, since delay affects attention

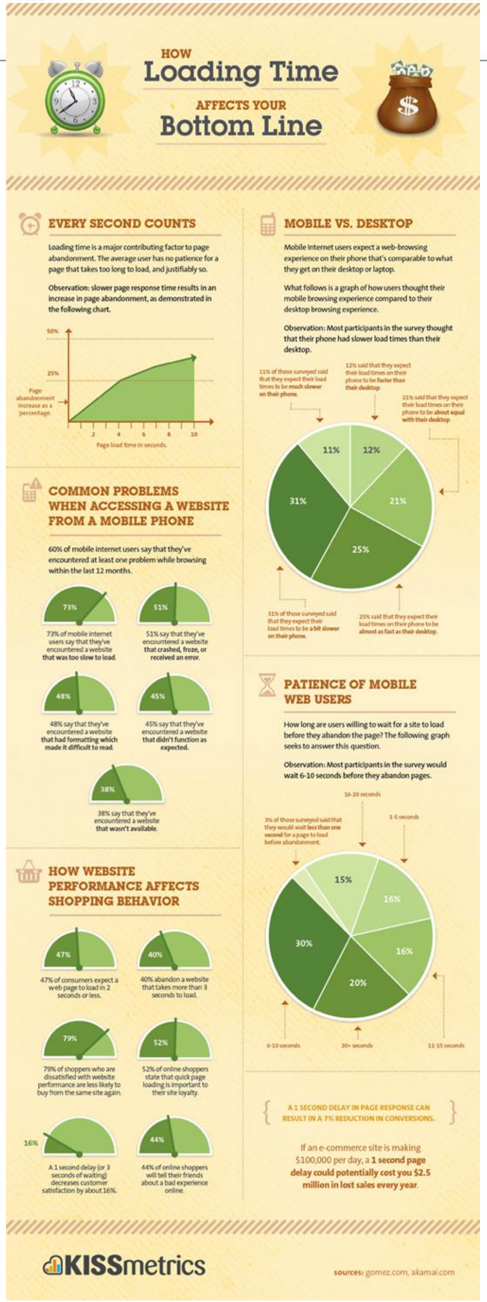
- search engines need to respond in tenths of a second
- and have been engineered to be as fast as is possible

Inverted Indices are the building blocks of search engines

- made up of **Posting Lists** mapping: TermIDs => DocumentIDs
- use integer compression algorithms that allow for fast decompression to reduce space requirement

Calculating retrieval function involves computing joins over posting lists

- documents in posting list are **sorted** by term count to allow for **early termination** of results list computation
- index pruning techniques used to get rid of documents that would never be retrieved for a certain query http://engineering.nyu.edu/~suel/queryproc/



Source: https://neilpatel.com/wp-content/uploads/2011/04/loading-time-sml.jpg

Positional Indices

Document more likely to be relevant if query terms appear close together

- most indices record locations of terms in document and allows for proximity between keywords to be calculated

Words at start of webpage more important in general

Statistically significant bigram/trigrams

- found using pointwise mutual information
- often indexed with their own posting list

Source: https://commons.wikimedia.org/wiki/File:White_House_DC.JPG

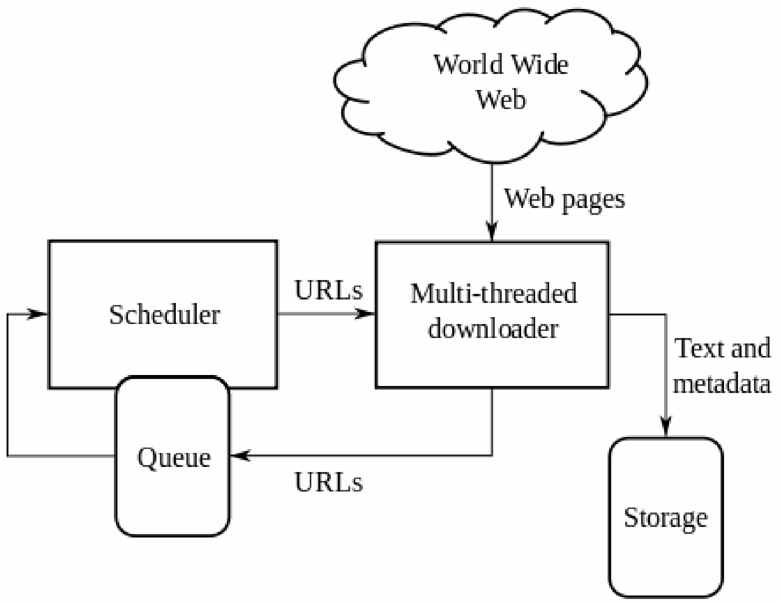
"**a** tree next to **the** white house"
vs
"**the** tree next to **a** white house"

Source: https://www.nps.gov/york/learn/historyculture/moore-house.htm

Crawlers

Scour web following hyperlinks for pages to add to index

- efficient crawling requires learning to **prioritize URLs** effectively
- and determining how frequently to re-visit a website to update index



Source: An Introduction to Information Retrieval. Manning, Raghavan & Schütze

Web scale crawlers need to be robust to all types of content

- including generated pages
- content-based duplicate page detection
  - many URLs may map to the same content
- **distributed crawler** architecture with centralised URL list
- respect **robots.txt** files
  - text file in root directory of website that tells crawler what content it can/can't crawl

Crawlers - Aside

Source https://commons.wikimedia.org/wiki/File:Tor-logo-2011-flat.svg

Dark Web

- anonymous Web built around TOR (The Onion Router) gateways
  - full of all sorts of nasty content
- crawling the Dark Web is interesting because there is no DNS linking urls to IP-addresses behind TOR gateways

Learning to Rerank

Once initial set of potentially relevant documents has been found,
why not rerank them based on all available information
in order to improve maximise overall quality of search results?

Why Rerank?

For **web search** many indicative features include:

- multiple retrieval functions (BM25, Embeddings-based, BERT,…)
- different document parts/views (titles, anchor-text, bookmarks, …)
- query-independent features (PageRank, HITS, spam scores, …)
- personalized information (user click history, …)
- context features (location, time, previous query in session, …)

Search engines like Google combine **hundreds of signals** together

- According to 2017 article (https://searchengineland.com/8-major-google-ranking-signals-2017-278450), Google's major ranking aspects were:
  - **incoming links:** who links to the page? how relevant are those links (anchortext)?
  - **content:** keywords, length, comprehensiveness
  - **technical quality:** load speed, quality of mobile page
  - **past users:** click through rate (CTR) from previous user searches

Rank learning provides an **automated & coherent** method:

- for combining diverse signals into a single retrieval score
- while optimising a measure users care about, e.g. NDCG, ERR

generating dataset: query + initial ranking

Query:

Tourism Amsterdam

1. Start with a query
2. Generate initial ranking using keyword-based ranker

	bm25
doc 1	108
doc 2	106
doc 3	92
doc 4	88
doc 5	43
doc 6	12
doc 7	4
doc 8	3
doc 9	2
doc10	1
doc11	1
doc12	1

generating dataset: truncate @ *k*

<u>Query</u>:

Tourism Amsterdam

1. Start with a query
2. Generate initial ranking using keyword-based ranker
3. Truncate ranking as candidates for re-ranking

	bm25
doc 1	108
doc 2	106
doc 3	92
doc 4	88
doc 5	43
doc 6	12
doc 7	4
doc 8	3

k

generating dataset: compute features

Query:

Tourism Amsterdam

1. Start with a query
2. Generate initial ranking using keyword-based ranker
3. Truncate ranking as candidates for re-ranking
4. Calculated feature values for each candidate

	bm25	bm25_title	anchortext	PageRank	…
doc 1	108	23	23	0.02	
doc 2	106	12	49	0.04	
doc 3	92	35	11	0.11	
doc 4	88	1	33	0.005	
doc 5	43	7	1	0.35	
doc 6	12	1	0	0.21	
doc 7	4	3	20	0.19	
doc 8	3	0	4	0.55	

generating dataset: normalise features

Query:

Tourism Amsterdam

1. Start with a query
2. Generate initial ranking using keyword-based ranker
3. Truncate ranking as candidates for re-ranking
4. Calculated feature values for each candidate
5. Normalize each feature at the query level

	bm25	bm25_title	anchortext	PageRank	...
doc 1	1.00	0.66	0.47	0.03	
doc 2	0.98	0.34	1.00	0.06	
doc 3	0.85	1.00	0.22	0.19	
doc 4	0.81	0.03	0.67	0.00	
doc 5	0.38	0.20	0.02	0.63	
doc 6	0.09	0.03	0.00	0.38	
doc 7	0.01	0.09	0.41	0.34	
doc 8	0.00	0.00	0.08	1.00	

normalize features

- usually perform min-max normalization at query-level for each feature
- necessary to make values **comparable across queries**

generating dataset: manual labelling

Query:

Tourism Amsterdam

1. Start with a query
2. Generate initial ranking using keyword-based ranker
3. Truncate ranking as candidates for re-ranking
4. Calculated feature values for each candidate
5. Normalize each feature at the query level
6. TRAINING: Provide ground-truth relevance labels for each query-document pair

	bm25	bm25_title	anchortext	PageRank	...	RELEVANCE LABELS	
doc 1	1.00	0.66	0.47	0.03		2	relevant
doc 2	0.98	0.34	1.00	0.06		0	spam
doc 3	0.85	1.00	0.22	0.19		1	not rel
doc 4	0.81	0.03	0.67	0.00		3	highly rel
doc 5	0.38	0.20	0.02	0.63		1	not rel
doc 6	0.09	0.03	0.00	0.38		1	not rel
doc 7	0.01	0.09	0.41	0.34		2	relevant
doc 8	0.00	0.00	0.08	1.00		0	spam

query-document pair

feature vector

observed label

generating dataset: repeat for all queries

Query:

Tourism Amsterdam

	bm25	bm25_title	anchortext	PageRank	...	RELEVANCE LABELS	
doc 1	1.00	0.66	0.47	0.03		2	relevant
doc 2	0.98	0.34	1.00	0.06		0	spam
doc 3	0.85	1.00	0.22	0.19		1	not rel
doc 4	0.81	0.03	0.67	0.00		3	highly rel
doc 5	0.38	0.20	0.02	0.63		1	not rel
doc 6	0.09	0.03	0.00	0.38		1	not rel
doc 7	0.01	0.09	0.41	0.34		2	relevant
doc 8	0.00	0.00	0.08	1.00		0	spam

1. Start with a query
2. Generate initial ranking using keyword-based ranker
3. Truncate ranking as candidates for re-ranking
4. Calculated feature values for each candidate
5. Normalize each feature at the query level
6. TRAINING: Provide ground-truth relevance labels for each query-document pair

ice skating Rome

	bm25	bm25_title	anchortext	PageRank	...	RELEVANCE LABELS	
doc 1	1.00	0.36	1.00	0.39		2	relevant
doc 2	0.92	0.39	0.02	0.66		3	highly rel
doc 3	0.88	0.20	0.82	0.89		2	relevant

Evaluating Search Results

Gathering Relevance Judgments

Search engines employ people to annotate search results with relevance information!

- Google's detailed guidelines are for its Raters:
  https://www.hobo-web.co.uk/google-quality-rater-guidelines/

Other organisations (e.g. Wikipedia) can't afford to pay raters and try to collect judgments directly from users

- "Would this document have been relevant to query … ?"
  https://blog.wikimedia.org/2017/09/19/search-relevance-survey/

Usually don't train models directly from click data

- because causes a feedback loop

Latest approach is to use an LLM to judge relevance

Metrics for Evaluating Search Results

Traditional Measures:

- Precision at depth k: $P@k = \#\{Relevant\ docs\ in\ top\ k\} / \mathrm{k}$
  What percentage of the top results are relevant?
- Recall at depth k: $R@k = \#\{Relevant\ docs\ in\ top\ k\} / \#\{Relevant\ docs\ in\ total\}$
  What percentage of all the relevant documents available were found?
- F-measure at depth k: $F_1@k = 1/([1/P@k + 1/R@k]/2)$
  Combining precision and recall, how well did we do?

More recent:

- MAP: Mean Average Position
  - "Average Precision" is average of P@k values at all rank positions containing relevant documents
  - estimates area under precision-recall curve

$$\mathrm{AveP} = \frac{\sum_{k=1}^{n}(P(k) \times \mathrm{rel}(k))}{\text{number of relevant documents}}$$

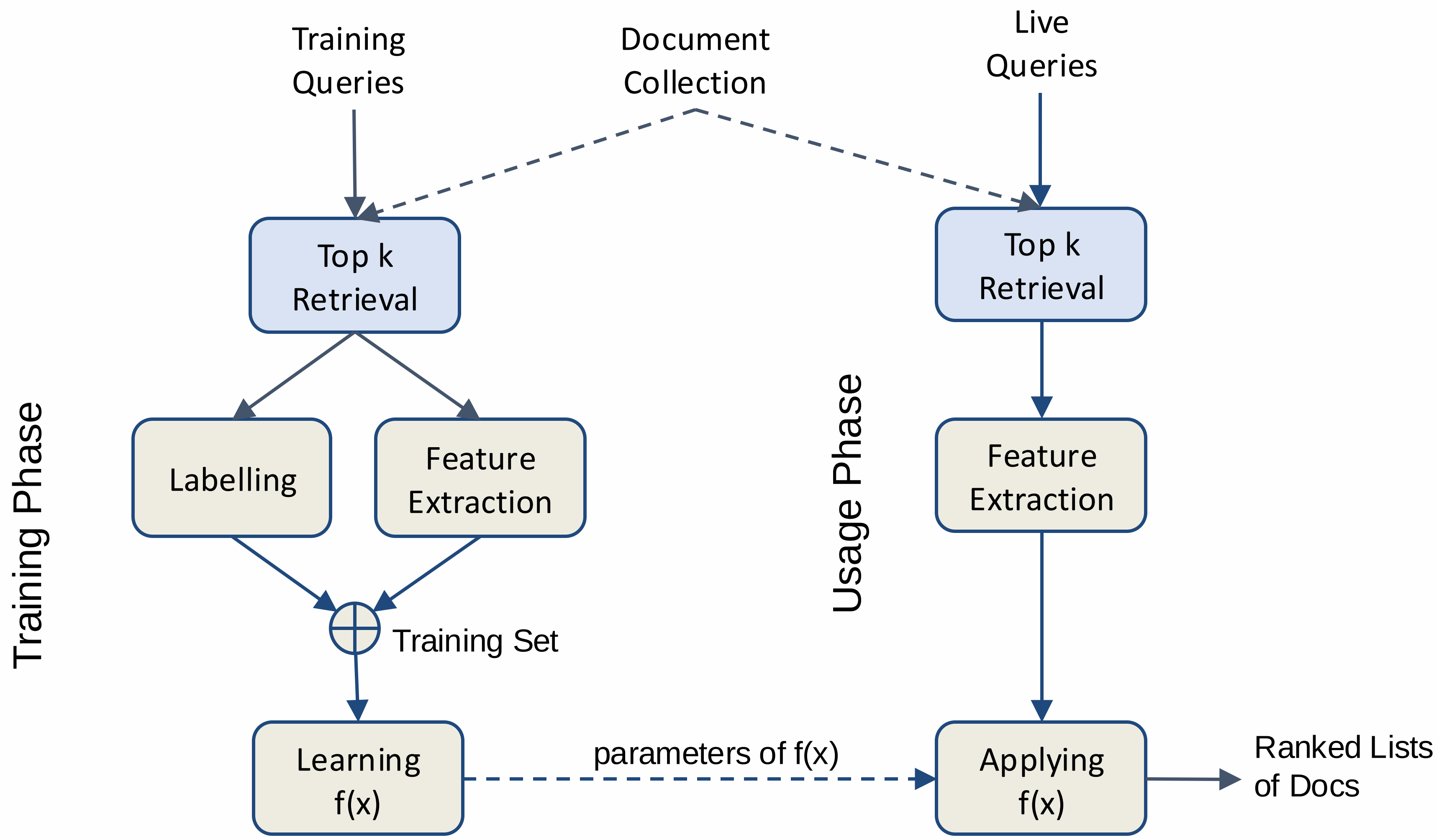
- NDCG@k: Normalized Discounted Cumulative Gain
  - more faithful to user experience by discounting lower ranked docs
  - normalized at the query level

$$\mathrm{NDCG}(Q,k) = \frac{1}{|Q|}\sum_{j=1}^{|Q|} Z_{kj} \sum_{m=1}^{k} \frac{2^{R(j,m)}-1}{\log_2(1+m)}$$

Note: $P@k$ and NDCG@k usually most important measure for retrieval

Formulating Learning-to-Rank Problem

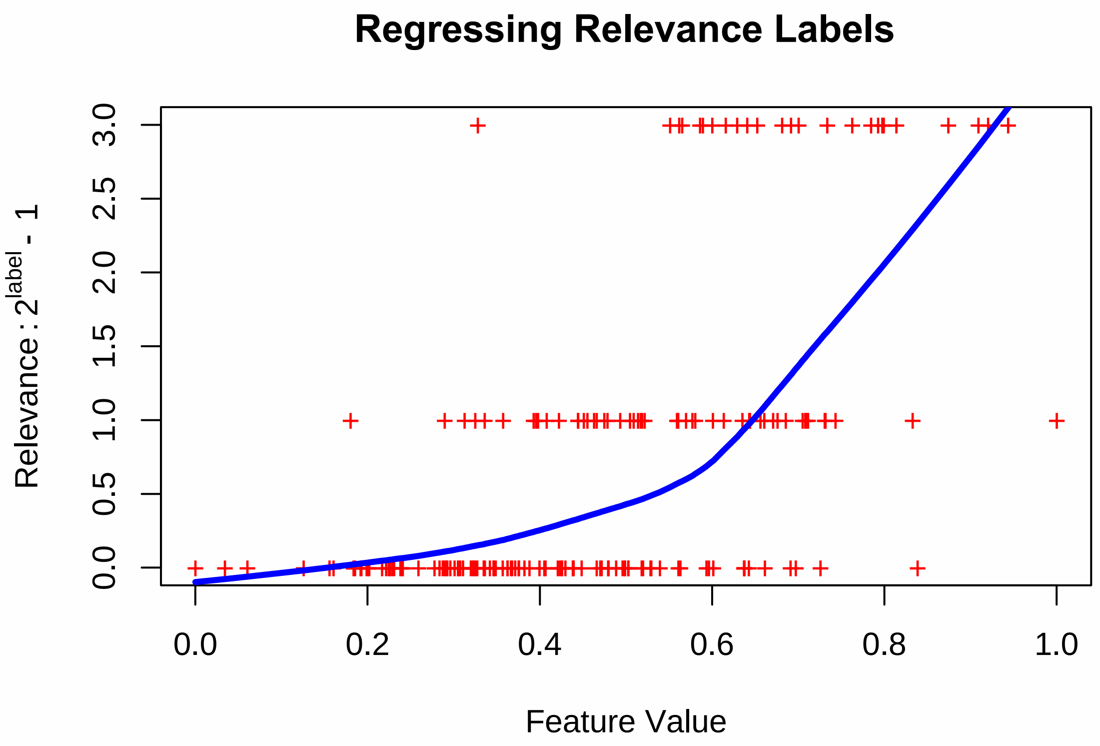
two stage (re)ranking process



rank learning – treat as regression problem

Can treat rank learning as a simple regression problem:

- predict the relevance label based on feature values
- standard regression techniques can be applied



Caveat:

- optimising 'pointwise' objective is **suboptimal**, since it doesn't preference ordering **at top of list**
- ends up predicting well scores for less relevant documents

loss functions in learning to rank

During learning, loss function can be defined in 3 ways:

- Pointwise: (e.g. MSE)

x = feature vector

$f()$ = predicted score

y = relevance label

$$total\_loss = \sum_{i=1}^{m} \sum_{j=1}^{n_{q_i}} loss(f(x_i^j), y_i^j)$$

- Pairwise: (e.g. # incorrectly ordered pairs)

$$total\_loss = \sum_{i=1}^{m} \sum_{j=1}^{n_{q_i}} \sum_{k=j+1}^{n_{q_i}} loss(\overbrace{f(x_i^j), f(x_i^k)}^{\text{Predicted scores}}, \overbrace{y_i^j, y_i^k}^{\text{Relevance labels}})$$

- Listwise: (e.g. NDCG)

$$total\_loss = \sum_{i=1}^{m} loss(\overbrace{f(x_i^1), f(x_i^2), ..., f(x_i^{n_{q_i}})}^{\text{Predicted scores}}, \overbrace{y_i^1, y_i^2, ..., y_i^{n_{q_i}}}^{\text{Relevance labels}})$$

LambdaMART

~~kmart~~

~~Walmart~~

λMART

- Listwise rank learner that makes use of the boosted regression trees
- Name comes from:

  Lambda (an approximation of loss gradient)

  \+ MART (Multiple Additive Regression Trees)
- Performs very well in practice
  - has become the default/baseline learner in most applications
- We'll now describe the algorithms in a bit more detail

Conclusions

Conclusions

In this lecture we have:

- introduced classic term weighting schemes and query-document similarity measures used in information retrieval
- discussed basics of web crawling
- introduced the learning-to-rank (reranking) methodology for combining many relevance signals into a single retrieval score